Оглавление

Содержание

Таблица обозначений

Символ	Описание
γ	γ-код
γ	Функция классификации или кластеризации: $\gamma(d)$ — класс или кластер элемента d
Γ	Метод обучения с учителем в главах 13 и 14: $\Gamma(\mathbb{D})$ — функция классификации γ, настроенной по обучающему множеству $\mathbb{D}$
λ	Собственное значение
$\vec{\mu}(\cdot)$	Центроид класса (в классификации Роккио) или кластера (в методе K-средних и центроидной кластеризации)
Φ	Обучающий пример
σ	Сингулярное значение
$\Theta(\cdot)$	Оценка сложности алгоритма
ω, ω_k	Кластер
Ω	Разбиение или множество кластеров $\{\omega_1, ..., \omega_K\}$
$\arg\max_x f(x)$	Значение x, в котором функция f достигает максимума
$\arg\min_x f(x)$	Значение x, в котором функция f достигает минимума
c, c_j	Класс или категория в классификации
cf_t	Частота термина t в коллекции (общее количество словоупотреблений термина в коллекции документов)
$\mathbb{C}$	Множество $\{c_1, ..., c_J\}$
C	Случайная величина, принимающая значения в множестве $\mathbb{C}$
C	Матрица “термин–документ”
d	Индекс d-го документа в коллекции D
d	Документ
$\vec{d}, \vec{q}$	Вектор документа, вектор запроса
D	Множество $\{d_1, ..., d_N\}$ всех документов
D_c	Множество документов, принадлежащих классу c
$\mathbb{D}$	Множество $\{<d_1, c_1>, ..., <d_N, c_N>\}$ всех помеченных документов в главах 13–15
df_t	Документная частота термина t (общее количество документов из коллекции, содержащих термин t)
H	Энтропия
H_M	M-е гармоническое число

Продолжение табл.

Символ	**Описание**
$I(X; Y)$	Взаимная информация случайных величин X и Y
idf_t	Обратная документная частота термина t
J	Количество классов
k	Первые k элементов множества, т.е. k ближайших соседей в методе kNN, первые k возвращенных документов, первые k признаков, выбранных из словаря V
k	Последовательность k символов
K	Количество кластеров
L_d	Длина документа d (в лексемах)
L_a	Длина тестового документа (или документа приложения) в лексемах
L_{ave}	Средняя длина документа (в лексемах)
M	Размер лексикона ($\|V\|$)
M_{a}	Размер лексикона тестового документа (или документа приложения)
M_{ave}	Средний размер лексикона одного документа в коллекции
M_{d}	Языковая модель документа d
N	Количество документов в коллекции
N_{c}	Количество документов в классе c
$N(\omega)$	Количество повторений события ω
$O(\cdot)$	Граница сложности алгоритма
$O(\cdot)$	Шансы события
P	Точность
$P(\cdot)$	Вероятность
P	Матрица вероятностей переходов
q	Запрос
R	Полнота
s_i	Строка
s_i	Булевы значения в зонном ранжировании
$\mathrm{sim}(d_1,d_2)$	Сходство документов d_1 и d_2
T	Общее количество лексем в коллекции документов
T_{ct}	Количество повторений слова t в документах из класса c
t	Индекс t-го термина в лексиконе V
t	Термин в лексиконе
$\mathrm{tf}_{t,d}$	Частота термина t в документе d (общее количество повторений термина t в документе d)
U_t	Случайная величина, принимающая значение 0 (термин t есть) и 1 (термина t нет)
V	Лексикон терминов $\{t_1, ..., t_M\}$ в коллекции
$\vec{v}(d)$	Вектор документа, нормализованный по длине

Окончание табл.

Символ	**Описание**
$\vec{V}(d)$	Вектор документа, не нормализованный по длине
$\mathrm{wf}_{t,d}$	Вес термина t в документе d
w	Вес, например, зоны или термина
$\vec{w}^T\vec{x}=b$	Гиперплоскость; $\vec{w}$ — вектор нормали к гиперплоскости, а w_i — i-й компонент вектора w
$\vec{x}$	Вектор инцидентности $\vec{x}=(x_1,...,x_M)$; представление признаков документа
X	Случайная величина, принимающая значения из лексикона V (например, в k-й координате в документе)
$\mathbb{X}$	Пространство документов в классификации текстов
$\lvert A\rvert$	Кардинальное число множества A: количество элементов множества A
$\lvert S\rvert$	Определитель квадратной матрицы S
$\lvert s_i\rvert$	Длина строки s_i в символах
$\lvert\vec{x}\rvert$	Длина вектора $\vec{x}$
$\lvert\vec{x}-\vec{y}\rvert$	Евклидово расстояние между векторами $\vec{x}$ и $\vec{y}$, т.е. длина вектора $\vec{x}-\vec{y}$

Введение в информационный поиск

Введение в информационный поиск — это первый учебник, в котором наряду с классическим поиском рассматриваются веб-поиск, а также классификация и кластеризация текстов. Учебник написан с точки зрения информатики и содержит современное изложение всех аспектов проектирования и реализации систем сбора, индексирования и поиска документов, методов оценки таких систем, а также введение в методы машинного обучения на базе коллекций текстов.

Несмотря на то что учебник задуман как вводный курс по информационному поиску, он будет интересен исследователям и профессионалам. Полный набор слайдов для лекций и упражнений, сопровождающих книгу, доступен в Интернете.

Кристофер Д. Маннинг (Christopher D. Manning) — ассоциированный профессор компьютерных наук в Стэнфордском университете (Stanford University).

Прабхакар Рагхаван (Prabhakar Raghavan) — директор исследовательского департамента корпорации Yahoo! Pesearch и профессор-консультант по компьютерным наукам Стэнфордского университета.

Хинрич Шютце (Hinrich Schьtze) — заведующий кафедрой теоретической вычислительной лингвистики Института обработки текстов на естественных языках (Штутгардский университет).

Предисловие

Еще в 1990-х годах результаты социологических исследований свидетельствовали о том, что большинство людей предпочитают получать информацию от других людей, а не с помощью информационно-поисковых (Information Retrieval — IR) систем. Например, в то время для бронирования билетов и гостиниц люди чаще обращались к сотрудникам туристических агентств. Однако за последние десять лет благодаря постоянному совершенствованию методов информационного поиска поисковые системы в вебе поднялись на новый качественный уровень, позволяющий лучше удовлетворять потребности все большего количества людей, а веб-поиск стал стандартным и часто предпочтительным механизмом поиска информации. Например, в 2004 году опрос Pew Internet Survey (Fallows 2004) показал, что "92% пользователей сети Интернет считают ее удобной для получения повседневной информации". К удивлению многих, информационный поиск из преимущественно академической дисциплины стал базисом для средств доступа к информации, на который полагается большинство людей. В книге изложены научные основы этой дисциплины на уровне, доступном как студентам старших курсов университетов, так и способным студентам младших курсов.

Информационный поиск возник раньше веба. Его эволюция стимулировалась разнообразными проблемами, связанными с обеспечением поиска и доступа к информационным источникам. Сначала информационный поиск касался научных публикаций и библиотечных каталогов, однако вскоре он распространился и на другие сферы, в которых важна роль информации, — на журналистику, право и медицину. Многие исследования в области информационного поиска проводились именно в этом контексте, и до сих пор большая доля практических приложений этой дисциплины связана с обеспечением доступа к неструктурированной информации, хранящейся в многочисленных корпоративных и правительственных базах данных. Именно этим методам посвящена б*о*льшая часть книги.

Тем не менее в последние годы основным двигателем прогресса является веб, открывший возможность публиковать информацию десяткам миллионов пользователей. Эта лавина публикаций осталась бы недоступной, если бы информацию было невозможно найти, сопроводить аннотацией и проанализировать так, чтобы каждый пользователь мог быстро найти необходимые ему релевантные и исчерпывающие сведения. В конце 1990-х годов многие люди поняли, что дальнейшая индексация всего веба вскоре станет невозможной из-за его экспоненциального роста. Однако значительные научные инновации и превосходные инженерные решения, быстро снижающаяся стоимость компьютерного аппаратного обеспечения и появление коммерческой заинтересованности в веб-поиске в совокупности способствовали возникновению крупных поисковых систем, способных с высоким качеством и за доли секунды выполнить сотни миллионов запросов в день по базе, состоящей из миллиардов веб-страниц.

Структура книги и учебного курса

Книга является результатом объединения нескольких учебных курсов, прочитанных в Стэнфордском университете (Stanford University) и Штутгартском университете (University of Stuttgart) в разных вариантах: на протяжении одной четверти, одного семестра и двух четвертей. Эти курсы предназначались для старшекурсников, изучавших компьютерные науки, но оказались полезными и для студентов младших курсов, а также для студентов, осваивавших юриспруденцию, медицинскую информатику, статистику, лингвистику и разнообразные технические дисциплины. Книга организована так, чтобы осветить то, что мы считаем важным для студентов, изучающих информационный поиск на протяжении одного семестра. Кроме того, каждая глава содержит материал одной лекции продолжительностью 75–90 минут.

Главы 1–8 посвящены основам информационного поиска и, в частности, сущности поисковых систем; мы считаем, что этот материал является ядром любого курса по информационному поиску. В главе 1 введены инвертированные индексы (inverted indexes) и показано, как с их помощью можно обработать простые булевы запросы (Boolean queries). В главе 2 детально описываются способы предварительной обработки документов перед индексированием и методы усовершенствования индексов для расширения функциональных возможностей и повышения скорости поиска. В главе 3 рассматриваются поисковые структуры для словарей и методы обработки запросов, содержащих орфографические ошибки и другие неточности. В главе 4 описывается несколько алгоритмов построения инвертированного индекса по коллекции текстов с особым акцентом на масштабируемые и распределенные алгоритмы, допускающие применение к очень большим коллекциям. В главе 5 излагаются методы сжатия словарей и инвертированных индексов. Эти методы очень важны для обеспечения быстрой (за доли секунды) обработки пользовательских запросов в больших поисковых системах. Индексы и запросы, изучаемые в главах 1–5, касаются лишь *булева поиска* (Boolean retrieval), при котором документ либо соответствует запросу, либо нет. Желание измерить *степень* соответствия документа запросу, или релевантность (score) документа, стимулировало разработку методов взвешивания терминов (term weighting) и ранжирования (computation of scores), описанных в главах 6 и 7, и далее, к концепции списка документов, упорядоченных по степени соответствия запросу. Глава 8 посвящена оценке информационно-поисковых систем на основании экспертных оценок релевантности найденных документов, что позволяет сравнивать относительное качество систем на стандартных коллекциях документов и запросов.

Главы 9–21 основаны на материале, изложенном в главах 1–8, и охватывают широкий спектр более сложных тем. В главе 9 обсуждаются методы повышения эффективности поиска с помощью таких приемов, как обратная связь по релевантности (relevance feedback) и расширение запросов (query expansion), предназначенных для увеличения вероятности нахождения релевантных документов. В главе 10 рассматриваются методы информационного поиска по документам, структурированным с помощью языков разметки, таких как XML и HTML. Мы сводим поиск по структурированным документам к применению методов ранжирования на основе векторной модели (vector space scoring), изложенных в главе 6. В главах 11 и 12 для ранжирования документа по отношению к запросу используется теория вероятностей. Глава 11 посвящена традиционному вероятностному информационному поиску, позволяющему вычислить вероятность релевантности документа при заданном наборе слов запроса. Впоследствии эту вероятность можно использовать как показатель релевантности при ранжировании. В главе 12 иллюстриру-

ется альтернатива, в рамках которой для каждого документа в коллекции создается языковая модель, позволяющая оценить вероятность того, что она порождает заданный запрос. Эта вероятность является еще одним количественным показателем, с помощью которого осуществляется ранжирование документов.

В главах 13–18 излагаются методы машинного обучения и численные методы информационного поиска. Главы 13–15 посвящены проблеме классификации документов по известным категориям на основе набора документов и классов, которым они принадлежат. В главе 13 представлены доказательства того, что классификация на основе статистики представляет собой одну из ключевых технологий, необходимых для успешного функционирования поисковой системы. В ней излагается наивный байесовский подход (Naive Bayes), представляющий собой концептуально простой и эффективный метод классификации текстов, а также основы стандартной методологии оценки текстовых классификаторов. В главе 14 описано применение модели векторного пространства, введенной в главе 6, а также изложены два метода классификации: метод Роккио (Rocchio method) и метод k ближайших соседей (k nearest neighbor — kNN), применяемые к векторам документов. В ней также рассматривается компромисс между смещением и разбросом (дисперсией), представляющий собой важную характеристику задач обучения и позволяющий установить критерии для выбора подходящего метода классификации текстов. В главе 15 вводится метод опорных векторов (support vector machine), который многие исследователи в настоящее время считают наиболее эффективным методом классификации текстов. Кроме того, в данной главе исследуются связи между задачей классификации и, на первый взгляд, совершенно посторонними темами, таким как вывод функций ранжирования по набору обучающих примеров.

Главы 16–18 посвящены идентификации кластеров близких документов в коллекции. В главе 16 сначала приводится обзор нескольких важных приложений кластеризации в области информационного поиска, а затем рассматриваются два алгоритма плоской кластеризации (flat clastering): эффективный и широко используемый для кластеризации документов алгоритм *K* средних (*K*-means algorithm) и EM-алгоритм (expectation-maximization algorithm), который с вычислительной точки зрения является более затратным, но более гибким. В главе 17 обосновывается необходимость иерархически структурированной кластеризации (вместо плоской) для многих приложений в области информационного поиска, а также рассматриваются алгоритмы кластеризации, порождающие иерархии кластеров. В этой главе также рассматривается сложная проблема автоматической разметки кластеров. Глава 18 посвящена методам линейной алгебры, представляющим собой расширение методов кластеризации и открывающим захватывающие перспективы для применения алгебраических методов, разрабатываемых в рамках латентного семантического индексирования (latent semantic indexing).

Главы 19–21 посвящены проблемам поиска в вебе. В главе 19 приводятся краткий обзор основных задач, связанных с поиском в вебе, а также набор широко распространенных методов информационного поиска в вебе. В главе 20 описываются архитектура и требования, предъявляемые к веб-роботам (web-crawlers). В главе 21 рассматривается мощь анализа ссылок (link analysis) в ходе веб-поиска с помощью нескольких методов линейной алгебры и теории вероятностей.

Эта книга является исчерпывающим источником знаний по всем темам, связанным с информационным поиском. За ее пределами осталось множество тем, выходящих за рамки вводного курса по информационному поиску. Тем не менее все, кого интересуют эти темы, могут обратиться к следующим учебникам.

Cross-language IR, Grossman and Frieder, 2004, ch. 4, and Oard and Dorr, 1996.

Image and multimedia IR, Grossman and Frieder, 2004, ch. 4; Baeza-Yates and Ribeiro-Neto, 1999, ch. 6; Baeza-Yates and Ribeiro-Neto, 1999, ch. 11; Baeza-Yates and Ribeiro-Neto, 1999, ch. 12; del Bimbo, 1999; Lew, 2001; and Smeulders et al., 2000.

Speech retrieval, Coden et al., 2002.

Music retrieval, Downie, 2006 and `http://www.ismir.net/`.

User interfaces for IR, Baeza-Yates and Ribeiro-Neto, 1999, ch. 10.

Search User Interfaces, Marti A. Hearst, 2009.

Parallel and peer-to-peer IR, Grossman and Frieder, 2004, ch. 7; Baeza-Yates and Ribeiro-Neto, 1999, ch. 9; and Aberer, 2001.

Digital libraries, Baeza-Yates and Ribeiro-Neto, 1999, ch. 15, and Lesk, 2004.

Information science perspective, Korfhage, 1997; Meadow et al., 1999; and Ingwersen and Järvelin, 2005.

Logic-based approaches to IR, van Rijsbergen, 1989.

Natural language processing techniques, Manning and Schütze, 1999; Jurafsky and Martin, 2008; and Lewis and Jones, 1996.

Предварительные требования к уровню подготовки читателей

Чтобы понять содержание книги, достаточно знать основы структур данных и алгоритмов, линейной алгебры и теории вероятностей. Для удобства читателей и преподавателей, желающих сосредоточиться лишь на отдельных главах, мы приводим следующую информацию.

Для освоения глав 1–5 необходимо знать основы структур данных и алгоритмов. Главы 6 и 7 требуют в дополнение к этому знания основ линейной алгебры, включая векторы и скалярное произведение. Далее вплоть до главы 11 никаких дополнительных знаний не требуется. Для освоения материала, изложенного в главе 11, следует разбираться в основах теории вероятностей; краткий обзор понятий, используемых в главах 11–13, приведен в разделе 11.1. Глава 15 требует от читателя знания понятий нелинейной оптимизации, хотя ее можно читать, не имея глубоких познаний об алгоритмах нелинейной оптимизации. Глава 18 предполагает знание основ линейной алгебры, включая ранг матрицы и собственные векторы; краткий обзор этих понятий приведен в разделе 18.1. Знание определений собственных значений и собственных векторов требуется также при чтении главы 21.

Разметка

Примеры в тексте сопровождаются пиктограммой, на которой изображен карандаш.

Сложный материал выделяется пиктограммой с изображением ножниц.

?

Упражнения отмечены знаком вопроса. Сложность задачи указывается с помощью звездочек: простая — одна [*], средняя — две [**] и сложная — три [***] звездочки.

Благодарности

Мы благодарим издательство Cambridge University Press за то, что оно позволило опубликовать рабочий вариант книги в Сети, благодаря чему мы получили обратную связь с потенциальными читателями уже в процессе написания книги. Мы также выражаем признательность Лорен Коулз (Lauren Cowles), превосходному редактору, многократно изучившему рукопись каждой главы и сделавшему множество полезных замечаний относительно стиля, организации и охвата материала, а также ряд глубоких замечаний по существу книги. В том, что мы достигли поставленной цели, — большая ее заслуга.

Мы очень благодарны многим людям, сделавшим комментарии, предложения и исправления, основываясь на черновом варианте книги. Вот их имена: Шерил Аашейм (Cheryl Aasheim), Джош Аттенберг (Josh Attenberg), Бъерн Андрист (Björn Andrist), Люк Беланжер (Luc Bélanger), Том Бройель (Tom Breuel), Даниэль Буркхардт (Daniel Burckhardt), Георг Бушер (Georg Buscher), Фазли Кан (Fazli Can), Динцюань Чен (Dingquan Chen), Эрнест Дэвис (Ernest Davis), Педро Домингос (Pedro Domingos), Родриго Панчиниак Фернандес (Rodrigo Panchiniak Fernandes), Паоло Ферраджина (Paolo Ferragina), Норберт Фюр (Norbert Fuhr), Вигнеш Ганапати (Vignesh Ganapathy), Элмер Гардуно (Elmer Garduno), Сюбо Гэн (Xiubo Geng), Дэвид Гондек (David Gondek), Серджио Говони (Sergio Govoni), Миклош Эрдели (Miklós Erdélyi), Коринна Хабетц (Corinna Habets), Бен Хэнди (Ben Handy), Донна Харман (Donna Harman), Бенджамин Хаскелл (Benjamin Haskell), Томас Хюнн (Thomas Hühn), Дипак Джейн (Deepak Jain), Ральф Янкович (Ralf Jankowitsch), Динакар Джаяраджан (Dinakar Jayarajan), Винай Какаде (Vinay Kakade), Марек Ковалькевич (Marek Kowalkiewicz), Мэй Кобаяси (Mei Kobayashi), Вессель Краий (Wessel Kraaij), Рик Лефлер (Rick Lafleur), Флориан Лоус (Florian Laws), Хан Ли (Hang Li), Дэвид Манн (David Mann), Эннио Мази (Ennio Masi), Джуна Макконен (Juna Makkonen), Фрэнк Маккоун (Frank McCown), Пол Макнами (Paul McNamee), Свен Мейер цу Эссен (Sven Meyer zu Eissen), Александер Мурзаку (Alexander Murzaku), Гонзало Наварро (Gonzalo Navarro), Скотт Олссон (Scott Olsson), Даниэль Паива (Daniel Paiva), Тао Цинь (Tao Qin), Мега Рагхаван (Megha Raghavan), Картик Рагунатан (Karthik Raghunathan), Гулям Раза (Ghulam Raza), Михал Розен-Цви (Michal Rosen-Zvi), Клаус Ротенхауслер (Klaus Rothenhäusler), Кенью Л. Раннер (Kenyu L. Runner), Александер Саламанка (Alexander Salamanca), Григорий Сапунов (Grigory Sapunov), Тобиас Шеффер (Tobias Scheffer), Нико Шлафер (Nico Schlaefer), Евгений Шадчнев (Evgeny Shadchnev), Ян Соборофф (Ian Soboroff), Бенно Штейн (Benno Stein), Марцин Сидоу (Marcin Sydow), Эндрю Тёрнер (Andrew Turner), Джейсон Утт (Jason Utt), Хьюи Во (Huey Vo), Трэвис Уэйд (Travis Wade), Майк Уолш (Mike Walsh), Чанлян Ван (Changliang Wang), Жэньцзин Ван (Renjing Wang), Делл Джанг (Dell Zang) и Томас Цойме (Thomas Zeume).

Многие люди присылали свои замечания к отдельным главам как по нашей просьбе, так и по своей инициативе. Мы очень благодарны за это Джеймсу Аллану (James Allan), Омару Алонзо (Omar Alonso), Исмаилу Сеньору Альтинговде (Ismail Sengor Altingovde), Во Нгок Ану (Vo Ngoc Anh), Ру Бланко (Roi Blanco), Эрику Бреку (Eric Breck), Эрику Брауну (Eric Brown), Марку Карману (Mark Carman), Карлосу Кастильо (Carlos Castillo), Юнгху Чо (Junghoo Cho), Арону Кулотте (Aron Culotta), Дугу Каттингу (Doug Cutting), Мегане Деодхар (Meghana Deodhar), Сьюзан Дюма (Susan Dumais), Йоханнесу Фюрнк-

ранцу (Johannes Fürnkranz), Андреасу Хессу (Andreas Heß), Дьорду Хиемстра (Djoerd Hiemstra), Дэвиду Халлу (David Hull), Торстену Йоахимсу (Thorsten Joachims), Сиддхартхе Джонатану Дж. Б. (Siddharth Jonathan J. B.), Йаапу Кампсу (Jaap Kamps), Мунье Лалмас (Mounia Lalmas), Эми Лэнгвиль (Amy Langville), Николасу Лестеру (Nicholas Lester), Дэйву Льюису (Dave Lewis), Стивену Лиу (Stephen Liu), Даниэлю Лоуду (Daniel Lowd), Йоси Массу (Yosi Mass), Джеффу Мишелзу (Jeff Michels), Алессандро Москитти (Alessandro Moschitti), Амиру Найми (Amir Najmi), Марку Найорку (Marc Najork), Джорджио Марие Ди Нунцио (Giorgio Maria Di Nunzio), Полю Огилви (Paul Ogilvie), Приянке Патель (Priyank Patel), Яну Педерсену (Jan Pedersen), Кэтрин Педингс (Kathryn Pedings), Виссилису Плахурасу (Vassilis Plachouras), Даниэлю Рамаге (Daniel Ramage), Стивену Ризлеру (Stefan Riezler), Майклу Шиелену (Michael Schiehlen), Хельмуту Шмиду (Helmut Schmid), Фальку Николасу Шолеру (Falk Nicolas Scholer), Сабине Шульте им Вальде (Sabine Schulte im Walde), Фабрицио Себастиани (Fabrizio Sebastiani), Сарабжит Сингху (Sarabjeet Singh), Валентину Спитковскому (Valentin I. Spitkovsky), Александру Штрелю (Alexander Strehl), Джону Тейту (John Tait), Шивакумару Вайтианатану (Shivakumar Vaithyanathan), Эллен Ворхиз (Ellen Voorhees), Герхарду Вайкуму (Gerhard Weikum), Дэвиду Вайсу (Dawid Weiss), Джимингу Янгу (Yiming Yang), Йисонгу Ю (Yisong Yue), Джиану Чангу (Jian Zhang) и Джастин Цобель (Justin Zobel).

Кроме того, мы хотели бы упомянуть рецензентов, количество и качество комментариев которых трудно переоценить. Мы благодарны Павлу Берхину (Pavel Berkhin), Стефану Бютчеру (Stefan Büttcher), Джейми Каллан (Jamie Callan), Байрону Дому (Byron Dom), Торстену Зулю (Torsten Suel) и Эндрю Тротману (Andrew Trotman) за их значительное влияние на содержание и структуру книги.

Исходные черновики глав 13–15 были основаны на слайдах, созданных Рэем Муни (Ray Mooney). Несмотря на то что этот материал подвергся существенной переработке, мы благодарны Рэю за его вклад в эти три главы в целом и за описание вопросов, связанных с временно*й* сложностью алгоритмов классификации текстов, в частности.

К сожалению, мы не в состоянии перечислить всех, поскольку до сих пор получаем отзывы от наших читателей. Как и все самоуверенные авторы, мы не всегда прислушивались к их советам. Опубликованный вариант книги полностью остается на совести авторов.

Авторы выражают благодарность Стэнфордскому университету (Stanford University) и Штутгардскому университету (University of Stuttgart) за создание академической среды для плодотворных научных дискуссий и возможность читать учебные курсы, ставшие основанием для этой книги. Кристофер Маннинг (Christopher Manning) благодарит свою семью за многие часы, которые она позволила ему провести, работая на книгой, и надеется, что ему удастся наверстать упущенное на уик-эндах в следующем году. Прабхакар Рагаван (Prabhakar Raghavan) благодарит свою семью за терпение и поддержку во время работы над книгой, а также выражает признательность компании Yahoo!, Inc. за плодотворную атмосферу, в которой проходила эта работа. Хайнрих Шютце (Hinrich Schütze) хотел бы поблагодарить своих родителей, семью и друзей за поддержку в процессе работы над книгой.

Веб-адреса и контактная информация

Этой книге посвящен веб-сайт `http://informationretrieval.org`, который, помимо ссылок на различные ресурсы, содержит слайды для каждой главы, которые можно использовать для лекций. Мы приветствуем обратную связь с читателями и с благодарностью примем их замечания и предложения по улучшению книги, которые можно отсылать авторам по адресу `informationretrieval@yahoogroups.com`.

Глава 1

Булев поиск

Смысл термина информационный поиск (information retrieval — IR) может быть очень широким. Даже извлечение кредитной карточки из бумажника для того, чтобы узнать ее номер, уже можно трактовать как вид информационного поиска. Тем не менее как научная дисциплина информационный поиск может быть определен следующим образом.

> Информационный поиск (IR) — это процесс поиска в большой коллекции (хранящейся, как правило, в памяти компьютеров) некоего неструктурированного материала (обычно — документа), удовлетворяющего информационные потребности.

Если рассматривать информационный поиск таким образом, то можно отметить, что раньше подобной деятельностью занимались лишь отдельные специалисты: библиографы-консультанты, помощники юристов и другие профессионалы. Мир изменился, и теперь сотни миллионов людей делают это ежедневно, используя поисковые системы или просматривая свою электронную почту.[1] Информационный поиск быстро становится основной формой доступа к информации, вытесняя традиционный поиск по ключу (когда продавец говорит вам: "К сожалению, я смогу найти ваш заказ, только если вы назовете мне его номер").

К информационному поиску можно также отнести некоторые задачи, не подпадающие под данное выше базовое определение. Когда мы говорим "неструктурированные данные", мы подразумеваем данные, которые не имеют ясной, семантически очевидной и легко реализуемой на компьютере структуры. Они представляют собой противоположность структурированным данным, каноническим примером которых являются реляционные базы данных наподобие тех, которые обычно используются предприятиями для хранения реестров продукции и персональных данных сотрудников. В реальности же совершенно "неструктурированных данных" практически не существует. Например, обычные текстовые данные имеют скрытую структуру, характерную для естественных языков. Однако даже если требовать явного наличия структуры, то большинство текстов таковую очевидно имеют, поскольку в них есть заголовки, абзацы и сноски, которые обычно представлены в тексте в виде явной разметки (например, в коде веб-страниц). Поэтому методы информационного поиска используются также для "полуструктурированного" поиска, например для нахождения документа, в заголовке которого содержится слово Java, а в теле — слово threading.

К информационному поиску относятся и такие задачи, как навигация пользователей по коллекции документов и фильтрация документов, а также дальнейшая обработка найденных документов. Если имеется набор документов, то возникает задача кластеризации, которая заключается в определении наилучшей группировки документов по их содержа-

---

[1] В современном английском языке слово "search" ("поиск") все чаще вытесняет словосочетание "(information) retrieval" (буквально — "получение информации"). И хотя термин "search" сам по себе довольно многозначен, в данной книге эти слова используются как синонимы.

нию. Это напоминает расстановку книг на полке по темам. Если задан набор тем, постоянные информационные потребности и другие категории (например, пригодность текстов для различных возрастных групп), то возникает задача классификации – определить, к какому классу относится (и относится ли вообще) каждая группа документов. Для ее решения часто сначала вручную классифицируют какое-то количество документов, надеясь классифицировать новые документы автоматически.

Системы информационного поиска можно классифицировать по масштабу их действия. Для этого полезно выделить три уровня. В процессе веб-поиска (Web search) система должна выполнить поиск среди миллиардов документов, размещенных на миллионах компьютеров. Отличительными особенностями веб-поиска является необходимость сбора документов для индексации, способность создавать системы, эффективно работающие с огромными массивами информации, а также учет определенных аспектов веба, таких как использование гипертекста и противодействие попыткам веб-мастеров искусственно повысить позиции своих сайтов, манипулируя содержимым веб-страниц. Эти вопросы будут рассмотрены в главах 19–21. На другом полюсе находится персональный информационный поиск (personal information retrieval). В последние годы многие операционные системы имеют встроенные возможности поиска (например, Mac OS X компании Apple, реализующая технологию Spotlight, и операционная система Windows Vista, предоставляющая функцию Instant Search). Кроме того, программы электронной почты обычно предоставляют не только функции поиска, но и средства классификации текстов, по крайней мере они должны иметь фильтры от спама, а также средства ручной или автоматической сортировки писем по папкам. Для этих систем критично обрабатывать все многообразие форматов документов на обычном персональном компьютере. Важны простое сопровождение и инсталляция, а также небольшой объем занимаемой памяти, позволяющий выполнять работу, не раздражая пользователя. Промежуточное положение между упомянутыми двумя классами занимают системы корпоративного (enterprise), ведомственного (institutional) и ориентированного на предметную область (domain-specific) поиска, которые работают, например, с коллекциями внутренних документов корпораций, базами патентов или научными статьями по биохимии. В этих случаях документы, как правило, хранятся в централизованных файловых системах, и одна или несколько специализированных машин осуществляют поиск по коллекции. В книге рассматриваются методы, применяемые во всем спектре приложений, однако методы параллельного и распределенного поиска в системах веб-поиска излагаются нами довольно поверхностно, поскольку детали их работы описаны лишь в небольшом количестве публикаций. Однако, как правило, разработчики программного обеспечения интересуются вопросами персонального и корпоративного поиска, если, конечно, они не работают в одной из немногочисленных компаний, специализирующихся на веб-поиске.

Глава начинается с очень простого примера информационного поиска и введения идеи матрицы “термин–документ” (раздел 1.1). Затем рассматривается базовая структура данных — инвертированный индекс (раздел 1.2). После этого изучаются модель булева поиска (раздел 1.3), а также методы обработки булевых запросов (разделы 1.3 и 1.4).

1.1. Пример информационного поиска

В библиотеках многих людей есть толстый том собрания произведений Шекспира. Допустим, вы хотите определить, в какой пьесе используются слова Brutus AND Caesar AND NOT Calpurnia. Для этого можно, например, прочитать текст от начала до конца,

отмечая пьесы, содержащие слова Brutus и Caesar, и исключая из рассмотрения пьесы, в которых встречается слово Calpurnia. Простейший компьютерный метод решения этой задачи сводится к последовательному просмотру (linear scanning) всех документов. Этот процесс часто называют прямым поиском или, на английском, grepping (от названия команды grep, которая в операционной системе Unix выполняет этот процесс). Прямой поиск по тексту может быть очень эффективным, особенно на современных компьютерах. Довольно часто такая обработка допускает поиск по шаблону с джокерами (wildcard pattern matching). На современных компьютерах выполнения простых запросов на коллекциях данных среднего размера (общий объем тома собрания избранных произведений Шекспира составляет чуть меньше одного миллиона слов) вполне достаточно для рядового пользователя.

Тем не менее во многих случаях необходимо нечто большее.

1. Иногда нужно быстро обработать большую коллекцию документов. Объем данных, доступных онлайн, растет по меньшей мере с той же скоростью, с которой увеличивается скорость работы компьютеров. Теперь мы хотели бы выполнять поиск информации в массивах данных, содержащих миллиарды и триллионы слов.
2. Иногда требуется бо́льшая гибкость при выполнении операции сравнения. Например, было бы непрактично выполнять запрос Romans NEAR countrymen с помощью команды `grep`, где слово NEAR может означать “не далее 5 слов” или “в пределах одного предложения”.
3. Иногда необходимо выполнить ранжированный поиск. Во многих ситуациях нам нужен не просто ответ, а наилучший ответ на запрос относительно документов, содержащих определенное слово.

Для того чтобы избежать последовательного просмотра текстов при выполнении каждого запроса, заранее составляется индекс документов. Вернемся к тому Шекспира и опишем на его примере основы модели булева поиска. Допустим, что для каждого документа (в данном примере — для каждой пьесы Шекспира) мы создали запись, в которой указано, содержит ли он конкретное слово из лексикона Шекспира (состоящего из 32 тысяч слов). В результате возникнет бинарная матрица инцидентности “термин–документ” (term-document incidence matrix), представленная на рис. 1.1. Термины (terms) — это единицы индексирования (это понятие обсуждается в разделе 2.2). Как правило, это обычные слова, но в литературе по информационному поиску употребляется название термин, поскольку некоторые выражения могут быть непохожими на “обычные слова”, например I–9 или Hong Kong. Теперь в зависимости от порядка просмотра матрицы (по строкам или по столбцам) мы можем получить либо вектор термина, в котором указано, в каких документах он встречается, либо вектор документа, в котором указано, какие термины в нем употребляются. [2]

[2] Формально говоря, для того чтобы векторы терминов стали столбцами, достаточно транспонировать матрицу.

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
Antony	1	1	0	0	0	1	
Brutus	1	1	0	1	0	0	
Caesar	1	1	0	1	1	1	
Calpurnia	0	1	0	0	0	0	
Cleopatra	1	0	0	0	0	0	
mercy	1	0	1	1	1	1	
worser	1	0	1	1	1	0	
...							

Рис. 1.1. Матрица инцидентности "термин–документ". Элемент матрицы (t,d) равен 1, если пьеса в столбце d содержит слово из строки t, и 0, если не содержит

Для обработки запроса Brutus AND Caesar AND NOT Calpurnia мы берем вектор для терминов Brutus, Caesar и Calpurnia, вычисляем двоичное дополнение к последнему вектору и выполняем поразрядную операцию AND.

110100 AND 110111 AND 101111 = 100100

Ответ на запрос выглядит так: Antony and Cleopatra и Hamlet (рис. 1.2).

Antony and Cleopatra, Act III, Scene ii

Agrippa [Aside to Domitius Enobarbus] Why, Enobarbus,
When Antony found Julius Caesar dead
He cried almost to roaring; and he wept
When at Philippi he found Brutus slain.

Hamplet, Act IIi, Scene ii

Lord Polonius: I did enact Julius Caesar: I was killed i' the
Capitol; Brutus killed me.

Рис. 1.2. Результаты обработки запроса Brutus AND Caesar AND NOT Calpurnia

Модель булева поиска — это модель информационного поиска, в ходе которого можно обрабатывать любой запрос, имеющий вид булева выражения, т.е. выражения, в котором термины используются в сочетании с операциями AND, OR и NOT. В рамках этой модели документ рассматривается просто как множество слов.

Рассмотрим теперь более реалистичный сценарий, на примере которого удобно ввести несколько понятий и обозначений. Допустим, что в нашем распоряжении есть N = 1 миллион документов. Под документами (documents) подразумеваются любые объекты, на основе которых решено строить систему информационного поиска. Документами могут быть личные заметки или главы книги (см. раздел 2.1.2). Группу документов, по которой осуществляется поиск, мы будем называть коллекцией (collection). Ее также иногда называют корпусом (corpus) или массивом текстов (body of texts). Допустим, что каждый документ состоит примерно из 1 000 слов (2–3 книжные страницы). Если предположить, что в среднем на хранение слова с учетом пробелов и знаков препинания отводится 6 байт, то такая коллекция документов займет около 6 Гбайт (GB). Как правило,

в подобных документах может оказаться около M = 500 000 разных терминов. В выбранных числах нет ничего особенного, и они могут варьироваться в ту или другую сторону, но они все же дают представление о размерности рассматриваемых задач. Этот вопрос будет изучен в разделе 5.1.

Наша цель — разработать систему, выполняющую поиск по произвольному запросу (ad hoc retrieval). Это самая распространенная задача информационного поиска. Система должна найти в коллекции документы, которые являются наиболее релевантными по отношению к произвольным информационным потребностям пользователя, взаимодействующего с системой посредством одноразовых запросов. Информационная потребность (information need) — это тема, о которой пользователь хочет знать больше. Ее следует отличать от запроса, т.е. от того, что пользователь вводит в компьютер, пытаясь выразить свою информационную потребность. Документ называется релевантным (relevant), если, с точки зрения пользователя, он содержит ценную информацию, удовлетворяющую его информационную потребность. Приведенный выше пример был довольно искусственным, поскольку информационная потребность была сведена к поиску конкретных слов, в то время как обычно пользователи интересуются более неопределенными темами, такими как “pipeline leaks” (“утечки из трубопроводов”), и хотят получать документы независимо от того, употребляются в них именно эти слова или идея выражается другими словами, например pipeline rupture (“прорыв трубопровода”). Для того чтобы оценить эффективность системы информационного поиска (качество результатов ее работы), пользователь обычно желает знать два основных статистических показателя, характеризующих результаты, возвращаемые системой.

Точность (precision). Какая доля результатов является релевантной по отношению к информационной потребности?

Полнота (recall). Какая доля релевантных документов из коллекции возвращена системой?

Подробнее релевантность и показатели качества, включая точность и полноту, рассматриваются в главе 8.

Теперь мы уже не можем создать матрицу “термин–документ” наивным способом. Матрица размером 500 К × 1 М содержит полтриллиона нулей и единиц — слишком много, чтобы поместиться в памяти компьютера. Тем не менее следует подчеркнуть, что эта матрица является чрезвычайно разреженной, т.е. содержит лишь небольшое количество ненулевых элементов. Поскольку каждый документ состоит из 1 000 слов, матрица будет содержать не более одного миллиарда единиц, поэтому как минимум 99,8% ячеек будут содержать нули. Намного целесообразнее хранить в памяти только единицы.

Эта идея является основной по отношению к первой важной концепции информационного поиска — инвертированному индексу. Это название на самом деле является избыточным: индекс всегда ставит в соответствие терминам те части документа, в которых они встречаются. Тем не менее выражения инвертированный индекс, и иногда — инвертированный файл — стали стандартными в области информационного поиска.[3] Основная идея инвертированного индекса проиллюстрирована на рис. 1.3. Сначала мы записы-

[3] Некоторые специалисты по информационному поиску предпочитают термин *инвертированный файл*, но выражения наподобие *создание индекса* и *сжатие индекса* стали более привычными, чем *создание инвертированного файла* и *сжатие инвертированного файла*. Для определенности мы на протяжении всей книги будем использовать термин (*инвертированный*) *индекс*.

ваем в память словарь (dictionary) терминов (иногда его также называют словником (vocabulary) или лексиконом (lexicon); в этой книге мы называем словарем структуру данных, а лексиконом — множество терминов той или иной коллекции). Затем для каждого термина мы создаем список, в котором указаны документы, содержащие данный термин. Каждый элемент списка, содержащий информацию о том, что термин находится в документе, а также (довольно часто) о позиции термина в этом документе, называется словопозицией (posting).[4] Соответствующий список называется списком словопозиций (postings list) или инвертированным списком. Словарь, представленный на рис. 1.3, упорядочен в алфавитном порядке, а каждый список словопозиций упорядочен по идентификаторам документов. В разделе 1.3 показано, почему это оказывается полезным; позднее (в разделе 7.1.5) мы рассмотрим альтернативы этому приему.

Рис. 1.3. Две части инвертированного индекса. Словарь обычно находится в памяти вместе с указателями на каждый список словопозиций, которые хранятся на диске

1.2. Первая попытка создать инвертированный индекс

Для того чтобы получить выигрыш в скорости поиска, необходимо построить инвертированный индекс заранее. Этот процесс состоит из следующих этапов.

1. Собираем документы, подлежащие индексации.

Friends, Romans, contrymen	So let it be with Caesar	...

2. Размечаем текст, превращая каждый документ в список лексем (tokens).

Friends	Romans	countrymen	So	...

3. Проводим предварительную лингвистическую обработку, создаем список нормализованных лексем, представляющих собой индексируемые термины.

friends	romans	countrymen	So	...

[4] Если речь идет о (непозиционном) инвертированном индексе, то единицей записи является идентификатор документа, но он неразрывно связан с термом через список единиц записи; кроме того, иногда под единицей записи мы будем подразумевать пару (“термин, идентификатор документа”).

4. Индексируем документы, в которых встречается термин, создавая инвертированный индекс, состоящий из словаря и позиций.

Ранние этапы обработки, т.е. этапы 1–3, будут рассмотрены в разделе 2.2. Пока же лексемы и нормализованные лексемы мы будем считать приблизительным эквивалентом слов. Далее мы предположим, что первые три этапа уже выполнены, и рассмотрим процесс создания инвертированного индекса на основе индексирования с сортировкой.

Будем считать, что в коллекции документов каждый документ имеет уникальный последовательный номер, который называется идентификатором документа (docID). В ходе построения индекса мы можем просто присваивать последовательные номера каждому новому документу при первом обращении. Входной информацией для индексирования является список нормализованных лексем для каждого документа, который мы можем рассматривать как список пар "термин–docID", как показано на рис. 1.4. Основным этапом в процессе индексирования является такая сортировка (sorting) списка, в результате которой термины располагаются в алфавитном порядке, как показано в средней колонке на рис. 1.4. Затем многократные повторения одного и того же термина в одном и том же документе объединяются. [5] После этого экземпляры одного и того же термина группируются, а результат разделяется на словарь (dictionary) и словопозиции (postings), как показано в правой части рис. 1.4. Поскольку термин обычно встречается во многих документах, этот способ организации данных уже уменьшает объем индекса. В словаре также хранятся некоторые статистические показатели, например количество документов, содержащих каждый термин, т.е. документная частота (document frequency), которая в данном случае равна длине каждого списка словопозиций. Эта информация не существенна для базовой системы булева поиска, но позволяет ускорить обработку запросов, и эти данные могут быть использованы позже во многих ранжирующих моделях информационного поиска. Затем списки позиций сортируются по идентификаторам документов, что позволяет обеспечить эффективную обработку запросов. Такая структура инвертированного индекса практически не имеет конкурентов, поскольку является наиболее эффективной для текстового поиска по произвольному запросу.

Такая структура индекса требует памяти для хранения словаря и списков словопозиций. Списки словопозиций намного крупнее, но словари обычно хранятся в памяти, а списки — на жестком диске, поэтому оптимизация размера каждого из них играет важную роль. В главе 5 мы покажем, как оптимизировать хранение этих данных и доступ к ним. Какой должна быть структура данных для списков словопозиций? Массив фиксированной длины может оказаться слишком затратным; одни слова могут встречаться во многих документах, а другие — лишь в нескольких. Для списков словопозиций, хранящихся в памяти, существуют две альтернативы: односвязные списки или массивы переменной длины. Односвязные списки (singly linked lists) позволяют просто вставлять документы в списки словопозиций (например, при обновлении документов в ходе повторного обхода веба в поисках обновившихся страниц) и естественным образом применять усовершенствованные структуры индекса, например списки с пропусками (раздел 2.3), требующие дополнительных указателей. Массивы переменной длины позволяют сэкономить память, избежав издержек на указатели, и получить выигрыш в быстродействии за счет использования непрерывных областей памяти на современных процессорах с кэшем. На практике дополнительные указатели можно встроить в списки как смещения.

---

[5] Пользователи операционной системы Unix могут заметить, что эти этапы аналогичны использованию команд `sort` и `uniq`.

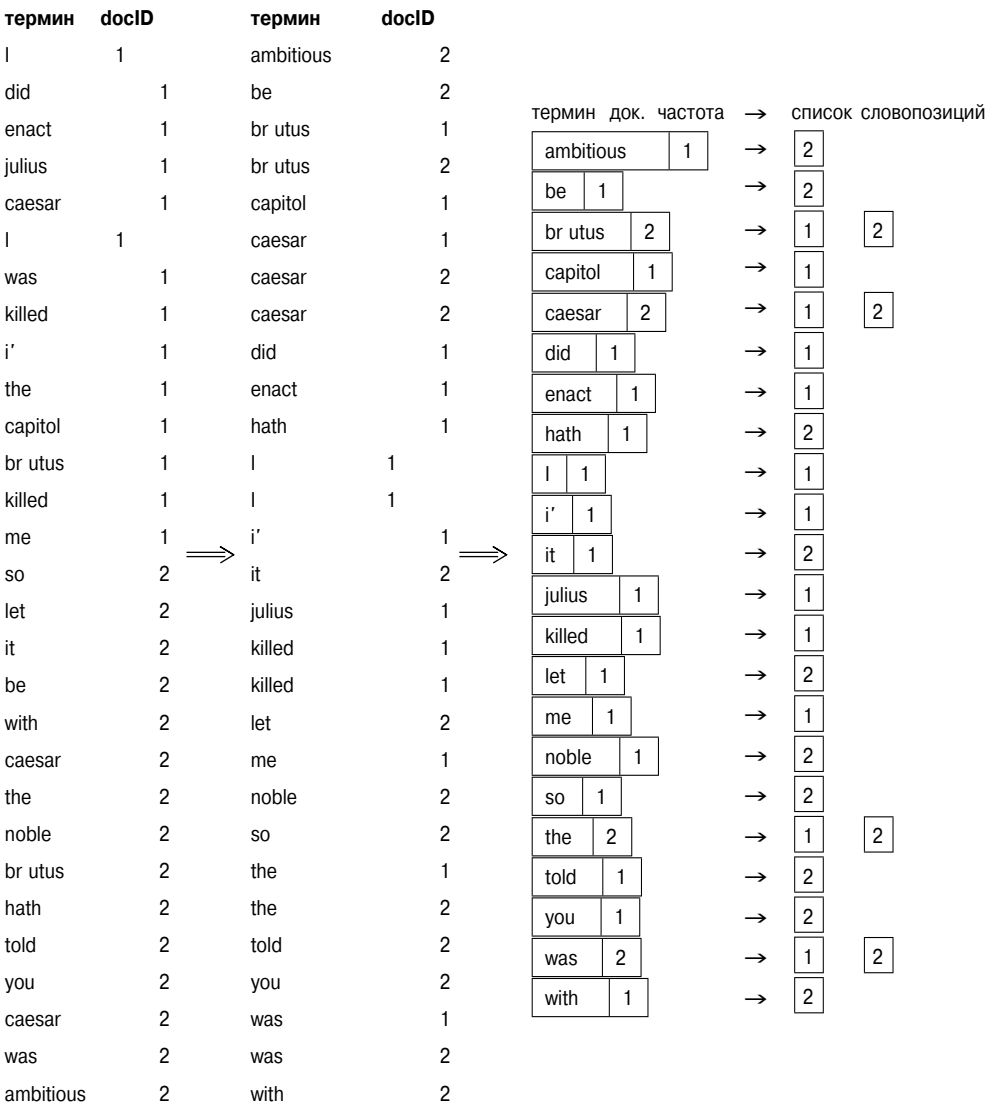


Рис. 1.4. Построение индекса с помощью сортировки и группирования. Последовательность терминов в каждом документе в сопровождении соответствующих идентификаторов документов (слева) упорядочивается в алфавитном порядке (посередине). После этого экземпляры одинаковых терминов группируются по слову, а затем — по идентификатору документов. Далее термины и идентификаторы документов отделяются друг от друга (справа). Термины хранятся в словаре вместе с указателем на список словопозиций для каждого термина. Распространенной практикой является хранение вместе с терминами другой описательной информации, например документной частоты для каждого термина. Эта информация используется для повышения эффективности обработки запросов и позднее — для взвешивания в моделях ранжирующего поиска. В каждом списке словопозиций хранится список документов, содержащих данный термин, а иногда и другая информация, например частота термина (частота каждого термина в каждом документе) или координата (координаты) термина в каждом документе

Если информация обновляется относительно редко, то массивы переменной длины оказываются более компактными, и проход по ним осуществляется быстрее. Можно также использовать гибридные схемы со связанным списком массивов фиксированной длины для каждого термина. Если списки словопозиций хранятся на диске (возможно, в сжатом виде), то они хранятся непрерывным блоком без явных указателей (как на рис. 1.3), чтобы минимизировать размер списка словопозиций и количество позиционирований головки диска для считывания списков в память.

? ***Упражнение 1.1*** [*]. Постройте инвертированный индекс для следующей коллекции документов. (См., например, рис. 1.3.)

Doc 1 new home sales top forecasts

Doc 2 home sales rise in july

Doc 3 increase in home sales in july

Doc 4 july new home sales rise

Упражнение 1.2 [*]. Рассмотрите следующие документы.

Doc 1 breakthrough drug for schizophrenia

Doc 2 new schizophrenia drug

Doc 3 new approche for treatment of schizophrenia

Doc 4 new hopes for schizophrenia patients

1. Постройте матрицу инцидентности “термин–документ” для этой коллекции документов.
2. Постройте инвертированный индекс для этой коллекции, как показано на рис. 1.3.

Упражнение 1.3 [*]. Какие результаты вы получите в ответ на запросы относительно коллекции документов, изображенной на рис. 1.2?

1. schoziphrenia AND drug
2. for AND NOT (drug OR approach)

1.3. Обработка булевых запросов

Как происходит обработка запросов с помощью инвертированного индекса и базовой модели булева поиска? Рассмотрим обработку простого конъюнктивного запроса (simple conjunctive query)

Brutus and Calpurnia (1.1)

по инвертированному индексу, частично показанному на рис. 1.3. Эта обработка сводится к следующему.

1. Обнаруживаем термин Brutus в словаре.
2. Находим список его словопозиций.
3. Обнаруживаем термин Calpurnia в словаре.
4. Находим список записи его словопозиций.
5. Находим пересечение этих двух списков, как показано на рис. 1.5.

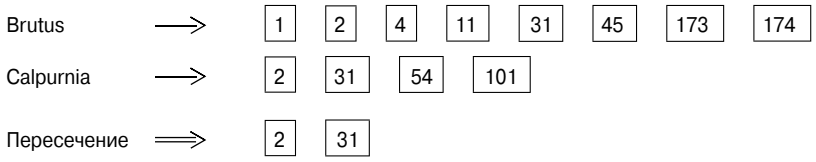


Рис. 1.5. Пересечение списков словопозиций для терминов Brutus *и* Calpurnia *из рис. 1.3*

Операция пересечения (intersection) является чрезвычайно важной: для того чтобы быстро найти документы, содержащие оба термина, необходимо обеспечить эффективное пересечение этих списков. (Эту операцию иногда называют слиянием (merging) списков словопозиций, хотя такое название немного противоречит интуиции, так как оно связано с использованием термина алгоритмы слияния (merge algorithms), обозначающего семейство алгоритмов, которые объединяют несколько упорядоченных списков с помощью чередования указателей, которые движутся по каждому из списков; в данном случае речь идет о слиянии списков с помощью логической операции AND.)

Существует простой и эффективный метод пересечения списков словопозиций с помощью алгоритма слияния (рис. 1.6). Мы поддерживаем указатели на оба списка и проходим по этим спискам одновременно за время, линейно зависящее от количества словопозиций в списках. На каждом этапе мы сравниваем идентификаторы документов, на которые ссылаются оба указателя. Если они совпадают, то мы помещаем идентификатор документа в список результатов, а затем продвигаем оба указателя вперед. В противном случае мы продвигаем вперед только указатель, соответствующий списку с меньшим идентификатором. Если длины списков словопозиций равны x и y, то пересечение осуществляется за O(x + y) операций. С формальной точки зрения сложность обработки запроса равна Θ(N), где N — количество документов в коллекции.[6] Сложность такого метода

```
INTERSECT(p1, p2)
 1  answer ← ⟨⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then Add(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```

Рис. 1.6. Алгоритм пересечения двух списков словопозиций p_1 и p_2

[6] Символ Θ(·) представляет собой строгую асимптотическую оценку сложности алгоритма. Неформально говоря, его часто записывают как O(·), хотя этот символ относится к верхней асимптотической оценке, которая может не быть строгой (Cormen et al., 1990).

индексирования отличается от последовательного просмотра лишь константой, но на практике эта константа часто оказывается огромной. Для использования данного алгоритма важно, чтобы словопозиции были упорядочены в соответствии с единым глобальным критерием. Для этого можно, например, использовать сортировку числовых идентификаторов документов docID.

Этот алгоритм можно обобщить для обработки более сложных запросов, таких как

$$\text{(Brutus or Caesar) AND NOT Calpurnia} \tag{1.2}$$

Оптимизация запроса — это выбор такого способа организации обработки запроса, чтобы можно было минимизировать общий объем работы, которую должна выполнить система. Основным фактором, влияющим на эффективность обработки булевых запросов, является порядок, в котором осуществляется доступ к спискам словопозиций. Какой наилучший порядок обработки запроса? Рассмотрим запрос, состоящий из t терминов, объединенных операцией AND. Например,

$$\text{Brutus AND Caesar AND Calpurnia} \tag{1.3}$$

Для каждого из t терминов необходимо сначала найти списки словопозиций, а затем применить к ним операцию AND. На практике используется стандартный эвристический прием, который заключается в том, что термины обрабатываются в порядке возрастания частоты документов; если начать с пересечения двух наименьших списков словопозиций, то все промежуточные результаты не должны превышать наименьшего списка словопозиций, а значит, мы выполним при этом наименьший объем работы. Итак, для списков словопозиций, представленных на рис. 1.3, обработка запроса должна выглядеть так.

$$\text{(Calpurnia AND Brutus) AND Caesar} \tag{1.4}$$

Это первый аргумент в пользу того, чтобы подсчитывать частоту терминов в словаре; это позволяет принять решение об упорядочении на основе данных, хранящихся в памяти компьютера, не требуя доступа к спискам словопозиций.

Рассмотрим теперь оптимизацию запросов более общего вида.

$$\text{(madding AND crowd) AND (ignoble OR strife) AND (killed OR slain)} \tag{1.5}$$

Как и раньше, определим частоту каждого термина, а затем получим оценки сверху размера для каждого оператора OR, суммируя частоты его операндов. После этого запрос можно обрабатывать в порядке возрастания размера каждого дизъюнктного термина.

Для произвольных булевых запросов мы должны найти и временно сохранить ответы для промежуточных выражений, входящих в более сложные выражения. Однако во многих ситуациях запросы являются исключительно конъюнкцией терминов. Это объясняется либо природой языка запросов, либо широкой популярностью такой формы запросов среди пользователей. В этом случае вместо просмотра объединенных списков словопозиций с помощью функции, имеющей два входных аргумента и отдельное возвращаемое значение, эффективнее пересечь все результирующие списки словопозиций с текущим промежуточным результатом в памяти. При этом промежуточный результат инициализируется путем загрузки списка словопозиций самого редко встречающегося термина. Этот алгоритм показан на рис. 1.7. Из его описания следует, что операция пересечения асимметрична: промежуточные результаты хранятся в памяти, в то время как очередной список, с которым они пересекаются, считывается с диска. Кроме того, длина списка промежуточных результатов никогда не превышает длину других списков, а обычно на

порядок короче этих списков. Конечно, пересечение словопозиций можно осуществить и с помощью алгоритма, описанного на рис. 1.6, но, если длины списков сильно отличаются одна от другой, возникает возможность применить альтернативные методы. В этих методах (рис. 1.7), как было сказано выше, очередной искомый объект берется из последовательно считываемого с диска длинного инвертированного списка, а пересечение совершается с помощью удаления объектов или их пометки как удаленных в хранящемся в памяти контейнере для промежуточных результатов. Это возможно с помощью (1) прямого доступа (если промежуточные результаты представлены в памяти как вектор) либо (2) бинарного поиска, либо (3) хеш-таблицы. Как бы то ни было, вхождение очередного документа в список промежуточных результатов можно вычислить за константное время. Однако такие альтернативные методы трудно сочетать со сжатыми инвертированными списками наподобие описанных в главе 5. Более того, без стандартных операций пересечения списков словопозиций нельзя обойтись, если оба термина запроса являются высокочастотными.

```
INTERSECT(⟨t1, ..., tn⟩)
1   terms ← SortByIncreasingFrequency(⟨t1, ..., tn⟩)
2   result ← postings(first(terms))
3   terms ← rest(terms)
4   while terms ≠ NIL and result ≠ NIL
5   do result ← Intersect(result, postings(first(terms))
6            terms ← rest(terms)
7   return results
```

Рис. 1.7. Алгоритм обработки конъюнктивных запросов, возвращающий набор документов, которые содержат каждый из терминов, указанных во входном списке

? ***Упражнение 1.4*** [*]. Можно ли для каждого из запросов, указанных ниже, выполнить операции пересечения за время O(x+y), где x и y — длины списков словопозиций для терминов Brutus и Caesar? Если нельзя, то каких результатов следует ожидать?

1. Brutus AND NOT Caesar
2. Brutus OR NOT Caesar

Упражнение 1.5 [*]. Усовершенствуйте алгоритм слияния списков словопозиций для произвольных булевых запросов. Какова его временная сложность? Рассмотрите следующий пример.

1. (Brutus OR Caesar) AND NOT (Antony OR Cleopatra)

Всегда ли можно выполнить слияние за линейное время? Относительно какого параметра эта оценка является линейной? Можно ли ее улучшить?

Упражнение 1.6 [**]. Для изменения формы запроса можно использовать дистрибутивные законы для операций AND и OR.

1. Покажите, как с помощью дистрибутивных законов можно преобразовать запрос из примера 1.5 в дизъюнктную нормальную форму.
2. Можно ли утверждать, что полученный запрос будет более (или менее) эффективно обрабатываться, чем исходный?
3. Этот результат верен в любой ситуации или он зависит от слов и содержания коллекции документов?

Упражнение 1.7 [*]. Порекомендуйте порядок обработки следующего запроса.

1. (tangerine OR trees) AND (marmalade OR skies) AND (kaleidoscope OR eyes)

Будем считать, что список словопозиций имеет следующий вид.

Термин	**Размер списка словопозиций**
eyes	213312
kaleidoskope	87009
marmalade	107913
skies	271658
tangerine	46653
trees	316812

Упражнение 1.8 [*]. Допустим, что запрос имеет следующий вид.

1. friends AND romans AND (NOT countrymen)

Как использовать частоту термина countrymen для оценки наилучшего порядка обработки запроса? В частности, предложите способ выполнения оператора NOT при определении порядка обработки запроса.

Упражнение 1.9 [**]. Обеспечивает ли обработка списков словопозиций в порядке возрастания их размера оптимальную обработку конъюнктивного запроса? Если да, то аргументируйте свой ответ, а если нет, приведите пример.

Упражнение 1.10 [**]. Напишите программу, реализующую алгоритм слияния, показанный на рис. 1.6, для запроса x OR y.

Упражнение 1.11 [**]. Как обработать булев запрос x AND NOT y? Почему наивная оценка для этого запроса слишком завышена? Напишите программу, реализующую алгоритм слияния списков словопозиций и обеспечивающую эффективную обработку запроса.

1.4. Сравнение расширенной булевой модели и ранжированного поиска

Альтернативой модели булева поиска являются модели поиска с ранжированием (ranked retrieval models), например модель векторного пространства (раздел 6.3), в рамках которых пользователи в основном применяют свободные текстовые запросы (free

text queries), т.е. набирают одно или несколько слов, а не используют строгие языковые конструкции с операторами; система же сама решает, какие документы лучше других удовлетворяют этим запросам. Несмотря на десятилетия академических исследований преимуществ ранжирующего поиска, системы, основанные на модели булева поиска, до начала 1990-х годов (приблизительная дата появления сети World Wide Web) 30 лет оставались основным или даже единственным средством поиска в крупных коммерческих информационных службах. Однако эти системы использовали не только логические операции (AND, OR и NOT), которые мы рассматривали до сих пор. Строгие логические выражения с терминами с неупорядоченными результатами накладывают слишком много ограничений при поиске информации и не позволяют полностью удовлетворить информационные запросы пользователей, поэтому в этих системах были реализованы расширенные модели булева поиска с дополнительными операторами, такими как операторы близости терминов. Оператор учета расстояния между словами запроса (proximity operator) позволяет указать, насколько близко друг к другу должны быть расположены термины запроса в документе, причем близость между терминами может измеряться количеством промежуточных слов или ссылаться на структурные единицы текста, такие как предложения и абзацы.

Пример 1.1. **Коммерческая служба булева поиска: Westlaw**. Westlaw (`http://www.westlaw.com/`) — крупнейшая юридическая поисковая служба (по количеству подписчиков). Ежедневно полмиллиона ее подписчиков выполняют миллионы поисковых запросов в десятках терабайтов текстовых данных. Она основана в 1975 году. В 2005 году булев поиск (который в системе Westlaw называется *Terms and Connectors*) оставался основным механизмом поиска и использовался большим количеством пользователей, хотя еще в 1992 году у них появилась возможность представлять свободные текстовые запросы (которые в системе Westlaw называются *Natural Language*). Рассмотрим несколько примеров булевых запросов в системе Westlaw.

> *Информационные потребности*: информация о юридических теориях, связанных с предотвращением раскрытия коммерческой тайны уволенными сотрудниками, перешедшими на службу в конкурирующие компании.
>
> *Запрос*: “trade secret” /s disclos! /s prevent /s employee!
>
> *Информационные потребности*: требования людей с ограниченными возможностями, касающиеся доступа к рабочему месту.
>
> *Запрос*: disab! /p access! /s work-site work-place (employment /3 place)
>
> *Информационные потребности*: дела, касающиеся ответственности хозяев за поведение пьяных гостей.
>
> *Запрос*: host! /p (responsib! liab!) /p (intoxicat! drunk!) /p guest

Обратите внимание на длинный и точный запрос и использование операторов близости, не характерных для веба. В среднем предъявляемые запросы состоят из десяти слов. В отличие от правил, установленных в вебе, пробелы между словами представляют собой дизъюнкцию (оператор наиболее сильного связывания), символ & означает оператор AND, а символы /s, /p и /k устанавливают поиск совпадений в одном и том же предложении, в абзаце или в окрестности *k* слов соответственно. Двойные кавычки означают *фразовый поиск* (phrase search), т.е. поиск последовательных слов (см. раздел 2.4). Знак восклицания (!) представляет собой замыкающий джокер (см. раздел 3.2); таким образом,

слово liab! соответствует всем словам, начинающимся буквами liab. Слова work-site соответствуют любому варианту: *worksite, work-site* и *work site* (см. раздел 2.2.1). Типичный запрос эксперта обычно тщательно формулируется и постепенно уточняется, пока не приведет к желаемому результату.

Многие пользователи, особенно профессионалы, предпочитают использовать булевы запросы. Такие запросы отличаются точностью: документ либо удовлетворяет запросу, либо не удовлетворяет. Это дает пользователю больше возможностей для контроля и обеспечивает прозрачность результатов. Некоторые области знаний, такие как юриспруденция, допускают эффективное ранжирование документов в рамках булевой модели: система Westlaw возвращает документы в обратном хронологическом порядке, что на практике достаточно эффективно. В 2007 году большинство библиотекарей, работающих с юридической литературой, продолжали рекомендовать использовать термины и логические операции для поиска с высокой полнотой, а большинство пользователей информационно-правовых систем считают, что они получают с их помощью больший контроль над системой. Однако это не значит, что булевы запросы более эффективны для людей, которые профессионально занимаются поиском. На самом деле, экспериментируя с частью коллекции документов в системе Westlaw, Тертл (Turtle, 1994) выяснил, что в большинстве случаев свободные текстовые запросы приводят к лучшим результатам, чем булевы запросы, подготовленные профессиональными библиотекарями — сотрудниками Westlaw, для информационных потребностей в рамках эксперимента. Общая проблема, связанная с булевыми запросами, заключается в том, что использование оператора AND повышает точность запроса, но понижает полноту поиска, в то время как оператор OR снижает точность, но повышает полноту поиска, а компромисс между ними найти очень трудно или даже невозможно.

В этой главе мы изучили структуру и процесс создания базового инвертированного индекса, включая словарь и списки словопозиций. Мы ввели модель булева поиска и рассмотрели, как осуществить эффективный поиск на основе алгоритмов слияния с линейной сложностью и простой оптимизацией запросов. В главах 2–7 мы подробнее рассмотрим более сложные модели запросов и более сложные структуры индекса, необходимые для эффективной обработки таких запросов. Здесь же мы просто перечислим некоторые свои пожелания.

1. Мы хотели бы лучше определять набор терминов в словаре и осуществлять поиск, малочувствительный к опечаткам и нечеткому выбору слов.
2. Часто хотелось бы иметь возможность находить сложносоставные слова или фразы, обозначающие какое-то понятие, например “операционная система”. Как свидетельствуют примеры, связанные с системой Westlaw, хотелось бы также иметь возможность формулировать запросы о близости терминов, например Gates NEAR Microsoft. Для ответа на такие запросы индекс должен быть пополнен средствами идентификации близости между терминами в документах.
3. Модель булева поиска позволяет лишь определить наличие или отсутствие термина, но часто хотелось бы накапливать данные, присваивая больший вес документам, в которых термин встречается несколько раз, и меньший вес — документам, в которых термин встречается лишь однажды. Для этого в список словопозиций необходимо ввести информацию о *частоте термина*, т.е. о количестве появлений термина в документе.

4. В ответ на булевы запросы просто возвращается (неупорядоченное) множество документов, но обычно нам необходим эффективный метод, позволяющий упорядочить (*ранжировать*) результаты поиска. Для этого нужен механизм определения ранга документа, который включал бы в себя степень соответствия документа запросу.

Эти дополнительные идеи легли в основу большинства современных технологий, поддерживающих поиск по произвольному запросу в массивах неструктурированной информации. Поиск по произвольному запросу среди документов недавно завоевал мир; он не только был реализован в поисковых системах веба, но и лег в основу поиска на крупных веб-сайтах электронной торговли. Несмотря на то что поисковые системы веба делают акцент на обработке свободных текстовых запросов, базовые механизмы и технологии индексации и обработки запросов принципиально не отличаются от изложенных в этой главе, как мы увидим далее. Более того, с течением времени поисковые системы веба включили в себя частичную реализацию некоторых наиболее популярных операторов из расширенной булевой модели (особенно популярен поиск фраз). Большинство поисковых систем имеет частичную реализацию логических операторов. Тем не менее, несмотря на то что эти опции очень нравятся “профессионалам поиска”, они мало используются большинством пользователей и не принадлежат к основным методам повышения эффективности работы поисковых веб-систем.

? ***Упражнение 1.12*** [*]. Напишите запрос, используя синтаксис системы Westlaw, и найдите слова professor, teacher и lecturer, входящие в одно и то же предложение вместе с любой формой слова explain.

Упражнение 1.13 [*]. Испытайте основные механизмы булева поиска в наиболее распространенных поисковых веб-системах. Например, выберите слово burlgar и задайте запросы (1) burlgar, (2) burlgar AND burglar и (3) burglar OR burglar. Просмотрите примерное количество результатов и ответы, расположенные в начале списка. Соответствуют ли они логике запроса? Довольно часто ответ на этот вопрос оказывается отрицательным. Можете ли вы предположить, почему это происходит? А что произойдет, если поискать другие слова? Попробуйте представить другой запрос, например (1) knight, (2) conquer и (3) knight OR conquer. Какая связь существует между результатами обработки двух первых запросов и третьего запроса? Обнаруживается ли эта связь?

1.5. Библиография и рекомендации для дальнейшего чтения

Практические исследования поиска компьютеризированной информации начались в конце 1940-х годов (Cleverdon, 1991; Liddy, 2005). Огромный объем научной литературы, часто представленной в виде менее формальных технических отчетов, а не традиционных журнальных статей, а также возросшая доступность компьютеров, повысили интерес к автоматизированному поиску документов. Однако в то время поиск документов выполнялся по фамилии автора, заглавию и ключевым словам; полнотекстовый поиск появился позднее.

Привлекательность этой области исследований была обоснована в статье Буша (Bush), опубликованной в 1945 году.

> Представим себе будущее устройство для индивидуального пользования, похожее на личный архив или библиотеку. Назовем его “memex”. На этом устройстве человек сможет хранить все свои книги, записи и письма и консультироваться с ним с необыкновенной гибкостью и скоростью. Это расширенная память человека.

Термин информационный поиск (information retrieval) был введен Кельвином Муерсом (Calvin Mooers) в 1948–1950 годы (Mooers, 1950).

В 1958 году газеты облетела новость о демонстрации на конференции машины для автоматической индексации, созданной компанией IBM (Taube and Wooster, 1958). В основу этой машины были положены работы Х. Луна (H.P. Luhn). Коммерческий интерес быстро привел к появлению систем булева поиска, но в первые годы вокруг используемых технологий велись бурные дискуссии. Например, в 1961 году Муерс писал следующее.

> Распространенное заблуждение, которое подкрепляется многомиллионными инвестициями в разные аппаратные поисковые устройства, заключается в том, что алгебра Джорджа Буля (1847) является приемлемым формализмом для разработки поисковых систем. Эта точка зрения столь же общепринята, сколь и ошибочна.

Исследование операторов AND и OR, демонстрирующее противоречия, а не компромисс между точностью и полнотой, можно найти в работе Ли и Фокса (Lee and Fox, 1988).

Книга Уиттена и др. (Witten et al., 1999) представляет собой стандартный справочник, в котором проведено глубокое сравнение инвертированного индекса с другими структурами данных как с точки зрения памяти, так и с точки зрения скорости работы; книга Цобеля и Моффата (Zobel and Moffat, 2006) — более краткая и современная публикация на эту тему. Эти вопросы мы рассмотрим в главе 5.

Практические аспекты использования регулярных выражений для поиска рассмотрены в работе Фридла (Friedl, 2006). Теоретические аспекты этих проблем изложены в книге Хопкрофта и др. (Hopcroft et al., 2000).

Глава 2

Лексикон и списки словопозиций

Напомним основные этапы построения инвертированного индекса.

1. Собираем документы, подлежащие индексированию.
2. Разбиваем текст на лексемы.
3. Выполняем предварительную лингвистическую обработку лексем.
4. Индексируем документы по каждому термину.

В этой главе мы сначала кратко опишем, как определить основные единицы документа и обнаружить последовательность символов, которые в нем содержатся (раздел 2.1). После этого мы подробно исследуем несколько важных тем, связанных с выделением лексем и предварительной лингвистической обработкой, в результате которой возникает лексикон терминов, используемый системой (раздел 2.2). *Выделение лексем* (tokenization) — это процесс разделения потока символов на лексемы. В ходе предварительной лингвистической обработки возникают классы эквивалентных лексем, образующие множество терминов, по которым происходит индексирование. Индексирование описано в главах 1 и 4. Затем рассматривается реализация инвертированных списков. В разделе 2.3 речь идет о расширенной структуре инвертированного списка, повышающей скорость обработки запросов, а в разделе 2.4 — о построении структуры словопозиции, подходящей для обработки фразовых запросов и запросов с указанием расстояния, которые часто встречаются как в расширенных булевых моделях, так и в вебе.

2.1. Схематизация документа и декодирование последовательности символов

2.1.1. Выделение последовательности символов в документе

Цифровые документы, являющиеся входной информацией для процесса индексирования, как правило, представляют собой набор байтов в файле или на веб-сервере. На первом этапе обработки эта последовательность байтов преобразуется в линейную последовательность символов. Для английского текста, набранного в системе кодирования ASCII, такая задача тривиальна. Однако часто задачи оказываются намного сложнее. Последовательность символов может быть закодирована в одной из многих одно- или многобайтовых кодировок, например UNICODE UTF-8, а также в национальном стандарте или в стандарте, зависящем от поставщика. Сначала необходимо определить правильную кодировку. Эту проблему можно интерпретировать как задачу классификации на основе

машинного обучения (см. главу 13 [1]), но на практике ее часто решают с помощью эвристических методов, выбора пользователя или имеющихся метаданных о документе. После выявления кодировки последовательность байтов преобразуется в последовательность символов. Выбор кодировки следует зафиксировать, поскольку это дает некоторое представление о языке, на котором написан документ.

Возможно, символы нужно декодировать из двоичного представления, например из doc-файла текстового процессора Microsoft Word и/или архивных файлов наподобие zip-файлов. Следовательно, сначала необходимо определить формат документа, а затем выбрать соответствующий декодер. Даже для простых текстовых документов может понадобиться дополнительное декодирование. В XML-документах (раздел 10.1) некоторые символы, такие как `&`, следует декодировать специально, так, чтобы выражению `&` соответствовал именно символ `&`, а не что-либо другое. В заключение текстовую часть документа может понадобиться отделить от другого материала, который не подвергается обработке. Это часто практикуется при обработке XML-файлов, если требуется проигнорировать разметку, а также почти всегда при обработке postscript- и PDF-файлов. Мы больше не будем возвращаться к этому вопросу в нашей книге и будем предполагать, что наши документы представляют собой списки символов. Коммерческие программы обычно поддерживают обработку широкого спектра форматов и кодировок документов, поскольку пользователи желают, чтобы программы работали с имеющимися данными. Часто они трактуют документ как текст, помещенный в какую-то программу, и совершенно не интересуются, как именно он записан на диск. Эта проблема обычно решается с помощью лицензирования программных библиотек, обрабатывающих форматы документов и кодировки.[2]

Трактовка документа как линейной последовательности символов часто оказывается неадекватной в некоторых системах письменности, например в арабских языках, в которых текст является двумерным и не имеет строгого порядка следования символов, как показано на рис. 2.1 и 2.2. Однако, несмотря на довольно сложные правила письма, тексту соответствует последовательность звуков и, таким образом, сохраняется по существу линейная структура. Именно это явление продемонстрировано на рис. 2.1 на примере цифрового представления документа на современном арабском языке.

---

[1] Классификатор — это функция, получающая объекты определенного вида и распределяющая их по различным классам. Обычно классификация осуществляется с помощью методов машинного обучения, например с помощью вероятностных моделей, но может быть реализована и на основе эмпирических правил.

[2] На практике системы веб-поиска сначала обрабатывают разные форматы файлов, а затем определяют кодировку и язык. При обработке разных форматов возникает задача сохранения информации о разметке текста (заголовков, подзаголовков, списков, ссылок), которая часто содержится в них (см. раздел 6.1). Еще один аспект — толерантность к ошибкам формата, например к “супу тегов”, описанному в черновике спецификации HTML5. При определении кодировки стоит отметить связь этой задачи с определением языка, без предположений о котором часто невозможно корректно определить кодировку. — *Примеч. ред.*

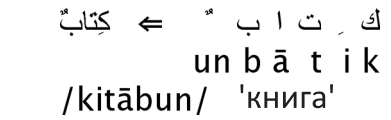


Рис. 2.1. Пример озвученного слова в современном арабском языке. Запись выполняется справа налево, а буквы сплетаются в сложные сочетания. Краткие гласные (здесь — /i/ и /u/), а также финальная буква /n/ (нунация) нарушают линейный порядок следования и изображаются диакритическими символами выше и ниже букв. Несмотря на это представленный текст, несомненно, представляет собой последовательную кодировку звуков. Полное озвучивание, продемонстрированное здесь, обычно встречается только в Коране и в детских книгах. Повседневные тексты либо не озвучиваются вообще (краткие гласные не выделяются, хотя буква ā по-прежнему встречается), либо озвучивается частично, когда краткие гласные вставляются в те места, где может возникнуть неоднозначность. Все это создает дополнительные сложности для индексирования

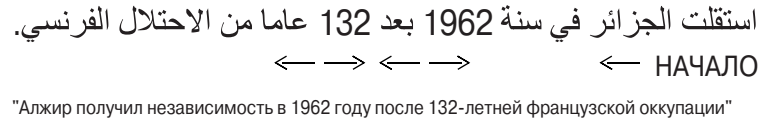


Рис. 2.2. Концептуальный линейный порядок символов необязательно соответствует порядку символов на странице. В языках с письмом справа налево, например в иврите и арабском, встречается порядок следования слева направо, например, при указании чисел и денежных единиц. В современной системе Unicode порядок символов в файле соответствует концептуальному порядку, а обратный порядок представления символов имитируется системой визуализации, хотя для документов, записанных в старой системе кодирования, это правило может не выполняться

2.1.2. Выбор структурной единицы документа

На следующем этапе определяется *структурная единица документа* (document unit) для индексирования. До сих пор мы предполагали, что документы представляют собой фиксированные единицы, предназначенные для индексирования. Например, мы можем рассматривать каждый файл в каталоге как документ. Однако во многих ситуациях может понадобиться сделать нечто иное. Например, в системе Unix (формат mbox) последовательность электронных сообщений (каталог писем) хранится в одном файле, но вы можете пожелать работать с электронными сообщениями, как с отдельными документами. Многие электронные сообщения содержат присоединенные файлы, и вы можете считать сообщение и каждый присоединенный файл отдельными документами. Если к электронному сообщению присоединен архивный файл, то вы можете его распаковать и рассматривать каждый файл, который в нем содержится, по отдельности. Можно привести и противоположные примеры. Некоторые веб-приложения (такие, как latex2html) работают с единым документом (например, с файлом Powerpoint или документом LATEX) и разделяют их на отдельные HTML-страницы для каждого слайда или подраздела, хранящегося в отдельных файлах. В этих ситуациях желательно объединить многие файлы в один документ.

Для очень длинных документов возникает проблема *детализации индексирования* (indexing granularity). Работая с коллекцией книг, было бы нецелесообразно индексировать всю книгу, как отдельный документ. В этом случае поиск фразы Chinese toys привел бы к тому, что в ответ вы получили бы книгу, в которой слово China встречается в первой главе, а слово toys — в последней, что было бы неудовлетворительным результатом. Вместо этого следовали бы индексировать каждую главу или абзац как мини-документ. В этом случае совпадения оказались бы более релевантными, поскольку документы уменьшились бы и пользователь быстрее находил бы в них релевантные фрагменты. Однако почему на этом следует остановиться? Можно было бы рассматривать отдельные предложения как мини-документы. Очевидно, что в этом случае возникает противоречие между точностью и полнотой поиска. Если единицы разбиения слишком маленькие, то мы можем пропустить важную информацию из-за того, что термины могут быть распределены по разным мини-документам. Если же единицы разбиения слишком велики, то мы чаще будем получать фиктивные совпадения и пользователю труднее найти релевантную информацию.

Проблемы, возникающие при работе с большими документами, можно решить с помощью явного или неявного *поиска с учетом близости слов запроса в документе* (proximity search), описанного в разделах 2.4.2 и 7.2.2. Влияние этих методов поиска на производительность систем обсуждается в главе 8. Проблема детализации индекса, в частности необходимость одновременной индексации документов на разных уровнях детализации, ярко проявляется при поиске информации с помощью языка XML. Эта проблема рассматривается в главе 10. Система информационного поиска должна предоставлять возможность выбора уровня детализации. Для того чтобы этот выбор был правильным, тот, кто внедряет эту систему, должен иметь хорошее представление о коллекции документов, а также об информационных потребностях и привычках пользователей. Пока будем предполагать, что мы выбрали элемент индексирования подходящего размера, а также способ разделения или агрегации файлов, если это необходимо.

2.2. Определение лексикона терминов

2.2.1. Разделение текста на лексемы

После идентификации последовательности символов и выделения структурных единиц документа текст разделяется на *лексемы*. Кроме того, иногда из него одновременно удаляют некоторые символы, например знаки пунктуации. Рассмотрим пример.

Ввод: Friends, Romans, Countrymen, lend me your ears

Вывод:

Friends	Romans	Countrymen	lend	me	your	ears

Эти лексемы иногда ошибочно называют терминами или словами, но иногда следует четко отличать класс лексем и экземпляр лексемы (type/token). *Лексема* (tokens) — это экземпляр последовательности символов в определенном документе, объединенных в семантическую единицу для обработки. *Тип* (type) — это класс всех лексем, состоящих из одной и той же последовательности символов. *Термин* (term) — это (возможно, нормализованный) тип, включенный в словарь системы информационного поиска. Множество терминов индекса может полностью отличаться от лексем, которые, например, могут

быть семантическими идентификаторами в иерархии, но на практике в современных системах информационного поиска они напрямую связаны с лексемами в документе. Однако термины – это не лексемы в точно таком виде, в каком они встречаются в документе. Обычно к ним применяют различные процессы нормализации, рассматриваемые в разделе 2.2.3. [3] Например, если индексируемым документом является фраза *to sleep perchance to dream*, то существует пять лексем, но лишь четыре типа (поскольку в эту фразу частица *to* входит дважды). Однако если частица *to* не включена в индекс (как стоп-слово; см. раздел 2.2.2), то останутся только три термина: *sleep*, *perchance* и *dream*.

Основной вопрос, связанный с разделением текста на лексемы, звучит так: "Как правильно разделить текст на лексемы?" В нашем примере ответ очевиден: достаточно разделить текст по пробелам и отбросить знаки пунктуации. Это отправная точка, но даже в английском языке здесь таится много сложностей. Например, что делать с разными формами апострофа, используемого для образования притяжательной формы и сокращений.

Mr. O'Neill thinks that the boys' stories about Chile's capital aren't amusing.

Какое из следующих разбиений является правильным для имени O'Neill?

neill	
oneill	
o'neill	
o'	neill
o	neill

А для *aren't*?

aren't	
arent	
are	n't
aren	t

Самая простая стратегия сводится к разбиению по небуквенным символам, но хотя лексема

o	neill

выглядит правильно, интуитивно очевидно, что лексема

aren	t

неверна. Каждый вариант определяет, каким булевым запросам будут найдены соответствия. Запросу neill AND capital соответствуют три варианта из пяти. А сколько раз был бы удовлетворен запрос o'neill AND capital? Без предварительной обработки он будет удовлетворен только в одном варианте из пяти. Как для булевых, так и для свободных

[3] Иначе говоря, неиндексируемые *лексемы* (стоп-слова) — это не термины. Если после нормализации несколько лексем "склеиваются", то они индексируются в нормализованной форме, как один термин. Однако позднее, в главах 13–18, при обсуждении классификации и кластеризации, не использующих индексирование, мы ослабим это определение. В этих главах мы отменим требование о включении в словарь. Термин — это нормализованное слово.

текстовых запросов желательно, чтобы разбиение на лексемы запросов и документов происходило одинаково. Для этого обработка запросов осуществляется с помощью одного и того же лексического анализатора. Это гарантирует, что последовательность символов в тексте всегда совпадает с такой же последовательностью, набранной в запросе. [4]

Эти проблемы существенно зависят от языка. Следовательно, язык документа должен быть известен заранее. *Определение языка* (language identification), основанное на классификаторах, использующих в качестве признаков короткие подпоследовательности символов, является очень эффективным. Большинство языков имеют отличительные характерные черты (см. библиографию в конце главы).

В большинстве языков и конкретных предметных областей существуют необычные специфические лексемы, которые следует распознавать, как термины: языки программирования C++ и C#, названия самолетов, например B-52, или телешоу, например M*A*S*H. Эти слова вошли в культурный контекст, поэтому в текстах можно встретить выражения *M*A*S*H-style hospitals*. [5] Компьютерная технология породила новые типы последовательностей символов, которые следует распознавать, как одну лексему, в частности адреса электронной почты (jblack@mail.yahoo.com), веб-адреса (URL) (http://stuff.big.com/new/specials.html), числовые IP-адреса (142.32.48.231), номера для отслеживания бандеролей (1Z9999W99845399981) и т.д. Можно было бы отказаться от индексирования таких лексем, как денежные суммы, числа и веб-адреса, поскольку их наличие существенно увеличивает размер лексикона. Однако это значительно снизило бы полезность поисковых систем. Например, люди могут искать в базе данных ошибок номер строки программы, в которой возникла проблема. Такие единицы текста, как даты и электронные адреса, имеющие явный семантический тип, часто индексируются отдельно как метаданные документа (см. раздел 6.1).

В английском языке *расстановка дефисов* (hyphenation) используется для разных целей: для разделения *гласных букв* в словах (*co-education*), для объединения существительных в названиях (*Hewlett-Packard*), а также для группирования слов (*the hold-him-back-and-drag-him-away maneuver*). Легко видеть, что первый пример нужно рассматривать как одну лексему (на самом деле это слово чаще записывается как *coeducation*), последний следует разделить на несколько слов, а второй пример остается неясным. Следовательно, автоматическая обработка дефиса может оказаться сложной: можно либо рассматривать ее как задачу классификации, либо применять некие эвристические правила, например допускать в терминах только короткие префиксы.

В принципе, разделение текста по пробелам приводит к разрыву слов, которые следовало бы рассматривать как цельные лексемы. Это часто происходит с названиями (*San Francisco*, *Los Angeles*), а также с фразами, заимствованными из иностранных языков (*au fait*), сложными словами, которые могут записываться как с пробелами, так и слитно (например, *white space* и *whitespace*). К словам с внутренними пробелами, которые сле-

---

[4] Для свободных текстовых запросов это очевидно. В случае логического поиска ситуация усложняется, так как соответствующий генератор лексем может из одного слова запроса порождать несколько терминов. Решить эту проблему можно, объединив термы с помощью оператора *AND* или создав фразовый запрос (см. раздел 2.4). Намного сложнее для системы решить противоположную задачу, когда пользователь вводит два терма, в то время как соответствующие слова в процессе разделения текста на лексемы были объединены в одну лексему.

[5] Речь идет о культовой американской кинокомедии Роберта Олтмена “MASH” (1970), которая на отечественном телевидении шла под названием “Военно-полевой госпиталь MASH”. — *Примеч. ред.*

довало бы рассматривать как отдельную лексему, относятся телефонные номера (800)-234-2333) и даты (Mar 11, 1983). Разделение лексем по пробелам может привести к неверным результатам поиска. Например, при обработке запроса York University в основном возвращаются документы, содержащие словосочетание *New York University*. Проблемы, связанные с дефисом и неразделяющими пробелами, могут оказаться взаимосвязанными. Например, реклама авиационных перелетов часто содержит слова *San Francisco-Los Angeles*. Совершенно очевидно, что в этом случае разделение по пробелам приведет к неправильным результатам. В подобных ситуациях проблемы выделения лексем пересекаются с проблемами обработки запросов на поиск фраз (см. раздел 2.4), в частности, если бы мы потребовали, чтобы запросы на поиск слов *lowercase*, *lower-case* и *lower case* приводили к одинаковым результатам. Последние два запроса можно было бы обработать, проведя разделение лексем по дефисам и использовав фразовый индекс. Для того чтобы получить правильный ответ при обработке первого запроса, необходимо знать, что это сложное слово, состоящее из двух слов, и индексировать его с учетом этой информации. Одна из эффективных и практичных стратегий, реализованных в системах булева поиска Westlaw и Lexis-Nexis (пример 1.1), заключается в том, чтобы поощрять пользователей вводить дефисы во всех случаях, где они могут встретиться. Если запрос содержит дефис, то эти системы обобщают его так, чтобы охватить все три варианта: с дефисом, слитно и раздельно, так что запрос over-eager преобразуется в форму over-eager OR "over eager" OR overeager. Однако эта стратегия требует обучения пользователей; если пользователь введет одну из двух остальных форм запроса, то обобщение проводиться не будет.

Каждый новый язык порождает новые проблемы. Например, в французском языке апостроф может использоваться для сокращения определенного артикля перед словом, начинающимся с гласного звука (например, *l'ensemble*). Кроме того, в французском языке дефис может использоваться в клитических [6] местоимениях, расположенных после существительного в императивных и вопросительных предложениях (например, *donne-moi* — "дай мне"). Правильная обработка первого случая приводит к корректному индексированию большого количества существительных и прилагательных, т.е. к требованию, чтобы слова *l'ensemble* и *un ensemble* индексировались термином *ensemble*. Другие языки порождают еще более сложные проблемы. Так, в немецком языке *сложные слова* записываются слитно (например, *Computerlinguistik* — "компьютерная лингвистика"; *Lebensversicherungsgesellschaftsangestellter* — "сотрудник общества по страхованию жизни"). Чтобы повысить эффективность работы поисковых систем для немецкого языка, используется модуль, разбивающий сложные слова на несколько составных частей (compound splitter). Наиболее ярко этот эффект проявляется в восточно-азиатских языках (например, в китайском, японском, корейском и тайском): в них вообще нет пробелов. Соответствующий пример приведен на рис. 2.3. Для решения этой проблемы можно было бы на этапе предварительной лингвистической обработки провести *сегментацию на слова* (word segmentation). Методы сегментации на слова весьма разнообразны: от использования большого лексикона (выполняется поиск самой длинной последовательности, представленной в лексиконе, дополнительно используются эвристики для незнакомых слов) до использования методов машинного обучения, таких как скрытые марковские модели или *условные случайные поля* (CRF), обученные на основе слов, выделенных вручную (см. ссылки в разделе 2.5). Из-за неоднозначности сегментации последователь-

[6] Клитика — это слово, которое при произнесении сливается с соседним словом. — *Примеч. ред.*

ностей символов (рис. 2.4) все эти методы иногда порождают ошибки и поэтому не могут гарантировать непротиворечивое и однозначное разбиение на лексемы. Альтернативный метод основан на отказе от индексирования слов и переходе к индексированию коротких подпоследовательностей символов независимо от того, пересекают ли эти подпоследовательности границы слов. В пользу этого подхода можно высказать три довода: символы китайского языка представляют собой скорее слоги, чем буквы, и обычно имеют семантическое содержание; большинство слов являются короткими (чаще всего слово состоит из двух символов) и, с учетом отсутствия стандарта разбиения слов в разных системах письменности, не всегда ясно, где расположены границы слов. Даже в английском языке границы некоторых слов определяются в соответствии с орфографическими соглашениями (например, *notwithstanding*, но *not to mention*, и *into*, но *on to*), однако в процессе обучения люди привыкают писать слова так, как принято.

莎拉波娃现在居住在美国东南部的佛罗里达。今年４月９日，莎拉波娃在美国第一大城市纽约度过了１８岁生日。生日派对上，莎拉波娃露出了甜美的微笑。

Рис. 2.3. Стандартная несегментированная форма текста на китайском языке, в которой использованы упрощенные иероглифы континентального диалекта. Между словами и даже между предложениями нет никаких разделителей — явный символ пробела после символа конца предложения (°) является всего лишь типографской иллюзией, вызванной размещением этого символа слева от его квадратной рамки. Первое предложение — это просто набор китайских слов, не разделенных пробелами. Второе и третье предложения включают арабские цифры и знаки пунктуации

和尚

Рис. 2.4. Неоднозначность выделения китайских слов. Эти два символа могут означать одно слово “монах” или два слова, означающие “и” и “спокойствие”

2.2.2. Игнорирование распространенных термининов: стоп-слова

Иногда некоторые очень распространенные слова, не представляющие ценности для удовлетворения информационных потребностей пользователей, вообще исключаются из лексикона. Они называются *стоп-словами* (stop-words). Как правило, для создания списков стоп-слов термины упорядочиваются по *частоте в коллекции* (которая равна общему количеству повторений термина в коллекции документов), а затем наиболее часто встречающиеся термины, часто отфильтрованные вручную с учетом их семантической связи с предметной областью индексируемых документов, включаются в *список стоп-слов* (stop-list), элементы которого при индексировании отбрасываются. Пример списка стоп-слов приведен на рис. 2.5. Списки стоп-слов существенно сокращают количество позиций, которые должны храниться в системе; некоторые статистические показатели по этой теме будут приведены в главе 5 (см. табл. 5.1). В большинстве случаев игнорирование стоп-слов при индексировании не вызывает проблем: поиск по ключевым словам наподобие the и by вряд ли можно назвать очень полезным. Однако при поиске фраз это не так. Запрос на поиск фразы President of the United States, содержащей два стоп-слова точнее, чем запрос President AND "United States". Смысл фразы flights to London, скорее всего, будет потерян, если выбросить из него стоп-слово to. Поиск статьи Ванневара

Буша (Vannevar Bush) *As we may think* был бы весьма затруднен, если бы первые три слова были проигнорированы, а система просто искала бы документы, содержащие слово think. Одни типы запросов пострадали бы больше других. Некоторые названия песен и хорошо известные отрывки стихотворений целиком состоят из стоп-слов (To be or not to be, Let it be, I don't want to be, ...).

a	an	and	are	as	at	be	by	for	from
has	he	in	is	it	its	of	on	that	the
to	was	were	will	with					

Рис. 2.5. Список из 25 стоп-слов, не имеющих семантического значения, но часто встречающихся в коллекции Reuters–RCV1

Со временем длина списков запретных слов в системах информационного поиска сократилась с 200–300 до 7–12, а в некоторых системах от них вообще отказались. Например, в поисковых веб-системах списки стоп-слов обычно не используются. Разработчики некоторых современных информационно-поисковых систем сосредоточили свое внимание на использовании статистических характеристик языка, чтобы наиболее эффективно обрабатывать самые распространенные слова. В разделе 5.3 будет показано, как снизить стоимость хранения позиций для распространенных слов с помощью эффективных методов сжатия. В разделе 6.2.1 обсуждается стандартный метод взвешивания терминов, позволяющий снизить влияние слишком часто употребляемых слов на ранжирование документов. В заключение, в разделе 7.1.5, мы покажем, как системы информационного поиска с индексами, упорядоченными по важности, могут прервать проход по инвертированному списку, если веса становятся небольшими. Это значит, что часто встречающиеся слова не добавляют существенной нагрузки при обработке среднестатистических запросов, хотя инвертированные списки для стоп-слов очень длинные. Для систем информационного поиска последнего времени дополнительная стоимость включения стоп-слов невелика ни с точки зрения размера индекса, ни с точки зрения времени обработки запросов[7].

2.2.3. Нормализация (классификация терминов по классам эквивалентности)

Допустим, что ваш документ и запрос уже разделены на лексемы. В простейшем случае лексемы в запросе точно совпадают с лексемами в документе. Однако очень часто две последовательности символов не совпадают в точности, но считаются эквивалентными. Например, если вы ищете слово *USA*, то, возможно, хотели бы также видеть документы, содержащие аббревиатуру *U.S.A.*

Нормализация лексем (token normalization) — это процесс приведения лексем к канонической форме, чтобы устранить несущественные различия между последовательностями символов. [8] Самый распространенный способ нормализации заключается в неявном создании *классов эквивалентности* (equivalence classes), которые обычно называются по имени одного из их членов. Например, если в тексте документа и запросах

[7] В частности, одним из практических подходов к обработке стоп-слов, является подход, при котором система старается игнорировать стоп-слова везде, если они не расположены рядом с другими словами запроса. — *Примеч. ред.*

[8] Этот процесс часто называют нормализацией терминов, однако мы предпочитаем зарезервировать слово термин за результатом процесса нормализации.

лексемам *anti-discriminatory* и *antidiscriminatory* ставится в соответствие термин antidiscriminatory, то в ходе поиска одного термина обязательно будут выявлены документы, в которых встречается и другой.

Преимущество использования правил отображения, которые, скажем, удаляют символы, например дефис, проявляется в том, что классы эквивалентности возникают неявно, а не вычисляются заранее: термины, которые в результате применения этих правил становятся идентичными, относятся к одному и тому же классу эквивалентности. Несмотря на то что правила удаления символов из лексем написать относительно легко, обратная задача является довольно сложной. Поскольку классы эквивалентности возникают неявно, совершенно неясно, куда следовало бы вставить недостающие символы. Например, трудно понять, как превратить слово *antidiscriminatory* в слово *anti-discriminatory*.

Альтернативный метод создания классов эквивалентности основан на поддержании связей между ненормализованными лексемами. Он может использовать списки синонимов, составленных вручную, например списки, содержащие слова *car* и *automobile*. Эту тему мы рассмотрим в главе 9. Зависимости между терминами можно создать двумя способами. Обычно сначала выполняется индексирование ненормализованных лексем, а затем для конкретного термина из запроса создается список расширенных запросов, состоящий из нескольких вариантов соответствующего термина, включенных в лексикон. В этом случае термин запроса является результатом дизъюнкции нескольких инвертированных списков. В качестве альтернативы расширение можно выполнять в ходе построения индекса. Если документ содержит слово automobile, мы его индексируем в том числе и термином car (и, как правило, наоборот). Использование каждого из этих методов менее эффективно, чем создание классов эквивалентности, поскольку в этих случаях приходится хранить и объединять большее количество позиций. Первый метод предусматривает создание дополнительного словаря для расширения запросов и увеличивает время их обработки, в то время как второй метод требует больше пространства для хранения позиций. Традиционно дополнительные требования к объему памяти, необходимой для хранения инвертированных списков, считались серьезным недостатком, но с учетом снижения стоимости устройств хранения данных повышение гибкости поиска, которая появляется благодаря отдельным инвертированным спискам, заслуживает внимания.

Оба указанных подхода являются более гибкими по сравнению с построением классов эквивалентности, поскольку списки расширений могут перекрываться, но при этом не совпадать полностью. Это значит, что можно реализовать асимметричные расширения. Пример использования этой асимметрии приведен на рис. 2.6: если пользователь введет слово windows, то мы допускаем, что он имеет в виду операционную систему *Windows* (с большой буквы). Однако, если пользователь ввел слово window, такое предположение становится неправдоподобным, даже несмотря на то что для этого запроса вполне допустим вариант *windows* (с маленькой буквы).

Термин запроса	**Соответствующие термины в документе**
Windows	Windows
windows	Windows, windows, window
window	window, windows

Рис. 2.6. Пример, в котором асимметричное расширение терминов, входящих в запрос, может удачно моделировать ожидания пользователей

Непонятно, до какой степени следует использовать классы эквивалентности и расширение запросов. С одной стороны, их применение кажется хорошей идеей. С другой стороны, слишком широкое использование может быстро привести к непредвиденным последствиям в виде чрезмерного расширения запросов. Например, построение класса эквивалентности для слов *U.S.A.* и *USA* путем удаления точек из лексем, на первый взгляд, выглядит целесообразно, поскольку вторая форма аббревиатуры встречается чаще первой. Однако, введя аббревиатуру *C.A.T.*, вы, скорее всего, огорчитесь, если в ответ получите все документы, содержащие слово *cat*. [9]

Ниже приведено несколько широко распространенных видов нормализации и показаны способы их реализации. Во многих ситуациях они полезны, но иногда могут причинить вред. Можно продумать массу деталей при построении классов эквивалентности, но в итоге в ходе обработки запроса тщательно продуманные детали часто не оказывают значительного воздействия на эффективность поиска.

Ударения и диакритические символы. Диакритические знаки (diacritics) в английском языке встречаются очень редко, поэтому естественно считать, что слова *cliché* и *cliche* или *naive* и *naïve* совпадают. Для этого в процессе нормализации следует удалить диакритические символы. Во многих других языках диакритические символы являются важной частью письменности и позволяют отличать разные звуки. Иногда слова отличаются лишь ударением. Например, в испанском языке слово *peca* означает *утёс*, а слово *pena* означает *горе*. Тем не менее на первый план выходят не лингвистические или грамматические проблемы, а вопрос, в каком виде пользователи, скорее всего, будут писать эти слова. Часто пользователи вводят свои запросы, игнорируя диакритические знаки либо для повышения скорости ввода, либо из-за лени, либо из-за ограничений программного обеспечения, либо просто по привычке, возникшей во времена, когда в большинстве компьютерных систем доступными были лишь ASCII-символы. В этих случаях наилучшим решением будет все слова сделать эквивалентными форме без диакритических знаков.

Использование заглавных букв/обработка без учета регистра. Как правило, в поисковых системах игнорируется регистр букв (case-folding), например все буквы переводятся в нижний регистр. Часто это оказывается удачным решением, позволяющим все слова *Automobile*, с которых начинаются соответствующие предложения, считать соответствующим запросу *automobile*. Это также помогает поисковым веб-системам в ситуациях, когда большинство пользователей набирают слово *ferrari*, интересуясь автомобилем *Ferrari*. С другой стороны, игнорирование регистра может привести к совпадению слов, которые следовало бы считать разными. Многие имена собственные являются производными от имен нарицательных и отличаются от них лишь прописной первой буквой. Например, это относится к названиям компаний (*General Motors*, *The Associated Press*) и правительственных организаций (*the Fed* и *fed*), а также к фамилиям (*Bush*, *Black*). Мы уже приводили пример непреднамеренного расширения запроса, содержащего акронимы, в которых используется не только нормализация акронима (*C.A.T.* → *CAT*), но и игнорирование регистра (*CAT* → *cat*).

В английском языке в качестве альтернативы переводу всех лексем в нижний регистр можно использовать выборочный перевод. Простейшее эвристическое правило предполагает перевод в нижний регистр только слов, начинающих предложения, и всех слов,

[9] Во время написания этой главы (август 2005 года) поисковая машина Google так и делала: в первой строке ответа на запрос C.A.T. она вернула адрес веб-сайта, посвященного кошкам (Cat Franciers Web Site, `www.franciers.com/`).

встречающихся в заглавиях, в которых все буквы являются прописными или в которых большинство слов начинаются с прописных букв. Как правило, это обычные слова, которые должны быть набраны прописными буквами. Слова, встречающиеся в середине предложений и набранные с помощью прописных букв, остаются неизменными (что обычно является правильным решением), и это позволяет избежать нежелательных совпадений слов, различающихся лишь регистром. Данную задачу можно решить точнее, построив с помощью машинного обучения соответствующую последовательную модель, учитывающую больше признаков для принятия решения об изменении регистра. Эта стратегия называется *учетом истинного регистра* (truecasing). Однако попытка учесть истинный регистр с помощью этой стратегии может оказаться безрезультатной, если пользователи регулярно применяют в запросах нижний регистр вместо орфографически корректного.

Другие проблемы, связанные с английским языком. При других подходах к нормализации учитываются особенности английского языка. Например, можно потребовать, чтобы слова *ne'er* и *never* всегда считались одинаковыми или не различать британское написание слова *colour* и американское *color*. Даты, время и другие аналогичные элементы могут встречаться в разных форматах, что создает новые проблемы. Поэтому можно потребовать, чтобы записи *3/12/91* и *Mar. 12, 1991* считались одинаковыми. Однако правильная обработка таких запросов усложняется тем, что в США запись *3/12/91* означает *Mar. 12, 1991*, а в Европе — *3 Dec. 1991*.

Другие языки. Английский язык является доминирующим языком в вебе; примерно 60% веб-страниц написаны на английском языке (Gerrand, 2007). Тем не менее остальные 40% веб-страниц созданы на других языках, причем их доля со временем будет возрастать, поскольку меньше трети пользователей Интернета и меньше 10% людей во всем мире считают английский своим основным языком. Признаки изменений наблюдаются уже: по некоторым данным (Sifry, 2007) только треть записей в блогах написаны на английском языке.

Другие языки, в свою очередь, создают разнообразные проблемы в выделении классов эквивалентных слов. Например, французский определенный артикль имеет разные формы, зависящие не только от рода (мужского или женского) и от количества последующих имен существительных, но и от того, с какого звука начинается следующее слово (*la*, *le*, *l'* или *les*). Естественно создать класс эквивалентности для разных форм определенного артикля. В немецком языке существует правило, согласно которому гласные буквы с умлаутом могут быть представлены двумя буквами. По этой причине слова *Schütze* и *Schuetze* считаются эквивалентными.

Всеми признано, что японская система письменности является очень сложной (рис. 2.7). Современный японский алфавит представляет собой смесь разных алфавитов (в основном китайских символов), двух слоговых азбук (хирагана и катакана), а также символов из западных языков (латинские буквы, арабские цифры и другие символы). Несмотря на строгие правила и стандартизацию японской письменности, во многих случаях одно и то же слово может быть записано по-разному. Например, слово может быть записано с помощью слоговой азбуки катакана для выразительности (аналогично выделению курсивом) или слоговой азбуки хирагана, или китайскими символами. Таким образом, для успешного поиска информации необходимо выполнить сложную процедуру создания классов эквивалентности слов из разных систем японской письменности. В частности, пользователь может применять исключительно слоговую азбуку хирагана, поскольку ее символы легче набирать на клавиатуре, точно так же как западные пользователи в основном применяют нижний регистр.

ノーベル平和賞を受賞したワンガリ・マータイさんが名誉会長を務めるＭＯＴＴＡＩＮＡＩキャンペーンの一環として、毎日新聞社とマガジンハウスは「私 の、もったいない」を募集します。皆様が日ごろ「もったいない」と感じて実践していることや、それにまつわるエピソードを８００字以内の文章にまとめ、簡 単な写真、イラスト、図などを添えて１０月２０日までにお送りください。大賞受賞者には、５０万円相当の旅行券とエコ製品２点の副賞が贈られます。

Рис. 2.7. Японская система письменности является смесью разных систем и, как и китайская, не предусматривает выделения слов. В этом тексте использованы в основном китайские символы в сочетании со слоговой азбукой хирагана для флективных окончаний и служебных слов. Латинские буквы на самом деле являются частью японских выражений, описывающих кампанию за охрану окружающей среды, которую ведет Вангари Маатаи (Wangari Maathai), нобелевский лауреат премии мира 2004 года. Его имя в середине первой строки записано с помощью слоговой азбуки катакана. Первые четыре символа в последней строке представляют запись суммы, эквивалентной 500 тысячам иен

Индексируемые коллекции часто содержат документы на разных языках. Впрочем, отдельный документ также может быть многоязычным. Например, электронное сообщение на французском языке может содержать цитату из контракта на английском. Как правило, сначала осуществляется распознавание языка, а затем — разбиение текста на лексемы и нормализация по правилам для данного языка, учитывающим заранее установленную степень детализации, например, по всему документу или по абзацам. Однако такой подход не срабатывает, если документ содержит краткие цитаты на других языках. Если коллекции документов являются многоязычными, то индекс может содержать термины из разных языков. В частности, можно сначала применить к документу классификатор языков, а затем пометить термины в лексиконе соответствующего языка. Иногда эта разметка игнорируется, поскольку последовательности символов на разных языках редко совпадают.

При обработке иностранных или сложных слов, в особенности иностранных имен, орфография может быть неясной или могут существовать различные правила транслитерации, приводящие к различным вариантам (например, *Chebyshev* и *Tchebycheff* или *Beijing* и *Peking*). Для решения этой проблемы можно использовать эвристические правила создания классов эквивалентности или расширять термины с помощью фонетических эквивалентов. Чаще всего для этого применяются такие алгоритмы, как Soundex, который мы рассмотрим в разделе 3.4.[10]

2.2.4. Стемминг и лемматизация

По грамматическим причинам в документах встречаются разные формы одних и тех же слов, например *organize*, *organizes* и *organizing*. Кроме того, существуют семейства производных слов, имеющих близкий смысл, например *democracy*, *democratic* и

[10] На практике и правила транслитерации, и словарь транслитерационных пар необходимо строить с использованием статистики встречаемости в коллекциях запросов и текстов, так как эвристические правила не дают удовлетворительного качества. — *Примеч. ред.*

democratization. Во многих ситуациях кажется полезным по одному из них находить документы, содержащие другие слова из этого семейства.

Цель стемминга и лемматизации — привести словоформы и производные формы слова к общей основной форме. Рассмотрим примеры.

am, are, is ⇒ be
car, cars, car's, cars' ⇒ car

В результате может возникнуть следующее преобразование текста.

the boy's cars are different colors ⇒
the boy car be different color

Однако стемминг отличается от лемматизации. *Стеммингом* (stemming) обычно называется приближенный эвристический процесс, в ходе которого от слов отбрасываются окончания в расчете на то, что в большинстве случаев это себя оправдает. Стемминг часто подразумевает удаление производных аффиксов. *Лемматизация* (lemmatization) — это точный процесс с использованием лексикона и морфологического анализа слов, в результате которого удаляются только флективные окончания и возвращается основная, или словарная, форма слова, называемая *леммой*. Например, лексема *saw* в ходе стемминга может превратиться в букву *s*, в то время как лемматизация вернет либо слово *see*, либо слово *saw* в зависимости от того, является ли лексема глаголом или именем существительным. Важное различие состоит в том, что обычно стемминг "склеивает" производные однокоренные слова, а лемматизация "склеивает" только флективные формы одной леммы. Стемминг и лемматизация часто выполняются с помощью дополнительных программных компонентов, встраиваемых в процесс индексирования. В настоящее время существует множество таких программ, как коммерческих, так и свободно распространяемых.

Наиболее распространенным алгоритмом стемминга английских слов, который неоднократно демонстрировал свою эффективность в практических приложениях, является *алгоритм Портера* (Porter, 1980). Весь алгоритм слишком длинный и сложный для того, чтобы привести его здесь полностью, но мы опишем его суть. Алгоритм Портера состоит из пяти этапов сокращения слов, выполняемых последовательно. На каждом этапе применяются разные соглашения по выбору правил, таких как выбор правила из каждой группы правил, применимых к самому длинному суффиксу. На первом этапе это соглашение применяется к следующей группе правил.

Правило		**Пример**	
SSES	→ SS	caresses	→ caress
IES	→ I	ponies	→ poni
SS	→ SS	caress	→ caress
S	→	cats	→ cat

Во многих современных правилах используется концепция *меры слова* (measure of a word), которая примерно оценивает количество слогов, чтобы определить, достаточно ли длинным является слово для применения правила, относящегося к суффиксу, а не к основе. Рассмотрим пример.

(m > 1) EMENT →

Это правило отображает слово *replacement* в слово *replac*, но не слово *cement* в слово *c*. Официальный сайт, посвященный алгоритму Портера (Porter Stemmer), расположен по следующему адресу.

`www.tartarus.org/~martin/PorterStemmer/`

Существуют и другие алгоритмы стемминга: относительно старый однопроходный алгоритм Ловинса (Lovins, 1968) и относительно новый алгоритм Пейса–Хаска (Paice/Husk) (Paice, 1990). Они представлены на следующих веб-страницах.

`www.cs.waikato.ac/nz/~eibe/stemmers/`
`www.comp.lancs.ac.uk/computing/research/stemming/`

Пример. Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation.

Алгоритм Ловинса. Such an analysis can reve featur that ar not eas vis from th vari in th individu gen and can lead to a pictur of expres that is mor biolog transpar and acces to interpres.

Алгоритм Портера. Such an analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to a pictur of express that is more biolog transpar and access to interpret.

Алгоритм Пейса. Such an analysis can rev feat that are not easy vis from the vary in the individ gen and can lead to a pict of express that is mor biolog transp and access to interpret.

Рис. 2.8. Сравнение трех алгоритмов морфологического поиска на конкретном примере

На рис. 2.8 продемонстрированы результаты неформального сравнения трех разных алгоритмов стемминга. Они используют правила, характерные для английского языка, но при этом требуют меньше знаний, чем алгоритмы лемматизации, для работы которых необходимы полный лексикон и морфологический анализ. Отдельные области знаний также могут потребовать специальных правил стемминга. Однако нас интересуют не сами формы, полученные в результате стемминга, а лишь классы эквивалентности, которые они образуют.

Вместо алгоритма стемминга можно использовать *лемматизатор* (lemmatizer), инструмент из области обработки естественного языка (natural language processing), выполняющий полный морфологический анализ для точного определения леммы каждого слова. Полный морфологический анализ приносит весьма скромный выигрыш при информационном поиске. Трудно сказать что-то более конкретное, так как ни одна из форм нормализации не повышает суммарную эффективность поиска информации на английском языке, по крайней мере не способна увеличить ее значительно. Несмотря на то что для некоторых запросов лемматизация может оказаться очень полезной, для остальных запросов она существенно снижает производительность. Стемминг повышает полноту, но снижает точность поиска. Для того чтобы продемонстрировать недостатки этих алгоритмов, отметим, что алгоритм Портера обрезает все слова

operate operating operates operation operative operatives operational

до слова *oper*. Однако слово *operate* во всех своих формах является распространенным глаголом, точность для следующих запросов сильно снизится при использовании алгоритма Портера.

operational AND research
operating AND system
operative AND dentistry

В таких ситуациях лемматизация не помогает полностью решить проблему, так как конкретные словоформы используются в словосочетаниях: предложение со словами *operate* и *system* не полностью соответствуют запросу operating AND system. Эффект от нормализации терминов зависит в большей степени от прагматических аспектов словоупотребления, чем от формальных морфологических аспектов.

Иначе обстоят дела для языков с более сложной морфологической структурой (например, для испанского, немецкого и финского). Результаты европейского форума CLEF (Cross Language Evaluation Forum) многократно демонстрировали, что использование алгоритмов стемминга (и разбиения составных слов для немецкого языка) дает существенный выигрыш. Соответствующие ссылки приведены в разделе 2.5.

? ***Упражнение 2.1*** [*]. Определите, являются ли следующие утверждения истинными или ложными.

1. В системах логического информационного поиска стемминг никогда не снижает точность.
2. В системах логического информационного поиска стемминг никогда не снижает полноту.
3. Стемминг увеличивает размер лексикона.
4. Стемминг следует выполнять в ходе индексирования, а не при обработке запроса.

Упражнение 2.2 [*]. Какую нормализованную форму следует использовать для следующих слов (включая как вариант само слово)?

1. 'Cos
2. Shi'ite
3. cont'd
4. Hawai'i
5. O'Rourke

Упражнение 2.3 [*]. Следующие пары слов алгоритм Портера приводит к одной и той же форме. Какие пары, по вашему мнению, не следует объединять?

1. abandon/abandonment
2. absorbency/absorbent
3. marketing/markets
4. university/universe
5. volume/volumes

Упражнение 2.4 [*]. Проанализируйте группу правил алгоритма Портера, описанную выше.

1. Для чего в нее включено правило тождества, например SS → SS?

2. Примените эту группу правил к следующим словам.

   circus canaries boss

3. Какое правило следовало бы включать в эту группу, чтобы правильно обработать слово *pony*?

4. Результаты стемминга для слов *ponies* и *pony* могут показаться странными. Может ли это оказать вредное влияние на обработку запроса? Почему "да" или почему "нет"?

2.3. Быстрое пересечение инвертированных списков с помощью указателей пропусков

В оставшейся части главы мы обсудим расширение структур данных для инвертированных списков и способы повышения эффективности использования этих списков. Напомним основную операцию пересечения инвертированных списков (см. раздел 1.3): мы одновременно перемещаемся по двум инвертированным спискам за время, пропорциональное общему количеству элементов в этих списках. Если длины списков равны m и n, то пересечение выполняется за $O(m+n)$ операций. Можно ли улучшить эту оценку? Иначе говоря, нельзя ли найти эмпирический способ выполнить пересечение списков за сублинейное время? Можно, если индекс изменяется не слишком быстро.

Один из методов ускорения операции пересечения списков основан на использовании *списка с пропусками* (skip list). Для этого надо снабдить инвертированные списки указателями пропусков (во время индексирования), как показано на рис. 2.9. Указатели пропусков представляют собой эффективный инструмент, позволяющий избежать обработки элементов инвертированного списка, которые не будут включены в результаты поиска. Возникают два вопроса: "Где разместить указатели пропусков?" и "Как осуществить эффективное слияние списков с их помощью?"

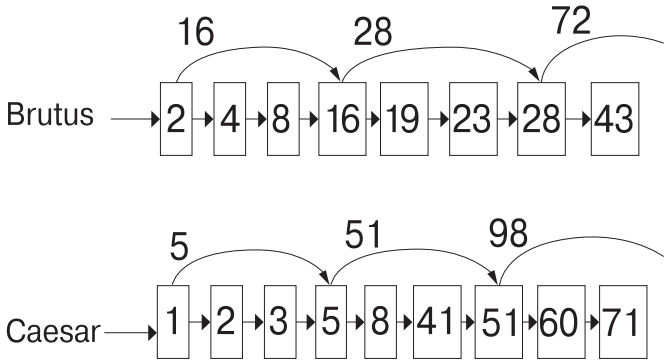


Рис. 2.9. Инвертированные списки с указателями пропусков. Операция пересечения списков может использовать эти указатели, если последний элемент списка меньше, чем элемент в другом списке

Рассмотрим сначала эффективное слияние, взяв в качестве примера рис. 2.9. Допустим, что мы проходили по спискам, изображенным на рисунке, пока не нашли число **8**

в каждом из списков и не включили его в список результатов. Тогда мы переместим оба указателя вперед, так что в первом списке указатель будет установлен на число **16**, а во втором — на число **41**. Наименьшим из этих двух элементов является число **16** в первом списке. Вместо обычного перемещения указателя вперед по первому списку мы сначала проверим указатель пропусков и выясним, что число **28** также меньше, чем **41**. Значит, мы можем проследовать за указателем пропусков и переместить указатель в первом списке вперед, установив его на число **28**. Таким образом, мы избежали прохода по числам **19** и **23** в первом списке. Возможны несколько вариантов пересечения инвертированных списков в зависимости от того, когда именно проверяется указатель пропусков. Один из этих вариантов показан на рис. 2.10. Указатели пропусков есть только у исходных инвертированных списков. Для промежуточных результатов обработки сложных запросов вызов функции *hasSkip*(*p*) всегда возвращает значение *false*. В заключение отметим, что наличие указателей пропусков помогает обрабатывать запросы, содержащие операцию AND, но не запросы, содержащие операцию OR.

```
IntersectWithSkips(p1, p2)
1  answer ← ⟨⟩
2  while p1 ≠ NIL and p2 ≠ NIL
3  do if docID(p1) = docID(p2)
4     then Add(answer, docID(p1))
5          p1 ← next(p1)
6          p2 ← next(p2)
7     else if docID(p1) < docID(p2)
8             then if hasSkip(p1) and (docID(skip(p1)) ≤ docID(p2)
9                  then while hasSkip(p1) and (docID(skip(p1)) ≤ docID(p2)
10                      do p1 ← skip(p1)
11                 else p1 ← next(p1)
12            else if hasSkip(p2) and (docID(skip(p2)) ≤ docID(p1)
13                 then while hasSkip(p2) and (docID(skip(p2)) ≤ docID(p1)
14                      do p2 ← skip(p2)
15                 else p2 ← next(p2)
16 return answer
```

Рис. 2.10. Операция пересечения инвертированных списков с указателями пропусков

Где лучше делать пропуски? Ответ на этот вопрос является результатом компромисса. Чем больше пропусков, тем меньше их шаг и тем выше вероятность пропусков. Однако одновременно это означает, что количество сравнений с указателями пропусков и объем памяти для их хранения также возрастают. Чем меньше пропусков, тем меньше сравнений с указателями пропусков, но больше их шаг и меньше вероятность перехода. Простые эвристические правила размещения указателей пропусков, хорошо зарекомендовавшие себя на практике, утверждают, что если длина инвертированного списка равна *P*, то следует использовать $\sqrt{P}$ равномерно размещенных указателей пропусков. Тем не менее это эвристическое правило можно усовершенствовать, поскольку оно не учитывает распределение терминов запросов.

Создать эффективные указатели пропусков легко, если индекс изменяется редко; задача намного усложняется, если индекс постоянно обновляется. Злонамеренные стратегии удаления могут вообще сделать списки указателей пропусков совершенно неэффективными.

Выбор оптимального способа кодирования инвертированного индекса представляет собой постоянно изменяющуюся задачу для разработчика системы, поскольку этот выбор сильно зависит от используемых компьютерных технологий, их относительной скорости работы и размера системы. В прошлом процессоры работали медленно, поэтому стратегии сильного сжатия данных были неоптимальными. Однако в настоящее время процессоры работают быстро, а дисковые устройства — относительно медленно, поэтому на первый план выходит задача сокращения длины инвертированного списка, хранящегося на диске. Однако если запустить поисковую систему, при условии, что все данные загружены в оперативную память, то ситуация вновь изменится. Влияние параметров аппаратного обеспечения на время построения индекса обсуждается в разделе 4.1, а влияние размера индекса на скорость работы системы — в главе 5.

? ***Упражнение 2.5*** [*]. Почему указатели пропусков не ускоряют обработку запроса вида x OR y?

Упражнение 2.6 [*]. Допустим, что вы ввели запрос, состоящий из двух слов. Для одного из терминов инвертированный список состоит из 16 элементов.

[4, 6, 10, 12, 14, 16, 18, 20, 22, 32, 47, 81, 120, 122, 157, 180]

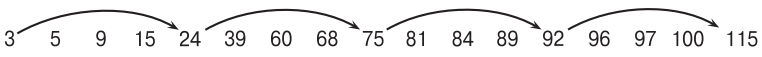
Инвертированный список для второго термина состоит из одного элемента.

[47]

Вычислите, сколько сравнений пришлось бы выполнить при пересечении этих двух списков при использовании следующих двух стратегий. Кратко обоснуйте свои ответы.

1. Обработка выполняется с помощью стандартных инвертированных списков.
2. Инвертированные списки хранят указатели пропусков с рекомендованным шагом пропусков $\sqrt{P}$.

Упражнение 2.7 [*]. Рассмотрите пересечение указанных выше инвертированных списков с использованием указателей пропусков



и списка записи промежуточных результатов (в котором нет указателей пропусков)

3 5 89 95 99 100 101

Примените алгоритм пересечения инвертированных списков, описанный на рис. 2.10.

1. Как часто происходит переход по указателю пропусков (т.е. указатель p_1 устанавливается в позицию $skip(p_1)$?

2. Сколько сравнений инвертированных списков выполняется в ходе этого алгоритма при пересечении этих двух списков?
3. Сколько сравнений инвертированных списков было бы выполнено, если бы их пересечение вычислялось без использования указателей перехода?

2.4. Словопозиции с координатами и фразовые запросы

Многие сложные понятия, названия организаций и торговые марки представляют собой многословные словосочетания или фразы. Мы хотели бы иметь возможность так представлять запросы наподобие Stanford University, рассматривая их как единую фразу, чтобы документ *The inventor Stanford Ovshinsky never went to university* не попадал в список результатов. Современные поисковые системы поддерживают синтаксис двойных кавычек ("stanford university") для *фразовых запросов* (phrase queries). Этот синтаксис прост и широко используется многими пользователями. Не менее 10% всех вебзапросов представляют собой фразовые запросы. Однако неявных фразовых запросов, которые вводятся без двойных кавычек, намного больше (например, при поиске людей по именам и фамилиям). Для обработки таких запросов инвертированные списки уже не могут быть простыми списками документов, содержащих отдельные термины. В этом разделе рассматриваются два подхода к обработке фразовых запросов и их комбинация. Поисковые машины должны не только поддерживать возможность задания фразовых запросов, но и обеспечивать их эффективную обработку. Связанная с этим, но все же отдельная концепция предусматривает взвешивание на основе близости терминов запроса в документе: чем ближе термины запроса в документе, тем выше вес документа. Этот метод описан в разделе 7.2.2 в процессе анализа поиска с ранжированием.

2.4.1. Двухсловные индексы

Один из подходов к обработке фразовых запросов предусматривает интерпретацию каждой пары последовательных терминов в документе как фразы. Например, текст *Friends, Romans, Countrymen* порождает следующие двухсловия.

```
friends romans
romans contrymen
```

В этой модели мы рассматриваем каждое двухсловие как термин словаря. Это сразу дает нам возможность обрабатывать двухсловные фразовые запросы. Более длинные фразы обрабатываются путем разбиения на двухсловные части. Запрос stanford university palo alto можно разбить на логический запрос, состоящий из двухсловий.

```
"stanford university" AND "university palo" AND "palo alto"
```

Этот запрос на практике может себя оправдать, но иногда он может и должен возвращать ложно положительные ответы. Без проверки этих документов невозможно убедиться, что документы, удовлетворяющие этому булеву запросу, действительно содержат первоначальную фразу из четырех слов.

Среди всех возможных запросов имена существительные и именные словосочетания занимают особое положение при описании понятий, которые хотят найти люди. Однако связанные между собой имена существительные часто оказываются отделенными друг от друга разными служебными словами, как, например, в фразах *the abolition of slavery* и *renegotiation of the constitution*. Эту особенность можно учесть в рамках модели двух-

словной индексации следующим образом. Сначала необходимо разбить текст на лексемы и выполнить разметку по частям речи (part-of-speech tagging). [11] Затем термины группируются по частям речи, в частности по именам существительным, включая собственные имена (N), и служебным словам, включая артикли и предлоги (X). Теперь любую строку терминов можно представить в виде NX*N, т.е. как расширенную фразу из двух слов. Каждое такое расширенное двухсловие образует термин в лексиконе. Рассмотрим пример.

renegotiation	of	the	constitution
N	X	X	N

Для поиска по такому расширенному двухсловному индексу запрос так же необходимо проанализировать на наличие существительных (*N*) и служебных слов (*X*), а затем сегментировать запрос на расширенные двухсловия, по которым можно производить поиск в индексе.

При преобразовании длинных запросов в логические запросы этот алгоритм не всегда работает оптимально. Например, с помощью описанного выше алгоритма запрос

cost overruns on a power plant

преобразуется в запрос

"cost overruns" AND "overruns power" AND "power plant"

В то же время было бы лучше пропустить среднее двухсловие. Лучшие результаты можно было бы получить, используя более точные шаблоны частей речи, определяющие двухсловные словосочетания, подлежащие индексированию.

Концепцию двухсловного индекса можно расширить на более длинные цепочки слов. Если индекс содержит цепочки слов переменной длины, говорят об *индексе фраз* (phrase index). Действительно, было бы неестественно обрабатывать однословные запросы с помощью двухсловных индексов (для этого пришлось бы просмотреть все пары слов словаря, содержащие этот термин), поэтому нам нужен индекс однословных терминов. Несмотря на то что всегда есть шанс ложного срабатывания, вероятность получить такой ответ при выполнении поиска по фразам, содержащим три и более слов, становится весьма малой. Однако, с другой стороны, хранение длинных фраз приведет к значительному увеличению размера словаря. Необходимость поддержки полного индекса фраз, которые длиннее двух слов, — пугающая перспектива, и даже использование полного двухсловного индекса очень сильно увеличивает размер лексикона. И все же в конце раздела мы обсудим целесообразность этой стратегии в рамках схемы составного индексирования.

2.4.2. Координатные индексы

По указанным причинам двухсловный индекс не стал стандартным решением. Вместо него широкое распространение получил *координатный индекс* (positional index). В нем для каждого термина из лексикона хранятся словопозиции в формате docID: ⟨position1, position2, ...⟩ (рис. 2.11), где position1, position2 и т.д. представляют собой

[11] В результате разметки по частям речи слова разбиваются по классам как имена существительные, глаголы и т.д. или используются более мелкие классы, например “имя существительное во множественном числе”. В настоящее время существуют очень точные инструменты разметки текста по частям речи (примерно 96% корректных меток), которые, как правило, используют методы машинного обучения на основе текстов, размеченных вручную (см. Manning and Schütze, 1999, ch. 10).

координаты лексемы в документе. В словопозициях обычно содержится частота термина в документе (причины этого решения обсуждаются в главе 6).

to, 993427:
⟨1, 6: ⟨7, 18, 33, 72, 86, 231⟩;
2, 5: ⟨1, 17, 74, 222, 255⟩;
4, 5: ⟨8, 16, 190, 429, 433⟩;
5, 2: ⟨363, 367⟩;
7, 3: ⟨13, 23, 191, ...⟩; ... ⟩
be, 178239:
⟨1, 2: ⟨17, 25⟩;
4, 5: ⟨17, 191, 291, 430, 434⟩;
5, 3: ⟨14, 19, 101⟩; ... ⟩

Рис. 2.11. Пример координатного индекса. Слово to *встречается в документах 993 477 раз, причем в документе 1 оно встречается на позициях 7, 18, 33 и т.д.*

Для обработки фразовых запросов нам по-прежнему необходимо иметь доступ к элементам инвертированного индекса для каждого отдельного термина. Как и прежде, мы начинаем с наиболее редко встречающегося термина, а затем все больше ограничиваем список кандидатов. При выполнении операции слияния используется прежний общий подход, но вместо простой проверки, что оба термина содержатся в документе, теперь необходимо убедиться, что их координаты в документе соответствуют фразовому запросу. Для этого необходимо подсчитать расстояние между словами.

Пример 2.1. **Обработка фразовых запросов.** Допустим, что инвертированные списки для слов *to* и *be* имеют такой вид, как на рис. 2.11, а пользователь ввел запрос to be or not to be. Нам необходим доступ к инвертированным спискам для слов to, be, or и not. Проверяем пересечение инвертированных списков для слов to и be. Сначала смотрим на документы, в которых содержатся оба слова. Затем находим места в списках, где координата лексемы для слова *be* на единицу больше координаты лексемы *to*. После этого находим следующее вхождение каждого слова, координата лексем которых на четыре больше, чем первое вхождение. В указанных выше списках этим условиям удовлетворяют следующие записи.

to: ⟨...; 4: ⟨..., 429, 433⟩; ...⟩

be: ⟨...; 4: ⟨..., 430, 434⟩; ...⟩

Точно такой же метод можно применить для поиска среди *k* ближайших слов, описанного в примере 1.1.

employment /3 place

Здесь /k означает “среди *k* слов (с обеих сторон)”. Очевидно, что координатный индекс вполне подходит для обработки таких запросов, а двухсловный индекс — не подходит. На рис. 2.12 продемонстрирован алгоритм поиска среди *k* ближайших слов; мы еще рассмотрим его в упражнении 2.12.

```
Positional Intersect(p1, p2, k)
1   answer ← ⟨⟩
2   while p1 ≠ NIL and p2 ≠ NIL
3   do if docID(p1) = docID(p2)
4      then l ← ⟨⟩
5        pp1 ← positions(p1)
6        pp2 ← positions(p2)
7        while pp1 ≠ NIL
8        do while pp2 ≠ NIL
9           do if |pos(pp1) – pos(pp2)| ≤ k
10             then Add(l, pos(pp2))
11             else if pos(pp2) > pos(pp1)
12                  then break
13                  pp2 ← next(pp2)
14             while l ≠ ⟨⟩ and |l[0] – pos(pp1)| > k
15          do Delete(l[0])
16          for each ps ∈ l
17          do Add(answer, ⟨docID(p1), pos(pp1), ps⟩
18           pp1 ← next(pp1)
19       p1 ← next(p1)
20       p2 ← next(p2)
21  else if docID(p1) < docID(p2)
22             then p1 ← next(p1)
23             else p2 ← next(p2)
24  return answer
```

Рис. 2.12. Алгоритм пересечения инвертированных списков p_1 и p_2 с учетом близости терминов. Алгоритм находит места, в которых два термина встречаются на расстоянии не более k слов, и возвращает список троек, состоящих из идентификатора документа docID, а также координат терминов в списках p_1 и p_2

Размер координатного индекса. Применение координатного индекса существенно увеличивает объем хранимых данных, даже если использовать компактное представление координат, как описано в разделе 5.3. Действительно, переход к координатному индексу также изменяет асимптотическую сложность операции пересечения инвертированных списков, поскольку количество сравниваемых элементов теперь ограничено не количеством документов, а количеством словоупотреблений в коллекции документов T. Иначе говоря, сложность логического запроса равна $\Theta(T)$, а не $\Theta(N)$. Однако в большинстве приложений это уже неизбежно, потому что большинство пользователей рассчитывают, что система обладает функциональностью поиска фразовых запросов и запросов с учетом близости терминов.

Рассмотрим требования к объему хранимых данных, возникающие при использовании координатного индекса. Теперь в словопозициях необходимо место для хранения

координаты термина. Следовательно, размер индекса зависит от среднего размера документа. Типичная веб-страница содержит меньше тысячи терминов, но такие документы, как биржевые листинги комиссии SEC, книги, а также некоторые эпические поэмы, содержит более ста тысяч терминов. Рассмотрим термин со средней частотой появления, равной единице среди тысячи терминов. В результате для обработки больших документов требования к объему памяти, предназначенной для хранения инвертированного списка, возрастают на два порядка.

Размер документа	Число словопозиций	Число координат термина на одну словопозицию
1 000	1	1
100 000	1	100

Несмотря на то что точные числа зависят от типа индексируемых документов и языка, существуют эмпирические правила, согласно которым размер координатного индекса в два-четыре раза больше, чем размер некоординатного (*документного*) индекса, а сжатый координатный индекс составляет от трети до половины размера исходного текста (после удаления разметки и т.д.) в несжатых документах. Конкретные числа для коллекций, которые мы рассматриваем в качестве примера, приведены в табл. 5.1 и 5.6.

2.4.3. Комбинированные схемы

Стратегии, основанные на двухсловных и координатных индексах, можно плодотворно комбинировать. Если пользователи часто посылают запросы на поиск конкретной фразы, например Michael Jackson, то продолжать пересекать соответствующие инвертированные списки нецелесообразно. Комбинированная стратегия для обработки определенных запросов использует индекс фраз или двухсловный индекс, а для всех остальных фразовых запросов — только координатный индекс. В индекс фраз включаются часто встречающиеся запросы пользователей. Однако это не единственный критерий: наиболее дорогостоящими являются те фразовые запросы, в которых отдельные слова являются распространенными, но вся фраза встречается относительно редко. Например, добавление словосочетания *Britney Spears* в индекс фраз может лишь примерно в три раза ускорить обработку этого запроса, поскольку большинство документов, содержащих хотя бы одно из этих слов, будут правильными ответами, в то время как добавление *The Who* может ускорить обработку этого запроса в тысячу раз. Следовательно, второй вариант является более предпочтительным, хотя такой запрос встречается реже.

В работе Уильямса и др. (Williams et al., 2004) приводится оценка еще более сложной схемы, в которой используются индексы двух описанных типов, а также частичный индекс следующих слов, который представляет собой промежуточное решение по отношению к двум первым. Для каждого термина *индекс "следующее слово"* (next word index) хранит записи о терминах, которые следуют за указанным термином в документе. Такая стратегия позволяет обработать типичный набор фразовых веб-запросов примерно в четыре раза быстрее, чем при использовании координатного индекса, в то же время требования к размеру хранимых данных увеличиваются лишь на 26%.

? ***Упражнение 2.8*** [*]. Рассмотрим двухсловный индекс. Приведите пример документа, который будет возвращен при обработке запроса New York University, хотя на самом деле его возвращать не следовало.

Упражнение 2.9 [*]. Ниже приведена часть координатного индекса в формате term:doc1:⟨position1, position2, ...⟩; doc2:⟨position1, position2, ...⟩ и т.д.

angels: 2: ⟨36,174,252,651⟩; 4: ⟨12,22,102,432⟩; 7: ⟨17⟩;

fools: 2: ⟨1,17,74,222⟩; 4: ⟨8,78,108,458⟩; 7: ⟨3,13,23,193⟩;

fear: 2: ⟨87,704,722,901⟩; 4: ⟨13,43,113,433⟩; 7: ⟨18,328,528⟩;

in: 2: ⟨3,37,76,444,851⟩; 4: ⟨10,20,110,470,500⟩; 7: ⟨5,15,25,195⟩;

rush: 2: ⟨2,66,194,321,702⟩; 4: ⟨9,69,149,429,569⟩; 7: ⟨4,14,404⟩;

to: 2: ⟨47,86,234,999⟩; 4: ⟨14,24,774,944⟩; 7: ⟨199,319,599,709⟩;

tread: 2: ⟨57,94,333⟩; 4: ⟨15,35,155⟩; 7: ⟨20,320⟩;

where: 2: ⟨67,124,393,1001⟩; 4: ⟨11,41,101,421,431⟩; 7: ⟨16,36,736⟩;

Какие документы (если они есть) удовлетворяют каждому из следующих фразовых запросов?

1. “fools rush in”
2. “fools rush in” and “angels fear to tread”

Упражнение 2.10 [*]. Проанализируйте следующий фрагмент координатного индекса, имеющего формат

word:document:⟨position, position, ...⟩;document: ⟨position, ...⟩

...

Gates: 1: ⟨3⟩; 2: ⟨6⟩; 3: ⟨2,17⟩; 4: ⟨1⟩;

IBM: 4: ⟨3⟩; 7: ⟨14⟩;

Microsoft: 1: ⟨1⟩; 2: ⟨1,21⟩; 3: ⟨3⟩; 5: ⟨16,22,51⟩;

Оператор /k в запросе word1 */k* word2 находит появление слова word1 в окрестности *k* слов от слова word2 (с обеих сторон), где *k* — положительное целое число. Таким образом, при $k = 1$ слово word1 должно быть соседом слова word2.

1. Опишите набор документов, удовлетворяющих запрос Gates /2 Microsoft.
2. Опишите множества значений *k*, при которых запрос Gates */k* Microsoft возвращает разные наборы документов.

Упражнение 2.11 [**]. Проанализируйте общую процедуру слияния двух инвертированных списков для заданного документа и определите позиции в документе, где удовлетворяется условие */k* (в принципе, для каждого термина существуют несколько позиций, где он появляется в документе). Начните с указателя на позицию, в которой впервые появляется каждый термин, и переходите далее по списку его появлений в документе, проверяя условие */k*. Каждый перенос указателя следует считать отдельным шагом. Обозначим через *L* общее количество появлений двух терминов в документе. Какова *O*-сложность процедуры слияния, если мы желаем в результате получить инвертированный список, содержащий позиции терминов?

Упражнение 2.12 [**]. Продумайте адаптацию основного алгоритма пересечения двух инвертированных списков (см. рис. 1.6) к варианту, изображенному на рис. 2.12, в котором обрабатываются запросы в окрестности термина. Наивный алгоритм для этой операции имеет сложность $O(PL_{\max}^2)$, где P — сумма длин инвертированных списков (т.е. сумма документных частот), а L_{max} — максимальная длина документа (в лексемах).

1. Тщательно изучите алгоритм и объясните, как он работает.
2. Какова сложность алгоритма? Тщательно обоснуйте свой ответ.
3. Существует ли более эффективный алгоритм для обработки определенных запросов и распределений данных? Какую сложность он имеет?

Упражнение 2.13 [**]. Предположим, что необходимо использовать процедуру пересечения лишь для определения списка документов, удовлетворяющих условию */k*, не возвращая инвертированный список, как показано на рис. 2.12. Для простоты будем считать, что $k \geq 2$. Обозначим через L общее количество появлений двух терминов в коллекции документов (т.е. сумму их частот в коллекции). Какое из следующих утверждений является истинным? Обоснуйте свой ответ.

1. Слияние можно выполнить за количество шагов, которое линейно зависит от параметра L и не зависит от параметра k, причем можно доказать, что каждый указатель перемещается только вправо.
2. Слияние можно выполнить за количество шагов, которое линейно зависит от параметра L и не зависит от параметра k, но указатель может перемещаться немонотонно (т.е. не только вправо).
3. В некоторых случаях для слияния может потребоваться kL шагов.

Упражнение 2.14 [**]. Как система информационного поиска может использовать сочетание координатного индекса и стоп-слов? В чем заключается потенциальная проблема и как ее разрешить?

2.5. Библиография и рекомендации для дальнейшего чтения

Исчерпывающее обсуждение посимвольной обработки текстов на восточно-азиатских языках изложено в работе Лунде (Lunde, 1998). Вероятно, наиболее распространенным подходом к индексированию китайских текстов являются индексы биграмм, хотя некоторые системы используют выделение слов из строки. Из-за различия между языком и системой письменности выделение строк из строки наиболее полезно при обработке японских текстов (Luk and Kwok, 2002; Kishida et al., 2005). Структура индекса k-грамм, построенного на основе несегментированного текста, отличается от структуры, описанной в разделе 3.2.2: там словарь k-грамм ссылается на список элементов в обычном словаре, в то время как здесь он ссылается непосредственно на инвертированные списки документа. Дальнейшее обсуждение вопросов выделения китайских слов можно найти в работах Шпрота, Эмерсона, Ценга, Гао и др. (Sproat et al., 1996; Sproat and Emerson, 2003; Tseng et al., 2005; и Gao et al., 2005).

Учет истинного регистра рассматривается в работе Лита и соавторов (Lita et al., 2003). Обзор работ по машинной морфологии содержится в работах Шпрота, Бисли и Картунена (Sproat, 1992; Beesley and Karttunen, 2003).

Идентификация языка, вероятно, впервые была применена в криптографии; например, в работе Кохейма (Konheim, 1981) описан алгоритм идентификации языка на основе *k*-грамм символов. Несмотря на то что позднее, по мере все более широкого использования текстов в электронной форме, были использованы другие методы, в частности на основе поиска характерных служебных слов и комбинаций букв, многие люди довольно успешно исследовали метод *k*-грамм (Beesley, 1998; Cavnar and Trenkle, 1994; Dunning, 1994). Определение языка письменного текста является относительно простой проблемой, в то время как идентификация языка звучащей речи продолжает сталкиваться с трудностями (Hughes et al., 2006).

Описанию экспериментов и обсуждению преимуществ и недостатков стемминга для английского языка посвящены работы Сэлтона, Хармана, Кровеца и Халла (Salton, 1989; Harman, 1991; Krovetz, 1995; Hull, 1996). В работе Холлинка и др. (Hollink et al., 2004) приведены подробные результаты, характеризующие эффективность методов, учитывающих языковую специфику на примере восьми европейских языков. В процентном выражении макроусредненная средняя точность, по сравнению с базовой системой, после удаления диакритических символов увеличивается на 23% (особенно полезным этот способ оказался при обработке текстов на финском, французском и шведском языках). Стемминг особенно эффективен при обработке финских (улучшение на 30%) и испанских (улучшение на 10%) текстов, однако для всех остальных языков, включая английский, выигрыш составил от 0 до 5%, а результаты, полученные с помощью лемматизации, еще хуже. Разделение сложных слов дает выигрыш для шведского языка на 25%, для немецкого — на 15%, а для голландского — только на 4%. В отличие от методов, учитывающих специфику конкретных языков, индексация символьных *k*-грамм (как в китайском языке) часто приводит к аналогичным или даже лучшим результатам: использование внутрисловных 4-грамм вместо слов дает выигрыш в 37% при работе с финским языком, 27% — при работе со шведским языком и 20% — при работе с немецким языком, хотя для остальных языков, таких как голландский, испанский и английский, выигрыш оказывается незначительным. В работе Томлисона (Tomlinson, 2003) представлены примерно такие же результаты. Бар-Илан и Гутман (Bar-Ilan and Gutman, 2005) предположили, что в во время их исследования (2003) основные коммерческие поисковые веб-системы испытывали трудности от недостатка подходящих методов обработки, учитывающих специфику языка; например, поисковая система `www.google.fr` в запросе l'électricité оказалась неспособной отделить артикль *l'* и возвращала ссылки на страницы, на которых содержалась точная строка вида "артикль+имя существительное".[12]

Классическое описание указателей пропусков для систем информационного поиска можно найти в работе Моффата и Цобеля (Moffat and Zobel, 1996). Более сложные методы описаны в работе Болди и Винья (Boldi and Vigna, 2005). Основной статьей по алгоритмам этого типа считается работа Пу (Pugh, 1990), в которой описаны многоуровневые указатели пропусков, обеспечивающие ожидаемую сложность поиска по спискам с оценкой $O(\log P)$ (эта сложность сравнима с оценкой сложности поиска по древовидным структурам данных) с меньшей сложностью реализации. На практике эффективность ис-

[12] Для русского языка можно считать доказанной важность учета морфологии в процессе поиска: все известные коммерческие системы в настоящий момент ее учитывают. — *Примеч. ред.*

пользования указателей пропусков зависит от параметров системы. В работе Моффата и Цобеля (Moffat and Zobel, 1996) сообщается о конъюнктивных запросах, которые с помощью указателей пропусков обрабатывались примерно в пять раз быстрее, но Бале и соавторы (Bahle et al., 2002) сообщают, что на современных процессорах использование указателей пропусков замедляет поиск, поскольку при этом увеличивается размер инвертированного списка (т.е., на эффективность поиска влияет в основном скорость ввода-вывода диска). В противоположность этому Штроман и Крофт (Strohman and Croft, 2007) вновь продемонстрировали хорошую производительность, достигнутую благодаря указателям пропусков в сочетании с архитектурой системы, предназначенной для оптимизации работы с большими объемами памяти и несколькими ядрами современных процессоров.

Джонсон и соавторы (Johnson et al., 2006) сообщают, что 11,7% всех запросов в двух логах веб-систем в 2002 году были фразовыми, хотя Камменхубер и соавторы (Kammenhuber et al., 2006) утверждают, что фразовые запросы образуют только 3% всех запросов на другом наборе данных. Сильверстейн и соавторы (Silverstein et al., 1999) отмечают, что многие запросы, не содержащие явных фразовых операторов, на самом деле являются неявными запросами на поиск фраз.

Глава 3

Словари и нечеткий[1] поиск

В главах 1 и 2 изложены идеи, лежащие в основе инвертированных индексов, которые предназначены для обработки булевых запросов и запросов с учетом близости слов запроса в документе. В данной главе рассматриваются методы, устойчивые к опечаткам в запросах и альтернативным вариантам написания слов. В разделе 3.1 описываются структуры данных, облегчающие поиск терминов в лексиконе инвертированного индекса. В разделе 3.2 исследуется идея о *запросе с джокерами* (wildcard query), т.е. о запросе, имеющем вид `a*e*i*o*u*` и предназначенном для поиска документов, содержащих любой термин, в который входят все пять гласных букв в указанном порядке. Символ `*` означает любую (возможно, пустую) строку символов. Пользователи посылают такие запросы в поисковые системы, когда они не имеют точного представления об правильном правописании термина или ищут документы, содержащие разные варианты указанного термина; например, запрос `automat*` предназначен для поиска документов, содержащих термин `automatic`, `automation` или `automated`.

В разделе 3.3 рассматриваются другие формы неточно сформулированных запросов с акцентом на орфографические ошибки. Пользователи делают орфографические ошибки либо случайно, либо потому, что искомый термин (например, `Herman`) в коллекции документов может встречаться в различных вариантах написания. В этом разделе подробно описывается множество приемов исправления орфографических ошибок как в отдельных терминах, так и во всей строке запроса в целом. В разделе 3.4 исследуется метод поиска в лексиконе фонетически схожих терминов. Это может оказаться особенно полезным в ситуациях, когда пользователь не знает точно, как пишется имя собственное в документах из коллекции (например, при поиске имени `Herman`).

Поскольку в этой главе разрабатываются разные варианты инвертированных индексов, мы иногда используем выражение *стандартный инвертированный индекс* (standard inverted index), подразумевая инвертированный индекс, описанный в главах 1 и 2, т.е. индекс, в котором каждому проиндексированному термину соответствует инвертированный список, в котором перечислены документы из коллекции.

3.1. Поисковые структуры для словарей

Имея инвертированный индекс и запрос, необходимо определить, существует ли в лексиконе каждый термин из запроса, и в случае положительного ответа идентифицировать указатель на соответствующие термину словопозиции. В этой операции просмотра

[1] В англоязычной литературе чаще употребляется “fuzzy”, чем “tolerant”. Соответственно, мы выбрали наиболее распространенный русский вариант для перевода: “нечеткий” поиск, вместо непонятного русскоязычному читателю выражения “толерантный” поиск. Чуть реже используется термин “поиск по сходству”. — *Примеч. ред.*

лексикона используется классическая структура данных — *словарь* (dictionary). Для ее реализации существует два семейства методов: хеширование и деревья поиска. В литературе по структурам данным элементы лексикона (в нашем случае — термины) часто называются *ключами* (keys). Выбор решения (хеширование или дерево поиска) зависит от множества обстоятельств: 1) сколько ключей требуется, 2) останется ли это количество неизменным или будет часто изменяться (во втором случае возникает новый вопрос: "Хотите ли вы только включать в словарь новые ключи или еще и удалять старые?"), 3) какова относительная частота обращения к различным ключам.

Некоторые поисковые системы для просмотра словаря используют *хеширование* (hashing). Каждый элемент словаря (ключ) отображается в целое число (хешируется) на достаточно большом интервале, в котором коллизии маловероятны; коллизии разрешаются с помощью вспомогательных структур, иногда требующих дополнительной поддержки. [2] В момент запроса каждый термин запроса хешируется отдельно, затем совершается переход по указателю на соответствующую запись. Для разрешения коллизий используется дополнительная логика. К сожалению, близкие варианты термина из запроса найти непросто (например, варианты слова `resume` со знаком ударения и без), поскольку они могут хешироваться в совершенно разные целые числа. Например, мы не можем найти все термины, начинающиеся с префикса `automatic` (см. раздел 3.2). В средах, где размер лексикона постоянно растет (например, в вебе), хеш-функция, разработанная с учетом текущей ситуации, через несколько лет уже может оказаться неработоспособной.

Многие из этих проблем решаются с помощью деревьев поиска; например, они позволяют легко находить все термины лексикона, начинающиеся со слова `automat`. Наиболее известным деревом поиска является *двоичное дерево* (binary tree), в котором каждый внутренний узел имеет два дочерних узла. Поиск термина начинается с корня дерева. Каждый внутренний узел (включая корень) представляет собой двоичный тест, в зависимости от результата которого поиск продолжается по одному из двух поддеревьев, ветвящихся из данного узла. На рис. 3.1 приведено двоичное дерево поиска, использованное для организации словаря. Эффективный поиск (при котором количество сравнений составляет $O(\log M)$) требует, чтобы дерево было сбалансированным: количество терминов в обоих поддеревьях любого узла должно быть одинаковым или отличаться только на единицу. Принципиальным моментом является *восстановление баланса*: после вставки или удаления термина следует восстановить баланс двоичного дерева.

Для того чтобы облегчить восстановление баланса, можно допустить наличие нескольких поддеревьев у каждого внутреннего узла в фиксированном интервале. Как правило, для организации словарей используется *B-дерево* (B-tree) — дерево, количество дочерних узлов которого у каждого внутреннего узла изменяется в диапазоне $[a, b]$, где a и b — соответствующие положительные целые числа. На рис. 3.2 показан пример, в котором $a = 2$, а $b = 4$. Как и в двоичном дереве, изображенном на рис. 3.1, каждый узел ветвления, расположенный под внутренним узлом, представляет собой тест для определенного диапазона символьных последовательностей. B-дерево может рассматриваться как дерево, в котором несколько уровней двоичного дерева "сливаются" в один. Это обстоятельство является особенно ценным преимуществом в ситуациях, когда часть словаря хранится на диске. В таком случае это слияние позволяет предварительно выбирать

[2] Так называемые совершенные хеш-функции предназначены для предотвращения коллизий, но они немного сложнее с точки зрения реализации и вычислений. — *Примеч. ред.*

данные для непосредственно следующих двоичных тестов. В таких случаях числа a и b определяются размерами блоков памяти на диске. Библиография о деревьях поиска и B-деревьях содержится в разделе 3.5.

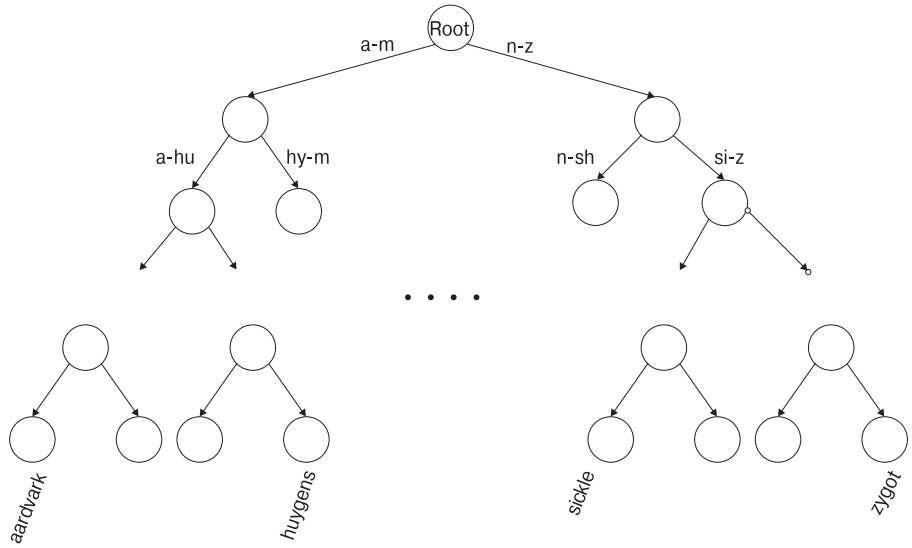


Рис. 3.1. Дерево двоичного поиска. В данном примере, начиная с корня, термины лексикона разделяются на два поддерева: в первом дереве записаны все слова, начинающиеся с букв от `a` *до* `m`*, а во втором — все остальные*

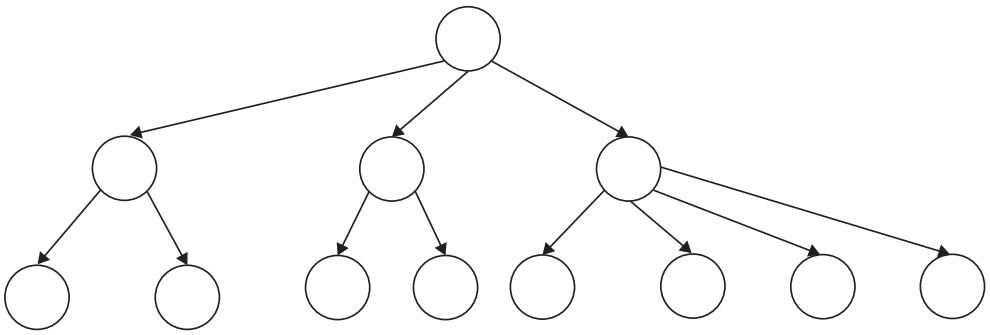


Рис. 3.2. B-дерево. В данном примере каждый внутренний узел имеет от двух до четырех дочерних узлов

Следует подчеркнуть, что в отличие от хеширования для применения деревьев поиска требуется, чтобы символы, использованные в документах из коллекции, были упорядочены по заранее установленной схеме; например, 26 букв алфавита английского языка всегда перечисляются в определенном порядке от `A` до `Z`. Некоторые восточно-азиатские языки, например китайский, не имеют однозначного порядка символов, хотя в настоящее время все языки (включая китайский и японский) уже приняли стандартную систему упорядочения своих наборов символов.

3.2. Запросы с джокером

Запросы с джокерами используются в следующих ситуациях: 1) когда пользователь не знает, как правильно пишется термин (например, варианты `Sydney` или `Sidney` можно объединить в запросе с джокером `S*dney`); 2) когда пользователь знает о существовании нескольких вариантов написания термина и (сознательно) ищет документы, в которых содержится хотя бы один из этих вариантов (например, `color` и `colour`); 3) когда пользователь ищет документы, содержащие вариант термина, унифицированный в результате стемминга, но не уверен, что поисковая система выполняет стемминг (например, варианты `judicial` и `judiciary` можно объединить в шаблонном запросе `judicia*`); 4) когда пользователь не уверен в правильности воспроизведения иностранного слова или фразы (например, `Universit* Stuttgart`).

Запрос, имеющий вид, например, `mon*`, называется *запросом с замыкающим джокером* (trailing wildcard query), поскольку символ `*` используется только один раз в самом конце поисковой строки. Для обработки таких запросов удобно использовать дерево поиска над словарем: перемещаясь по дереву вниз и поочередно переходя по ветвям, соответствующим буквам `m`, `o` и `n`, можно перебрать все множество W терминов словаря с префиксом `mon`. В заключение, после $|W|$ просмотров стандартного инвертированного индекса, можно извлечь все документы, содержащие хотя бы один термин из множества W.

А что можно сказать о шаблонных запросах, в которых символ `*` не обязательно стоит в конце поисковой строки? Прежде чем решить эту общую задачу, рассмотрим небольшое обобщение запроса с замыкающим джокером. Сначала рассмотрим *запросы с ведущим джокером* (leading wildcard query), т.е. запросы вида `*mon`. Затем рассмотрим *обратные B-деревья* (reverse B-tree) над словарем, в которых каждый путь от корня к листу в B-дереве соответствует термину в словаре, записанному *в обратном порядке* (backwards). Итак, термин `lemon` в B-дереве будет представлен путем `root-n-o-m-e-l`. Проход вниз по обратному B-дереву позволяет перечислить в лексиконе все термины R, имеющие заданный постфикс.

Фактически с помощью обычного B-дерева в сочетании с обратным B-деревом можно решить еще более общую задачу: обработку запросов, в которых есть один символ `*`, например `se*mon`. При этом для перечисления множества W терминов, содержащих префикс `se` и непустой суффикс, можно использовать обычное B-дерево, а для перечисления множества R терминов, заканчивающихся суффиксом `mon`, можно использовать обратное B-дерево. Далее мы определяем пересечение $W \cap R$ этих двух множеств и получаем множество терминов, начинающихся префиксом `se` и заканчивающихся суффиксом `mon` (например, запрос `ba*ba` приведет к совпадению с термином `ba` из $W \cap R$; такой случай надо устранить). В заключение с помощью стандартного инвертированного индекса можно получить все документы, содержащие хотя бы один термин из пересечения. Следовательно, с помощью нормального и обратного B-деревьев можно обрабатывать запросы, содержащие один символ `*`.

3.2.1. Запросы с джокером общего вида

Перейдем к изучению двух методов обработки запросов с джокером общего вида. Эти методы используют одну и ту же стратегию: сначала заданный запрос с джокером q_w выражается как булев запрос Q с помощью специально сконструированного индекса, так чтобы ответ на запрос Q являлся объемлющим множеством (superset) множества терминов лексикона, со-

ответствующих запросу q_w, а затем выполняется проверка каждого термина в ответе на запрос Q на основе запроса q_w, при этом отбрасываются термины лексикона, не соответствующие запросу q_w. В этот момент мы получаем термины лексикона, соответствующие запросу q_w, и можем прибегнуть к стандартному инвертированному индексу.

Перестановочные индексы. Первым специальным индексом для запросов с джокером общего вида является *перестановочный индекс* (permuterm index), представляющий собой разновидность инвертированного индекса. Сначала введем в набор символов специальный символ `$`, обозначающий конец термина. Следовательно, термин `hello` представляется как дополненный термин `hello$`. Далее построим перестановочный индекс терминов, в котором все варианты, полученные циклической перестановкой символов исходного термина (снабженного символом `$`), связаны с исходным термином лексикона. Фрагмент перестановочного индекса, содержащий элементы для термина `hello`, представлен на рис. 3.3.

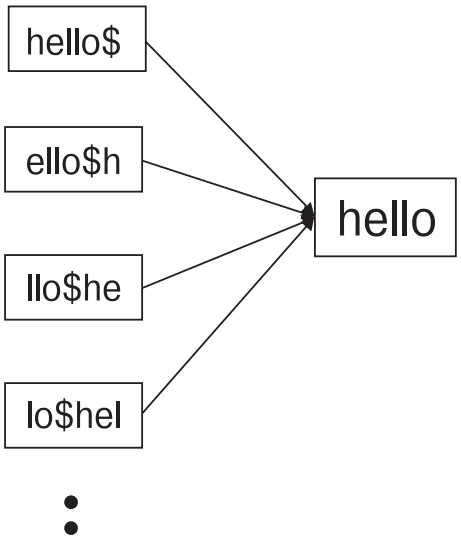


Рис. 3.3. Фрагмент перестановочного индекса

Набор терминов с перестановкой символов называется *лексиконом перестановок* (permuterm vocabulary).

Как этот индекс помогает обрабатывать запросы с джокером? Рассмотрим запрос с джокером `m*n`. Ключевой момент — *применить к запросу циклическую перестановку* (rotate) так, чтобы символ `*` оказался в конце строки. Таким образом, преобразованный запрос принимает вид `n$m*`. Далее осуществляется просмотр этой строки в перестановочном индексе, в ходе которого поиск термина `n$m*` (по дереву поиска) приводит к нахождению перестановок (помимо прочих) терминов `man` и `moron`.

Теперь, когда перестановочный индекс позволяет идентифицировать исходные термины лексикона, соответствующие запросу с джокером, с помощью стандартного инвертированного индекса можно найти соответствующие документы. Таким образом, можно обработать любой запрос с одиночным джокером. А как обработать запрос вида `fi*mo*er`? В таком случае сначала следует перечислить термины словаря, которые принадлежат перестановочному индексу `er$fi*`. Не все такие термины словаря имеют

в середине строку mo — мы отфильтровываем их с помощью полного перебора, проверяя каждого кандидата, содержит ли он символы mo. В данном примере термин fishmonger пройдет фильтрацию, а термин filibuster — нет. Затем с помощью стандартного инвертированного индекса производится поиск по терминам, которые прошли фильтрацию. Недостатком перестановочного индекса является то, что его словарь становится довольно большим, поскольку он содержит все перестановки каждого термина.

Обратите внимание на тесную связь между B-деревом и перестановочным индексом. Действительно, эту структуру можно интерпретировать как перестановочное B-дерево. Однако мы предпочитаем использовать стандартную терминологию и отличаем перестановочный индекс от B-дерева, позволяющего выбирать все перестановки с заданным префиксом.

3.2.2. k-граммный индекс для шаблонных запросов

Несмотря на простоту перестановочного индекса его использование может привести к чрезмерному увеличению из-за большого количества перестановок для каждого термина; для словаря английских терминов это увеличение может оказаться десятикратным. Рассмотрим теперь второй метод обработки запросов с джокерами, основанный на *k*-граммном индексе. Этот метод используется также в разделе 3.3.4. *K-грамма* (*k*-gram) — это последовательность, состоящая из *k* символов.[3] Таким образом, cas, ast и stl — это 3-граммы, содержащиеся в термине castle. Для обозначения начала или конца термина используется специальный символ $, так что полный набор 3-грамм, образованных от слова castle, таков: $ca, cas, ast, stl, tle и le$.

При использовании *k-граммного индекса* словарь содержит все *k*-граммы, образованные из всех терминов лексикона. Каждый инвертированный список ставит в соответствие *k*-грамме все термины лексикона, содержащие данную *k*-грамму. Например, 3-грамма etr должна ссылаться на термины лексикона, такие как metric и retrieval. Соответствующий пример приведен на рис. 3.4.

Рис. 3.4. Пример инвертированного списка для 3-граммного индекса. В данном случае продемонстрирована 3-грамма etr. Соответствующие термины лексикона в инвертированном списке перечислены в лексикографическом порядке

Как обработать запрос с джокером с помощью такого индекса? Рассмотрим в качестве примера запрос re*ve. Мы ищем документы, содержащие любой термин, начинающийся с префикса re и заканчивающийся суффиксом ve. Соответственно, необходимо обработать булев запрос $re AND ve$. Поиск производится по 3-граммному индексу. В результате возникает список соответствующих терминов, таких как relive, remove и retrieve. Затем осуществляется поиск каждого из этих терминов в стандартном инвертированном индексе и выявляются документы, соответствующие данному запросу.

Однако с использованием *k*-граммных индексов связана одна трудность, для устранения которой требуется дополнительный шаг. Рассмотрим использование 3-граммного индекса, предназначенного для обработки запроса red*. Следуя указанным инструкциям, сначала передадим 3-граммному индексу булев запрос $re AND RED. В результате

[3] Интересно отметить, что для последовательности слов, а не символов, и в отечественной, и в зарубежной литературе чаще используются “n-грамма” и “n-gram” соответственно. — *Примеч. ред.*

оказывается, что ему соответствуют такие термины, как `retired`, содержащие конъюнкцию двух 3-грамм `$re` и `red`, но не соответствующие исходному шаблонному запросу `red*`.

Для того чтобы справиться с этой проблемой, введем новый этап — *постфильтрацию* (postfiltering), в ходе которой все термины, полученные в результате выполнения булева запроса к 3-граммному индексу, по отдельности сравниваются с исходным шаблоном `red*`. Эта операция сводится к простому сравнению строк и отбрасывает все термины, не соответствующие исходному запросу, такие как `retired`. Затем, как обычно, осуществляется поиск оставшихся терминов в стандартном инвертированном индексе.

Как мы убедились, обработку шаблонного запроса можно свести к нескольким терминам, каждый из которых становится однословным запросом к стандартному инвертированному индексу. Поисковые системы позволяют комбинировать запросы с джокерами и логические операторы, например `re*d AND fe*ri`. Какова семантика этого запроса? Поскольку каждый запрос с джокером превращается в дизъюнкцию однословных запросов, приемлемой интерпретацией данного примера может быть конъюнкция дизъюнкций: мы ищем все документы, содержащие любой термин, соответствующий запросу `re*d`, *и* любой термин, соответствующий запросу `fe*ri`.

Даже без логических комбинаций запросов с джокерами их обработка требует довольно больших затрат, поскольку появляется дополнительный просмотр специального индекса, фильтрация и, в заключение, стандартный инвертированный индекс. Поисковая система, разумеется, может поддерживать такие широкие функциональные возможности, но, как правило, они скрываются за интерфейсом (например, “Расширенный поиск”), который большинство пользователей никогда не применяют. Присутствие таких функциональных возможностей в интерфейсе поисковой системы часто провоцирует пользователей применять их без особой надобности (например, пользователи могут набрать префикс запроса, за которым следует символ `*`), что приводит к увеличению нагрузки на поисковую систему.

? ***Упражнение 3.1.*** В перестановочном индексе каждый термин перестановочного лексикона ссылается на термин (или термины) исходного лексикона, от которого он происходит. Как много терминов из исходного лексикона может оказаться в инвертированном списке для термина из перестановочного лексикона?

Упражнение 3.2. Запишите элементы перестановочного индексного словаря, порождаемые термином mama.

Упражнение 3.3. Допустим, что вы хотите обработать запрос s*ng с помощью перестановочного шаблонного индекса. Какие ключи или ключ следует искать?

Упражнение 3.4. Проанализируйте рис. 3.4. В подписи к рисунку указано, что термины лексикона в инвертированном списке перечислены в лексикографическом порядке. Чем полезен такой порядок?

Упражнение 3.5. Вернемся к запросу fi*mo*er из раздела 3.2.1. Какой булев запрос по биграммному индексу порождается этим запросом? Можете ли вы привести пример термина, который соответствует перестановочному запросу из раздела 3.2.1, но при этом не удовлетворяет этому булеву запросу?

Упражнение 3.6. Приведите пример предложения, которое ошибочно соответствует шаблонному запросу mon*h, если в ходе поиска просто используется конъюнкция биграмм.

3.3. Исправление опечаток

Перейдем к проблеме исправления опечаток в запросах. Предположим, что пользователь желает найти документы, содержащие термин `carrot`, введя запрос `carot`. Поисковая система Google (`www.google.com/jobs/britney.html`) сообщает, что все нижеперечисленные запросы считаются искаженным запросом `britney spears`: `britian spears`, `britney's spears`, `brandy spears` и `prittany spears`. Проанализируем два этапа решения этой проблемы: первый этап основан на *расстоянии редактирования* (edit distance), а второй — на *пересечении k-грамм* (*k*-gram overlap). Прежде чем перейти к алгоритмическим деталям этих методов, посмотрим, как поисковые системы исправляют опечатки.

3.3.1. Реализация исправления опечаток

В основе большинства алгоритмов исправления опечаток лежат два фундаментальных принципа.

1. Из всех альтернативных правильных способов написания искаженного запроса выбирается “ближайший”. Для этого необходимо понятие близости между двумя запросами. Меры близости описываются в разделе 3.3.3.
2. Если два правильно записанных запроса связаны (или почти связаны) друг с другом, то выбирается более распространенный вариант. Например, запросы `grunt` и `grant` выглядят одинаково подходящими вариантами для исправления запроса `grnt`. Следовательно, алгоритм должен выбрать тот вариант, который чаще используется. Наиболее простой оценкой частоты использования слова является количество появлений этого термина в коллекции документов; следовательно, если слово `grunt` встречается чаще, чем слово `grant`, то следует выбрать именно его. Во многих поисковых системах, особенно в вебе, используется другой способ оценки распространенности термина. Его идея заключается в том, чтобы использовать в качестве исправления то, что чаще всего встречается в запросах других пользователей. В частности, если слово `grunt` в запросах встречается чаще, чем слово `grant`, то, скорее всего, пользователь, напечатавший слово `grnt`, хотел напечатать слово `grunt`.

В начале раздела 3.3.3 мы рассмотрим понятие близости между запросами, а также способы ее эффективного вычисления. На этих способах основаны алгоритмы исправления ошибок в запросах пользователей; затем эти функциональные возможности предоставляются пользователям в разных вариантах.

1. В ответ на запрос `carot` всегда возвращаются документы, содержащие слово `carot`, а также все возможные “исправленные” варианты этого слова, включая `carrot` и `tarot`.
2. Если слова `carot` нет в словаре, выполняется вариант 1.

3. Если количество документов, возвращенных в ответ на запрос, меньше установленного порога (например, меньше пяти документов), выполняется вариант 1.
4. Если количество документов, возвращенных в ответ на запрос, меньше установленного порога, то поисковая система предлагает пользователю *вариант исправления* (spelling suggestion): это предложение состоит из исправленных терминов. Таким образом, поисковая система в ответ на запрос пользователя может спросить: "Вы имели в виду слово `carrot`?"

3.3.2. Формы исправления ошибок

Остановимся на двух формах исправления ошибок: *исправлении изолированного термина* (isolated-term correction) и *исправлении с учетом контекста* (context-sensitive correction). При исправлении изолированного термина термины запроса исправляются по отдельности. Пример со словом `carot` иллюстрирует этот подход. Этот метод не позволяет, например, обнаружить, что запрос `flew form Heathrow` содержит искаженный термин `from`, поскольку каждый из терминов запроса по отдельности записан правильно.

Рассмотрим два метода решения задачи исправления изолированных терминов: расстояние редактирования и перекрытие k-грамм. После этого мы перейдем к исправлению с учетом контекста.

3.3.3. Расстояние редактирования

Расстояние редактирования между двумя строками символов s_1 и s_2 — это минимальное количество *операций редактирования* (edit operations), с помощью которых строку s_1 можно трансформировать в строку s_2. Операции редактирования, позволяющие это сделать, включают в себя следующие преобразования: 1) вставка символа в строку, 2) удаление символа из строки и 3) замена символа в строке другим символом. При указанном наборе операций редактирования расстояние редактирования называется *расстоянием Левенштейна* (Levenshtein distance). Например, расстояние редактирования между словами `cat` и `dog` равно трем. Расстояние редактирования можно обобщить, если разным операциям редактирования присвоить разные веса. Например, операции замены символа `s` символом `p` можно присвоить более высокий вес, чем операции замены символа `s` символом `a` (так как буква `a` ближе к букве `s` на клавиатуре). Присвоение весов с учетом правдоподобности замены на практике оказалось очень эффективным (см. раздел 3.4, в котором рассматривается проблема фонетической схожести). Однако в нашем изложении мы будем придерживаться допущения, что все операции редактирования имеют один и тот же вес.

Хорошо известно, как вычислить расстояние редактирования между двумя строками за время $O(|s_1|\times|s_2|)$, где $|s_i|$ — длина строки s_i. Для этого используется алгоритм динамического программирования, представленный на рис. 3.5, где строки s_1 и s_2 представлены в виде массивов. Этот алгоритм заполняет все (целочисленные) ячейки матрицы m, размеры которой равны длинам соответствующих строк; после выполнения алгоритма элемент (i, j) содержит расстояние редактирования между строками, состоящими из первых i символов строки s_1 и первых j символов строки s_2. Основной шаг динамического программирования отображен в строках 8–10 (рис. 3.5), в которых вычисляется минимум трех величин с учетом замены символа в строке s_1, вставки символа в строку s_1 и вставки символа в строку s_2.

```
EditDistance(s1, s2)
 1  int m[|s1|, |s2|] = 0
 2  for i ← 1 to |s1|
 3  do m[i, 0] = i
 4  for j ← 1 to |s2|
 5  do m[0, j] = j
 6  for i ← 1 to |s1|
 7  do for j ← 1 to |s2|
 8        do m[i, j] = min{m[i–1,j–1] + if(s1[i] = s2[j]) then 0 else 1fi,
 9                         m[i–1,j] +1,
10                         m[i,j–1] + 1}
11  return m[|s1|, |s2|]
```

Рис. 3.5. Алгоритм динамического программирования для вычисления расстояния редактирования между строками s_1 *и* s_2

Пример вычисления расстояния Левенштейна по алгоритму, представленному на рис. 3.5, продемонстрирован на рис. 3.6. Типичная ячейка $[i, j]$ разделена на четыре части, образующие ячейки размером 2×2. В правой нижней ячейке записывается минимум остальных трех чисел, которые соответствуют основному шагу динамического программирования в алгоритме, представленном на рис. 3.5. Остальные три ячейки содержат числа $m[i-1, j-1] + 0$ или 1 в зависимости от того, выполняется ли условие $s_1[i] = s_2[j]$, а также числа $m[i-1, j] + 1$ и $m[i-1, j] + 1$. Ячейки с числами, выделенными курсивом, отражают путь, определяющий расстояние Левенштейна.

		f		a		s		t	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
c	**1** **1**	*1* *2*	*2* *1*	**2** **2**	3 **2**	**3** **3**	4 **3**	**4** **4**	5 **4**
a	**2** **2**	**2** 3	**2** **2**	*1* *3*	*3* *1*	*3* *2*	*4* *2*	4 **3**	5 **3**
t	**3** **3**	**3** 4	**3** **3**	3 4	**2** **2**	*2* *3*	*3* *2*	*2* *3*	*4* *2*
s	**4** **4**	**4** 5	**4** **4**	4 5	**3** **3**	**2** 4	3 **2**	*3* *3*	*3* *3*

Рис. 3.6. Пример вычисления расстояния Левенштейна. Ячейка (i, j) таблицы разбита на четыре части размером 2×2, в которых записаны три числа, минимум которых дает четвертое число. Ячейки, определяющие расстояние редактирования в данном примере, выделены курсивом

Тем не менее проблема исправления опечаток, помимо вычисления расстояния редактирования, требует несколько большего: для заданных множества строк *V* (соответствующих терминам в лексиконе) и строки запроса *q* необходимо найти строку или строки из множества *V* с минимальным расстоянием редактирования до запроса *q*. Эту проблему можно рассматривать как задачу декодирования, в которой кодовые слова (строки из множества *V*) заданы заранее. Очевидный способ решения — вычислить расстояние редактирования от запроса *q* до каждой из строк, принадлежащих множеству *V*, а затем выбрать строку или строки с минимальным расстоянием. Этот исчерпывающий поиск является чрезмерно затратным. По этой причине для эффективного поиска терминов в словаре, имеющих небольшое расстояние редактирования до терминов запроса, на практике применяется большое количество эвристических правил.

Простейший эвристический прием заключается в ограничении поиска терминами словаря, начинающимися с той же буквы, что и строка запроса; расчет делается на то, что орфографические ошибки редко делаются с первой же буквы запроса. Более сложный вариант этого эвристического приема предусматривает использование перестановочного индекса, в котором игнорируется заключительный символ `$`. Рассмотрим множество всех перестановок строки запроса `q`. Для каждой перестановки `r` из этого множества мы проходим по B-дереву перестановочного индекса, тем самым идентифицируя все термины словаря, перестановки которых начинаются с буквы `r`. Например, если запрос *q* представляет собой строку `mase` и мы рассматриваем перестановку `r = sema`, то мы можем извлечь из словаря такие термины, как `semantic` и `semaphore`, имеющие довольно большое расстояние редактирования до запроса *q*. К сожалению, при этом мы можем пропустить более подходящие термины словаря, такие как `mare` и `mane`. Для того чтобы решить эту проблему, следует уточнить схему перестановок: при каждой перестановке до прохода по B-дереву мы пропустим суффикс, состоящий из *l* символов. Это гарантирует нам, что каждый термин из множества *R*, извлеченный из словаря, будет содержать "длинную" подстроку, общую с запросом *q*. Число *l* может зависеть от длины строки *q*. В качестве альтернативы число *l* можно сделать фиксированной константой, например двойкой.[4]

3.3.4. K-граммные индексы для исправления опечаток

Для того чтобы еще больше ограничить множество терминов лексикона, для которых вычисляется расстояние редактирования до терминов из запроса, можно использовать *k*-граммный индекс, описанный в разделе 3.2.2. Этот индекс позволяет найти в лексиконе термины с небольшим расстоянием редактирования до запроса *q*. Найдя эти термины, мы можем определить среди них термины с минимальным расстоянием редактирования до запроса *q*.

---

[4] К сожалению, этот метод не только требует поиска минимального общего префикса в B-дереве (логарифмическая сложность от размера словаря), но и не слишком эффективен, если мы хотим разрешить ошибку более чем в одной позиции слова (см. замечание о поиске с двумя джокерами в разделе 3.2.1). Альтернативной эвристикой в методе расстояния редактирования является метод скелетов (skeleton), заключающийся в выделении всех скелетов — подпоследовательностей букв из терминов индекса и запроса, таких, что их длина меньше исходного термина на заданный порог (1 или 2 буквы). Хеширование скелетов терминов словаря и хеш-поиск всех скелетов термина из запроса дают за линейное время полный список кандидатов на коррекцию. — *Примеч. ред.*

Фактически *k*-граммный индекс используется для поиска терминов лексикона, содержащих большое количество *k*-грамм, общих с запросом. Идея заключается в том, что при разумной трактовке выражения "большое количество общих *k*-грамм" процесс поиска по существу сводится к однократному просмотру "словопозиций"[5] для *k*-грамм, входящих в запрос *q*.

Например, на рис. 3.7 показана часть словопозиций для трех биграмм в запросе `bord`. Допустим, что мы желаем найти термины лексикона, содержащие по крайней мере две из этих трех биграмм. Однократное сканирование записей (как описано в главе 1) позволило бы перечислить все такие термины; в примере, показанном на рис. 3.7, перечислены термины `aboard`, `boardroom` и `border`.

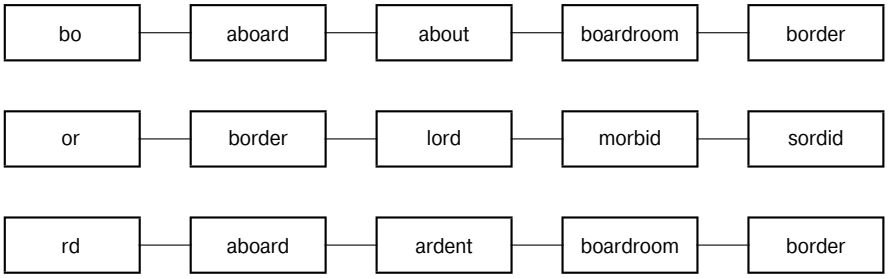


Рис. 3.7. Совпадение по крайней мере двух из трех биграмм в запросе `bord`

Применение "в лоб" пересечения инвертированных списков с помощью их последовательного просмотра сразу обнаруживает недостатки требования лишь присутствия в терминах лексикона фиксированного количества *k*-грамм из запроса *q*: при этом идентифицируются термины наподобие `boardroom`, представляющие собой "неправдоподобное" исправление слова `bord`. Следовательно, необходимы более тонкие меры перекрытия *k*-грамм между термином лексикона и запросом *q*. Метод пересечения списков с помощью их последовательного просмотра можно модифицировать, применив *коэффициент Жаккара* (Jaccard coefficient), характеризующий перекрытие двух множеств *A* и *B* и равный $|A \cap B|/|A \cup B|$. Мы рассматриваем множество *k*-грамм в запросе *q* и множество *k*-грамм в термине лексикона. В процессе сканирования мы переходим от одного термина лексикона *t* к следующему, вычисляя "на лету" коэффициент Жаккара для строк *q* и *t*. Если коэффициент превышает установленный порог, то термин *t* добавляется в результаты; если нет, то происходит переход к следующему термину в инвертированном списке. Для того чтобы вычислить коэффициент Жаккара, необходимо иметь множества *k*-грамм для строк *q* и *t*.

Поскольку мы просматриваем "словопозиции" всех *k*-грамм в запросе *q*, то сразу получаем все эти *k*-граммы в свое распоряжение. А как получить *k*-граммы для термина *t*? В принципе, можно было бы выделить их все "на лету". На практике это не только неэффективно, но и практически невозможно; в подавляющем большинстве случаев сами записи содержат не саму строку *t*, а лишь ее идентификатор. Следует подчеркнуть, что для вычисления коэффициента Жаккара необходимо знать лишь длину строки *t*. Для того чтобы убедиться в этом, напомним пример, изображенный на рис. 3.7, и обратим

[5] В данном случае в качестве "слова" выступает k-грамма, а в качестве "документа" — весь термин словаря. — *Примеч. ред.*

внимание на точку, в которой сканирование запроса q = `bord` достигает термина t = `boardroom`. Нам известно, что обе эти биграммы соответствуют критерию поиска. Если словопозиции хранят (заранее вычисленное) количество биграмм в слове `boardroom` (а именно — 8), то у нас имеется вся информация для вычисления коэффициента Жаккара: 2/(8+3–2); числитель вычисляется по количеству совпадений в записях (в данном случае — два: `bo` и `rd`), а знаменатель — это разность между суммой биграмм в словах `bord` и `boardroom` и количеством совпадений в записях.

Коэффициент Жаккара можно заменить другими показателями, допускающими эффективное вычисление "на лету" в ходе сканирования записей. Как использовать их для исправления опечаток? Один из эмпирических методов заключается в следующем: сначала для перечисления множества кандидатов среди терминов лексикона, представляющих собой потенциальные исправления запроса q, используется k-граммный индекс. Затем вычисляется расстояние редактирования между запросом q и каждым термином из указанного множества. Далее из этого множества выбираются термины с небольшим расстоянием редактирования от запроса q.

3.3.5. Исправление опечаток с учетом контекста

Иногда механизм исправления ошибок в отдельных терминах не срабатывает, как, например, в запросе `flew form Heathrow`, где все три термина написаны правильно. Если в ответ на фразу, подобную этой, будет возвращено лишь небольшое количество документов, то поисковая система может принять решение исправить этот запрос на `flew from Heathrow`. Простейший способ сделать это — перечислить исправления для каждого из трех терминов (используя методы, описанные в разделе 3.3.4), даже если все термины запроса написаны правильно, а затем попробовать произвести замену каждого термина фразы. В примере `flew form Heathrow` перечисление должно содержать фразы наподобие `fled from Heathrow` и `flew fore Heathrow`. Для каждой такой фразы поисковая система выполняет запрос и определяет количество соответствий.

Если исправлений отдельных терминов будет слишком много, то такое перечисление может оказаться очень затратным; мы можем столкнуться с большим количеством комбинаций альтернатив. Для того чтобы сэкономить память, используется несколько эвристических приемов. В нашем примере при создании альтернатив для терминов `flew` и `form` мы оставим только те комбинации, которые чаще остальных встречаются в коллекции документов или в логах запросов, в которых хранятся предыдущие запросы пользователей. Например, мы можем оставить словосочетание `flew from` как возможную альтернативу и попробовать расширить ее до трехсловного запроса, проигнорировав варианты `fled fore` и `flea form`. В данном случае словосочетание из двух слов `filed fore` является менее вероятным, чем сочетание `flew form`. Затем достаточно просто расширить список наиболее часто встречающихся словосочетаний из двух слов (например, `flew from`) с помощью исправлений термина `Heathrow`. В качестве альтернативы использованию статистики биграмм в коллекции можно использовать лог запросов, заданных пользователями; разумеется, он может содержать в том числе запросы с ошибками.

? ***Упражнение 3.7.*** Докажите, что расстояние редактирования между строками s_1 и s_2 не может превышать $\max\{|s_1|, |s_2|\}$, где $|s_i|$ — длина строки s_i.

Упражнение 3.8. Вычислите расстояние редактирования между строками paris и alice. Запишите массив размером 5×5, содержащий расстояния между всеми префиксами, вычисленные по алгоритму, описанному на рис. 3.5.

Упражнение 3.9. Напишите псевдокод, демонстрирующий детали вычислений коэффициента Жаккара "на лету" при сканировании записей *k*-граммного индекса.

Упражнение 3.10. Вычислите коэффициенты Жаккара для запросов bord и каждого термина, изображенного на рис. 3.7 и содержащего биграмму or.

Упражнение 3.11. Проанализируйте запрос catched in the rye, состоящий из четырех терминов, в предположении, что каждый из этих терминов при изолированном исправлении опечаток имеет пять альтернатив. Сколько возможных правильных фраз следует рассмотреть, если память для хранения правильных фраз не ограничена и для каждого из терминов можно испытать все шесть вариантов?

Упражнение 3.12. Для каждого из префиксов в запросе — catched, catched in и catched in the — существует большое количество подстановочных префиксов, возникающих на основе каждого термина и его альтернатив. Допустим, что мы ограничились только десятью наиболее вероятными подстановочными префиксами, ориентируясь на частоту их появления в коллекции документов. Остальные префиксы исключаются из рассмотрения и не используются для расширения до более длинных префиксов. Таким образом, если словосочетание batched in не входит в первую десятку наиболее распространенных двухсловных запросов в коллекции документов, мы не должны рассматривать никаких расширений комбинации batched in в качестве возможных вариантов исправления catched in the rye. Сколько возможных подстановочных префиксов мы исключаем на каждом этапе?

Упражнение 3.13. Можно ли утверждать, что использование только десяти наиболее вероятных подстановочных префиксов для фразы catched in приведет к одному из десяти наиболее распространенных подстановочных префиксов для фразы catched in the?

3.4. Фонетические исправления

Последний метод, используемый при нечетком поиске, предназначен для *фонетических исправлений* (phonetic correction), т.е. ошибок, возникающих, когда пользователь записывает запрос так, как он его слышит. Такие алгоритмы особенно полезны для поиска имен людей. Основная идея заключается в генерировании для каждого термина "фонетического хеша", т.е. терминам, звучащим одинаково, ставится в соответствие одно и то же число. Эта идея возникла в начале XX века в международных отделах полиции, перед которыми стояла задача поиска преступников, не обращая внимания на разное написание этих имен в разных странах. В основном этот метод используется для исправления фонетических ошибок в собственных именах.

Алгоритмы фонетического хеширования обычно называются *алгоритмами Soundex*. Однако существует оригинальный алгоритм Soundex, имеющий несколько вариантов. Его схема выглядит так.

1. Преобразуем каждый индексируемый термин в четырехсимвольную сокращенную форму. По этим сокращенным формам строим инвертированный индекс для поиска исходных терминов; назовем его soundex-индексом.

2. Делаем то же самое для терминов запроса.
3. Если поступает запрос на сравнение строк по звучанию, выполняем поиск по индексу Soundex.

Варианты реализации алгоритмов Soundex связаны с методом преобразования терминов в четырехсимвольное представление. В результате применения наиболее распространенного метода возникает четырехсимвольный код, в котором первый символ — это буква алфавита, а остальные три — цифры от 0 до 9.

1. Запишем первую букву термина.
2. Заменим все буквы A, E, I, O, U, H, W и Y нулем '0'.
3. Заменим буквы цифрами следующим образом.
4. B, F, P, V на 1.
5. C, G, J, K, Q, S, X, Z на 2.
6. D, T на 3.
7. I на 4.
8. M, N на 5.
9. R на 6.
10. Циклически удалим одну из каждой пары соседних одинаковых цифр.
11. Из полученной строки удалим все нули. Дополним результат замыкающими нулями и возвратим первые четыре позиции, представляющие собой букву, за которой следуют три цифры.

В качестве примера укажем, что слово `Hermann` отображается в код H655. Получив запрос (скажем, `herman`), мы вычисляем его soundex-код, а затем находим в лексиконе все термины, соответствующие этому soundex-коду в soundex-индексе. Теперь выполним поиск результирующего запроса в стандартном обратном индексе.

Этот алгоритм основан на нескольких наблюдениях: 1) гласные могут заменять друг друга в транскрипции имен; 2) согласные с подобными звуками (например, `D` и `T`) относятся к одному и тому же классу эквивалентности. Благодаря этому похожие имена часто имеют одинаковые soundex-коды. Несмотря на то что эти правила во многих случаях выполняются, особенно если речь идет о европейских языках, они зависят от системы письменности. Например, китайские имена могут быть записаны как в транскрипции Вейда–Джайлса (Wade–Giles), так и в транскрипции пиньинь (Pinyin). Несмотря на то что алгоритмы Soundex с некоторыми изменениями работают с обеими транскрипциями — например, термин `hs` в транскрипции Вейда–Джайлса и термин `x` в транскрипции пиньинь переводятся в один и тот же код 2, — в некоторых случаях происходит сбой: например, символ `j` в транскрипции Вейда–Джайлса и символ `r` в транскрипции пиньинь переводятся в разные коды.

? ***Упражнение 3.14.*** Укажите два имени собственных, которые записываются поразному, но имеют одинаковые Soundex-коды.

Упражнение 3.15. Укажите два фонетически схожих имени собственных, имеющих разные soundex-коды.

3.5. Библиография и рекомендации для дальнейшего чтения

Исчерпывающим источником информации о деревьях поиска, включая B-деревья и их использование для поиска по словарям, является монография Кнута (Knuth, 1997).[6]

Одно из первых полных описаний перестановочного индекса приведено в работе Гарфилда (Garfield, 1976). Метод решения проблемы резкого роста объема памяти, необходимого для хранения перестановочных индексов, изложен в работе Феррагины и Вентурини (Ferragina and Venturinin, 2007).

Одно из наиболее ранних формальных описаний методов исправления орфографических ошибок содержится в работе Дамеро (Damerau, 1964). Понятие расстояния редактирования предложено Левенштейном (В.И. Левенштейн, 1964), алгоритм, представленный на рис. 3.5, изобретен Вагнером и Фишером (Wagner and Fischer, 1974). Варианты методов, основанных на расстоянии редактирования, предложены Петерсоном (Peterson, 1980) и Кукичем (Kukich, 1992). Очень подробное описание этих методов приведено в работе Цобеля и Дарта (Zobel and Dart, 1995), показавших, что индексирование с помощью *k*-грамм является очень эффективным способом выявления возможных несовпадений, но для идентификации наиболее вероятных ошибок его следует сочетать с более тонкими методами, таким как метод, основанный на расстоянии редактирования. Стандартным справочником по алгоритмам работы со строками, например по алгоритмам, использующим расстояние редактирования, является монография Гасфилда (Gusfield, 1997).[7]

Вероятностные модели (модели “зашумленных каналов”) для исправления орфографических ошибок впервые были предложены Керниганом и соавторами (Kernigan et al., 1990), а в дальнейшем были усовершенствованы Брилем и Муром (Brill and Moore, 2000), а также Тутановой и Муром (Toutanova and Moore, 2002). В этих моделях ошибочный запрос рассматривается как вероятностное искажение правильного запроса. Эти модели основаны на таком же математическом аппарате, как и модели языка, описанные в главе 12. Кроме того, они позволяют включать в модель фонетическое сходство и близость на клавиатуре, а также могут использовать данные о реальных орфографических ошибках пользователей. Многие считают эти алгоритмы наиболее совершенными на данный момент. Кучержан и Бриль (Cucerzan and Brill, 2004) продемонстрировали, как можно расширить эти алгоритмы для настройки моделей исправления орфографических ошибок на основе переформулировок запросов в логах поисковых систем.

Изобретение алгоритма Soundex приписывается Маргарет К. Оделл и Роберту С. Русселли (Margaret K. Odell and Robert C. Russelli), получившим американские патенты в 1918 и 1922 годах соответственно. Вариант метода, описанный в книге, был предложен Берном и Фордом (Bourne and Ford, 1961). Цобель и Дарт (Zobel and Dart, 1996) сравнили несколько алгоритмов для сравнения строк по звучанию и выяснили, что алгоритм Soundex плохо подходит для решения общей задачи исправления орфографических ошибок, а другие алгоритмы, основанные на фонетической схожести произношения терминов, работают хорошо.

---

[6] Имеется русский перевод: Кнут Д., Искусство программирования. В 3-х т. — М.: Издательский дом Вильямс, 2000

[7] Гасфилд Д. Строки, деревья и последовательности в алгоритмах. — СПб.: Невский диалект, 2003.

Глава 4

Построение индекса

Из настоящей главы вы узнаете, как создать инвертированный индекс. Этот процесс называется *построением индекса* (index construction) или *индексированием* (indexing). Процесс или машина, выполняющая такую работу, называется *индексатором* (indexer). Разработка алгоритмов индексирования ограничена возможностями аппаратного обеспечения. По этой причине мы начнем эту главу с обзора основ компьютерного аппаратного обеспечения, имеющего отношение к индексированию. После этого мы опишем блочное индексирование, основанное на сортировке (blocked sort-based indexing) (раздел 4.2), — эффективный одномашинный алгоритм для работы со статическими коллекциями. Его можно рассматривать как масштабируемый вариант простого алгоритма индексирования, основанного на сортировке (см. главу 1). В разделе 4.3 описывается однопроходное индексирование в оперативной памяти (single-pass in-memory indexing). Этот алгоритм обладает еще более широкими возможностями масштабирования, поскольку не предусматривает хранения лексикона в памяти. Для поиска по очень большим коллекциям — например, в сети веб — индексирование должно быть распределено по кластерам, состоящим из сотен или тысяч компьютеров. Эта проблема обсуждается в разделе 4.4. Для работы с коллекциями, подверженными частым изменениям, необходимо *динамическое индексирование*, описанное в разделе 4.5. Оно обеспечивает немедленное отражение в индексе изменений, произошедших в коллекции. В заключение мы рассмотрим несколько более сложных вопросов, возникающих в связи с индексами, например безопасность и индексы для поиска с ранжированием (раздел 4.6).

Построение индекса связано с несколькими темами, изложенными в других главах. Для индексирования необходим исходный текст, но документы кодируются многими способами (см. главу 2). Индексатор читает и пишет, а также сжимает и распаковывает как промежуточные файлы индекса, так и окончательный индекс (глава 5). При вебпоиске документы хранятся не в локальной файловой системе, а должны загружаться специальным роботом (глава 20). В корпоративных системах большинство документов инкапсулированы в разных информационных системах, почтовых приложениях и базах данных. Соответствующие примеры описаны в разделе 4.7. Несмотря на то что большинство этих приложений доступны по протоколу http, обычно более эффективными оказываются их собственные интерфейсы прикладного программирования (Application Programming Interfaces — API). Читателям следует знать, что построение подсистем, поставляющих исходный текст процессу индексирования, само по себе может оказаться сложной проблемой.

4.1. Основы аппаратного обеспечения

При построении систем информационного поиска многие решения зависят от характеристик компьютерного аппаратного обеспечения, на котором будет развернута система. По этой причине мы начинаем изложение с краткого обзора компьютерного

аппаратного обеспечения. Рабочие характеристики типичных систем в 2007 году приведены в табл. 4.1.

Таблица 4.1. Параметры типичной системы в 2007 году. Время позиционирования характеризует скорость перемещения головки жесткого диска в новую позицию. Время передачи байта — это скорость пересылки данных с диска в оперативную память, когда головка диска находится в правильной позиции.

Символ	*Показатель*	*Значение*
s	Среднее время позиционирования	5 мс = 5×10^{-3} с
b	Среднее время передачи байта;	0,02 мкс = 2×10^{-8} с
	тактовая частота процессора	10^9 $с^{-1}$
P	Низкоуровневая операция (например, сравнить слова и поменять местами);	0,01 мкс = 10^{-8} с
	объем основной памяти;	Несколько гигабайтов
	объем дисков	1 Тбайт или больше

Список характеристик аппаратного обеспечения, влияющих на архитектуру систем информационного поиска, приведен ниже.

- Доступ к данным, находящимся в памяти, осуществляется намного быстрее, чем доступ к данным на диске. Для доступа к байту, записанному в памяти, достаточно нескольких тактов процессора (возможно, 5×10^{-9} с), но загрузка байта с диска выполняется намного медленнее (около 2×10^{-8} с). Следовательно, как можно больше данных желательно хранить в памяти, особенно данных, к которым часто требуется доступ. Метод, при котором часто используемые данные хранятся в основной памяти, называется *кэшированием* (caching).
- При считывании данных с диска и записи данных на диск возникает интервал времени, в течение которого головка диска перемещается на участок памяти, где хранятся данные. Это время называется *временем позиционирования* (seek time). Для типичных дисков оно составляет около 5 мс. Во время позиционирования головки никакие данные не передаются. По этой причине для максимального ускорения передачи данных блоки, которые должны быть считаны вместе, следует хранить на диске в соседних участках. Например, с учетом параметров, перечисленных в табл. 4.1, загрузка 10 Мбайт данных с диска в память может быть выполнена за 0,2 с, если данные хранятся одним неразрывным блоком. В то же время, если эти данные разбросаны по ста не связанным друг с другом фрагментам, на их загрузку потребуется $0{,}2 + 100 \times (5 \times 10^{-3})$ мс = 0,7 с, поскольку при этом позиционирование головки будет выполнено сто раз.
- Операционные системы обычно считывают и записывают данные целыми блоками. Таким образом, считывание одного байта с диска может продолжаться столько же времени, сколько и считывание целого блока. Размеры блока обычно равны 8, 16, 32 и 64 Кбайт. Часть оперативной памяти, в которую записываются блоки и из которой они считываются, называется *буфером* (buffer).
- Передача данных с диска в память обрабатывается системной шиной, а не процессором. Это значит, что в момент выполнения дисковых операций ввода-

вывода процессор может обрабатывать данные. Мы можем использовать этот факт для ускорения передачи данных, если будем хранить сжатые данные на диске. Если распаковка данных выполняется с помощью эффективного алгоритма, то общее время считывания и распаковки сжатых данных обычно меньше, чем время считывания несжатых данных.

- Оперативная память серверов, используемых в системах информационного поиска, обычно составляет несколько гигабайтов, иногда — десятки гигабайтов. Объем доступного пространства на диске, как правило, на несколько порядков больше.

4.2. Блочное индексирование, основанное на сортировке

Основные этапы построения некоординатного индекса представлены на рис. 1.4. Сначала выполняется проход по коллекции и сбор всех пар “термин–docID”. Затем эти пары сортируются, причем в качестве главного ключа используется термин, а в качестве вторичного ключа — идентификатор документа docID. На заключительном этапе идентификаторы документа для каждого термина заносятся в инвертированный список и вычисляются статистические характеристики, например частота термина в документе (TF) и документная частота (DF). Для небольших коллекций все это можно проделать в оперативной памяти. В этой главе описываются методы, предназначенные для обработки крупных коллекций, которые требуют использования вторичной памяти.

Для повышения эффективности индексирования представим термины в виде идентификаторов терминов termID (вместо строк, как показано на рис. 1.4), где каждый идентификатор *termID* представляет собой уникальный порядковый номер. Отображение терминов в идентификаторы терминов можно осуществить “на лету” в момент обработки коллекции или, в двухпроходных методах, на первом проходе скомпилировать лексикон, а на втором проходе — создать инвертированный индекс. В этой главе описаны только однопроходные алгоритмы создания индекса. Ссылки на работы, посвященные многопроходным методам, которые являются предпочтительными в некоторых приложениях, например, если память на диске ограничена, приведены в разделе 4.7.

В качестве примера в этой главе будем использовать коллекцию *Reuters–RCV1*, размер которой составляет примерно 1 Гбайт. Она состоит приблизительно из 800 тысяч документов новостной ленты агентства Рейтер на протяжении одного года — с 20 августа 1996 года по 19 августа 1997 года. На рис. 4.1 показан типичный документ. Следует отметить, что в этой книге мы игнорируем мультимедийную информацию и основное внимание уделяем тексту. Коллекция Reuter–RCV1 охватывает широкий диапазон международных тем, включая политику, бизнес, спорт и (как в приведенном примере) науку. Некоторые ключевые характеристики этой коллекции приведены в табл. 4.2.

Объем коллекции Reuters–RCV1 — 100 миллионов лексем. Следовательно, для создания множества всех пар “termID–docID” для этой коллекции, использующей по 4 байт на идентификаторы termID и docID, требуется 0,8 Гбайт памяти. В настоящее время типичные коллекции на один и даже на два порядка больше, чем коллекция Reuters–RCV1. Можно легко увидеть, что попытка отсортировать пары “termID–docID” в оперативной памяти для такой коллекции провалится даже на мощных современных компьютерах.

Таблица 4.2. Характеристики коллекции Reuters–RTV1. Значения округлены для удобства вычислений, которые приводятся в данной книге. Неокругленные значения таковы: 806 791 документ, 222 лексемы на документ, 391 523 (разных) термина, 6,04 байт на лексему с пробелами и знаками пунктуации, 4,5 байт на лексему без пробелов и знаков пунктуации, 7,5 байт на термин и 96 969 056 лексем. Числа в этой таблице соответствуют третьей строке табл. 5.1 ("без учета регистра").

Символ	*Показатель*	*Значение*
N	Документы	800 000
L_{ave}	Среднее количество лексем в документе	200
M	Термины;	400 000
	среднее количество байтов в лексеме (включая пробелы и знаки пунктуации);	6
	среднее количество байтов в лексеме (без пробелов и знаков пунктуации);	4,5
	среднее количество байтов в термине;	7,5
T	лексемы	100 000 000

Рис. 4.1. Новостное сообщение агентства Reuters

Если размер промежуточных файлов индексирования сравним с размером доступной оперативной памяти, то проблему можно решить с помощью методов сжатия, описанных в главе 5. Однако инвертированные файлы для многих больших коллекций не помещаются в память даже после сжатия.

Если объема оперативной памяти недостаточно, необходим *алгоритм внешней сортировки* (external sorting algorithm), т.е. алгоритм, использующий диск. Для того чтобы достичь приемлемой скорости, такой алгоритм должен минимизировать количество случайных перемещений головки по диску во время сортировки — последовательное считывание данных выполняется намного быстрее (см. раздел 4.1). Одним из решений этой проблемы является *алгоритм блочного индексирования, основанного на сортировке*

(blocked sort-based indexing), или *BSBI*, продемонстрированный на рис. 4.2. Алгоритм BSBI 1) сегментирует коллекцию на равные части; 2) сортирует пары "termID–docID" каждой части в памяти, 3) сохраняет промежуточные отсортированные результаты на диске и 4) объединяет все промежуточные результаты в окончательный индекс.

```
BSBIndexConstruction()
1    n ← 0
2    while (не просмотрены все документы)
3    do n ← n + 1
4       block ← ParseNextBlock()
5       BSBI-Invert(block)
6       WriteBlockToDisk(block, f_n)
7    MergeBlocks(f_1, ..., f_n; f_merged)
```

Рис. 4.2. Блочное индексирование, основанное на сортировке. Этот алгоритм хранит инвертированные блоки в файлах f_1, ..., f_n, а объединенный индекс — в файле f_{merged}

Алгоритм формирует по документам пары "termID–docID" и накапливает их в памяти, пока не будет заполнен блок фиксированного размера (функция `ParseNextBlock` на рис. 4.2). Размер блока подбирается так, чтобы он помещался в памяти и допускал быструю сортировку. Затем блок инвертируется и записывается на диск. *Инвертирование* (inversion) выполняется в два этапа. Во-первых, сортируются пары "termID–docID". Во-вторых, все пары "termID–docID" с одинаковыми идентификаторами termID формируют инвертированный список, а в качестве *словопозиции* (posting) выступает только идентификатор документа docID. Результат, представляющий собой инвертированный индекс для считанного блока, записывается на диск. Применяя этот алгоритм к коллекции Reuters–RTV1, и, предполагая, что мы можем разместить в оперативной памяти 10 миллионов пар "termID–docID", мы получаем десять блоков, каждый из которых является обратным индексом для одной из частей коллекции.

На заключительном этапе алгоритм одновременно выполняет слияние десяти блоков в один большой объединенный индекс. Пример двух блоков представлен на рис. 4.3, где i-й документ в коллекции обозначен как d_i. Для слияния все десять блоков открываются одновременно, для считывания используются небольшие буферы чтения, а для записи окончательного объединенного индекса используется буфера записи. На каждой итерации мы выбираем наименьший идентификатор termID, не обработанный до текущего момента. Для этого используется либо ***очередь с приоритетами***, либо аналогичная структура данных. Все инвертированные списки для данного идентификатора termID считываются и объединяются, а затем объединенный список записывается обратно на диск. Каждый буфер чтения по мере необходимости снова заполняется очередной порцией данных из файла.

Насколько затратным является метод BSBI? Его временн*а*я сложность равна $\Theta(T \log T)$, поскольку наиболее сложным этапом является сортировка, а количество сортируемых элементов ограничено числом T (количеством пар "termID–docID"). Однако главный вклад в реальное время индексирования обычно вносят парсирование докумен-

тов (функция `ParseNextBlock`) и окончательное слияние (функция `MergeBlocks`).[1] В упражнении 4.6 предлагается вычислить общее время, затраченное на создание индекса для коллекции RCV1, включая инвертирование блоков и их запись на диск.

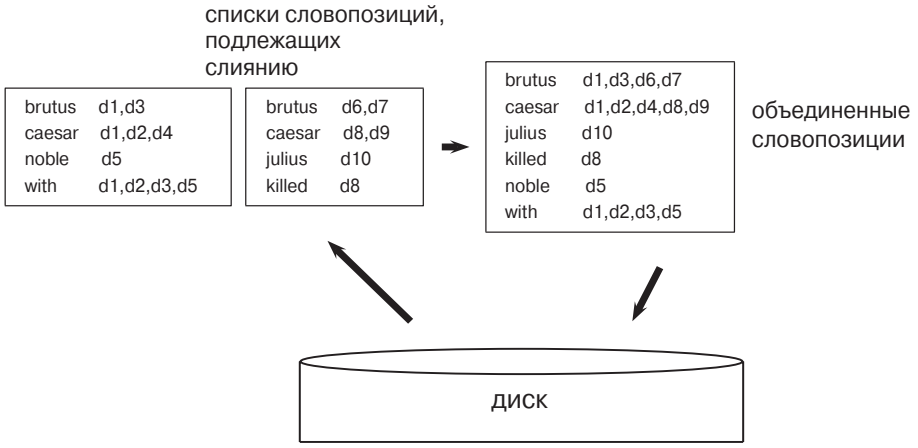


Рис. 4.3. Слияние в алгоритме блочного индексирования, основанного на сортировке. Два блока ("инвертированные списки, подлежащие слиянию") загружаются с диска в память, объединяются там ("объединенные инвертированные списки") и записываются обратно на диск. Для большей ясности вместо идентификаторов termID показаны термины

Обратите внимание на то, что коллекция Reuters–RCV1 не считается большой в наше время, когда один или несколько гигабайтов оперативной памяти являются стандартом для персональных компьютеров. При соответствующем сжатии (глава 5), мы могли бы создать инвертированный индекс для коллекции RCV1, используя лишь оперативную память и даже не имея мощного сервера. Методы, которые мы описали, необходимы для коллекций, размер которых на несколько порядков превышает размер коллекции RCV1.

? ***Упражнение 4.1.*** Допустим, что нам необходимо провести $T\log_2 T$ сравнений, где T — количество пар "termID–docID", и при каждом сравнении выполняется две установки головки диска. Сколько времени понадобится для создания индекса для коллекции Reuters-RCV1, если вместо памяти используется диск, а алгоритм сортировки является неоптимальным (т.е. не алгоритмом внешней сортировки)? Используйте параметры системы, указанные в табл. 4.1.

Упражнение 4.2 **[*].** Как "на лету" создать словарь в алгоритме блочного индексирования, основанного на сортировке, чтобы избежать дополнительного прохода по данным?

[1] Однако при большом размере блока фазу сортировки нельзя сбрасывать со счетов, тем более что в алгоритме BSBI сортируются словопозиции, а не термины. — *Примеч. ред.*

4.3. Однопроходное индексирование в оперативной памяти

Блочное индексирование, основанное на сортировке, обладает превосходными возможностями для масштабирования, но для его реализации каждому термину необходимо поставить в соответствие идентификатор termID. Для очень больших коллекций необходимая структура данных может не поместиться в памяти.[2] Более эффективным с точки зрения масштабирования является *однопроходное индексирование в памяти* (single-pass in-memory indexing), или алгоритм *SPIMI*. Этот алгоритм использует термины, а не их идентификаторы, записывает словарь каждого блока на диск, а затем для нового блока начинает создавать новый словарь. При наличии достаточного объема памяти на диске с помощью алгоритма SPIMI можно проиндексировать коллекцию любого размера.

Алгоритм SPIMI продемонстрирован на рис. 4.4. Часть алгоритма, выполняющая парсирование документов и превращающая их в поток пар “термин–docID”, которые мы в данном контексте будем называть *лексемами* (tokens), на этом рисунке пропущена. Функция `SPIMI-Invert` многократно применяется к потокам лексем, пока не будет обработана вся коллекция.

```
SPIMI-Invert(token_stream)
1    output_file ← NewFile()
2    dictionary ← NewHash()
3    while (есть свободная память)
4    do token ← next(token_stream)
5       if term(token) ∉ dictionary
6          then postings_list = AddToDictionary(dictionary, term(token))
7          else postings_list = GetPostingsList(dictionary, term(token))
8       if full(postings_list)
9         then postings_list = DoublePostingsList(dictionary, term(token))
10      AddToPostingsList(postings_list, docID(token))
11   sorted_terms ← SortTerms(dictionary)
12   WriteBlockToDisk(sorted_terms, dictionary, output_file)
13   return output_file
```

Рис. 4.4. Инверсия блока с помощью однопроходного индексирования в оперативной памяти

Лексемы обрабатываются по очереди (строка 4). Когда термин встречается впервые, он добавляется в словарь (лучшей реализацией для словаря является хеш) и создается новый инвертированный список (строка 6). Вызов в строке 7 возвращает этот инвертированный список для последующих появлений данного термина.

Разница между алгоритмами BSBI и SPIMI заключается в том, что алгоритм SPIMI добавляет записи непосредственно в инвертированный список (строка 10). Вместо того чтобы коллекционировать все пары “termed–docID”, а затем их сортировать (как это де-

[2] Строго говоря, этот недостаток (необходимость глобального словаря) является совершенно независимым от главной особенности алгоритма SPIMI (отделение словопозиций от терминов в памяти), описываемой ниже, и вполне может быть преодолен независимо. — *Примеч. ред.*

лает алгоритм BSBI), в алгоритме SPIMI каждый инвертированный список является динамическим (т.е. его размер по мере надобности возрастает) и немедленно становится доступным для хранения записей. У такого подхода есть два преимущества: он быстрее, поскольку не подразумевает сортировки, и экономит память, поскольку отслеживает связь термина с соответствующим инвертированным списком и хранить идентификаторы termID для этих списков нет необходимости. В результате блоки, к которым применяется функция `SPIMI-Invert`, могут быть намного больше, а процесс составления индекса в целом становится более эффективным.

Поскольку нам неизвестно, насколько большим будет инвертированный список для термина, когда мы встретим его впервые, сначала мы выделяем память для короткого инвертированного списка, а затем каждый раз удваиваем ее после заполнения (строки 8 и 9).[3] Это значит, что часть памяти будет потеряна и сэкономить память за счет игнорирования идентификаторов терминов в промежуточных структурах данных не удастся. Однако общие требования к памяти для динамически конструируемого индекса блока в алгоритме SPIMI ниже, чем в алгоритме BSBI.

После того как память будет исчерпана, мы записываем индекс блока (состоящий из словаря и инвертированных списков) на диск (строка 12). Однако перед этим термины необходимо отсортировать (строка 11), так как инвертированные списки должны быть записаны в лексикографическом порядке. Это облегчает выполнение заключительного этапа слияния. Если бы инвертированные списки каждого блока были записаны беспорядочно, что слияние блоков в ходе однократного последовательного перебора каждого блока выполнить не удалось бы.

При каждом вызове функция `SPIMI-Invert` записывает блок на диск, так же как и в алгоритме BSBI. На последнем этапе алгоритма SPIMI (соответствующем строке 7 на рис. 4.2 и не показанном на рис. 4.4) блоки объединяются в окончательный инвертированный индекс.

Помимо создания новой словарной структуры для каждого блока и исключения затратного этапа сортировки, алгоритм SPIMI имеет еще одну важную особенность: сжатие[4]. Если применить сжатие, то как словопозиции, так и термины словаря можно компактно хранить на диске. Сжатие увеличивает эффективность алгоритма еще больше, так как это позволяет обрабатывать более крупные блоки и экономить память для хранения отдельных блоков на диске. Этот аспект алгоритма подробно описан в литературе (см. раздел 4.7).

Временн*ая* сложность алгоритма SPIMI равна $\Theta(T)$, поскольку сортировка лексем не нужна и сложность всех операций не более чем линейно зависит от размера коллекции.[5]

---

[3] Строки 8 и 9 говорят о том, что алгоритм SPIMI предполагает использование вектора (один физически непрерывный отрезок памяти) в качестве структуры данных для хранения списков позиций в оперативной памяти. Однако односвязный список в качестве контейнера для позиций является не менее, если не более, эффективной структурой данных в данном алгоритме, а он не требует этих проверок и реаллокаций. — *Примеч. ред.*

[4] Это отличие также непринципиально отличает алгоритмы (см. предыдущее примечание). — *Примеч. ред.*

[5] Здесь термин “лексема” употребляется в смысле всех выделенных из входного потока лексем (равно числу словоупотреблений). Сортировка же словаря в строке 11 (SortTerms) все равно нужна, поэтому сложность алгоритма SPIMI равна k*OMEGA((T/k)^b), где k — число блоков, b — константа закона Хипса (обычно — 0.5–0.7–0.9, см. главу 5), связывающая число лексем в блоке и размер его лексикона, т.е. число разных терминов блока. — *Примеч. ред.*

4.4. Распределенное индексирование

Коллекции документов бывают такими большими, что на отдельном компьютере невозможно осуществить их эффективное индексирование. Это особенно актуально для сети веб; для создания веб-индекса разумного размера необходимы большие компьютерные *кластеры* [6]. По этой причине поисковые машины в сети веб для создания индексов используют алгоритмы *распределенного индексирования* (distributed indexing). Результатом этого процесса является *распределенный индекс* (distributed index), разделенный между несколькими машинами, — либо по терминам, либо по документам. В этом разделе мы опишем распределенное индексирование для создания индекса, разделенного по терминам. Большинство крупных поисковых машин предпочитают индексы, распределенные по документам (которые легко создать на основе индексов, распределенных по терминам). К этой теме мы еще вернемся в разделе 20.3.

Метод создания распределенного индекса, описанный в этой главе, основан на общей архитектуре распределенных вычислений *MapReduce*. Архитектура MapReduce разработана для крупных компьютерных кластеров. Кластер предназначен для решения крупных вычислительных задач на дешевых серийных компьютерах, или *узлах* (nodes), которые состоят из стандартных частей (процессор, память, диск), в отличие от суперкомпьютера, имеющего специализированное аппаратное обеспечение. Несмотря на то что в такой кластер входят сотни и тысячи машин, каждая из них в любой момент может выйти из строя. Следовательно, для надежного индексирования необходимо разделить работу на порции, которые легко распределить и в случае сбоя перераспределить. Процессом распределения и перераспределения задач среди отдельных рабочих узлов управляет *главный узел* (master node).

Фазы отображения (map) и свертки (reduce) алгоритма MapReduce разделяют вычислительную работу на *порции (chunk)*, с которыми стандартные машины могут справиться за короткое время. Некоторые этапы алгоритма MapReduce показаны на рис. 4.5, а пример применения к коллекции, состоящей из двух документов, — на рис. 4.6. На первом этапе входные данные, в нашем случае — коллекция веб-документов, разделяются на *n разделов* (splits), размер которых выбирается так, чтобы гарантировать равномерное (порции не должны быть слишком крупными) и эффективное (общее количество порций не должно быть слишком большим) распределение. Удобными размерами разделов для распределенного индексирования являются 16 и 64 Мбайт. Распределение разделов по компьютерам заранее не определено — эта задача возложена на главный узел, который должен ее решать постоянно. Когда компьютер заканчивает обработку одной части, ему назначается другой. Если компьютер выходит из строя или работает слишком медленно из-за аппаратных проблем, то соответствующая часть переназначается другому компьютеру.

Архитектура MapReduce разделяет большую вычислительную задачу на более мелкие части, манипулируя парами *“ключ–значение”*. При индексировании пара “ключ–значение” имеет вид (termID, docID). При распределенном индексировании преобразование терминов в идентификаторы терминов termID также является распределенным, а значит, представляет собой более сложную задачу, чем при индексировании на отдельной машине. Для этого достаточно просто поддерживать (возможно, предвычисленное) преобра-

[6] В этой главе под кластером подразумевается группа связанных друг с другом компьютеров, которые работают координированно. Эта трактовка отличается от интерпретации кластера как группы документов, имеющих близкую семантику (см. главы 16–18).

зование термина в termID для часто встречающихся терминов на всех узлах, а при работе с относительно редко встречающимися терминами использовать сами термины, а не их идентификаторы. Решение этой задачи здесь не приводится, и предполагается, что отображение "term → termID" правильно осуществляется во всех узлах.

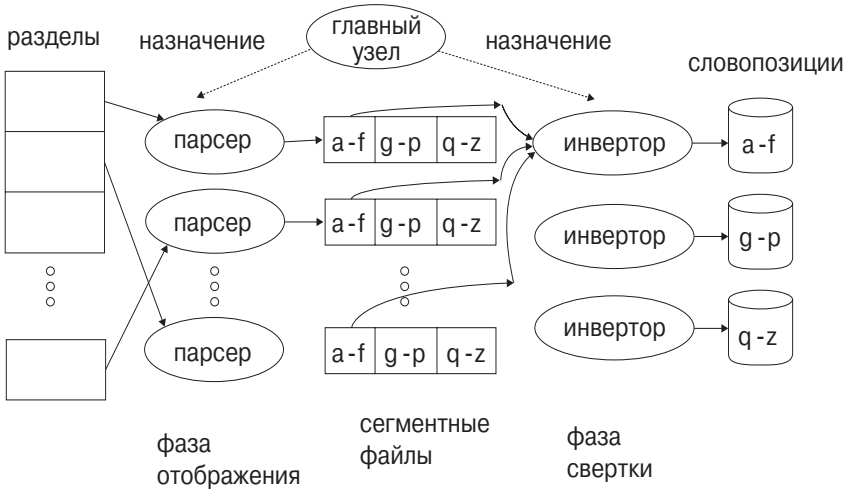


Рис. 4.5. Пример распределенного индексирования с помощью архитектуры MapReduce. Цитируется по работе Дина и Гемават (Dean and Ghemawat, 2004)

Схема функций отображения и свертки

map: input	→ list(*k*, *v*)
reduce: (*k*, list(*v*))	→ output

Создание экземпляра для построения индекса

map: web collection	→ list(termID, docID)
reduce: (⟨termID$_1$, list(docID)⟩, ... , ⟨termID$_2$, list(docID)⟩, ...)	→ (postings_list$_1$, postings_list$_2$, ...)

Экземпляр для построения индекса

map: d_1: `C died`. d$_1$: `C came, C c'ed`	→(⟨`C`, d_2⟩, ⟨`died`, d_2⟩, ⟨`C`, d_1⟩, ⟨`came`, d_1⟩, ⟨`C`, d_1⟩, ⟨`c'ed`, d_1⟩)
reduce: (⟨`C`, (d_2, d_1, d_1)⟩, ⟨`died`, (d_2)⟩, ⟨`came`, (d_1)⟩, ⟨`c'ed`, (d_1)⟩)	→(⟨`C`, (d_1:2, d_2:1)⟩, ⟨`died`, (d_2:1)⟩, ⟨`came`, (d_2:1)⟩, ⟨`c'ed`, (d_1:1)⟩)

Рис. 4.6. Функции отображения и свертки в архитектуре MapReduce. Функция map (отображение) создает список пар "ключ–значение". На этапе reduce (свертка) все значения, соответствующие ключу, собираются в один список. Затем этот список обрабатывается далее. На рисунке продемонстрированы экземпляры двух функций и пример создания индекса. Поскольку на фазе отображения документы обрабатываются в распределенном режиме, пары "termID–docID" изначально не обязательно должны быть упорядочены. Для большей ясности вместо идентификаторов терминов termID на рисунке приведены сами термины. Термин `Caesar` заменен аббревиатурой `C`, а термин `conquered` — аббревиатурой `c'ed`

Этап отображения (map phase) в архитектуре MapReduce состоит из отображения разделов входных данных в пары "ключ–значение". Эта задача ничем не отличается от

задачи парсирования документов, которая решается в рамках алгоритмов BSBI и SPIMI, поэтому мы называем компьютеры, задействованные на этапе отображения, *парсерами* (parser). Каждый парсер записывает свои результаты в локальные промежуточные файлы, которые называются *сегментными* (segment files) (рис. 4.5).

На *этапе свертки* (reduce phase) все значения, соответствующие заданному ключу, записываются рядом друг с другом, так, чтобы процесс их чтения и обработки выполнялся быстро. Это достигается благодаря разделению ключей на j диапазонов терминов и записи результата работы парсера (пар "ключ–значение") над каждым диапазоном в свой сегментный файл. На рис. 4.5 показано разделение терминов по первым буквам: a–f, g–p, q–z и $j = 3$. (Такое разделение выбрано по соображениям наглядности. В принципе диапазоны ключей не обязательно соответствуют смежным терминам или идентификаторам). Разделение терминов на диапазоны определяется человеком, управляющим системой индексирования (упражнение 4.10). После этого парсер записывает соответствующие сегментные файлы — по одному для каждого диапазона терминов. Таким образом, каждому диапазону терминов соответствуют r сегментных файлов, где r — количество парсеров. Например, на рис. 4.5 показаны три сегментных файла для раздела a–f. Им соответствуют три парсера, изображенные на этом рисунке.

Сбор всех значений (в данном случае — идентификаторов docID), соответствующих заданному ключу (в данном случае — идентификатору termID), в один список на этапе свертки возлагается на *инвертор* (inverter). Главный узел назначает каждый диапазон терминов разным инверторам и, как и в случае с парсерами, если инвертор вышел из строя или работает медленно, происходит переназначения диапазонов терминов. Каждый диапазон терминов (которому соответствуют r сегментных файлов, по одному на каждый парсер) обрабатывается отдельным инвертором. Здесь предполагается, что сегментные файлы имеют размер, подходящий для обработки на одной машине. Наконец, список значений, соответствующих каждому ключу, упорядочивается и записывается в окончательный отсортированный инвертированный список. (Обратите внимание на то, что словопозиции, изображенные на рис. 4.6, включают в себя частоту термина в документе (TF), в то время как в других разделах главы они представляют собой просто идентификатор docID без указания частоты термина). На рис. 4.5 показан поток данных для диапазона "a–f". На этом создание инвертированного индекса завершается.

Парсеры и инверторы — это не разные машины. Главный узел идентифицирует простаивающие машины и назначает им задания. Одна и та же машина на этапе отображения может играть роль парсера, а на этапе свертки — инвертора. Часто возникает необходимость параллельно с созданием индекса выполнять и другую работу, поэтому между моментами, когда компьютер играет роль парсера или инвертора, он может решать другие задачи.

Для минимизации времени записи перед тем, как инверторы свернут данные, каждый парсер записывает свой сегментный файл на свой *локальный* диск. На этапе свертки главный узел сообщает инвертору расположение соответствующих сегментных файлов (например, r сегментных файлов для раздела "a–f"). Для каждого сегментного файла требуется только одно последовательное считывание, поскольку все данные, предназначенные конкретному инвертору, записываются парсером в отдельный сегментный файл. Такой прием минимизирует объем сетевого трафика во время индексирования.

На рис. 4.6 продемонстрирована общая схема функций архитектуры MapReduce. Ввод и вывод часто представляют собой списки пар "ключ–значение", поэтому несколько заданий в рамках архитектуры MapReduce могут выполняться по порядку. Фактически этот

подход был разработан для системы индексирования поисковой машины Google в 2004 году. В этом разделе описывается только одна из пяти-десяти операций библиотеки MapReduce в этой системе индексирования. Другая операция библиотеки MapReduce преобразует индекс, разделенный по терминам, в индекс, разделенный по документам.

Архитектура MapReduce представляет собой надежный и концептуально простой инструмент для построения индексов в распределенной среде. Предоставляя возможность полуавтоматического разделения процесса построения индекса на более мелкие задачи, при наличии компьютерных кластеров достаточного размера этот алгоритм позволяет масштабировать сколь угодно большие коллекции.

? ***Упражнение 4.3.*** Допустим, что данные для построения индекса разделены на $n = 15$ частей, количество сегментов $r = 10$, а ключи разделены на $j = 3$ диапазонов терминов. Как долго будет выполняться создание индекса для коллекции Reuters–RCV1 в архитектуре MapReduce? При вычислении используйте данные из табл. 4.1.

4.5. Динамическое индексирование

До сих пор мы предполагали, что коллекция документов является статической. Это справедливо для коллекций, которые никогда не изменяются либо изменяются редко (например, Библия и собрание сочинений Шекспира). Однако большинство коллекций изменяются часто и при этом документы добавляются, удаляются или обновляются. Это значит, что в словарь необходимо добавлять новые термины, а инвертированные списки необходимо обновлять с учетом изменений.

Простейший способ решения этой задачи заключается в периодической перестройке индекса заново. Это хорошее решение, если частота изменений невелика, а задержка, связанная с тем, что новые документы не сразу появляются в поиске, вполне приемлема, причем в нашем распоряжении достаточно ресурсов, чтобы в процессе создания нового индекса сохранить доступ к старому.

Если новые документы необходимо добавлять быстро, то целесообразно поддерживать два индекса: крупный основной индекс и небольшой *вспомогательный индекс* (auxiliary index), в котором хранятся новые документы. Вспомогательный индекс хранится в памяти. Поиск выполняется по обоим индексам, а результаты объединяются. Удаленные документы хранятся в битовом векторе недействительных документов. Таким образом, перед возвращением результатов все удаленные документы можно отфильтровать. Документы обновляются с помощью удаления и повторной вставки.

Каждый раз, когда вспомогательный индекс становится слишком большим, мы объединяем его с основным индексом. Стоимость этой операции слияния зависит от того, как индекс хранится в файловой системе. Если каждый инвертированный список хранится в отдельном файле, то слияние сводится к расширению каждого списка из основного индекса соответствующим списком из вспомогательного индекса. В этой схеме вспомогательный индекс позволяет сократить количество позиционирований головки диска. Для отдельного обновления каждого документа требуется M_{ave} перемещений диска, где M_{ave} — средний размер лексикона документов из коллекции. При работе с вспомогательным индексом возникает только дополнительная нагрузка на диск при слиянии с основным индексом.

К сожалению, схема, в которой каждый инвертированный список хранится в отдельном файле, неосуществима, поскольку большинство файловых систем не могут эффективно работать с очень большим количеством файлов. Проще всего хранить индекс в одном большом файле, т.е. в виде конкатенации всех инвертированных списков. На практике часто используется компромиссное решение (раздел 4.7). Для простоты предположим, что индекс хранится в одном большом файле.

В данной схеме каждая словопозиция обрабатывается $\lfloor T/n \rfloor$ раз, поскольку мы обращаемся к ней в ходе каждого из $\lfloor T/n \rfloor$ слияний, где n — размер вспомогательного индекса, а T — общее количество позиций. Следовательно, общая временная сложность равна $\Theta(T^2/n)$. (Мы пренебрегли представлением терминов и учитывали только идентификаторы docID. При оценке временной сложности инвертированный список рассматривается просто как список идентификаторов *docID*.)

Введя $\log_2(T/n)$ индексов I_0, I_1, I_2, ... размерами $2^0 \times n$, $2^1 \times n$, $2^2 \times n$, ... соответственно, мы можем улучшить оценку временной сложности. Словопозиции "профильтровываются" через эту последовательность индексов, причем на каждом уровне обрабатываются только один раз. Эта схема называется *логарифмическим слиянием* (logarithmic merging) (рис. 4.7). Как и прежде, во вспомогательном индексе, хранящемся в памяти, накапливается до n позиций, которые мы назовем Z_0. Как только предел n превышен, $2^0 \times n$ записей в индексе Z_0 переносятся в новый индекс I_0, созданный на диске. В следующий раз, когда индекс Z_0 окажется заполненным, осуществляется его слияние с индексом I_0 и создается

```
LMergeAddToken(indexes, Z0, token)
 1  Z0 ← Merge(Z0, {token})
 2  if |Z0| = n
 3    then for i ← 0 to ∞
 4     do if Ii ∈ indexes
 5       then Zi+1 ← Merge(Ii, Zi)
 6            (Zi+1 — временный индекс на диске)
 7            indexes ← indexes – {Ii}
 8       else Ii ← Zi (Zi становится постоянным индексом Ii)
 9            indexes ← indexes ∪ {Ii}
10            Break
11  Z0 ← ∅
LogarithmicMerge()
 1  Z0 ← ∅ (Z0 — индекс, хранящийся в памяти)
 2  indexes ← ∅
 3  while true
 4  do LMergeAddToken(indexes, Z0, getNextToken())
```

Рис. 4.7. Логарифмическое слияние. Каждая лексема (termID,docID) сначала с помощью алгоритма `LMergeAddToken` *добавляется в индекс* Z_0*, хранящийся в памяти. Индекс* Z_0 *и множество* indexes *инициализируются алгоритмом* `LogarithmicMerge`

индекс Z_1, имеющий размер $2^1 \times n$. Индекс Z_1 либо хранится как I_1 (если индекса I_1 до этого момента не было), либо объединяется с индексом I_1, образуя индекс Z_2 (если индекс I_1 существовал), и т.д. Запрос на поиск направляется индексу Z_0, хранящемуся в памяти, и всем существующим в данный момент индексам I_i, записанным на диске, а результаты поиска объединяются. Читатели, знающие свойства биномиальной кучи [7], легко узнают эту структуру данных в структуре обратных индексов, созданных методом логарифмического слияния.

Временная сложность создания индекса равна $\Theta(T\log(T/n))$, поскольку каждая словопозиция на каждом из $\log(T/n)$ уровней обрабатывается только один раз. Этот выигрыш частично компенсируется замедлением обработки запроса; теперь нам необходимо производить слияние результатов из $\log(T/n)$ индексов, хотя раньше объединялись только два индекса (основной и вспомогательный). Как и в схеме со вспомогательным индексом, иногда приходится объединять очень большие индексы, что замедляет процесс поиска, однако это происходит реже, а размер объединяемых индексов в среднем меньше.

Работа с многочисленными индексами усложняет сбор статистики по коллекции. Например, она влияет на алгоритм исправления орфографических ошибок, описанный в разделе 3.3, который выбирает правильную альтернативу с наибольшим количеством совпадений. При наличии многочисленных индексов и битового вектора недействительных данных определение правильного количества совпадений термина уже не является простой задачей. Фактически все аспекты работы информационно-поисковых систем — поддержка индекса, обработка запросов, распределение и т.д. — при логарифмическом слиянии усложняются.

Из-за сложности динамического индексирования некоторые крупные поисковые машины используют стратегию повторного создания индекса “с нуля”. Они не применяют динамическое построение индекса. Вместо этого периодически создается совершенно новый индекс. После этого все запросы перенаправляются этому индексу, а старый индекс удаляется.

? ***Упражнение 4.4.*** Допустим, что $n = 2$ и $1 \le T \le 30$. Выполните пошаговую трассировку алгоритма, приведенного на рис. 4.7. Создайте таблицу, демонстрирующую для каждого момента времени, в который обработано $T = 2 \times k$ лексем ($1 \le k \le 15$), какой из индексов I_0, I_1, I_2 используется. Первые три строки этой таблицы приведены ниже.

	I_3	I_2	I_1	I_0
2	0	0	0	0
4	0	0	0	1
6	0	0	1	0

[7] См, например, книгу Кормена и др. (Cormen et al., 1990). Русский перевод: Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ. — М.: Издательский дом “Вильямс”, 2005. — *Примеч. ред.*

4.6. Другие типы индексов

В данной главе рассматривается процесс построения только некоординатных индексов. Помимо большего объема данных, который следует обработать, координатные индексы отличаются тем, что хранят тройки (termID, docID, (position1, position2,...)), а не пары (termID, docID), а лексемы и словопозиции, помимо идентификатора docID, содержат информацию о позиции термина. С учетом этих особенностей все алгоритмы. описанные в этой главе, можно применять и для координатных индексов.

В индексах, рассмотренных нами выше, инвертированные списки были упорядочены по идентификаторам docID. Как будет показано в главе 5, это можно с успехом использовать для сжатия индекса. Вместо идентификаторов docID мы можем сжимать меньшие *интервалы* (gaps) между идентификаторами docID, смягчив требования к размеру индекса. Однако при создании систем *поиска с ранжированием* (главы 6 и 7), а не *булева* поиска эта структура индекса оказывается неоптимальной. При поиске с ранжированием документы в инвертированном списке часто упорядочиваются по весу или влиянию, причем документы с максимальным весом оказываются первыми. При такой организации поиска просмотр инвертированных списков в ходе обработки запросов обычно прекращается тогда, когда веса становятся настолько малыми, что любые следующие документы можно заведомо считать слабо соответствующими запросу (глава 6). В индексе, упорядоченном по идентификаторам docID, новые документы всегда вставляются в конец инвертированных списков. В индексе, упорядоченном по влиянию (раздел 7.1.5), вставка может производиться в любое место, усложняя обновление инвертированного индекса.

Безопасность (security) — это важный аспект функционирования информационно-поисковых систем в корпорациях. Работники низших звеньев не должны иметь доступа к данным о зарплатах сотрудников, а уполномоченные менеджеры должны иметь возможность найти эти данные. Списки результатов не должны содержать документов, закрытых для доступа; сама информация о существовании этих документов может быть конфиденциальной.

Авторизация пользователя часто управляется с помощью *списков контроля доступа* (Access Control List — ACL).[8] Списки ACL могут взаимодействовать с информационно-поисковой системой путем связывания каждого документа с набором пользователей, имеющих к ним доступ (рис. 4.8), и обращения полученной матрицы “пользователь–документ”. Инвертированный индекс ACL для каждого пользователя содержит “инвертированный список” документов, к которым он имеет доступ, — список доступа пользователя. Однако такой индекс трудно поддерживать, если права доступа изменяются; мы уже обсуждали подобные сложности в контексте инкрементального индексирования в случае обычных инвертированных списков (см. раздел 6.5). Кроме того, если пользователь имеет доступ к большому подмножеству документов, то приходится обрабатывать очень длинные инвертированные списки. Права пользователя часто верифицируются непосредственно в ходе поиска информации в момент запроса, даже если это замедляет сам поиск.

---

[8] Аналогом этой постановки в веб-поиске может служить задача персонализированного и социального поиска, в которой идентифицированный пользователь может иметь индивидуальное ранжирование, использующее данные о страницах, которые она посещала сама или прочитала в “ленте друзей” той или иной социальной сети. — *Примеч. ред.*

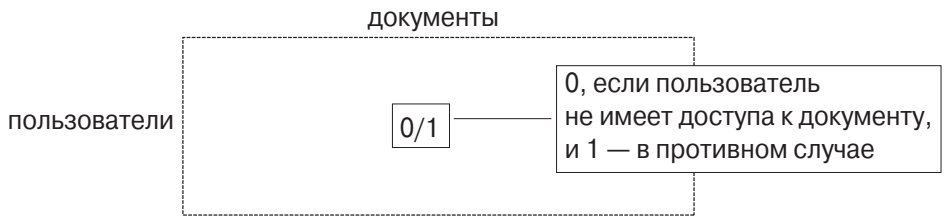


Рис. 4.8. Матрица "пользователь–документ" для списков контроля доступа. Элемент (i, j) равен единице, если пользователь i имеет доступ к документу j, и нулю, если не имеет. Во время обработки запроса список записей о праве доступа пользователя пересекается со списком результатов, возвращенным текстовой частью индекса

Индексы, используемые для хранения и поиска терминов (в противоположность документам), описаны в главе 3.

? ***Упражнение 4.5.*** Может ли исправление орфографических ошибок нарушить безопасность на уровне документов? Рассмотрите случай, когда исправление орфографических ошибок относится к документам, к которым пользователь доступа не имеет.

Упражнение 4.6. Общее время, затрачиваемое на создание индекса с помощью блочного индексирования, основанного на сортировке, продемонстрировано в табл. 4.3. Заполните столбец "Время" на основе параметров системы, указанных в табл. 4.1.

Таблица 4.3. Пять этапов построения индекса для коллекции Reuters–RCV1 с помощью блочного индексирования, основанного на сортировке. Номера строк соответствуют рис. 4.2

	Этап
1	Считывание коллекции (строка 4)
2	10 начальных сортировок 107 записей (строка 5)
3	Запись 10 блоков (строка 6)
4	Общее время передачи данных на диск для слияния (строка 7)
5	Время фактического слияния (строка 7)

Упражнение 4.7. Повторите упражнение 4.6 для более крупной коллекции, описанной в табл. 4.4. Выберите размер блока, соответствующий уровню современных технологий (помните, что блок должен без проблем помещаться в оперативной памяти). Сколько блоков нам потребуется?

Таблица 4.4. Статистические характеристики крупной коллекции

Символ	*Статистическая характеристика*	*Значение*
N	Количество документов	1 000 000 000
L_{ave}	Количество лексем в документе	1 000
M	Количество разных терминов	44 000 000

Упражнение 4.8. Предположим, что в нашем распоряжении находится коллекция среднего размера, индекс которой можно построить с помощью простого алгоритма индексирования в оперативной памяти, продемонстрированного на рис. 1.4. Сравните требования к размерам оперативной и дисковой памяти, а также к времени индексирования, которые предъявляют к этой коллекции простой алгоритм, описанный на рис. 1.4, и алгоритм блочного индексирования, основанный на сортировке.

Упражнение 4.9. Предположим, что компьютеры в архитектуре MapReduce имеют по 100 Гбайт дисковой памяти. Допустим также, что инвертированный список для термина the имеет размер 200 Гбайт. В таком случае архитектуру MapReduce невозможно применить для создания индекса. Как модифицировать подход MapReduce, чтобы решить эту проблему?

Упражнение 4.10. Для оптимального балансирования загрузки инверторы в архитектуре MapReduce должны получать сегментированные инвертированные списки примерно одинаковых размеров. Для новой коллекции распределение пар "ключ–значение" может оказаться заранее неизвестным. Как решить эту проблему?

Упражнение 4.11. Примените подход MapReduce для решения проблемы подсчета появления термина в наборе файлов. Опишите операции отображения и свертки для этой задачи. Запишите пример по строкам, как на рис. 4.6.

Упражнение 4.12. Мы утверждали (в конце раздела 4.5), что вспомогательный индекс может ухудшить качество статистических характеристик коллекции. Например, в методе взвешивания терминов ухудшается параметр *idf*, равный $\log(N/\mathrm{df}_i)$, где N — общее количество документов, а df_i — количество документов, содержащих термин i (раздел 6.2.1). Покажите, что даже маленький вспомогательный индекс может вызвать значительные ошибки при определении параметра idf, если он будет вычислен только по основному индексу. Рассмотрите редкий термин, который вдруг стал встречаться часто (например, Flossie в статье Tropical Storm Flossie).

4.7. Библиография и рекомендации для дальнейшего чтения

Глубокое исследование процесса построения индекса и дополнительных алгоритмов индексирования с разными требованиями к объемам оперативной и дисковой памяти, а также к временным ресурсам изложено в работе Уиттена и др. (Witten et al., 1999). В целом блочное индексирование, основанное на сортировке, хорошо соответствует всем трем требованиям. Однако, если основным критерием является экономия оперативной или дисковой памяти, более подходящими могут оказаться другие алгоритмы (Witten et al., 1999). Больше остальных на алгоритм "параллельного слияния, основанного на сортировке" (sort-based multi-way merge) похож алгоритм BSBI. Однако эти два алгоритма различаются структурой словаря и применением сжатия.

В работе Моффата и Белла (Moffat and Bell, 1995) показано, как создать индекс *in situ*, "на месте", т.е. чтобы размер дисковой памяти, используемой в процессе построения, был близок к размеру памяти, которую будет занимать окончательный индекс, а количество временных файлов было минимальным (см. также работу Хармана и Кандела (Harman and Candela, 1990)). Авторы указывают, что Леск (Lesk, 1988) и Сомогий (Somo-

giy, 1990) были среди первых, кто предложил использовать сортировку для построения индекса.

Метод SPIMI, описанный в разделе 4.3, предложен Хайнцем и Цобелем (Heinz and Zobel, 2003). Мы упростили некоторые аспекты алгоритма, включая сжатие и тот факт, что структура данных для каждого термина, кроме инвертированного списка, также содержит частоту документа и другую вспомогательную информацию. В качестве наиболее современного и глубокого исследования процесса конструирования индекса мы рекомендуем работы Хайнца и Цобеля (Heinz and Zobel, 2003) и Цобеля и Моффата (Zobel and Moffat, 2006). Другие алгоритмы с хорошими возможностями масштабирования по размеру лексикона требуют нескольких проходов по данным, например FAST-INV (Fox and Lee, 1991; Harman et al., 1992).

Архитектура MapReduce была предложена Дином и Гемаватом (Dean and Ghemawat, 2004). Открытая реализация архитектуры MapReduce доступна по адресу `http://lucene.apache.org/hadoop/`. Другие подходы к распределенному индексированию описаны в работах Рибейро-Нето и др. (Ribeiro-Neto et al., 1999) и Мельника и др. (Melnik et al., 2001). В качестве введения в распределенные информационно-поисковые системы можно использовать главы из книг Баеза-Йейтса и Рибейро-Нето (Baeza-Yates and Ribeiro-Neto, 1999) и Гроссмана и Фридера (Grossman and Frieder, 2004). См. также работу Каллана (Callan, 2000).

Свойства логарифмического слияния и результаты их сравнения с другими методами описаны в работах Лестера и др. (Lester et al., 2005), а также Бютчера и Кларка (Bьttcher and Clarke, 2005a). Одной из первых применений этих методов была система Lucene (`http://lucene.apache.org`). Другие методы динамического индексирования обсуждались в работе Бютчера и др. (Büttcher et al., 2006) и Лестера и др. (Lester, 2006). Во второй статье также обсуждается стратегия замены старого индекса индексом, созданным заново.

Хайнц и др. (Heinz et al., 2002) сравнил структуры данных для накопления лексикона в памяти. Бютчер и Кларке (Bьttcher and Clarke, 2005b) рассмотрели модели безопасности обычного инвертированного индекса и многочисленных пользователей. Подробная характеристика коллекции Reuters–RCV1 изложена в работе Льюиса и др. (Lewis et al., 2004). Эту коллекцию распространяет Национальный институт стандартов и технологии (National Institute of Standards and Technology — NIST) (см. `http://trec.nist.gov/data/reuters/reuters.html`).

Обзор компьютерного аппаратного обеспечения в контексте разработки информационно-поисковых систем содержится в работе Грасия-Молина и др. (Garcia-Molina et al., 1999).

Эффективный индексатор для корпоративного поиска должен иметь надежную и быструю связь со многими приложениями, в которых хранятся текстовые корпоративные данные, включая программы Microsoft Outlook, электронные таблицы Lotus компании IBM, базы данных, например Oracle и MySQL, системы управления контентом, такие как Open Text, а также системы планирования ресурсов, например SAP.

Глава 5

Сжатие индекса

В главе 1 были описаны словарь и инвертированный индекс — основные структуры данных в системах информационного поиска. В этой главе мы применим к словарю и инвертированному индексу методы сжатия, позволяющие повысить производительность работы информационно-поисковых систем.

Преимущества сжатия очевидны: требуется меньше дисковой памяти и, как будет показано далее, легко можно достичь коэффициента сжатия 1:4 и тем самым снизить стоимость хранения индекса на 75%.

Однако у сжатия есть два менее заметных преимущества. Во-первых, оно повышает эффективность использования кэша. Поисковые системы применяют некоторые части словарей и индекса чаще остальных. Например, если поместить в кэш инвертированный список часто используемого термина запроса *t*, то все вычисления, необходимые для поиска ответа на запрос, состоящий из одного термина *t*, можно выполнить в оперативной памяти. Применив сжатие, мы можем поместить в оперативную память больше информации. При обработке запросов, содержащих термин *t*, вместо выполнения затратных операций по перемещению головки диска можно обращаться к инвертированному списку, хранящемуся в оперативной памяти, и распаковывать его. Как будет показано ниже, существуют простые и эффективные методы распаковки данных, поэтому накладные расходы на распаковку инвертированного списка невелики. В результате мы можем существенно уменьшить время ответа информационно-поисковой системы. Поскольку оперативная память дороже, чем память на диске, основным стимулом для использования сжатия часто является ускорение работы за счет кэширования, а не за счет уменьшения требований к объему дискового пространства.

Во-вторых, еще менее заметное преимущество сжатия — более быстрая загрузка данных с диска в оперативную память. На современном аппаратном обеспечении эффективные алгоритмы распаковки работают настолько быстро, что общее время, затрачиваемое на передачу сжатой порции данных с диска, а затем на распаковку, обычно меньше, чем время, необходимое для передачи той же самой порции данных в несжатом виде. Например, загрузив инвертированный список намного меньшего размера, можно сократить время на выполнение операций ввода-вывода, даже с учетом дополнительного времени, требуемого для распаковки. Итак, в большинстве случаев системы информационного поиска, использующие сжатые инвертированные списки, работают быстрее, чем системы, в которых эти списки хранятся в развернутом виде.

Если основной целью сжатия является экономия дисковой памяти, то скорость сжатия не играет большой роли. Однако для более рационального использования кэша и более быстрой загрузки данных с диска в оперативную память скорость распаковки должна быть высокой. Алгоритмы сжатия, обсуждаемые в этой главе, весьма эффективны, а значит, позволяют использовать все преимущества сжатия индекса.

В данной главе *словопозиция* (posting) в инвертированном списке интерпретируется как идентификатор документа docID. Например, инвертированный список (6; 20, 45, 100),

где 6 — идентификатор термина из списка, содержит три координаты. Как указывалось в разделе 2.4.2, словопозиции в большинстве поисковых систем, как правило, содержат информацию о частоте и координатах термина. Однако мы рассмотрим лишь простые инвертированные списки, содержащие только идентификаторы документов. Библиография о сжатии информации о частотах и координатах приведена в разделе 5.4.

В начале главы приводятся статистические характеристики распределения сущностей, которые мы хотим сжать, — терминов и словопозиций в большой коллекции (раздел 5.2). Затем мы рассмотрим сжатие словаря с помощью метода, трактующего словарь как одну строку (dictionary-as-a-string), и хранения по блокам (раздел 5.2). В разделе 5.3 описываются два метода сжатия инвертированного файла: байтовые коды переменной длины (variable byte encoding) и γ-кодирование (γ-encoding).

5.1. Статистические характеристики терминов в информационном поиске

Как и в предыдущей главе, в качестве модельной коллекции мы будем использовать коллекцию Reuters–RCV1 (см. табл. 4.2). Для начала в табл. 5.1 приведем несколько статистических показателей, характеризующих термины и словопозиции в этой коллекции. Символ Δ% означает процент сжатия по отношению к предыдущей строке. Символ T% означает общий процент сжатия по отношению к исходным данным.

Таблица 5.1. Эффект предварительной обработки терминов, некоординатных словопозиций и лексем для коллекции Reuters–RCV1. Символ Δ% означает процент сжатия по отношению к предыдущей строке, за исключением строк "30 стоп-слов" и "150 стоп-слов", которые вычисляются по отношению к строке "Свертывание регистра". Символ T% означает общий процент сжатия по отношению к исходным данным. Стемминг осуществлялся с помощью алгоритма Портера (см. главу 2).

	Количество терминов			*Некоординатные словопозиции*			*Лексемы (= количество словоупотреблений в индексе)*		
	Кол-во	Δ%	T%	Кол-во	Δ%	T%	Кол-во	Δ%	T%
Исходные данные	484 494			109 971 179			197 879 290		
Без чисел	473 723	–2	–2	100 680 242	–8	–8	179 158 204	–9	–9
Свертывание регистра	391 523	–17	–19	96 969 056	–3	–12	179 158 204	–0	–9
30 стоп-слов	391 493	–0	–19	83 390 443	–14	–24	121 857 825	–31	–38
150 стоп-слов	391 373	–0	–19	67 001 847	–30	–39	94 516 599	–47	–52
Стемминг	322 383	–17	–33	63 812 300	–4	–42	94 516 599	–0	–52

В этой таблице приведено количество терминов на разных уровнях предварительной обработки (столбец 2). Количество терминов является основным фактором, влияющим на определение размера словаря. Количество некоординатных словопозиций (столбец 3) представляет собой индикатор ожидаемого размера некоординатного индекса коллекции. Ожидаемый размер координатного индекса связан с количеством координат, которые он должен закодировать (столбец 4).

В целом статистические показатели из табл. 5.1 свидетельствуют о том, что предварительная обработка сильно влияет на размер словаря и количество некоординатных словопозиций. Стемминг и свертывание регистра сокращают количество различных терминов на 17% каждый, а количество некоординатных словопозиций — на 4 и 3% соответственно. Обработка наиболее часто встречающихся слов также важна. *Правило тридцати* (rule of 30) утверждает, что 30 наиболее распространенных слов образуют 30% лексем в письменном тексте (в табл. 5.1 этот показатель равен 31%). Исключение из процесса индексирования 150 наиболее распространенных слов (так называемых стоп-слов, описанных в разделе 2.2.2) сокращает количество некоординатных словопозиций на 25–30%. Несмотря на то что список 150 стоп-слов сокращает количество словопозиций на четверть или более, эквивалентного уменьшения размера не наблюдается на сжатом индексе. Как будет показано далее, инвертированный список часто употребляемых слов после сжатия требует лишь несколько битов на запись после сжатия.

Показатели Δ, указанные в таблице, типичны для больших коллекций. Однако следует отметить, что у некоторых текстовых коллекций процентное сокращение может сильно отличаться. Например, в коллекции веб-страниц с высокой долей французского текста лемматизатор сокращает размер словаря намного больше, чем стеммер Портера, примененный к коллекции, состоящей только из английских документов, поскольку морфология французского языка намного богаче морфологии английского.[1]

Методы сжатия, описываемые в оставшейся части главы, являются методами *сжатия информации без потерь* (lossless compression), т.е. вся информация сохраняется. *Сжатие информации с потерями* (lossy compression) позволяет достичь более высокой степени сжатия за счет отбрасывания некоторых данных. Примерами сжатия информации с потерями являются свертывание регистра, стемминг и исключение стоп-слов. Аналогично модель векторного пространства (глава 6) и методы уменьшения размерности, такие как латентно-семантическое индексирование (глава 18), позволяют создать компактное представление, по которому невозможно восстановить исходную коллекцию.[2] Сжатие с потерей информации целесообразно, когда “потерянная” информация не используется системой поиска. Например, для веб-поиска характерны огромное количество документов, короткие запросы и пользователи, просматривающие лишь несколько первых страниц результата. Вследствие этого можно отбросить словопозиции для документов, расположенных в списке результатов слишком далеко. Таким образом, существуют сценарии поиска, в которых методы сжатия информации с потерями можно использовать без риска снижения эффективности.

Перед тем как описать методы сжатия словаря, оценим количество разных терминов M в коллекции. Иногда говорят, что языки имеют словарь определенного размера. Во втором издании словаря *Oxford English Dictionary* (OED) содержится более 600 тысяч слов. Однако словари большинства крупных коллекций намного больше, чем словарь

---

[1] Стемминг в целом более агрессивно приводит словоформы к одной основе, чем принятая для русского языка лемматизация, поэтому, хотя русская морфология ничуть не беднее французской, примерное соотношение числа форм и лемм в больших русских коллекциях с лемматизацией может не сильно отличаться от приведенного примера. — *Примеч. ред.*

[2] Как ни странно, в практике поисковых систем иногда возникают задачи восстановления текста документов только по индексу, например в условиях нехватки дискового пространства или в случаях аварии; для решения этой задачи необходимы алгоритмы эффективного восстановления упрощенного текста по индексу (**разындексации**). — *Примеч. ред.*

OED. Он не содержит фамилий большинства людей, географических названий, наименований продуктов или научных понятий, например названий генов. Эти имена необходимо включить в инвертированный индекс, чтобы пользователь мог их найти.

5.1.1. Закон Хипса: оценка количества терминов

Лучший способ оценить число M — использовать *закон Хипса* (Heaps' law), позволяющий определить размер словаря как функцию, зависящую от размера коллекции.

$$M = kT^b, \tag{5.1}$$

где T — количество лексем в коллекции. Типичные значения параметров k и b таковы: $30 \le k \le 100$ и $b \approx 0{,}5$. Обоснование для закона Хипса заключается в том, что простейшей зависимостью между размерами коллекции и словаря является линейная функция в логарифмической системе координат, а предположение о линейности на практике часто оказывается справедливым (как показано на рис. 5.1 для коллекции Reuters–RCV1). В данном случае при $T > 10^5 = 100\,000$ совпадение является отличным, причем $b = 0{,}49$ и $k = 44$. Например, для первых 1 000 020 лексем закон Хипса предсказывает 38 323 термина.

$$44 \times 1\,000\,020^{0{,}49} \approx 38\,323$$

На самом деле количество терминов равно 38 365, что очень близко к предсказанному значению.

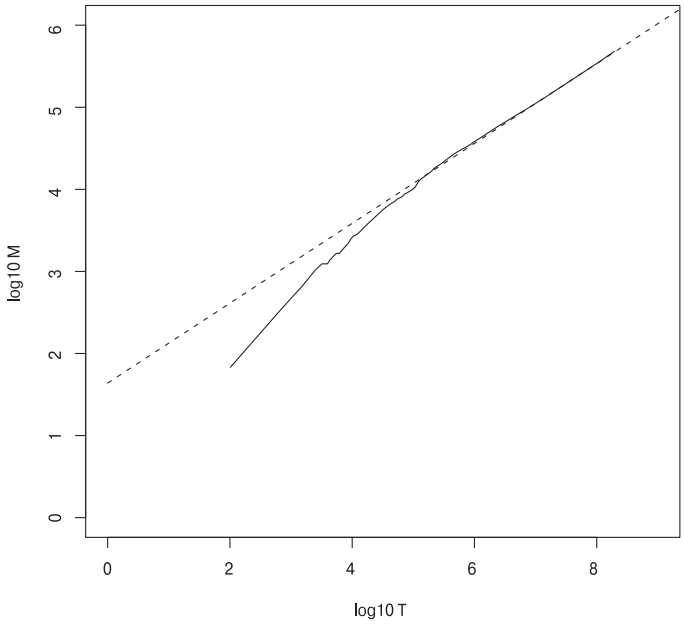


Рис. 5.1. Закон Хипса. Размер словаря M как функция от размера коллекции T (количества лексем) для коллекции Reuters–RCV1. Для этих данных пунктирная линия $\log_{10}M = 0{,}49 \times \log_{10}T + 1{,}64$ *построена методом наименьших квадратов. Следовательно,* $k = 10^{1{,}64} \approx 44$ и $b = 0{,}49$

Параметр k изменяется в довольно широких пределах, поскольку рост лексикона во многом зависит от природы коллекции и способа ее обработки.[3] Свертывание регистра и стемминг уменьшают скорость роста размера лексикона, в то время как включение чисел и орфографических ошибок увеличивает ее. Независимо от значений параметров для конкретной коллекции закон Хипса утверждает, что 1) при увеличении количества документов в коллекции размер лексикона продолжает возрастать, пока не достигнет максимального уровня, и 2) размер лексикона для крупных коллекций достаточно велик. Эти две гипотезы для больших коллекций доказаны эмпирически (раздел 5.4). Таким образом, сжатие словаря играет важную роль для повышения качества систем информационного поиска.

5.1.2. Закон Ципфа: моделирование распределения терминов

Определим также распределение терминов среди документов. Это поможет нам описать свойства алгоритмов сжатия инвертированных списков в разделе 5.3.

Для моделирования распределения терминов в коллекции широко используется *закон Ципфа* (Zipf's law). Он утверждает, что, если t_1 — наиболее распространенный термин в коллекции, t_2 — следующий по распространенности термин и т.д., то частота i-го по распространенности термина в коллекции cf_i пропорциональна $1/i$.

$$\mathrm{cf}_i \propto 1/i \tag{5.2}$$

Таким образом, если наиболее распространенный термин встречается cf_1 раз, второй по распространенности термин встречается вдвое реже, третий — втрое реже и т.д. Интуиция подсказывает, что по мере увеличения ранга частота уменьшается очень быстро. Равенство (5.2) — один из простейших способов формализации такого быстрого убывания; оно представляет собой вполне разумную модель.

Эквивалентно мы можем записать закон Ципфа как $\mathrm{cf}_i = ci^k$ или $\log \mathrm{cf}_i = \log c + k \log i$, где $k = -1$, а c — константа, которую мы определим в разделе 5.3.2. Следовательно, закон Ципфа является *степенным законом* с показателем $k = -1$. Другой степенной закон, характеризующий распределение ссылок между веб-страницами, описан в главе 19.

Логарифмический график, представленный на рис. 5.2, изображает зависимость частоты термина в коллекции Reuters–RCV1 от его ранга. На рисунке изображена также линия, имеющая угловой коэффициент, равный -1. Это соответствует функции Ципфа $\log \mathrm{cf}_i = \log c - \log i$. Соответствие данных закону Ципфа в таком случае не слишком хорошее, но этого вполне достаточно для моделирования распределения терминов при вычислениях в разделе 5.3.

? ***Упражнение 5.1* [*].** Допустим, что каждая словопозиция занимает одно машинное слово. Вычислите размер несжатого (некоординатного) индекса для разных методов выделения лексем по данным из табл. 5.1. Сравните эти числа с данными из табл. 5.6.

---

[3] Коллекция Reuters отличается от коллекций веб-документов весьма существенно. В частности, она составлена из проверенных орфографическим корректором текстов, написанных профессиональными, грамотными журналистами, не содержит искусственных, автоматически порождаемых лексем, например имен файлов или компонентов пути URL. Из-за опечаток и технических лексем значение b в веб-коллекциях существенно ближе к 1.0, чем к 0.5. — *Примеч. ред.*

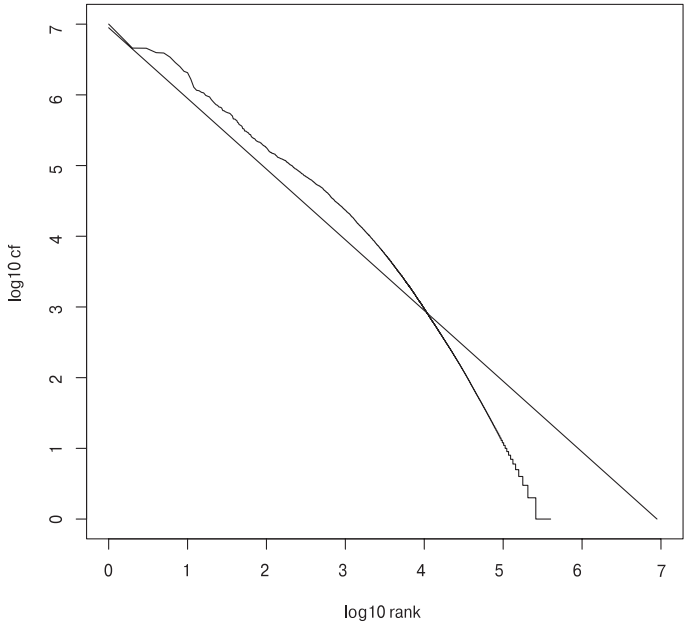


Рис. 5.2. Закон Ципфа для коллекции Reuters–RTV1. Частота представлена как функция, зависящая от частотного ранга терминов в коллекции. Прямая представляет собой распределение, предсказанное по закону Цифпа (с помощью метода взвешенных наименьших квадратов; пересечение — в точке 6,95)

5.2. Сжатие словаря

В этом разделе описывается ряд структур данных, обеспечивающих последовательно возрастающие коэффициенты сжатия. Как следует из табл. 5.1, словарь имеет малый размер по сравнению с инвертированным файлом. Зачем же его сжимать, если он занимает лишь небольшую часть памяти, которая необходима информационно-поисковой системе?

Одним из основных факторов, влияющих на время отклика информационно-поисковой системы, является количество перемещений головки диска, необходимое для обработки запроса. Если части словаря размещены на диске, то для обработки запросов потребуется намного больше перемещений. Следовательно, основная цель сжатия словаря — разместить его в оперативной памяти целиком, или по крайней мере бо́льшую его часть, чтобы обеспечить высокую производительность системы. Несмотря на то что лексиконы для очень крупных коллекций помещаются в памяти стандартных настольных компьютеров, во многих случаях это не так. Например, поисковый сервер большой корпорации, возможно, должен индексировать коллекцию в несколько терабайтов со сравнительно большим лексиконом из-за присутствия в коллекции документов на нескольких языках. Кроме того, иногда возникает необходимость разрабатывать поисковые системы на базе аппаратного обеспечения с ограниченными ресурсами, скажем, на основе мобильного телефона или бортовых компьютеров. Другая причина для экономии памяти

может заключаться в желании обеспечить быстрый запуск (старт) системы или необходимость совместного использования ресурсов наряду с другими приложениями. Например, поисковая система на вашем персональном компьютере должна функционировать одновременно с “жадным до памяти” текстовым процессором, который вы используете в это же время.

5.2.1. Словарь как строка

Для хранения словаря проще всего расположить его элементы в лексикографическом порядке и записать в массив записей фиксированной длины, как показано на рис. 5.3. Мы выделяем 20 байт для термина (поскольку в английском языке лишь немногие термины имеют больше 20 символов), 4 байт — для документной частоты и 4 байт — для указателя на инвертированный список. Четырехбайтные указатели позволяют адресовать 4 Гбайт памяти. Для крупных коллекций, например, в сети веб, для указателя требуется больше байтов. Поиск термина в массиве осуществляется с помощью метода бинарного поиска. В коллекции Reuters–RCV1 по этой схеме для хранения словаря потребуется $M \times (20 + 4 + 4) = 400\,000 \times 28 = 11{,}2$ Мбайт.

	Термин	*Частота документа*	*Указатель на инвертированный список*
	a	656 265	→
	aachen	65	→
	…	…	→
	zulu	221	→
Необходимая память, байт	20	4	4

Рис. 5.3. Хранение словаря в виде массива с элементами фиксированной длины

Хранить термины как элементы фиксированной длины нецелесообразно. Средняя длина термина в английском языке — примерно восемь символов (см. табл. 4.2), поэтому при такой схеме в среднем мы тратим впустую 12 символов. Кроме того, у нас нет возможности хранить термины длиннее 20 символов, такие как `hydrochlorofluorocarbons` или `supercalifragilisticexpialidocious`. Для решения этой проблемы можно хранить словарь в виде одной длинной строки, как показано на рис. 5.4. Указатель на следующий термин можно использовать для идентификации конца текущего термина. Как и прежде, мы ищем термины в структуре данных с помощью бинарного поиска в (теперь меньшей) таблице. По сравнению с предыдущей схемой эта схема экономит 60% памяти, поскольку мы выделяем на термин в среднем 12 байт вместо 20. Однако теперь нам требуется память для хранения указателей на термины. Указатели терминов адресуют $400\,000 \times 8 = 3{,}2 \times 10^6$ координат, поэтому нам необходимо $\log_2 3{,}2 \times 10^6 \approx 22$ бит на указатель, или 3 байт.

В новой схеме для хранения лексикона коллекции Reuters–RCV1 требуется $400\,000 \times (4 + 4 + 3 + 8) = 7{,}6$ Мбайт: по 4 байт — для хранения частоты и указателя на инвертированный список, 3 байт — для хранения указателя на термин и 8 байт в среднем — для хранения одного термина. Таким образом, требования к памяти снижаются на треть: с 11,2 до 7,6 Мбайт.

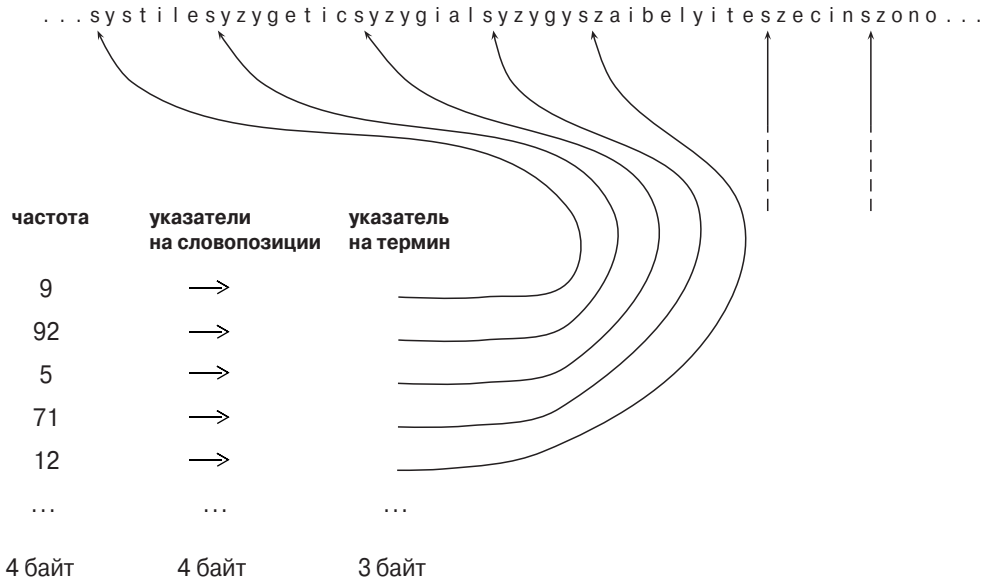


Рис. 5.4. Хранение словаря в виде одной строки. Указатели отмечают конец предыдущего термина и начало следующего. Например, первые три термина в данном примере — `sys-tile`, `syzygetic` и `syzygial`

5.2.2. Блочное хранение

Словарь можно сжать еще больше, сгруппировав термины в строке по блокам размером k и храня указатель только на первый термин каждого блока (рис. 5.5). Длина термина хранится в строке в виде дополнительного байта в начале термина. Таким образом, мы исключаем k - 1 указателей на термины, но требуем дополнительно k байт для хранения длины каждого термина. При $k = 4$ мы экономим $(k - 1) \times 3 = 9$ байт на каждом указателе на термин, но требуем дополнительно $k = 4$ байт для хранения длин терминов. Итак, общие требования к памяти для хранения лексикона коллекции Reuters–RTV1 уменьшаются на 5 байт на каждый из блоков, состоящих из четырех терминов. В целом этот объем составляет $400\,000 \times 1/4 \times 5 = 0{,}5$ Мбайт, т.е. размер требуемой памяти снижается до 7,1 Мбайт.

Увеличивая размер блока k, мы можем достичь еще большей степени сжатия. Однако существует противоречие между сжатием и скоростью поиска термина. На рис. 5.6 показаны бинарный поиск (двойные линии) и поиск по списку (одинарные линии) в словаре из восьми терминов. Пункт *а* соответствует бинарному поиску в несжатом словаре. В сжатом словаре мы сначала находим блок термина с помощью бинарного поиска, а затем определяем его координату в списке путем линейного поиска по блоку (п. *б*). Поиск в несжатом словаре в п. *а* в среднем занимает $(0 + 1 + 2 + 3 + 2 + 1 + 2 + 2)/8 \approx 1{,}6$ шагов, если считать, что каждый термин с одинаковой вероятностью появляется в запросе. Например, поиск двух терминов, `aid` и `box`, занимает три и два шага соответственно. В п. *б* при размере блока, равном $k = 4$, в среднем требуется $(0 + 1 + 2 + 3 + 4 + 1 + 2 + 3)/8 = 2$ шага, т.е. примерно на 25% больше. Например, для нахождения термина `den` выполняется один шаг бинарного поиска для определения блока и два шага внутри блока.

Рис. 5.5. Хранение словаря по четыре термина в блоке. Первый блок содержит термины `systile`, `syzygetic`, `syzygial` *и* `syzygy`, *имеющие длины, равные 7, 9, 8 и 6 символов соответственно. Перед каждым термином записывается байт, в котором кодируется его длина, указывающая, сколько байтов следует пропустить, чтобы достичь следующего термина*

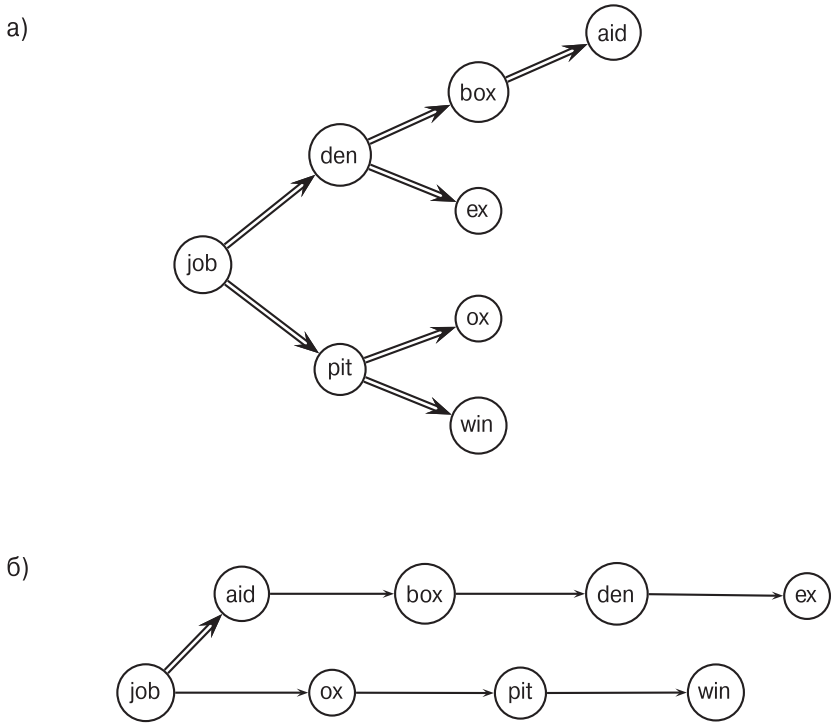


Рис. 5.6. Поиск в несжатом словаре (а) и в словаре, сжатом с помощью метода хранения по блокам при k = 4 (б)

Увеличивая параметр k, мы можем сделать размер сжатого словаря сколь угодно близким к минимуму, т.е. $400\,000 \times (4 + 4 + 1 + 8) = 6{,}8$ Мбайт, но поиск термина при больших значениях k становится недопустимо медленным.

Мы не использовали еще один источник избыточности при хранении словаря — тот факт, что последовательные элементы в списке, упорядоченном по алфавиту, имеют одинаковые префиксы. Это наблюдение приводит к *фронтальной упаковке* (front coding) (рис. 5.7). Мы сначала идентифицируем общий префикс для фрагмента списка терминов, а затем помечаем его специальным символом. Эксперимент показал, что для коллекции Reuters фронтальная упаковка экономит еще 1,2 Мбайт.

Один блок в схеме хранения по блокам ($k = 4$)

8automata**8**automate**9**automatic**10**automation

⇓

...дальнейшее сжатие с помощью фронтальной упаковки

8autimat*a1◊e2◊ic3◊ion

*Рис. 5.7. Фронтальная упаковка. Последовательность терминов с одинаковыми префиксами ("automat") кодируется с помощью пометки конца префикса символом * и его замены в последующих терминах символом ◊. Как и раньше, первый байт каждого элемента кодирует количество символов*

Другие схемы с еще более высокой степенью сжатия основываются на минимальном идеальном хешировании. При этом функция хеширования отображает M терминов в множество $[1, ..., M]$ без коллизий. Однако мы не можем применять идеальное хеширование неограниченно, так как каждый новый термин вызывает коллизии и, следовательно, требует создания новой идеальной функции хеширования. По этой причине данный метод невозможно применять в динамической среде.

При работе с очень большими текстовыми коллекциями на аппаратном обеспечении с ограниченными ресурсами даже при наилучшей схеме сжатия невозможно записать весь словарь в оперативную память. Если словарь необходимо разбить на части для записи на диск, то можно проиндексировать первый термин каждой страницы с помощью B-дерева. Для обработки большинства запросов поисковая система все равно должна обратиться к диску, чтобы достать инвертированные списки. Одно дополнительное позиционирование головки для поиска страницы термина в словаре, записанном на диске, является хотя и немаленькой, но приемлемой добавкой к времени обработки запроса.

Степени сжатия, достигнутые с помощью четырех структур данных для хранения словаря, указаны в табл. 5.2.

Таблица 5.2. Сжатие словаря для коллекции Reuters–RCV1

Структура данных	*Размер, Мбайт*
Словарь с фиксированным размером ячеек	11,2
Словарь с указателями на термины в строке	7,6
Словарь с указателями на термины в строке с блоками, k = 4	7,1
Словарь с указателями на термины в строке с блоками и фронтальной упаковкой	5,9

? ***Упражнение 5.2.*** Оцените использование памяти для хранения словаря коллекции Reuters–RCV1 по блокам при $k = 8$ и $k = 16$.

Упражнение 5.3. Оцените время, требуемое для поиска термина в сжатом словаре коллекции Reuters–RCV1 при размерах блока, равных $k = 4$ (рис. 5.6, *б*), $k = 8$ и $k = 16$. Насколько велико замедление по сравнению с вариантом, в котором размер блока равен $k = 1$ (рис. 5.6, *а*)?

5.3. Сжатие инвертированного файла

Напомним (см. табл. 4.2), что коллекция Reuters–RCV1 содержит 800 тысяч документов, 200 лексем в документе, шесть символов в лексеме и 100 миллионов словопозиций, причем для простоты в рамках этой главы мы считаем, что эта словопозиция состоит только из идентификатора документа docID, т.е. не содержит информации о частоте и координатах слова в документе. Эти числа соответствуют строке 3 ("свертывание регистра") в табл. 5.1. Идентификаторы документов занимают $\log_2 800\,000 \approx 20$ бит. Следовательно, размер коллекции равен примерно $800\,000 \times 200 \times 6$ байт = 960 Мбайт, а размер несжатого инвертированного файла равен $100\,000\,000 \times 20/8 = 250$ Мбайт.

Для более рационального представления инвертированных файлов, использующих менее 20 бит на документ, отметим, что словопозиции частых терминов имеют близкие значения. Мысленно пройдем по документам коллекции и поищем часто встречающийся термин `computer`. Мы найдем документ, содержащий слово `computer`, затем пропустим несколько документов, не содержащих это слово, после найдем документ, содержащий этот термин, и т.д. (табл. 5.3). Ключевая идея заключается в том, что *интервалы* между словопозициями являются короткими и для их хранения требуется меньше 20 бит. На практике интервалы между наиболее частыми терминами, таким как `the` и `for`, чаще всего равны единице. Однако пробелы между редкими терминами, встречающимися в коллекции лишь один или два раза (например, слово `arachnocentric` в табл. 5.3), по величине мало отличаются от идентификаторов документов и требуют для хранения 20 бит. Для экономного представления такого распределения интервалов необходим *метод упаковки с использованием кодов переменной длины* (variable encoding method), который для более коротких интервалов использует меньше битов.

Таблица 5.3. Кодирование интервалов вместо идентификаторов документов. Например, мы храним интервалы 107, 5, 43, ... вместо идентификаторов документов 283154, 283159, 283202, ... для термина computer. Первый идентификатор документа остается неизменным (показан только для термина arachnocentric)

	Кодировка	*Инвертированный список*								
The	ID документов	...		283042		283043		283044	283045	...
	Интервалы				1		1	1		...
Computer	ID документов	...		283047		283154		283159	283202	...
	Интервалы				107		5	43		...
Arachno-centric	ID документов	252000		500100						
	Интервалы	252000	2481000							

Для того чтобы закодировать небольшие числа, используя меньше памяти, чем для кодирования больших чисел, рассмотрим два вида методов: байтовое сжатие (bytewise compression) и битовое сжатие (bitwise compression). Как следует из их названий, эти методы кодируют интервалы в минимальное количество байтов и битов соответственно.

5.3.1. Байтовое кодирование переменной длины

Байтовое кодирование переменной длины (variable byte encoding — VB, или variable byte coding — VBC) использует для кодирования интервалов целое количество байтов. Последние 7 бит в каждом байте являются "полезной нагрузкой" и кодируют часть интервала. Первый бит байта является *битом продолжения* (continuation bit). Он равен единице у последнего байта закодированного интервала и нулю в остальных случаях.

```
VBEncodeNumber(n)
1   bytes ← 0
2   while true
3   do Prepend(bytes, n mod 128)
4      if n < 128
5        then Break
6      n ← n div 128
7   bytes[Length(bytes)] += 128
8   return bytes
VBEncode(numbers)
1   bytestream ← 0
2   for each n ∉ numbers
3   do bytes ← VBEncodingNumber(n)
4      bytestream ← Extend(bytestream, bytes)
5   return bytestream
VBDecode(bytestream)
1   numbers ← 0
2   n ← 0
3   for i ← 1 to Length(bytestream)
4   do if bytestream[i] < 128
5         then n ← 128 × n + bytestream[i]
6         else n ← 128 × n + (bytestream[i] – 128)
7              Append(numbers, n)
8              n ← 0
9   return numbers
```

Рис. 5.8. Кодирование и декодирование по схеме VB. Функции div и mod вычисляют целочисленное частное и остаток после целочисленного деления соответственно. Функция `Prepend` добавляет элемент в начало списка, например `Prepend`(⟨1, 2⟩, 3) = (3, 1, 2). Функция `Extend` расширяет список, например `Extend`(⟨1, 2⟩, ⟨3, 4⟩) = (1, 2, 3, 4)

Для декодирования кода, закодированного переменным количеством байтов, считывается последовательность байтов, бит продолжения которых равен нулю, завершающаяся байтом, в котором бит продолжения равен единице. Затем извлекаются и конкатенируются семибитовые части. Псевдокод байтового кодирования и декодирования переменной длины приведен на рис. 5.8, а пример инвертированного списка, закодированного по схеме VB, — в табл. 5.4. [4]

Таблица 5.4. Байтовое кодирование переменной длины. Интервалы закодированы с помощью целого числа байтов. Первый бит представляет собой бит продолжения и означает, заканчивается код этим байтом (1) или нет (0).

ID документов	824		829	215406		
Интервал			5	214577		
Код VB	00000110	10111000	10000101	00001101	00001101	10110001

Как показал эксперимент, при использовании байтового кодирования переменной длины размер сжатого индекса для коллекции Reuters–RCV1 равен 116 Мбайт. Это означает, что экономия памяти по сравнению с несжатым индексом превышает 50% (табл. 5.6).

Идею байтового кодирования переменной длины можно применить к единицам памяти, которые могут быть как больше, так и меньше байта: 32-, 16- и 4-битовые слова, или *полубайты* (nibbles). Более крупные слова еще сильнее уменьшают объем необходимых манипуляций битами за счет потери качества сжатия. Размеры слов, не превышающие байта, позволяют достичь еще более высоких степеней сжатия за счет увеличения количества манипуляций битами. В целом байты представляют собой приемлемый компромисс между степенью сжатия и скоростью распаковки.

Для большинства систем информационного поиска байтовое кодирование переменной длины обеспечивает превосходный компромисс между скоростью поиска и размером памяти. Кроме того, его легко реализовать — большинство альтернатив, упомянутых в разделе 5.4, являются более сложными. Однако, если дисковая память ограничена, можно достичь более высокой степени сжатия, применив побитовое кодирование, в частности две тесно связанные одна с другой схемы: γ-коды, которые будут рассмотрены ниже, и δ-коды (упражнение 5.9).

5.3.2. γ-коды

В схеме VB переменное количество *байтов* зависит от размера интервала. Методы битового уровня адаптируют длину кода на уровне *битов*. Простейшим битовым кодом является *унарный код* (unary code). Унарный код числа n представляет собой строку, состоящую из n единиц, за которыми следует нуль (см. первые два столбца в табл. 5.5). Очевидно, что это не очень эффективный код, но в свое время он нам пригодится.

[4] Обратите внимание на то, что в таблице начальное значение равно нулю. Поскольку кодировать идентификатор документа или нулевой интервал бессмысленно, на практике начальное значение обычно равно единице, так что 10000000 — это код единицы, 10000101 — шестерки (а не пятерки, как в таблице), и т.д.

Таблица 5.5. Примеры унарных и γ-кодов. Унарные коды показаны только для небольших чисел. Запятые в γ-кодах вставлены только для читабельности и не являются частью реальных кодов

Число	Унарный код	Длина	Смещение	γ-код
0	0			
1	10	0		0
2	110	10	0	10,0
3	1110	10	1	10,1
4	11110	110	00	110,00
9	1111111110	1110	001	1110,001
13		1110	101	1110,101
24		11110	1000	11110,1000
511		111111110	11111111	111111110,11111111
1025		11111111110	0000000001	11111111110,0000000001

Насколько эффективным может быть код в принципе? Предположим, что 2^n интервалов G, где $1 \le G \le 2^n$, являются равновероятными. Тогда оптимальное кодирование для каждого интервала G должно использовать n битов. В таком случае некоторые интервалы (в данном случае — $G = 2^n$) невозможно закодировать с помощью менее чем $\log_2 G$ бит. Наша цель — максимально приблизиться к нижней границе.

Метод, близкий к оптимальному, называется γ-*кодированием* (γ-*encoding*). Эта схема использует кодирование с переменной длиной с помощью разбиения интервала G на пару, состоящую из *длины* (length) и *смещения* (offset). *Смещение* представляет собой запись числа G в двоичном виде с удаленной ведущей единицей. [5] Например, для числа 13 (бинарный код — 1101) *смещение* равно 101. Параметр *длина* кодирует длину *смещения* в унарном коде. Для числа 13 длина *смещения* равна 3 бит. В унарном коде это число выглядит как 1110. Следовательно, γ-код числа 13 имеет вид 1110101. Он представляет собой конкатенацию длины 1110 и смещения 101. Другие примеры γ-кода приведены в правом столбце табл. 5.5.

Для того чтобы декодировать γ-код, сначала необходимо прочитать унарный код, пока не будет обнаружен нуль, который его завершает, например четыре бита 1110 при расшифровке числа 1110101. Теперь нам известна длина смещения: 3 бит. После этого можно правильно считать смещение 101 и добавить единицу, удаленную при кодировании: $101 \rightarrow 1101 = 13$.

Длина *смещения* (offset) равна $\lfloor \log_2 G \rfloor$ бит, а длина параметра *длина* (length) равна $\lfloor \log_2 G \rfloor + 1$ бит. Таким образом, длина всего кода равна $2 \times \lfloor \log_2 G \rfloor + 1$ бит. Все γ-коды имеют нечетную длину и всего в два раза больше длины оптимального кода, которая равна $\lfloor \log_2 G \rfloor$. Это оптимальное значение определено на основе предположения, что все 2^n интервалов от 1 до 2^n являются равновероятными. Однако на практике это условие может не выполняться. Как правило, априори распределение вероятностей интервалов неизвестно.

[5] Мы предполагаем, что интервал G не имеет ведущих нулей. Если бы они были, то их следовало бы удалить перед удалением ведущей единицы.

Свойства дискретного распределения вероятностей [6] P (включая то, является ли код оптимальным) характеризуются *энтропией* $H(P)$, которая определяется следующим образом.

$$H(P) = -\sum_{x \in X} P(x) \log_2 P(x)$$

Здесь X — множество всех возможных чисел, которые необходимо закодировать (и следовательно, $\sum_{x \in X} P(x) = 1,0$). *Энтропия* (entropy) — это мера неопределенности, что иллюстрирует рис. 5.9 для распределения вероятностей P среди двух исходов $X = [x_1, x_2]$. Энтропия достигает максимального уровня ($H(P) = 1$) при $P(x_1) = P(x_2) = 0{,}5$, когда неопределенность относительно того, какой из исходов x_i будет следующим, является наибольшей. Минимальный уровень энтропии ($H(P) = 0$) достигается при $P(x_1) = 0$ и $P(x_2) = 1$, когда существует полная определенность.

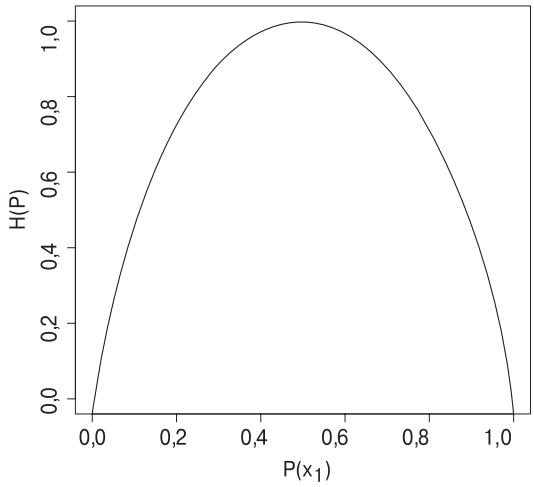


Рис. 5.9. Энтропия H(P) как функция, зависящая от $P(x_1)$ на пространстве, состоящем из двух исходов, x_1 и x_2

Можно показать (см. библиографию), что при определенных условиях нижняя граница $E(L)$ ожидаемой длины кода L равна $H(P)$. Кроме того, можно доказать, что при условии $1 < H(P) < \infty$ γ-кодирование не больше чем в три раза отличается от оптимального, а при больших значениях $H(P)$ этот показатель приближается к двум.

$$\frac{E(L_\gamma)}{H(P)} \leq 2 + \frac{1}{H(P)} \leq 3$$

Примечательно, что это соотношение сохраняется при любых распределениях вероятностей P. Итак, даже не зная никаких свойств распределения интервалов, мы можем применять γ-кодирование и быть уверенными, что при большой энтропии полученный

[6] Читатели, желающие ознакомиться с основными понятиями теории вероятностей, могут обратиться к учебникам Райса (Rice, 2006) и Росса (Ross, 2006). Отметим, что нас интересуют распределения вероятностей среди целых чисел (интервалы, частоты и т.д.), но свойства кодирования, присущие распределению вероятностей, не зависят от природы исходов (целые числа или нет).

код лишь примерно в два раза больше оптимального. Код, длина которого при произвольном распределении вероятностей P отличается от оптимальной лишь на множитель, называется *универсальным* (universal).

Кроме универсальности, γ-коды обладают еще двумя свойствами, полезными при сжатии индекса. Первое свойство — γ-код является *беспрефиксным* (prefix free), т.е. ни один γ-код не является префиксом другого кода. Это значит, что декодирование последовательности γ-кодов всегда является однозначным, и нам не нужны разделители между ними, которые снижают эффективность кодов. Второе свойство заключается в том, что γ-коды являются *непараметрическими* (parameter free). При использовании многих других эффективных кодов необходимо уточнять параметры распределения интервалов в индексе (например, параметры биномиального распределения). Это усложняет реализацию сжатия и распаковки. Например, параметры необходимо хранить в памяти и извлекать из нее. Кроме того, при динамическом индексировании распределение интервалов может изменяться, поэтому исходные параметры больше использовать нельзя. Непараметрический код позволяет избежать этих проблем.

Какой степени сжатия инвертированного индекса достигают γ-коды? Для ответа на этот вопрос применим закон Ципфа — модель распределения терминов, введенную в разделе 5.1.2. В соответствии с законом Ципфа частота термина в коллекции cf_i обратно пропорциональна рангу, т.е. существует константа c', такая, что

$$cf_i = \frac{c'}{i}. \tag{5.3}$$

Можно выбрать другую константу c, такую, что дроби c/i представляют собой относительные частоты, а их сумма равна единице (т.е. $c/i = \mathrm{cf}_i/T$).

$$1 = \sum_{i=1}^{M} \frac{c}{i} = c\sum_{i=1}^{M} \frac{1}{i} = cH_M \tag{5.4}$$

$$c = \frac{1}{H_M} \tag{5.5}$$

Здесь M — количество различных терминов, а H_M — M-е гармоническое число.[7] В коллекции Reuters–RCV1 содержится $M = 400$ тысяч разных терминов, а $H_M \approx \ln M$. Таким образом,

$$c = \frac{1}{H_M} \approx \frac{1}{\ln M} = \frac{1}{\ln 400000} \approx \frac{1}{13}.$$

Следовательно, относительная частота i-го термина примерно равна $1/(13i)$, а ожидаемое среднее количество терминов в документе длины L равно

$$L\frac{c}{i} \approx \frac{200 \times \frac{1}{13}}{i} \approx \frac{15}{i}.$$

Здесь вероятность появления термина заменена его относительной частотой. Напомним, что 200 — это среднее количество лексем в документе из коллекции Reuters–RCV1 (табл. 4.2).

Итак, мы определили статистические показатели термина, характеризующие распределение термина в коллекции и распределение интервалов в инвертированном списке. На

[7] К сожалению, общепринятым обозначением как энтропии, так и гармонического является буква H. Какое понятие имеется в виду, ясно из контекста.

основе этих показателей можно вычислить требования к памяти, необходимой для хранения инвертированного индекса, сжатого с помощью γ-кодирования. Сначала разобьем отсортированный по убыванию частоты лексикон терминов коллекции на блоки, размеры которых равны $Lc = 15$. В среднем i-й термин в документе встречается $15/i$ раз. Следовательно, среднее количество появлений в документе $\bar{f}$ для терминов из первого блока удовлетворяет условию $\bar{f} \geq 1$, что соответствует общему количеству интервалов в расчете на один термин (gaps per term), равному N. Среднее количество появлений терминов из второго блока в документе удовлетворяет условию $\frac{1}{2} \leq \bar{f} < 1$, что соответствует общему количеству интервалов в расчете на один термин, равному $N/2$. Среднее количество появлений терминов из третьего блока в документе удовлетворяет условию $\frac{1}{3} \leq \bar{f} < \frac{1}{2}$, что соответствует общему количеству интервалов в расчете на один термин, равному $N/3$ и т.д. (Мы выбираем нижнюю границу, поскольку это упростит дальнейшие вычисления. Как будет показано, даже при таком предположении окончательный результат слишком пессимистичен.) Примем довольно нереалистичное предположение, что все интервалы для заданного термина имеют одинаковые размеры, как показано на рис. 5.10.

	N документов
Lc наиболее частых терминов	N интервалов размером 1 в каждом
Lc следующих по частоте терминов	$N/2$ интервалов размером 2 в каждом
Lc следующих по частоте терминов	$N/3$ интервалов размером 3 в каждом

Рис. 5.10. Стратификация терминов для оценки размера инвертированного индекса, закодированного с помощью γ-кода

Сделав предположение, что интервалы распределены равномерно, мы в первом блоке получим интервалы размером 1, во втором блоке — интервалы размером 2, и т.д.

Кодируя N/j интервалов размером j с помощью γ-кодов, можно определить количество битов, необходимых для хранения инвертированного списка термина в j-м блоке (соответствует одной строке на рис. 5.10).

$$\textit{Бит в строке} = \frac{N}{j} \times \left(2 \times \lfloor \log_2 j \rfloor + 1\right) \approx \frac{2N \log_2 j}{j}$$

Для кодировки всего блока необходимо $Lc \cdot 2N\log_2 j/j$ бит. Общее количество блоков равно $M/(Lc)$, поэтому инвертированный файл в целом займет

$$\sum_{j=1}^{\frac{M}{Lc}} \frac{2NLc\log_2 j}{j} \text{ бит.} \tag{5.6}$$

Для коллекции Reuters–RCV1 $\frac{M}{Lc} \approx 400\,000/15 \approx 27\,000$ и

$$\sum_{j=1}^{27\,000} \frac{2\times10^6\times15\log_2 j}{j} \approx 224 \text{ Мбайт.} \tag{5.7}$$

Итак, инвертированный файл сжатого инвертированного индекса для коллекции объемом 960 Мбайт занимает 224 Мбайт, т.е. одну четвертую исходной коллекции.

Запустив γ-сжатие коллекции Reuters–RCV1, мы сжали индекс еще больше — до 101 Мбайт, что составляет чуть больше одной десятой размера коллекции. Причины такого расхождения между предсказанным и реальным значениями заключается в том, что (1) закон Ципфа не очень хорошо аппроксимирует фактическое распределение частот терминов в коллекции Reuters–RCV1 и (2) интервалы распределены неравномерно. В соответствии с моделью Ципфа для неокругленных данных из табл. 4.2 размер индекса должен быть равен 251 Мбайт. Если сгенерировать частоты терминов по модели Ципфа и для этих искусственных терминов создать сжатый индекс, то его размер будет равен 254 Мбайт. Следовательно, предсказания модели остаются верными в той мере, в какой выполняются предположения относительно распределения частот терминов.

Результаты применения методов сжатия, описанных в главе, приведены в табл. 5.6. Размер матрицы инцидентности терминов (рис. 1.1) для коллекции Reuters–RCV1 равен $400\,000 \times 800\,000 = 40 \times 8 \times 10^9$ бит, или 40 Гбайт.

Таблица 5.6. Сжатие индекса и лексикона для коллекции Reuters–RCV1. Коэффициент сжатия зависит от доли собственно текста в коллекции. Коллекция Reuters–RCV1 содержит большое количество разметки XML. Поэтому, используя две лучшие схемы сжатия — γ-кодирование и блочное хранение с фронтальной упаковкой, — можно получить очень маленькие значения коэффициента сжатия индекса относительно размера исходной коллекции для Reuters–RCV1: (101 + 5,9)/3600 ≈ 0,03

Структура данных	*Размер, Мбайт*
Словарь с фиксированным размером ячеек	11,2
Словарь с указателями на термины в строке	7,6
Словарь с указателями на термины в строке с блоками, k = 4	7,1
Словарь с указателями на термины в строке с блоками и фронтальной упаковкой	5,9
Коллекция (текст, разметка xml и т.д.)	3600,0
Коллекция (текст)	960,0
Матрица инцидентности терминов	40 000,0
Словопозиции (несжатые, 32-битовые слова)	400,0
Словопозиции (несжатые, 20 бит)	250,0
Словопозиции, переменное байтовое кодирование	116,0
Словопозиции, γ-кодирование	101,0

Схема γ-кодирования коллекции Reuters–RCV1 позволяет достичь отличного сжатия — почти на 15% лучшего, чем схема байтового кодирования переменной длины. Од-

нако ее декодирование является затратным. Это объясняется большим количество побитовых операций — сдвигов и применения масок, — которые необходимы для дешифровки последовательности γ-кодов, поскольку границы между кодами обычно расположены где-то в середине машинного слова. В результате процесс обработки запроса при использовании γ-кодирования становится более затратным, чем при использовании схемы байтового кодирования переменной длины. Выбор между этими двумя схемами зависит от особенностей приложения, например от относительной важности экономии дисковой памяти по сравнению со скоростью обработки запросов.

Коэффициент сжатия индекса в табл. 5.6 приблизительно равен 25%: 400 Мбайт (несжатый, каждая словопозиция хранится в виде 32-битового слова) по сравнению с 101 Мбайт (γ-кодирование) и 116 Мбайт (байтовое кодирование переменной длины). Это значит, что как схема γ-кодирования, так и схема байтового кодирования переменной длины достигают цели, поставленной в начале главы. Сжатие индекса существенно повышает скорость поиска и эффективность использования памяти за счет уменьшения требуемой дисковой памяти, одновременно увеличивая объем информации, которую необходимо хранить в кэше, и ускоряя передачу данных с диска в оперативную память.

? ***Упражнение 5.4*** **[*].** Вычислите коды, полученные с помощью схемы байтового кодирования переменной длины, для чисел из табл. 5.3 и 5.5.

Упражнение 5.5 **[*].** Вычислите коды, полученные с помощью схем байтового кодирования переменной длины и γ-кодирования для следующего инвертированного списка: ⟨777, 17743, 294068, 31251336⟩. Везде, где возможно, используйте интервалы, а не идентификаторы документов. Запишите бинарные коды блоками по 8 бит.

Упражнение 5.6. Рассмотрите инвертированный список ⟨4, 10, 11, 12, 15, 62, 63, 265, 268, 270, 400⟩ и соответствующий список интервалов ⟨4, 6, 1, 1, 3, 47, 1, 202, 3, 2, 130⟩. Предположим, что длина инвертированного списка хранится отдельно, так что система знает, когда он заполняется полностью. Используя схему байтового кодирования переменной длины, ответьте на следующие вопросы. 1) Какой максимальный интервал можно закодировать с помощью одного байта? 2) Какой максимальный интервал можно закодировать с помощью двух байтов? 3) Сколько байтов потребуется для кодировки указанного выше списка? (Учитывайте только память, необходимую для кодирования последовательности чисел.)

Упражнение 5.7. Заметив, что интервал не может иметь нулевую длину и что кодируемая часть, оставшаяся после сдвига, не может быть нулевой, можно прибегнуть к небольшому трюку.

1. Предложите модификацию схемы байтового кодирования переменной длины, позволяющую кодировать немного более длинные интервалы, используя тот же самый объем памяти.
2. Какой максимальный интервал можно закодировать с помощью одного байта?
3. Какой максимальный интервал можно закодировать с помощью двух байтов?
4. Сколько байтов потребуется для кодировки списка из упражнения 5.6 при таком подходе? (Учитывайте только память, необходимую для кодирования последовательности чисел.)

***Упражнение 5.8* [*].** Восстановите последовательность интервалов и последовательность словопозиций, используя следующую последовательность γ-кодов: 1110001110101011111101101111011.

Упражнение 5.9. Схема γ-кодирования для больших чисел является относительно неэффективной (например, 1025 в табл. 5.5), поскольку длина смещения в этой схеме кодируется с помощью неэффективного унарного кода. Схема δ-кодирования отличается от γ-кодирования тем, что она кодирует первую часть кода (*длину*) с помощью γ-кода, а не унарного кода. Кодирование *смещения* производится так же. Например, δ-код числа 7 равен 10,0,11 (как и прежде, мы добавили запятую для наглядности). Код 10,0 является γ-кодом *длины* (в данном случае — 2), а кодировка *смещения* (11) остается неизменной.

1. Вычислите δ-коды для других чисел из табл. 5.5. В каком диапазоне чисел δ-коды короче γ-кодов?
2. Схема γ-кодирования эффективнее схемы байтового кодирования переменной длины (см. табл. 5.6), так как индекс содержит стоп-слова, а значит, имеет много маленьких интервалов. Покажите, что при доминировании более крупных интервалов схема байтового кодирования переменной длины является более компактной.
3. Сравните коэффициенты сжатия для δ-кода и байт-кода переменной длины для распределения с преобладанием больших интервалов.

***Упражнение 5.10* [*].** Унарный код “10” представляет собой последовательность единиц, завершающуюся нулем. Если поменять нули и единицы ролями, то мы получим эквивалентный унарный код “01”. При использовании унарного кода “01” создание γ-кода можно начать так: а) запишем интервал G в двоичном виде, используя $b = \lfloor \log_2 j \rfloor + 1$ бит; б) добавим спереди $(b-1)$ нулей.

1. Закодируйте числа из табл. 5.5 с помощью этого альтернативного γ-кода.
2. Покажите, что этот метод позволяет создать достойную альтернативу γ-коду, т.е. его код имеет ту же длину и так же может быть однозначно расшифрован.

***Упражнение 5.11* [***].** Унарный код не является универсальным. Однако существует распределение интервалов, при котором унарный код становится оптимальным. Какое это распределение?

Упражнение 5.12. Приведите примеры терминов, нарушающих предположение о том, что интервалы имеют одинаковые размеры (которое мы сделали при оценке требований к объему памяти для γ-кодирования). Какими свойствами обладают эти термины?

Упражнение 5.13. Рассмотрите термин, инвертированный список которого имеет размер n, где n равно, скажем, 10 000. Сравните размер инвертированного списка, сжатого по схеме γ-кодирования интервалов, если распределение термина является равномерным (т.е. все интервалы имеют одинаковый размер), с его размером, когда это распределение не является равномерным. Какой из этих списков меньше?

Упражнение 5.14. Вычислите сумму в равенстве (5.7) и покажите, что она составляет около 251 Мбайт. Используйте числа из табл. 4.2, но не округляйте параметры *Lc*, *c* и количество блоков лексикона.

Упражнение 5.15. Проанализируйте вычисление размера индекса, приведенное выше, и укажите явно все округления, сделанные при выводе формулы (5.6).

Упражнение 5.16. Для выбранной вами коллекции определите количество документов и терминов, а также среднюю длину документа.

1. Насколько большим является размер индекса, предсказанный по формуле (5.6)?
2. Реализуйте индексатор, создающий инвертированный индекс этой коллекции, сжатый с помощью γ-кодирования. Насколько большим является реальный индекс?
3. Реализуйте индексатор, использующий схему байтового кодирования переменной длины. Насколько большим является индекс, сжатый по этой схеме?

Упражнение 5.17. Для того чтобы хранить в оперативной памяти как можно больше инвертированных списков, целесообразно сжать промежуточный индекс в ходе индексации. Во-первых, это усложняет слияние блоков в схеме блочного индексирования, основанного на сортировке. В качестве примера примените эту схему к объединенной последовательности интервалов, сжатой с помощью γ-кодирования и представленной в табл. 5.7. Во-вторых, создание индекса с использованием сжатия является более эффективным. Можно ли ожидать, что такая схема окажется еще и более быстрой?

Таблица 5.7. Две последовательности интервалов, подлежащих слиянию в схеме блочного индексирования, основанного на сортировке

Последовательность интервалов, сжатая с помощью γ-кодирования на первом шаге	1110110111111001011111111110100011111001
Последовательность интервалов, сжатая с помощью γ-кодирования на втором шаге	11111010000111111000100011111110010000011111010101

Упражнение 5.18. 1) Покажите, что в соответствии с законом Ципфа размер лексикона является конечным, а по закону Хипса он бесконечен. 2) Можно ли вывести закон Хипса из закона Ципфа?

5.4. Библиография и рекомендации для дальнейшего чтения

Закон Хипса был открыт Хипсом (Heaps, 1978). См. также учебник Баеза-Йейтса и Рибейро-Нето (Baeza-Yates and Ribeiro-Neto, 1999). Подробное изучение роста лексикона в крупных коллекциях провели Уильямс и Цобель (Williams and Zobel, 2005). Закон Ципфа открыт Ципфом (Zipf, 1949). Качество прогнозов, сделанных с помощью этого закона, исследовали Уиттен и Белл (Witten and Bell, 1990). Другие модели распределения терми-

нов, включая K-смеси и смеси двух пуассоновских распределений, рассматривались в книге Маннинга и Шютце (Manning and Schütze, 1999). Кармел и др. (Carmel et al., 2001), Бютчер и Кларке (Büttcher and Clarke, 2006), Бланко и Баррейро (Blanco and Barreiro, 2007), а также Нтулас и Чо (Ntoulas and Cho, 2007) показали, что сжатие с потерей информации может обеспечить хорошее сжатие без существенного снижения качества поиска.

Сжатие словаря детально исследовалось в работе Уиттена и др. (Witten et al., 1999), которую мы рекомендуем для дополнительного чтения.

Материал, изложенный в подразделе 5.3.1, основан на работе Шолера и др. (Scholer et al., 2002). Эти авторы выяснили, что схема байтового кодирования переменной длины позволяет обработать запрос вдвое быстрее, чем схема побитового сжатия индекса или отказ от сжатия вообще. При этом коэффициент сжатия оказался на 30% хуже, чем в лучшей схеме побитового сжатия. Кроме того, они показали, что сжатые индексы имеют преимущество перед несжатыми не только за счет экономии дисковой памяти, но и за счет более быстрой обработки запросов. В одном из экспериментов коды, основанные на "переменных полубайтах" (variable nibbles), оказались на 5–10% компактнее схемы байтового кодирования переменной длины, но их эффективность оказалась на треть ниже (Anh and Moffat, 2005). Тротман (Trotman, 2003) также рекомендует использовать схему байтового кодирования переменной длины, если только дисковое пространство не является критическим ресурсом. В своих последних работах Анх и Моффат (Anh and Moffat, 2005, 2006*a*), а также Жуковский и др. (Zukowski et al., 2006) сконструировали бинарные коды, выровненные по границам машинных слов, которые не уступают схеме байтового кодирования переменной длины по степени сжатия, при этом обеспечивая более быстрое декодирование. Чанг и др. (Zhang et al., 2007) исследовал повышенную эффективность кэширования в сочетании с использованием многочисленных методов сжатия инвертированных списков на современном аппаратном обеспечении.

Схемы δ- и γ-кодирования были предложены Элиасом (Elias, 1975), доказавшим, что оба эти кода являются универсальными. Кроме того, δ-коды являются асимптотически оптимальными при $H(P) \to \infty$. При доминировании больших чисел (больше 15) схема δ-кодирования является более эффективной, чем схема γ-кодирования. Хорошим введением в теорию информации, включая концепцию энтропии, является книга Кавера и Томаса (Cover and Thomas, 1991). В то время как коды Элиаса являются лишь асимптотически оптимальными, можно построить арифметические коды (Witten et al., 1999), сколь угодно близкие к оптимальному значению $H(P)$ при любом распределении P.

Несколько дополнительных методов сжатия индекса описаны в книге Уиттена и др. (Witten et al., 1999). Авторы рекомендуют использовать для сжатия индекса *параметризованные коды* (parametrized codes), явно моделирующие распределение вероятностей интервалов для каждого термина. Например, они показали, что для крупных коллекций *коды Голомба* (Golomb codes) обеспечивают более высокий коэффициент сжатия, чем γ-коды. Моффат и Цобель (Moffat and Zobel, 1992) сравнили несколько параметризованных методов, включая метод LLRUN (Fraenkel and Klein, 1985).

Распределение интервалов в инвертированном списке зависит от присвоения идентификаторов документам. Многие исследователи искали способ назначения идентификаторов, обеспечивающих эффективное сжатие последовательностей интервалов (Moffat and Stuiver, 1996; Blandforf and Blelloch, 2002; Silvestri et al., 2004; Blanco and Barriero, 2006; Silvestri, 2007). Эти методы позволяют назначать идентификаторы из небольшого диапазона документам, образующим кластер. Кластер может содержать все документы за определенный временной период, находящиеся на определенном веб-сайте или обладаю-

щие определенными общими свойствами. В результате, когда последовательность документов из кластера оказывается в инвертированном списке, их интервалы становятся малыми и могут быть эффективно сжаты.

Сжатие частот термина и координат слов отличается от сжатия идентификаторов документов (Scholer et al., 2002; Zobel and Moffat, 2006). Глубокое и современное описание инвертированных индексов, включая сжатие индексов, содержится в обзоре Цобеля и Моффата (Zobel and Moffat, 2006).

В данной главе изложены методы сжатия индекса лишь для логического поиска. При ранжированном поиске (глава 6) выгоднее упорядочить словопозиции по частотам терминов, а не по идентификаторам документов. Тогда в ходе обработки запросов сканирование многих инвертированных списков может завершаться на ранних этапах, поскольку более низкие веса не изменяют ранжирование k документов, имеющих в текущий момент наивысший показатель релевантности. Вычислять заранее и хранить веса в индексе (в отличие от частот) нецелесообразно, поскольку их невозможно сжать так же, как целые числа (см. раздел 7.1.5).

Сжатие документов (document compression) также может играть важную роль в информационно-поисковых системах. Де Моура и др. (de Moura et al., 2000) и Брисбо и др. (Brisboa et al., 2007) описали схемы сжатия, обеспечивающие прямой поиск терминов и фраз в сжатом тексте, который невозможно осуществить с помощью стандартных средств сжатия наподобие `gzip` и `compress`.

Глава 6

Ранжирование[1], взвешивание терминов и модель векторного пространства

До сих пор мы рассматривали индексы, поддерживающие обработку булевых запросов: документ либо соответствовал, либо не соответствовал запросу. При работе с большими коллекциями документов итоговое количество документов, соответствующих запросу, может быть таким большим, что человек просто не в состоянии просмотреть их все. Соответственно, одной из важных задач поисковых машин является ранжирование документов по степени их соответствия запросу. Для решения этой задачи поисковые машины для каждого найденного документа вычисляют его степень соответствия заданному запросу, т.е. *(вычисленная) релевантность* [2] (score). В данной главе мы начинаем изложение с вычисления релевантности пары (запрос, документ). В главе освещаются три основные идеи.

1. В разделе 6.1 описываются параметрические и зонные индексы, предназначенные для достижения двух целей. Во-первых, они позволяют индексировать и находить документы по метаданным, например по языку, на котором этот документ написан. Во-вторых, они позволяют относительно просто ранжировать документы по степени их соответствия запросу.
2. В разделе 6.2 развивается идея о взвешивании важности терминов в документе на основе статистических данных об их встречаемости.

---

[1] Для перевода термина “scoring” мы выбрали в конце концов не “вычисление/присвоение рейтинга/релевантности”, а “ранжирование”, так как хотя и не совсем идентичный термин “ranking” гораздо популярнее в англо- и русскоязычной литературе для описания тех же самых процесса и задачи, (в частности, потому что последние годы в машинном обучении ранжированию (machine learning to rank) доминируют методы, решающие задачу упорядочивания (ранжирования) в альтернативных постановках: pair-wise (попарное упорядочивание) или list-wise (списковое упорядочивание)). Они могут вообще не оперировать одним значением score (рейтинга или вычисленного значения релевантности) для документа. Кроме того, физический и логический смысл этого значения (score) в них существенно утерян. — *Примеч. ред.*

[2] Термин “релевантность” (полный вариант “вычисленное значение релевантности”) избран нами как перевод термина “score” вместо существенно более редко используемого слова “рейтинг”. В главах об оценке качества поиска термин “relevance” (также переводимый нами как “релевантность”) трактуется исключительно как бинарная оценка эксперта, но мы надеемся, что это не помешает пониманию текста: в спорных случаях мы будем говорить “оценка релевантности экспертом”. — *Примеч. ред.*

3. В разделе 6.3 показано, что если рассматривать каждый документ как вектор таких весов, то можно вычислить соответствие между запросом и каждым из документов. Этот подход называется ранжированием на основе модели векторного пространства (vector space scoring).

В разделе 6.4 изложено несколько вариантов взвешивания терминов в модели векторного пространства. Вычислительные аспекты и другие темы, связанные с ранжированием в векторном пространстве, изложены в главе 7.

Разработка этих идей требует уточнения понятия запроса. В разделе 6.1 рассматриваются запросы, в которых конкретные термины встречаются в конкретных частях соответствующего документа. Начиная с раздела 6.2 мы ослабим требование, касающееся соответствия конкретных частей документа, перейдя к так называемым свободным текстовым запросам, состоящим из терминов без указания порядка их следования, важности и местоположения в документе. Основная часть нашего исследования методов ранжирования связана именно с представлением запроса как множества терминов.

6.1. Параметрические и зонные индексы

До сих пор мы рассматривали документ как последовательность терминов. На самом деле большинство документов имеют дополнительную структуру. Электронные документы обычно сопровождаются *метаданными* (metadata), которые кодируются в виде, распознаваемом компьютерами. Под метаданными мы понимаем конкретные виды данных о документе, например фамилию автора, название и дату публикации. Эти метаданные обычно содержат *поля метаданных* (fields), например дату создания и формат документа[3], а также фамилию автора и, возможно, название документа. Множество возможных значений этих полей следует считать конечным, например множество всех дат создания документа ограничено.

Рассмотрим запрос “Найти документы, созданные Вильямом Шекспиром в 1601 году и содержащие слова alas poor Yorick”. В этом случае обработка запроса сводится, как обычно, к поиску пересечения инвертированных списков, за исключением того, что мы можем объединить словопозиции как из стандартных инвертированных, так и из *параметрических индексов* (parametric indexes)[4]. Для каждого поля (например, для даты создания) существует один параметрический индекс; он позволяет выбрать только те документы, которые соответствуют дате, указанной в запросе. На рис. 6.1 показан интерфейс пользователя для параметрического поиска. Некоторые поля подразумевают упорядоченные значения (например, даты); в указанном выше запросе одним из таких значений является год “1601”. Поисковая машина может поддерживать запросы с указанием диапазонов таких упорядоченных значений; для хранения набора значений поля можно использовать такие структуры данных, как B-деревья.

Зоны (zones) напоминают поля, но содержанием зоны может быть произвольный текст. В то время как поле может иметь относительно небольшое множество значений, зона может содержать произвольный и неограниченный объем текста. Например, названия документов и аннотации обычно трактуются как зоны. Для того чтобы поддержать

---

[3] В отечественной и зарубежной практике очень часто вместо “поле метаданных” или “поле” используется “атрибут” или “атрибут документа”. — *Примеч. ред.*

[4] Также “атрибутивный индекс”, индекс “атрибутов документа” и “атрибутивный поиск”. — *Примеч. ред.*

обработку запросов вида “Найти документы со словами merchant в названии и William — в списке авторов, а также с фразой gentle rain в тексте” для каждой зоны документа можно создать стандартный инвертированный индекс. Такой индекс выглядит примерно так, как показано на рис. 6.2. В то время как словарь для параметрического индекса создается на основе фиксированного списка значений (множества языков, множества дат), словарь для зонного индекса формируется на основе текстов, размещенных в данной зоне.

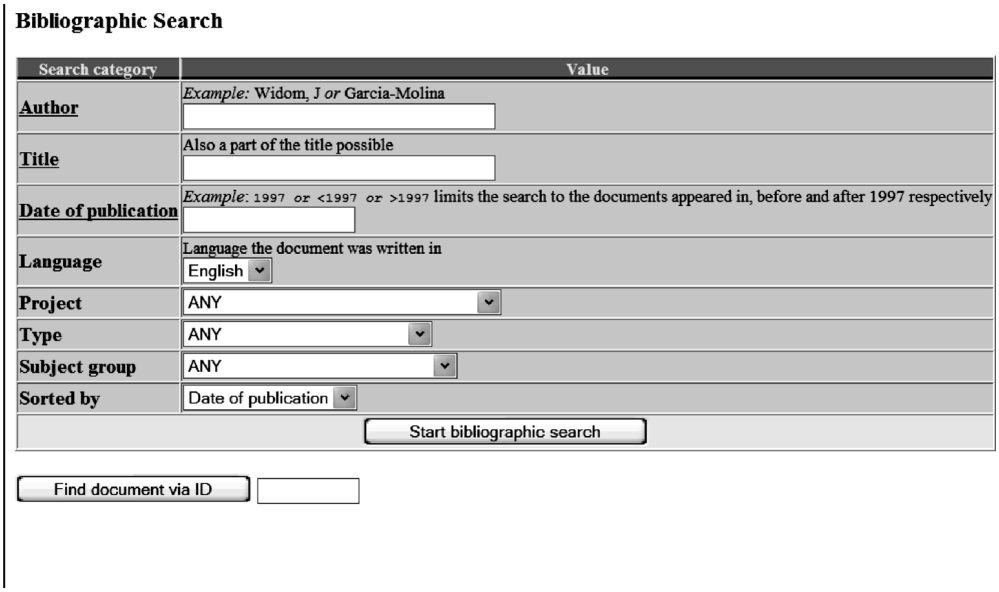


Рис. 6.1. Параметрический поиск. В данном примере мы имеем коллекцию с полями, позволяющими отбирать публикации по зонам, например Author (àâòîð) или Language (язык)

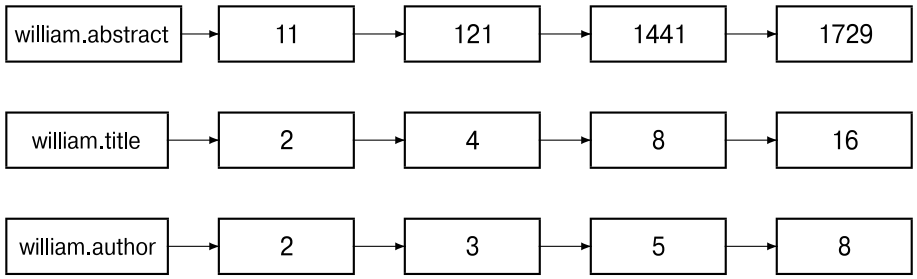


Рис. 6.2. Элементарный зонный индекс; зоны кодируются как расширения элементов словаря

На самом деле размер словаря можно уменьшить, закодировав зону, в которой встречается термин, в словопозиции. Например, на рис. 6.3 показано, как кодируется появление слова william в названии и в зоне авторов разных документов. Такое кодирование полезно, когда размер словаря имеет значение (поскольку мы требуем, чтобы словарь хранился в оперативной памяти). Однако существует еще одна причина, по которой кодирование, показанное на рис. 6.3, оказывается полезным: эффективное ранжирование *взвешиванием по зонам* (weighted zone scoring).

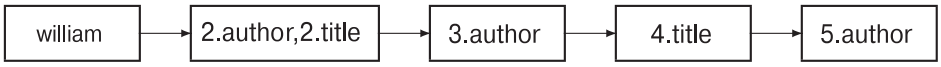


Рис. 6.3. Зонный индекс, в котором зона закодирована в словопозиции, а не в словаре

6.1.1. Взвешенное зонное ранжирование

До сих пор в разделе 6.1 мы были сосредоточены на поиске документов на основе булевых запросов относительно полей и зон. Теперь перейдем ко второму приложению зон и полей.

Обозначим булев запрос буквой q, а документ — буквой d. Метод взвешенного зонного ранжирования присваивает паре (q, d) значение релевантности из отрезка [0,1], вычисляя линейную комбинацию *зонных показателей* (zone score), в которую каждая зона документа вносит булево значение. Говоря конкретнее, рассмотрим множество документов, каждый из которых имеет l зон. Пусть $g_1, g_2, ..., g_l \in [0,1]$, так что $\sum_{i=1}^{l} g_i = 1$. Пусть s_i, где $1 \le i \le l$, — булева величина, означающая соответствие (или его отсутствие) между запросом q и i-й зоной. Например, если все термины запроса принадлежат конкретной зоне, то ее булево значение должно быть равным единице, а если нет — нулю. Действительно, это отображение может осуществлять любая булева функция, отображающая наличие терминов запроса в зоне в множество $\{0, 1\}$. Таким образом, взвешенную зонную релевантность можно определить как

$$\sum_{i=1}^{l} g_i s_i \ . \qquad (6.1)$$

Взвешенное зонное ранжирование иногда называется *булевым поиском с ранжированием* (ranked Boolean retrieval).

Пример 6.1. Рассмотрите запрос shakespeare к коллекции, в которой каждый документ имеет три зоны: *author* (автор), *title* (заголовок) и *body* (основной текст). Булева функция ранжирования (Boolean score function) для зоны принимает значение, равное единице, если термин запроса shakespeare принадлежит этой зоне, и нулю, если нет. Взвешенное зонное ранжирование в такой коллекции подразумевает использование трех весов, g_1, g_2 и g_3, соответствующих зонам *author*, *title* и *body*. Допустим, что $g_1 = 0{,}2$, $g_2 = 0{,}3$ и $g_3 = 0{,}5$ (так что сумма всех трех весов равна единице); это соответствует приложению, в котором соответствие в зоне *author* менее важно по сравнению со всеми другими зонами, соответствие в зоне *title* является более важным, а соответствие в зоне *body* важнее всего.

Таким образом, если термин shakespeare появился в зонах *title* и *body*, но отсутствует в зоне *author*, то релевантность документа будет равна 0,8.

Как вычислить взвешенную зонную релевантность? Для этого можно просто поочередно вычислить релевантность для каждого документа, суммируя вклады разных зон. Однако взвешенную зонную релевантность можно вычислить прямо по инвертированным индексам. Алгоритм, приведенный на рис. 6.4, предназначен для варианта, в котором запрос q содержит два термина, q_1 и q_2, и булеву функцию AND: 1 — если оба термина запроса присутствуют в зоне, и 0 — если нет. После алгоритма мы опишем его расширение для более сложных запросов и булевых функций.

Читатели могли заметить большое сходство между алгоритмами, представленными на рис. 6.4 и 1.6. Действительно, они реализуют один и тот же проход по словопозициям, за исключением того, что вместо простого добавления документа в множество результатов для булева запроса AND в данном случае мы вычисляем релевантность каждого документа. В некоторых работах массив текущих значений релевантности scores[] называется множеством *накопителей* (accumulator). Это объясняется тем, что для более сложных по сравнению с операцией AND булевых функций релевантность документа может быть ненулевой, даже если он не содержит все термины запроса.

```
ZoneScore(q1, q2)
1   float scores[N] = [0]
2   constant g[l]
3   p1 ← postings(q1)
4   p2 ← postings(q2)
5   // scores[] — массив релевантностей для каждого документа, инициализированный нулем
6   // p1 и p2 вначале ссылаются на начало соответствующих словопозиций
7   // массив g[] инициализируется соответствующими зонными весами
8   while p1 ≠ NIL and p2 ≠ NIL
9   do if docID(p1) = docID(p2)
10      then score[docID(p1)] ← WeightedZone(p1, p2, g)
11           p1 ← next(p1)
12           p2 ← next(p2)
13      else if docID(p1) < docID(p2)
14           then p1 ← next(p1)
15           then p2 ← next(p2)
16  return scores
```

Рис. 6.4. Алгоритм взвешенного зонного ранжирования по двум инвертированным спискам. Функция WeightedZone (здесь не показана) вычисляется во внутреннем цикле формулы (6.1)

6.1.2. Определение весов на основе машинного обучения

Как определить веса g_i при взвешенном зонном ранжировании? Они могут быть указаны экспертами (или пользователем). Однако гораздо чаще веса определяются на основе обучающих примеров, оцененных заранее. Этот метод относится к общему классу методов ранжирования в информационном поиске под названием “методы *ранжирования на основе машинного обучения*” (machine-learned relevance). В данном разделе мы даем краткое введение в эту тему, поскольку взвешенное зонное ранжирование — хороший материал для такого изложения. Более полное описание требует понимания принципов машинного обучения, поэтому отложено до главы 15.

1. В нашем распоряжении имеется множество обучающих примеров (training examples), каждый из которых представляет собой кортеж, состоящий из запроса q, документа d, а также оценки релевантности d и q. В простейшем случае каждая оценка релевантности (relevance judgment) является бинарной, т.е. *релевантный* или *нерелевантный*. В более сложных реализациях этой методологии используются более тонкие оценки.

2. Веса g_i определяются путем "обучения" на данных примерах так, чтобы полученные оценки аппроксимировали оценки релевантности обучающих примеров.

При взвешенном зонном ранжировании этот процесс можно рассматривать как подбор коэффициентов линейной функции от булевых признаков вхождения в соответствующие зоны. Затратной частью этой методологии является трудоемкий сбор оценок релевантности, по которым производится обучение, особенно если коллекция часто изменяется (как веб). Опишем простой пример, показывающий, как свести задачу определения весов g_i на основе обучения к простой задаче оптимизации.

Рассмотрим простой вариант взвешенного зонного ранжирования, в котором каждый документ имеет зоны *title* и *body*. Применим к заданным запросу q и документу d булеву функцию соответствия и вычислим булевы значения $s_T(d, q)$ и $s_B(d, q)$, указывающие, соответствует ли заголовок (соответственно, тело) документа d запросу q. Например, алгоритм, представленный на рис. 6.4, использует для этого функцию AND, примененную к терминам запроса. Вычислим значение, лежащее между нулем и единицей, для каждой пары (документ, запрос), используя значения $s_T(d, q)$ и $s_B(d, q)$ и константу $g \in [0,1]$:

$$score(d, q) = g \cdot s_T(d, q) + (1–g)s_B(d, q). \quad (6.2)$$

Теперь опишем, как определить константу g по *обучающим примерам*, каждый из которых представляет собой тройку вида $\Phi_j = (d_j, q_j, r(d_i, q_j))$. Каждому обучающему примеру, содержащему документ d_i и запрос q_i, человеком-редактором дается оценка релевантности $r(d_i, q_j)$, принимающая значение *релевантный* или *нерелевантный*. Этот процесс проиллюстрирован на рис. 6.5, на котором показаны семь примеров.

Пример	*DocID*	*Запрос*	s_T	$s_в$	*Суждение*
Φ_1	37	linux	1	1	Релевантный
Φ_2	37	penguin	0	1	Нерелевантный
Φ_3	238	system	0	1	Релевантный
Φ_4	238	penguin	0	0	Нерелевантный
Φ_5	1741	kernel	1	1	Релевантный
Φ_6	2094	driver	0	1	Релевантный
Φ_7	3191	driver	1	0	Нерелевантный

Рис. 6.5. Пример обучающих примеров

Для каждого обучающего примера Φ_j у нас есть булевы значения $s_T(d_j, q_j)$ и $s_B(d_j, q_j)$, используемые для ранжирования по формуле (6.2).

$$score(d_j, q_j) = g \cdot s_T(d_j, q_j) + (1–g)s_B(d_j, q_j) \quad (6.3)$$

Теперь сравним этот вычисленную релевантность с оценкой релевантности той же пары (d_j, q_j), сделанной человеком; для этого каждая оценка *релевантный* кодируется единицей, а *нерелевантный* — нулем. Допустим, что ошибка весовой функции с весом g определяется по следующей формуле.

$$\varepsilon(g, \Phi_j) = (r(d_j,q_j) – score(d_j,q_j))^2$$

Здесь оценка редактора $r(d_j,q_j)$ кодируется единицей или нулем. Таким образом, суммарная ошибка, соответствующая множеству обучающих примеров, равна

$$\sum_j \varepsilon\left(g, \Phi_j\right). \tag{6.4}$$

Теперь задача определения константы g по обучающим примерам сводится к нахождению значения g, минимизирующего суммарную ошибку (6.4).

Определение наилучшего значения g на основе формулы (6.4) в формулировке раздела 6.1.3 сводится к задаче минимизации квадратичной функции, зависящей от g, на отрезке [0,1]. Эта тема обсуждается в разделе 6.1.3.

6.1.3. Оптимальный вес g

Отметим, что для любого обучающего примера Φ_j, такого, что $s_T(d_j, q_j) = 0$ и $s_B(d_j, q_j) = 1$, релевантность, вычисленная по формуле (6.2), равен $1–g$. Аналогично можно записать релевантность, вычисленную по формуле (6.2) при трех других возможных комбинациях значений $s_T(d_j, q_j)$ и $s_B(d_j, q_j)$; эти комбинации продемонстрированы на рис. 6.6.

s_T	s_B	*Релевантность*
0	0	0
0	1	$1–g$
1	0	g
1	1	1

Рис. 6.6. Четыре возможные комбинации s_T *и* s_B

Обозначим через n_{01r} (соответственно, n_{01n}) количество обучающих примеров, для которых $s_T(d_j, q_j) = 0$ и $s_B(d_j, q_j) = 1$, а экспертная оценка — *релевантный* (соответственно, *нерелевантный*). Тогда вклад в суммарную ошибку в выражении (6.4) от обучающих примеров, для которых $s_T(d_j, q_j) = 0$ и $s_B(d_j, q_j) = 1$, равен

$$[1 - (1 - g)]^2 n_{01r} + [0 - (1 - g)]^2 n_{01n}.$$

Записав аналогичным образом вклады в ошибку от обучающих примеров при трех других комбинациях значений $s_T(d_j, q_j)$ и $s_B(d_j, q_j)$ (и расширив обозначения очевидным образом), приходим к выводу, что суммарная ошибка, определяемая по формуле (6.4), равна

$$(n_{01r} + n_{10n})g^2 + (n_{10r} + n_{01n})(1–g)^2 + n_{00r} + n_{11n}. \tag{6.5}$$

Дифференцируя выражение (6.5) по g и приравнивая результат к нулю, получаем, что оптимальное значение g равно

$$\frac{n_{10r} + n_{01n}}{n_{10r} + n_{10n} + n_{01r} + n_{01n}}. \tag{6.6}$$

Упражнение 6.1. Необходимо ли, чтобы при использовании метода взвешенного зонного ранжирования все зоны использовали одну и ту же булеву функцию совпадения (соответствия)?

Упражнение 6.2. Примем в примере 6.1 следующие веса: $g_1 = 0{,}2$, $g_2 = 0{,}31$ и $g_3 = 0{,}49$. Перечислите все отличающиеся друг от друга значения, которые может иметь документ.

Упражнение 6.3. Перепишите алгоритм, представленный на рис. 6.4, для случая, когда запрос содержит больше двух терминов.

Упражнение 6.4. Напишите псевдокод для функции WeightedZone для случая двух инвертированных списков, как на рис. 6.4.

Упражнение 6.5. Примените формулу (6.6) к набору обучающих примеров, представленному на рис. 6.5, и оцените наилучшее значение g.

Упражнение 6.6. Для значения g, найденного в упражнении 6.5, вычислите релевантность документа с помощью метода взвешенного зонного ранжирования для каждого примера (запрос, документ). Как эта релевантность соотносится с экспертными оценками, указанными на рис. 6.5 (суждения закодированы числами 0 и 1)?

Упражнение 6.7. Почему выражение для величины g в формуле (6.6) не учитывает обучающие примеры, для которых значения $s_T(d_j, q_j)$ и $s_B(d_j, q_j)$ одинаковы?

6.2. Частота термина и взвешивание

До сих пор ранжирование документа зависело от того, присутствует ли термин запроса в зоне документа. Теперь мы сделаем следующий логичный шаг: документ или зона, где термин запроса встречается чаще, следует считать более релевантным запросу и присвоить ему более высокое значение релевантности. Для обоснования этой точки зрения напомним понятие свободного текстового запроса, введенное в разделе 1.4: запрос, в котором термины вводятся в интерфейс поисковой машины в свободном виде, без соединительных операторов (таких, как булевы операторы). Такой стиль, весьма популярный в сети веб, рассматривает запрос просто как множество слов. Следовательно, для подсчета показателя документа достаточно просто суммировать показатели соответствия документа каждому из слов запроса.

Для этого присвоим каждому термину, обнаруженному в документе, *вес* (weight), зависящий от количества появлений этого термина в данном документе. Мы хотим оценить соответствие между термином запроса t и документом d, основываясь на весе термина t в документе d. Проще всего положить этот вес равным количеству вхождений термина t в документ d. Эта схема взвешивания называется *частотой термина* (term frequency) и обозначается как $\mathrm{tf}_{t,d}$, где индекс t обозначает термин, а индекс d — документ.

Для документа d набор весов tf (или определенных с помощью любой другой весовой функции, которая ставит в соответствие количеству появлений термина t в документе d некое положительное действительное число) можно интерпретировать как дайджест документа, выраженный в числовом виде. Эта модель в научной литературе называется *мешком слов* (bag of words model). В рамках этой модели точный порядок следования терминов в документе игнорируется, а основное значение придается количеству вхождений каждого термина в документ (в противоположность булеву поиску). Мы только накапливаем информацию о количестве вхождений каждого термина. Таким образом, документ *Mary is quicker than John* в этой модели идентичен документу *John is quicker than Mary*. Тем не менее интуитивно ясно, что два документа с одинаковыми “мешками слов” по содержанию очень сходны. Мы разовьем этот подход в разделе 6.3.

Сначала ответим на следующий вопрос: все ли слова в документе одинаково важны? Очевидно, нет; в разделе 2.2.2 мы рассмотрели идею *стоп-слов* (stop words), т.е. слов, которые не включаются в индекс вообще и, следовательно, никак не влияют на поиск и ранжирование.

6.2.1. Обратная документная частота

Подсчет “в лоб” частоты термина, описанный выше, имеет серьезный недостаток: при ранжировании документа по запросу все термины считаются одинаково важными. На самом деле некоторые термины имеют малую или нулевую различительную силу при определении релевантности. Например, коллекция документов об автомобильной промышленности, скорее всего, содержит термин “auto” практически в каждом документе. Для того чтобы устранить указанный недостаток, мы введем механизм ослабления влияния термина, который встречается в коллекции слишком часто, чтобы его имело смысл учитывать при определении релевантности документов. На ум сразу же приходит идея снизить веса у терминов с высокой *частотой в коллекции* (collection frequency), представляющей собой общее количество вхождений термина в коллекцию. Идея состоит в том, чтобы уменьшить вес термина *tf* на коэффициент, который увеличивается по мере увеличения его частоты в коллекции.

Вместо этого чаще встречается использование *документной частоты* df_i (document frequency), представляющей собой количество документов в коллекции, содержащих термин *t*. Это объясняется тем, что, пытаясь найти различия между документами с целью их ранжирования по запросу, лучше использовать статистические показатели именно самих документов (например, количество документов, содержащих заданный термин), чем статистические показатели коллекции в целом. Преимущество документной частоты перед частотой в коллекции продемонстрировано на рис. 6.7, где на простом примере продемонстрировано, что частота в коллекции (cf) и документная частота (df) могут вести себя по-разному. В частности, частоты терминов try и insurance в коллекции примерно одинаковы, в то время как документные частоты этих терминов сильно отличаются друг от друга. Интуитивно ясно, что документы, содержащие слово insurance, должны иметь б*о*льшую релевантность по отношению к запросу, содержащему термин insurance, чем документы, содержащие слово try, по отношению к запросу, содержащему термин try.

Слово	*cf*	*Df*
try	10422	8760
insurance	10440	3997

Рис. 6.7. Частота в коллекции (cf) и документная частота (df)

Как использовать документную частоту терма для коррекции его веса? Обозначим, как обычно, общее количество документов в коллекции через *N* и определим *обратную документную частоту* (inverse document frequency) термина *t* следующим образом.

$$idf_t = \log \frac{N}{df_t} \tag{6.7}$$

Таким образом, обратная документная частота редко встречающегося термина является большой, в то время как для часто встречающегося термина она невелика. На рис. 6.8 приведены примеры обратных документных частот в коллекции Reuters–RCV1, состоящей из 806 791 документа. В этих примерах использованы десятичные логарифмы. На самом деле, как показано в упражнении 6.12, выбор базы логарифма при ранжировании не имеет значения. Обоснование конкретной формы равенства (6.7) будет приведено позднее.

Термин	cf_t	idf_t
car	18 165	1,65
auto	6 723	2,08
insurance	19 241	1,62
best	25 335	1,6

Рис. 6.8. Примеры обратных документных частот. Приведены обратные документные частоты терминов с разными частотами встречаемости в коллекции Reuters–RCV1 (806 791 документ).

6.2.2. Взвешивание на основе комбинации частоты и обратной документной частоты термина

Скомбинируем частоту термина в документе (term frequency — tf) и обратную документную частоту (inverse document frequency — idf), чтобы получить вес каждого термина в каждом документе. Схема взвешивания *tf–idf* присваивает каждому термину t его вес в документе d на основе формулы

$$\text{tf–idf}_{t,d} = \text{tf}_{t,d} \times \text{idf}_t. \tag{6.8}$$

Иначе говоря, вес $\text{tf–idf}_{t,d}$ термина t в документе d обладает следующими свойствами.

1. Он достигает максимального значения, если термин t встречается много раз в небольшом количестве документов (тем самым усиливая их отличие от других документов).
2. Он уменьшается, если термин встречается в каком-то документе лишь несколько раз или встречается во многих документах (тем самым формируя менее выраженный сигнал о релевантности документа).
3. Он достигает минимального значения, если термин встречается практически во всех документах.

Каждый документ можно интерпретировать как *вектор* (vector), состоящий из компонентов, соответствующих каждому термину в словаре, и весов каждого компонента, вычисленных по формуле (6.8). Вес каждого словарного термина, не встречающегося в документе, равен нулю. Эта векторная форма весьма важна для взвешивания и ранжирования (связанные с ней идеи будут развиты в разделе 6.3). На первом этапе введем *меру перекрытия* (overlap score measure): релевантность документа d равна сумме вхождений всех терминов запроса в этот документ. Впоследствии мы уточним эту идею так, чтобы суммировать не количество вхождений каждого термина запроса t в документ d, а их веса tf–idf.

$$Score(q,d) = \sum_{t \in d} tf - idf_{t,d} \tag{6.9}$$

В разделе 6.3 мы придем к более точному варианту формулы (6.9).

? ***Упражнение 6.8.*** Почему обратная документная частота термина всегда является конечной?

Упражнение 6.9. Чему равна обратная документная частота термина, встречающегося в каждом документе? Сравните этот результат с использованием списка стоп-слов.

Упражнение 6.10. Рассмотрите таблицу частот терминов для трех документов, обозначенных на рис. 6.9 через Doc1, Doc2 и Doc3. Вычислите веса tf–idf для терминов car, auto, insurance и best для каждого документа, используя значения idf из рис. 6.8.

Термин	*Doc1*	*Doc2*	*Doc3*
car	27	4	24
auto	3	33	0
insurance	0	33	29
best	14	0	17

Рис. 6.9. Таблица частот терминов для упражнения 6.10

Упражнение 6.11. Может ли вес tf–idf термина в документе превысить единицу?

Упражнение 6.12. Как основание логарифма в равенстве (6.7) влияет на вычисления в формуле (6.9)? Как основание логарифма влияет на относительные релевантности двух документов для заданного запроса?

Упражнение 6.13. Предложите простую аппроксимацию обратной документной частоты термина, если логарифм в формуле (6.7) вычисляется по основанию 2.

6.3. Модель векторного пространства для ранжирования

В разделе 6.2 мы ввели представление документа в виде вектора, отражающего относительную важность терминов в документе. Представление множества документов в виде векторов в общем векторном пространстве называется *моделью векторного пространства* (vector space model) и является фундаментальным для многих задач информационного поиска, включая ранжирование документов по запросу, классификацию и кластеризацию документов. Сначала мы изложим основные идеи, лежащие в основе ранжирования документов в векторном пространстве; ключевым моментом здесь является представление запросов (раздел 6.3.2) в виде векторов того же векторного пространства, что и коллекция документов.

6.3.1. Скалярное произведение

Обозначим через $\vec{V}(d)$ вектор, построенный по документу d, в котором каждому термину словаря соответствует отдельный компонент. Если не указано иное, читатели могут считать, что каждый компонент вычисляется путем взвешивания терминов по схеме tf–idf, хотя конкретная схема взвешивания в этой модели значения не имеет. Множество документов в коллекции можно представить в виде векторов в векторном пространстве, в котором каждому термину соответствует отдельная ось. При таком представлении происходит потеря относительного порядка следования терминов в каждом документе; напомним

пример из раздела 6.2, в котором указано, что документы *Mary is quicker than John* и *John is quicker than Mary* в рамках модели "мешка слов" идентичны.

Как выразить сходство между двумя документами в векторном пространстве в количественной форме? Например, можно попытаться оценить модуль разности между векторами, соответствующими разным документам. Такая мера сходства имеет недостаток: векторная разность между двумя документами с очень близким содержанием может быть значительной, если один из них существенно длиннее другого. Иначе говоря, относительное распределение терминов в двух документах может быть идентичным, но абсолютные частоты терминов в одном из них могут быть намного больше.

Для того чтобы компенсировать влияние длины документа, для двух документов, d_1 и d_2, обычно вычисляется *косинусная мера сходства* (cosine similarity) между их векторными представлениями $\vec{V}(d_1)$ и $\vec{V}(d_2)$.

$$sim(d_1,d_2)=\frac{\left(\vec{V}(d_1),\vec{V}(d_2)\right)}{\left\|\vec{V}(d_1)\right\|\left\|\vec{V}(d_2)\right\|}. \tag{6.10}$$

Здесь числитель представляет собой *скалярное произведение* (dot product, inner product) векторов $\vec{V}(d_1)$ и $\vec{V}(d_2)$, а знаменатель равен произведению *евклидовых норм* (Euclidean lengths) этих векторов. Скалярное произведение $(\vec{x},\vec{y})$ двух векторов равно $\sum_{i=1}^{M} x_i y_i$. Обозначим через $\vec{V}(d)$ вектор документа *d* с компонентами $\vec{V}_1(d),...,\vec{V}_M(d)$. Евклидова длина вектора *d* равна $\sqrt{\sum_{i=1}^{M}\vec{V}_i^2(d)}$.

Знаменатель в формуле (6.10) *нормирует по длине* (length-normalize) векторы $\vec{V}(d_1)$ и $\vec{V}(d_2)$, так, чтобы их длина стала равна единице: $\vec{v}(d_1)=\frac{\vec{V}(d_1)}{\left\|\vec{V}(d_1)\right\|}$ и $\vec{v}(d_2)=\frac{\vec{V}(d_2)}{\left\|\vec{V}(d_2)\right\|}$. Теперь формулу (6.10) можно переписать в виде

$$sim(d_1,d_2)=\left(\vec{v}(d_1),\vec{v}(d_2)\right). \tag{6.11}$$

Пример 6.2. Рассмотрим три документа, представленные на рис. 6.9. Применим к их векторам, содержащим частоты терминов, евклидову нормировку. Величины $\sqrt{\sum_{i=1}^{M}\vec{V}_i^2(d)}$ для документов Doc1, Doc2 и Doc3 равны 30,56, 46,84 и 41,30 соответственно. Итоговые результаты евклидовой нормировки этих векторов приведены на рис. 6.10.

Термин	*Doc1*	*Doc2*	*Doc3*
Car	0,88	0,09	0,58
Auto	0,10	0,71	0
insurance	0	0,71	0,70
Best	0,46	0	0,41

Рис. 6.10. Нормированные векторы частот терминов для документов, представленных на рис. 6.9

Таким образом, выражение (6.11) можно интерпретировать как скалярное произведение нормированных векторов, соответствующих двум документам. Как показано на рис. 6.11, эта мера сходства равна косинусу угла θ между двумя векторами. Как использовать меру сходства sim(d_1, d_2)? Для документа d (возможно, одного из документов d_i в коллекции) попробуем найти документ из коллекции, больше других похожий на документ d. Такой поиск полезен в системе, в которой пользователь может идентифицировать документ и искать похожие на него, — в результатах поисковых машин такая функциональность обозначается как “похожие документы” или “more like this”. Сведем задачу поиска документов, наиболее похожих на документ d, к поиску документов d_i с наибольшим скалярным произведением $\left(\tilde{v}(d),\tilde{v}(d_i)\right)$. Для этого можно было бы вычислить скалярные произведения вектора $\tilde{v}(d)$ и всех векторов $\tilde{v}(d_1),...,\tilde{v}(d_N)$, а затем взять документы с максимальными значениями меры сходства.

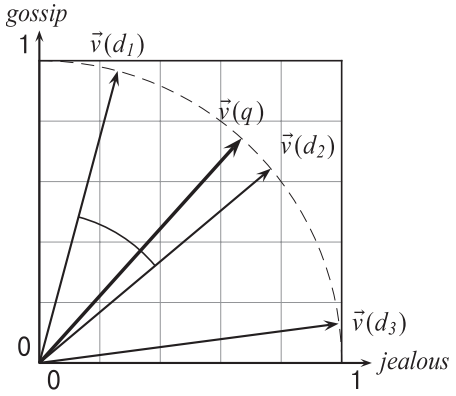


Рис. 6.11. Косинусная мера сходства: sim(d_1, d_2) = cos θ

Пример 6.3. На рис. 6.12 показано количество вхождений трех терминов (affection, jealous и gossip) в каждый из романов Джейн Остин “Чувство и чувствительность” и “Гордость и предубеждение” (Jane Austen, *Sense and Sensibility* (SaS) и *Pride and Prejudice* (PaP)), а также Эмили Бронте “Грозовой перевал” (Emily Bronte, *Wuthering Heights* (WH)). Разумеется, в каждом из этих романов содержится много других терминов. В данном примере мы представили каждый из этих романов в виде единичного трехмерного вектора, соответствующего только этим трем терминам; в таблице приведены частоты терминов без умножения на обратную документную частоту. Результаты расчетов приведены на рис. 6.13.

Термин	*SaS*	*PaP*	*WH*
affection	115	58	20
jealous	10	7	11
gossip	2	0	6

Рис. 6.12. Частоты терминов в трех романах: Джейн Остин “Чувство и чувствительность” и “Гордость и предубеждение” и Эмили Бронте “Грозовой перевал”

Термин	SaS	PaP	WH
affection	0,996	0,993	0,847
jealous	0,087	0,120	0,466
gossip	0,017	0	0,254

Рис. 6.13. Векторы терминов для трех романов из рис. 6.12, основанные на обычных частотах терминов в предположении, что в коллекцию не входят никакие другие термины. (Поскольку термины affection и jealous встречаются во всех трех документах, их вес tf–idf был бы равен нулю.)

Теперь рассмотрим косинусную меру сходства между парами итоговых трехмерных векторов. Простые вычисления показывают, что $sim(\tilde{v}(SaS),\tilde{v}(PaP)) = 0{,}999$ и $sim(\tilde{v}(SaS),\tilde{v}(WH)) = 0{,}888$. Таким образом, две книги Джейн Остин (SaS и PaP) намного ближе друг к другу, чем к роману Эмили Бронте "Грозовой перевал". Сходство первых двух романов друг к другу является практически идеальной (если ограничиться тремя указанными терминами). Здесь мы рассмотрели веса, основанные на обычных частотах, но вместо них можно использовать другие весовые функции.

Представление коллекции, состоящей из N документов, как коллекции векторов приводит к естественной трактовке коллекции как *матрицы "термин–документ"* (term–document matrix), т.е. матрицы размером $M \times N$, в которой строки содержат M терминов (размерностей), расположенных в N столбцах, соответствующих документам. Как всегда, к терминам, подлежащим индексированию, может применяться стемминг; например, после выделения основы термины jealous и jealousy можно рассматривать как одну размерность. Полезность матричного представления обсуждается в главе 18.

6.3.2. Запросы как векторы

Существует еще одна, намного более весомая, причина представлять документ в виде вектора: *запрос* также можно рассматривать как вектор. Рассмотрим запрос q = jealous gossip. Его можно преобразовать в трехмерный вектор единичной длины $\tilde{v}(q) = (0; 0{,}707; 0{,}707)$, представленный в координатах рис. 6.12 и 6.13. Основная идея этого подхода заключается в том, чтобы присвоить каждому документу d значение релевантности, равное скалярному произведению

$$(\tilde{v}(q),\tilde{v}(d)).$$

В примере, представленном на рис. 6.13, наиболее релевантным этому запросу является роман *Грозовой перевал* (его релевантность равна 0,509), а романы *Pride and Prejudice* и *Sense and Sensibility* с большим отставанием занимают второе и третье места с релевантностью 0,085 и 0,074 соответственно. Этот пример в некотором смысле может ввести в заблуждение: количество размерностей на практике намного больше трех — оно равно размеру словаря M.

Подводя итоги, отметим, что, рассматривая запрос как "мешок слов", мы можем интерпретировать его как очень короткий документ. Следовательно, можно использовать косинусную меру сходства между вектором запроса и вектором документа как значение

релевантности документа запросу. Затем значения релевантности можно применять для выбора лучших документов по соответствию запросу. Итак,

$$score(q,d) = \frac{\left(\tilde{V}(q), \tilde{V}(d)\right)}{\left\|\tilde{V}(q)\right\| \left\|\tilde{V}(d)\right\|}. \tag{6.12}$$

Документ может иметь большую косинусную меру сходства к запросу, даже если он не содержит все термины запроса. Отметим, что в предыдущем рассуждении мы не конкретизировали способ взвешивания терминов в векторе документа; можно считать, что применяются схемы tf и tf–idf. На самом деле как к запросу, так и к документу можно применять самые разные схемы взвешивания терминов, как показано в примере 6.4 и будет подробно описано в разделе 6.4.

Вычисление косинусной меры сходства между вектором запроса и каждым вектором документа коллекции, сортировка по релевантности и выбор K лучших документов может оказаться довольно затратным процессом: вычисление меры сходства для одного документа может повлечь вычисление скалярного произведения для векторов, состоящих из десятков тысяч компонентов, для чего требуются десятки тысяч арифметических операций. В разделе 7.1 мы покажем, как для этой цели можно использовать инвертированный индекс и некоторые эвристические правила для повышения эффективности этой операции.

Пример 6.4. Рассмотрим запрос best car insurance к фиктивной коллекции, состоящей из $N = 1$ миллиона документов. Документные частоты терминов auto, best и insurance равны 5 000, 50 000, 10 000 и 1 000 соответственно.

Термин	*Запрос*				*Документ*			*Скалярное произведение*
	tf	Df	idf	$w_{t,q}$	tf	wf	$w_{t,d}$	
Auto	0	5000	2,3	0	1	1	0,41	0
Best	1	50000	1,3	1,3	0	0	0	0
Car	1	10000	2,0	2,0	1	1	0,41	0,82
Insurance	1	1000	3,0	3,0	2	2	0,82	2,36

В данном примере вес термина в запросе просто равен обратной документной частоте (idf) и нулю, если термина в запросе нет (например, слова auto). Это отражено в столбце $w_{t,q}$ (напротив термина auto в этом столбце стоит нуль, так как этого слова в запросе нет). К документам применена схема взвешивания по обычным частотам терминов (tf), без обратных документных частот (idf), но с использованием евклидовой нормировки (см. значение в столбце $w_{t,d}$). Подставляя эти значения в формулу (6.12), получаем итоговое значение релевантности: 0 + 0 + 0,82 + + 2,46 = 3,28.

6.3.3. Ранжирование в векторной модели

На практике у нас обычно есть коллекция документов, каждый из которых представлен в виде вектора, свободный текстовый запрос, представленный в виде вектора, а также положительное целое число K. Мы ищем K документов коллекции с наибольшими значениями релевантности запросу. Покажем, как определить эти K документов. Обычно просматриваемые K документов расположены в порядке убывания релевантности. На-

пример, во многих поисковых машинах число K равно десяти, и первая страница содержит ранжированную десятку лучших результатов. Приведем основной алгоритм для таких вычислений. Более полный анализ эффективных приемов и приближенных методов решения этой задачи будет приведен в главе 7.

Базовый алгоритм ранжирования в векторном пространстве приведен на рис. 6.14. Массив *Length* содержит длины (коэффициенты нормировки) векторов, соответствующих каждому из N документов, а массив *Scores* содержит релевантность каждого документа. После того как массив *Scores* будет вычислен на этапе 9, остается лишь на этапе 10 извлечь K документов с наибольшей релевантностью.

CosineScore(q)

1 float *Scores*[N] = 0
2 Инициализация *Length*[N]
3 **for each** термин запроса t // для каждого термина запроса
4 **do** вычислить $w_{t,q}$ и извлечь инвертированный список для термина t
5 **for each** пары (d, $\mathrm{tf}_{t,d}$) в инвертированном списке
6 **do** *Scores*[d] += $wf_{t,d} \times w_{t,q}$
7 Прочитать массив *Length*[d]
8 **for each** d
9 **do** *Scores*[d] = *Scores*[d]/*Length*[d]
10 **return** наибольшие K компонентов массива *Scores*[]

Рис. 6.14. Базовый алгоритм ранжирования в векторном пространстве

Внешний цикл, начинающийся на этапе 3, обновляет массив Scores, просматривая по очереди каждый термин t. На этапе 5 мы вычисляем вес термина t в векторе q. Этапы 6–8 обновляют релевантность каждого документа, прибавляя вклад термина t. Процесс суммирования вкладов, или *аккумуляция* (accumulation), от каждого термина по очереди иногда называется *ранжированием термин за термином* (term-at-a-time scoring), а N элементов массива *Scores* называются *аккумуляторами* (accumulators). На первый взгляд, для этой цели необходимо для каждого элемента инвертированного списка хранить вес $\mathrm{wf}_{t,d}$ термина t в документе d (до сих пор в качестве веса мы использовали показатели tf или tf–idf термина, но, как будет показано в разделе 6.4, возможны и другие схемы взвешивания). На самом деле это слишком неэкономно, поскольку для хранения такого веса потребуется действительное число с плавающей точкой. Решить эту проблему можно двумя способами. Во-первых, если использовать обратную документную частоту, то вычислять частоту idf_t заранее не обязательно; достаточно хранить число N/df_t в начале словопозиции термина t. Во-вторых, можно хранить частоту термина $\mathrm{tf}_{t,d}$ для каждого элемента инвертированного списка. В заключение на этапе 10 происходит извлечение K документов с наибольшей релевантностью. Для этого требуется очередь с приоритетами, которая часто реализуется с помощью структуры данных под названием “куча” (heap). Для создания такой кучи потребуется не более $2N$ сравнений, после которых можно извлечь первые K документов, выполнив $O(\log N)$ сравнений на документ.

Обратите внимание на то, что общий алгоритм, приведенный на рис. 6.14, не регламентирует обход инвертированного списка разных терминов; мы можем обходить эти термины один за другим, как в цикле, начинающемся на этапе 3, или делать это параллельно, как показано на рис. 1.6. При параллельном обходе словопозиций релевантность

документов вычисляется поочередно, поэтому данный метод иногда называется *ранжированием документ за документом* (document-at-a-time scoring). Более подробно он будет описан в разделе 7.1.5.

? ***Упражнение 6.14.*** Как следует изменить определения частоты термина и обратной документной частоты термина, если перед конструированием векторного пространства в результате стемминга термины jealous и jealousy сводятся к общей основе?

Упражнение 6.15. Вспомните веса tf–idf, вычисленные в упражнении 6.10. Вычислите нормированные векторы документов для каждого из документов, где каждый вектор содержит четыре компонента, соответствующих каждому их четырех терминов.

Упражнение 6.16. Убедитесь, что сумма квадратов компонентов каждого из векторов документов в упражнении 6.15 равна единице (с точностью до ошибок округления). Объясните этот факт.

Упражнение 6.17. Используя веса, вычисленные в упражнении 6.15, определите релевантность трех документов с помощью вычисленных мер сходства с запросом car insurance для каждого из приведенных ниже вариантов вычисления веса термина в запросе.

1. Вес термина равен единице, если он есть в запросе, и нулю, если его в запросе нет.
2. Вес равен нормированной обратной документной частоте.

6.4. Варианты функций tf–idf

Помимо рассмотренных выше схем tf и tf–idf, существует большое количество вариантов взвешивания терминов в документах. Рассмотрим наиболее важные из них. Более подробное изложение этой темы содержится в главе 11. Обзор этих альтернатив приведен в разделе 6.4.3.

6.4.1. Сублинейное масштабирование tf

Кажется маловероятным, что двадцать вхождений термина в документ на самом деле в двадцать раз важнее однократного вхождения. По этой причине были проведены многочисленные исследования, цель которых заключалась в определении способов вычисления *частоты термина* (term frequency), которые бы не просто считали количество вхождений термина. Распространенной модификацией стало использование логарифма частоты термина:

$$wf_{t,d} = \begin{cases} 1 + \log tf_{t,d}, \text{ если } tf_{t,d} > 0, \\ 0 - \text{в противном случае.} \end{cases} \tag{6.13}$$

Точно так же можно заменить величину tf любой другой функцией wf и получить

$$\text{wf-idf}_{t,d} = \text{wf}_{t,d} \times \text{idf}_t. \tag{6.14}$$

Равенство (6.9) можно модифицировать, заменив величину tf–idf величиной wf–idf в соответствии с формулой (6.14).

6.4.2. Нормировка tf на максимальный tf в документе

Одной из наиболее хорошо изученных схем взвешивания является нормировка весов tf всех терминов, встречающихся в документе, с помощью максимальной величины tf в этом документе. Для каждого документа d введем обозначение $\mathrm{tf}_{\max}(d) = \max_{\tau \in d} \mathrm{tf}_{\tau, d}$, где индекс τ пробегает по всем терминам в документе d. Нормированную частоту термина t в документе d можно определить по формуле

$$ntf_{t,d} = a + (1-a)\frac{tf_{t,d}}{tf_{max}(d)}. \qquad (6.15)$$

Здесь a — величина, лежащая между нулем и единицей (как правило. ее полагают равной 0,4, хотя в некоторых ранних работах использовалось значение 0,5). Величина a в формуле (6.15) называется *сглаживающим коэффициентом* (smoothing term). Его роль — уменьшать вклад второго члена, который представляет собой величину tf, деленную на максимальное значение tf в документе d. Мы еще вернемся к сглаживанию в главе 13, когда будем обсуждать классификацию; основная идея заключается в том, чтобы избежать больших колебаний величины $\mathrm{ntf}_{t,d}$ при небольших изменениях $\mathrm{tf}_{t,d}$ (скажем, от одного до двух). Нормировка частоты термина по максимуму предназначена для того, чтобы избежать следующей аномалии: в более длинных документах наблюдаются более высокие частоты терминов просто потому, что в более длинных документах чаще содержатся повторяющиеся слова. Чтобы разобраться в этом, рассмотрим следующий экстремальный пример: допустим, что мы взяли документ d и создали новый документ d', просто добавив в документ d его копию. Несмотря на то что документ d' не должен быть более релевантным по отношению к любому запросу, чем документ d, формула (6.9) удваивает его релевантность по сравнению с релевантностью документа d. Замена величины $\mathrm{tf–idf}_{t,d}$ в формуле (6.9) величиной $\mathrm{ntf–idf}_{t,d}$ позволяет исключить эту аномалию. Нормировка величины tf по максимуму имеет следующие недостатки.

1. Данный метод является неустойчивым: изменение списка стоп-слов может резко изменить веса терминов (а значит, и ранжирование). Следовательно, его трудно настроить.
2. Документ может содержать аномальный термин с необычно высокой частотой встречаемости, который не отражает содержание документа.
3. Иными словами, документ, в котором наиболее часто повторяющийся термин встречается примерно так же часто, как и многие другие термины, должен обрабатываться иначе, чем документ, имеющий более асимметричное распределение[5].

6.4.3. Схемы взвешивания документов и запросов

Формула (6.12) носит фундаментальный характер для информационно-поисковых систем, использующих ту или иную форму взвешивания в векторном пространстве. Методы ранжирования в векторном пространстве отличаются друг от друга конкретным выбором весов в векторах $\vec{V}(d)$ и $\vec{V}(q)$. На рис. 6.15 перечислены основные схемы взвешива-

[5] Существует нормировка, лишенная указанного недостатка и экономная по длине кода, основанная на сортировке терминов документа по убыванию частоты в документе и замене частоты tf номером интервала в полученном массиве (A Statistical View of Binned Retrieval Models, Metzler, 2008, или Vector-space ranking with effective early termination, N. Anh et al, 2001). — *Примеч. ред.*

ния для векторов $\vec{V}(d)$ и $\vec{V}(q)$, а также мнемонические обозначения разных комбинаций весов; эта система мнемонических обозначений иногда называется SMART в честь одной из первых текстовых систем информационного поиска. Мнемонические обозначения, обозначающие комбинации весов, имеют вид *ddd.qqq*, где первый триплет соответствует системе взвешивания терминов в векторе документа, а второй — в векторе запроса. Первая буква каждого триплета обозначает компонент, соответствующий частоте термина, вторая — документную частоту, а третья — вид нормировки. Довольно часто векторы $\vec{V}(d)$ и $\vec{V}(q)$ нормируются по-разному. Например, весьма распространенной является схема взвешивания *lnc.ltc*, где для составления вектора документа используется log-взвешенная частота термина, не применяется обратная документная частота термина (по соображениям эффективности поиска и производительности системы) и выполняется косинусная нормировка, в то время как для создания вектора запроса используются log-взвешенная частота термина, обратная документная частота термина и косинусная нормировка.

Частота термина	*Документная частота*	*Нормировка*
n (natural) $\mathrm{tf}_{t,d}$	n (no) 1	n (none) 1
l (logarithm) $1+\log(\mathrm{tf}_{t,d})$	t (idf) $\log\frac{N}{dt_i}$	c (cosine) $\frac{1}{\sqrt{w_1^2+w_2^2+\ldots+w_M^2}}$
a (augmented) $0{,}5+\frac{0{,}5\mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf) $\max\left[0,\log\frac{N-df_t}{df_t}\right]$	u (pivoted uniqie) 1/*u* (раздел 17.4.4)
b (boolean) $\begin{cases}1, \text{если } \mathrm{tf}_{t,d}>0,\\ 0-\text{в противном случае}\end{cases}$		b (byte size) 1/CharLength[a], a<1
L (log ave) $\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(ave_{te d}(\mathrm{tf}_{t,d}))}$		

Рис. 6.15. Система обозначений SMART для вариантов tf–idf. Здесь CharLength — количество символов в документе

6.4.4. Опорная нормировка длины документов

В разделе 6.3.1 мы нормировали каждый вектор с помощью его евклидовой нормы, так что все векторы документов имели единичную длину. Тем самым мы исключили всю информацию о длине исходного документа, что может оказаться довольно важным при обработке длинных документов. Во-первых, длинные документы имеют более высокие значения tf, поскольку они содержат больше терминов. Во-вторых, длинные документы содержат больше разных терминов. Эти факторы могут способствовать завышению релевантности более длинных документов, что (по крайней мере для некоторых информационных потребностей) является неестественным. Длинные документы можно разделить на две категории: 1) *многословные* (verbose) документы, в которых содержание часто повторяется; в таких документах длина не влияет на относительные веса разных терминов; 2) документы, посвященные нескольким разным темам, в которых искомые термины, вероятно, соответствуют небольшим фрагментам документа, но не соответствуют всему документу; в этом случае относительные веса терминов существенно отличаются от ве-

сов для отдельных коротких документов, которые соответствуют терминам запроса. Для компенсации этого эффекта используется нормировка длины документа, не зависящая ни от частоты термина, ни от документной частоты. Для этого вводится нормировка векторов документов из коллекции, при которой длина "нормированных" векторов не обязательно равна единице. Тогда при вычислении скалярного произведения между (единичным) вектором запроса и нормированным документом релевантность должен "корректироваться", чтобы учесть влияние длины документа. Этот вид компенсации длины документа называется *опорной нормировкой длины документа* (pivoted document length normalization).

Рассмотрим коллекцию документов вместе с набором запросов к этой коллекции. Допустим, что для каждого запроса q и каждого документа d известна бинарная оценка о том, релевантен документ d запросу q или нет. В главе 8 мы обсудим способы получения таких оценок релевантности для набора запросов и коллекции документов. Имея такое множество оценок релевантности, можно вычислить *вероятность релевантности* (probability of relevance) как функцию, зависящую от длины документа и усредненную по всем запросам в наборе запросов. Итоговый график может выглядеть так, как жирная кривая на рис. 6.16. Для вычисления этой кривой мы отсортировали документы по длине и разбили их на группы, равные по числу документов в каждой, вычислили долю релевантных документов в каждой группе, а затем построили график зависимости этой доли от медианы длин документов в группе. (Таким образом, даже если кривая на рис. 6.16 кажется непрерывной, на самом деле она представляет собой гистограмму дискретных групп длин документов.)

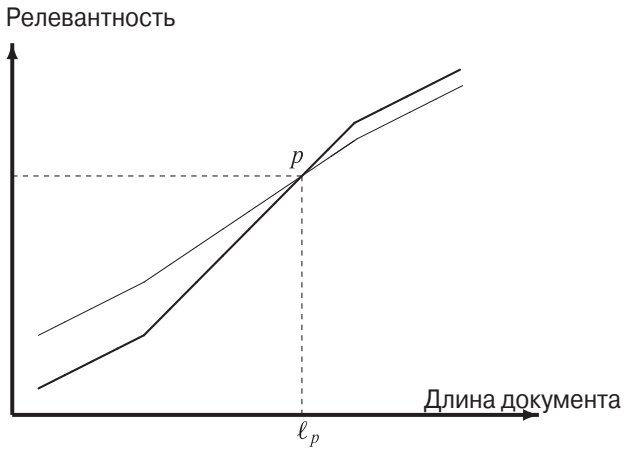


Рис. 6.16. Опорная нормировка длины документа

С другой стороны, тонкая кривая показывает, что может случиться с теми же документами и набором запросов, если для ранжирования использовать косинусную нормировку по формуле (6.12). Как видим, косинусная нормировка искажает вычисленную релевантность по отношению к истинной релевантности за счет длинных документов. Тонкая и жирная линии пересекаются в точке p, соответствующей длине документа l_p, которую мы будем называть *опорной длиной* (pivot length); эта точка отмечена пунктирными перпендикулярами, опущенными на оси x и y. Идея опорной нормировки длины документа состоит в том, чтобы так "повернуть" кривую, соответствующую косинусной

нормировке, против часовой стрелки вокруг точки p, чтобы она как можно больше совпала с жирной линией, соответствующей зависимости релевантности документа от длины документа. Как указывалось в начале раздела, это можно сделать, включив для каждого вектора документа $\vec{V}(d)$ в равенстве (6.12) коэффициент нормировки, не являющийся евклидовой нормой вектора. Он выбирается бо́льшим евклидовой нормы для документов, длина которых меньше l_p, и меньшим для более длинных документов.

Отметим, что коэффициент нормировки вектора $\vec{V}(d)$ в знаменателе равенства (6.12) представляет собой евклидову норму, обозначенную как $\left\|\vec{V}(d)\right\|$. В простейшем варианте опорной нормировки длины документов в знаменателе используется коэффициент нормировки, линейно зависящий от нормы $\left\|\vec{V}(d)\right\|$ с угловым коэффициентом меньше единицы, как показано на рис. 6.17. Ось x на этом рисунке соответствует норме $\left\|\vec{V}(d)\right\|$, а ось y — возможным коэффициентам нормировки. Жирная линия $y = x$ соответствует косинусной нормировке. Отметим следующие свойства жирной линии, отражающей опорную нормировку.

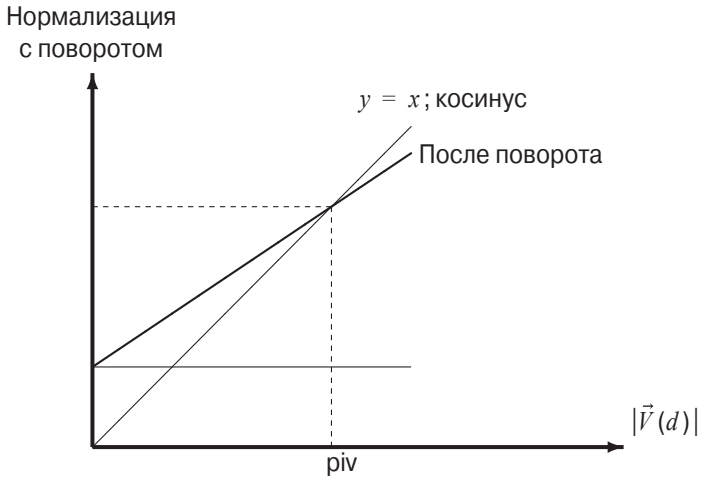


Рис. 6.17. Реализация опорной нормировки длины документа с помощью линейного масштабирования

1. Этот график представляет собой прямую линию и соответствует функции

$$a\left\|\vec{V}(d)\right\|+(1-a)\,piv\,, \tag{6.16}$$

   где piv — значение косинусной нормировки, при которой две кривые пересекаются.
2. Угловой коэффициент графика меньше единицы.
3. График пересекается с прямой $y = x$ в точке piv.

На практике функция (6.16) хорошо аппроксимируется выражением

$$au_d + (1 - a)piv,$$

где u_d — количество уникальных терминов в документе d.

Разумеется, опорная нормировка длины документа подходит не для всех приложений. Например, в коллекции ответов на часто задаваемые вопросы (скажем, на веб-сайте клиентской службы) релевантность может слабо зависеть от длины документа. В других ситуациях зависимость может быть более сложной, чем та, которую учитывает простая линейная опорная нормировка. Тогда длину документа можно использовать как признак для машинного обучения (см. раздел 6.1.2).

Упражнение 6.18. Одной из мер сходства между двумя векторами является *евклидово расстояние* (или L_2 -расстояние) между ними.

$$\|\vec{x}-\vec{y}\| = \sqrt{\sum_{i=1}^{M}(x_i - y_i)^2}$$

Имея запрос q и документы d_1, d_2, ..., мы можем ранжировать документы d_i в порядке возрастания евклидового расстояния от запроса q. Покажите, что, если запрос q и документы d_i преобразованы в единичные вектора, то ранжирование по евклидовому расстоянию эквивалентно ранжированию на основе косинусной меры сходства.

Упражнение 6.19. Вычислите меру сходства в векторном пространстве между запросом “digital cameras” и документом “digital cameras and video cameras”, заполнив пустые столбцы в табл. 6.1. Предположим, что N = 10 000 000, термины запросов и документов взвешены по логарифмической схеме (wf), термины запроса — с использованием схемы idf, а к векторам документов применена косинусная нормировка. Слово And является стоп-словом. Подсчитайте количество терминов и заполните столбцы tf. Чему равна итоговая мера сходства?

Таблица 6.1. Вычисление косинусной меры сходства в упражнении 6.19.

	Запрос					Документ			
	tf	wf	Df	idf	q_i = wf-idf	tf	wf	d_i = нормированный wf	(q_i, d_i)
Digital			10 000						
Video			100 000						
cameras			50 000						

Упражнение 6.20. Покажите, что по запросу affection ранжирование трех документов на рис. 6.13 является обратным по отношению к ранжированию по запросу jealous gossip.

Упражнение 6.21. Преобразовывая запрос в единичный вектор на рис. 6.13, мы приписывали всем терминам запроса одинаковые веса. Какие другие подходы можно предложить для этой задачи?

Упражнение 6.22. Рассмотрим вариант, в котором термин запроса не принадлежит множеству M индексированных терминов; таким образом, стандартное конструирование итогового вектора запроса приводит к вектору $\vec{V}(q)$, не принадлежащему векторному пространству, образованному по этой коллекции. Как адаптировать представление векторного пространства для обработки этого случая?

Упражнение 6.23. Проанализируйте величины tf и idf для четырех терминов и трех документов из упражнения 6.10. Найдите два документа с наибольшей релевантностью по отношению по запросу best car insurance для каждой из следующих схем: 1) `nnn.atc` и 2) `ntc.atc`.

Упражнение 6.24. Предположим, что слово coyote не встречается в коллекции, использованной в упражнениях 6.10 и 6.23. Как вычислить показатели `ntc.atc` документов по отношению к запросу coyote insurance?

6.5. Библиография и рекомендации для дальнейшего чтения

В главе 7 более углубленно рассматриваются вычислительные аспекты ранжирования в векторном пространстве. В работах Луна (Luhn, 1957, 1958) сообщается о первых приложениях взвешивания терминов. В них подчеркнута важность терминов со средней частотой (термины, которые встречаются ни редко, ни часто), их можно рассматривать как предтечу схемы схемы tf–idf и аналогичных подходов. Спарк Джоунс (Spärck Jones, 1972) на основе этих интуитивных предположений провела эксперименты, продемонстрировавшие использование обратной документной частоты в схеме взвешивания терминов. Развитие и теоретическое обоснование схемы idf было продолжено в работах Солтона и Бакли (Salton and Buckley, 1987), Робертсона и Джоуса (Robertson and Jones, 1976), Крофта и Харпера (Croft and Harper, 1979) и Папинени (Papineni, 2001). Робертсон поддерживает веб-страницу (www.soi.city.ac.uk/~ser/idf.html), посвященную истории схемы idf, включая электронные копии ранних публикаций, которые предшествовали электронным версиям журнальных статей. Метод опорной нормировки длины документа разработали Сингал и др (Singhal et al., 1996*a*). В главе 11 описаны вероятностные языковые модели, на основе которых разрабатываются более тонкие схемы взвешивания, чем схема tf–idf (см. раздел 11.4.3).

Как мы выяснили, в результате назначения веса каждому термину в документе последний можно интерпретировать как вектор весов всех терминов коллекции. Информационно-поисковая система SMART в университете Корнелла (Salton, 1971*b*) была разработана Солтоном и его коллегами и, вероятно, впервые использовала представление документа как вектора весов. Схема для вычисления косинусных мер сходства (раздел 6.3.3) — заслуга Цобеля и Моффата (Zobel and Moffat, 2006). Две стратегии ранжирования, “термин за термином” и “документ за документом”, обсуждаются в работе Тертла и Флуда (Turtle and Flood, 1995).

Система обозначений SMART в схеме взвешивания терминов tf–idf, продемонстрированная на рис. 6.15, была предложена в работах Солтона и Бакли, а также Сингала и др. (Salton and Buckley, 1988; Singhal et al., 1995, 1996*b*). Не все варианты обозначений являются согласованными; мы старались придерживаться системы, предложенной Сингалом и др. (Singhal et al., 1996*b*). Более подробная и исчерпывающая система обозначений была разработана Моффатом и Цобелем (Moffat and Zobel, 1998), предложившими более широкую палитру схем взвешивания на основе частот и документных частот терминов. Помимо системы обозначений, Моффат и Цобель (Moffat and Zobel, 1998) стремились определить пространство весовых функций, позволяющее построить эффективные схемы взвешивания путем поиска экстремума. Однако они сообщили, что эти методы поиска экстремума не позволяют определить наилучшие схемы взвешивания.

Глава 7

Ранжирование в полнофункциональной поисковой системе

В главе 6 была рассмотрена теория, лежащая в основе взвешивания терминов в документах для их ранжирования. Она привела нас к построению модели векторного пространства и базового алгоритма для вычисления косинусной меры сходства, описанного в разделе 6.3.3. В данной главе мы начинаем с эвристических приемов, позволяющих ускорить эти вычисления (раздел 7.1). Многие из этих приемов повышают скорость поиска за счет риска не найти лучшие K документов, соответствующих запросу. Некоторые эвристические методы обобщают метод ранжирования с использованием косинусной меры сходства. В разделе 7.1 описаны практически все компоненты, необходимые для создания полноценной поисковой системы. Поэтому мы делаем шаг назад от вычисления косинусной меры сходства к решению более общей задачи ранжирования в поисковых системах. Раздел 7.2 содержит схему полноценной поисковой системы, включая индексы и структуры для поддержки ранжирования не только на основе косинусной меры сходства, но и с помощью более общих факторов ранжирования, например близости терминов запроса. Как все это работает в комплексе, показано в разделе 7.2.4. В разделе 7.3 обсуждаются вопросы взаимодействия модели векторного пространства для свободных текстовых запросов с поддержкой популярных операторов языка запросов.

7.1. Эффективное ранжирование

Вернемся к алгоритму, представленному на рис. 6.14. Относительно запросов вида (“ку”) q = `jealous gossip` можно сразу сделать два замечания.

1. Единичный вектор $\vec{v}(q)$ содержит только два ненулевых компонента.
2. Если не применять взвешивание терминов запроса, то эти ненулевые компоненты окажутся одинаковыми и равными 0,707.

Для ранжирования документов, соответствующих этому запросу, нам нужны относительные (а не абсолютные) веса документов в коллекции. Для этого достаточно вычислить косинусную меру сходства между каждым единичным вектором документа $\vec{v}(d)$ и вектором $\vec{v}(q)$ (в котором все ненулевые компоненты вектора запроса приравнены к единице), а не единичным вектором $\vec{v}(q)$. Для любых двух документов d_1 и d_2 справедливо следующее утверждение.

$$\left(\vec{V}(q),\vec{v}(d_1)\right) > \left(\vec{V}(q),\vec{v}(d_2)\right) \Leftrightarrow \left(\vec{v}(q),\vec{v}(d_1)\right) > \left(\vec{v}(q),\vec{v}(d_2)\right) \tag{7.1}$$

Для любого документа d косинусная мера сходства $(\vec{v}(q),\vec{v}(d))$ представляет собой взвешенную сумму по всем терминам запроса q, встречающимся в документе d. Ее можно вычислить, определив пересечение словопозиций (postings) точно так же, как в алгоритме, показанном на рис. 6.14, где строку 7 следует заменить, поскольку в этом случае вес $w_{t,q}$ полагается равным единице, так что операции умножения и сложения превра-

щаются просто в операцию сложения. Результат этой модификации приведен на рис. 7.1. Необходимо пройти по словопозициям терминов запроса q в инвертированном индексе и накопить итоговую релевантность для каждого документа — так же, как и при обработке булева запроса, за исключением того, что каждому документу, появляющемуся в любой из просматриваемых записей, теперь приписывается положительная величина – значение релевантности. Как указывалось в разделе 6.3.3, для каждого словарного термина мы храним его обратную документную частоту idf, а для каждой (например, некоординатной) словопозиции — частоту термина tf. Эта схема позволяет ранжировать каждый документ, встречающийся в словопозициях любого из терминов запроса; общее количество таких документов может быть значительно меньше N.

Вычислив релевантность, мы должны сделать последний шаг до того, как выдать пользователю список результатов, — определить K документов с наибольшей релевантностью. Несмотря на то что можно было бы отсортировать по релевантности все документы, лучше все же использовать частичную пирамидальную сортировку и извлечь только K лучших документов. Пусть J — количество документов с ненулевыми косинусными мерами сходства. Тогда для создания самой пирамиды (кучи) потребуется $2J$ сравнений, а для извлечения из кучи каждого из K документов с наибольшей релевантностью достаточно еще $\log J$ сравнений.

```
CosineScore(q)
1     float Scores[N] = 0
2     for each d
3     Инициализация Length[d] длиной документа d
4     for each термин запроса t
5     do извлечь инвертированный список для термина t
6         for each пары (d, tf_{t,d}) в инвертированном списке
7         do Scores[d] += wf_{t,d}
8     Считать массив Length[d]
9     for each d
10    do Scores[d] = Scores[d]/Length[d]
11    return первые K компонентов массива Scores[]
```

Рис. 7.1. Ускоренное ранжирование в векторном пространстве

7.1.1. Неточный поиск лучших K документов

До сих пор мы уделяли внимание поиску ровно K лучших документов по запросу. Теперь рассмотрим схемы, позволяющие создать список K документов, *которые, вероятно,* относятся к K лучшим документам, соответствующим запросу. Это должно резко снизить стоимость поиска таких K документов без существенной потери релевантности результатов с точки зрения пользователя. Следовательно, в большинстве приложений достаточно найти K документов, релевантность которых мало отличаются от наилучших. В последующих разделах мы детализируем схемы поиска таких документов и покажем, что они позволяют избежать ранжирования бо́льшей части документов из коллекции.

С точки зрения пользователя такой приблизительный поиск K документов не всегда плох. В любом случае нет никаких гарантий, что K документов, определенных на основе

косинусной меры сходства, на самом деле лучше всего соответствуют запросу: косинусная мера сходства сама по себе лишь приблизительно оценивает релевантность документа запросу. В разделах 7.1.2–7.1.6 мы опишем эвристические приемы, с помощью которых можно найти K документов, релевантность которых, определенные на основе косинусной меры сходства, близки к релевантности лучших K документов. Основные вычислительные затраты обусловлены необходимостью определения косинусных мер сходства между запросом и большим количеством документов. Кроме того, при большом количестве документов с близкими значениями релевантности стоимость поиска K наилучших документов алгоритмом частичной пирамидальной сортировки также увеличивается. Рассмотрим ряд идей, позволяющих удалить из рассмотрения большое количество документов, не вычисляя для них косинусную меру сходства. Эти эвристические приемы выполняются в два этапа.

1. Найдем множество A, состоящее из документов, являющихся кандидатами на включение в список, где $K < |A| << N$. Совершенно не обязательно, чтобы это множество содержало K документов с наибольшей релевантностью запросу q, но в нем должно быть достаточно много документов, релевантность которых близка к первым K документам.
2. Вернем K документов, имеющих наибольшую релевантность в множестве A.

Из этого описания ясно, что такая схема подразумевает использование параметров, которые необходимо настраивать с учетом коллекции и конкретного приложения; ссылки на практические примеры такой настройки содержатся в конце главы. Кроме того, следует подчеркнуть, что большинство этих эвристических приемов предназначены для обработки свободных текстовых запросов, а не булевых или фразовых запросов.

7.1.2. Сокращение индекса

Очевидно, что для запроса q, состоящего из многих терминов, достаточно рассмотреть документы, содержащие по крайней мере один из этих терминов. Мы можем использовать это обстоятельство, дополнив его эвристическими правилами.

1. Рассмотрим только документы, содержащие термины, значение idf которых превосходит заданный порог. Таким образом, при обходе словопозиций достаточно просмотреть только термины с высоким значением idf. Это дает значительный выигрыш: инвертированные списки для терминов с низкими значениями idf, как правило, очень длинные; исключив их из рассмотрения, можно значительно сократить список документов, для которых вычисляется косинусная мера сходства. Этот эвристический прием можно интерпретировать так: термины с низким значением idf рассматриваются как стоп-слова и не вносят вклад в релевантность документа. Например, для запроса `catcher in the rye` достаточно обойти только словопозиции терминов `catcher` и `rye`. Разумеется, порог отсечения можно сделать зависимым от запроса.
2. Рассмотрим только документы, содержащие многие (в специальном случае — все) термины запроса. Это можно сделать в процессе обхода словопозиций; релевантность вычисляется только для документов, содержащих все (или многие)

термины запроса.[1] Недостаток этой схемы заключается в том, что, требуя, чтобы в документе были представлены все (или даже многие) термины запроса до вычисления их косинусной меры сходства, мы можем получить в результате меньше K документов-кандидатов. Эта проблема будет рассмотрена в разделе 7.2.1.

7.1.3. Чемпионский список

Идея *чемпионских списков* (champion lists), который иногда называют также *списками фаворитов* (fancy lists) или *списками top-документов* (top docs), заключается в предварительном определении для каждого словарного термина t набора из r документов с наибольшими весами по отношению к термину t. При этом величина r выбирается заранее. В схеме взвешивания tf-idf эти r документов имеют наибольшие значения tf по отношению к термину t. Этот набор из r документов называется *чемпионским списком* для термина t.

Теперь, имея запрос q, мы можем создать множество A следующим образом: объединим списки чемпионов для каждого термина, содержащегося в запросе q. Вычисление косинусной меры сходства теперь ограничивается только документами множества A. Критический параметр к этой схеме — значение r, сильно зависящее от приложения. Интуитивно ясно, что величина r должна быть большой по сравнению с K, особенно если используется любая форма сокращения индекса, описанная в разделе 7.1.2. Одна из проблем заключается в том, что величина r определяется в момент построения индекса, в то время как число K зависит от приложения и может оставаться неопределенным вплоть до момента, пока не будет получен запрос q. В результате мы можем (как и в случае сокращения индекса) обнаружить, что множество A содержит меньше K документов. Нет никаких причин полагать число r одинаковым для всех терминов в словаре; например, для более редких терминов его можно установить на более высоком уровне.

7.1.4. Статический ранг и упорядочение индекса

Разовьем дальше идею чемпионских списков в направлении более общей идеи о статическом ранге. Во многих поисковых системах существует возможность вычислить меру качества $g(d)$ для каждого документа d, не зависящую от запроса и потому являющуюся *статической* (static). Эту меру качества можно рассматривать как число, лежащее между нулем и единицей. Например, в контексте новостных статей в сети веб ранг $g(d)$ может быть определен по количеству благожелательных отзывов пользователей о статье. Более подробное обсуждение этой темы содержится в разделе 4.6, а также — в контексте веб-поиска — в главе 21.

---

[1] Следует заметить, что и при обработке **текстовых запросов в свободной форме** в **модели поиска с ранжированием** (обсуждаемого в последующих главах) требования к производительности поиска накладывают запрет на ранжирование максимально полного списка документов, получаемого в режиме OR. Поэтому описываемый здесь компромисс является актуальным предметом изучения в практике информационного поиска. Часто применяются техники, предполагающие включение в список для последующего ранжирования только тех документов, которые по сумме весов содержащихся в них слов запроса превосходят некоторый порог, называемый *кворумом*. В этом случае первая фаза поиска, на которой документы признаются достойными включения в список для ранжирования, называется *фазой фильтрации*, а вторая, собственно ранжирующая, — *фазой ранжирования*. Вопрос выбора порога для кворума и других деталей этого процесса лежит за рамками данного учебника. — *Примеч. ред.*

Итоговая релевантность документа d является комбинацией статического показателя $g(d)$ с динамической релевантностью, зависящей от запроса, например вычисленным по формуле (6.12). Точную комбинацию можно определить с помощью методов машинного обучения, описанных в разделе 6.1.2 и рассматриваемых далее в разделе 15.4.1. Однако для ясности изложения пока будем считать, что он вычисляется как простая сумма

$$\text{Конечный ранг } (q,d) = q(d) + \frac{\left(\vec{V}(q),\vec{V}(d)\right)}{\left\|\vec{V}(q)\right\| \bullet \left\|\vec{V}(d)\right\|} \tag{7.2}$$

В этом простом виде статический ранг $g(q)$ и релевантность из формулы (6.10), зависящий от запроса, вносят одинаковые вклады от нуля до единицы. Возможны и другие варианты относительного взвешивания; эффективность этого эвристического приема зависит от конкретной схемы.

Сначала рассмотрим упорядочение документов в инвертированном списке для каждого термина по убыванию $g(d)$[2]. Это позволяет выполнить алгоритм пересечения списков, представленный на рис. 1.6. Для того чтобы определить пересечение за один проход по словопозициям каждого термина запроса, в алгоритме, представленном на рис. 1.6, используются словопозиции, упорядоченные по идентификаторам документов. Однако на самом деле нам требуется лишь, чтобы все словопозиции были упорядочены единообразно; в данном случае для упорядочения мы используем значения $g(d)$. Эта схема продемонстрирована на рис. 7.2, где инвертированные списки составлены в порядке убывания значений $g(d)$.

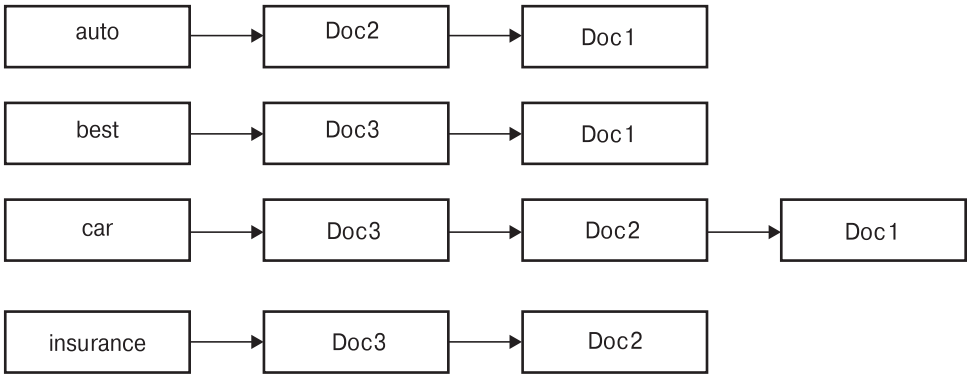


Рис. 7.2. Индекс, упорядоченный по статическому рангу. В данном примере мы предполагаем, что статические ранги документов Doc1, Doc2 и Doc3 равны $g(1) = 0{,}25$, $g(2) = 0{,}5$ *и* $g(3) = 1$

Следующая идея представляет собой непосредственное расширение чемпионских списков. Для тщательно выбранного значения r и для каждого термина t мы храним *глобальный чемпионский список* (global champion list), состоящий из r документов, имеющих наибольшие значения $g(d) + \text{tf-idf}_{t,d}$. Этот список упорядочен так, как и все инвертированные списки (либо по идентификаторам документов, либо по статическим рангам). Тогда в момент запроса остается только вычислить итоговую релевантность (7.2) для документов, принадлежащих объединению этих глобальных чемпионских списков. Интуитивно ясно, что это выведет на первый план документы с большими итоговой релевантностью.

[2] На практике это всего лишь означает, что в момент создания поискового индекса идентификаторы документов следует перенумеровать по возрастанию в соответствии с убыванием $g(d)$. — *Примеч. ред.*

Завершим обсуждение глобальных чемпионских списков следующей идеей. Для каждого термина t будем хранить два инвертированных списка, состоящих из непересекающихся множеств документов и упорядоченных по убыванию величины $g(d)$. Первый список, который называется *верхним* (high), состоит из m документов, имеющих наибольшие значения tf для термина t. Второй список, который называется *нижним* (low), содержит все другие документы, содержащие термин t. Обрабатывая запрос, мы сначала сканируем только верхние списки терминов запроса, вычисляя итоговую релевантность документов, содержащихся в верхних списках для всех (или бо́льшей части) терминов запроса. Если в ходе этого процесса нам удастся получить релевантность K документов, то алгоритм останавливается. Если нет, то мы вычисляем релевантность документов из нижних списков. Далее эта идея будет развита в разделе 7.2.1.

7.1.5. Упорядочение по важности

Во всех инвертированных списках, описанных ранее, документы были упорядочены в соответствии с некоторым общим критерием: как правило, по идентификаторам документов, хотя в разделе 7.1.4 мы использовали статические показатели качества. Как указывалось в конце раздела 6.3.3, такое упорядочение позволяет осуществлять параллельный обход инвертированных списков для всех терминов запросов, ранжируя каждого обнаруженного документа. Этот процесс иногда называют *ранжированием "документ за документом"* (document-at-a-time scoring). Теперь опишем метод неточного поиска наилучших K документов, в котором не все словопозиции упорядочены одинаково, что препятствует осуществлению параллельного обхода. Следовательно, необходимы аккумуляторы для накопления релевантности термин за термином как в схеме на рис. 6.14. Такая схема называется *ранжированием "термин за термином"* (term-at-a-time scoring).

Идея заключается в том, чтобы документы d в инвертированном списке для термина t были расположены в порядке убывания значения $\mathrm{tf}_{t,d}$ (частоты термина t в документе d). Следовательно, упорядочение документов изменяется от одного инвертированного списка к другому, и мы не можем ранжировать их в ходе параллельного обхода инвертированных списков для всех терминов запроса. Если инвертированные списки составлены в порядке убывания значения $\mathrm{tf}_{t,d}$, то существуют два способа значительно снизить количество документов, для которых аккумулируется релевантность: 1) при обходе инвертированного списка для термина запроса t мы просматриваем только начальный участок списка либо после того, как будет просмотрено фиксированное количество документов r, либо после того, как значение $\mathrm{tf}_{t,d}$ опустится ниже заданного порога; 2) аккумулируя релевантность во внешнем цикле алгоритма, описанного на рис. 6.14, мы рассматриваем термины запросов в порядке убывания значения idf, так что термины запроса, которые могут внести бо́льший вклад в итоговую релевантность, рассматриваются первыми. Последняя идея допускает адаптацию в момент обработки запроса: обнаружив термин запроса с небольшим значением idf, мы можем определить, следует ли продолжить обработку на основе изменения релевантности документов, определенных в ходе обработки предыдущего термина запроса. Если эти изменения являются минимальными, то мы можем отказаться от аккумуляции релевантности для оставшихся терминов запроса или обрабатывать более короткие начальные участки их инвертированных списков.

Эти идеи представляют собой обобщение методов, описанных в разделах 7.1.2–7.1.4. Мы можем также реализовать вариант статического упорядочения, в котором каждый инвертированный список упорядочен на основе аддитивной комбинации статических рангов и динамических показателей, зависящих от запроса. В этом случае согласован-

ность упорядочения инвертированных списков также нарушается и, следовательно, термины запроса необходимо обрабатывать поочередно, накапливая релевантность для всех просмотренных документов. В зависимости от конкретных функций ранжирования инвертированный список для документа можно упорядочить в соответствии с другими показателями, отличающимися от частоты термина; эта более общая схема называется *упорядочением по важности* (impact ordering).

7.1.6. Отсечение кластеров

В методе *отсечения кластеров* (cluster pruning) предусмотрен предварительный этап, на котором происходит кластеризация векторов документов. Затем, в момент обработки запроса, мы будем рассматривать только документы, принадлежащие небольшому количеству кластеров, являющихся кандидатами для ранжирования, основанных на косинусной мере сходства. Конкретнее, предварительный (до выполнения запроса) этап выглядит так.

1. Случайным образом извлекаем из коллекции $\sqrt{N}$ документов. Назовем их *ведущими* (leaders).
2. Для каждого документа, не являющегося ведущим, находим ближайший к нему ведущий документ.

Документы, не являющиеся ведущими, называются *ведомыми* (follower). Интуитивно ясно, что при разделении ведомых, возникающем при случайном извлечении $\sqrt{N}$ ведущих документов, ожидаемое количество ведомых документов для каждого ведущего документа примерно равно $N/\sqrt{N} = \sqrt{N}$. Затем, на этапе обработки запроса, происходит следующее.

1. По запросу q найдем ведущий документ L, ближайший к запросу q. Для этого необходимо вычислить косинусные меры сходства запроса q с каждым из $\sqrt{N}$ ведущих документов.
2. Множество кандидатов A состоит из документа L и его ведомых документов. Основываясь на косинусной мере сходства, ранжируем все документы из множества кандидатов.

Использование случайно выбранных ведущих документов для кластеризации работает быстро и с некоторой вероятностью отражает распределение векторов документов в векторном пространстве. Область векторного пространства, плотно заполненная документами, вероятно, породит несколько лидеров, что приведет к более мелкому разбиению на подобласти (рис. 7.3).

Введя дополнительные параметры b_1 и b_2, являющиеся положительными целыми числами, можно получить разные варианты отсечения кластеров. На этапе предварительной обработки мы свяжем каждый ведомый документ с его b_1 ближайшими ведущими документами, а не просто с единственным ближайшим ведущим документом. В момент поступления запроса мы рассмотрим b_2 ведущих документов, ближайших к запросу q. Очевидно, что основная схема соответствует случаю $b_1 = b_2 = 1$. Кроме того, увеличивая параметры b_1 и b_2, мы увеличиваем вероятность обнаружения K документов, которые с большей вероятностью принадлежат списку истинных K документов с максимальной релевантностью. Правда, при этом увеличивается объем вычислений. В главе 16 мы еще вернемся к этому методу, когда будем описывать кластеризацию.

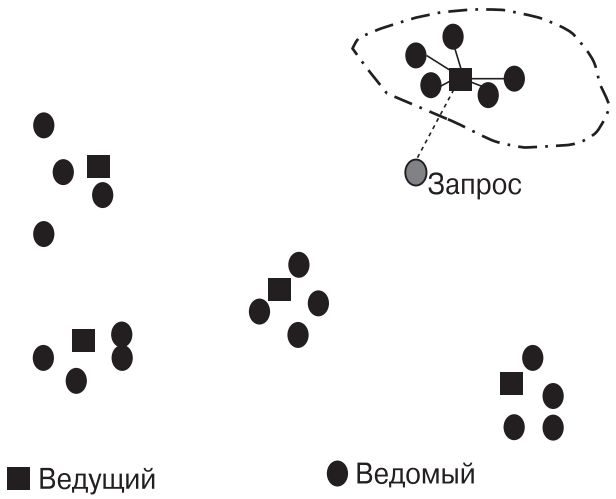


Рис. 7.3. Отсечение кластеров

? ***Упражнение 7.1.*** Мы предполагали (см. рис. 7.2), что словопозиции для упорядочения по статическому рангу отсортированы в порядке убывания ранга $g(d)$. Почему мы выбрали убывающий, а не возрастающий порядок?

Упражнение 7.2. Обсуждая чемпионские списки, мы просто использовали r документов с наибольшими значениями tf для термина t. Однако, рассматривая глобальные чемпионские списки, мы также использовали idf, идентифицируя документы с наибольшими значениями величины $g(d)$ + tf-idf$_{t,d}$. Почему мы разделяем эти два варианта?

Упражнение 7.3. Допустим, что запросы содержат только один термин. Объясните, почему для нахождения K документов с наибольшей релевантностью достаточно использовать глобальные чемпионские списки с $r = K$ документами. Как выглядит простая модификация этой идеи, если запросы состоят только из s терминов, где s — фиксированное целое число, которое больше единицы?

Упражнение 7.4. Объясните, почему общее глобальное упорядочение в соответствии со значениями $g(d)$ во всех верхних и нижних списках позволяет повысить эффективность ранжирования.

Упражнение 7.5. Еще раз проанализируйте данные из упражнения 6.23 для схемы `nnn.atc` в ранжировании, зависящем от запроса. Допустим, что статический ранг документа Doc1 равен единице, а статический ранг документа Doc2 равен двум. С помощью равенства (7.2) определите диапазоны статического ранга документа Doc3, при которых он оказывается на первом, втором и третьем местах при запросе best car insurance.

Упражнение 7.6. Нарисуйте словопозиции, упорядоченные по частоте, для данных, показанных на рис. 6.9.

Упражнение 7.7. Допустим, что статические ранги документов Doc1, Doc2 и Doc3 равны 0,25, 0,5 и 1. Нарисуйте словопозиции для упорядочения по важности,

если каждый инвертированный список упорядочен в соответствии с суммой статического ранга и нормализованных значений tf (см. рис. 6.10).

Упражнение 7.8. Задача о ближайшем соседе на плоскости заключается в следующем. Дано множество, состоящее из N точек на плоскости; на этапе предварительной обработки создается такая структура данных, что по запросу Q находится одна из N точек, ближайшая к запросу Q по евклидовому расстоянию. Очевидно, что в качестве одного из методов решения этой задачи можно использовать отсечение кластеров, позволяющее избежать вычисления евклидового расстояния от запроса Q до каждой их точек. Приведите простой пример точек на плоскости, в котором есть две ведущие точки, и ответ, полученный с помощью метода отсечения кластеров, является неправильным (т.е. он не является точкой, ближайшей к запросу Q).

7.2. Компоненты информационно-поисковой системы

В этом разделе мы объединим идеи, изложенные выше, для описания простейшей поисковой системы, извлекающей и ранжирующей документы. Сначала мы разовьем идеи взвешивания документов, которые выходят за пределы векторных пространств. После этого мы объединим все эти элементы для схематического описания полной системы. Поскольку мы рассматриваем полнофункциональную систему, в этом разделе мы не будем ограничиваться только векторной моделью информационного поиска. Действительно, наша поисковая система может обеспечивать реализацию не только методов векторного пространства, но и другие способы обработки запросов и виды поиска. В разделе 7.3 мы вернемся к вопросу о том, как запросы, интерпретируемые, как элементы векторного пространства, взаимодействуют с другими операторами запроса.

7.2.1. Многоуровневые индексы

Как указано в разделе 7.1.2, при использовании эвристических приемов, таких как сокращение индекса для неточного поиска K лучших документов может оказаться, что множество A содержит меньше K документов. Как правило, в этих ситуациях используются *многоярусные индексы* (tiered indexes)[3], которые можно рассматривать, как обобщение чемпионских списков. Эта идея проиллюстрирована на рис. 7.4, на котором продемонстрированы документы и термины, показанные на рис. 6.9. В данном примере пороговое значение tf для первого яруса установлено равным 20, для второго яруса — 10. Это значит, что индекс первого яруса содержит только словопозиции, у которых значение tf превышает 20, а индекс второго яруса содержит только словопозиции, у которых значение tf превышает 10. В данном примере словопозиции внутри яруса упорядочены по идентификаторам документов.

[3] В отечественной литературе иногда применяются термины “эшелонирование” и “эшелон” вместо “многоярусный” и “ярус”. — *Примеч. ред.*

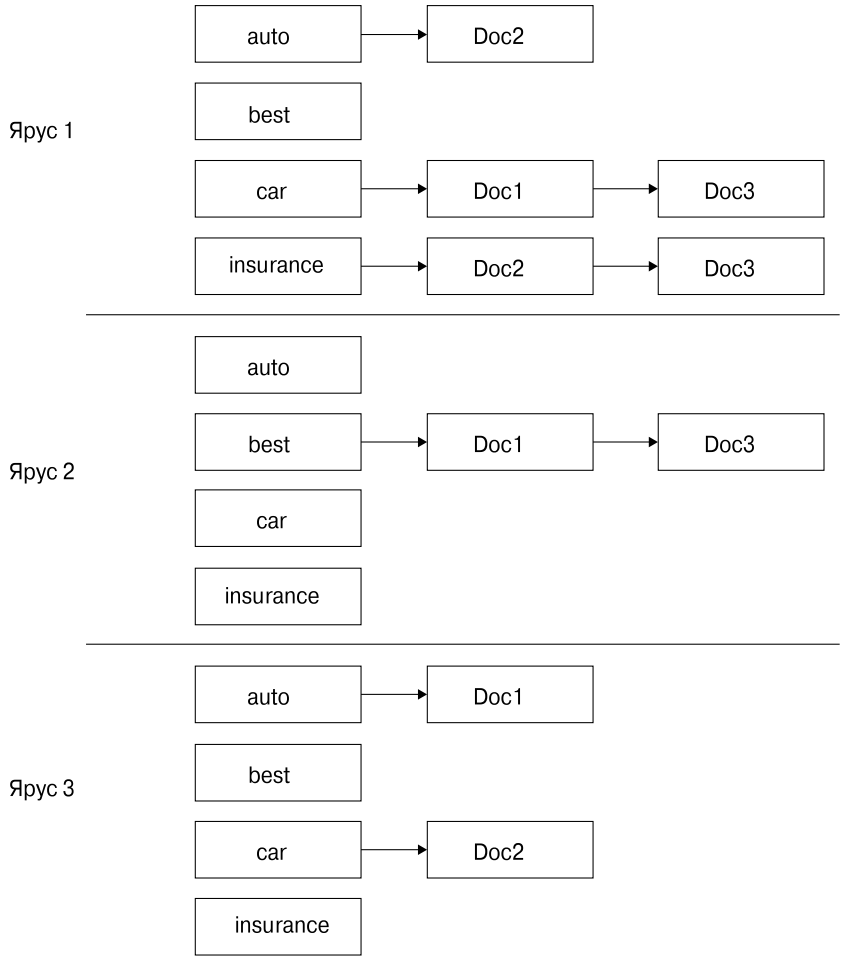


Рис. 7.4. Многоярусные индексы. Если на первом уровне не удается получить K результатов, обработка запроса "откатывается" на второй ярус и т.д. На каждом ярусе словопозиции упорядочены по идентификаторам документов

7.2.2. Близость терминов запроса

Пользователи предпочитают документы, в которых все или большинство терминов запроса расположены близко друг к другу, поскольку это является свидетельством того, что документ направлен на цели запроса. Это утверждение особенно справедливо для пользователей сети веб (см. главу 19). Рассмотрим запрос, содержащий несколько терминов t_1, t_2, ..., t_k. Обозначим через ω ширину наименьшего окна в документе d, содержащего все термины запроса. Ширину этого окна будем измерять количеством слов в окне. Например, если документ состоит только из предложения `The quality of mercy is not strained`, то ширина наименьшего окна для запроса `strained mercy` равна четырем. Интуитивно ясно, что чем меньше ширина ω, тем лучше документ d соответствует запросу. Если документ содержит не все термины запроса, то вели-

чину ω можно установить равной какому-нибудь очень большому числу[4]. Кроме того, можно рассмотреть варианты, когда стоп-слова не участвуют в вычислениях. Такие функции ранжирования, основанные на близости терминов запроса в документе (proximity-weighted scoring functions), отличаются от чистой косинусной меры сходства и больше похожи на семантику "мягкого логического И", которая, очевидно, используется в Google и других поисковых системах.

Как же придумать такую функцию ранжирования на основе близости, зависящую от величины ω? Проще всего использовать подход "закодировать вручную", описанный в разделе 7.2.3. Более масштабируемый подход основан на материале, изложенном в разделе 6.1.2. В этом случае величина ω рассматривается как еще один численный признак (фактор) ранжирования, важность которого определяется в ходе машинного обучения (см. раздел 15.4.1).

7.2.3. Парсер запроса и ранжирование

Обычно интерфейсы поисковых систем, особенно систем, ориентированных на массового пользователя в сети веб, стараются замаскировать операторы запроса от конечного пользователя, чтобы скрыть сложность этих операторов от неподготовленной аудитории, поощряя ее вводить *свободные текстовые запросы* (free text queries). Как же поисковая система, оснащенная таким интерфейсом и индексами для разных операторов поиска, должна обрабатывать запрос наподобие `rising interest rates`? Или, обобщая вопрос, как учесть все факторы, влияющие на релевантность документа?

Разумеется, ответ зависит от пользовательской аудитории, распределения запросов и коллекции документов. Для преобразования нескольких ключевых слов, набранных пользователем, в запрос с операторами, который поступает в индексы, используется *парсер запросов* (query parser). Иногда эта операция может повлечь за собой выполнение нескольких запросов к различным индексам; например, парсер может построить серию запросов.

1. Сначала запрос пользователя обрабатывается, как фразовый. Затем он ранжируется с помощью метода векторного пространства, который интерпретирует запрос, как вектор, состоящий из трех терминов: `rising interest rates`.
2. Если фраза `rising interest rates` содержится меньше чем в десяти документах, то генерируются два двусловных запроса: `rising interest` и `interest rates`. Документы снова ранжируются на основе модели векторного пространства.
3. Если у нас по-прежнему меньше десяти результатов, то генерируются три однословных запроса, которые также ранжируются на основе модели векторного пространства.

На каждом из этих этапов (если они выполняются) возникает список оцененных документов, для каждого из которых необходимо вычислить итоговую релевантность. Она может учитывать релевантность, определенную по модели векторного пространства, статический ранг, меру близости и, возможно, другие факторы, поскольку документ может быть включен в списки найденных документов на разных этапах. Необходима агреги-

[4] Иными словами, поисковая система может отбирать документы, основываясь на факте попадания минимального подмножества наиболее важных слов запроса в некоторое окно текста. — *Примеч. ред.*

рующая функция ранжирования, *накапливающая свидетельства* релевантности документов, полученные из разных источников. Как разработать такой парсер запроса и как создать такую агрегирующую функцию ранжирования?

Ответ зависит от ситуации. Во многих корпоративных системах конечное приложение строится с использованием набора доступных операторов ранжирования и гибкого парсера запроса: функции ранжирования и парсер настраиваются вручную. Для настройки используются имеющиеся в коллекции зоны, метаданные и знания о типичных документах и запросах. В корпоративных приложениях значительные изменения коллекции и свойств запросов обычно редки и связаны с такими событиями, как внедрение нового формата документов или системы документооборота, а также слияние с другой фирмой. Иначе обстоит дело при веб-поиске, который связан с непрерывно изменяющейся коллекцией документов, в которой новые факторы возникают постоянно. Кроме того, количество факторов, влияющих на релевантность, может достигать нескольких сотен, что делает ручной подбор параметров сложной задачей. Для решения этой проблемы обычно используются методы машинного обучения, расширяющие идеи, изложенные в разделе 6.1.2, которые мы разовьем в разделе 15.4.1.

7.2.4. Соберем все это вместе

Итак, мы изучили все компоненты, необходимые для типичной поисковой системы, поддерживающей свободные текстовые и булевы запросы, а также запросы по зонам и полям. Кратко рассмотрим сборку все этих частей в единую систему (рис. 7.5).

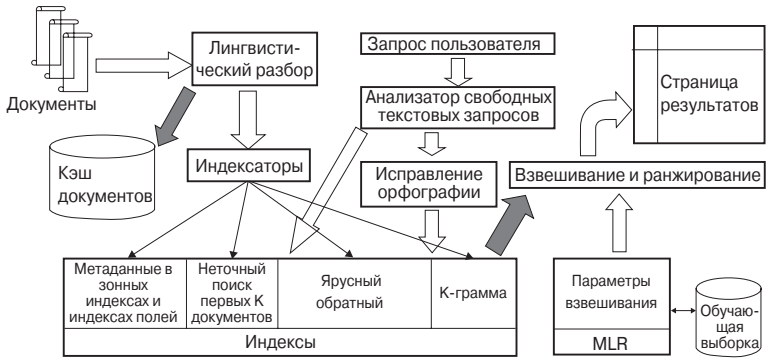


Рис. 7.5. Полная поисковая система. Потоки данных показаны в основном для свободных текстовых запросов

На этом рисунке потоки документов поступают на грамматический парсер и в подсистему лингвистической обработки (определение языка и формата, разбиение на лексемы и стемминг). Итоговый поток лексем подается на два модуля. Во-первых, мы сохраняем копию каждого проанализированного документа в кэше документов. Это позволяет нам генерировать сниппеты (snippet) документа — фрагменты текста, сопровождающие каждый документ в списке результатов поиска. Они предназначены для краткого объяснения пользователю, почему этот документ соответствует заданному запросу. Генерация сниппетов описана в разделе 8.7. Вторая копия лексем поступает в модуль индексаторов, создающих группу индексов, включая индексы по зонам и полям, хранящие метаданные для каждого документа, (многоярусные) позиционные индексы, индексы для исправления орфографических ошибок и обеспечения нечеткого поиска, а также структуры для

ускорения приближенного поиска K лучших документов. Свободный текстовый запрос пользователя (вверху в центре рисунка) посылается индексам как непосредственно, так и через модуль для генерации кандидатов на исправление орфографических ошибок. Как указано в главе 3, этот модуль может вызываться при необходимости, только если по оригинальному запросу найдено мало документов. Найденные документы (черная стрелка) поступают в модуль ранжирования на основе машинного обучения (machine-learned ranking). Этот модуль строится по принципам, изложенным в разделе 6.1.2 (мы развиваем эти идеи в разделе 15.4.1). В итоге ранжированные документы отображаются на странице результатов.

? ***Упражнение 7.9.*** Объясните, как адаптировать алгоритм пересечения инвертированных списков, описанный в разделе 1.3, для определения наименьшего целого числа ω, содержащего все термины запроса.

Упражнение 7.10. Адаптируйте эту процедуру для ситуации, в которой не все термины запроса представлены в документе.

7.3. Влияние операторов языка запросов на ранжирование в векторном пространстве

Мы ввели в рассмотрение модель векторного пространства как парадигму для свободных текстовых запросов. Завершим главу обсуждением того, как ранжирование на основе модели векторного пространства связана с операторами запроса, рассмотренными в предыдущих главах. Эта связь должна исследоваться на двух уровнях: на уровне выразительных возможностей языка запросов, которыми может воспользоваться опытный пользователь, и на уровне индекса, применяемого в разных методах поиска. При создании поисковой системы мы можем принять решение поддерживать различные операторы запроса для конечного пользователя. Для этого необходимо понять, какие компоненты индекса могут совместно использоваться при выполнении разных операторов запроса, а также как обрабатывать запросы пользователя, в которых смешаны разные операторы.

Ранжирование в векторном пространстве поддерживает так называемый свободный текстовый поиск, в котором запрос рассматривается как набор слов, не содержащий никаких связывающих операторов. Это позволяет ранжировать документы, соответствующие запросу, в отличие от рассмотренных ранее булевых и фразовых запросов, а также запросов с джокерами. Традиционная интерпретация таких “свободных” текстовых запросов когда-то сводилась к поиску документов, содержащих хотя бы один термин запроса. Однако в настоящее время благодаря поисковым системам в вебе, таким как Google, множество терминов, набранных в поле ввода запроса (выглядящего как свободный текстовый запрос), интерпретируется как конъюнктивный запрос, по которому находятся документы, содержащие все или большинство терминов запроса.

Булев поиск

Очевидно, что для обработки булева запроса можно использовать индекс, построенный в модели векторного пространства, поскольку вес термина t в векторе документа d не равен нулю, если термин t содержится в документе d. Обратное утверждение неверно; булев индекс не обязан по умолчанию содержать информацию о весе термина. С точки

зрения пользователя, объединить векторное пространство и булевы запросы непросто. Запросы в векторном пространстве по своей сути представляют собой форму *накопления свидетельств* (evidence accumulation), при которой наличие большого количества терминов в документе повышает его релевантность. С другой стороны, булев поиск требует от пользователя указывать формулу для *выбора* документов с помощью комбинаций ключевых слов без указания какого бы то ни было относительного порядка между ними. С математической точки зрения для сочетания булевых запросов и запросов в векторном пространстве можно применять так называемые *p*-нормы, но нам неизвестны системы, использующие этот факт.

Запросы с джокером

Запросы с джокером и запросы в векторном пространстве требуют разных индексов (за исключением базового уровня), которые можно реализовать с помощью словопозиций и словаря (например, словарь триграмм для шаблонных запросов). Если поисковая система позволяет пользователю применять оператор с джокером в свободном текстовом запросе (например, запрос `rom* restourant`), то компонент запроса с джокером можно реализовать как "размножение" терминов в векторном пространстве (в данном случае двумя такими терминами являются слова `rome` и `roman`) и каждый из которых добавляется в вектор запроса. Таким образом, вектор запроса обрабатывается, а документы сравниваются и ранжируются, как обычно. Следовательно, документ, содержащий оба термина, `rome` и `roma`, скорее всего, будет иметь более высокую релевантность, чем документ, содержащий только одно из этих слов. Разумеется, точное ранжирование зависит от относительных весов каждого термина в найденных документах.

Фразовые запросы

Представление документа в виде вектора в основе своей приводит к потере информации. При кодировании документа в виде вектора теряется относительный порядок терминов в документе. Даже если мы попытаемся интерпретировать каждое двухсловие (biword) как термин (т.е. как ось в векторном пространстве), то веса по разным осям не будут независимыми. Например, фраза `German shepherd` кодируется по оси `german sheperd` и немедленно получает ненулевой вес по осям `german` и `shepherd`. Более того, для двухсловий необходимо как-то распространить такие понятия, как обратная документная частота. Таким образом, индекс, построенный для поиска в векторном пространстве, в принципе, не может быть использован для обработки фразового запроса. Более того, не существует никакого способа ранжирования документов в векторном пространстве относительно фразового запроса — мы знаем лишь относительные веса каждого термина в документе.

При обработке запроса `german shepherd` мы могли бы использовать модель векторного пространства для идентификации документов, в которых эти термины встречаются часто, но не можем учесть порядок их следования. С другой стороны, фразовый поиск позволяет выявить наличие фразы `german shepherd` в документе без указания ее относительной частоты или веса. Несмотря на то что эти две парадигмы поиска (фразовая и векторная) имеют разные реализации на уровне индексов и алгоритмов поиска, в некоторых случаях их комбинация оказывается полезной (см. пример трех этапов анализа запроса в разделе 7.2.3).

7.4. Библиография и рекомендации для дальнейшего чтения

Эвристические приемы предварительной обработки запросов описаны в работах Ана и др. (Anh et al., 2001), Гарсиа и др. (Garcia et al., 2004), Ана и Моффата (Anh and Moffat, 2006*b*), а также Персина и др. (Persin et al., 1996). Отсечение кластеров исследовано Сингитамом и др. (Singitham et al., 2004) и Чиеричетти и др. (Chierichetti et al., 2007). См. также раздел 16.6. Чемпионские списки описаны Персином (Persin, 1994) и (под названием *топ-документы*) Брауном (Brown, 1995). В дальнейшем эта идея развивалась в работах Брина и Пейджа (Brin and Page, 1998), а также Лонга и Сьюэла (Long and Suel, 2003). Несмотря на то что эти эвристические методы хорошо подходят для обработки свободных текстовых запросов, которые можно интерпретировать как векторы, их трудно применить к фразовым запросам. Структура индекса, поддерживающего как взвешенный, так и булев/фразовый поиск, описана в работе Ана и Моффата (Anh and Moffat, 2006*c*). Оценка близости терминов для поиска документов описана в работах Кармела и др. (Carmel et. al, 2001), Кларка и др. (Clarke et al., 2000), а также Сонга и др. (Song et al., 2005). Пионерские работы, посвященные получению функции ранжирования на основе обучения, были выполнены Фуром (Fuhr, 1989), Фуром и Пфайфером (Fuhr and Pfeifer, 1994), Купером и др. (Cooper et al., 1994), Бартеллом и др. (Bartell et al., 1998), а также Коэном и др. (Cohen et al., 1998)

Глава 8

Оценка информационного поиска

В предыдущих главах описано много альтернативных вариантов проектирования систем информационного поиска. Как определить, какой из указанных методов наиболее эффективен в соответствующих приложениях? Следует ли использовать списки стоп-слов? Нужно ли проводить стемминг? Стоит ли применять взвешивание с помощью показателя idf? Информационный поиск представляет собой преимущественно эмпирическую дисциплину, требующую тщательной и осторожной оценки эффективности новых методов на репрезентативных коллекциях документов.

Настоящая глава начинается с обсуждения способов оценки систем информационного поиска (раздел 8.1) и тестовых коллекций, которые наиболее часто используются для этой цели (раздел 8.2). Затем мы введем понятие релевантного и нерелевантного документов и опишем формальный метод оценки неранжированных результатов поиска (раздел 8.3). Для этого перечислим некоторые метрики, которые стандартно используются и при оценке неранжирующего поиска, и в родственных задачах, таких как классификация текстов, а также покажем, почему они являются здесь приемлемыми. Далее мы распространим наши подходы на оценку ранжированных результатов поиска и выработаем для них новые метрики (раздел 8.4). Кроме того, мы обсудим вопросы, связанные с формированием надежных и информативных тестовых коллекций (раздел 8.5).

После этого будут рассмотрены вопросы, связанные с полезностью поиска, а также описано, как это понятие можно заменить релевантностью документа (раздел 8.6). Основным показателем полезности поиска является удовлетворение пользователя, которое, в свою очередь, зависит в том числе и от скорости ответа и размера индекса. При этом вполне естественно предположить, что релевантность результатов является самым важным фактором: быстрые и бесполезные ответы не доставят пользователям удовольствия. Однако мнение пользователей не всегда совпадает с представлениями разработчиков о качестве системы. Например, степень удовлетворения пользователя обычно очень сильно зависит от интерфейса, включая верстку страницы, ясность представления результатов и скорость отклика, хотя эти понятия не связаны с качеством возвращаемых результатов. Мы затронем и другие показатели качества системы, в частности генерацию высококачественных сниппетов, сопровождающих список результатов, которые сильно влияют на удовлетворенность пользователей, но не измеряются в рамках парадигмы ранжирования документов по их релевантности (раздел 8.7).

8.1. Оценка информационно-поисковой системы

Для оценки стандартным способом информационно-поисковой системы по произвольным (“ad hoc”, т.е. “случайным”, “пришедшимся к случаю”) запросам необходима тестовая коллекция, состоящая из трех компонентов.

1. Коллекция документов.
2. Набор тестовых информационных потребностей, выраженных в виде запросов.
3. Набор оценок релевантности, представленных, как правило, в виде бинарных утверждений *релевантный* и *нерелевантный* относительно каждой пары "запрос–документ".

Стандартный подход к оценке информационно-поисковых систем опирается на понятие *релевантных* и *нерелевантных* документов. В соответствии с информационными потребностями пользователей документ из тестовой коллекции проходит бинарную классификацию: релевантный или нерелевантный. Это решение называется *эталонной* (gold standard or ground truth) *оценкой релевантности*. Коллекция тестовых документов и набор информационных потребностей должны иметь достаточный объем: оценку следует усреднять по действительно большой совокупности тестов, поскольку результаты поиска сильно отличаются для разных документов и информационных потребностей. В качестве первого очень грубого приближения достаточным минимумом считается набор из 50 информационных потребностей.

Релевантность оценивается по отношению к информационной потребности, а не по *запросу*. Например, информационная потребность может быть сформулирована так.

> Правда ли, что красное вино более эффективно снижает риск сердечных приступов, чем белое?

Этот вопрос можно преобразовать в запрос.

```
wine AND red AND white AND heart AND attack AND effective
```

Документ является релевантным, если он соответствует заданной информационной потребности, а не просто если он содержит все слова из запроса. На практике эту тонкость часто не понимают, поскольку информационная потребность неочевидна. Тем не менее она существует. Если пользователь задает запрос python поисковой веб-системе, то, возможно, его интересует, где можно купить ручного питона. А может быть, ему нужна информация о языке программирования Python. По однословному запросу системе трудно понять, в чем заключается информационная потребность. Тем не менее пользователю она известна и он может оценивать полученные результаты на основе их релевантности своей информационной потребности. Для того чтобы оценить систему, необходимо явно сформулировать информационную потребность, относительно которой мы будем судить о релевантности или нерелевантности найденных документов. Для простоты можно допустить, что релевантность можно оценить по шкале, т.е. одни документы являются сильно релевантными, а другие — слабо. Однако пока мы будем использовать лишь бинарное решение о релевантности. Причины этого выбора и его альтернативы мы обсудим в разделе 8.5.1.

Многие системы содержат разные веса (часто называемые параметрами), с помощью которых можно настроить их производительность. При оценке таких систем не следует учитывать результаты поиска по тестовой коллекции, полученные путем подбора параметров, обеспечивающих максимум производительности для данной коллекции. Это объясняется тем, что такая настройка завышает ожидаемую производительность системы, поскольку веса специально настраиваются так, чтобы обеспечить максимальную производительность на конкретном множестве запросов, а не на случайной их выборке. В таких случаях правильно было бы сформировать одну или несколько *рабочих тесто-*

вых коллекций документов (development test collection) и подбирать параметры для них. После этого система с настроенными параметрами проходит испытание на тестовой коллекции. Результаты, полученные при таком тестировании, можно рассматривать как несмещенную оценку производительности системы.

8.2. Стандартные тестовые коллекции

Перечислим основные стандартные тестовые коллекции и проекты по их оценке. Мы сосредоточили свое внимание на тестовых коллекциях для оценки поиска по произвольному запросу (ad hoc retrieval), но упомянем также аналогичные тестовые коллекции для классификации текстов.[1]

1. *Коллекция Cranfield* (*Cranfield collection*) была первой тестовой коллекцией, позволявшей дать точную количественную оценку качества информационного поиска. Для современных исследований она слишком мала и может использоваться лишь в самых элементарных пилотных экспериментах. Коллекция формировалась в Великобритании с конца 1950-х годов и содержит 1 398 аннотаций статей из журналов по аэродинамике, 225 запросов и полный набор оценок релевантности каждой пары (запрос, документ).
2. *Конференция по оценке поиска текстов (Text Retrieval Evaluation Conference — TREC).* Американский национальный институт по стандартам и технологиям (U.S. National Institute of Standard and Technology — NIST) начиная с 1992 года проводит серию масштабных испытаний методов информационного поиска. В рамках конференции были проведены многочисленные “дорожки” (tracks) по разным тестовым коллекциям, но наиболее известными являются коллекции, использованные в дорожке TREC Ad Hoc на протяжении первых восьми кампаний с 1992 по 1999 год. В целом эти текстовые коллекции занимают шесть компакт-дисков, содержащих 1,89 миллиона документов (в основном, но не исключительно, новостные сообщения) и оценки релевантности для 450 информационных потребностей, которые называются *темами* (topics) и уточняются с помощью подробных текстовых описаний. На основе разных подмножеств этих данных сформированы отдельные текстовые коллекции. Дорожки TREC 1–5 использовали 50 информационных потребностей, которые оценивались на разных, но перекрывающихся наборах документов. Дорожки TREC 6–8 предусматривали 150 информационных потребностей и коллекцию более чем из 529 тысяч новостных сообщений и статьи службы Foreign Broadcast Information Service. Вероятно, это наилучшее подмножество данных TREC для будущей работы, поскольку оно является самым большим, а темы более тесно согласованы одна с другой. Из-за большого размера тестовых коллекций они не содержат оценок релевантности для всех документов.

[1] Заслуживает упоминания также Российский семинар по оценке методов информационного поиска (РОМИП), проводящийся ежегодно с 2003 года, с числом ежегодных участников 15–20 и такими дорожками, как классическая задача поиска по запросу (ad-hoc track) (по коллекции нормативно-правовых документов и по веб-коллекции), тематическая классификация (нормативно-правовых документов, веб-сайтов и страниц), кластеризация новостного потока, контекстно-зависимое аннотирование текстовых документов, вопросно-ответный поиск, фактографический поиск по новостной коллекции, поиск по изображениям и т.д. (примеры дорожек с РОМИП-2009). — *Примеч. ред.*

Вместо этого существуют только оценки релевантности, сделанные асессорами (assessors) института NIST для документов, входящих в число лучших *k* документов, найденных одной из систем, исследуемой в проекте TREC, для заданной информационной потребности[2].

В последние годы институт NIST проводил оценку на более крупных коллекциях документов, включая коллекцию *GOV2*, содержащую 25 миллионов веб-документов. С самого начала размеры тестовых коллекций документов, сформированных институтом NIST, на порядки превышали все доступные исследователям коллекции, а коллекция GOV2 на данный момент является самой большой веб-коллекцией, доступной для исследовательских целей. Коллекция GOV2 до сих пор более чем на два порядка меньше коллекций документов, проиндексированных крупными коммерческими веб-системами поиска.

3. *Тестовые коллекции Национального института информатики Японии (National Institute of Informatics of Japan — NII) для информационно-поисковых систем (NII Test Collections for IR — NTCIR).* В рамках проекта NTCIR были созданы тестовые коллекции, размеры которых сопоставимы с коллекциями проекта TREC. Эти коллекции посвящены преимущественно восточно-азиатским языкам и *межъязыковому информационному поиску* (cross-language information retrieval), когда запрос формулируется на одном языке, а поиск осуществляется по коллекции документов на других языках (http://research.nii.ac.jp/ntcir/data/data-en.html).
4. *Проект Cross Language Evaluation Forum (CLEF).* Эта серия мероприятий по оценке посвящена европейским языкам и поиску разноязычной информации (http://clef-campaign.org/).
5. *Коллекции Reuters–21578 и Reuters–RCV1.* Для классификации текстов чаще всего используется тестовая коллекция Reuters–21578, содержащая 21 578 новостных сообщений (см. главу 13). Позднее агентство Reuters распространило намного более крупную коллекцию Reuters Corpus Volume 1 (RCV1), состоящую из 806 791 документа (см. главу 4). Благодаря размеру и подробным аннотациям эта коллекция очень хорошо подходит для будущих исследований.
6. *Коллекция 20 Newsgroup.* Это еще одна широко используемая коллекция для классификации текстов, собранная Кеном Лангом (Ken Lang). Для нее из 20 групп новостей Usenet (каждая группа новостей рассматривается как отдельная категория) были извлечены по тысяче статей. После удаления дубликатов эта коллекция состоит из 18 941 статьи.

8.3. Оценка неранжированных результатов поиска

Итак, как оценить качество системы, имея все эти ингредиенты? Чаще всего для оценки информационного поиска используются два показателя: точность (precision) и полнота (recall). Они определены для простого случая, когда информационно-поисковая система возвращает набор документов, соответствующих запросу. Позднее мы покажем, как эти показатели распространить на ранжированные результаты поиска.

Точность (*P*) — это доля релевантных документов среди найденных.

[2] Эта методология называется методологией “pooling” или “общего котла”. — *Примеч. ред.*

$$Точность = \frac{количество\ релевантных\ найденных\ документов}{количество\ найденных\ документов} = P(релевантный | найденный) \quad (8.1)$$

Полнота (*R*) — это доля найденных релевантных документов среди всех релевантных.

$$Полнота = \frac{количество\ релевантных\ найденных\ документов}{количество\ релевантных\ документов} = P(найденный | релевантный) \quad (8.2)$$

Эти понятия можно прояснить, рассмотрев следующую таблицу сопряженных признаков.

	Релевантные	Нерелевантные
Найденные	Истинно положительные (tp)	Ложно положительные (fp)
Не найденные	Ложно отрицательные (fn)	Истинно отрицательные (tn)

(8.3)

Тогда указанные понятия можно определить так.

$$P = \frac{tp}{tp + fp},$$
$$R = \frac{tp}{tp + fn} \quad (8.4)$$

Очевидная альтернатива, которую могут легко найти читатели, — оценивать информационно-поисковые системы по их *правильности* (accuracy), т.е. по доле правильных ответов. В терминах таблицы сопряженности признаков правильность определяется так: *Правильность* = (*tp* + *tn*)/(*tp* + *fp* + *fn* + *tn*). Это кажется разумным, поскольку существуют два действительных класса документов — релевантные и нерелевантные, а информационно-поисковую систему можно интерпретировать как соответствующий бинарный классификатор (возвращающий документы, которые счел релевантными). Именно этот показатель качества часто используется для оценки задач классификации с помощью машинного обучения.

Однако существует важная причина, по которой правильность не подходит для оценки информационного поиска. Практически во всех ситуациях данные крайне несимметричны; как правило, более 99,9% документов являются нерелевантными. Система, настроенная на достижение максимальной правильности, может просто считать все документы нерелевантными всем запросам. Даже если такая система является приемлемой, попытка пометить некоторые документы как релевантные почти всегда приводит к повышению уровня ложно положительных. Однако пометка всех документов нерелевантными совершенно не соответствует потребностям пользователей информационно-поисковых систем. Пользователи всегда хотят видеть какие-то документы и вполне вероятно, что они смирятся с определенной долей ложно положительных, если наряду с ними они получат также некую полезную информацию. Точность и полнота концентрируют оценку систем на возвращаемых ими истинно положительных, как бы спрашивая, какая доля релевантных документов найдена и сколько возвращено ложно положительных.

Преимущество использования двух показателей заключается в том, что во многих ситуациях один из них оказывается важнее другого. Типичные пользователи веба хотели бы, чтобы все результаты, выведенные на первой странице, были релевантными (высокая точность), но не хотят иметь представления обо всех релевантных документах, тем более просматривать их. В противоположность этому профессиональные пользователи, занимающиеся поиском в вебе, например помощники адвокатов и работники спецслужб,

очень заинтересованы в том, чтобы полнота поиска была как можно более высокой, и готовы мириться с низкой точностью. Люди, которые ищут информацию на своих жестких дисках, также часто заинтересованы в высокой полноте поиска. Тем не менее точность и полнота поиска очевидным образом противоречат друг другу: полноту всегда можно повысить до единицы (при очень низкой точности), возвращая все документы на все запросы! Полнота не убывает при увеличении количества найденных документов. С другой стороны, в хороших системах при увеличении количества найденных документов точность обычно снижается. В целом хотелось бы достичь определенной полноты поиска при небольшом уровне ложно положительных.

Показатель, позволяющий найти баланс между точностью и полнотой поиска, называется *F-мерой* (F measure) и представляет собой их среднее гармоническое взвешенное.

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P + R} \tag{8.5}$$

Здесь $\beta^2 = \frac{1-\alpha}{\alpha}$, $\alpha \in [0,1]$, т.е. $\beta^2 \in [0,\infty]$. По умолчанию *сбалансированная F-мера* (balanced F measure) присваивает точности и полноте одинаковые веса, т.е. $\alpha = 1/2$, или $\beta = 1$. Эту меру обычно записывают как F_1, что является сокращением записи $F_{\beta=1}$, хотя запись с использованием символа α более точно отражает свойства меры F как взвешенного среднего гармонического. При $\beta = 1$ формула упрощается.

$$F_{\beta=1} = \frac{2PR}{P+R} \tag{8.6}$$

Однако равные веса не являются единственно возможным вариантом. Значения $\beta < 1$ отдают предпочтение точности, а $\beta > 1$ — полноте. Например, значения $\beta = 3$ или $\beta = 5$ можно использовать, если для пользователя важнее полнота поиска. Точность, полнота и F-мера лежат в отрезке от нуля до единицы, но иногда выражаются в процентах по шкале от нуля до ста.

Почему в качестве F-меры используется среднее гармоническое, а не арифметическое? Напомним, что, возвращая все документы, мы обеспечиваем 100%-ную полноту, следовательно, мы всегда можем добиться, чтобы среднее арифметическое было равным 50%. Это значит, что среднее арифметическое не подходит для оценки. И наоборот, если предположить, что только один документ из десяти тысяч является релевантным запросу, то среднее гармоническое этой стратегии будет равно 0,02%. Среднее гармоническое никогда не превышает ни среднее арифметическое, ни среднее геометрическое. Если эти два средних значительно отличаются друг от друга, то среднее гармоническое ближе к их минимуму, чем к их среднему арифметическому (рис. 8.1).

? ***Упражнение 8.1*** **[*].** Информационно-поисковая система возвращает восемь релевантных и десять нерелевантных документов. Коллекция содержит двадцать релевантных документов. Какова точность и полнота поиска?

Упражнение 8.2 **[*].** Сбалансированная F-мера (F_1) представляет собой среднее гармоническое значение точности и полноты. В чем заключается преимущество среднего гармонического перед арифметическим?

Упражнение 8.3 **[**].** Докажите эквивалентность двух формул для вычисления F-меры в равенстве (8.5) при условии, что $\alpha = 1/(\beta^2 + 1)$.

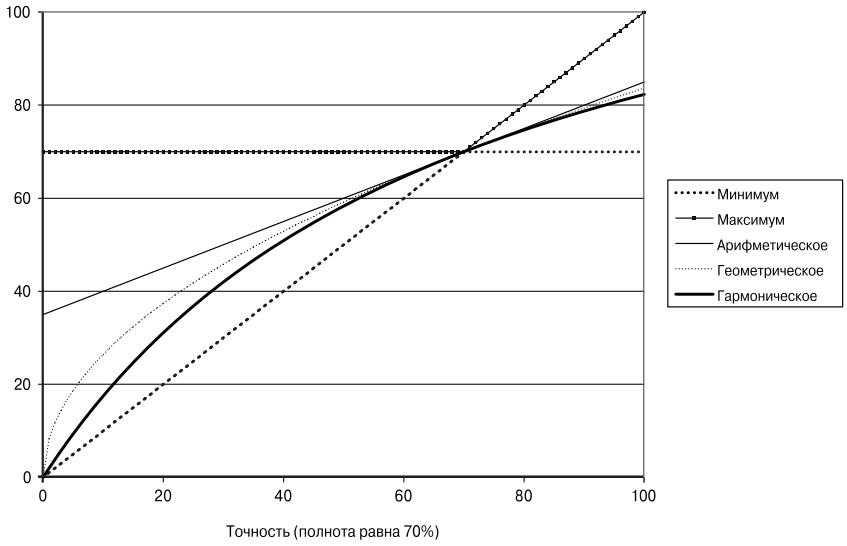


Рис. 8.1. Сравнение среднего гармонического с другими средними значениями. График демонстрирует вычисление средних значений точности и полноты при фиксированной полноте, равной 70%. Среднее гармоническое всегда меньше как среднего арифметического, так и среднего геометрического и часто мало отличается от минимума этих чисел. Когда точность также равна 70%, все показатели совпадают

8.4. Оценка ранжированных результатов поиска

Точность, полнота и *F*-мера — это показатели, вычисляемые по неупорядоченным совокупностям документов. Чтобы оценить стандартные для современных поисковых систем ранжированные результаты, необходимо расширить эти метрики (или определить новые). В этой ситуации в качестве подходящих множеств найденных документов естественно выбирать первые k документов. Для каждого такого множества точность и полноту можно изобразить в виде *кривой "точность–полнота"* (рис. 8.2). Кривые "точность–полнота" имеют пилообразный вид: если (k+1)-й найденный документ оказывается нерелевантным, то полнота остается такой же, как и для первых k документов, но точность снижается. Если же этот документ оказывается релевантным, то как точность, так и полнота увеличиваются, а кривая делает скачок вверх и вправо. Во многих случаях полезно удалить эти зубцы и использовать *интерполированную точность* p_{interp} (interpolated precision), которая при определенном уровне полноты r представляет собой наибольшую точность для всех значений полноты $r' \geq r$.

$$p_{interp}(r) = \max_{r' \geq r} p(r') \tag{8.7}$$

Это объясняется тем, что почти все пользователи согласны просматривать на несколько документов больше, если это увеличит точность поиска. На рис. 8.2 интерполированная точность выделена тонкой линией. Согласно этому определению интерполированная точность при нулевой полноте является полностью определенной (упражнение 8.4).

Изучение кривой "точность–полнота" является весьма информативным, однако довольно часто желательно представить всю эту информацию с помощью нескольких и

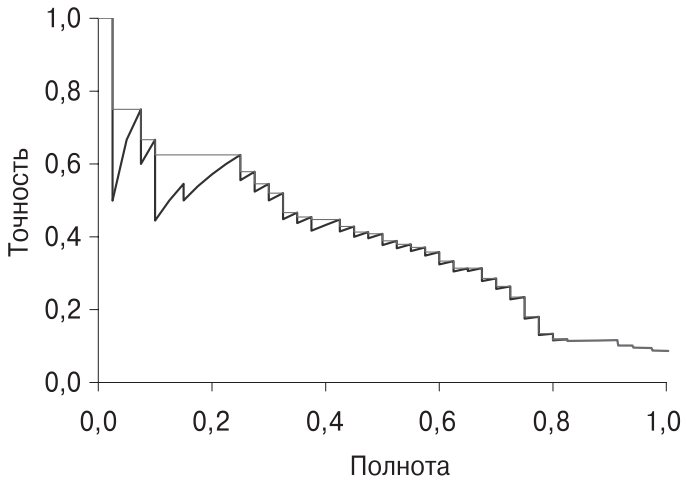


Рис. 8.2. График "точность–полнота"

даже одного значения. Традиционно для этого используется *средняя точность, интерполированная по одиннадцати точкам* (eleven-point interpolated average precision). Именно этот показатель был использован, например, в первых восьми экспериментах TREC Ad Hoc. Для каждой информационной потребности интерполированная точность измеряется на 11 уровнях полноты: 0,0; 0,1; 0,2; ...; 1,0. Для кривой "точность–полнота", представленной на рис. 8.2, эти 11 точек приведены в табл. 8.1. Для каждого уровня полноты вычисляется среднее арифметическое значение интерполированной точности, обеспечивающее заданную полноту поиска по коллекции для каждой информационной потребности. После этого можно построить общий 11-точечный график "точность–полнота". На рис. 8.3 приведен пример такого графика для результатов одной из систем, участвовавших в проекте TREC 8.

Таблица 8.1. Вычисление средней точности, интерполированной по одиннадцати точкам. Числа относятся к рис. 8.2

Полнота	*Интерполированная точность*
0,0	1,00
0,1	0,67
0,2	0,63
0,3	0,55
0,4	0,45
0,5	0,41
0,6	0,36
0,7	0,29
0,8	0,13
0,9	0,10
1,0	0,08

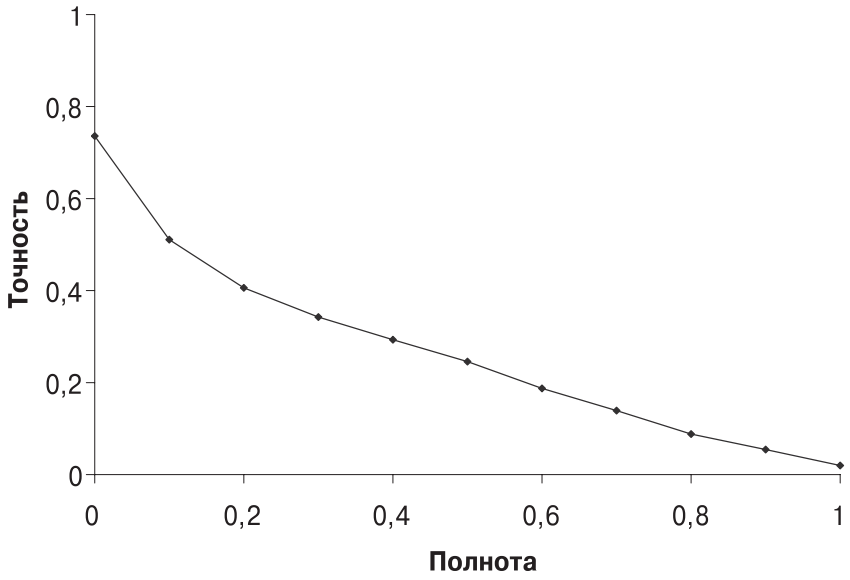


Рис. 8.3. График зависимости средней интерполированной точности от полноты, построенный по пятидесяти запросам к типичной системе TREC. Показатель MAP для этой системы равен 0,2553

В последние годы все большее распространение получают другие меры точности. Чаще всего в среде исследователей, участвующих в проекте TREC, используется *(макро)усредненная средняя точность* (Mean Average Precision — MAP), позволяющая оценить качество системы при разных уровнях полноты одним числом. Показано, что среди других показателей качества MAP обладает особенно хорошими дискриминирующими свойствами и устойчивостью. Рассмотрим множество документов, выданных системой вплоть до позиции очередного релевантного документа, и вычислим для этого множества значение точности. Усреднив значения точности всех таких множеств, мы получим среднюю точность (average precision) одного запроса (информационной потребности). Далее, для вычисления MAP (макроусредненной средней точности), среднюю точность (average precision) усредняют по всем запросам (информационным потребностям). Иначе говоря, если множество релевантных документов для информационной потребности $q_j \in Q$ имеет вид $\{d_1, d_2, ..., d_{m_j}\}$ и R_{jk} — множество упорядоченных результатов поиска из первых по порядку документов вплоть до документа d_k, то

$$MAP(Q) = \frac{1}{|Q|}\sum_{j=1}^{|Q|}\frac{1}{m_j}\sum_{k=1}^{m_j}P(R_{jk}). \qquad (8.8)$$

Если релевантный документ вообще не обнаружен[3], то значение точности в равенстве (8.8) полагается равным нулю. Для отдельной информационной потребности средняя точность приближенно равна площади под неинтерполированной кривой “точность–

[3] Система может не полностью упорядочить все документы в коллекции в ответ на запрос, или оценка эффективности может основываться только на изучении первых k результатов для каждой информационной потребности.

полнота", поэтому показатель MAP приближенно равен средней площади под интерполированной кривой "точность–полнота" для множества запросов.

При использовании показателя MAP фиксированные уровни полноты не выбираются и интерполяция не производится. Значение MAP для тестовой коллекции равно среднему арифметическому средней точности (average precision) для отдельных информационных потребностей. (Такой подход дает эффект приписывания всем информационным потребностям одинаковых весов, даже если одним запросам соответствует много релевантных документов, а другим — мало[4].) Значения MAP, вычисленные для разных информационных потребностей в рамках одной системы, обычно варьируются в широких пределах, например от 0,1 до 0,7. Действительно, значения MAP обычно более согласованы для отдельной информационной потребности в разных системах, чем для разных информационных потребностей в рамках одной и той же системы. Это значит, что множество тестовых информационных потребностей должно быть большим и достаточно разнообразным, чтобы корректно отражать качество работы системы на разных запросах.

Указанные выше показатели учитывают значения точности при всех уровнях полноты. Для многих важных приложений, особенно для веб-поиска, это может оказаться неуместным. Более важным показателем является то, сколько правильных результатов содержится на первой странице или на первых трех страницах. По этой причине точность измеряется при фиксированных уровнях списка найденных результатов, например на уровне десяти или тридцати документов. Этот показатель называется *точностью на уровне k* (precision at *k*), например "точность на уровне десять". Преимущество этого показателя проявляется в том, что не требуется оценка объема множества релевантных документов, а недостаток заключается в том, что он наименее устойчивый из всех распространенных мер точности и плохо усредняется, поскольку на точность на уровне *k* сильно влияет общее количество документов, релевантных запросу.

В качестве альтернативы, позволяющей решить возникшую проблему, используется *R-точность* (*R*-precision). Для ее определения необходимо иметь множество заранее известных релевантных документов *Rel*, по которому вычисляется точность для первых |*Rel*| документов. (Множество *Rel* может быть неполным, например, когда оно формируется путем оценки методом "общего котла" для первых *k* систем, участвующих в эксперименте по оценке.) *R*-точность зависит от количества релевантных документов. *R*-точность поиска для каждого запроса в идеальной системе должна быть равной единице. В то же время, если коллекция содержит только восемь документов, релевантных информационной потребности, то точность на уровне 20 даже у идеальной системы равна только 0,4. Таким образом, усреднение этого показателя по запросам более корректно. Этот показатель труднее объяснить неопытным пользователям, чем точность на уровне *k*, но легче, чем показатель MAP. Допустим, что в коллекции существует |*Rel*| документов, релевантных запросу, тогда мы смотрим на первые |*Rel*| результатов системы и обнаруживаем, что *r* из них являются релевантными. Тогда по определению не только точность (а значит, и *R*-точность), но и полнота этого множества результатов равна *r*/|*Rel*|. Таким образом, *R*-точность оказывается эквивалентной *точке равновесия* (break-even point), другому иногда используемому показателю. Как и точность на уровне *k*, *R*-точность описывает только одну точку на кривой "точность–полнота" и не является суммарной

[4] Из-за равенства вкладов запросов между собой мы, говоря о MAP, выбрали русский перевод ***макроусредненная средняя точность***, вместо неблагозвучного "усредненная средняя точность". — *Примеч. ред.*

оценкой качества по всей кривой. По этой причине иногда непонятно, зачем интересоваться точкой равновесия, а не наилучшей точкой на кривой (точкой максимума *F*-меры) или качеством поиска, актуальным для конкретного приложения (точность на уровне *k*). Тем не менее эмпирические данные свидетельствуют о том, что *R*-точность сильно коррелирует с показателем MAP, несмотря на то что она содержит информацию только об одной точке на кривой.

Иногда для оценки системы поиска используется *кривая соотношений правильного и ложного обнаружения*, или *кривая ROC* (receiver operating characteristics, название не очень понятное). Кривая ROC представляет собой зависимость доли истинно положительных или чувствительности от доли ложно положительных результатов, равной (1 – специфичность). *Чувствительность* (sensitivity) — это просто синоним полноты. Доля ложноположительных результатов вычисляется по формуле *fp*/(*fp* + *tn*). На рис. 8.4 показана кривая ROC, соответствующая кривой "точность–полнота", представленной на рис. 8.2. Кривая ROC всегда следует из левого нижнего угла в правый верхний угол. Для хорошей системы график в левом нижнем углу резко поднимается вверх. Для неупорядоченных множеств результатов *специфичность* (specificity), вычисляемая по формуле *tn*/(*fp* + *tn*), считается не слишком полезным понятием. Поскольку множество истинно отрицательных всегда велико, уровень специфичности для всех информационных потребностей всегда будет близким к единице (и соответственно, доля ложно положительных всегда почти равна нулю). Иначе говоря, "интересной" частью рис. 8.2 является интервал полноты от нуля до 0,4, расположенный на рис. 8.4 слева внизу. Однако кривая ROC может оказаться информативной при анализе полного спектра поиска, позволяя иначе взглянуть на данные. Во многих областях в качестве агрегированного показателя используется площадь фигуры, ограниченной кривой ROC. Этим кривая ROC похожа на показатель MAP. Кривые "точность–полнота" иногда ошибочно называют кривыми ROC. Это объяснимо, но неправильно.

В заключение отметим, что в последнее время все более широкое признание, особенно при использовании методов ранжирования на основе машинного обучения (см. раздел 15.4) получает *совокупная выгода* (cumulative gain) и, в частности, *нормированная дисконтированная совокупная выгода* (Normalized Discounted Cumulative Gain — NDCG). Показатель NDCG разработан для ситуаций, в которых оценки релевантности являются многозначными (ср. раздел 8.5.1). Как и точность на уровне *k*, он вычисляется по *k* найденным документам, имеющим наивысшие позиции. Пусть $R(j, d)$ — оценка релевантности документа *d* относительно запроса *j*. Тогда

$$NDCG(Q,k) = \frac{1}{|Q|}\sum_{j=1}^{|Q|} Z_k \sum_{m=1}^{k} \frac{2^{R(j,m)}-1}{\log(1+m)}. \qquad (8.9)$$

Здесь Z_k — коэффициент нормировки, вычисленный так, чтобы показатель NDCG на уровне *k* при идеальном ранжировании был равен единице. Для запросов, по которым найдено $k' < k$ документов, последнее суммирование проводится до k'.

? ***Упражнение 8.4*** **[*].** Укажите возможные значения интерполированной точности при нулевом уровне полноты.

Упражнение 8.5 **[**].** Всегда ли существует точка равновесия между точностью и полнотой? Либо докажите, что существует, либо приведите контрпример.

Упражнение 8.6 **[**].** Как связаны между собой мера F_1 и точка равновесия?

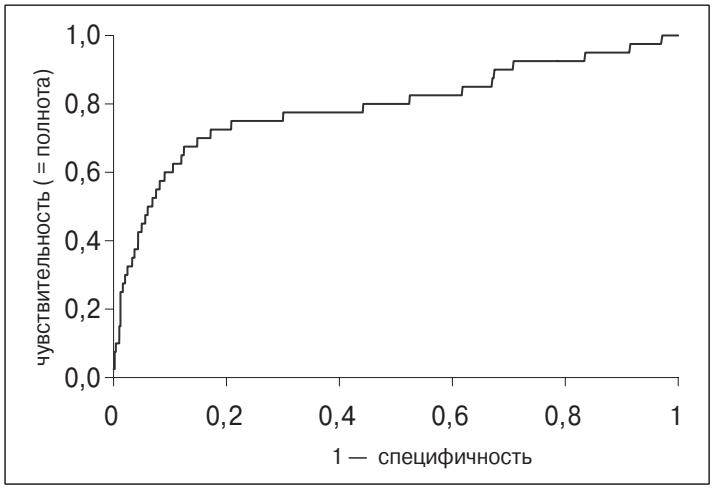


Рис. 8.4. Кривая ROC, соответствующая кривой "точность–полнота", представленной на рис. 8.2

***Упражнение 8.7* [**].** *Коэффициент Дайса* (Dice coefficient) для двух множеств — это мера их пересечения, поделенная на их объем (так, чтобы результат лежал в диапазоне от нуля до единицы).

$$Dice(X,Y) = \frac{2|X \cap Y|}{|X| \cup |Y|}$$

Покажите, что сбалансированная F-мера (F_1) равна коэффициенту Дайса для множеств найденных и релевантных документов.

***Упражнение 8.8* [*].** Допустим, что для некоторой информационной потребности в коллекции содержится четыре релевантных документа. Сравните работу двух систем на примере этой коллекции. Релевантность их первых десяти результатов оценивается следующим образом (крайний слева элемент имеет наивысший порядок).

Система 1	R	N	R	N	N	N	N	N	R	R
Система 2	N	R	N	N	R	R	R	N	N	N

1. Чему равен коэффициент MAP для каждой системы? Какая система имеет более высокий коэффициент MAP?
2. Согласуется ли этот результат с интуицией? Что является важным фактором для получения хороших значений MAP?
3. Чему равна R-точность каждой системы? (Совпадает ли ранжирование систем по R-точности с ранжированием по коэффициенту MAP?)

***Упражнение 8.9* [**].** В следующем упорядоченном списке указаны оценки релевантности (R) и нерелевантности (N) двадцати документов, найденных по запросу в коллекции, состоящей из 10 тысяч документов. Документ с наивысшим порядком (документ, который система считает наиболее релевантным) указан первым. В списке содержится шесть релевантных документов. Допустим, что в коллекции всего восемь релевантных документов.

R R N N N N N N R N R N N N R N N N

1. Чему равна точность системы, оцененная по первым двадцати документам с наивысшими позициями?
2. Чему равна мера F_1, оцененная по первым двадцати документам с наивысшими позициями?
3. Чему равна интерполированная точность системы, полнота которой равна 25%?
4. Чему равна интерполированная точность системы на уровне 33%?
5. Допустим, что эти двадцать документов представляют собой полное множество результатов, возвращенных системой. Чему равен показатель MAP для указанного запроса?

   Теперь допустим, что в ответ на запрос система возвращает все 10 тысяч документов в виде упорядоченного списка, а указанные 20 результатов занимают в этом списке первые места.
6. Чему равен максимально возможный показатель MAP системы?
7. Чему равен минимально возможный показатель MAP системы?
8. В серии экспериментов вручную оцениваются только 20 первых результатов. Для приближенного вычисления диапазона в пп. 6 и 7 используется результат п. 5. Насколько большой (в абсолютных величинах) может быть ошибка при вычислении показателя MAP по данным п. 5, а не 6 или 7?

8.5. Оценка релевантности

Для правильной оценки системы тестовые информационные потребности должны соответствовать документам, хранящимся в тестовой коллекции, и будущему использованию системы. Лучше всего, чтобы эти информационные потребности разрабатывались экспертами в предметной области. Использовать в качестве информационных потребностей случайные сочетания терминов запросов нецелесообразно, поскольку обычно они не отражают их реальное распределение.

Кроме информационных потребностей и документов, необходимо собрать оценки релевантности. Этот процесс требует времени и денег, поскольку связан с участием людей. Для маленьких коллекций наподобие коллекции Cranfield были получены оценки релевантности для каждой пары “запрос–документ” из коллекции. При использовании больших современных коллекций обычно оценивается релевантность лишь части документов для каждого запроса. Чаще всего для этого используется *метод общего котла* (pooling), при котором релевантность оценивается для подмножества коллекции, которое состоит из k первых документов, возвращенных несколькими информационно-поисковыми системами (как правило, системами, подлежащими оценке) и, возможно, полученных из других источников, например из результатов булева поиска по ключевым словам или документов, найденных экспертами в ходе интерактивного процесса.

Человек — это не робот, невозмутимо возвращающий стандартные выводы о релевантности документа по отношению к запросу. Его суждения о релевантности носят субъективный и переменчивый характер. Однако это не проблема: в окончательном итоге успех информационно-поисковой системы зависит от того, как она удовлетворяет информационные потребности именно этих субъективных пользователей.

Тем не менее интересно изучить и оценить согласованность экспертов при оценке релевантности документов. В социальных науках наиболее распространенным показателем согласованности оценок является *каппа-статистика* (kappa statistics). Она разработана для категориальных оценок и делает поправку на случайное совпадение оценок.

$$Каппа = \frac{P(A) - P(E)}{1 - P(E)} \tag{8.10}$$

Здесь $P(A)$ — доля совпавших оценок экспертов, $P(E)$ — ожидаемая доля случайно совпавших оценок. Параметр $P(E)$ можно оценить по-разному. Если выводы касаются лишь двух классов, то ожидаемый уровень случайных совпадений равен 0,5. Однако обычно распределение классов является асимметричным, поэтому для вычисления ожидаемого уровня согласованности обычно используется *маргинальная статистика* (marginal statistics). [5] Есть две возможности оценить величину $P(E)$ в зависимости о того, используется ли объединенное маргинальное распределение по экспертам или маргинальные статистики для каждого эксперта отдельно. На практике применяются оба метода, но мы остановимся на объединенном маргинальном распределении, поскольку при систематических расхождениях между экспертами оно позволяет получить более консервативные оценки. Вычисления продемонстрированы в табл. 8.2. Если два эксперта всегда соглашаются друг с другом, то статистика каппа равна единице; если их мнения совпадают случайно, то каппа-статистика равна нулю; и она отрицательна, если расхождения больше, чем может объяснить случайность. Если экспертов больше двух, то можно вычислить среднее попарных значений каппа-статистики. На практике считается, что значение каппа, превышающее 0,8, означает хорошее согласование, между 0,67 и 0,8 — удовлетворительное, а меньше 0,67 — сомнительное основание для оценки. Однако точные пороги зависят от предназначения данных.

Таблица 8.2. Вычисление каппа-статистики

		Выводы эксперта 2 о релевантности		
		Да	Нет	Всего
Выводы эксперта 1 о релевантности	Да	300	20	320
	Нет	10	70	80
	Всего	310	90	400

Наблюдаемая доля согласованных выводов

$P(A) = (300 + 70)/400 = 370/400 = 0{,}925$

Объединенные маргинальные статистики

$P(\text{нерелевантный}) = (80 + 90)/(400 + 400) = 170/800 = 0{,}2125$

$P(\text{релевантный}) = (320 + 310)/(400 + 400) = 630/800 = 0{,}7878$

Вероятность того, что мнения экспертов совпали случайно

$P(E) = P(\text{нерелевантный})^2 + P(\text{релевантный})^2 = 0{,}2125^2 + 0{,}7878^2 = 0{,}665$

Каппа-статистика

$\kappa = (P(A) - P(E))/(1 - P(E)) = (0{,}925 - 0{,}665)/(1 - 0{,}665) = 0{,}776$

[5] Для табл. 8.2 маргинальная статистика вычисляется путем суммирования по строке или столбцу: $a_{i.k} = \sum_j a_{ijk}$.

Согласованность выводов относительно релевантности документов измерялись в проектах TREC и для медицинских информационно-поисковых коллекций. Как правило, каппа-статистика лежала в удовлетворительном диапазоне (от 0,67 до 0,8). Тот факт, что бинарные выводы экспертов о релевантности документов довольно слабо согласованы друг с другом, является одной из причин, по которым при оценке не используется более подробная шкала. Для ответа на вопрос, являются ли результаты оценки информационно-поисковых систем корректными несмотря на колебания оценок отдельных экспертов, проводились эксперименты, в которых за эталон принимались оценки одного эксперта. В этом случае *абсолютные* оценки качества могут существенно измениться, но *относительное* ранжирование разных систем или вариантов одной и той же системы в целом сохраняется.

8.5.1. Преимущества и недостатки концепции релевантности

Преимущество оценки системы на основе стандартной модели релевантных и нерелевантных документов заключается в том, что мы имеем фиксированные условия, так что мы можем проводить сравнительные эксперименты с участием разных систем или модификаций одной системы. Такое формальное тестирование намного дешевле и позволяет более точно выявить влияние параметров системы на ее качество, чем на основе экспериментов с участием пользователей. Действительно, опираясь на формальный показатель, которому мы доверяем, мы можем оптимизировать качество системы с помощью методов машинного обучения, а не настраивать параметры вручную. Разумеется, если формальный показатель плохо описывает потребности пользователя, то такой подход никак не поможет повысить удовлетворенность пользователя. По нашему мнению, стандартные формальные показатели качества информационно-поисковых систем, несмотря на упрощения, достаточно хороши, и недавние работы, посвященные оптимизации формальных оценок, блестяще это подтвердили. Существует множество методик, разработанных на основе формальных оценок, которые повысили качество реальных систем, например методы нормировки длины документов в рамках проекта TREC (разделы 6.4.4 и 11.4.3) и методы машинного обучения для настройки весов параметров при ранжировании (раздел 6.1.2).

Это не значит, что с этими абстракциями нет никаких проблем. Например, считается, что релевантность одного документа в коллекции не зависит от релевантности другого. (На этом предположении на самом деле построено большинство информационно-поисковых систем и методов оценки их качества — документы оцениваются относительно запросов, а не относительно друг друга.) Выводы носят бинарный характер: никаких нюансов при оценке релевантности нет. Оценка релевантности документа по отношению к информационной потребности считается абсолютной и объективной. В то же время оценки релевантности документов являются субъективными, причем разные люди высказывают разные суждения. На практике эксперты также делают ошибки и проявляют невнимательность. Кроме того, при оценке информационно-поисковых систем часто полагают, что информационные потребности не изменяются во время просмотра результатов поиска. Результаты, основанные на одной коллекции, сильно зависят от самой коллекции, а также от запросов и набора оценок релевантности. При этом иногда невозможно перенести результаты оценки на другую предметную область или группу пользователей.

Некоторые из этих проблем можно решить. В недавних экспериментах по оценке качества систем, включая проекты INEX, некоторые дорожки TREC и NTCIR, применялась шкала оценок релевантности, при этом документы разделялись на три или четыре класса:

от слабо релевантных до сильно релевантных. Детальное обсуждение организации оценки качества в рамках проекта INEX изложено в разделе 10.4.

С методами оценки поиска на основе релевантности документов, описанными выше, связана одна очевидная проблема: разница между релевантностью и *маргинальной релевантностью* (marginal relevance), т.е. сохраняет ли документ хоть какую-то полезность, после того как пользователь просмотрел другие документы (Carbonell and Goldstein, 1998). Даже если документ имеет высокую релевантность, его информация может оказаться излишней и вторичной, если она уже содержится в ранее найденных документах. Наиболее выраженный вариант это явления — документы-дубликаты, часто встречающиеся в вебе, или случай, когда несколько документов описывают одно событие. В этих условиях маргинальная релевантность лучше оценивает полезность документа для пользователя. Максимизация маргинальной релевантности требует, чтобы найденные документы были разнообразными и новыми. Для того чтобы оценить ее, в качестве единиц оценки можно использовать отдельные факты или сущности. Вероятно, маргинальная релевантность является более точной оценкой полезности системы с точки зрения пользователя, но для такой оценки сложнее создать тестовую коллекцию.

? ***Упражнение 8.10* [**].** Ниже приведена таблица, в которой содержатся оценки релевантности двух экспертов относительно двенадцати документов по отношению к определенной информационной потребности (0 — релевантный, 1 — нерелевантный). Предположим, что вы разработали информационно-поисковую систему, которая в ответ на этот запрос возвращает множество документов {4, 5, 6, 7, 8}.

docID	Эксперт 1	Эксперт 2
1	0	0
2	0	0
3	1	1
4	1	1
5	1	0
6	1	0
7	1	0
8	1	0
9	0	1
10	0	1
11	0	1
12	0	1

1. Вычислите каппа-статистику для этих двух экспертов.
2. Вычислите точность, полноту и F_1 вашей системы, если документ считается релевантным при условии, что оценки обоих экспертов совпали.
3. Вычислите точность, полноту и F_1 вашей системы, если документ считается релевантным, при условии, что хотя бы один эксперт посчитал его релевантным.

8.6. Более широкая точка зрения: качество системы и ее полезность для пользователя

Формальные показатели довольно далеки от нашей конечной цели — измерить полезность системы для человека, т.е. оценить, насколько каждый пользователь удовлетворен ответами системы на его информационные потребности. Обычно удовлетворенность людей оценивается в ходе разнообразных исследований с участием пользователей. Они могут включать в себя количественные показатели, как объективные, например время выполнения задания, так и субъективные, такие как баллы, выставляемые пользователями поисковой системе, а также качественные оценки, такие как комментарии пользователей по поводу поискового интерфейса. В этом разделе мы коснемся других аспектов системы, позволяющих давать количественные оценки полезности системы, а также проблем полезности для пользователя.

8.6.1. Характеристики системы

Существует много стандартных тестов, помимо качества поиска, по которым можно оценивать информационно-поисковые системы.

- Скорость индексирования, иначе говоря, количество документов, индексируемых в час для определенного распределения длин документов (см. главу 4)
- Скорость поиска, т.е. насколько велика задержка, зависящая от размера индекса
- Насколько выразителен язык запросов и насколько быстро обрабатываются сложные запросы
- Размер коллекции документов, т.е. сколько документов накоплено в коллекции по разным темам

Все эти критерии, помимо выразительности языка запросов, допускают непосредственное *измерение*, ведь скорость и размер можно измерить. Выразительность языка можно приблизительно оценить с помощью различных перечней свойств.

8.6.2. Полезность для пользователя

На самом деле нам нужна общая количественная оценка степени удовлетворенности пользователя, основанная на релевантности, скорости и удобстве интерфейса. Для этого, в частности, необходимо понимать потребности людей, для которых создается система. Например, пользователи поисковых веб-систем удовлетворены, если они нашли то, что искали. Косвенно степень этой удовлетворенности можно оценить по количеству людей, которые регулярно пользуются такими системами. Этот показатель является эффективным, но мы получили бы более точную оценку, если бы могли знать, как часто эти пользователи прибегают к помощи других поисковых систем. Еще одну группу пользователей поисковых веб-систем образуют рекламодатели. Они удовлетворены, если клиенты заходят на их веб-сайты и делают там покупки. На сайтах электронной коммерции пользователи обычно что-нибудь покупают. Таким образом, можно оценить время, прошедшее до совершения покупки, или долю посетителей, ставших покупателями. На веб-сайте, играющем роль витрины магазина, потребности пользователя и магазина считаются удовлетворенными, если посетитель совершил покупку. И все же необходимо решить, что мы оптимизируем: удовлетворенность конечного пользователя или владельца сайта электронной коммерции. Ведь за оптимизацию обычно платит владелец сайта.

Для корпоративной поисковой системы, работающей во внутренней сети компании, правительственного или академического учреждения, более подходящим показателем может быть производительность труда пользователя: сколько времени пользователь затрачивает в поисках нужной информации. Кроме того, существуют другие практические критерии, касающиеся, например, информационной безопасности (см. раздел 4.6).

Степень удовлетворения пользователя довольно трудно оценить, отчасти поэтому стандартные методологии используют показатели, основанные на оценках релевантности результатов поиска. Проще всего оценить удовлетворенность напрямую в рамках специальных экспериментов с участием пользователей. В ходе этих экспериментов производятся наблюдения за работой пользователей, решающих типичные задачи, измеряются разные показатели и применяются методы этнографических интервью, которые позволяют получать качественную информацию об удовлетворенности. Эксперименты с участием пользователей эффективны при проектировании системы, но связаны с большими затратами времени и денег. Кроме того, для правильной организации таких экспериментов необходимы опыт и знания, позволяющие корректно интерпретировать их результаты. Детальное описание этих методов не входит в наши планы.

8.6.3. Настройка работающей системы

Если система информационного поиска уже создана и эксплуатируется многочисленными пользователями, то ее разработчики могут оценить возможные изменения, развернув разные варианты системы и фиксируя показатели, свидетельствующие об удовлетворенности пользователей тем или иным вариантом. Этот метод часто используется в системах веб-поиска.

Чаще всего используется метод *A/B теста* (A/B testing). Этот термин заимствован из рекламной индустрии. В ходе такого теста в функционирующей системе изменяется только один аспект и на ее модифицированный вариант случайным образом направляется небольшая доля трафика (скажем, 1–10% пользователей), в то время как большинство пользователей по-прежнему применяют текущую версию. Например, если мы хотим исследовать модификацию алгоритма ранжирования, то должны перенаправить случайную выборку пользователей на новый вариант системы и оценить определенные показатели, такие как частота кликов по первым позициям выдачи или по любому результату на первой странице. (Этот конкретный вариант метода называется *анализом кликов* (clickthrough log analysis or clickthrough mining)). В дальнейшем мы будем называть его методом неявной обратной связи (см. раздел 9.1.7).

В основе A/B-тестирования лежит проведение серии тестов (последовательно или параллельно), в каждом из которых исследуется влияние только одной переменной. При выполнении каждого теста изменяется только один параметр по сравнению с контрольной версией (текущей версией системы). Это позволяет легко выяснить, положительно или отрицательно влияет изменение этого параметра на работу системы. Такое тестирование работающей системы позволяет просто и дешево оценить влияние изменений на пользователей и, при достаточно большом количестве пользователей, учесть даже очень маленькие эффекты. В принципе, изменяя одновременно несколько параметров случайным образом, можно было бы использовать более мощные аналитические методы многомерного статистического анализа, например множественную линейную регрессию. Однако на практике A/B-тестирование используется широко, поскольку его легко организовать, понять и объяснить руководству.

8.7. Сниппеты

После того как мы нашли по запросу и ранжировали документы, хотелось бы представить пользователю информативный список результатов. Во многих случаях пользователи не желают проверять все найденные документы, поэтому список результатов должен быть достаточно информативным, чтобы пользователь мог произвести окончательное собственное ранжирование документов в соответствии с их релевантностью информационной потребностью.[6] Как правило, для этого используются *сниппеты* (snippets) — короткие аннотации документов, по которым пользователи могли бы оценить релевантность документов. Как правило, спиппеты состоят из заголовка документа и короткой аннотации, извлекаемой автоматически. Проблема состоит в том, чтобы представить аннотацию в максимально полезном для пользователя виде.

Аннотации подразделяются на *статические* (static), одинаковые для всех запросов, и *динамические* (dynamic), или *запросозависимые* (query dependent), которые соответствуют информационной потребности (запросу) пользователя. Динамические аннотации предназначены для того, чтобы объяснить, почему именно этот документ был найден в ответ на запрос.

Статическая аннотация обычно состоит из фиксированного фрагмента документа и/или метаданных, связанных с ним. Простейшая форма аннотации содержит первые два предложения или пятьдесят слов документа или определенные зоны документа, например заголовок и фамилию автора. Вместо зон документа аннотация может использовать метаданные, ассоциированные с документом. Эти метаданные могут также содержать фамилию автора или дату и, например, включать описание (`description`), которое содержится в поле `meta` HTML-страницы. Такая аннотация обычно извлекается из документа и сохраняется на этапе индексирования, так, чтобы ее можно было быстро извлечь и включить в результаты поиска. Доступ к оригинальному документу часто связан с довольно большими затратами.

Большая работа была проведена в области обработки естественных языков (Natural Language Processing — NLP) для повышения качества *реферирования текста* (text summarization). В большинстве случаев задача сводится к методам выбора хороших предложений исходного текста, из которых формируется реферат. Эти модели часто сочетают факторы, связанные с позицией предложения (предпочтение отдается первому и последнему абзацам документов, а также первому и последнему предложениям в абзаце), с тематическими факторами, которые повышают вес предложений с ключевыми терминами (т.е. с терминами с низкой частотой в коллекции, но высокой частотой и хорошим распределением в найденном документе). При более сложном подходе система синтезирует предложения реферата с нуля или на основе редактирования комбинации оригинальных предложений документа. Например, система может удалить придаточное предложение или заменить местоимение именной группой, на которую оно ссылается. Этот последний класс методов до сих пор остается лишь предметом академических исследований и редко используется в практике информационного поиска. Легче, безопаснее и часто даже лучше просто извлечь предложения из исходного документа.

---

[6] Некоторые предметные области, в которых критична полнота, представляют собой исключения. Например, во многих случаях поиска правовой информации юристы просматривают *каждый* документ, соответствующий запросу из ключевых слов.

Динамические аннотации показывают один или несколько фрагментов документа, которые помогут пользователю оценить соответствие документа информационной потребности. Обычно эти фрагменты содержат один или несколько терминов запроса и поэтому часто называются *сниппетами с ключевыми словами в контексте* (Keyword-In-Context — KWIC), хотя иногда могут включать фрагменты текста, например заголовок, которые не зависят от запроса, как в статическом реферировании (рис. 8.5). Формирование динамических аннотаций связано с ранжированием. Если в документе найдена фраза, совпадающая с запросом, вхождения этой фразы в документ будут показаны в качестве сниппета. В противном случае будет выбран фрагмент документа, содержащий несколько терминов запроса. Как правило, эти фрагменты просто охватывают определенное количество слов слева и справа от терминов запроса. В этом месте можно применить методы обработки естественного языка. Пользователи предпочитают читабельные сниппеты, содержащие законченные фразы.

. . . ***In recent years, Papua New Guinea has faced severe economic difficulties and*** economic growth has slowed, partly as a result of weak governance and civil war, and partly as a result of external factors such as the Bougainville civil war which led to the closure in 1989 of the Panguna mine (at that time the most important foreign exchange earner and contributor to Government finances), the Asian financial crisis, a decline in the prices of gold and copper, and a fall in the production of oil. ***PNG's economic development record over the past few years is evidence that*** governance issues underly many of the country's problems. Good governance, which may be defined as the transparent and accountable management of human, natural, economic and financial resources for the purposes of equitable and sustainable development, flows from proper public sector management, efficient fiscal and accounting mechanisms, accounting mechanisms, mechanisms, and a willingness to make service delivery a priority in practice. . . .

Рис. 8.5. Пример выбора текста для создания динамического сниппета. Этот сниппет сгенерирован по документу в ответ на запрос new guinea economic development. Полужирным шрифтом выделен текст сниппета в исходном тексте

Считается, что динамические аннотации сильно повышают удобство информационно-поисковых систем, но усложняют их разработку. Динамическую аннотацию нельзя создать заранее, но, с другой стороны, если система содержит только позиционный индекс, то нельзя легко реконструировать контекст слов запроса в документе, чтобы создать динамическую аннотацию. Это одна из причин, по которым используются статические аннотации. Стандартное решение этой проблемы с учетом низкой цены и большого объема дисковой памяти — во время индексирования сохранять локальную копию всех документов (несмотря на то что при таком подходе возникает множество юридических проблем, а также вопросов, связанных с информационной безопасностью, которые до сих пор далеки от решения), как показано на рис. 7.5. В этом случае система может просто сканировать документ, который должен появиться в списке результатов, выполняя поиск фрагментов, содержащих слова из запроса. Помимо простого доступа к тексту, при создании KWIC-сниппетов требуется определенная осторожность. Если в документе встречаются многие ключевые слова, то наша цель — выбрать фрагмент, который был бы 1) максимально информативным по отношению к терминам запроса, 2) достаточно само-

стоятельным, чтобы быть читабельным, и 3) достаточно коротким, чтобы поместиться в ограниченный объем, отведенный для аннотаций.

Генерация сниппетов должна быть быстрой, поскольку система обычно генерирует много сниппетов для каждого обрабатываемого запроса. Вместо всего документа обычно сохраняется обширный, но все же имеющий ограниченный размер префикс документа, например первые 10 тысяч символов. Чаще всего при этом короткие документы сохраняются полностью, а огромные документы не загромождают память на локальных дисках. Аннотации документов, длина которых превышает размер префикса, основываются только на содержании префикса, но, как правило, этого достаточно для того, чтобы понять его смысл.

Если документ был изменен с момента его последней загрузки и индексирования, то эти изменения не скажутся ни на сохраненной копии, ни на индексе. В этих ситуациях ни индекс, ни аннотация не отражают текущего содержания документа, но именно разница между аннотацией и фактическим содержанием документа бросается в глаза пользователю.

8.8. Библиография и рекомендации для дальнейшего чтения

Первое определение и использование понятия релевантности запросу относятся к 1953 году. Свансон (Swanson, 1988) сообщает, что при проведении оценки в этом году два коллектива согласились, что 1 390 документов являются в разной степени релевантными по отношению к 98 вопросам, но разошлись во мнениях относительно остальных 1 577 документов, причем эти разногласия так никогда и не были устранены.

Впервые строгое формальное тестирование информационно-поисковых систем было проведено в ходе экспериментов в университете Кранфилда (Cranfield), начавшихся в конце 1950-х годов. Ретроспективный обзор тестовой коллекции Кранфилда и описание экспериментов можно найти в работе Клевердона (Cleverdon, 1991). Другая серия пионерских экспериментов в области информационного поиска была проведена Джерардом Салтоном (Gerard Salton) и его коллегами (Salton, 1971, *б*, 1991) на системе SMART. Эксперименты TREC подробно описаны Ворхес и Харман (Voorhees and Harman, 2005). Эта информация доступна на сайте http://trec.nist.gov/. Сначала немногие исследователи оценивали статистическую значимость своих экспериментальных результатов, но сообщество специалистов по информационному поиску все настоятельнее требовало расширения этих исследований (Hull, 1993). Исследования качества информационно-поисковых систем с участием пользователей начались позднее (Saracevic and Kantor, 1988, 1996).

Понятия точности и полноты впервые были использованы Кентом и др. (Kent et al., 1955), хотя термин *точность* (precision) был введен немного позднее. *F*-мера (или, точнее, ее дополнение $E = 1 - F$) была предложена Рийсбергеном (Rijsbergen, 1979). Он провел широкое теоретическое исследование, в рамках которого с помощью принципа уменьшающейся маргинальной релевантности (в какой-то момент пользователь не склонен жертвовать точностью для достижения полноты) доказал, что гармоническое среднее является приемлемым способом сочетания точности и полноты (в противоположность минимуму или геометрическому среднему).

Бакли и Ворхес (Buckley and Voorhees, 2000) сравнили несколько показателей качества поиска, включая точности на уровне *k*, MAP и R-точность, а также оценили погрешность каждого показателя. R-точность была принята в качестве официальной оценки в

дорожке TREC HARD (Allan, 2005). Аслам и Йилмаз (Aslam and Yilmaz, 2005) исследовали ее удивительно высокую корреляцию с показателем MAP, отмеченную в более ранних исследованиях (Tague-Sutcliffe and Blustein, 1995; Buckley and Voorhees, 2000). Стандартной программой для оценки информационно-поисковых систем, вычисляющей многие показатели качества ранжированного поиска, является программа `trec_eval`, написанная Крисом Бакли (Chris Buckley) и использованная в проекте TREC. Ее можно загрузить с сайта http://trec.nist.gov/trec_eval/.

Кекалайнен и Ярвелин (Kekäläinen and Järvelin, 2002) приводят доводы о преимуществе градуированных оценок релевантности при работе с очень большими коллекциями. В их работе (Järvelin and Kekäläinen, 2002) предложены методы оценки информационно-поисковых систем, основанные на совокупной выгоде. Сакаи (Sakai, 2007) провел исследование устойчивости и чувствительности метрик, основанных на градуированных оценках релевантности в рамках проекта NTCIR, и пришел к выводу, что наилучшим показателем для ранжирования документов является показатель NDCG.

Шамбер и др. (Schamber et al., 1990) изучили концепцию релевантности с учетом ее многомерной и контекстно-зависимой природы и при этом пришли к выводу, что ее можно эффективно оценить. Статья Ворхес[IS5] (Voorhees, 2000) стала классической работой на тему неустойчивости оценок релевантности, а также ее влияния на оценку и ранжирование систем в рамках проекта TREC Ad Hoc. Ворхес пришла к выводу, что, несмотря на изменения абсолютных значений, ранжирование остается довольно устойчивым. Херш и др. (Hersh et al., 1994) провели аналогичный анализ на примере коллекций медицинских документов. В противоположность этим работам Кекалайнен (Kekäläinen, 2005) проанализировал результаты более поздних экспериментов TREC, используя четырехуровневые оценки релевантности и понятие совокупной выгоды и сделал вывод, что используемые показатели существенно влияют на ранжирование систем (см. также работу Хартера (Harter, 1998)). Цобель (Zobel, 1998) исследовал вопрос, является ли метод общего котла, используемый в проекте TREC для создания подмножества документов, подлежащих оценке, надежным и корректным, и пришел к положительному выводу.

Каппа-статистика и ее применение в языковых приложениях обсуждалась в работе Карлетта (Carletta, 1996). Классические работы (Siegel and Castellan, 1988) содержат объединенные вычисления ожидаемого согласия, но Ди Эугенио и Гласс (Di Eugenio and Glass, 2004) отдают предпочтение раздельному согласованию (хотя и со многими показателями). Более подробную информацию об альтернативных оценках согласия, которые могут быть даже лучше, можно найти в работах Ламбарда и др. (Lombard et al., 2002) и Криппендорфа (Krippendorff, 2003).

Реферирование текста активно исследуется многие годы. Современные работы по выбору предложений были инициированы Купиецем и др. (Kupiec et al, 1995). Среди более поздних работ отметим статьи Бразилая и Эльхадада (Barzilay and Elhadad, 1997) и Джинга (Jing, 2000). Кроме того, широкий спектр работ на эту тему публикуется в трудах ежегодной конференции (Document Understanding Conference — DUC) и на других научных конференциях, посвященных обработке естественного языка. Преимущество динамических аннотаций в контексте информационно-поисковых систем продемонстрировано в работе Томброза и Сандерсона (Tombros and Sanderson, 1998). Эффективные методы генерации сниппетов обсуждается в работе Терпина и др. (Turpin et al., 2007).

Анализ кликов (clickthrough log analysis) изучен в работах Йоахимса и др. (Joachims, 2002*b*; Joachims et al., 2005).

В серии статей Херша, Терпина и их коллег показано, что улучшение формального качества поиска, полученное в групповых экспериментах, не всегда приводит к улучшению системы с точки зрения пользователя (Hersh et al., 2000*a*, 2000*b*, 2001; Turpin and Hersh, 2001, 2002).

Пользовательские интерфейсы и значение человеческого фактора для информационно-поисковых систем, а также модели поиска информации и тестирование удобства использования (usability testing) выходят за рамки нашей книги. Подробную информацию на эту тему читатели найдут в других учебниках (Baeza-Yates and Ribeiro-Neto, 1999; Korfhage, 1997) и коллекциях, посвященных когнитивным аспектам (Spink and Cole, 2005).

Глава 9

Обратная связь по релевантности и расширение запроса

Во многих коллекциях одно и то же понятие может выражаться разными словами. Это явление, известное как *синонимия* (synonymy), влияет на полноту поиска в большинстве информационно-поисковых систем. Например, пользователи хотели бы, чтобы запросу `aircraft` соответствовало также слово `plane` (но только в смысле самолет, а не *столярный рубанок*), а запросу `thermodynamics` (термодинамика) — слово `heat` (тепло) в соответствующем контексте. Пользователи часто стараются самостоятельно разрешить эту проблему, уточняя запросы (см. раздел 1.4). В этой главе мы рассмотрим способы, с помощью которых система может сама уточнить запрос либо автоматически, либо с участием пользователя.

Методы решения этой задачи разделяются на две основные категории: глобальные и локальные. Глобальные методы предусматривают расширение или новую формулировку запроса независимо от запроса и возвращаемых результатов, так что изменения в формулировке запроса приводят к появлению нового запроса, соответствующего другим семантически близким терминам. К глобальным относятся следующие методы.

- Расширение/новая формулировка запроса с помощью специального тезауруса или тезауруса WordNet (раздел 9.2.2)
- Расширение запроса с помощью автоматической генерации тезауруса (раздел 9.2.3)
- Методы, похожие на приемы исправления опечаток (см. главу 3)

Локальные методы изменяют запрос с учетом документов, найденных по исходному запросу. К локальным относятся следующие методы.

- Обратная связь по релевантности (раздел 9.1.1)
- Обратная связь по псевдорелевантности, известная также как *слепая обратная связь по релевантности* (раздел 9.1.6)
- (Глобальная) неявная обратная связь по релевантности (раздел 9.1.7)

В этой главе мы коснемся всех упомянутых подходов, но сосредоточим свое внимание лишь на методе обратной связи по релевантности, который оказался наиболее распространенным и успешным.

9.1. Обратная связь по релевантности и псевдорелевантности

Идея *обратной связи по релевантности* (Relevance Feedback — RF) заключается в привлечении пользователя к процессу поиска, чтобы улучшить итоговый список результатов. В частности, пользователь сообщает системе о релевантности документов в первоначальном списке результатов. Вкратце эта процедура выглядит следующим образом.

- Пользователь делает (короткий, простой) запрос.
- Система возвращает первоначальный список найденных результатов.
- Пользователь отмечает некоторые из найденных документов как релевантные или нерелевантные.
- Система определяет улучшенное представление информационной потребности, основываясь на обратной связи с пользователем.
- Система выводит на экран уточненный набор найденных результатов.

Метод RF может предусматривать одну или несколько итераций. В основе этого процесса лежит идея, согласно которой пользователь не в состоянии сформулировать точный запрос, не зная хорошо содержания коллекции, но может оценить документы. Поэтому целесообразно выполнить несколько таких итераций, чтобы уточнить запрос. В рамках этого сценария метод RF может способствовать эволюции информационной потребностей пользователя. Просмотр некоторых документов может помочь пользователю уточнить свои представления об информации, которую он ищет.

Ярким примером метода RF является поиск изображений. Результаты поиска изображений легко просмотреть, но именно в этой области пользователю трудно сформулировать свой запрос словами, но легко указать, какие изображения являются релевантными или нерелевантными. После того как пользователь введет исходный запрос `bike` на странице

`http://nayana.ece.ucsb.edu/imsearch/imsearch.html`,

он получит первоначальный список результатов (в данном случае — изображений). На рис. 9.1, *а* пользователь выбрал изображения, которые считает релевантными. Они используются для уточнения запроса, в то время как остальные изображения на новую формулировку запроса не влияют. На рис. 9.1, *б* показаны новые результаты, ранжированные после выполнения итерации по методу RF.

На рис. 9.2 приведен пример использования метода RF для текстового поиска; в данном случае пользователь хочет узнать о новых применениях космических спутников.

9.1.1. Алгоритм Роккио для обратной связи по релевантности

Алгоритм Роккио (Rocchio algorithm) — классический алгоритм для реализации метода RF. Он инкорпорирует модель обратной связи по релевантности в модель векторного пространства, описанную в разделе 6.3.

Теория. Мы хотим найти вектор запроса $\vec{q}$, максимально близкий к релевантным документам и минимально похожий на нерелевантные документы. Если C_r — это множество релевантных документов, а C_{nr} — нерелевантных, то целью наших поисков является следующий вектор.[1]

$$\vec{q}_{opt} = \arg\max_{\vec{q}} \left[sim(\vec{q}, C_r) - sim(\vec{q}, C_{nr}) \right] \tag{9.1}$$

[1] В этом равенстве выражение $\arg\max_x f(x)$ означает значение x, при котором функция $f(x)$ достигает максимума. Аналогично выражение $\arg\min_x f(x)$ означает значение x, при котором функция $f(x)$ достигает минимума.

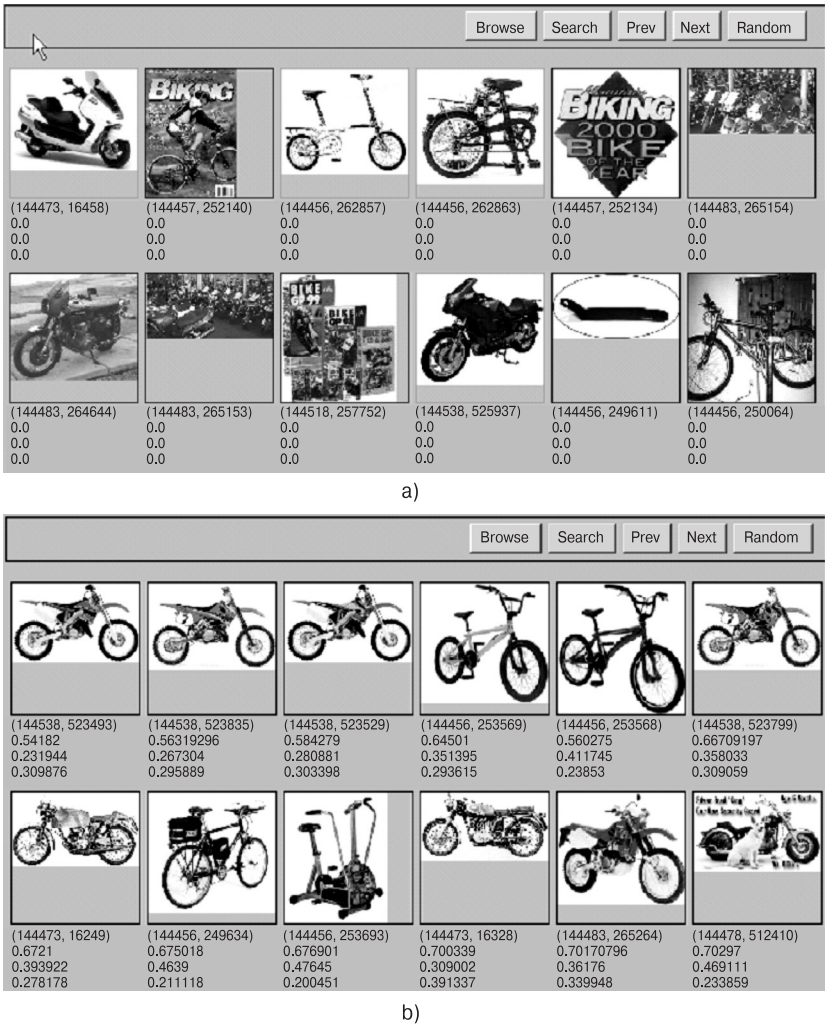


Рис. 9.1. Метод поиска изображений с помощью метода RF: а) пользователь просматривает первоначальный список результатов поиска по запросу `bike`*, выбирает в качестве релевантных первое, третье и четвертое изображения в первом ряду и четвертое — в нижнем ряду, а затем возвращает их в качестве обратной связи; б) пользователь получает уточненный набор результатов. Точность существенно повысилась. Снимок с сайта http://nayana.ece.ucsb.edu/imsearch/imsearch.html (Newsam et al., 2001)*

Здесь величина *sim* определена равенством (6.10). Если в качестве меры близости используется косинусная мера сходства, то оптимальный вектор запроса $\vec{q}_{opt}$ для разделения релевантных и нерелевантных документов определяется следующим равенством.

$$\vec{q}_{opt} = \frac{1}{|C_r|}\sum_{\vec{d}_j \in C_r} \vec{d}_j - \frac{1}{|C_{nr}|}\sum_{\vec{d}_j \in C_{nr}} \vec{d}_j \qquad (9.2)$$

а) Query: New space satellite applications

б) + 1. 0.539, 08/13/91, NASA Hasn't Scrapped Imaging Spectrometer
2. 0.533, 07/09/91, NASA Scratches Environment Gear From Satellite Plan
3. 0.528, 04/04/90, Science Panel Backs NASA Satellite Plan, But Urges Launches of Smaller Probes
4. 0.526, 09/09/91, A NASA Satellite Project Accomplishes Incredible Feat: Staying Within Budget
5. 0.525, 07/24/90, Scientist Who Exposed Global Warming Proposes Satellites for Climate Research
6. 0.524, 08/22/90, Report Provides Support for the Critics Of Using Big Satellites to Study Climate
7. 0.516, 04/13/87, Arianespace Receives Satellite Launch Pact From Telesat Canada
+ + 8. 0.509, 12/02/87, Telecommunications Tale of Two Companies
2.074 new 15.106 space
30.816 satellite 5.660 application
5.991 nasa 5.196 eos
4.196 launch 3.972 aster
3.516 instrument 3.446 arianespace
3.004 bundespost 2.806 ss
2.790 rocket 2.053 scientist
2.003 broadcast 1.172 earth
0.836 oil 0.646 measure

г) 1. 0.513, 07/09/91, NASA Scratches Environment Gear From Satellite Plan
* 2. 0.500, 08/13/91, NASA Hasn't Scrapped Imaging Spectrometer
* 3. 0.493, 08/07/89, When the Pentagon Launches a Secret Satellite, Space Sleuths Do Some Spy Work of Their Own
4. 0.493, 07/31/89, NASA Uses 'Warm' Superconductors For Fast Circuit
5. 0.492, 12/02/87, Telecommunications Tale of Two Companies
* 6. 0.491, 07/09/91, Soviets May Adapt Parts of SS-20 Missile For Commercial Use
7. 0.490, 07/12/88, Gaping Gap: Pentagon Lags in Race To Match the Soviets In Rocket Launchers
8. 0.490, 06/14/90, Rescue of Satellite By Space Agency To Cost $90 Million

*Рис. 9.2. Пример использования метода RF на текстовой коллекции: а) исходный запрос; б) пользователь отмечает релевантные документы (помечены плюсом); в) запрос расширяется до 18 терминов с указанными весами; г) пользователю возвращается уточненный список результатов. Символом * отмечены документы, которые на этапе обратной связи по релевантности пользователь отметил как релевантные*

Иначе говоря, оптимальный запрос — это вектор разницы между центроидами релевантных и нерелевантных документов (рис. 9.3). Однако это наблюдение не очень полезно, поскольку полное множество релевантных документов неизвестно — именно его мы ищем.

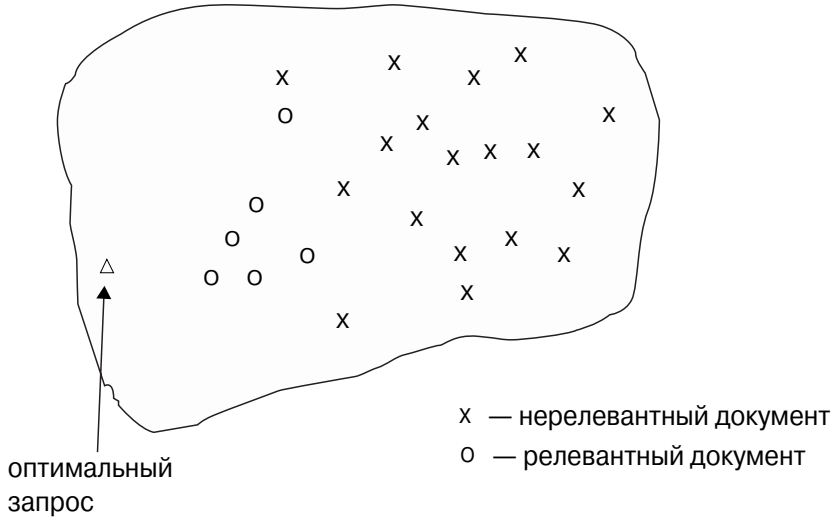


Рис. 9.3. Оптимальный по Роккио запрос для разделения релевантных и нерелевантных документов

Алгоритм Роккио (1971). Описанный механизм обратной связи по релевантности был реализован в системе Дж. Солтона SMART около 1970 года и стал известен благодаря этой системе. Пусть у нас есть запрос пользователя и — частично — знание о релевантности документов. Алгоритм предлагает использовать модифицированный запрос $\vec{q}_m$.

$$\vec{q}_m = \alpha\vec{q}_0 + \beta\frac{1}{|D_r|}\sum_{\vec{d}_j \in D_r}\vec{d}_j - \gamma\frac{1}{|D_{nr}|}\sum_{\vec{d}_j \in D_{nr}}\vec{d}_j \qquad (9.3)$$

Здесь q_0 — оригинальный вектор запроса, D_r и D_{nr} — множества известных релевантных и нерелевантных документов соответственно, α, β и γ — веса каждого слагаемого. Эти величины управляют вкладом множества оцененных документов и первоначального запроса. Если у нас много оцененных документов, то мы можем увеличить веса β и γ. Начиная с запроса q_0, новый запрос перемещается на определенное расстояние в направлении центроида релевантных документов и на некоторое расстояние от центроида нерелевантных документов. Этот новый запрос можно использовать для поиска документов в стандартной модели векторного пространства (см. раздел 6.3). Мы можем уйти из положительного квадранта векторного пространства, вычитая вектор нерелевантного документа. В алгоритме Роккио отрицательные веса терминов игнорируются. Иначе говоря, вес такого термина полагается равным нулю. Эффект применения обратной связи по релевантности показан на рис. 9.4.

Обратная связь по релевантности может улучшить как полноту, так и точность. Однако на практике было показано, что она является наиболее полезной для повышения полноты в тех ситуациях, когда полнота критична. Частично это объясняется тем, что метод Роккио расширяет запрос (т.е. добавляются новые термины), а частично — контекстом использования. Когда пользователи хотят достичь высокой полноты, они готовы потратить время на просмотр результатов и повторно выполнить поиск. Положительная обратная связь также оказывается более ценной, чем отрицательная, поэтому в большинстве информационно-поисковых систем принято устанавливать параметры так, чтобы $\gamma < \beta$. Разумно выбрать следующие параметры: $\alpha = 1$, $\beta = 0{,}75$ и $\gamma = 0{,}15$. Многие системы, на-

пример система поиска изображений, продемонстрированная на рис. 9.1, допускают только положительную обратную связь, что эквивалентно установке $\gamma = 0$. В качестве альтернативы можно использовать как отрицательную обратную связь только нерелевантные документы с высоким рангом (т.е. в равенстве (9.3) $|D_{nr}| = 1$). Несмотря на то что многочисленные экспериментальные результаты сравнения разных вариантов обратной связи по релевантности не позволяют сделать окончательные выводы, некоторые исследователи считают, что именно этот вариант, получивший название *Ide dec-hi*, является наиболее эффективным или по крайней мере наиболее непротиворечивым.

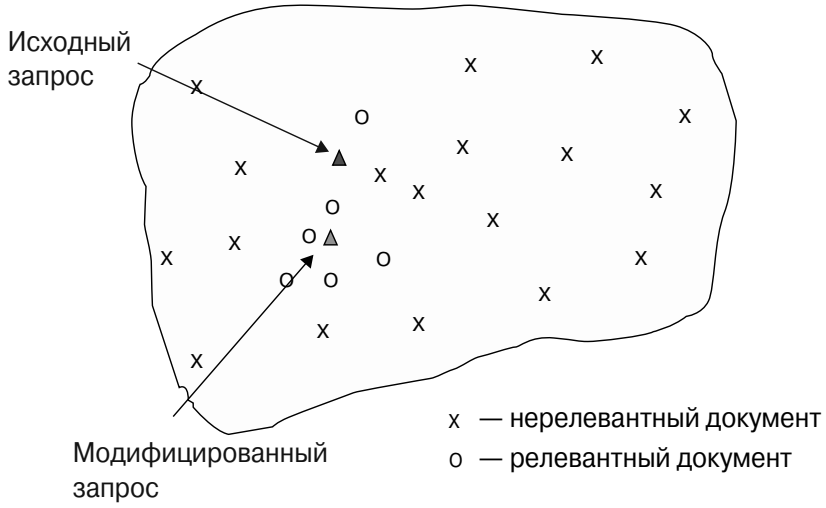


Рис. 9.4. Применение алгоритма Роккио. Некоторые документы помечены как релевантные или нерелевантные, а вектор исходного запроса перемещается в ответ на сигнал обратной связи

? ***Упражнение 9.1.*** При каких условиях модифицированный запрос q_m в равенстве (9.3) совпадает с исходным запросом q_0? Можно ли утверждать, что во всех других вариантах запрос q_m ближе к центроиду релевантных документов, чем запрос q_0?

Упражнение 9.2. Почему положительная обратная связь в информационно-поисковых системах считается более полезной, чем отрицательная? Почему использование только одного нерелевантного документа может быть более эффективным, чем использование нескольких?

Упражнение 9.3. Допустим, что исходный запрос пользователя имеет вид cheap CDs cheap DVDs extremely cheap CDs. Пользователь сравнивает два документа: d_1 и d_2. Он считает документ d_1, содержащий фразу *CDs cheap software cheap CDs*, релевантным, а документ d_2, содержащий фразу *cheap thrills DVDs*, нерелевантным. Допустим, что мы используем обычную частоту термина (без масштабирования и без документной частоты). Нормировать [pb1]вектор не обязательно. Как изменится вектор запроса после поступления сигнала обратной связи Роккио, как указано в равенстве (9.3)? Будем считать, что $\alpha = 1$, $\beta = 0{,}75$ и $\gamma = 0{,}25$.

***Упражнение 9.4* [*].** Омар (Omar) реализовал систему веб-поиска, основанную на методе RF, в которой обратная связь по релевантности использует только слова в

заголовке страницы (для эффективности). Пользователь собирается ранжировать три результата. Первый пользователь, Джинксинг, послал запрос

banana slug

и получил в ответ три заглавия:

banana slug Ariolimax columbianus

Santa Cruz mountains banana slug

Santa Cruz Campus Mascot

Джинксинг считает два первых документа релевантными, а третий — нерелевантным. Допустим, что поисковая система Омара использует частоту термина и не применяет нормализацию длины векторов и обратную документную частоту. Предположим также, что параметры алгоритма Роккио выбраны так: $\alpha = \beta = \gamma = 1$. Покажите, как выглядит окончательный запрос. (Элементы вектора перечислите в алфавитном порядке.)

9.1.2. Вероятностная обратная связь по релевантности

Кроме изменения веса термина запроса в векторном пространстве, существуют другие способы организации обратной связи по релевантности. Например, если пользователь указал несколько релевантных и нерелевантных документов, то можно построить классификатор. В частности, основой классификатора может стать наивная байесовская вероятностная модель. Пусть R — булева индикаторная переменная, отражающая релевантность документа. Тогда мы можем оценить величину $P(x_t = 1)$, т.е. вероятность того, что термин t встретится в документе, в зависимости от того, релевантный это документ или нет.

$$\begin{aligned} \hat{P}\left(x_t = 1 \middle| R = 1\right) &= |VR_t| / |VR|, \\ \hat{P}\left(x_t = 0 \middle| R = 0\right) &= \left(df_t - |VR_t|\right) / \left(N - |VR|\right). \end{aligned} \tag{9.4}$$

Здесь N — общее количество документов, df_t — количество документов, содержащих термин t, VR — множество известных релевантных документов и VR_t — подмножество документов из VR, содержащих термин t. Хотя множество известных релевантных документов, скорее всего, является небольшим подмножеством множества всех релевантных документов, если предположить, что само множество релевантных документов представляет собой небольшое подмножество всех документов, то приведенные выше оценки вполне разумны. На основе этих оценок можно предложить новый способ изменения весов терминов запроса. Более подробно вероятностные подходы будут исследованы в главах 11 и 13. В частности, применение этого подхода к обратной связи по релевантности будет изложено в разделе 11.3.4. Пока заметим, что равенств (9.4) еще недостаточно для изменения весов терминов. Эти равенства используют статистику по коллекции и информацию о распределении термина в документах, считающихся релевантными, но не информацию о конкретном запросе.

9.1.3. Когда обратная связь по релевантности эффективна?

Успех метода RF зависит от определенных предположений. Во-первых, пользователь должен иметь достаточно знаний, чтобы сформулировать исходный запрос, хоть в какой-то мере близкий к искомым документам. Это условие является необходимым в любой

информационно-поисковой системе, но следует отметить несколько проблем, которые метод RF сам по себе устранить не может.

- *Неправильное правописание*. Если пользователь пишет термин запроса в том виде, в котором он не встречается ни в одном документе коллекции, то метод RF вряд ли окажется эффективным. Эту проблему можно разрешить с помощью исправления ошибок, описанного в разделе 3.
- *Многоязычные информационно-поисковые системы*. Документы на разных языках в векторном пространстве находятся далеко друг от друга, а документы на одном и том же языке группируются ближе друг к другу.
- *Несогласованность словаря пользователя и словаря коллекции*. Если пользователь ищет термин `laptop`, а все документы содержат слово `notebook`, то запрос не принесет результата, и обратная связь по релевантности, скорее всего, станет неэффективной.

Во-вторых, метод обратной связи по релевантности требует, чтобы релевантные документы были похожи друг на друга, т.е. образовывали кластеры. В идеальном случае распределение термина по всем релевантным документам должно быть похожим на распределение термина во всех документах, отмеченных пользователем, а распределение термина по всем нерелевантным документам должно отличаться от распределения термина в релевантных документах. Метод работает хорошо, если все релевантные документы образуют кластер вокруг отдельного прототипа или существуют разные прототипы, словари релевантных документов сильно перекрываются и схожесть между релевантными и нерелевантными документами мала. Модель Роккио неявно интерпретирует релевантные документы как отдельный *кластер*, который моделируется с помощью центроида. Этот подход не работает, если релевантные документы образуют мультимодальный класс, т.е. состоят из нескольких кластеров в векторном пространстве. Это может произойти в следующих ситуациях.

- Подмножества документов используют разные словари, например `Burma`[2] или `Myanmar`
- Запрос, множество ответов на который разнородно (дизъюнктивно) по своей природе, например `Pop stars who once worked at Burger King`
- Общие понятия, которые часто распадаются на дизъюнкцию нескольких более конкретных понятий, например `felines`

Качественно подготовленное содержание документов коллекции часто может помочь в решении этой проблемы. Например, статья об отношении различных групп к ситуации в Бирме может содержать терминологию, которую используют разные стороны: в результате возникают связи между кластерами документов.

Обратная связь по релевантности не всегда нравится пользователям. Они часто отказываются от явной обратной связи или вообще не желают продолжать. Более того, часто по результатам поиска на основе обратной связи трудно понять, почему был найден конкретный документ.

Кроме того, обратная связь по релевантности порождает несколько практических проблем. Длинные запросы, генерируемые в результате применения метода RF, в обыч-

[2] “Бирма” — прежнее название государства Мьянма (Myanmar). — *Примеч. ред.*

ных информационно-поисковых системах оказываются неэффективными. Это приводит к большим вычислительным затратам и увеличивает время отклика на запрос пользователя. Частичное решение этой проблемы можно получить, изменив веса самых важных терминов в релевантном документе, например первых двадцати наиболее часто встречающихся терминов. Согласно некоторым экспериментальным результатам использование ограниченного количества терминов может дать более хорошие результаты (Harman, 1992), хотя в другой публикации авторы утверждают, что использование большего количества терминов повышает качество найденных документов (Buckey et al., 1994*b*).

9.1.4. Обратная связь по релевантности в вебе

Некоторые системы веб-поиска предоставляют функциональность поиска похожих/связанных документов: пользователь указывает в списке результатов документ, который, по его мнению, лучше всего соответствует его информационной потребности, и запрашивает другие документы, похожие на него. Такую функциональность можно рассматривать как упрощенный вариант обратной связи по релевантности. Однако в целом метод RF редко используется в вебе. Исключением была система веб-поиска Excite, которая первоначально предоставляла полноценную обратную связь по релевантности. Однако от этой опции со временем отказались, так как она не была востребована пользователями[3]. В вебе лишь немногие люди используют расширенные возможности поиска, а большинство предпочитает ограничиться единственным запросом. Такое неприятие может объясняться двумя причинами: во-первых, суть метода RF трудно объяснить типичному пользователю и, во-вторых, основной целью этого метода является повышение полноты поиска, которая, как правило, редко интересует пользователей систем веб-поиска.

В 2000 году Спинк и др. (Spink et al., 2000) опубликовали результаты использования метода RF на поисковой системе Excite. Этот метод применялся только примерно в 4% поисковых сессий, причем большинство из них ограничивались использованием опции “More like this” (“Похожие документы”), которая сопровождала каждый результат. Около 70% пользователей просмотрели первую страницу результатов и не искали документы на следующих. При использовании метода RF результаты были улучшены примерно в двух третях случаев.

Более важным направлением в последнее время стал анализ кликов пользователя по результатам поиска, что обеспечивает неявную обратную связь по релевантности. Использование таких данных подробно изучено в работах Йоахимса (Joachims, 2002*b*; Joachims et al., 2005). Очень успешным стало использование структуры гиперссылок в вебе (см. главу 21), которую также можно рассматривать как вид неявной обратной связи, хотя эта связь устанавливается с авторами страницы, а не с ее читателями (хотя на практике большинство авторов также являются читателями).

9.1.5. Оценка стратегий обратной связи по релевантности

Интерактивная обратная связь по релевантности позволяет получить существенный выигрыш в качестве поиска. С эмпирической точки зрения часто очень полезным оказывается даже один цикл обратной связи по релевантности. Два цикла иногда оказываются

---

[3] Функция, модифицирующая первоначальный запрос с помощью включения в него выбранного пользователем релевантного документа, присутствует в форме *[текст_старого_запроса related:URL_документа]* во многих поисковых системах, например в Google и Bing. — *Примеч. ред*.

лишь ненамного полезнее. Для успешного использования RF требуется достаточно большое количество оцененных документов, иначе процесс становится неустойчивым и может уйти в сторону от информационной потребности пользователя. Соответственно, рекомендуется иметь не менее пяти оцененных документов.

Трудно оценить эффективность метода RF в полной мере и наглядно. Очевидно, что в качестве первой стратегии можно начать с исходного запроса q_0 и построить график "точность–полнота". После одного цикла обратной связи от пользователя мы вычисляем модифицированный запрос q_m и снова строим график "точность–полнота". В ходе обоих циклов оценивается качество работы системы по всем документам в коллекции, которая допускает простое сравнение. В этом случае можно получить впечатляющий выигрыш: увеличение показателя MAP примерно на 50%. Но, к сожалению, это обман. Этот выигрыш частично объясняется тем фактом, что известные релевантные документы (оцененные пользователем) теперь получают более высокий ранг. Для корректной оценки качества сравнение следует проводить только по документам, неизвестным пользователю.

Вторая идея заключается в том, чтобы использовать в ходе второго цикла документы из *остаточной коллекции* (residual collection), т.е. множество документов за исключением оцененных как релевантные. Такая оценка кажется более реалистичной. К сожалению, измеренная эффективность часто оказывается ниже, чем для исходного запроса. Этот эффект особенно ярко проявляется, когда существует небольшое количество релевантных документов, и значительная их доля была оценена пользователем в ходе первого цикла. Относительную эффективность разных методов обратной связи по релевантности можно корректно оценить, но очень трудно обоснованно сравнить качество систем с обратной связью по релевантности и без нее, поскольку размер коллекции и количество релевантных документов до и после цикла обратной связи различаются.

Таким образом, ни один из этих методов не является вполне удовлетворительным. Третий метод предполагает работу с двумя коллекциями, одна из которых используется для исходного запроса и оценок релевантности, а вторая — для сравнительной оценки. Качество обработки запросов q_0 и q_m можно корректно сравнить с помощью второй коллекции.

Возможно, наилучшим способом оценки полезности метода RF являются эксперименты с участием пользователей, в частности, путем сравнения временны*х* показателей: насколько быстро пользователь находит релевантные документы с помощью RF по сравнению с другой стратегией (например, с новой формулировкой запроса) или как много релевантных документов пользователь находит за определенный отрезок времени. Такие оценки полезности являются наиболее корректными и близкими к реальному использованию системы.

9.1.6. Обратная связь по псевдорелевантности

Обратная связь по псевдорелевантности (pseudo relevance feedback), или *слепая обратная связь по релевантности* (blind relevance feedback), — это метод автоматического локального анализа. Он позволяет автоматизировать ту часть RF, которая выполняется вручную, так что пользователь повышает качество поиска, не вступая в дополнительное взаимодействие с системой. В рамках этого метода сначала выполняется поиск и находится исходная совокупность наиболее релевантных документов, в которой первые k документов, имеющие наибольшие ранги, *предполагаются* релевантными, а затем к ним применяется метод RF с учетом этого предположения.

В большинстве случаев этот автоматический метод работает хорошо[4]. Опыт показывает, что он работает лучше, чем глобальный анализ (раздел 9.2). Было показано, что метод повышает качество выполнения заданий дорожки TREC ad hoc (рис. 9.5). Однако автоматический процесс не лишен недостатков. Например, если запрос имеет вид `copper mines` (медные рудники) и в нескольких документах, занимающих первые места, речь идет о рудниках в Чили, то возможен дрейф запросов в направлении документов о Чили.

	Точность на уровне k = 50	
Взвешивание термина	Без RF, %	ПсевдоRF, %
lnc.ltc	64,2	72,7
Lnu.ltu	74,2	87,0

Рис. 9.5. Результаты, свидетельствующие о том, что метод обратной связи по псевдорелевантности существенно улучшает качество поиска. Эти результаты получены на системе Cornell SMART в рамках эксперимента TREC 4 (Buckley et al., 1995). В этом исследовании сравнивались две разные схемы нормализации длины (l и L, см. рис. 6.15). Метод обратной связи по псевдорелевантности сводился к добавлению двадцати терминов к каждому запросу

9.1.7. Неявная обратная связь по релевантности

В качестве базы для обратной связи можно использовать косвенные свидетельства вместо явных оценок релевантности. Этот метод часто называется *неявной обратной связью по релевантности* (implicit relevance feedback). Неявная обратная связь менее надежна, чем явная, но более полезна, чем обратная связь по псевдорелевантности, не учитывающая мнение пользователей. К тому же пользователи часто отказываются участвовать в явной обратной связи, а собрать данные большого объема о неявной обратной связи в случае большой системы, например системы веб-поиска, несложно.

В контексте веба был предложен метод DirectHit, идея которого состоит в том, чтобы присваивать более высокий ранг тем документам, которые пользователи выбирают для просмотра чаще[5]. Иначе говоря, предполагается, что клики по ссылкам на страницы являются показателями релевантности этих страниц запросу. В этом подходе делается несколько предположений, например что сниппеты документов в результатах поиска (с помощью которых пользователи выбирают, на какие страницы перейти) являются индикаторами релевантности этих документов. В оригинальной системе поиска DirectHit данные о кликах собирались глобально и не были связаны с конкретными пользователями или запросами. Этот метод является одной из разновидностей общего подхода *анализа кликов* (clickstream mining). В настоящее время очень похожий метод используется для ранжирования рекламных объявлений, соответствующих поисковым запросам в вебе (глава 19).

---

[4] Важно не забывать об упомянутой выше дополнительной нагрузке на систему, которую создает обратная связь по псевдорелевантности: предполагается не просто собрать большое количество новых терминов, но и затем автоматически задать еще один запрос системе. — *Примеч. ред.*

[5] Или тех документов, на просмотре которых пользователи задерживаются дольше, или тех, которые чаще являются последним выбором пользователя в поисковой сессии, и т.д. — *Примеч. ред.*

9.1.8. Резюме

Как показали исследования, метод RF позволяет очень эффективно повысить релевантность результатов. Для его успешного использования необходимы запросы, для которых существует достаточно много релевантных документов. Полная обратная связь по релевантности является обременительной для пользователя, а ее реализация в большинстве информационно-поисковых систем не очень эффективна. Во многих случаях достичь аналогичного улучшения можно с помощью других методов интерактивного поиска, затратив меньше усилий.

Помимо основного сценария поиска по произвольному запросу, обратная связь по релевантности используется в следующих ситуациях.

- Отслеживание изменяющихся информационных потребностей (например, марки автомобилей, которыми интересуется пользователь, со временем изменяются).
- Поддержка информационных фильтров (например, для получения новостей). Такие фильтры рассматриваются в главе 13.
- Активное обучение (нахождение примеров, которые полезно классифицировать вручную, чтобы сократить затраты на формирование обучающей выборки).

? ***Упражнение 9.5.*** Какие значения весов α, β и γ в алгоритме Роккио соответствуют команде “найти похожую страницу”?

Упражнение 9.6 **[*].** Назовите три причины, по которым метод обратной связи по релевантности редко используется в поисковых веб-системах.

9.2. Глобальные методы для переформулирования запроса

В данном разделе мы кратко обсудим три глобальных метода расширения запроса: путем подсказок и помощи пользователю с использованием тезауруса, создаваемого вручную, и с использованием тезауруса, создаваемого автоматически.

9.2.1. Словарные инструменты для переформулирования запроса

Для того чтобы пользователь мог определить, насколько был успешен его поисковый запрос, существуют разные методы поддержки. К ним относятся информация о стоп-словах, исключенных из запроса, результат применения стемминга к словам запроса, количество совпадений для каждого термина или фразы, а также информация о том, какие последовательности слов запроса обрабатывались как фразы. Информационно-поисковая система может также предложить поисковые термины с помощью тезауруса или контролируемого словаря. Пользователю можно предоставить возможность просмотра словаря инвертированного индекса и таким образом помочь найти хорошие термины, встречающиеся в коллекции.

9.2.2. Расширение запроса

В методе обратной связи по релевантности пользователи вводят дополнительную информацию о документах (помечая релевантные и нерелевантные документы в списке ре-

зультатов), и эта информация используется для изменения весов терминов в запросе. С другой стороны, при *расширении запроса* (query expansion) пользователи вводят дополнительную информацию о словах запроса или фразах, возможно, предлагая дополнительные термины запроса. Некоторые поисковые системы (особенно в вебе) предлагают в ответ на запрос ряд связанных запросов; пользователи могут выбрать один из этих альтернативных запросов. На рис. 9.6 приведен пример вариантов запроса, предлагаемых поисковой веб-системой Yahoo!. Основной вопрос, связанный с этой формой расширения запроса, — как сгенерировать альтернативные или расширенные запросы для пользователя. Чаще всего для расширения запроса применяется метод глобального анализа, использующий определенные форму тезауруса. Для каждого термина t запрос можно автоматически расширить с помощью синонимов или близких слов из тезауруса. Использование тезауруса можно сочетать с идеями взвешивания термина; например, можно приписать добавленным терминам веса, которые меньше весов исходных терминов запроса.

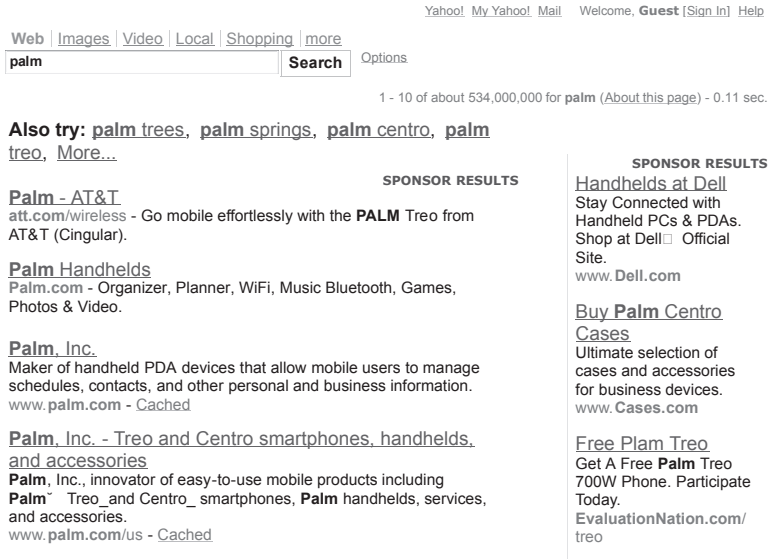


Рис. 9.6. Пример расширения запроса в интерфейсе поисковой веб-системы Yahoo! в 2008 году. Варианты расширенных запросов появляются непосредственно после панели "Search Results" (результаты поиска)

Существуют различные методы построения тезауруса, предназначенного для расширения запросов.

- *Использование контролируемого словаря, поддерживаемого редакторами*. Для каждого понятия в этом словаре есть канонический термин. Примерами контролируемых словарей являются тематические рубрики в каталогах традиционных библиотек, такие как тематические рубрики Библиотеки конгресса США (Library of Congress Subject Headings) или десятичная система классификации Дьюи. Использование контролируемых словарей характерно для предметных областей с большим количеством источников информации. Ярким примером является Unified Medical Language System (UMLS), которая используется службой Medline для поиска статей по биологии и медицине. Например, на рис. 9.7

к поиску слова `cancer` (рак) добавляется слово `neoplasms` (новообразования). Это расширение запроса к системе Medline контрастирует с примером расширения запроса к поисковой системе Yahoo! Интерфейс Yahoo! представляет собой пример интерактивного расширения запроса, а система PubMed расширяет запрос автоматически. Если пользователь не захочет проверить посланный запрос, он может даже не узнать, что произошло расширение запроса.

- User query: `cancer`
- PubMed query: ("neoplasms"[TIAB] NOT Medline[SB]) OR "neoplasms"[MeSH Terms] OR cancer[Text Word]
- User query: `skin itch`
- PubMed query: ("skin"[MeSH Terms] OR "integumentary system"[TIAB] NOT Medline[SB]) OR "integumentary system"[MeSH Terms] OR skin[Text Word] AND (("pruritus"[TIAB] NOT Medline[SB]) OR "pruritus"[MeSH] OR itch[TextWord])

Рис. 9.7. Примеры расширения запроса с помощью тезауруса PubMed. Запрос пользователя, заданный через интерфейс PubMed системе Medline по адресу `www.ncbi.nim.nih.gov/entrez/`, отображается в словарь Medline, как показано на рисунке

- *Тезаурус, создаваемый вручную*. Здесь редакторы создают множества синонимов для понятий без назначения канонического термина. Одним из таких тезаурусов является метатезаурус UMLS. Система Statistics Canada поддерживает тезаурус предпочтительных терминов, синонимов, а также более широких и узких терминов по отраслям, по которым правительство собирает статистические данные, например товары и услуги. К тому же этот тезаурус является двуязычным (на английском и французском языках).
- Автоматически *создаваемый тезаурус*. Для автоматического создания тезауруса используются статистические данные о совместной встречаемости слов в документах предметной области (см. раздел 9.2.3).[6]
- *Переформулирование запроса на основе анализа лога запросов*. Здесь используется предыдущее переформулирование запросов, сделанных вручную пользователями, чтобы предложить их в качестве подсказки новому пользователю. Для этого требуется огромное количество запросов, поэтому метод подходит для веб-поиска.

Расширение запроса с помощью тезауруса имеет одно преимущество: оно не требует дополнительного ввода информации от пользователя. Использование расширения запроса обычно увеличивает полноту поиска. Этот метод широко применяется во многих научных и технических областях. Кроме глобального анализа, для расширения запроса можно применять методы локального анализа, например, анализируя документы в списке результатов. В этом случае пользователь должен ввести дополнительную информацию, причем сохраняется различие: обратная связь может относиться к терминам запроса или к документам.

[6] В последнее время чаще используется совместная встречаемость не столько в одном документе, сколько в множестве ссылок на один документ или в множестве запросов пользователя внутри одной поисковой сессии. — *Примеч. ред.*

9.2.3. Автоматическая генерация тезауруса

В качестве альтернативы затратному ручному созданию тезауруса можно попытаться сгенерировать тезаурус автоматически путем анализа коллекции документов. Существует два основных метода. Один из них просто использует совместную встречаемость слов. Считается, что слова, встречающиеся в одном документе или абзаце, являются близкими по смыслу или связанными между собой, поэтому для поиска таких слов используется простой подсчет текстовых статистических показателей. Другой подход основан на использовании поверхностного грамматического анализа текста, а также грамматических отношений или зависимостей. Например, если нечто выращивается, готовится, съедается и переваривается, то, скорее всего, речь идет о еде. Простое использование смежных слов более надежно (на него не влияют ошибки грамматического анализа), но поиск с помощью грамматических отношений является более точным.

Для того чтобы составить тезаурус на основе совместной встречаемости, проще всего опираться на попарное сходство терминов. Начнем с матрицы "термин–документ" A, в которой каждая ячейка $A_{t,d}$ содержит взвешенное количество вхождений $w_{t,d}$ термина t в документ d, в котором вес выбирается так, чтобы строки матрицы A были нормализованы по длине. Затем вычисляется матрица $C = AA^T$, где $C_{u,v}$ — мера сходства между терминами u и v (чем больше эта мера, тем лучше). На рис. 9.8 приведен пример тезауруса, созданного с помощью описанного выше метода за исключением того, что для снижения размерности в нем предусмотрен еще один этап — латентное семантическое индексирование (latent semantic indexing), которое будет рассмотрено в главе 18. Одни термины тезауруса хороши или по крайней мере могут служить подсказкой, другие — несущественны и плохи. Качество ассоциаций между словами обычно представляет собой проблему. Неоднозначность терминов легко создает нерелевантные статистически коррелированные термины. Например, запрос `Apple computer` может быть расширен до `Apple red fruit computer`[7]. Недостатки таких тезаурусов — как ложноположительные, так

word	nearest neighbors
absolutely	absurd, whatsoever, totally, exactly, nothing
bottomed	dip, copper, drops, topped, slide, trimmed
captivating	shimmer, stunningly, superbly, plucky, witty
doghouse	dog, porch, crawling, beside, downstairs
makeup	repellent, lotion, glossy, sunscreen, skin, gel
mediating	reconciliation, negotiate, case, conciliation
keeping	hoping, bring, wiping, could, some, would
lithographs	drawings, Picasso, Dali, sculptures, Gauguin
pathogens	toxins, bacteria, organisms, bacterial, parasite
senses	grasp, psyche, truly, clumsy, naive, innate

Рис. 9.8. Пример автоматически сгенерированного тезауруса. Этот пример основан на работе Шютце (Schütze, 1998), использующей латентное семантическое индексирование (см. главу 18)

[7] Из этого примера следует, что расширение слов запроса через тезаурус является контекстнозависимым, т.е. расширение слова, пригодное для одного запроса, непригодно для другого: запрос [hot dogs] не равен [boiling puppies], хотя *в* иных контекстах эти замены были бы уместны. — *Примеч. ред.*

и ложноотрицательные связи. Более того, поскольку термины, представленные в автоматическом тезаурусе, и так сильно коррелированны в документе (причем часто для создания тезауруса и индексирования используется одна и та же коллекция), эта форма расширения запроса не позволяет найти много новых документов.

Расширение запроса часто эффективно повышает полноту. Однако ручное создание тезауруса и его дальнейшее обновление с учетом научного развития области и изменения терминологии связано с большими затратами. В принципе, тезаурус предметной области необходим: универсальные тезаурусы и словари плохо покрывают богатую и специфичную терминологию научных дисциплин. В то же время расширение запроса может сильно снизить точность, особенно если запрос содержит неоднозначные термины. Например, если пользователь формулирует запрос `interest rate`, расширение запроса до `interest rate fascinate evaluate` вряд ли целесообразно. Итак, расширение запроса менее полезно, чем применение метода обратной связи по релевантности, хотя оно может оказаться так же эффективно, как метод обратной связи по псевдорелевантности. Преимущество расширения запроса в том, что оно в большей степени понятно пользователю.

? ***Упражнение 9.7.*** Допустим, что матрица A является двоичной матрицей встречаемости термина в документе. Какой смысл имеют элементы матрицы C?

9.3. Библиография и рекомендации для дальнейшего чтения

Информационно-поисковые системы рано столкнулись с проблемой вариантных выражений, когда слова запроса не содержатся в документе, несмотря на то что сам документ является релевантным этому запросу. Ранний эксперимент, проведенный примерно в 1960-м году, на который ссылается Свансон (Swanson, 1988), выяснил, что только одиннадцать из двадцати трех документов, индексированных по теме *toxicity*, содержали слово с корнем *toxi*. Кроме того, существует проблема перевода, когда пользователи не знают, какие термины использует документ. Блэр и Марон (Blair and Maron, 1985) пришли к выводу, что “пользователю слишком сложно предсказать точные слова, словосочетания и фразы, которые будут использованы во всех (или в большинстве) релевантных документах и только (или в основном) в этих документах”.

Метод обратной связи по релевантности с помощью модели векторного пространства впервые описан в статье Солтона (Salton, 1971*b*), алгоритм Роккио — в его работе (Rocchio, 1971), а вариант Ide dec–hi вместе с оценкой нескольких вариантов — в статье Иде (Ide, 1971). Другой вариант обратной связи сводится к рассмотрению *всех* документов в коллекции как нерелевантных, кроме тех, которые были оценены как релевантные, вместо документов, явно оцененных как нерелевантные. Однако Шютце и др. (Schütze et al., 1995) и Сингал и др. (Singhal et al., 1997) показали, что лучшие результаты можно получить, используя только документы, близкие к запросу, а не все документы. Более поздние исследования описаны в статьях Солтона и Бакли (Salton and Buckley, 1990), Ризлера и др. (Riezler et al., 2007), посвященных статистическому NLP-подходу к методу RF, а также в недавнем обзоре Рутвена и Лалмаса (Ruthven and Lalmas, 2003).

Качество интерактивных систем RF обсуждается в работах Солтона, Хармана, Бакли и др. (Salton, 1989; Harman, 1992; Buckley et al., 1994*b*). Эксперименты с участием поль-

зователей по исследованию эффективности обратной связи по релевантности описаны в работе Конеманн и Белкина (Koenemann and Belkin, 1996).

Традиционно наиболее известным тезаурусом английского языка считается *тезаурус Роже* (Roget's thesausus), описанный в работе (Roget, 1946). В настоящее время исследователи практически всегда используют тезаурус WordNet не только потому, что он свободно распространяется, но и благодаря его богатой структуре связей. Этот тезаурус доступен по адресу `http://wordnet.princeton.edu`.

Автоматическая генерация тезаурусов обсуждалась в работах Кью и Фрая (Qui and Frei, 1993) и Шютце (Schütze, 1998). Использование локальных и глобальных методов расширения запросов исследовано в работе Ксу и Крофта (Xu and Croft, 1996).

Глава 10

XML-поиск

Информационно-поисковые системы часто противопоставляются реляционным базам данных. Традиционно поисковые системы извлекали информацию из *неструктурированного текста*, под которым понимается "простой" текст без разметки. В противоположность им базы данных предназначены для обработки запросов к *реляционным данным* (relational data), представляющим собой совокупность записей, хранящих значения заранее определенных атрибутов[1], таких как табельный номер сотрудника, должность и зарплата. Между информационно-поисковыми системами и базами данных существуют фундаментальные различия в модели поиска, структурах данных и языках запроса (табл. 10.1).[2]

Таблица 10.1. Сравнение поиска в реляционных базах данных (Relational Database — RDB), а также информационный поиск по неструктурированным и структурированным данным. До сих пор нет единого мнения о том, какой из методов лучше всего подходит для структурного поиска, хотя многие исследователи уверены, что язык XQuery со временем станет стандартом для структурированных запросов.

	Поиск RDB	*Неструктурированный поиск*	*Структурированный поиск*
Объект	Записи	Неструктурированные документы	Деревья с текстом на листьях
Модель	Реляционная модель	Векторное пространство и другие модели	?
Основная структура данных	Таблица	Инвертированный индекс	?
Запросы	SQL	Свободные текстовые запросы	?

Некоторые задачи поиска текста в структурированных данных наиболее эффективно решаются с помощью реляционных баз данных. Например, если таблица сотрудников содержит атрибут с кратким текстовым описанием его обязанностей и вы хотите найти всех сотрудников, занимающихся выставлением счетов (invoicing), то SQL-запрос может выглядеть так.

```
select lastname from employees where job_desc like 'invoic%'
```

Такого запроса может оказаться достаточно для того, чтобы удовлетворить вашу информационную потребность с высокой точностью и полнотой.

---

[1] Термин "атрибут" в отечественной литературе предпочтительнее термина "поле" и столь же популярен в англоязычной литературе. — *Примеч. ред.*

[2] В большинстве современных баз данных возможен полнотекстовый поиск по текстовым полям. Как правило, это значит, что создан инвертированный индекс и можно осуществлять логический поиск или поиск в векторном пространстве, т.е. технологии баз данных эффективно объединены с технологиями информационно-поисковых систем.

Однако многие источники структурированных данных, содержащие текст, лучше всего моделируются как структурированные документы, а не как реляционные данные. В дальнейшем будем называть поиск по таким документам *структурированным* (structured retrieval). Запросы на структурированный поиск могут быть как структурированными, так и неструктурированными, но в этой главе мы будем считать, что коллекция содержит только структурированные документы. Приложения структурированного поиска включают в себя электронные библиотеки, патентные базы данных, блоги, тексты, в которых некоторые сущности, например имена людей и географические названия, помечены (в процессе так называемой *разметки именованных сущностей* (named entity tagging)), а также тексты, созданные с помощью офисных приложений, таких как OpenOffice, в которых документы сохраняются как размеченные тексты. Во всех этих приложениях мы хотим иметь возможность обрабатывать запросы, сочетающие текстуальные и структурные критерии. Приведем несколько примеров таких запросов: `give me a full-length article on fast fourier transforms` (электронные библиотеки), `give me patents whose claims mention RSA public key encryption and that cite` `USA patent 4 405 829` (патенты) и `give me articles about sightseeing tours of the Vatican and the Coliseum` (текст с именованными сущностями). Эти три запроса являются примерами таких структурных запросов, которые не могут корректно обрабатываться информационно-поисковыми системами, не выполняющими ранжирование. Как указано в примере 1.1, модели поиска без ранжирования, такие как булева модель, отличаются низкой полнотой. Например, система поиска без ранжирования вернет большое количество статей, в которых упоминаются слова Vatican, Coliseum и sightseeing tours, но не присвоит ранг наиболее релевантным документам. Известно, что большинство пользователей плохо умеют формулировать структурные ограничения. Например, пользователи могут не знать, какие структурные элементы можно применять в поисковой системе. В нашем примере пользователь может быть не уверен, в какой форме лучше задать запрос: `sightseeing AND COUNTRY: Vatican AND LANDMARK:Coliseum` или `sightseeing AND STATE:Vatican AND BUILDINGS:Coliseum`, или в какой-то другой. Пользователи могут также совершенно не знать о принципах работы интерфейсов структурированного и расширенного поиска или не желать их использовать. В этой главе мы покажем, как адаптировать методы ранжированного поиска к структурированным документам, чтобы решить эти проблемы.

Мы рассмотрим лишь один стандарт кодирования структурированных документов: язык *Extensible markup language*, или *XML*, который в настоящее время используется наиболее широко. Мы не будем глубоко изучать отличия языка XML от других языков разметки текстов, таких как HTML и SGML. Однако большинство из того, что будет сказано в этой главе, относится и к другим языкам разметки.

В контексте информационного поиска язык XML интересует нас лишь как язык для кодирования текстов и документов, хотя, возможно, он более широко используется для кодирования нетекстовых данных. Например, можно перевести данные из корпоративной системы планирования ресурсов в формат XML, а затем ввести в аналитическую программу, чтобы построить графики для презентации. Этот тип приложений языка XML называется *ориентированным на данные* (data-centric), поскольку в них доминируют числовые данные и нетекстовые значения атрибутов, а текст обычно образует небольшую часть всех данных. Основные средства языка XML, ориентированного на данные, хранятся в базах данных — в противоположность методам на основе инвертированных индексов для языка XML, ориентированного на текст. Именно второй тип методов рассматривается в данной главе.

На протяжении этой главы XML-поиск называется *структурированным поиском*. Некоторые исследователи предпочитают термин *полуструктурированный поиск* (semi-structural retrieval), чтобы отличить XML-поиск от обработки запросов в базах данных. Мы используем терминологию, которая широко распространена в среде сообщества, занимающегося XML-поиском. Например, обычно XML-запросы называются *структурированными*, а не *полуструктурированными*. Термин *структурированный поиск* редко используется в контексте запросов к базам данных и в этой книге относится исключительно к XML-поиску.

Существует второй тип задач, связанных с информационно-поисковыми системами и занимающих промежуточное место между поиском по неструктурированным данным и запросами к реляционным базам данных: параметрический и зонный поиск[3], который был рассмотрен в разделе 6.1. В модели параметрического и зонного поиска предусмотрены поля параметров (реляционные атрибуты наподобие *data* или *file-size*) и зоны — тестовые атрибуты, каждый из которых хранит фрагмент неструктурированного текста как значение, например, *author* или *title* (см. рис. 6.1). Эта модель данных является плоской, т.е. в ней нет вложенности атрибутов. Количество атрибутов мало. В противоположность этому XML-документы имеют более сложную древовидную структуру, в которой, как показано на рис. 10.1, атрибуты вложены. Количество атрибутов и узлов в этой структуре больше, чем при параметрическом и зонном поиске.

```
<play>
<author>Shakespeare</author>
<title>Macbeth</title>
<act number="I">
<scene number="vii">
<title>Macbeth's castle</title>
<verse>Will I with wine and wassail ... </verse>
</scene>
</act>
</play>
```

Рис. 10.1. XML-документ

Изложив в разделе 10.1 основные концепции, связанные с языком XML, мы рассмотрим проблемы, возникающие при XML-поиске (раздел 10.2). Далее мы опишем модель векторного пространства для XML-поиска (раздел 10.3). В разделе 10.4 представлен проект INEX (INitiative for the Evaluation of XML Retrieval), в рамках которого проводится оценка общих заданий методов поиска. Этот проект реализуется уже несколько лет и стал наиболее важным мероприятием в области XML-поиска. В разделе 10.5 мы обсудим разницу между подходами к языку XML, ориентированными на данные и текст.

10.1. Основные концепции языка XML

XML-документ (XML document) — это упорядоченное и размеченное дерево. Каждый его узел — это *XML-элемент* (XML element). Он записывается с помощью открывающего и закрывающего тегов (tag). Элемент может иметь один или несколько *XML-атрибутов* (XML attributes). В XML-документе, представленном на рис. 10.1, эле-

[3] Или же “атрибутивный и зонный поиск”. — *Примеч. ред.*

мент *scene* заключен между двумя метками `<scene ...>` и `</scene>`. Он имеет атрибут *number* со значением *vii* и два дочерних элемента: *title* и *verse*.

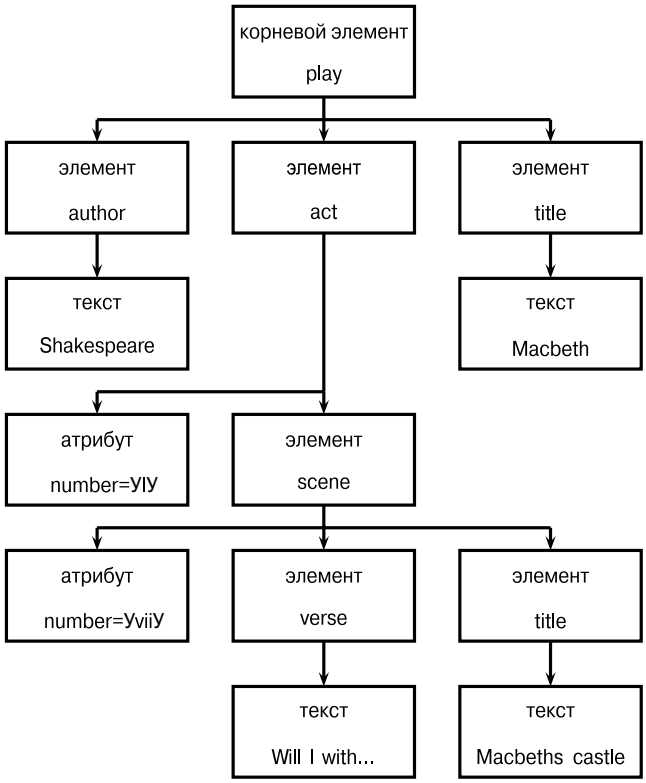


Рис. 10.2. XML-документ как упрощенный DOM-объект

На рис. 10.1 документ, представленный на рис. 10.2, показан в виде дерева. *Листья дерева* (leaf nodes) содержат текст, например `Shakespeare`, `Macbeth` и `Macbeth's castle`. *Внутренние узлы дерева* (internal nodes) описывают либо структуру документа (*title*, *act* и *scene*), либо метаданные (*author*).

Как правило, для доступа к XML-документам и их обработки используется объектная модель документа, или *DOM* (Document Object Model). Эта модель представляет элементы, атрибуты и текст внутри элементов как узлы дерева. Рис. 10.2 представляет собой упрощенное представление документа, изображенного на рис. 10.1, с помощью модели DOM.[4] С помощью инртерфейса прикладного программирования модели DOM можно обработать XML-документ, начиная с корневого узла и спускаясь вниз по дереву от родительских узлов к дочерним.

Для перебора путей в коллекции XML-документов обычно используется *язык XPath*. В данной главе эти пути называются *контекстами XML* (XML contexts) или просто *контекстами*. Для наших целей можно ограничиться лишь небольшим подмножеством

[4] Это представление упрощено по многим параметрам. Например, мы не показали *корневой* узел и в *текстовых* узлах не показан текст. См. `www.w3.org/DOM/`.

языка XPath. Выражение языка XPath `node` выбирает все узлы по заданному имени. Последовательные элементы пути разделяются косой чертой: команда `act/scene` выбирает элементы `scene`, родительский узел которой является элементом `act`. Двойная косая черта означает, что в промежутке может встретиться произвольное количество элементов: команда `play//scene` выбирает все элементы *scene*, встречающиеся в элементе *play*. На рис. 10.1 это множество состоит из одного элемента *scene*, достичь которого можно из корня через узлы *play*, *act* и *scene*. Начальная косая черта обозначает начало пути, которое расположено в корневом элементе. На рис. 10.2 команда `/play/title` выбирает названия пьес, команда `/play//title` выбирает множество, состоящее из двух элементов (названия пьесы и названия сцены), а команда `/scene/title` не выбирает ни одного элемента. Для удобства обозначений мы допускаем, что финальный элемент пути является проиндексированным термином из лексикона коллекции, и отделяем его от пути символом #, хотя это и не соответствует стандарту языка XPath. Например, команда `title#"Macbeth"` выбирает все названия, содержащие термин `Macbeth`.

В этой главе нам понадобится понятие *схемы* (schema). Схема накладывает определенные ограничения на структуру XML-документов, допустимых в конкретном приложении. Схема для пьес Шекспира может поставить условие, что сцены могут быть только дочерними элементами актов и только акты и сцены могут иметь атрибут *number*. Существует два стандарта схем для XML-документов: *XML DTD* (document type definition) и *XML* (XML schema). Пользователи могут задавать структурированные запросы системе поиска по XML-документам, только если у них есть хотя бы минимальные знания о схеме коллекции.

Общим форматом XML-запросов является *NEXI* (Narrowed Extended XPath I). Его пример показан на рис. 10.3. Для удобства представления мы разделили запрос на четыре строки, хотя он должен читаться, как одно целое, без разбиения на строки. В частности, команда `//section` вложена в команду `//article`.

```
//article
[.//yr = 2001 or .//yr = 2002]
//section
[about(.,summer holidays)]
```

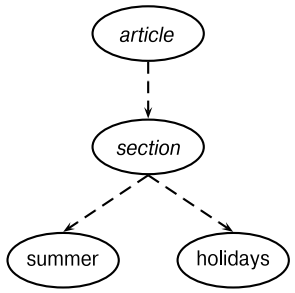


Рис. 10.3. XML-запрос в формате NEXI и его частичное представление в виде дерева

Запрос, представленный на рис. 10.3, описывает поиск разделов, посвященных летним каникулам, в статьях, опубликованных с 2001 по 2002 год. Как и в языке XPath, двойная косая черта означает, что на пути может встречаться произвольное количество элементов. Точка в выражении внутри квадратных скобок ссылается на элемент, который это выражение уточняет. Так, оператор `[.//yr = 2001 or .//yr = 2002]` уточняет команду `//article`. Таким образом, точка в данном случае относится к ко-

манде `//article`. Аналогично точка в выражении `[about(.,summer holidays)]` относится к разделу, который уточняется данным выражением.

Условия `yr` (год) являются ограничениями реляционного атрибута. Иначе говоря, рассматриваться должны только статьи, атрибут `yr` которых равен 2001 или 2002 (т.е. содержат элемент, атрибут `yr` которого равен 2001 или 2002). Выражение `about` является ранжирующим условием: разделы, встречающиеся в статьях правильного типа, ранжируются с учетом их релевантности теме `summer holidays`.

Ограничения реляционного атрибута обрабатываются с помощью предварительной или последующей фильтрации: все элементы, не соответствующие ограничениям реляционного атрибута, просто исключаются из совокупности результатов. В данной главе мы не будем останавливаться на том, как это сделать эффективнее, а вместо этого сосредоточим свое внимание на основной проблеме в XML-поиске — как ранжировать документы по критериям, заданным в поле `about` запроса NEXI.

Отбрасывая реляционные атрибуты, документы можно представить в виде деревьев, содержащих узлы единственного типа — элементы. Иначе говоря, из XML-документа удаляются все узлы атрибутов, такие как атрибут *number* на рис. 10.2. Поддерево документа, представленного на рис. 10.2, в виде дерева, содержащего лишь узлы элементов, показано на рис. 10.4 (дерево d_1).

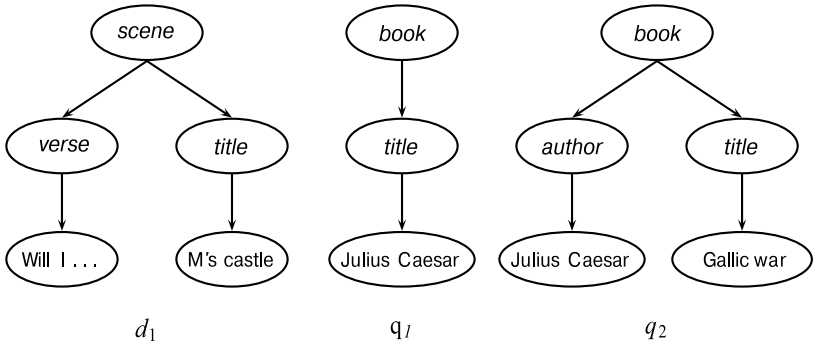


Рис. 10.4. Представление XML-документов и запросов в виде деревьев

Запросы можно точно так же представить в виде дерева. Такой подход к языку запросов называется *запросом по образцу* (query-by-example), поскольку пользователь задает запросы, создавая объекты, удовлетворяющие тому же самому формальному описанию, что и документы. На рис. 10.4 дерево q_1 описывает поиск книг, названия которых имеют высокий вес по отношению к ключевым словам `Julius Caesar`, а дерево q_2 — поиск книг, имена авторов которых имеют высокий вес по отношению к ключевым словам `Julius Caesar`, а названия имеют высокий вес по отношению к ключевым словам `Gallic war`.[5]

[5] Для полного представления семантики запросов в формате NEXI необходимо пометить один из узлов дерева как “целевой”, например узел *section* на дереве, представленном на рис. 10.3. Без этого данное дерево описывает не поиск разделов, вложенных в статьи (как указано в формате NEXI), а поиск статей, содержащих эти разделы.

10.2. Проблемы, связанные с XML-поиском

В этом разделе мы обсудим несколько проблем, которые делают структурированный поиск более сложным, чем неструктурированный. Напомним основные предположения, на которых основывается структурированный поиск: коллекция состоит из структурированных документов, а запросы могут быть как структурированными (как на рис. 10.3), так и неструктурированными (например, `summer holiday`).

Первая проблема, связанная со структурированным поиском, заключается в том, что пользователи хотят получить части документов (т.е. XML-элементы), а не документы целиком, как в неструктурированном поиске. Если мы запрашиваем поиск по пьесам Шекспира, содержащим фразу `Macbeth's castle`, то что вернуть в ответ: сцену, акт или всю пьесу (см. рис. 10.1)? В этом случае пользователи, вероятно, ищут сцену. С другой стороны, в ответ на неконкретный запрос на слово `Macbeth` следует возвращать пьесу с таким названием, а не ее часть.

Одним из критериев выбора наиболее подходящей части документа является *принцип структурированного поиска документов* (structured document retrieval principle).

> **Принцип структурированного поиска документов.** Система должна всегда находить часть документа, которая наиболее точно соответствует запросу.

Этот принцип обосновывает стратегию поиска, который выявляет элемент наименьшего размера, содержащий искомую информацию, и не углубляется на более низкий уровень. Однако этот принцип трудно реализовать в виде алгоритма. Рассмотрим запрос `title#"Macbeth"` применительно к рис. 10.1. Этому запросу соответствуют как название трагедии, *Macbeth*, так и заглавие сцены vii из первого акта, *Macbeth's castle*. Тем не менее, поскольку название трагедии записано в узле более высокого уровня, предпочтение отдается именно ему. Решение, на каком уровне дерева следует остановить поиск, представляет собой трудную задачу.

Одновременно с задачей, какую часть документа следует вернуть пользователю, необходимо решить, какие части документа подлежат индексированию. В разделе 2.1.2 мы обсудили необходимость ввести понятие единицы документа, или *единицы индексирования* (indexing unit), для процессов индексирования и поиска. В неструктурированном поиске обычно ясно, какой должна быть единица документа: файлы на вашем компьютере, сообщения электронной почты, веб-страницы и т.д. В структурированном поиске существует множество разных подходов к определению единицы индексирования.

Один из этих подходов предусматривает группировку узлов в непересекающиеся псевдодокументы, как показано на рис. 10.5. В этом примере в качестве единиц индексации используются непересекающиеся книги, главы и разделы. Например, крайняя слева единица индексирования содержит только те части дерева, находящиеся под узлом *book*, которые еще не стали частью других единиц индексирования. Недостаток этого подхода заключается в том, что такие псевдодокументы могут не иметь смысла в глазах пользователя, поскольку между ними может не быть связи. Например, крайняя слева единица индексирования на рис. 10.5 объединяет три разных элемента: *class*, *author* и *title*.

В качестве единицы индексирования можно также использовать один из крупных элементов, например элемент *book* в коллекции книг или элемент *play* в собрании сочинений Шекспира. С помощью постобработки результатов поиска можно найти наиболее подходящий подэлемент. Например, в ответ на запрос `Macbeth's castle` можно вернуть пьесу *Macbeth*, которая затем обрабатывается дополнительно, чтобы идентифици-

ровать акт I, сцену vii, как элемент, наиболее точно соответствующий запросу. К сожалению, этот двухэтапный процесс поиска во многих случаях не позволяет вернуть наилучший подэлемент, поскольку релевантность всей книги часто плохо предсказывает релевантность ее небольших частей.

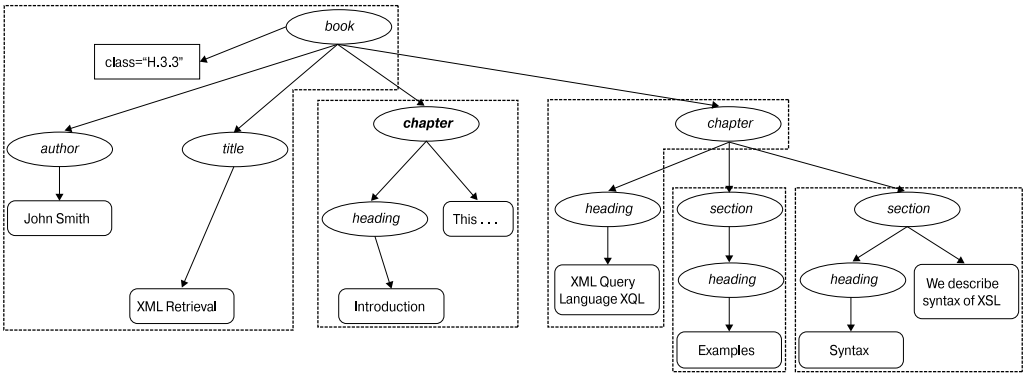


Рис. 10.5. Разделение XML-документа на непересекающиеся единицы индексирования

Вместо поиска крупных единиц и идентификации подэлементов (сверху вниз) можно найти все листья, выбрать среди них наиболее релевантные и расширить до более крупных единиц в ходе дополнительной обработки (снизу вверх). При выполнении запроса `Macbeth's castle`, показанного на рис. 10.2, на первом проходе найдем заголовок *Macbeth's castle*, а затем на этапе последующей обработки решим, что именно возвращать: заголовок, сцену, акт или всю пьесу. Этот подход имеет тот же недостаток, что и предыдущий: релевантность листового элемента часто слабо связана с релевантностью элементов, которым он принадлежит.

Наименее ограничительным подходом является индексирование всех элементов. Это также проблематично. Многие XML-элементы не являются осмысленными результатами поиска, например типографские элементы вида `<b>definitely</b>` или номер ISBN, которые невозможно интерпретировать без контекста. Кроме того, индексация всех элементов означает, что результаты поиска будут слишком избыточными. Для запроса `Macbeth's castle` и документа, изображенного на рис. 10.2, мы получим в ответ элементы *play*, *act*, *scene* и *title*, лежащие на пути от корня до узла `Macbeth's castle`. Листовой узел в этом случае будет четыре раза повторяться в списке результатов: один раз — непосредственно и три раза — как часть других элементов. Назовем элементы, содержащиеся в других элементах, *вложенными* (nested). Возврат избыточных вложенных элементов в списке результатов является не слишком дружественным актом по отношению к пользователю.

Поскольку избыточность порождается вложенными элементами, естественно ограничить совокупность элементов, подлежащих возврату. Существует несколько стратегий такого ограничения.

- Отбросить все маленькие элементы
- Отбросить все элементы, которые пользователи обычно не ищут (для этого необходимо, чтобы система XML-поиска регистрировала такую информацию)

- Отбросить все элементы, которые эксперты обычно считают нерелевантными (если существует возможность получить оценку релевантности)
- Хранить только элементы, которые разработчик системы или библиотекарь считает полезными для поиска

В большинстве этих подходов совокупности результатов все же содержат вложенные элементы. Таким образом, для уменьшения избыточности требуется удалить некоторые элементы на этапе постобработки. В качестве альтернативы можно свернуть несколько вложенных элементов в списке результатов и использовать *подсветку* терминов запроса, для того чтобы привлечь внимание пользователя к релевантным фрагментам. Если термины запроса выделены подсветкой, то просмотр элементов среднего размера (например, разделов) займет ненамного больше времени, чем просмотр небольших подэлементов (например, абзацев). Таким образом, если раздел и параграф одновременно оказываются в списке результатов, то достаточно показать раздел. Дополнительное преимущество этого подхода состоит в том, что абзац представляется вместе со своим контекстом (например, с разделом, содержащим этот абзац). Этот контекст может помочь интерпретировать абзац (например, выявить источник информации), даже если параграф сам по себе уже удовлетворяет запрос.

Если пользователь знает схему коллекции и может указать желательные типы элементов, то проблема избыточности разрешается легко, так как немногие вложенные элементы имеют один и тот же тип. Однако, как указывалось во введении, пользователи часто не знают, как называется тот или иной элемент в коллекции (например, `Vatican` — это *страна* или *город*?) или могут вообще не знать, как составить структурированный запрос.

Проблема вложенных узлов в XML-поиске заключается в том, что разработчики должны различать разные контексты термина при вычислении статистики встречаемости терминов для ранжирования, в частности при подсчете обратной документной частоты (idf), как указано в разделе 6.2.1. Например, термин `Gates` в узле *author* никак не связан с множественным числом слова `gate` в узле *section*. В таком случае мало смысла в том, чтобы вычислять единую документную частоту слова `Gates`.

Для решения этой задачи можно вычислить показатель idf для пар “XML-контекст/термин”, например вычислить разные веса idf для `author#"Gates"` и `section#"Gates"`. К сожалению, эта схема наталкивается на проблемы, связанные с разреженными данными: многие пары “XML-контекст” появляются слишком редко, чтобы надежно оценить их документную частоту (см. раздел 13.2). В качестве компромиссного решения для различения контекстов можно рассматривать только родительский узел *x* термина, а остальную часть пути от корня до узла *x* игнорировать. Тем не менее существуют объединения контекстов, которые оказываются опасными в этой схеме. Например, мы не можем различить имена авторов и названия корпораций, если у обоих родительским узлом является узел *name*. Однако большинство важных различий, скажем, между `author#"Gates"` и `section#"Gates"`, распознаются хорошо.

Во многих случаях в коллекции возникает несколько разных XML-схем, поскольку XML-документы в информационно-поисковых системах часто поступают из нескольких источников. Это явление называется *неоднородностью схем* (schema heterogeneity) или *разнообразием схем* (schema diversity) и представляет собой еще одну проблему. Как показано на рис. 10.6, сравнимые элементы могут иметь разные имена: *creator* в дереве d_2 и *author* в дереве d_3. В других случаях схемы могут иметь разную структурную организа-

цию: в дереве q_3 имена авторов являются непосредственными наследниками узла *author*, а в дереве d_3 существуют промежуточные узлы *firstname* и *lastname*. Если применить строгое сравнение деревьев, то по запросу q_3 не будут найдены ни документ d_2, ни документ d_3, хотя оба являются релевантными. Если применить приближенное сравнение имен элементов в сочетании с полуавтоматическим сопоставлением разных структур документов, то эту проблему можно разрешить. При этом установление соответствия между элементами в разных схемах с участием человека обычно эффективнее, чем автоматические методы.

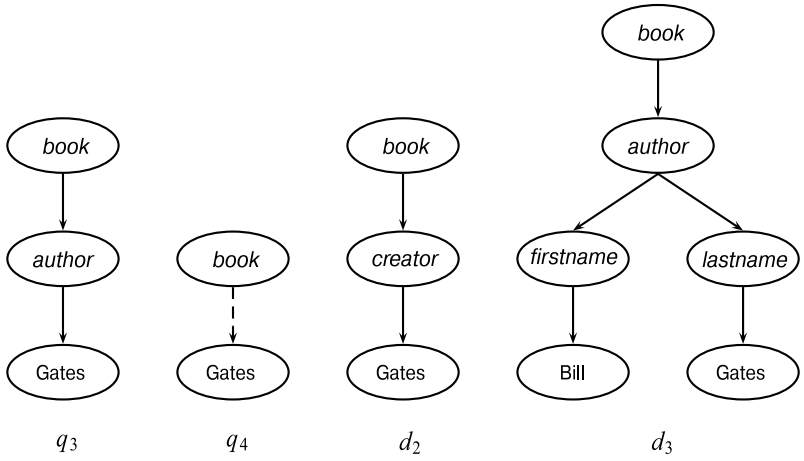


Рис. 10.6. Неоднородность схем: промежуточные узлы и несовпадающие имена

Неоднородность схем обычно является одной из причин несовпадений между парами "запрос–документ" вида q_3/d_2 и q_3/d_3. Другая причина этого явления заключается в том, что пользователи часто не знают имен элементов и структуры схем коллекций, в которых осуществляется поиск. Это ставит перед нами проблему разработки интерфейса для XML-поиска. В идеальном случае пользовательский интерфейс должен показывать древовидную структуру коллекции и позволять пользователю указывать элементы, к которым относится запрос. Если принять этот подход, то интерфейс для подачи структурированных запросов является намного более сложным, чем простое поле ввода запроса из ключевых слов в неструктурированном поиске.

Мы также можем оказать пользователю поддержку путем интерпретации всех отношений "родитель–потомок" в запросах как наследования с произвольным количеством промежуточных узлов. Такие запросы называются *расширенными* (extended queries). Дерево, изображенное на рис. 10.3, и дерево q_4 на рис. 10.6 представляют собой примеры расширенных запросов. Ветви на этих деревьях, соответствующие отношениям наследования, изображены как пунктирные стрелки. В дереве q_4 пунктирная стрелка связывает элементы *book* и `Gates`. В нотации, имитирующей язык XPath, это отношение можно обозначить как `book//#"Gates"`: книга, которая где-то внутри себя содержит слово `Gates`, причем путь от узла *book* до узла `Gates` может быть произвольно долгим. Запрос на псевдо-XPath, который дополнительно требует, чтобы слово `Gates` содержалось в элементе *section* узла *book*, выглядит так: `book//section//#"Gates"`. Для пользователей удобно иметь возможность формулировать такие расширенные запросы, не ука-

зывая точную структурную конфигурацию, в которой должен встретиться термин запроса, так как в этом случае они могут не беспокоиться о точной конфигурации и им не обязательно знать схему коллекции, чтобы использовать ее в запросе.

На рис. 10.7 продемонстрирована ситуация, в которой пользователь ищет главу под названием `FFT` (запрос q_5). Допустим, что в коллекции такой главы нет, но есть ссылки на книги по этой теме (дерево d_4). Ссылка на книгу по теме FTT — это не совсем то, что искал пользователь, но все же лучше, чем ничего. Расширенный запрос здесь не поможет: в ответ на запрос q_6 пользователь ничего не получит. В этой ситуации можно интерпретировать структурные ограничения в запросе как подсказки, а не как строгие условия. Как будет показано в разделе 10.4, пользователи предпочитают ослабленную интерпретацию структурных ограничений. Элементы, не точно соответствующие структурным ограничениям, должны получать более низкие веса, но не должны отсутствовать в списке результатов.

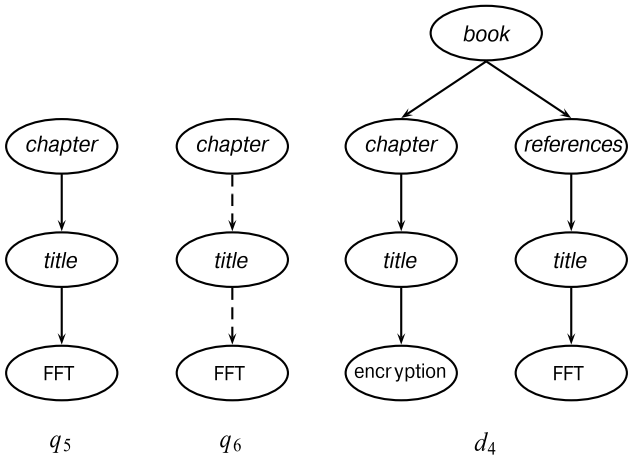


Рис. 10.7. Структурные несовпадения между двумя запросами и документом

10.3. Модель векторного пространства для XML-поиска

В этом разделе описывается простая модель векторного пространства для XML-поиска. Мы не претендуем на полное описание современного состояния дел в этой области, а лишь хотим дать читателям представление о том, как можно представить и найти документы в системах XML-поиска.

Для того чтобы учесть структуру в ходе поиска, проиллюстрированного на рис. 10.4, мы должны потребовать, чтобы книга с названием *Julius Caesar* соответствовала запросу q_1 и не соответствовала запросу q_2 (или же соответствовала ему с более низким рангом). В неструктурированном поиске термину `Caesar` соответствовала бы отдельная ось в векторном пространстве. В XML-поиске мы должны отличать слово `Caesar` в названии книги от имени автора `Caesar`. Для этого можно выделить в векторном пространстве отдельную ось для кодирования слова вместе с его позицией в XML-дереве.

Это представление показано на рис. 10.8. Сначала каждый текстовый узел (который в нашем примере всегда является листом) разделяется на несколько узлов, по одному на слово. Таким образом, лист `Bill Gates` разделяется на два листа — `Bill` и `Gates`. Затем каждая размерность в векторном пространстве определяется как *лексикализированное поддерево документов* (lexicalized subtree), т.е. поддеревьев, содержащих хотя бы один проиндексированный термин. Подмножество возможных лексикализированных поддеревьев показано на рисунке, но существуют и другие, например поддерево, соответствующее всему документу с удаленным узлом `Gates`. Теперь можно представлять запросы и документы как векторы в пространстве поддеревьев и сравнивать их друг с другом. Это значит, что для описания XML-поиска можно использовать формализм векторного пространства из главы 6. Основное отличие заключается в том, что размерности векторного пространства в неструктурированном поиске соответствуют терминам индекса, а в XML-поиске — лексикализированным поддеревьям.

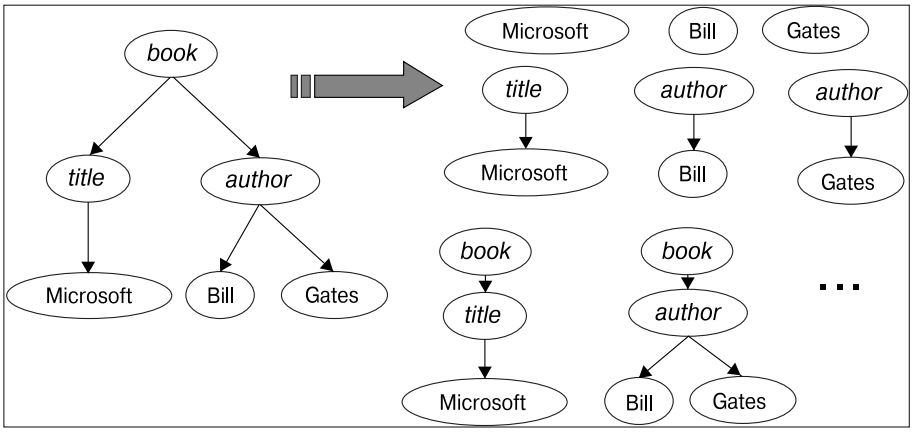


Рис. 10.8. Отображение XML-документа (слева) в множество лексикализированных поддеревьев (справа)

Существует противоречие между размерностью пространства и точностью результатов поиска. Если просто ограничить размерности проиндексированными терминами, то получится система поиска, основанная на стандартной модели векторного пространства, которая будет находить много документов, не соответствующих структуре запроса (например, `Gates` в заглавии, а не среди фамилий авторов). Если для каждого лексикализированного поддерева, встречающегося в коллекции, выделять отдельную ось, то размерность пространства станет очень большой. В качестве компромисса можно индексировать все пути, заканчивающиеся на отдельном термине индекса, т.е. все пары "XML-контекст/термин". Назовем такую пару "XML-контекст/термин" *структурным термином* (structural term) и обозначим его через <*c*, *t*>. Эта пара состоит из XML-контекста *c* и проиндексированного термина *t*. Документ, представленный на рис. 10.8, имеет девять структурных терминов. Семь из них показаны (например, `"Bill"` и `"Author#"Bill"`), а два нет: `/Book/Author#"Bill"` и `/Book/Author#"Gates"`. Дерево с листьями `Bill` и `Gates` представляет собой лексикализированное поддерево, не являющееся структурным термином. Здесь использована система обозначений структурных терминов, имитирующая нотацию языка XPath.

Как указывалось в предыдущем разделе, пользователи плохо помнят детали схемы и редко конструируют запросы, согласованные с этой схемой. Следовательно, все запросы следует интерпретировать как расширенные, т.е. для любой пары "родительский узел–дочерний узел" из запроса в документе может быть любое количество промежуточных узлов. Например, запрос q_5 на рис. 10.7 мы интерпретируем как запрос q_6.

Тем не менее мы по-прежнему отдаем предпочтение документам, соответствующим структуре запроса за счет вставки меньшего количества дополнительных узлов. Вычисляя вес для каждого соответствия, мы обеспечиваем согласованность результатов поиска с этим выбором. Меру схожести пути c_q в запросе и пути c_d в документе можно оценить с помощью функции *схожести контекстов* `Cr` (context resemblance).

$$Cr(c_q, c_d) = \begin{cases} \dfrac{1+|c_q|}{1+|c_d|}, & \textit{если путь } c_q \textit{ соответствует пути } c_d, \\ 0 & \textit{— в противоположном случае} \end{cases} \tag{10.1}$$

Здесь $|c_q|$ и $|c_d|$ — количество узлов на пути запроса и пути документа соответственно, и путь c_q соответствует пути c_d, если и только если путь c_q можно преобразовать в путь c_d с помощью вставки нескольких дополнительных узлов. Для двух примеров из рис. 10.6 $Cr(c_{q_4}, c_{d_2}) = 3/4 = 0{,}75$ и $Cr(c_{q_4}, c_{d_3}) = 3/5 = 0{,}6$, где c_{q_4}, c_{d_2} и c_{d_3} — релевантные пути от корня до листа в деревьях q_4, d_2 и d_3 соответственно. Если запрос q и документ d идентичны, то $Cr(c_q, c_d) = 1{,}0$.

Окончательный вес документа вычисляется с помощью одного из вариантов косинусной меры близости (6.10), который мы назвали `SimNoMerge`.

$$SimNoMerge(q,d) = \sum_{c_k \in B} \sum_{c_l \in B} Cr(c_k, c_l) \sum_{t \in V} weight(q,t,c_k) \frac{weight(d,t,c_l)}{\sqrt{\sum_{c \in B, t \in V} weight^2(d,t,c)}} \tag{10.2}$$

Здесь V — лексикон неструктурированных терминов, B — множество всех XML-контекстов, $weight(q, t, c)$ и $weight(d, t, c)$ — веса термина t в XML-контексте c в запросе q и в документе d соответственно. Эти веса вычисляются по формулам из главы 6, например $idf_t \cdot wf_{t,d}$. Как указывалось в разделе 10.2, обратная документная частота idf_t зависит от того, какие элементы используются для вычисления частоты df_t. Мера схожести `SimNoMerge`(q, d) не является истинной косинусной мерой, поскольку ее значение может быть больше 1,0 (упражнение 10.11). Для нормировки длины документа производится деление веса на $\sqrt{\sum_{c \in B, t \in V} weight^2(d,t,c)}$ (раздел 6.3.1). Для простоты нормировка длины запроса не производится. К тому же она не влияет на ранжирование. Отметим лишь, что для данного запроса коэффициент нормировки $\sqrt{\sum_{c \in B, t \in V} weight^2(q,t,c)}$ остается постоянным для всех документов.

Алгоритм для вычисления функции `SimNoMerge` для всех документов в коллекции приведен на рис. 10.9. Массив *normalizer* на рис. 10.9 содержит числа $\sqrt{\sum_{c \in B, t \in V} weight^2(d,t,c)}$, вычисленные по формуле (10.2) для каждого документа.

На рис. 10.10 показано, как в функции *SimNoMerge* вычисляется мера схожести запроса и документа. $\langle c_1, t \rangle$ — один из структурных терминов запроса. Последовательно находим все инвертированные списки для структурных терминов $\langle c', t \rangle$ с термином t. На рисунке показаны три инвертированных списка. Для первого $Cr(c_1, c_1) = 1{,}0$, поскольку

оба контекста идентичны. Второй контекст не соответствует контексту c_1: $Cr(c_1, c_2) = 0$, поэтому второй инвертированный список игнорируется. Мера схожести контекстов c_1 и c_3 равна 0,63. В данном примере наибольший ранг имеет документ d_9, мера схожести которого с запросом равна $1{,}0 \times 0{,}2 + 0{,}63 \times 0{,}6 = 0{,}578$. Для простоты вес запроса $<c_1, t>$ предполагается равным 1,0.

```
ScoreDocumentWithSimNoMerge(q, B, V, N, normalizer)
1   for n ← 1 to N
2   do score[n] ← 0
3   for each ⟨c_q, t⟩∈q
4   do w_q ← Weight(q, t, c_q)
5       for each c∈B
6       do if Cr(c_q, c) > 0
7          then postings ← GetPostings(⟨c, t⟩)
8               for each posting ∈ postings
9               do x ← Cr(c_q, c) * w_q * weight(posting)
10                 score[docID(posting)] += x
11  for n ← 1 to N
12  do score[n] ← score[n]/normilizer[n]
13  return score
```

Рис. 10.9. Алгоритм для вычисления весов документов с помощью функции `SimNoMerge`

Рис. 10.10. Вычисление веса для запроса с одним структурным термином в функции `SimNoMerge`

Функция схожести запроса и документа на рис. 10.9 называется `SimNoMerge (близость без слияния)`, поскольку разные XML-контексты при взвешивании не объединяются. Альтернативная функция схожести `SimMerge` ослабляет условия соответствия запроса и документа по трем аспектам.

- Статистические показатели, необходимые для вычисления весов *weight*(*q*, *t*, *c*) и *weight*(*d*, *t*, *c*), вычисляются по *всем* контекстам, имеющим ненулевое сходство с контекстом *c* (в противоположность функции `SimNoMerge`, в которой эти

показатели вычисляются только для контекста *c*). Например, для вычисления документной частоты структурного термина `atl#"recognition"` мы также подсчитываем количество вхождений слова `recognition` в XML-контекстах `fm/atl`, `article//atl` и т.д.

- В формуле (10.2) объединяются все структурные термины документа, имеющие ненулевое сходство с контекстом структурного термина запроса. Например, контексты `/play/act/scene/title` и `/play/title` в документе при сопоставлении с запросом `/play/title#"Macbeth"` будут объединены.
- Функция схожести контекстов еще больше упрощается. Теперь контексты имеют ненулевое сходство во многих случаях, в которых функция *Cr*, определенная формулой (10.1), возвращает нуль.

Детали можно найти в работах, ссылки на которые даны в разделе 10.6.

Эти три изменения упрощают проблему вычисления статистик для редко встречающихся терминов (см. раздел 10.2) и повышают надежность функции при сравнении документов с плохо сформулированными структурированными запросами. Оценки функций `SimNoMerge` и `SimMerge`, приведенные в следующем разделе, показывают, что ослабленные условия согласованности функции `SimMerge` повышают эффективность XML-поиска.

? ***Упражнение 10.1.*** Проанализируйте подход, когда документная частота структурного термина равна количеству вхождений этого термина ниже конкретного родительского узла. Предположим следующее: структурный термин $<c, t>$ = `author#"Herbert"` встречается один раз как потомок узла *squib*; в коллекции насчитывается десять узлов *squib*; структурный термин $<c, t>$ встречается как дочерний узел *article* одну тысячу раз; в коллекции хранится один миллион узлов *article*. Тогда вес idf структурного узла $<c, t>$ равен $\log_2 10/1 \approx 3{,}3$, если он встречается как дочерний узел *squib* и $\log_2 1\,000\,000/1000 \approx 10{,}0$, если он встречается как дочерний узел *article*. 1) Объясните, почему этот способ взвешивания не пригоден для структурного термина $<c, t>$. Почему структурному термину $<c, t>$ нельзя приписывать вес, который в три раза больше в статьях (articles), чем в эпиграммах (squibs). 2) Предложите более точный способ вычисления показателя idf.

Упражнение 10.2. Перечислите все структурные термины в XML-документе, изображенном на рис. 10.8.

Упражнение 10.3. Сколько структурных терминов порождает документ, представленный на рис. 10.1?

10.4. Оценка XML-поиска

Основным мероприятием по исследованию XML-поиска является программа *INEX* (**IN**itiative for the **E**valuation of **X**ML retrieval), в рамках которой созданы эталонные коллекции, наборы запросов и оценки релевантности. На ежегодных конференциях INEX представляются и обсуждаются результаты исследований. Коллекция INEX 2002 состоит примерно из 12 тысяч статей из журналов, издаваемых Институтом инженеров по электротехнике и электронике (Institute of Electrical and Electronics Engineers — IEEE). Статистические показатели этой коллекции приведены в табл. 10.2, а часть схемы продемонст-

рирована на рис. 10.11. Коллекция журналов IEEE была пополнена в 2005 году. Начиная с 2006 года в проекте INEX в качестве тестовой стала использоваться намного более крупная коллекция статей Wikipedia на английском языке. Релевантность документов оценивается экспертами с помощью методологии, описанной в разделе 8.1, соответствующим образом модифицированной для структурированных документов.

Таблица 10.2. Статистические показатели коллекции INEX 2002

12 107	Количество документов
494 Мбайт	Размер
1995–2002 годы	Время публикации статей
1 532	Среднее количество XML-узлов в документе
6,9	Средняя глубина узла
30	Количество тем CAS (content-and-structure)
30	Количество тем CO (content-only)

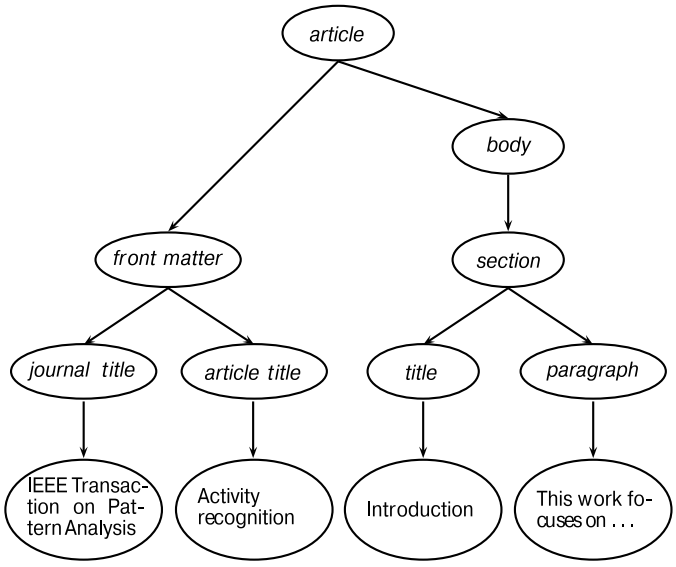


Рис. 10.11. Упрощенная схема документов в коллекции INEX

В проекте INEX существуют два типа информационных потребностей, или *тем* (topic): только по содержанию (CO — content-only) и по содержанию и структуре (CAS — content-and-structure). *Темы CO* (CO topis) — это обычные запросы из ключевых слов, как и в неструктурированном информационном поиске, а *темы CAS* (CAS topics) наряду с ключевыми словами содержат структурные ограничения. Один из примеров темы CO уже был показан на рис. 10.3. Ключевыми в этом примере являются слова `summer` и `holidays`, а структурные ограничения указывают, что ключевые слова появляются в разделе, который, в свою очередь, является частью статьи, а эта статья содержит атрибут года со значением 2001 или 2002.

Поскольку запросы CAS должны соответствовать определенным критериям и по содержанию, и по структуре, оценку релевантности получить сложнее, чем в неструктури-

рованном поиске. Эксперимент INEX 2002 определил, что покрытие компонентов и тематическая релевантность представляют собой ортогональные составляющие релевантности. *Покрытие компонентов* (component coverage) оценивает, правильно ли элемент был найден "со структурной точки зрения", т.е. не слишком ли низко и не слишком ли высоко на дереве. Различаются четыре варианта покрытия.

- Точное покрытие (exact coverage — E). Искомая информация совпадает с содержанием компонента, а сам компонент является осмысленной единицей информации.
- Слишком маленькое покрытие (too small — S). Искомая информация совпадает с содержанием компонента, но сам компонент не является осмысленной (самодостаточной) единицей информации.
- Слишком большое покрытие (too large — L). Искомая информация представлена в содержании компонента, но не является его основной темой.
- Нет покрытия (no coverage — N). Искомая информация не представлена в содержании компонента.

Тематическая релевантность (topical relevance) также имеет четыре уровня: высокорелевантный (3), вполне релевантный (2), слаборелевантный (1) и нерелевантный (0). Компоненты оцениваются по обеим составляющим, а оценки затем кодируются в виде цифр и букв. Код 2S означает вполне релевантный компонент, который имеет маленькое покрытие, а код 3E означает высокорелевантный компонент, имеющий точное покрытие. Теоретически возможны 16 комбинаций уровней покрытия и релевантности, но не все могут встречаться на практике. Например, нерелевантный компонент не может иметь точное покрытие, поэтому комбинация 3N невозможна.

Комбинации "релевантность–покрытие" дискретизируются следующим образом.

$$Q(rel,cov)=\begin{cases}1{,}00, & \text{если } (rel,cov)=3E,\\ 0{,}75, & \text{если } (rel,cov)\in\{2E,3L\},\\ 0{,}50, & \text{если } (rel,cov)\in\{1E,2L,2S\},\\ 0{,}25, & \text{если } (rel,cov)\in\{1S,1L\},\\ 0{,}00, & \text{если } (rel,cov)=0N.\end{cases}$$

Эта схема оценки учитывает тот факт, что бинарные оценки релевантности, являющиеся стандартными в неструктурированном поиске (раздел 8.5.1), непригодны для XML-поиска. Компонент 2S предоставляет неполную информацию и его трудно интерпретировать вне контекста, но он частично удовлетворяет запросу. Функция Q вместо бинарного вывода "релевантный/нерелевантный" позволяет использовать шкалу релевантности.

Количество релевантных компонентов в найденном множестве компонентов A можно вычислить следующим образом.

$$\#\,\text{релевантных компонентов} = \sum_{c\in A} Q\big(rel(c),cov(c)\big)$$

К этому модифицированному определению релевантных элементов в качестве приближения можно применить стандартные определения точности, полноты и F-меру из главы 8. Правда, при этом следует учесть некоторые тонкости, поскольку теперь мы используем

шкалу оценок, а не бинарные выводы. Библиографические ссылки, касающиеся дальнейшего обсуждения *направленного поиска* (focused retrieval), приведены в разделе 10.6.

У такого способа оценки релевантности есть один недостаток: он не учитывает перекрытие. Концепция маргинальной релевантности в контексте неструктурированного поиска рассматривалась в разделе 8.5.1. В контексте XML-поиска эта проблема становится еще острее, поскольку в списке результатов могут возникать вложенные узлы. Многие из недавних исследований в рамках INEX были посвящены разработке алгоритмов и количественных оценок, позволяющих находить неизбыточные списки результатов и правильно их оценивать. Библиографические ссылки приведены в разделе 10.6.

В табл. 10.3 приведены результаты двух прогонов INEX 2002 на основе модели векторного пространства, описанной в разделе 10.3. Более точным оказался метод `SimMerge`, учитывающий меньше структурных ограничений и в основном опирающийся на сравнение ключевых слов. Медианная средняя точность метода `SimMerge` (где медиана вычисляется по средней точности по темам) равна лишь 0,147. Качество XML-поиска часто ниже неструктурированного поиска, поскольку XML-поиск сложнее. Вместо того чтобы просто найти документ, мы должны найти его структурную часть, наиболее релевантную по отношению к запросу. Кроме того, качество XML-поиска (измеренная так, как описано в этом разделе) может быть ниже, чем неструктурированного, поскольку использование шкалы оценок снижает значение оценки. Рассмотрим систему, которая возвращает на первом месте списка результатов документ, оценка релевантности по градуированной шкале равна 0,6, а бинарная оценка — 1. Тогда при бинарных оценках интерполированная точность при нулевой полноте равна 1,0, а при использовании шкалы снижается до 0,6.

Таблица 10.3. Результаты эксперимента INEX 2002 для модели векторного пространства из раздела 10.3 для запросов CAS и функции дискретизации *Q*

Алгоритм	*Средняя точность*
SimNoMerge	0,242
SimMerge	0,271

Табл. 10.3 позволяет понять, каково качество XML-поиска, но не позволяет сравнить структурированный и неструктурированный поиск. В то же время табл. 10.4 ясно демонстрирует эффект учета структуры при поиске. Это результаты системы, основанной на языковой модели (см. главу 12). Система оценивалась по подмножеству тем CAS INEX 2003 и 2004. В качестве критерия для оценки выбрана точность на уровне k (глава 8). Функция дискретизации, использованная для получения этих оценок, ставит в соответствие высокорелевантным документам (около уровня 3E) единицу, а всем остальным элементам — нуль. Система поиска только по содержанию обрабатывает запросы и документы как неструктурированные “мешки слов”. Модель, полностью учитывающая структуру, присваивает элементам, удовлетворяющим структурным ограничениям, более высокие ранги, чем другим элементам. Например, для запроса, показанного на рис. 10.3, элемент, содержащий фразу `summer holidays` в узле *section*, получает более высокий ранг, чем элемент, содержащий фразу `summer holidays` в узле *abstract*.

Эта таблица демонстрирует, что структура помогает повысить точность в верхней части списка результатов. Точность на уровнях $k = 5$ и $k = 10$ повышается существенно, а на уровне $k = 30$ — практически нет. Эти результаты показывают выгоды структурного

поиска. Структурный поиск накладывает дополнительные ограничения на возвращаемые результаты, а документы, прошедшие структурный фильтр, скорее всего, более релевантные. Полнота при этом может снижаться, поскольку некоторые релевантные документы не проходят структурный фильтр, но для задач, ориентированных на точность, структурный поиск обладает несомненным преимуществом.

Таблица 10.4. Сравнение поиска только по содержанию и структурного поиска в проекте INEX 2003/2004

	По содержанию	*По структуре*	*Улучшение, %*
Точность на уровне 5[IS3]	0,2000	0,3265	63,3
Точность на уровне 10	0,1820	0,2531	39,1
Точность на уровне 20	0,1700	0,1796	5,6
Точность на уровне 30	0,1527	0,1531	0,3

10.5. Методы XML-поиска, ориентированные на текст и на данные

В разновидности структурного поиска, рассмотренной в этой главе, XML-структура играла роль конструкции, в рамках которой мы сравнивали текст запроса с текстом XML-документов. Этот вид систем оптимизирован для *языка XML, ориентированного на текст* (text-centric XML). Хотя и текст, и структура важны, мы присваиваем тексту более высокий приоритет. Для этого мы адаптируем неструктурированные методы поиска для обработки дополнительных структурных ограничений. Исходными предположениями нашего подхода является то, что поиск XML-документов характеризуется 1) длинными текстовыми полями (например, разделами документа), 2) неточным соответствием и 3) списком результатов, ранжированным по релевантности. Реляционные базы данных не способны хорошо решать такие задачи.

В противоположность этому *язык XML, ориентированный на данные* (data-centric XML), в основном кодирует числовые и нетекстовые атрибуты. Задавая запросы на языке XML, ориентированном на данные, в большинстве случаев пользователи требуют точного соответствия. В этом случае акцент делается на структурных аспектах XML-документов и запросов. Вот типичный пример.

Find employees whose salary is the same this month at it was 12 months ago.

Этот запрос не требует ранжирования; он, в первую очередь, относится к структуре. Возможно, для того чтобы удовлетворить информационную потребность пользователя, будет достаточно точного совпадения зарплат в двух периодах времени.

Методы, ориентированные на текст, подходят для данных, которые, по существу, представляют собой текстовые документы, размеченные с помощью языка XML. Дефакто этот метод стал стандартом для публикации текстовых баз данных, поскольку большинство текстовых документов имеют в какой-то форме структуру: параграфы, разделы, сноски и т.д. В частности, к таким документам относятся справочные руководства, номера журналов, собрание сочинений Шекспира и новостные сообщения.

Подходы, ориентированные на данные, обычно используются на коллекциях данных со сложной структурой и нетекстовым содержанием. Поисковые системы, ориентиро-

ванные на текст, с трудом справляются с данными о белках в биоинформатике или представлением карты города (вместе с названиями улиц и другими текстовыми элементами), которая лежит в основе навигационной системы.

Два других вида запросов, которые трудно обработать в модели поиска, ориентированной на текст, — запросы с объединением и запросы с ограничением на порядок следования терминов. Запрос, касающийся поиска сотрудников, у которых зарплата за последний год не изменилась, требует операции объединения. Запрос, налагающий ограничение на порядок следования терминов, выглядит так.

> Retrieve the chapter of the book *Introduction to algorithms* that follows the chapter *Binomial heaps*.

Этот запрос основан на порядке элементов в XML-структуре; в данном случае — на порядке следования глав под узлом *book*. Существуют мощные языки запросов для XML-поиска, которые могут обрабатывать ограничения на числовые атрибуты, объединения и порядок. Наиболее известным из них является язык XQuery, предложенный как стандарт консорциумом W3C (World Wide Web Consortium). Он может применяться во многих приложениях, в которых используется язык XML. Из-за его сложности очень трудно реализовать ранжирующую систему поиска на основе языка XQuery, обладающую качеством, которого пользователи ожидают от информационно-поисковой системы. В настоящее время эта задача является одной из наиболее активных областей исследований, связанных с XML-поиском.

Реляционные базы данных лучше приспособлены для обработки многих структурных ограничений, особенно объединений (но упорядочение в рамках баз данных также трудно учесть — кортежи отношений в реляционном исчислении не упорядочены). По этой причине большинство систем XML-поиска, ориентированных на данные, являются расширениями реляционных баз данных (см. библиографические ссылки в разделе 10.6). Если текстовые поля короткие, потребности пользователя можно удовлетворить с помощью точного сравнения и представление результатов поиска в неупорядоченном виде является приемлемым, то для XML-поиска можно использовать реляционные базы данных.

? ***Упражнение 10.4.*** Найдите коллекцию XML-документов разумного размера (или коллекцию, использующую другой язык разметки, например HTML) в вебе и загрузите ее. Для этого подойдет XML-редакция собрания сочинений Шекспира, выполненная Джоном Босаком (Jon Bosak), и многочисленные религиозные тексты, собранные на сайте `www.ibiblio.org/bosak/`, или первые 10 тысяч документов Википедии. Создайте три темы CAS, руководствуясь рис. 10.3, которые, по вашему мнению, будут лучше описывать информационную потребность, чем аналогичные темы CO. Объясните, почему система XML-поиска, способная учитывать структуру документов, может достичь более высоких результатов по сравнению с системой неструктурированного поиска.

Упражнение 10.5. Используя коллекцию и темы из упражнения 10.4, ответьте на следующие вопросы.

1. Существуют ли пары элементов e_1 и e_2, в которых элемент e_2 является подэлементом элемента e_1, причем оба элемента являются ответом на одну из тем?
2. Найдите вариант, в котором лучшим ответом на запрос является элемент e_1.
3. Найдите вариант, в котором лучшим ответом на запрос является элемент e_2.

Упражнение 10.6. Реализуйте алгоритмы SimMerge и SimNoMerge из раздела 10.3 и примените их к коллекции и темам из упражнения 10.4. Оцените результаты, сделав бинарный вывод о релевантности относительно первых пяти документов из трех списков результатов для каждого алгоритма. Какой из алгоритмов работает лучше?

Упражнение 10.7. Все ли элементы из упражнения 10.4 могут быть возвращены как результаты поиска или существуют элементы (как в примере `<b>definitely</b>`), которые следовало бы исключить?

Упражнение 10.8. Выше мы указали на противоречие между точностью результатов и размерностью векторного пространства. Приведите пример информационной потребности, которую можно правильно удовлетворить, если индексируются все лексикализированные поддеревья, и нельзя удовлетворить, если индексируются только структурные термины.

Упражнение 10.9. Какой будет зависимость размера индекса от размера текста, если проиндексировать все структурные термины?

Упражнение 10.10. Какой будет зависимость размера индекса от размера текста, если проиндексировать все лексикализированные поддеревья?

Упражнение 10.11. Приведите пример пары “запрос–документ”, для которой значение SimNoMerge(q, d) больше 1,0.

10.6. Библиография и рекомендации для дальнейшего чтения

Существует много хороших вводных курсов по языку XML, в частности книга Харольда и Минза (Harold and Means, 2004). Табл. 10.1 заимствована из учебника ван Рийсбергена (van Rijsbergen, 1979). В основу раздела 10.4 положен обзор INEX 2002, сделанный Гевертом и Казаи (Gövert and Kazai, 2003) и опубликованный в сборнике докладов конференции INEX (Fuhr et al., 2003*a*). Труды четырех последующих конференций INEX были опубликованы в сборниках под редакцией Фура и др. (2003*b*, 2005, 2006, 2007). Обзор современного состояния дел изложен в статье Фура и Лалмаса (Fuhr and Lalmas, 2007). Результаты, приведенные в табл. 10.4, взяты из работы Кампса и др. (Kamps et al., 2006). Дополнительные доказательства того, что XML-запросы повышают точность поиска по сравнению с неструктурированными запросами, опубликованы в работе Чу-Кэррола и др. (Chu-Carroll et al., 2006). В настоящее время в рамках проекта INEX вместо покрытия и полноты оцениваются родственные, но другие показатели: исчерпывающая полнота (exhaustivity) и специфичность (specificity) (Lalmas and Tombros, 2007). Тротман и др. (Trotman et al., 2006) связали задачи, исследованные в рамках проекта INEX, с реальными приложениями структурного поиска, например со структурным поиском книг на сайтах книжных электронных магазинов в Интернете.

Принцип поиска структурированных документов предложен Чиарамеллой и др. (Chiaramella et al., 1996). Рис. 10.5 заимствован из работы Фура и Гроссйохана (Guhr and Großjohann, 2004). Обзор схемы автоматического сопоставления, применимой к языку XML, можно найти в работе Рама и Бернштейн (Rahm and Bernstein, 2001). Метод XML-поиска, основанный на модели векторного пространства, описанный в разделе 10.3, по

существу, совпадает с методом, реализованным в системе JuruXML подразделением IBM в Хайфе (Mass et al., 2003; Carmel et al., 2003). Аналогичные подходы описаны в работах Шлидера и Мейсса (Schlieder and Meuss, 2002), а также Грабса и Шека (Grabs and Schek, 2002). Кармел и др. (Carmel et al., 2003) представили запросы в виде *XML-фрагментов*. Все деревья, представляющие XML-запросы в этой главе, являются XML-фрагментами, но XML-фрагменты допускают также применение операторов +, – и *phrase* к узлам, содержащим данные.

Мы выбрали модель векторного пространства для XML-поиска, поскольку она представляет собой простое и естественное расширение модели векторного пространства для неструктурированного поиска, описанной в главе 6. Однако для XML-поиска с таким же успехом можно применять и другие методы неструктурированного поиска. К этим методам относятся языковые модели (см. главу 12), описанные в работах Кампса и др. (Kamps et al., 2004), Листа и др. (List et al., 2005), Огилви и Каллана (Ogilvie end Callan, 2005), системы, сервер реляционных баз данных (Mihailović et al., 2005; Theobald et al., 2005, 2008), вероятностное взвешивание (Lu et al., 2007) и синтетические методы (Larson, 2005). В настоящее время нет общепризнанной точки зрения на то, какой из указанных подходов является лучшим для XML-поиска.

В первых работах по XML-поиску ранжирование релевантности осуществлялось по индивидуальным терминам запроса и документа, включая их структурные контексты. Как и в неструктурированном поиске, в более современных работах существует тенденция моделировать степень релевантности путем комбинирования разнородных характеристик запроса, документа и их совпадения. Функцию, осуществляющую такое комбинирование, можно настроить вручную (Arvola et al., 2005; Sigurbjörnsson et al., 2004) или с помощью машинного обучения (Vittaut and Gallinari, 2006).

Активной областью исследований по XML-поиску является *направленный поиск* (focused retrieval), описанный в работе Тротмана и др. (Trotman et. al., 2007). Этот подход ставит цель не возвращать вложенные элементы, имеющие один или несколько общих подэлементов (ср. обсуждение в разделе 10.2). Существуют свидетельства того, что пользователи не любят избыточности, порожденной вложенными элементами (Betsi et al., 2006). Для направленного поиска необходимо ввести оценки, штрафующие избыточность (Kazai and Lalmas, 2006; Lalmas et al., 2007). Тротман и Гева (Trotman and Geva, 2006) утверждают, что XML-поиск представляет собой разновидность *поиска отрывков* (passage retrieval), при котором поисковая система в ответ на запрос пользователя возвращает не документы, а короткие отрывки (Salton et al., 1993; Heasrt and Plaunt, 2991; Zobel et al., 1995; Hearst and Plaunt, 1993; Zobel et al., 1995; Hearst, 1997; Kaszkiel and Zobel, 1997). Несмотря на то что границы элементов в XML-документах облегчают идентификацию границ между отрывками, в большинстве случае наиболее релевантные отрывки не совпадают с XML-элементами.

В последние годы в рамках проекта INEX стандартом стал формат запросов NEXI, предложенный Тротманом и Сигурбьорнссоном (Trotman and Sigurbjörnsson, 2004). Рис. 10.3 заимствован из их статьи. О'Кифи и Тротман (O'Keefe and Trotman, 2004) привели свидетельства того, что пользователи не в состоянии надежно различать дочерние оси и оси наследников. Исходя из этого, в формате NEXI (и XML-фрагментах) разрешены только оси наследников. Эти структурные ограничения в недавних экспериментах INEX трактуются как "подсказки". Эксперты могут оценивать элемент как высокорелевантный, даже если он нарушает одно из структурных ограничений, указанных в запросе NEXI.

В качестве альтернативы языкам структурированных запросов, таким как NEXI, были предложены более развитые пользовательские интерфейсы для формулировки запросов (Tannier and Geva, 2005; van Zwol et al., 2006; Woodley and Geva, 2006).

Широкий обзор XML-поиска, охватывающий как базы данных, так и вопросы информационного поиска, опубликован Амер-Яхьей и Лалмасом (Amer-Yahia and Lalmas, 2006), а подробный список библиографических ссылок по этой теме можно найти в обзоре Амер-Яхьи и др. (Amer-Yahia et al., 2005). Хорошим введением в структурный поиск текстов с точки зрения баз данных является глава 6 книги Гроссмана и Фридера (Grossman and Frieder, 2004). Описание языка XQuery доступно по адресу `www.w3.org/TR/xquery/`, включая его расширение для полнотекстовых запросов (Amer-Yahia et al., 2006): `www.w3.org/TR/xquery-full-text/`. Комбинации реляционных баз данных и методов неструктурированного информационного поиска посвящены работы (Fuhr and Rцlleke, 1997; Navarro and Baeza-Yates, 1997; Cohen, 1988; Chaudhuri et al., 2006).

Глава 11

Вероятностная модель информационного поиска

Обсуждая обратную связь по релевантности в разделе 9.1.2, мы заметили, что, зная несколько релевантных и нерелевантных документов, можно непосредственно оценить вероятность того, что термин t появится в релевантном документе $P(t|R=1)$, и это может стать основой для создания классификатора, решающего, является ли документ релевантным. В этой главе вероятностный подход к информационному поиску разрабатывается более глубоко. Это позволяет создать другую формальную основу для модели поиска, что приводит к альтернативным методам определения весов терминов.

Сначала у пользователей есть информационные потребности, которые они затем переводят в форму запросов. Аналогично существуют документы, которые преобразовываются в представления документов (которые отличаются уже тем, как текст разбивается на лексемы, и, возможно, содержат намного меньше информации, как, например, при использовании некоординатного индекса). Основываясь на этих двух представлениях (запроса и документа), система пытается определить, насколько хорошо документы удовлетворяют информационным потребностям. В моделях булева поиска и векторного пространства сопоставление осуществляется в рамках формально определенного, но семантически неточного исчисления индексных терминов. Имея только запрос, система информационного поиска неточно понимает информационную потребность пользователя. Зная представления запроса и документов, система может лишь угадывать, является ли содержание документа релевантным данной информационной потребности. Для того чтобы принимать решения в условиях неопределенности, необходим математический аппарат теории вероятностей. В настоящей главе показано, как использовать эту теорию для оценки вероятности, что документ является релевантным по отношению к информационной потребности.

Существует несколько возможных вероятностных моделей поиска. В этой главе излагаются основы теории вероятностей и принцип вероятностного ранжирования (разделы 11.1 и 11.2), а также оригинальная бинарная модель независимости (раздел 11.3), являющаяся наиболее известной вероятностной моделью поиска. В заключение мы опишем усовершенствованные методы, использующие частоты терминов, включая схему взвешивания Okapi BM25, на практике доказавшую свою эффективность, и сетевые байесовские модели для информационного поиска (раздел 11.4). В главе 12 будет описан альтернативный вероятностный подход к информационному поиску, основанный на языковых моделях, который в последние годы разрабатывается с большим успехом.

11.1. Основы теории вероятностей

Надеемся, что читатель уже знаком с основами теории вероятностей, поэтому приведем краткий обзор основных положений; в конце главы указаны библиографические ссылки на источники по теории вероятностей для более глубокого изучения. Буква "*A*" обозначает событие (подмножество из пространства возможных исходов). Это подмножество можно также представить с помощью *случайной величины* (random variable) — функции, которая исходам ставит в соответствие действительные числа; в таком случае подмножество представляет собой область определения случайной величины *A*. Часто нам неизвестно с определенностью, произошло событие в реальном мире или нет. В таком случае следует оценить вероятность события $0 \le P(A) \le 1$. Наступление событий *A* и *B* одновременно описывается совместной вероятностью $P(A, B)$. Условная вероятность $P(A|B)$ выражает вероятность события *A*, если событие *B* уже наступило. Фундаментальная зависимость между совместной и условной вероятностями описывается *формулой умножения вероятностей* (chain rule).

$$P(A,\ B) = P(A \cap B) = P(A|B)P(B) = P(B|A)P(A) \tag{11.1}$$

Вероятность совместного события равна вероятности одного из событий, умноженной на вероятность другого события, при условии, что произошло первое событие.

Обозначив через $P(\bar{A})$ дополнение события *A*, мы получим аналогичную формулу.

$$P(\bar{A}, B) = P(B|\bar{A})P(\bar{A}) \tag{11.2}$$

В теории вероятностей есть также *правило разбиения* (partition rule), гласящее, что, если событие *B* можно разбить на исчерпывающее множество непересекающихся событий, то вероятность события *B* равна сумме вероятностей этих событий. Частный вариант этого правила записывается так.

$$P(B) = P(A, B) + P(\bar{A}, B) \tag{11.3}$$

Отсюда следует *правило Байеса* (Bayes' rule).

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} = \left[\frac{P(B|A)}{\sum_{X \in \{A, \bar{A}\}} P(B|X)P(X)}\right] P(A) \tag{11.4}$$

Эту формулу можно интерпретировать как способ уточнения вероятностей. Исходная оценка вероятности события *A* не использует никакой другой информации. Вероятность $P(A)$ называется *априорной вероятностью события A* (prior probability). Правило Байеса позволяет получить *апостериорную вероятность* $P(A|B)$ (posterior probability) при условии, что произошло событие *B* и мы знаем его *правдоподобие*[1] (likelihood) как в случае, если событие *A* произойдет, так и в случае, если событие *A* не произойдет.

В заключение отметим, что часто целесообразно говорить о *шансах* (odds) события.

$$O(A) = \frac{P(A)}{P(\bar{A})} = \frac{P(A)}{1 - P(A)} \tag{11.5}$$

---

[1] Термин *правдоподобие* (likelihood) — это просто синоним слова *вероятность*. Он означает вероятность события или того, что данные соответствуют модели. Этот термин обычно используется, когда данные фиксированы, а модель изменяется.

11.2. Принцип вероятностного ранжирования

11.2.1. Бинарные потери

Рассмотрим ранжированный поиск в коллекции документов (см. раздел 6.3), при котором пользователь отсылает запросы и получает в ответ упорядоченный список документов. Предположим также, оценки релевантности являются бинарными, как в главе 8. Для запроса q и документа d обозначим через $R_{d,q}$ случайную величину — индикатор релевантности документа d по отношению к запросу q. Эта величина равна единице, если документ релевантный, и нулю в противном случае. В ситуациях, не вызывающих недоразумений, мы будем часто писать R, а не $R_{d,q}$.

В рамках вероятностной модели естественно ранжировать результаты поиска по оцененным вероятностям их релевантности информационной потребности: $P(R = 1|d, q)$. Такой подход лежит в основе *принципа вероятностного ранжирования* (Probability Ranking Principle — PRP), предложенного ван Рийсбергеном (van Rijsbergen, 1979).

> Если в ответ на каждый запрос поисковая система ранжирует документы в коллекции в порядке убывания вероятности их релевантности для пользователя, отославшего запрос, где вероятности оценены как можно более точно на основе доступных данных, то общее качество системы является наилучшим на этих данных.

Простейший вариант принципа PRP игнорирует стоимость поиска и прочие соображения о полезности, в соответствии с которыми действия и ошибки системы могут иметь разные веса. Вы просто теряете балл, если система возвращает нерелевантный документ или не находит релевантный документ (такая бинарная модель оценки *правильности* (accuracy) называется *моделью бинарных потерь* (1/0 loss)). Цель системы — вернуть в качестве первых k документов наилучшие результаты для любого значения k, которое выбирает пользователь. В таком случае в соответствии с принципом PRP достаточно расположить все документы в порядке убывания величин $P(R = 1|d, q)$. Если необходимо вернуть неупорядоченное множество найденных результатов, то в соответствии с *байесовским правилом оптимального решения* (Bayes optimal decision rule), минимизирующим риск потерь, достаточно вернуть документы, которые скорее релевантные, чем нерелевантные.

$$d \text{ — релевантный} \Leftrightarrow P(R = 1|d, q) > P(R = 0|d, q) \tag{11.6}$$

Теорема 11.1. *Принцип PRP является оптимальным в том смысле, что он минимизирует ожидаемые потери* (*также известные как байесовский риск*) *в рамках модели бинарных потерь*.

Доказательство теоремы изложено в работе Рипли (Ripley, 1996). Однако она требует, чтобы все вероятности были определены точно. На практике это условие никогда не выполняется. Тем не менее принцип PRP представляет собой очень полезную основу для разработки моделей информационного поиска.

11.2.2. Принцип вероятностного ранжирования с учетом стоимости поиска

Допустим теперь, что модель учитывает стоимость поиска. Пусть C_1 — стоимость поиска релевантного документа, а C_0 — стоимость поиска нерелевантного документа.

Тогда принцип PRP утверждает, что, если для конкретного документа d и всех еще не найденных документов d' выполняется неравенство

$$C_0 \cdot P(R = 0|d) - C_1 \cdot P(R = 1|d) \leq C_0 \cdot P(R = 0|d') - C_1 \cdot P(R = 1|d'), \qquad (11.7)$$

то следующим нужно вернуть документ d. Такая модель дает формальный аппарат, с помощью которого можно моделировать разную стоимость ложноположительных и ложноотрицательных ответов и даже учитывать производительность системы на этапе моделирования, а не на этапе тестирования, как в разделе 8.6. Однако в дальнейшем мы не будем рассматривать вопросы потерь и полезности в этой главе.

11.3. Бинарная модель независимости

Бинарная модель независимости (Binary Independence Model — BIM), представленная в этом разделе, обычно используется совместно с принципом PRP. Она основана на некоторых достаточно простых предположениях, позволяющих на практике оценить вероятность $P(R|d, q)$. Здесь слово “бинарная” эквивалентно слову “булева”. Документы и запросы представляются в виде *бинарных векторов инцидентности терминов* (binary term incidence vectors). Иначе говоря, документ d представляется в виде вектора $\vec{x} = (x_1, x_2, ..., x_M)$, где $x_t = 1$, если термин t содержится в документе d, и $x_t = 0$, если термин t не содержится в документе d. В этом случае многие документы могут иметь одинаковые векторные представления. Аналогично запрос q представляется в виде вектора инцидентности $\vec{q}$ (различие между запросом q и вектором $\vec{q}$ менее важно, поскольку запрос q обычно является совокупностью слов). “Независимость” означает, что в соответствии с предположениями модели термины встречаются в документе независимо друг от друга. Модель не учитывает ассоциаций между терминами. Это предположение далеко от истины, тем не менее часто на практике оно дает удовлетворительные результаты; это же “наивное” предположение лежит в основе наивных байесовских моделей (раздел 13.4). Действительно, бинарная модель независимости точно совпадает с многомерной наивной байесовской моделью Бернулли (раздел 13.3). В некотором смысле это предположение эквивалентно предположению, лежащему в основе модели векторного пространства, в которой каждый термин представляет собой размерность, ортогональную всем другим терминам.

Сначала опишем модель, в которой предполагается, что пользователь удовлетворяет информационную потребность за один шаг. Как указывалось в главе 9, просмотрев список результатов, пользователь может уточнить свою информационную потребность. К счастью, модель BIM можно модифицировать так, чтобы создать основу для обратной связи по релевантности (раздел 11.3.4).

Для уточнения вероятностной стратегии поиска необходимо оценить, как термины в документе влияют на его релевантность; в частности, мы хотим знать, как частота термина, документная частота, длина документа и другие статистические показатели, которые можно вычислить, влияют на релевантность документов и как их правильно использовать для оценки вероятности релевантности документа. После этого мы расположим документы в порядке уменьшения оцененной вероятности их релевантности.

Предположим, что релевантность каждого документа не зависит от релевантности других документов. Как указывалось в разделе 8.5.1, это неверно; данное предположение особенно опасно на практике, если оно позволяет системе возвращать в качестве результатов дубликаты или очень похожие друг на друга документы. В рамках модели BIM мы

вычисляем вероятность $P(R|d, q)$ того, что документ является релевантным, через векторы инцидентности терминов $P(R|\vec{x},\vec{q})$. В таком случае, используя правило Байеса, мы получаем следующие формулы.

$$P(R=1|\vec{x},\vec{q}) = \frac{P(\vec{x}|R=1,\vec{q})P(R=1|\vec{q})}{P(\vec{x}|\vec{q})},$$
$$P(R=0|\vec{x},\vec{q}) = \frac{P(\vec{x}|R=0,\vec{q})P(R=0|\vec{q})}{P(\vec{x}|\vec{q})}. \quad (11.8)$$

Здесь $P(\vec{x}|R=1,\vec{q})$ и $P(\vec{x}|R=0,\vec{q})$ — вероятности того, что если был найден релевантный или нерелевантный документ соответственно, то его представление имеет вид $\vec{x}$. Эти вероятности следует считать функциями, определенными на пространстве возможных документов из предметной области. Как вычислить эти вероятности? Точные вероятности всегда неизвестны, и поэтому их приходится заменять оценками, полученными на основе реальной коллекции документов. Величины $P(R=1|\vec{q})$ и $P(R=0|\vec{q})$ — это априорные вероятности найти релевантный или нерелевантный документ по запросу $\vec{q}$ соответственно. Эти величины можно оценить, если известна доля релевантных документов в коллекции. Поскольку документ является либо релевантным, либо нерелевантным по отношению к запросу, должно выполняться следующее равенство.

$$P(R=1|\vec{x},\vec{q}) + P(R=0|\vec{x},\vec{q}) = 1 \quad (11.9)$$

11.3.1. Вывод функции ранжирования для терминов запроса

Для заданного запроса q мы хотим расположить найденные документы в порядке убывания величин $P(R = 1|d, q)$. В рамках модели BIM это сводится к упорядочению по величинам $P(R=1|\vec{x},\vec{q})$. Поскольку мы заинтересованы лишь в ранжировании документов, вместо непосредственной оценки этой вероятности мы будем работать с другими величинами, которые легче вычислить и которые порождают тот же порядок следования документов. В частности, мы можем ранжировать документы по шансам релевантности (поскольку шансы релевантности являются монотонной функцией, зависящей от вероятности релевантности). Это упрощает вычисления, поскольку можно игнорировать общий знаменатель в равенстве (11.8).

$$O(R|\vec{x},\vec{q}) = \frac{P(R=1|\vec{x},\vec{q})}{P(R=0|\vec{x},\vec{q})} = \frac{\dfrac{P(R=1|\vec{q})P(\vec{x}|R=1,\vec{q})}{P(\vec{x}|\vec{q})}}{\dfrac{P(R=0|\vec{q})P(\vec{x}|R=0,\vec{q})}{P(\vec{x}|\vec{q})}} = \frac{P(R=1|\vec{q})}{P(R=0|\vec{q})} \cdot \frac{P(\vec{x}|R=1,\vec{q})}{P(\vec{x}|R=0,\vec{q})} \quad (11.10)$$

Первый множитель последнего члена равенства (11.10) для заданного запроса является постоянным. Поскольку мы только ранжируем документы, оценивать его необязательно. Однако для вычисления второго множителя последнего члена равенства (11.10) необходима оценка, которую довольно трудно получить. Как правильно оценить вероятность появления целого вектора инцидентности терминов? Именно в этот момент нам на помощь приходит *наивное байесовское предположение об условной независимости* (naive Bayes conditional independence assumption), утверждающее, что присутствие или

отсутствие слова в документе не зависит от присутствия или отсутствия любого другого слова (при заданном запросе).

$$\frac{P(\vec{x}|R=1,\vec{q})}{P(\vec{x}|R=0,\vec{q})}=\prod_{t=1}^{M}\frac{P(x_t|R=1,\vec{q})}{P(x_t|R=0,\vec{q})} \tag{11.11}$$

Следовательно,

$$O(R|\vec{x},\vec{q})=O(R|\vec{q})\prod_{t=1}^{M}\frac{P(x_t|R=1,\vec{q})}{P(x_t|R=0,\vec{q})}. \tag{11.12}$$

Поскольку каждая величина x_t равна либо нулю, либо единице, выражение (11.12) можно переписать иначе.

$$O(R|\vec{x},\vec{q})=O(R|\vec{q})\prod_{t:x_t=1}^{M}\frac{P(x_t=1|R=1,\vec{q})}{P(x_t=1|R=0,\vec{q})}\prod_{t:x_t=0}^{M}\frac{P(x_t=0|R=1,\vec{q})}{P(x_t=0|R=0,\vec{q})} \tag{11.13}$$

Впредь будем обозначать через $p_t=P(x_t=1|R=1,\vec{q})$ вероятность появления термина в документе, релевантном запросу, а через $u_t=P(x_t=1|R=0,\vec{q})$ — вероятность появления термина в документе, нерелевантном запросу. Эти величины можно проиллюстрировать с помощью следующей таблицы сопряженности признаков, где сумма по столбцам равна единице.

	Документ	Релевантный (R=1)	Нерелевантный (R = 0)
Термин есть	x_t = 1	p_t	u_t
Термина нет	x_t = 0	$1-p_t$	$1-u_t$

(11.14)

Предположим для простоты, что термин, не встречающийся в запросе, с одинаковой вероятностью может встретиться в релевантных и нерелевантных документах, т.е. если $q_t = 0$, то $p_t = u_t$. (Это предположение можно изменить, например, при использовании обратной связи по релевантности, как показано в разделе 11.3.4.) Теперь достаточно учесть лишь термины, появляющиеся в запросе.

$$O(R|\vec{x},\vec{q})=O(R|\vec{q})\prod_{t:x_t=q_t=1}^{M}\frac{p_t}{u_t}\prod_{t:x_t=0,\,q_t=1}^{M}\frac{1-p_t}{1-u_t} \tag{11.15}$$

Первое произведение в правой части вычисляется по терминам запроса, найденным в документе, а второе — по не найденным.

Это выражение можно преобразовать, включив термины запроса, найденные в документе, во второе произведение, одновременно учтя их в первом произведении, чтобы результат остался неизменным.

$$O(R|\vec{x},\vec{q})=O(R|\vec{q})\prod_{t:x_t=q_t=1}^{M}\frac{p_t(1-u_t)}{u_t(1-p_t)}\prod_{t:q_t=1}^{M}\frac{1-p_t}{1-u_t} \tag{11.16}$$

Первое произведение в правой части по-прежнему вычисляется по терминам запроса, найденным в документе, а второе произведение теперь вычисляется по всем терминам запроса. Это значит, что второе произведение при заданном запросе является постоянным, так же как и шансы $O(R|\vec{q})$. Следовательно, единственной величиной, которую необходимо оценить для ранжирования документов по релевантности запросу, является первое произведение. Поскольку логарифм является монотонной функцией, то, прологарифмировав

это выражение, мы получим эквивалентное ранжирование. Итак, будем использовать для ранжирования *значение документа по запросу* (Retrieval Status Value — RSV).

$$RSV_d = \log \prod_{t:x_t=q_t=1}^{M} \frac{p_t(1-u_t)}{u_t(1-p_t)} = \sum_{t:x_t=q_t=1} \log \frac{p_t(1-u_t)}{u_t(1-p_t)} \tag{11.17}$$

Приступим к вычислению этой величины. Введем вспомогательную величину c_t.

$$c_t = \log \frac{p_t(1-u_t)}{u_t(1-p_t)} = \log \frac{p_t}{1-p_t} + \log \frac{1-u_t}{u_t} \tag{11.18}$$

Величина c_t представляет собой логарифм отношения шансов для терминов запроса. Шансы термина запроса появиться в релевантном документе равны $p_t/(1-p_t)$, а шансы термина запроса появиться в нерелевантном документе равны $u_t/(1-u_t)$. *Отношение шансов* (odds ratio) выражает относительную величину шансов двух событий. Логарифм этой величины равен нулю, если шансы термина запроса появиться в релевантных и нерелевантных документах одинаковы, и больше нуля, если шансы термина запроса появиться в релевантных документах больше шансов появиться в нерелевантных документах. Величины c_t играют роль весов терминов в модели, а вес документа относительно запроса равен $RCV_d = \sum_{x_t=q_t=1} c_t$. По существу, мы просто суммируем их по всем терминам запроса, появляющимся в документе, как в модели векторного пространства, описанной в разделе 7.1. Теперь вернемся к оценке величин c_t для конкретных коллекции и запроса.

11.3.2. Оценки вероятностей: теория

Как выглядят величины c_t для каждого термина t по всей коллекции? Таблица сопряженности признаков (11.19) содержит статистические показатели документов в коллекции, где df_t — это количество документов, содержащих термин t.

	Документ	Релевантный (R=1)	Нерелевантный (R = 0)	Всего
Термин есть	$x_t = 1$	s	$df_t - s$	df_t
Термина нет	$x_t = 0$	$S - s$	$(N - df_t) - (S - s)$	$N - df_t$
Всего		S	$N - S$	N

(11.19)

Используя эти величины, выразим вероятности появления термина t в релевантных и нерелевантных документах как $p_t = s/S$ и $u_t = (df_t - s)/(N - S)$.

$$c_t = K(N, df_t, S, s) = \log \frac{s/(S-s)}{(df_t - s)/((N - df_t) - (S - s))} \tag{11.20}$$

Для того чтобы избежать нулей в знаменателе (например, когда каждый релевантный документ содержит некий термин или ни один релевантный документ не содержит некий термин), к каждой величине, записанной в средних четырех ячейках факторной таблицы (11.19), обычно добавляют 1/2, а затем соответственно корректируют маргинальные частоты (в столбце “Всего”), так что в правой нижней ячейке сумма равна $N + 2$.

$$\hat{c}_t = K(N, df_t, S, s) = \log \frac{\left(s + \frac{1}{2}\right) \Big/ \left(S - s + \frac{1}{2}\right)}{\left(df_t - s + \frac{1}{2}\right) \Big/ \left(N - df_t - S + s + \frac{1}{2}\right)} \tag{11.21}$$

Добавление 1/2 представляет собой вид сглаживания. Для испытаний с категориальными исходами (например, когда термин либо присутствует, либо отсутствует) оценить вероятность по данным можно, подсчитав, сколько раз произошло событие, и разделив это число на общее количество испытаний. Полученное число называется *относительной частотой* события (relative frequency). Оценка вероятности на основе относительной частоты события представляет собой *оценку максимального правдоподобия* (Maximum Likelihood Estimate — MLE), поскольку именно это значение делает наблюдаемые значения максимально правдоподобными. Однако непосредственное применение оценки MLE приводит к завышению вероятности часто происходящих событий и недооценке вероятности редких событий, что к тому же приводит к неработающим моделям за счет нулевых множителей. Уменьшение вероятности часто происходящих событий и повышение вероятности редких событий называется *сглаживанием* (smoothing). Простое сглаживание можно осуществить, добавив число α к каждой частоте. Эти *псевдочастоты* в байесовском подходе соответствуют использованию равномерного распределения термина по словарю в качестве *априорного* (см. формулу (11.4)). Сначала мы предполагаем равномерное распределение по событиям, где величина α обозначает силу нашей уверенности в равномерности, а затем уточняем вероятность на основе наблюдаемых событий. Поскольку наша уверенность в равномерности распределения слаба, положим $\alpha = 1/2$. Такая оценка называется *максимальной апостериорной оценкой* (Maximum a Posteriori — MAP). В этом случае, следуя равенству (11.4), наиболее правдоподобное значение вероятности выбирается на основе априорного распределения и наблюдаемых событий. Методы сглаживания частот для построения вероятностных моделей описываются в разделе 12.2.2, а пока мы будем использовать простой метод, который сводится к добавлению 1/2 к каждой наблюдаемой частоте.

11.3.3. Оценки вероятностей: практика

В предположении, что доля релевантных документов в коллекции очень мала, целесообразно аппроксимировать статистические показатели для нерелевантных документов на основе частот, подсчитанных по всей коллекции. В этом случае величина u_t (вероятность появления термина в нерелевантных документах по отношению к запросу) равна df_t/N и

$$\log[(1-u_t)/u_t] = \log[(N-\mathrm{df}_t)/\mathrm{df}_t] \approx \log N/\mathrm{df}_t. \qquad (11.22)$$

Эту формулу можно рассматривать как теоретическое обоснование наиболее распространенного способа вычисления величины idf, описанного в разделе 6.2.1.

Метод аппроксимации, основанный на равенстве (11.22), трудно распространить на релевантные документы. Величину p_t можно оценить разными способами.

1. Можно использовать частоту появления термина в известных релевантных документах (если такие документы есть). На этом основаны вероятностные подходы к взвешиванию с помощью обратной связи по релевантности, которые будут рассмотрены в следующем подразделе.
2. Крофт и Харпер (Croft and Harper, 1979) предложили использовать константу в своей модели комбинационного согласования (combination match model). Например, можно предположить, что p_t — это константа для всех терминов x_t и $p_t = 0{,}5$. Это значит, что все термины имеют одинаковые шансы появиться в релевантных документах и множители p_t и $1-p_t$ в выражении для вычисления RSV сокращаются. Такая оценка является слабой, но не противоречит ожиданиям обнаружить ис-

комые термины во многих, но не во всех релевантных документах. Комбинируя этот метод с предыдущей аппроксимацией величины u_t, мы получаем ранжирование документов на основе суммы весов idf терминов запроса. Для коротких документов (заголовки или аннотации) в ситуации, когда итеративный поиск нежелателен, использование такого взвешивания терминов может быть достаточным, хотя во многих других условиях можно предложить лучшие решения.

3. Грейф (Greiff, 1998) утверждает, что модель Крофта и Харпера слабо обоснована с теоретической точки зрения и не находит подтверждения на практике: как и следовало ожидать, оказалось, что при росте df_t величина p_t возрастает. Исходя из этого, разумно предложить использовать оценку $p_t = 1/3+(2/3)\mathrm{df}_t/N$.

Итеративные методы оценки, использующие комбинацию изложенных выше идей, обсуждаются в следующем подразделе.

11.3.4. Вероятностные подходы к обратной связи по релевантности

В ходе итеративного процесса более точную оценку величины p_t можно получить с помощью обратной связи по (псевдо)релевантности (RF). Вероятностный подход к обратной связи по релевантности работает следующим образом.

1. Угадываем начальные вероятности p_t и u_t. Для этого можно использовать оценки из предыдущего раздела. Например, можно предположить, что величина p_t является постоянной для всех терминов x_t в запросе, в частности можно положить $p_t = 1/2$.
2. На основе текущих оценок вероятностей p_t и u_t формируем множество наилучших документов $R = \{d: R_{d,q} = 1\}$. Применяем эту модель для поиска вероятных релевантных документов и представляем их пользователю.
3. Взаимодействуя с пользователем, уточняем модель для определения множества R. Для этого собираем оценки релевантности, сделанные пользователем, для документов из некоторого подмножества документов V. Основываясь на этих оценках релевантности, разделяем множество V на подмножество $VR = \{d \in V, R_{d,q} = 1\} \subset R$ и подмножество $VNR = \{d \in V, R_{d,q} = 0\}$, которое не пересекается с R.
4. Оцениваем вероятности p_t и u_t заново на основе известных релевантных и нерелевантных документов. Если множества VR и VNR достаточно велики, то эти величины можно оценить непосредственно как оценки максимального правдоподобия.

$$p_t = |VR_t|/|VR| \tag{11.23}$$

Здесь VR_t — подмножество документов из множества VR, содержащих термин t. На практике эти оценки приходится сглаживать. Для этого к величине $|VR_t|$ и количеству релевантных документов, не содержащих термин t, прибавим 1/2.

$$p_t = \frac{|VR_t| + \frac{1}{2}}{|VR| + \frac{1}{2}} \tag{11.24}$$

Однако множество документов, оцененных пользователем (V), обычно очень мало, поэтому полученный результат довольно ненадежен (содержит шум), даже если оценки были сглажены. По этой причине в процессе байесовского уточнения часто лучше комбинировать новую информацию с исходной догадкой. В этом случае мы получим такую формулу.

$$p_t^{(k+1)} = \frac{|VR_t| + \kappa p_t^{(k)}}{|VR| + \kappa} \tag{11.25}$$

Здесь $p_t^{(k)}$ — это k-я оценка вероятности p_t в процессе итеративного уточнения. Она используется на следующей итерации как байесовская априорная вероятность с весом κ. Для того чтобы понять связь этого равенства с равенством (11.4), необходимо знать теорию вероятностей немного глубже (в частности, необходимо использовать априорное бета-распределение, сопряженное со случайной бернуллиевской величиной X_t). Однако интерпретировать полученное равенство несложно: вместо равномерного распределения псевдочастот мы распределяем κ псевдочастот в соответствии с предыдущими оценками. Если нет других данных (для простоты предположим, что пользователь указал около пяти релевантных или нерелевантных документов), можно положить $\kappa = 5$. Иначе говоря, априорная оценка получает большой вес, поэтому при очень небольшом количестве документов ее изменения будут незначительными.

5. Повторяем процесс, начиная с этапа 2, генерируя последовательные приближения множества R и вероятности p_t, пока пользователь не будет полностью удовлетворен.

На основе этого алгоритма легко можно создать его вариант с обратной связью по псевдорелевантности, в котором просто предполагается, что $VR = V$.

1. Пусть вероятности p_t и u_t определены, как описано выше.
2. Выдвигаем догадку о размере множества релевантных документов. Если существует неуверенность, будем использовать консервативную (очень низкую) оценку. В этом случае придется использовать множество V документов с наивысшими рангами фиксированного размера.
3. Уточним оценки вероятностей p_t и u_t. Для уточнения оценки вероятности p_t будем применять формулы (11.23) и (11.25), только вместо множества VR теперь будем использовать множество V. Если множество V_t — подмножество документов в множестве V, содержащих термин t, а для сглаживания использовать 1/2, то получим следующий результат.

$$p_t = \frac{|V_t| + \frac{1}{2}}{|V| + 1} \tag{11.26}$$

Если предположить, что ненайденные документы являются нерелевантными, то вероятность u_t можно уточнить следующим образом.

$$u_t = \frac{df_t - |V_t| + \frac{1}{2}}{N - |V| + 1} \tag{11.27}$$

4. Возвращаемся на этап 2, пока ранжирование найденных результатов не изменяется.

Веса c_t, используемые в формуле для вычисления величины RSV, выглядят очень похожими на значения tf–idf. Например, из равенств (11.18), (11.22) и (11.26) следует такая формула.

$$c_t = \log\left[\frac{p_t}{1-p_t}\cdot\frac{1-u_t}{u_t}\right] \approx \log\left[\frac{|V_t|+\frac{1}{2}}{|V|-|V_t|+1}\cdot\frac{N}{df_t}\right] \tag{11.28}$$

Однако теперь величина $p_t/(1–p_t)$ представляет собой оценку доли релевантных документов, в которых появляется термин t, а не частоту термина. Более того, если применить правило логарифмирования, то можно получить следующую формулу.

$$c_t = \log\frac{|V_t|+\frac{1}{2}}{|V|-|V_t|+1} + \log\frac{N}{df_t} \tag{11.29}$$

Как видим, теперь мы *складываем* два логарифма, а не перемножаем их.

? ***Упражнение 11.1.*** Проанализируйте вывод равенства (11.20) из равенств (11.18) и (11.19).

Упражнение 11.2. В чем заключается разница между схемами взвешивания tf–idf и вероятностной моделью поиска BIM в ситуации, когда информация о релевантности документа недоступна?

***Упражнение 11.3* [**].** Пусть X_t (случайная величина) — индикатор того, что термин t встречается в документе. Допустим, что в коллекции есть $|R|$ релевантных документов, причем термин встречается в s из этих документов. В качестве данных наблюдений возьмите величины X_t для каждого документа из множества R. Покажите, что оценка максимального правдоподобия (MLE) для параметра p_t равна $P\left(X_t = 1 \middle| R = 1, \vec{q}\right)$, т.е. значение p_t, обеспечивающее максимальное правдоподобие наблюдаемых данных, равно $p_t = s/|R|$.

Упражнение 11.4. Опишите разницу между обратной связью по релевантности в модели векторного пространства и вероятностной обратной связью по релевантности.

11.4. Вероятностные модели и некоторые модификации

11.4.1. Роль и место вероятностных моделей

Вероятностные модели — одни из старейших моделей в теории информационного поиска. Уже в 1970-х годах они рассматривались как средство теоретического обоснования методов информационного поиска. По мере возрождения вероятностных методов в компьютерной лингвистике в 1990-х годах эти надежды вернулись, и вероятностные методы снова стали одной из наиболее интенсивно изучаемых тем в теории информационного поиска. Традиционно вероятностные методы информационного поиска отличались изящными идеями, но никогда не выигрывали в производительности. Для того чтобы получить разумные приближения требуемых вероятностей в рамках этих моделей, необходимо сделать несколько предположений. Перечислим предположения, на которых основана модель BIM.

1. Документы, запросы и выводы о релевантности имеют бинарное представление.
2. Термины не зависят друг от друга.

3. Термины, не содержащиеся в запросе, не влияют на результаты поиска.
4. Оценки релевантности документов являются независимыми друг от друга.

Вероятно, слишком строгие предположения не позволяют вероятностным моделям достичь высокой производительности. По нашему мнению, основной проблемой является то, что вероятностные модели либо требуют частичной информации о релевантности, либо, в отсутствие такой информации, позволяют вывести лишь очевидно более слабые модели взвешивания терминов.

Положение дел стало изменяться в 1990-х годах, когда была продемонстрирована очень высокая производительность схемы взвешивания BM25, которая будет рассмотрена чуть ниже. С тех пор эта схема стала применяться для взвешивания терминов многими исследовательскими группами. Разница между векторными и вероятностными информационно-поисковыми системами невелика; и в том, и в другом случаях система информационного поиска создается точно так же, как описано в главе 7. Просто для вероятностной информационно-поисковой системы запросы оцениваются не с помощью косинусной меры сходства и схемы взвешивания tf–idf, как в векторном пространстве, а по немного другим формулам, вытекающим из теории вероятностей. По существу, в некоторых случаях разработчики просто модифицировали существующие информационно-поисковые системы, основанные на модели векторного пространства, просто позаимствовав вероятностные схемы взвешивания терминов. В этом разделе мы кратко рассмотрим три расширения традиционных вероятностных моделей, а в следующей главе изучим альтернативный вероятностный подход к информационному поиску — языковые модели.

11.4.2. Древовидные зависимости между терминами

Некоторые предположения модели BIM можно устранить. Например, можно избавиться от предположения о независимости терминов. Это предположение очень далеко от действительности. Например, оно нарушается для терминов Hong и Kong, которые сильно зависят друг от друга. Зависимости могут встречаться и в более сложных конфигурациях, например в множестве терминов New, York, England, City, Stock, Exchange и University. Ван Рийсберген (van Rijsbergen, 1979) предложил простую и надежную модель, допускающую древовидную зависимость между терминами (рис. 11.1). В этой модели

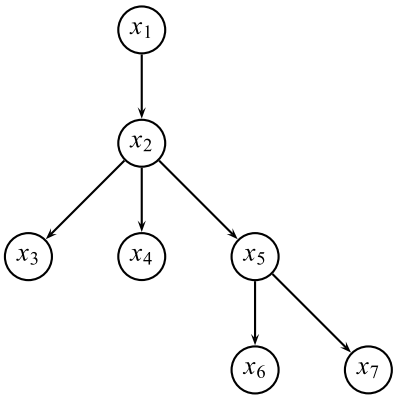


Рис. 11.1. Дерево зависимостей между терминами. В этом графическом представлении термин x_i непосредственно зависит от термина x_k, если существует стрелка $x_k \rightarrow x_i$

каждый термин может непосредственно зависеть только от одного другого термина, что порождает древовидную структуру зависимостей. Когда эта модель была изобретена в 1970-х годах, ее практическому применению помешали проблемы с получением оценок, но в 1996 году эта идея возродилась в наивной древовидной байесовской модели Фридмана и Голдшмидта (Friedman and Goldszmidt, 1996), которая с определенным успехом использовалась на разных наборах данных для машинного обучения.

11.4.3. Okapi BM25: небинарная модель

Модель BIM изначально предназначалась для коротких записей в библиотечных каталогах и аннотаций небольшой длины. В этом контексте она вполне оправдывала себя, но очевидно, что в современных полнотекстовых коллекциях поисковая модель должна учитывать частоту термина и длину документа, как указано в главе 6. *Схема взвешивания BM25* (BM25 weighting scheme), которая часто называется *схемой Okapi* (*Okapi weighting*) по имени системы, в которой она была впервые внедрена, была разработана как способ построения вероятностной модели, чувствительной к частоте термина и длине документа, но не использующей слишком много дополнительных параметров (Sparck Jones et al., 2000). Мы не будем излагать полную теорию, лежащую в основе этой модели, а приведем ряд формул, которые в настоящее время стали стандартными формулами ранжирования документов. Простейшая схема ранжирования документа d — это схема взвешивания, основанная на обратной документной частоте терминов запроса, как в равенстве (11.22).

$$RSV_d = \sum_{t \in q} \log \frac{N}{df_t} \tag{11.30}$$

Иногда используется альтернативная версия idf. Если начать с формулы (11.21), то в отсутствие обратной связи по релевантности делаем оценку $S = s = 0$. В этом случае альтернативная формула для idf выглядит так.

$$RSV_d = \sum_{t \in q} \log \frac{N - df_t + \frac{1}{2}}{df_t + \frac{1}{2}} \tag{11.31}$$

Этот вариант ведет себя несколько странно. Если термин появляется более чем в половине документов коллекции, то вес термина будет отрицательным, что, по идее, нежелательно. Однако при использовании списка стоп-слов этого обычно не происходит, и можно считать, что минимальное значение каждого слагаемого равно нулю.

Мы улучшим равенство (11.30), если учтем частоту каждого термина и длину документа.

$$RSV_d = \sum_{t \in q} \log\left[\frac{N}{df_t}\right] \cdot \frac{(k_1+1)tf_{td}}{k_1\left((1-b)+b\times\left(L_d/L_{ave}\right)\right)+tf_{td}} \tag{11.32}$$

Здесь tf_{td} — частота термина t в документе d, а L_d и L_{ave} — длина документа d и средняя длина документа во всей коллекции. Переменная k_1 — это положительный параметр настройки, с помощью которого производится калибровка частоты термина. Если $k_1 = 0$, то модель становится бинарной (частота термина не учитывается), а если параметр k_1 принимает большие значения, то это эквивалентно прямому подсчету частоты термина (raw term frequency). Переменная b — еще один параметр настройки ($0 \le b \le 1$), определяющий нормировку по длине документа: $b = 1$ соответствует полноценному масштаби-

рованию веса термина с помощью длины документа, а $b = 0$ не предусматривает нормировки по длине[2].

Если запрос является длинным, например подробное описание информационной потребности длиной в абзац, то аналогичное взвешивание можно применить к терминам запроса. Для коротких запросов в этом нет необходимости.

$$RSV_d = \sum_{t \in q} \log\left[\frac{N}{df_t}\right] \cdot \frac{(k_1+1)tf_{td}}{k_1\left((1-b)+b\times\left(L_d/L_{ave}\right)\right)+tf_{td}} \cdot \frac{(k_3+1)tf_{tq}}{k_3+tf_{tq}} \tag{11.33}$$

Здесь tf_{tq} — частота термина t в запросе q, а k_3 — еще один параметр настройки, который влияет на нормировку частоты термина запроса. В представленном равенстве не произведена нормировка длины (т.е. $b = 0$). Нормировка длины запроса необязательна, поскольку поиск выполняется по одному фиксированному запросу. Параметры настройки в этих формулах в идеальном случае должны обеспечить оптимальную производительность на рабочей тестовой коллекции. Иначе говоря, можно определить значения этих параметров, обеспечивающие максимум производительности на отдельной тестовой коллекции (либо вручную, либо с помощью методов оптимизации, таких как сеточный поиск (grid search) или какой-нибудь более совершенный метод), а затем использовать эти параметры на коллекции для окончательного тестирования. Без такой оптимизации, как показали эксперименты, разумно выбрать следующие параметры: k_1 и k_3 — между 1,2 и 2, а $b = 0{,}75$.[3]

Если есть оценки релевантности, то вместо приближения $\log(N/df_t)$, предложенного в формуле (11.22), можно использовать полную формулу (11.21).

$$RSV_d = \sum_{t \in q} \left[\log\left[\frac{\left(|VR_t| + \frac{1}{2}\right) \Big/ \left(|VNR_t| + \frac{1}{2}\right)}{\left(df_t - |VR_t| + \frac{1}{2}\right) \Big/ \left(N - df_t - |VR| + |VR_t| + \frac{1}{2}\right)} \right] \times \right.$$
$$\left. \times \frac{(k_1+1)tf_{td}}{k_1\left((1-b)+b\times\left(L_d/L_{ave}\right)\right)+tf_{td}} \times \frac{(k_3+1)tf_{tq}}{k_3+tf_{tq}} \right] \tag{11.34}$$

Здесь VR_t, VNR_t и VR — множества, введенные в разделе 11.3.4. Первая часть этого выражения отражает наличие обратной связи по релевантности (или просто схему взвешивания idf, если информации о релевантности нет), вторая часть содержит частоту термина и масштабирование с учетом длины документа, а третья учитывает частоту термина в запросе.

Обратная связь по релевантности предусматривает не только пересчет весов терминов запроса пользователя, но и расширение запроса (автоматическое или с ручной проверкой). Запрос расширяется лучшими (скажем, десятью или двадцатью) терминами из документов, оцененных как релевантные, на основе показателя $\hat{c}_t$ из равенства (11.21). В приведенной выше формуле может использоваться расширенный запрос q.

Схема взвешивания терминов BM25 применялась довольно широко и успешно для разных коллекций и поисковых задач. Особенно хорошо она зарекомендовала себя в

---

[2] Этот способ нормировки на длину документа схож с формулой “опорной нормировки” (раздел 6.4.4) и был предложен практически одновременно с ней: параметр b соответствует (1-alpha), а L_{ave} соответствует опорной длине документа piv. См., например, Chowdhury, 2002. — *Примеч. ред.*

[3] Часто встречается также значение b = 0.6. — *Примеч. ред.*

рамках экспериментов TREC. Мотивация ее применения и описание экспериментальных результатов приведены в работе Спарка Джоунс (Sparck Jones et al., 2000).

11.4.4. Байесовские сетевые подходы к информационному поиску

Как сообщают Йенсен и Йенсен (Jensen and Jensen, 2001), использовать *байесовские сети* (Bayesian networks), одну из форм вероятностных графических моделей, в информационном поиске впервые предложили Тертл и Крофт (Turtle and Croft, 1989, 1991). Мы не будем углубляться в детали, поскольку полное описание формализма байесовских сетей заняло бы слишком много места, но в принципе байесовские сети основаны на ориентированных графах, иллюстрирующих вероятностные зависимости между переменными, как, например, на рис. 11.1. Этот формализм привел к появлению сложных алгоритмов распространяющегося воздействия, чтобы обеспечить обучение и выводы на основе произвольных знаний в произвольных ориентированных ациклических графах. Тертл и Крофт использовали изощренную сеть для моделирования сложных зависимостей между документом и информационной потребностью пользователя.

Эта модель состоит из двух частей: сети коллекции документов и сети запроса. Сеть документов велика, но допускает предварительное вычисление; она отображает документы в термины, а затем — в концепции. Концепции — это расширения терминов, встречающихся в документе, с помощью тезауруса. Сеть запроса относительно мала, но при поступлении нового запроса ее необходимо создать заново, а затем присоединить к сети документов. Сеть запроса отображает термины запроса в выражения на основе запроса (построенные на основе вероятностных, или “зашумленных”, вариантов операторов AND и OR), которые, в свою очередь, отображаются в информационную потребность.

В результате возникает гибкая вероятностная сеть, которая способна обобщать разные булевы и вероятностные модели. Действительно, ее можно рассматривать как первый вариант вероятностной ранжирующей модели поиска, естественным образом поддерживающей структурные операторы в запросах. Эта система обеспечивала высокопроизводительный крупномасштабный поиск и стала основой текстовой поисковой системы InQuery, созданной в Университете Массачусетса. Она очень хорошо зарекомендовала себя в рамках экспериментов TREC и одно время распространялась на коммерческой основе. С другой стороны, эта модель по-прежнему использовала различные приближения и предположения о независимости, чтобы сделать возможными оценки параметров и другие вычисления. В этом направлении работа продвинулась недалеко, но следует отметить, что описываемая модель возникла на заре эры использования байесовских сетей, после чего были получены важные теоретические результаты. Возможно, что настало время для нового поколения информационно-поисковых систем, основанных на байесовских сетях.

11.5. Библиография и рекомендации для дальнейшего чтения

Более полное введение в теорию вероятностей можно найти в многочисленных учебниках (Grinstead and Snell, 1997; Rice, 2006; Ross, 2006).[4] Введение в байесовскую теорию полезности изложено в книге Рипли (Ripley, 1996).

[4] Существует огромное количество учебников по теории вероятностей на русском языке, перечислять которые здесь нецелесообразно. Читателям, интересующимся как фундаментальными, так и прикладными вопросами теории вероятностей, рекомендуем обратиться к монографии Феллер В., Введение в теорию вероятностей и ее приложения. Т. 1–2. — 1984. — М.: Наука. — *Примеч. ред.*

Вероятностный подход к информационному поиску возник в Великобритании в 1950-х годах. Первое описание вероятностной модели привели Марон и Кунс (Maron and Kuhns, 1960). Основы модели BIM заложили Робертсон и Джоунс (Robertson and Jones, 1979), а детали классической вероятностной модели BIM представил ван Рийсберген (van Rijsbergen, 1979). Идея принципа PRP приписывается Робертсону (S.E. Robertson), Марону (M.E. Maron) и Куперу (W.S. Cooper). Робертсон и Джоунс (Robertson and Jones, 1976) использовали термин *принцип вероятностного упорядочения* (Probabilistic Ordering Principle — POP), но позднее общепринятым стал *принцип вероятностного ранжирования* (Probability Ranking Principle — PRP). Более современное представление теории вероятностного информационного поиска изложено в работе Фура (Fuhr, 1992), в которой описаны и другие подходы, такие как вероятностная логика и байесовские сети. Другой обзор можно найти в работе Крестани и др. (Crestani et al., 1998). Спарк Джоунс и др. (Spärck Jones et al., 2000) дали исчерпывающие описания экспериментов в области вероятностного поиска, проведенных “Лондонской школой”, а Робертсон (Robertson, 2005) ретроспективно изложил результаты участия группы в проекте TREC, включая подробное обсуждение функции взвешивания Okapi BM25 и историю ее разработки. Робертсон и др. (Robertson et al., 2004) распространили метод BM25 на документы с различными полями.

Поисковая система с открытым кодом Indri, которая распространяется вместе в составе пакета Lemur (`www.lemurproject.org`), объединяет идеи байесовских сетей и подхода на основе языковых моделей (см. главу 12). В частности, система поддерживает структурные операторы запросов.

Глава 12

Языковые модели для информационного поиска

Для того чтобы составить хороший запрос, пользователям рекомендуется представить, какие слова могут встретиться в релевантных документах, и именно их использовать в запросе. Эта идея нашла свое непосредственное воплощение в языковых моделях информационного поиска. В рамках этого подхода считается, что документ соответствует запросу, если модель документа с большой вероятностью порождает этот запрос, что, в свою очередь, случается, если в документе часто встречаются слова запроса. Таким образом, этот подход представляет собой альтернативную реализацию одной из основных идей ранжирования документов (см. раздел 6.2). Вместо непосредственного моделирования вероятности $P(R = 1|q, d)$ релевантности документа d по отношению к запросу q, как в традиционных вероятностных моделях информационного поиска (см. главу 11), этот подход предусматривает создание вероятностной языковой модели M_d для каждого документа d и ранжирование документов в соответствии с вероятностью $P(q|M_d)$ того, что построенная модель M_d порождает запрос q.

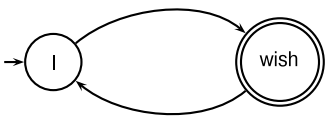
В данной главе сначала вводится понятие языковых моделей (раздел 12.1), а затем описывается наиболее распространенный в рамках информационного поиска подход к языковому моделированию — "модель правдоподобия запроса" (раздел 12.2). После сравнения языкового моделирования и других подходов к информационному поиску (раздел 12.3) мы закончим изложение кратким описанием разных расширений языковых моделей (раздел 12.4).

12.1. Языковые модели

12.1.1. Конечные автоматы и языковые модели

Что означает выражение "Модель документа порождает запрос"? Для того чтобы распознавать или генерировать строки, можно использовать традиционную *языковую порождающую модель* (generative model), концепция которой пришла из теории формальных языков. Например, конечный автомат, изображенный на рис. 12.1, может генерировать строки, включающие в себя приведенные примеры. Полная совокупность строк, которую может сгенерировать конечный автомат, называется *языком автомата* (language of the automation).[1]

[1] Результаты работы конечных автоматов можно приписывать как их состояниям, так и дугам; в данном случае мы использовали состояния, поскольку это соответствует обычному способу формализации вероятностных автоматов.



I wish
I wish I wish
I wish I wish I wish
I wish I wish I wish I wish I wish I wish
...

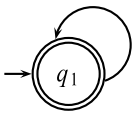
Не может генерировать:

Рис. 12.1. Простой конечный автомат и некоторые строки, которые он может генерировать. Стрелка показывает начальное состояние автомата, а двойная окружность — (возможное) конечное состояние.

Если каждый узел имеет распределение вероятностей над различными генерируемыми терминами, то мы получим языковую модель. Понятие языковой модели, по существу, является вероятностным. *Языковая модель* (language model) — это функция, которая каждой строке, выведенной из определенного словаря, приписывает некую вероятность. Иначе говоря, для языковой модели M над алфавитом Σ выполняется равенство

$$\sum_{s\in\Sigma^*} P(s)=1. \tag{12.1}$$

Простейшей разновидностью языковой модели является вероятностный конечный автомат, состоящий из одного узла и одного распределения вероятностей над генерируемыми терминами, так что $\sum_{t\in V} P(t)=1$, как показано на рис. 12.2. После генерирования каждого слова мы решаем, останавливаться или продолжать цикл и генерировать следующее слово, поэтому модель требует также указать вероятность остановки в конечном состоянии. Такие модели определяют вероятность любой последовательности слов. Такой автомат представляет собой модель порождения текста в соответствии с заданным распределением.



$P(\text{stop}\,|q_1) = 0.2$

the	0.2
a	0.1
frog	0.01
toad	0.01
said	0.03
likes	0.02
that	0.04
...	...

Рис. 12.2. Конечный автомат с одним состоянием, действующий как униграммная модель. Показано частичное определение вероятностей состояния

Пример 12.1. Для того чтобы вычислить вероятность последовательности слов, следует просто перемножить вероятности появления каждого из этих слов в последовательности, а также вероятность продолжения или остановки после генерации каждого слова, например

$$P(\text{frog said toad likes frog}) = (0{,}01 \times 0{,}03 \times 0{,}04 \times 0{,}01 \times 0{,}02 \times 0{,}01) \times \\ \times (0{,}8 \times 0{,}8 \times 0{,}8 \times 0{,}8 \times 0{,}8 \times 0{,}2) \approx 0{,}0000000000001573. \quad (12.2)$$

Как видим, вероятность конкретной строки или документа обычно очень мала! В данном случае мы остановились после генерирования слова *frog* во второй раз. Первая строка цифр представляет собой вероятности эмиссии термина, а вторая строка содержит вероятности продолжения или остановки после генерирования каждого слова. В соответствии с равенством (12.1), для того чтобы языковая модель была согласованной, конечный автомат должен иметь явную вероятность остановки. Тем не менее в большинстве случаев мы будем игнорировать вероятности состояний `STOP` и `1-STOP` (как и большинство авторов). Для сравнения двух моделей на одном и том же наборе данных можно вычислить *отношение правдоподобия* (likelihood ratio), представляющее собой вероятность данных в соответствии с одной моделью, деленную на вероятность данных по другой модели. Учитывая, что вероятность остановки является фиксированной, ее учет не влияет на отношение правдоподобия, возникающее в результате сравнения вероятностей того, что две языковые модели сгенерируют одну и ту же строку. Следовательно, она не влияет и на ранжирование документов.[2] Тем не менее с формальной точки зрения величины больше нельзя считать настоящими вероятностями, а можно считать лишь пропорциональными вероятностям (см. упражнение 12.4).

Пример 12.2. Допустим теперь, что мы сравниваем модели M_1 и M_2, частично продемонстрированные на рис. 12.2. Каждая дает оценку вероятностей последовательности терминов, как показано в примере 12.1. Если языковая модель приписывает последовательности терминов более высокую вероятность, то более вероятно, что именно она сгенерировала данную последовательность. На этот раз мы проигнорируем вероятности остановки при расчетах. Для продемонстрированной последовательности мы получаем следующие результаты.

s	frog	said	that	toad	likes	that	dog	
M_1	0,01	0,03	0,04	0,01	0,02	0,04	0,005	(12.3)
M_2	0,0002	0,03	0,04	0,0001	0,04	0,04	0,01	

$P(s|M_1) = 0{,}00000000000048$

$P(s|M_2) = 0{,}000000000000000384$

Как видим, $P(s|M_1) > P(s|M_2)$. Здесь формулы представляют собой произведение вероятностей, но, как обычно в приложениях теории вероятностей, на практике удобнее работать с суммами логарифмов вероятностей.

[2] В контексте информационного поиска, в котором мы рассматриваем языковые модели, фиксирование вероятности остановки в различных моделях представляется разумным. Это объясняется тем, что мы генерируем запросы, а распределение длин запросов фиксировано и не зависит от документа, по которому мы создаем языковую модель.

	Модель M_1		Модель M_2
the	0,2	the	0,15
a	0,2	a	0,12
frog	0,01	frog	0,0002
toad	0,01	toad	0,0001
said	0,03	said	0,03
likes	0,02	likes	0,04
that	0,04	that	0,04
dog	0,005	dog	0,01
cat	0,003	cat	0,015
monkey	0,001	monkey	0,002
...	...	...	...

Рис. 12.3. Частичная спецификация двух униграммных языковых моделей

12.1.2. Типы языковых моделей

Как вычислить вероятности последовательностей терминов? Для этого можно использовать правило умножения вероятностей (11.1), разложив вероятность последовательности на вероятности появления отдельных терминов при условии появления предыдущих терминов.

$$P(t_1t_2t_3t_4) = P(t_1)P(t_2|t_1)P(t_3|t_1t_2)P(t_4|t_1t_2t_3) \tag{12.4}$$

В простейшей форме языковой модели все условные вероятности просто отбрасываются, и все термины считаются независимыми друг от друга. Такая модель называется *униграммной языковой моделью* (unigram language model).

$$P_{\text{uni}}(t_1t_2t_3t_4) = P(t_1)P(t_2)P(t_3)P(t_4) \tag{12.5}$$

Существуют более сложные языковые модели, например *биграммные* (bigram language models), в которых используется вероятность предыдущего термина.

$$P_{\text{uni}}(t_1t_2t_3t_4) = P(t_1)P(t_2|t_1)P(t_3|t_2)P(t_4|t_3) \tag{12.6}$$

Еще более сложные языковые модели основаны на грамматике, например вероятностные контекстно-свободные грамматики. Эти модели очень важны для решения таких задач, как распознавание речи, исправление орфографических ошибок и машинный перевод, когда необходимо знать вероятность появления термина в зависимости от окружающего контекста. Однако в информационном поиске чаще всего использовались униграммные языковые модели. Информационный поиск — это не та область, в которой сразу необходимо применять сложные языковые модели, поскольку поиск не настолько зависит от структуры предложения, как другие задачи, например распознавание речи. Часто для того, чтобы понять тему текста, вполне достаточно униграммной модели. Более того, как будет показано ниже, языковые модели информационного поиска часто строятся на основе одного документа, и поэтому возникает вопрос, можно ли на этом небольшом объеме данных сделать что-то более сложное. Потери от разреженности данных (см. раздел 13.2) часто перевешивают любой выигрыш, достигнутый благодаря сложным моделям. В качестве примера можно назвать компромисс между смещением и дисперсией (ср. раздел 14.6). При ограниченном объеме обучающих данных лучше работают более простые модели. Кроме того, униграммные модели эффективнее получать и

применять, чем модели более высокого порядка. Тем не менее важность фразовых запросов и запросов с учетом близости слов запроса в документе мотивируют использование в будущем более сложных языковых моделей (некоторые результаты в этом направлении описаны в разделе 12.5). Действительно, движение в этом направлении вызывает аналогии с моделью ван Рийсбергена (глава 11).

12.1.3. Мультиномиальные распределения по словам

В униграммной языковой модели порядок слов не важен, поэтому такие модели часто называют “мешками слов”, как указывалось в главе 6. Хотя на предшествующий контекст не наложены никакие условия, эта модель все равно позволяет вычислить вероятность определенной последовательности терминов. Однако любой другой порядок следования терминов в этом “мешке слов” будет иметь такую же вероятность. Таким образом, название униграммной языковой модели и ее мотивация объясняются историческими, а не логическими причинами. Вместо этого ее следовало бы назвать мультиномиальной. Уравнения, приведенные выше, не отражают мультиномиальной вероятности “мешка слов”, поскольку в них нет суммирования по всем возможным вариантам следования этих слов, как это делается в мультиномиальном коэффициенте (первый множитель в правой части) стандартной мультиномиальной модели.

$$P(d) = \frac{L_d!}{tf_{t_1,d}!\, f_{t_2,d}!\ldots f_{t_M,d}!} P(t_1)^{tf_{t_1,d}} P(t_2)^{tf_{t_2,d}} \ldots P(t_M)^{tf_{t_M,d}} \qquad (12.7)$$

Здесь $L_d = \sum_{1 \le i \le M} tf_{t_i,d}$ — длина документа d, M — размер словаря терминов, а произведение вычисляется по терминам из словаря, а не по позициям в документе. Однако на практике мы можем игнорировать мультиномиальный коэффициент в своих вычислениях, так же как и вероятность остановки; для конкретного “мешка слов” он является постоянным и поэтому не влияет на отношение правдоподобия двух разных моделей, генерирующих конкретный “мешок слов”. Мультиномиальные распределения будут рассмотрены в разделе 13.2.

Основной проблемой, связанной с разработкой языковых моделей, является то, что мы не знаем точно, что именно следует использовать в качестве модели M_d. Однако, как правило, у нас есть пример текста, представляющий эту модель. Данная проблема играет важную роль в применении языковых моделей для решения тех классических задач, для которых она и была изначально предложена. Например, при распознавании речи у нас есть обучающие выборки произнесенного текста. И все же мы обязаны предполагать, что в будущем пользователь будет употреблять другие слова в другой последовательности, которые мы еще не видели, поэтому модель должна допускать обобщение наблюдаемых данных на неизвестные слова и последовательности. В случае информационного поиска, когда документ конечен и обычно фиксирован, такая интерпретация не столь очевидна. Стратегия, которую мы применяем для информационного поиска, выглядит следующим образом. Будем считать, что документ d является единственным репрезентативным примером текста, извлеченного из распределения текстов модели, рассматривая ее как специфическую “тему” [3]. Затем создадим языковую модель на основе этого примера и при-

[3] Ср. аналогию языковой модели с классификацией документов методом Байеса в следующем разделе. — *Примеч. ред.*

меним ее для вычисления вероятности любой последовательности слов, а в заключение отранжируем документы по вероятности сгенерировать запрос.

? **Упражнение 12.1 [*].** Чему равна сумма оценок вероятностей всех строк в языке, длина которых равна единице, с учетом вероятности остановки? Будем считать, что вы генерируете слово, а затем решаете, остановиться или нет (т.е. нулевая строка не является частью языка).

Упражнение 12.2 [*]. Чему равна сумма весов, приписанных строкам единичной длины в языке, если вероятность остановки не учитывается?

Упражнение 12.3 [*]. Чему равно отношение правдоподобия документа в соответствии с моделями M_1 и M_2 в примере 12.2?

Упражнение 12.4 [*]. В примере 12.2 вероятность остановки не учитывается. Допустим, что в каждой модели она равна 0,1. Изменит ли это отношение правдоподобия документа по отношению к обеим моделям?

Упражнение 12.5 [**]. Как можно использовать языковую модель в системе исправления орфографических ошибок? В частности, рассмотрите вариант исправления ошибок с учетом контекста и неправильного использования слов, таких как *their* в предложении “Are you their?” (см. библиографические ссылки на эту тему в разделе 3.5).

12.2. Модель правдоподобия запроса

12.2.1. Использование языковых моделей правдоподобия запроса в информационном поиске

Языковое моделирование представляет собой довольно общую формальную теорию в области информационного поиска, имеющую разнообразные реализации. Основным методом использования языковых моделей в информационном поиске является *модель правдоподобия запроса* (query likelihood model). В ней для каждого документа d из коллекции создается языковая модель M_d. Наша цель — ранжировать документы по вероятности $P(d|q)$, где вероятность документа интерпретируется как правдоподобие того, что он релевантен запросу. Используя правило Байеса (см. раздел 11.1), получим следующее равенство.

$$P(d|q) = P(q|d)P(d)/P(q)$$

Здесь $P(q)$ — вероятность, одинаковая для всех документов, и поэтому ее можно игнорировать. Априорное распределение вероятностей документа часто считается равномерным по всем документам, и поэтому вероятность $P(d)$ также может быть отброшена, хотя можно было бы попробовать оценить априорную вероятность, учитывая такие показатели, как авторитетность, длина, жанр, новизна и количество людей, уже прочитавших документ. Однако, сделав указанные упрощения, мы получим результаты, ранжированные просто по вероятности $P(q|d)$ того, что запрос q в рамках языковой модели выведен из документа d. Таким образом, языковое моделирование пытается имитировать процесс генерации запроса: документы ранжируются по вероятности того, что запрос может появиться как случайная выборка из соответствующей модели документа.

Чаще всего для этого используется мультиномиальная униграммная языковая модель, которая эквивалентна мультиномиальной наивной байесовской модели, в которой документы — это классы, играющие роль отдельных “языков”. В этой модели выполняется следующее равенство.

$$P(q|M_d) = K_q \prod_{t \in V} P(t|M_d)^{tf_{t,d}} \tag{12.8}$$

Здесь $K_q = L_q / (tf_{t_1,d}! f_{t_2,d}! \ldots f_{t_M,d}!)$ — мультиномиальный коэффициент для запроса q, который для конкретного запроса является постоянным и может не учитываться.

Для поиска на основе языковой модели генерирование запросов рассматривается как случайный процесс.

1. Выводим языковую модель для каждого документа.
2. Оцениваем $P(q|M_{d_i})$, вероятность генерирования запроса в соответствии с каждой из этих моделей.
3. Ранжируем документы в соответствии с этими вероятностями.

В основе данной модели лежит интуитивное предположение, что пользователь держит в уме прототип документа и генерирует запрос на основе слов, которые содержатся в этом документе. Часто пользователи имеют разумные предположения о терминах, которые, вероятно, могут встретиться в интересующих их документах, и выбирают термины в запросе так, чтобы отличить искомый документ от других документов коллекции. [4] Статистические характеристики коллекции являются неотъемлемой частью языковой модели, а не используются эвристически, как в других методах.

12.2.2. Оценка вероятности порождения запроса

В этом разделе мы покажем, как оценить вероятность $P(q|M_d)$. Вероятность сгенерировать запрос по заданной языковой модели M_d, построенной по документу d, на основе оценки максимального правдоподобия (MLE) и предположения о независимости терминов (т.е. в рамках униграммной модели) вычисляется по следующей формуле.

$$\hat{P}(q|M_d) = \prod_{t \in q} \hat{P}_{MLE}(t|M_d) = \prod_{t \in q} \frac{tf_{t,d}}{L_d} \tag{12.9}$$

Здесь M_d — языковая модель документа d, $tf_{t,d}$ — частота термина t в документе d, а L_d — количество лексем в документе d. Иначе говоря, мы просто подсчитываем, сколько раз появляется каждое слово, и делим частоту на общее количество слов в документе d. Тот же самый метод вычисления оценки MLE был приведен в разделе 11.3.2, но теперь мы используем мультиномиальное распределение частот слов.

Классическая проблема, связанная с языковыми моделями, — оценка вероятности (символ $\wedge$ над буквой P подчеркивает, что в модели используются оценки). Термины в документах повторяются очень редко. В частности, некоторые слова вообще не встречаются в документах, но являются вполне возможными для выражения информационной потребности. Если для термина, отсутствующего в документе, оценка вероятности $\hat{P}(t|M_d)$ равна нулю, то мы получим строго конъюнктивную семантику: запрос имеет

[4] Разумеется, во многих случаях это не так. Для решения проблемы в рамках данного подхода применяются языковые модели перевода (translation language models) (раздел 12.4).

ненулевую вероятность, только если все термины запроса содержатся в документе. Нулевые вероятности создают очевидные проблемы в других приложениях языковых моделей, например для предсказания следующего слова при распознавании речи, поскольку многие слова редко встречаются в обучающей выборке. Проблематичность нулевых вероятностей в информационном поиске не так очевидна. Можно считать, что эта сложность относится к аспектам человеко-машинного интерфейса: системы, основанные на модели векторного пространства, обычно отдают предпочтение неполным совпадениям, хотя недавние разработки в области веб-поиска демонстрируют тенденцию к использованию конъюнктивной семантики. Независимо от выбранного подхода проблема оценки остается: частота встречающихся слов оценивается очень плохо; в частности, вероятность появления слова, встречающегося в документе один раз, обычно переоценивается, поскольку это появление может носить отчасти случайный характер. Решением данной проблемы (как показано в разделе 11.3.2) является сглаживание. Тем не менее по мере того, как люди стали лучше понимать подход, основанный на языковых моделях, стало ясно, что роль сглаживания в этой модели не сводится только к устранению нулевых вероятностей. Сглаживание терминов стало важной частью вычисления их весов (упражнение 12.8). Это значит, что несглаженные модели имеют не только конъюнктивную семантику; несглаженные модели работают плохо, потому что они не учитывают важный компонент веса термина.

Таким образом, для того чтобы устранить нулевые вероятности и присвоить определенные вероятности словам, которые в тексте не встретились, в языковых моделях необходимо применять сглаживание. Для сглаживания распределений вероятностей предложено много методов. В разделе 11.3.2 мы уже обсуждали добавление числа (1, 1/2 или малого α) к наблюдаемым частотам и их нормировку.[5] В этом разделе мы рассмотрим ряд других методов сглаживания, связанных с комбинированием наблюдаемых частот с более общим распределением вероятности. Считается, что термины, которые не появляются в документе, могут появляться в запросах, однако их вероятность должна быть меньше или равна вероятности случайного появления термина во всей коллекции. Иначе говоря, если $\mathrm{tf}_{t,d} = 0$, то

$$\hat{P}\left(t|M_d\right) \le cf_t/T\,.$$

Здесь cf_t — частота термина в коллекции, а T — количество лексем в коллекции. Простая идея, которая оправдала себя на практике, состоит в том, чтобы использовать смесь мультиномиального распределения, полученного на основе документа, и мультиномиального распределения, полученного по всей коллекции.

$$\hat{P}\left(t|d\right) = \lambda \hat{P}_{MLE}\left(t|M_d\right) + \left(1-\lambda\right)\hat{P}_{MLE}\left(t|M_c\right) \tag{12.10}$$

Здесь $0 < \lambda < 1$, а M_c — языковая модель, построенная по всей коллекции документов. Это распределение представляет собой смесь распределений вероятностей термина по документу и по коллекции. Такая модель называется *языковой моделью, основанной на*

[5] В контексте теории вероятностей нормировкой называется подсчет количества каждого события и деление полученной величины на общую частоту событий, чтобы в результате получилось распределение вероятностей, сумма которых равна единице. Этот процесс отличается от нормировки, описанной в главе 2, и нормировки длины в главе 6, которая осуществляется с помощью нормы в пространстве L_2.

линейной интерполяции (linear interpolation LM). [6] Для хорошего функционирования этой модели важен правильный выбор параметра λ.

Альтернативой является использование языковой модели, построенной по всей коллекции, в качестве априорного распределения для *байесовского процесса уточнения данных* (Bayesian updating process) (в противоположность равномерному распределению, как в разделе 11.3.2).

$$\hat{P}(t|d) = \frac{tf_{t,d} + \alpha \hat{P}(t|M_c)}{L_d + \alpha} \tag{12.11}$$

Оба эти метода сглаживания хорошо показали себя в экспериментах. В оставшейся части раздела мы будем придерживаться метода сглаживания с помощью линейной интерполяции. Несмотря на различия в деталях оба метода, в принципе, похожи друг на друга. В обоих вариантах оценка вероятности появления слова в документе представляет собой сочетание дисконтированной оценки максимального правдоподобия и доли оценки на основе встречаемости во всей коллекции, причем для слов, не представленных в документе, оценка базируется только на встречаемости во всей коллекции.

Сглаживание в языковых моделях не сводится лишь к решению проблем, связанных с оценками. Этот факт еще не был ясен, когда модели были предложены впервые, но сейчас понятно, что сглаживание позволяет улучшить качество модели в целом. Причины этого явления будут исследованы в упражнении 12.8. Степень сглаживания в описанных моделях зависит от параметров λ и α. Небольшое значение параметра λ или большое значение параметра α означает сильное сглаживание. Управляя этими параметрами, можно оптимизировать производительность модели, используя линейный поиск (или, в случае модели линейной интерполяции другими методами, например, EM-алгоритм, см. раздел 16.5). Их величины не обязаны быть постоянными. Один из подходов — сделать значение функцией от размера запроса. Это полезно, поскольку для коротких запросов лучше подходит небольшое сглаживание (“почти конъюнктивный” поиск), а для длинных — сильное сглаживание.

Итак, функция ранжирования результатов поиска по запросу q в рамках базовой языковой модели выглядит так.

$$P(d|q) \propto P(d) \prod_{t \in q} \left((1-\lambda) P(t|M_c) + \lambda P(t|M_d) \right) \tag{12.12}$$

Это уравнение характеризует вероятность того, что документ, который пользователь имел в виду, — это действительно документ d.

Пример 12.3. Допустим, что коллекция состоит из двух документов.

- d_1: Xyzzy reports a profit but revenue is down
- d_2: Quorus narrows quarter loss but revenue decreases further

Модель представляет собой смесь двух MLE-моделей, построенных по документам и коллекции, и смешанных с $\lambda = 1/2$.

Допустим, что запрос имеет вид *revenue down*. Тогда

[6] Этот метод еще называют *сглаживанием Елинека–Мерсера* (Jelinek–Mercer smoothing).

$$P(q|d_1) = [(1/8 + 2/16)/2] \times [(1/8 + 2/16)/2] = 1/8 \times 3/32 = 3/256,$$
$$P(q|d_2) = [(1/8 + 2/16)/2] \times [(0/8 + 1/16)/2] = 1/8 \times 1/32 = 1/256. \quad (12.13)$$

Итак, ранг документа d_1 больше ранга документа d_2.

12.2.3. Эксперименты Понте и Крофта

Понте и Крофт (Ponte and Croft, 1998) описали первые эксперименты, касающиеся применения языковых моделей в области информационного поиска. В качестве основной они использовали описанную выше модель. Однако мы придерживались подхода, в котором языковая модель представляла собой смесь двух мультиномиальных распределений, как в работах Миллера, Хиемстра и др. (Miller et al., 1999; Hiemstra, 2000), в то время как Понте и Крофт использовали многомерную модель Бернулли. Использование мультиномиальных распределений со временем стало общепринятым в большинстве работ по языковым моделям и экспериментов по информационному поиску. Эксперименты по классификации текстов, рассмотренные в разделе 13.3, также свидетельствуют о том, что эти модели являются наилучшими. Понте и Крофт убедительно доказали, что веса терминов, определенные по языковым моделям, намного эффективнее, чем веса tf–idf. Часть результатов Понте и Крофта представлены на рис. 12.4. Здесь приведены результаты сравнения взвешивания на основе tf–idf и языковых моделей применительно к заданиям TREC 202–250 и данным дисков 2 и 3. Запросы представляют собой предложения на естественном языке. Языковые модели продемонстрировали превосходство над схемой взвешивания терминов tf–idf. Это преимущество затем было подтверждено в последующих работах.

Точность				
Полнота	tf–idf	LM	Процентное изменение	
0,0	0,7439	0,7950	+2,0	
0,1	0,4521	0,4910	+8,6	
0,2	0,3514	0,4045	+15,1	*
0,3	0,2761	0,3342	+21,0	*
0,4	0,2093	0,2572	+22,9	*
0,5	0,1558	0,2061	+32,3	*
0,6	0,1024	0,1405	+37,1	*
0,7	0,0451	0,0760	+68,7	*
0,8	0,0160	0,0432	+169,6	*
0,9	0,0033	0,0063	+89,3	
1,0	0,0028	0,0050	+76,9	
Среднее	0,1868	0,2233	+19,55	*

Рис. 12.4. Результаты сравнения схем взвешивания терминов tf–idf и LM, полученные Понте и Крофтом (1998). Вариант схемы tf–idf, реализованный в информационно-поисковой системе INQUERY, предусматривает нормировку tf по длине. В таблице приведены оценки, соответствующие средней точности по одиннадцати точкам. Статистически значимые результаты в соответствии с ранговым критерием Уилкоксона (Wilcoxon signed-rank test) отмечены звездочкой. Языковые модели в этих экспериментах всегда оказывались лучше, но следует заметить, что подход обеспечивает значительное превосходство в области больших значений полноты

? **Упражнение 12.6 [*].** Попробуйте создать языковую модель на основе следующего обучающего текста.

the martian has landed on the latin pop sensation ricky martin

1. Чему равны вероятности P(the) и P(martian) в униграммной вероятностной MLE-модели?
2. Чему равны вероятности P(sensation|pop) и P(pop|the) в биграммной вероятностной MLE-модели?

Упражнение 12.7 [**]. Представим себе коллекцию, состоящую из четырех документов, приведенных в следующей таблице.

docID	Текст документа
1	click go the shears boys click click click
2	click click
3	metal here
4	metal shears click here

Постройте языковую модель правдоподобия запроса для этой коллекции документов, предполагая, что веса моделей для документа и коллекции в смеси равны 1/2. Для оценки обеих униграммных моделей используйте метод максимального правдоподобия. Посчитайте на основе модели вероятности запросов click, shears и click shears для каждого документа и используйте эти вероятности для ранжирования документов, найденных по каждому запросу. Заполните следующую таблицу.

Запрос	doc 1	doc 2	doc 3	doc 4
click				
shears				
click shears				

Как окончательно ранжируются документы в ответ на запрос click shears?

Упражнение 12.8 [**]. Ориентируясь на вычисления из упражнения 12.7 в качестве примера, напишите по одному предложению, описывающему обработку моделью (12.10) следующих показателей. Для каждого показателя укажите, входит он в модель или нет и в каком виде: в масштабированном или обычном.

1. Частота термина в документе
2. Частота термина в коллекции
3. Документная частота термина
4. Нормировка термина по длине

Упражнение 12.9 [**]. В смешанном подходе к модели правдоподобия запроса (равенство (12.12)) оценка вероятности термина основывается на частотах слова в документе и коллекции. Это гарантирует, что каждый термин запроса (из словаря) имеет ненулевой шанс быть сгенерированным по каждому документу. Однако это условие оказывает менее заметное, но важное влияние на взвешивание терминов (см. главу 6). Объясните механизм этого явления. В частности, приведите конкретный числовой пример вычисления веса термина.

12.3. Сравнение языкового моделирования с другими подходами к информационному поиску

Языковые модели отражают новую точку зрения на проблему поиска текста, связанную с большим количеством исследований, посвященных распознаванию речи и обработке естественного языка. Как подчеркивают Понте и Крофт (Ponte and Croft, 1998), языковые модели в информационном поиске позволяют по-иному оценивать соответствие между запросам и документами, а также дают надежду на то, что вероятностные языковые модели точнее вычисляют веса, а значит, повышают качество поиска. Основной проблемой остается оценка модели документа, например выбор эффективного способа сглаживания. Языковые модели обеспечивают хорошие результаты поиска. По сравнению с другими вероятностными подходами, такими как модель BIM из главы 11, основное отличие непосредственно проявляется в том, что языковая модель не предназначена для явного моделирования релевантности (в то время как именно релевантность — центральное понятие в рамках модели BIM). Однако, как указано в работах, описанных в разделе 12.5, это, возможно, не вполне корректные рассуждения. Языковые модели предполагают, что документы и выражения информационных потребностей — это объекты одного рода, и оценивают их соответствие с помощью инструментов, заимствованных из арсенала методов обработки речи и естественного языка. В результате возникает математически строгая, концептуально простая, вполне реализуемая с вычислительной точки зрения и интуитивно привлекательная модель. Это напоминает XML-поиск (глава 10); там также подходы, основанные на предположении о принадлежности запросов и документов одному типу, оказались наиболее успешными.

С другой стороны, как и любые модели информационного поиска, языковые модели не лишены недостатков. Предположение об эквивалентности представления документа и информационной потребности нереалистично. В настоящее время используются очень простые, как правило, униграммные языковые модели. Без явного понятия релевантности обратную связь по релевантности, как и предпочтения пользователей, трудно интегрировать в модель. Кроме того, очевидно, существует необходимость выйти за пределы униграммных моделей, чтобы обеспечить сопоставление фраз и фрагментов, а также использование логических операторов. Более поздние работы в этой области были сосредоточены именно на этих проблемах, в том числе на включении релевантности в модель и допущении языковых несоответствий между языками запросов и документов.

Языковая модель тесно связана с традиционными моделями tf–idf. Частота термина в моделях tf–idf участвует непосредственно, причем в большинстве современных работ признается важность нормировки по длине документа. Смесь распределений вероятностей, определенных по документу и коллекции, немного напоминает idf; термины, которые редко встречаются в коллекции и часто — в некоторых документах, оказывают более сильное влияние на ранжирование документов. В большинстве конкретных реализаций на основе того или иного подхода термины обрабатываются как независимые друг от друга. С другой стороны, интуиция лучше выражается в терминах вероятностей, а не в геометрии, математические модели являются более строгими, а не эвристическими, а детали, например использование частоты термина и длины документа, в языковых моделях и модели tf–idf отличаются. Результаты недавних экспериментов по информационному поиску показывают, что языковые модели более эффективны, чем модели tf–idf и BM25. Тем не менее имеющихся доказательств преимущества языковых моделей перед хорошо

настроенными традиционными системами поиска на основе модели векторного пространства все еще недостаточно для того, чтобы заменять ими существующие реализации.

12.4. Расширения языковых моделей

В этом разделе мы кратко упомянем некоторые из работ, посвященных расширению фундаментального подхода, основанного на языковых моделях.

Существуют альтернативные точки зрения на использование языковых моделей для информационного поиска. Вместо исследования вероятности того, что языковая модель документа M_d сгенерирует запрос, можно ориентироваться на вероятность того, что языковая модель запроса M_q сгенерирует документ. Основная причина невысокой популярности *модели правдоподобия документа* (document likelihood model) заключается в том, что для построения языковой модели запроса имеется намного меньше текста, поэтому оценки в этой модели будут грубее, а значит, они будут сильнее зависеть от сглаживания с участием другой языковой модели. С другой стороны, нетрудно убедиться, что интегрирование обратной связи по релевантности в такую модель осуществить намного легче. Мы можем расширить запрос за счет терминов, взятых из релевантных документов, и тем самым уточнить языковую модель M_q (Zhai and Lafferty, 2001, *a*). Действительно, при правильном выборе параметров этот подход приводит к модели BIM, описанной в главе 11. Примером модели правдоподобия документа является модель релевантности, предложенная Лавренко и Крофтом (Lavrenko and Croft, 2001), в которую встроена обратная связь по псевдорелевантности. Это позволило достичь очень хороших эмпирических результатов.

Вместо того чтобы следовать в одном из направлений, мы можем построить языковые модели как по документу, так и по запросу и сравнить их друг с другом. Лафферти и Жаи (Lafferti and Zhai, 2001) реализовали все три подхода к решению поставленной задачи (рис. 12.5) и разработали общий метод минимизации риска при поиске документа. Например, для оценки риска вернуть документ d в качестве релевантного запросу q можно использовать дивергенцию Кульбака–Лейблера (Kullback–Leibler divergence) между соответствующими языковыми моделями.

$$R(d;q) = KL\left(M_d \middle\| M_q\right) = \sum_{t \in V} P\left(t \middle| M_q\right) \log \frac{P\left(t \middle| M_q\right)}{P\left(t \middle| M_d\right)} \tag{12.14}$$

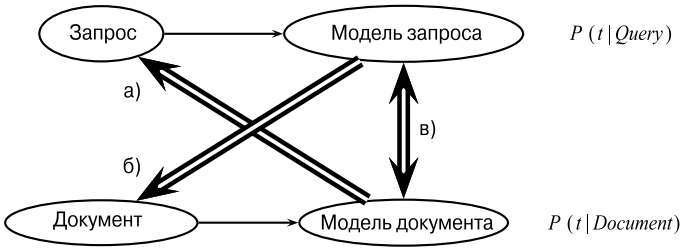


Рис. 12.5. Три подхода к разработке языковой модели: а) правдоподобие запроса, б) правдоподобие документа, в) модель сравнения

Дивергенция Кульбака–Лейблера — это асимметричная мера дивергенции, используемая в теории информации. Она позволяет оценить, насколько плохо распределение

вероятностей M_q приближает распределение вероятностей M_d (Cover and Thomas, 1991; Manning and Schutze, 1999). Лафферти и Жаи (Lafferty and Zhai, 2001) опубликовали результаты, позволяющие предположить, что модель сравнения является более эффективной, чем модели правдоподобия документа и запроса соответственно. Дивергенция Кульбака–Лейблера, используемая в качестве функции ранжирования, имеет один недостаток: невозможно сравнивать веса документов для разных запросов. Это не имеет большого значения для поиска по произвольному запросу (ad hoc retrieval), но важно для других приложений, например для отслеживания тем при обработке новостного потока. Краий и Спиттерс (Kraaij and Spitters, 2003) предложили альтернативный подход, в рамках которого схожесть моделируется с помощью нормированного логарифмического отношения правдоподобия (или, что эквивалентно, как разница между значениями перекрестной энтропии).

Основные языковые модели не учитывают альтернативные способы выражения, т.е. синонимы и любые другие различия в использовании языка между запросами и документами. Для того чтобы заполнить этот пробел, Бергер и Лафферти (Berger and Lafferty, 1999) ввели модели перевода. *Модель перевода* (translation model) позволяет генерировать слова запроса, не содержащиеся в документе, переводя их в альтернативные термины, близкие по смыслу. Это также открывает возможность для создания многоязычных систем информационного поиска. Мы предполагаем, что модель перевода можно представить с помощью распределения условной вероятности $T(\cdot|\cdot)$ по словарным терминам. Модель генерирования перевода запроса в этом подходе выглядит так.

$$P(q|M_d) = \prod_{t \in q} \sum_{v \in V} P(v|M_d) T(t|v) \tag{12.15}$$

Множитель $P(v|M_d)$ есть базовая модель языка документа, а множитель $T(t|v)$ осуществляет перевод. Разумеется, эта модель требует больше вычислений, к тому же необходимо построить модель перевода. Такая модель обычно создается с помощью отдельных ресурсов (например, традиционных тезаурусов, двуязычных словарей или словарей систем машинного перевода, полученных с помощью статистических методов), но может быть построена и с помощью коллекции документов, если существуют фрагменты текста, представляющие собой естественный парафраз или резюме других фрагментов текста. В качестве кандидатов на включение в такую коллекцию выступают документы и их заголовки либо аннотации или документы и тексты гиперссылок на эти документы.

Разработка расширенных языковых моделей остается областью активных исследований. В целом модели перевода, модели с обратной связью по релевантности и подход на основе сравнения моделей продемонстрировали повышении производительности поиска по сравнению с базовыми языковыми моделями правдоподобия запроса.

12.5. Библиография и рекомендации для дальнейшего чтения

Более подробно основы вероятностных языковых моделей и методов сглаживания описаны в книгах Маннинга и Шютце (Manning and Schütze, 1999), а также Джурафски и Мартина (Jurafsky and Martin, 2008).

Впервые применение языковых моделей к задачам информационного поиска было описано в работах Понте и Крофта (Ponte and Croft, 1998), Хиемстры (Hiemstra, 1998), Бергера и Лафферти (Berger and Lafferty, 1999) и Миллера и др. (Miller et al., 1999). Дру-

гие работы на эту тему были опубликованы в трудах конференции SIGIR следующих нескольких лет. Статьи по языковому моделированию напечатаны также в сборнике работ семинара под редакцией Крофта и Лафферти (Croft and Lafferty, 2003). Кроме того, обзор Хиемстры и Краий (Hiemstra and Kraaij, 2005) прослеживает применение языковых моделей в рамках экспериментов TREC. Жаи и Лафферти (Zhai and Lafferty, 2001*b*) осветили роль сглаживания в языковых моделях, предназначенных для информационного поиска, и провели подробное эмпирические сравнение разных методов сглаживания. Сарагоса и др. (Zaragoza et al., 2003) обосновали использование полных байесовских прогнозных распределений вместо точечных оценок MAP. Однако, несмотря на то что их метод оказался эффективнее байесовского сглаживания, он проиграл методу линейной интерполяции. Жаи и Лафферти (Zhai and Lafferty, 2002) утверждают, что модель двухэтапного сглаживания, в которой вслед за байесовским сглаживанием осуществляется линейная интерполяция, обладает еще более высокой эффективностью и является более устойчивой, чем эти виды сглаживания по отдельности. Отличительной особенностью языковых моделей является их способность учитывать разнообразную априорную информацию. Краий и др. (Kraaij et al., 2002) доказали это, продемонстрировав важность априорной информации о ссылках для высокопроизводительного поиска главных страниц в вебе. Лиу и Крофт (Liu and Croft, 2004) продемонстрировали определенное преимущество сглаживания языковой модели документа, полученного по кластеру похожих документов. Тао и др. (Tao et al., 2006) сообщили о большом превосходстве сглаживания на основе сходства документов (см. краткое обсуждение этого подхода в главе 16).

Хиемстра и Краий (Hiemstra and Kraaij, 2005) продемонстрировали на данных экспериментов TREC превосходство языковых моделей над схемой взвешивания BM25. Работы последнего времени демонстрируют некоторое преимущество моделей более высокого порядка, в частности модели более высокого порядка сглаживаются с помощью моделей более низкого порядка (Gao et al., 2004; Cao et al., 2005), хотя это преимущество остается довольно скромным. В 2004 году Спарк Джоунс (Sparck Jones, 2004) выступила с критикой обоснованности языковых моделей, но Лафферти и Жаи (Lafferty and Zhai, 2003) аргументировали, что в их основе лежит вероятностная семантика, являющаяся одновременно фундаментом классических вероятностных методов информационного поиска (см. главу 11). Для исследования языковых моделей, применяемых в информационном поиске, удобным инструментом является программное обеспечение с открытым кодом, входящее в состав пакета Lemur Toolkit (`www.lemurtoolkit.org/`).

Глава 13

Классификация текстов и наивный байесовский подход

До сих пор в этой книге в основном обсуждался *информационный поиск по произвольному запросу* (ad hoc retrieval), в ходе которого пользователи удовлетворяют свои преходящие информационные потребности, отсылая поисковой системе один или несколько запросов. Однако многие пользователи имеют постоянные информационные потребности. Например, они могут отслеживать новости, касающиеся разработки *многоядерных компьютерных микросхем* (multicore computer chips). Для удовлетворения такой потребности можно каждое утро искать новости по запросу `multicore AND computer AND chip`. В этой и следующих двух главах мы рассмотрим вопрос "Как автоматизировать решение данной повторяющейся задачи?" Для этого многие системы поддерживают *подписку пользователя на поисковый запрос (постоянные запросы)* (standing queries). Постоянный запрос похож на любой другой запрос, однако он выполняется периодически на коллекции, которая постоянно пополняется новыми документами.

Если постоянный запрос имеет вид `multicore AND computer AND chip`, то пользователь рискует пропустить много новых релевантных статей, в которых применяются другие термины, например *multicore processors*. Для того чтобы обеспечить достаточную полноту поиска, постоянные запросы со временем должны уточняться, и поэтому они постепенно становятся довольно сложными. В данном примере, применив булеву поисковую систему со стеммингом, можно в итоге прийти к примерно такому запросу: `(multicore OR multi-core) AND (chip OR processor OR microprocessor)`.

Для того чтобы описать степень общности и очертить границы предметной области, к которой относится обработка постоянных запросов, определим *задачу классификации* (classification problem): имея набор заданных *классов*, необходимо определить, к какому (или к каким) из них принадлежит заданный объект. В данном примере обработка постоянного запроса сводится к задаче классификации, в которой существуют два класса новостей: *документы о многоядерных компьютерных микросхемах* и *документы не о многоядерных компьютерных микросхемах*. Такая задача называется *бинарной классификацией* (two-class classification). Классификация на основе постоянных запросов называется также *маршрутизацией* (routing) или *фильтрацией* (filtering). Она будет рассмотрена в разделе 15.3.1.

Класс не обязательно должен быть узким, как в постоянном запросе *multicore computer chips*. Часто он охватывает довольно широкую предметную область, например *China* или *coffee*. Такие широкие классы обычно называются *темами* (topics), а задача классификации называется *классификацией текстов* (text classification), *категоризацией текстов* (text categorization), *тематической классификацией* (topic classification) или *определением тематики* (topic spotting). Пример класса *China* приведен на рис. 13.1. Постоянные запросы и темы отличаются по степени конкретизации, но методы маршрути-

зации, фильтрации и классификации текстов, по существу, совпадают. По этой причине мы включили задачи маршрутизации и фильтрации как частные примеры в более общую задачу классификации текстов, описываемую в этой и двух следующих главах.

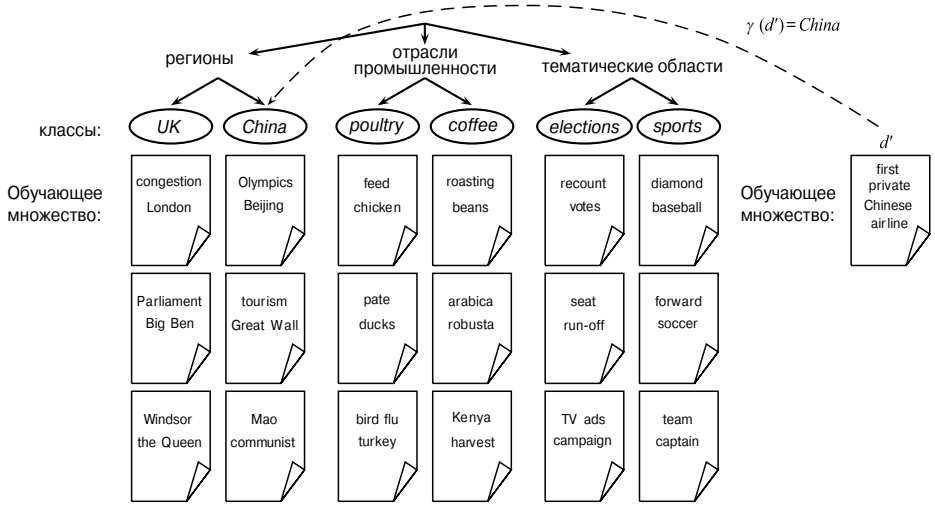


Рис. 13.1. Классы, обучающее множество и тестовое множество в задаче классификации текстов

Понятие классификации является очень общим и имеет много приложений как в области информационного поиска, так и за его пределами. Например, в области компьютерного зрения классификатор может использоваться для разделения образов по классам, таким как *landscape*, *portrait* и *neither*[1]. В этой главе мы сосредоточим внимание на примерах классификации в информационном поиске.

- Некоторые этапы предварительной обработки, необходимой для индексирования (см. главу 2): идентификация кодировки документа (ASCII, Unicode UTF-8 и т.д.), сегментация слов (является ли пробел между двумя буквами границей слова), определение истинного регистра слова, а также определение языка документа.
- Автоматическое определение веб-спама (такие страницы не включаются в индекс поисковой системы).
- Автоматическое определение содержания откровенно сексуального характера (такие материалы включаются в результаты поиска, только если пользователь отключил опцию Безопасный поиск).
- *Выявление мнений* (sentiment detection), или автоматическая классификация отзывов о фильме или товаре. Примером такого приложения является поиск, в ходе которого пользователь ищет негативные отзывы, прежде чем купить фотоаппарат, чтобы убедиться, что у него нет скрытых недостатков и проблем с качеством.

[1] Ландшафт, портрет и пр. — *Примеч. ред.*

- *Сортировка электронной почты*. Пользователь может создать несколько папок, например *Объявления*, *Счета*, *Письма от друзей* и т.д., классифицировать поступающие электронные сообщения и автоматически помещать их в соответствующую папку. Найти сообщение в упорядоченных папках сообщений намного легче, чем в очень большой папке входящих сообщений. Чаще всего эта возможность используется для фильтрации спама.
- *Тематический, или вертикальный, поиск*. Поисковые системы вертикального поиска ограничивают поиск определенной темой. Например, в ответ на запрос `computer science` система вертикального поиска по теме *China* вернет более точный и полный список китайских факультетов компьютерных наук, чем обычная поисковая система в ответ на запрос `computer science China`. Это объясняется тем, что система вертикального поиска не включает в свой индекс веб-страницы, содержащие термин `china` в ином смысле (например, посвященные китайскому фарфору), но включит в него релевантные страницы, даже если термин `China` в них явно не упоминается.
- Функция ранжирования в произвольном информационном поиске может основываться на классификаторе, как указано в разделе 15.4.

В этом списке перечислены основные приложения классификации в области информационного поиска. Большинство современных поисковых систем содержит несколько компонентов, использующих классификаторы в том или ином виде. В качестве основного примера классификации в нашей книге мы используем классификацию текстов.

Классификацию можно проводить и без компьютеров. Многие задачи классификации традиционно решались вручную. Например, библиотекари назначают книгам рубрики Библиотеки Конгресса (Library of Congress). Однако ручную классификацию трудно масштабировать. Пример запроса `multicore computer chips` иллюстрирует альтернативный подход: классификация с помощью постоянных запросов, которые можно интерпретировать как *правила*, чаще всего записанные вручную. В нашем примере правила эквивалентны логическим выражениям `(multicore OR multi-core) AND (chip OR processor OR microprocessor)`.

Правило состоит из нескольких ключевых слов, характерных для класса. Правила, составленные вручную, обладают хорошими возможностями для масштабирования, но со временем их создание и сопровождение становятся все сложнее. Опытные люди (например, эксперты в предметной области, хорошо знакомые с регулярными выражениями) могут создать набор правил, конкурирующий с автоматически сгенерированными классификаторами и даже превосходящий их, однако обычно таких людей сложно найти.

Кроме ручной классификации и правил, записанных вручную, существует третий подход к классификации текстов, а именно: классификация текстов на основе машинного обучения. Именно этот подход будет предметом изучения в следующих главах. При машинном обучении набор правил, или в более общем случае — критерий принятия решения, автоматически выводится на основе обучающей выборки. Если метод обучения является статистическим, этот подход называется *статистической классификацией текстов* (statistical text classification). Для статистической классификации текстов требуется некоторое количество хороших примеров (или обучающих документов) для каждого класса. Необходимость ручной классификации также не исключается, поскольку обучающие документы поступают от человека, который их размечает. *Разметка* (labeling) — это процесс указания класса каждого документа. Тематическая разметка доку-

ментов — намного более простая процедура, чем составление правил классификации. Практически любой человек, взглянув на документ, может сказать, говорится в нем о Китае или нет. Иногда такая разметка является частью существующего технологического процесса. Например, вы можете просматривать новости, получаемые каждое утро в ответ на постоянный запрос, и обеспечивать обратную связь по релевантности (см. главу 9), помещая релевантные статьи в специальную папку, например *multicore-processor*.

Эта глава начинается с общего введения в классификацию текстов, включая ее формальное определение (раздел 13.1). Затем мы опишем наивный байесовский подход, который является особенно простым и эффективным методом классификации (разделы 13.2–13.4). Во всех алгоритмах классификации, которые мы изучаем, документы рассматриваются как элементы пространств большой размерности. Для повышения эффективности этих алгоритмов обычно желательно понизить размерность пространства; для этого обычно используется метод *выбора признаков* (feature selection), рассматриваемый в разделе 13.5. Раздел 13.6 посвящен оценке классификации текстов. В главах 14 и 15 мы изучим два других семейства методов классификации: классификаторы в векторном пространстве и метод опорных векторов.

13.1. Классификация текстов

В задаче классификации текстов задано описание документа $d \in \mathbb{X}$, где $\mathbb{X}$ — *пространство документов* (document space), и фиксированное множество *классов* $\mathbb{C} = \{c_1, c_2, ..., c_j\}$. Классы также называются *категориями* (category) и *метками* (labels). Как правило, пространство документов $\mathbb{X}$ имеет большую размерность, а классы определяются экспертами в зависимости от приложения, как в примере *China* или *documents that talk about multicore computer chips*. Кроме того, задано *обучающее множество* $\mathbb{D}$ (training set) размеченных документов $\langle d, c \rangle$, где $\langle d, c \rangle \in \mathbb{X} \times \mathbb{C}$. Например, пара

$$\langle d, c \rangle = \langle \text{Beijing joins the World Trade Organization, } \textit{China} \rangle$$

принадлежит обучающему множеству, где d — документ, состоящий из одного предложения *Beijing joins the World Trade Organization*, *China*, а c — класс (или метка) *China*.

Используя *метод обучения* (learning method), или *алгоритм обучения* (learning algorithm), мы хотим получить классификатор, или *функцию классификации* γ (classification function), отображающую документы в классы.

$$\gamma : \mathbb{X} \to \mathbb{C} \tag{13.1}$$

Этот метод называется *обучением с учителем* (supervised learning), поскольку контролер (человек, который определяет классы и готовит обучающие документы) играет роль учителя, управляющего процессом обучения. Обозначим метод обучения с учителем буквой Γ и запишем $\Gamma(\mathbb{D}) = \gamma$. Метод обучения Γ получает на вход обучающее множество $\mathbb{D}$ и возвращает функцию классификации γ.

Классификатор γ и метод обучения Γ часто не разделяют. Когда говорят “наивный байесовский классификатор является устойчивым”, имеют в виду, что наивный байесовский *метод обучения* можно применять для решения многочисленных задач обучения, и очень маловероятно, что при этом будет построен катастрофически плохой классификатор. Однако когда мы говорим “ошибка наивного байесовского метода равна 20%”, то описываем эксперимент, в котором конкретный *классификатор* γ (созданный с помощью наивного байесовского метода обучения) имеет 20%-ную ошибку.

На рис. 13.1 показан пример классификации текстов из коллекции Reuters-RCV1, описанной в разделе 4.2. В этом примере существуют шесть классов (*UK*, *China*, ..., *sports*), каждый содержит по три обучающих документа. На рисунке показано несколько характерных слов из содержания каждого документа. Обучающее множество содержит типичные примеры для каждого класса, поэтому можно получить функцию классификации γ. Полученную функцию γ мы можем применить к *тестовому множеству* (или тестовым данным), например к новому документу *first private Chinese airline*, класс которого неизвестен. На рис. 13.1 функция классификации относит новый документ к классу $\gamma(d') = China$, т.е. дает правильный ответ.

Классы в задачах классификации текстов часто имеют интересную структуру, например иерархическую, показанную на рис. 13.1. В ней существует по две категории для региона, отрасли промышленности и тематической области. Иерархия может играть важную роль в решении задачи классификации. Дальнейшее обсуждение этой темы изложено в разделе 15.3.2, а пока мы просто предположим, что классы образуют множество, в котором подмножества никак не связаны между собой.

Определение (13.1) требует, чтобы документ был элементом только одного класса. Это не самая подходящая модель для иерархии на рис. 13.1. Например, документ об Олимпиаде - 2008 должен быть элементом двух классов: класса *China* и класса *sports*. Такой вид классификации называется *задачей многозначной классификации* (any-of problem). Мы вернемся к ней в разделе 14.5. Пока мы будем рассматривать только *задачи однозначной классификации* (one-of problem), в которых документ может быть элементом только одного класса.

Цель классификации текстов — обеспечить высокую точность на *тестовых*, или *новых*, данных, например на сообщениях о многоядерных процессорах, которые мы получим завтра. Достичь высокой точности на обучающей выборке довольно легко (например, мы можем просто запомнить метки). Однако высокая точность, достигнутая на обучающем множестве, не означает, что классификатор будет хорошо работать на новых данных. Используя обучающее множество для обучения классификатора, мы предполагаем, что обучающие и тестовые данные похожи или имеют *одно и то же распределение*. Точное определение этого понятия будет дано в разделе 14.6.

13.2. Наивная байесовская классификация текстов

Первый метод обучения с учителем, который мы рассмотрим, — *мультиномиальный наивный метод Байеса* (Naive Bayes — NB). В этом методе вероятность того, что документ d принадлежит классу c, вычисляется следующим образом.

$$P(c|d) \propto P(c) \prod_{1 \le k \le n_d} P(t_k|c) \tag{13.2}$$

Здесь $P(t_k|c)$ — условная вероятность того, что термин t_k появится в документе из класса c.[2] $P(t_k|c)$ — это наша оценка вклада термина t_k в то, что документ принадлежит классу c, а $P(c)$ — априорная вероятность того, что документ принадлежит классу c. Если термины документа не позволяют четко отделить один класс от другого, то следует выбрать тот из них, который имеет более высокую априорную вероятность. Последова-

[2] В следующем разделе мы объясним, почему вероятность P(c|d) пропорциональна (символ ~), а не равна правой части.

тельность $\langle t_1, t_2, ..., t_{n_d} \rangle$ состоит из лексем документа *d*, являющихся частью лексикона, используемого для классификации, а n_d — количество таких лексем в документе *d*. Например, последовательность $\langle t_1, t_2, ..., t_{n_d} \rangle$ для документа *Beijing and Taipei join the WTO*, состоящего из одного предложения, может иметь вид `<Beijing, Taipei, join, WTO>`, где $n_d = 4$, если удалить стоп-слово `and`.

Цель классификации текстов — найти *наилучший* класс для документа. В методе *NB* наилучшим считается наиболее вероятный класс, или класс c_{map}, имеющий *максимальную апостериорную вероятность* (maximum a posteriori — MAP).

$$c_{map} = \arg\max_{c \in \mathbb{C}} \hat{P}(c|d) = \arg\max_{c \in \mathbb{C}} \hat{P}(c) \prod_{1 \le k \le n_d} \hat{P}(t_k|c) \tag{13.3}$$

Мы пишем $\hat{P}$, а не *P*, потому что не знаем истинных значений параметров $P(c)$ и $P(t_k|c)$, а можем лишь оценить их с помощью обучающих множеств.

В равенстве (13.3) перемножается несколько условных вероятностей, по одной для каждого значения $1 \le k \le n_d$. Это может привести к потере значащих разрядов. Следовательно, лучше заменить произведение вероятностей сложением их логарифмов. Класс с наибольшим значением логарифма вероятности остается наиболее вероятным, так как $\log(xy) = \log(x) + \log(y)$, и логарифмическая функция монотонна. Следовательно, в наивном методе Байеса на самом деле требуется найти точку максимума следующей функции.

$$c_{map} = \arg\max_{c \in \mathbb{C}} \left[\log \hat{P}(c) + \sum_{1 \le k \le n_d} \log \hat{P}(t_k|c) \right] \tag{13.4}$$

Равенство (13.4) допускает простую интерпретацию. Каждый логарифм условной вероятности $\log \hat{P}(t_k|c)$ — это вес, указывающий, насколько важен термин t_k для класса *c*. Аналогично априорная вероятность $\log \hat{P}(c)$ — это вес, характеризующий относительную частоту класса *c*. Более часто встречающиеся классы, скорее всего, являются более правильными, чем редко встречающиеся. Таким образом, сумма логарифмов априорной вероятности и весов терминов отражает свидетельства в пользу того, что документ принадлежит классу, а равенство (13.4) идентифицирует класс, которому соответствует наибольшее количество свидетельств.

Пока мы ограничимся такой интуитивной интерпретацией мультиномиальной наивной байесовской модели и отложим ее формальный вывод до раздела 13.4.

Как оценить вероятности $\hat{P}(c)$ и $\hat{P}(t_k|c)$? Сначала попробуем получить оценку максимального правдоподобия (раздел 11.3.2), которая представляет собой относительную частоту и соответствует наиболее вероятной величине каждого параметра при заданных обучающих данных. Для априорных вероятностей оценка имеет следующий вид.

$$\hat{P}(c) = \frac{N_c}{N'} \tag{13.5}$$

Здесь N_c — количество документов в классе *c*, а *N* — общее количество документов.

Оценим условную вероятность $\hat{P}(t|c)$ как относительную частоту термина *t* в документах, принадлежащих классу *c*.

$$\hat{P}(t|c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}} \tag{13.6}$$

Здесь T_{ct} — количество появлений термина t в обучающих документах из класса c с учетом многократных появлений термина в документе. Эта оценка основана на *предположении о позиционной независимости* (positional independence assumption), которое будет анализироваться в следующем разделе. T_{ct} — это количество появлений термина во всех k координатах в документах из обучающего множества. Таким образом, мы не вычисляем разные оценки для разных координат, и, например, если слово дважды встречается в документе в координатах k_1 и k_2, то $\hat{P}\left(t_{k_1}|c\right)=\hat{P}\left(t_{k_2}|c\right)$.

С оценкой максимального правдоподобия связана одна проблема: если пара "термин–класс" не встречается в обучающих данных, то оценка MLE равна нулю. Например, если термин `WTO` в обучающих данных встречается только в документах класса *China*, то оценки MLE для других классов, например класса *UK*, равны нулю.

$$\hat{P}\left(WTO|UK\right)=0$$

Теперь условная вероятность принадлежности документа *Britain is a member of the WTO*, состоящего из одного предложения, классу *UK* будет равна нулю, поскольку в равенстве (13.2) мы перемножаем условные вероятности для всех терминов. Очевидно, что модель должна присваивать классу *UK* высокую вероятность, поскольку в предложении встречается термин *Britain*. Тем не менее нельзя просто отбросить нулевую вероятность для термина `WTO` независимо от того, насколько много есть свидетельств в пользу класса *UK*, обеспеченных другими признаками. Эта оценка равна нулю из-за *редкости* термина. Обучающие данные никогда не бывают большими настолько, чтобы частота редких терминов оценивалась адекватно, как, например, частота термина `WTO` в документах класса *UK*.

Для того чтобы избавиться от нуля, мы используем *сглаживание Лапласа* (Laplace smoothing), просто добавляя единицу к каждой частоте.

$$\hat{P}\left(t|c\right)=\frac{T_{ct}+1}{\sum_{t'\in V}\left(T_{ct'}+1\right)}=\frac{T_{ct}+1}{\sum_{t'\in V}T_{ct'}+B} \tag{13.7}$$

Здесь $B=|V|$ — количество терминов в лексиконе коллекции. Сглаживание Лапласа можно интерпретировать как априорное равномерное распределение (каждый термин встречается в каждом классе по одному разу), которое затем уточняется на основе поступающих обучающих данных. Отметим, что это — априорная вероятность появления *термина*, а не *класса*, которая оценивается формулой (13.5) на уровне документа.

Введем теперь все элементы, необходимые для обучения и применения наивного байесовского классификатора. Полный алгоритм приведен на рис. 13.2.

Пример 13.1. Основываясь на примерах, приведенных в табл. 13.1, и указанных ниже параметрах, требуется классифицировать тестовый документ.

$\hat{P}(c)=3/4,\ \hat{P}(\overline{c})=1/4,$

$\hat{P}$ (Chinese|c) = (5 + 1)/(8 + 6) = 6/14 = 3/7,

$\hat{P}$ (Tokio|c) = $\hat{P}$ (Japan|c) = (0 + 1)/(8 + 6) = 1/14,

$\hat{P}$ (Chinese|$\overline{c}$) = (1 + 1)/(3 + 6) = 2/9,

$\hat{P}$ (Tokio|$\overline{c}$) = $\hat{P}$ (Japan|$\overline{c}$) = (1 + 1)/(3 + 6) = 2/9.

Знаменатели равны (8 + 6) и (3 + 6), поскольку длины текстов $text_c$ и $text_{\overline{c}}$ равны 8 и 3 соответственно, а константа B в равенстве (13.7) равна 6, так как лексикон состоит из шести терминов. Таким образом,

$$\hat{P}(c|d_5) \propto 3/4 \cdot (3/7)^3 \cdot 1/14 \cdot 1/14 \approx 0,0003,$$

$$\hat{P}(c|d_5) \propto 1/4 \cdot (2/9)^3 \cdot 2/9 \cdot 2/9 \approx 0,0001.$$

Следовательно, классификатор отнесет тестовый документ к классу c = *China*. Причина такой классификации заключается в том, что три положительных индикатора вхождения термина Chinese в документ d_5 перевешивают негативные индикаторы терминов Japan и Tokio.

```
TrainMultinomialNB(ℂ, 𝔻)
1   V ← ExtractVocabulary(𝔻)
2   N ← CountDocs(𝔻)
3   for each c∈ ℂ
4   do N_c ← CountDocsInClass(𝔻, c)
5       prior[c] ← N_c/N
6       text_c ← ConcatenateTextOfAllDocsInClass(𝔻, c)
7       for each t∈ V
8       do T_ct ← CountTokensOfTerm(text_c, t)
9       for each t∈ V
10      do condprob[t][c] ← (T_ct + 1) / Σ_t' (T_ct' + 1)
11  return V, prior, condprob
ApplyMultinomialNB(ℂ, V, prior, condprob, d)
1   W ← ExtractTokensFromDoc(V, d)
2   for each c∈ ℂ
3   do score[c] ← log prior[c]
4      for each t ∈ W
5      do score[c]+=log condprob[t][c]
6   return arg max_c∈ℂ score[c]
```

Рис. 13.2. Наивный байесовский алгоритм (мультиномиальная модель): обучение и тестирование

Таблица 13.1. Данные для оценки параметров

	docID	*Слова в документе*	*c = China?*
Обучающее множество	1	Chinese Beijing Chinese	Да
	2	Chinese Chinese Shanghai	Да
	3	Chinese Makao	Да
	4	Tokio Japan Chinese	Нет
Тестовое множество	5	Chinese Chinese Chinese Tokio Japan	?

Чему равна временн*ая* сложность алгоритма NB? Сложность вычисления параметров равна $\Theta(|\mathbb{C}||V|)$, поскольку множество параметров состоит из $|\mathbb{C}||V|$ условных вероятностей и $|\mathbb{C}|$ априорных. Для вычисления этих параметров необходима определенная предварительная работа (извлечение термина из лексикона, подсчет и т.д.), которую можно выполнить за один проход по обучающим данным. Следовательно, временн*ая* сложность этого компонента равна $\Theta(|\mathbb{D}|L_{ave})$, где $|\mathbb{D}|$ — количество документов, а L_{ave} — средняя длина документа.

Здесь мы использовали обозначение $\Theta(|\mathbb{D}|L_{ave})$ как вариант обозначения $\Theta(T)$, где T — длина обучающей коллекции. Это нестандартное решение, поскольку величины $\Theta(\cdot)$ не определяются для средних величин. Мы предпочли выразить временн*ую* сложность через величины $\mathbb{D}$ и L_{ave}, поскольку они являются основными статистическими показателями, характеризующими обучающие коллекции.

Временн*ая* сложность алгоритма `ApplyMultinomialNB`, представленного на рис. 13.2, равна $\Theta(|\mathbb{C}|L_a)$. Величины L_a и M_a являются количеством лексем и типов в тестовом документе соответственно. Алгоритм `ApplyMultinomialNB` можно модифицировать так, чтобы его временн*ая* сложность стала равной $\Theta(L_a + |\mathbb{C}|M_a)$ (упражнение 13.8). В заключение отметим, что если длина тестовых документов ограничена, то $\Theta(L_a + |\mathbb{C}|M_a) = \Theta(|\mathbb{C}|M_a)$, поскольку при фиксированной константе b выполняется неравенство $L_a < b|C|M_a$. [3]

Оценки временн*ой* сложности приведены в табл. 13.2. В целом $|\mathbb{C}||V| < |\mathbb{D}|L_{ave}$, поэтому сложность обучения и тестирования линейно зависит от времени, необходимого для сканирования данных. Поскольку данные должны просматриваться минимум один раз, метод NB можно считать оптимальным по временн*ой* сложности. Такая производительность является одной из причин популярности метода NB для классификации текстов.

Таблица 13.2. Данные для оценки параметров

Алгоритм	*Временная сложность*
Обучение	$\Theta(\|\mathbb{D}\|L_{ave} + \|\mathbb{C}\|\|V\|)$
Тестирование	$\Theta(L_a + \|\mathbb{C}\|M_a) = \Theta(\|\mathbb{C}\|M_a)$

13.2.1. Связь с мультиномиальной униграммной языковой моделью

С формальной точки зрения мультиномиальная модель NB идентична мультиномиальной униграммной языковой модели (раздел 12.2.1). В частности, равенство (13.2) является частным случаем равенства (12.12), которое мы повторяем здесь при $\lambda = 1$.

$$P(d|q) \propto P(d)\prod_{t\in q} P(t|M_d) \tag{13.8}$$

Документ d в задаче классификации текстов (равенство (13.2)) играет роль запроса в языковой модели (равенство (13.8)), а класс c — роль документа. Равенство (13.8) используется для ранжирования документов в соответствии с вероятностью того, что они являются релевантными запросу q. В задаче наивной байесовской классификации нас обычно интересует только класс с наибольшим рангом.

[3] Здесь мы предполагаем, что длина тестовых документов ограничена. Для чрезвычайно длинных тестовых документов величина L_a превышает величину $b|C|M_a$.

Кроме того, в разделе 12.2.2 мы вычисляли оценки MLE и столкнулись с проблемой нулевых вероятностей, возникающих из-за разреженных данных, но для разрешения этой проблемы вместо сглаживания Лапласа мы использовали смесь двух распределений. Сглаживание Лапласа (добавление единицы) тесно связано со сглаживанием путем добавления 1/2, описанным в разделе 11.3.4.

? ***Упражнение 13.1.*** Почему для большинства коллекций текстов выполняется неравенство $|\mathbb{C}||V| < |\mathbb{D}|L_{ave}$?

13.3. Модель Бернулли

Существует два способа построения наивной байесовской модели. Мультиномиальная модель была описана выше. Она генерирует по одному термину из лексикона в каждой координате внутри документа. Эту порождающую модель мы обсудим подробнее в разделе 13.4 (см. также раздел 12.1).

Альтернативой мультиномиальной модели является *многомерная модель Бернулли* (multivariate Bernoulli model), или просто *модель Бернулли*. Она эквивалентна бинарной модели независимости (раздел 11.3), генерирующей индикатор для каждого термина лексикона: 1, если термин присутствует в документе, и 0, если отсутствует. На рис. 13.3 показаны алгоритмы обучения и тестирования для модели Бернулли. Временная сложность модели Бернулли такая же, как и мультиномиальной модели.

```
TrainBernoulliNB(ℂ, 𝔻)
1   V ← ExtractVocabulary(𝔻)
2   N ← CountDocs(𝔻)
3   for each c ∈ ℂ
4   do N_c ← CountDocsInClass(𝔻, c)
5      prior[c] ← N_c/N
6      for each t ∈ V
7      do N_ct ← CountDocsInClassContainingTerm(𝔻, c, t)
8         condprob[t][c] ← (N_ct + 1)/(N_c + 2)
9   return V, prior, condprob
ApplyBernoulliNB(ℂ, V, prior, condprob, d)
1   V_d ← ExtractTermsFromDoc(V, d)
2   for each c ∈ ℂ
3   do score[c] ← log prior[c]
4      for each t ∈ V
5      do if t ∈ V
6            then score[c] += log condprob[t][c]
7            else score[c] += log (1 – condprob[t][c])
8   return arg max_{c∈ℂ} score[c]
```

Рис. 13.3. Наивный байесовский алгоритм (модель Бернулли): обучение и тестирование. В строке 8 (вверху) применено сглаживание добавлением единицы, аналогичное формуле (13.7), где B = 2

Разные порождающие модели подразумевают разные стратегии оценки и разные правила классификации. В модели Бернулли вероятность $\hat{P}(t|c)$ оценивается как *доля документов из класса c*, содержащих термин *t* (рис. 13.3, алгоритм `TrainBernoulliNB`, строка 8). В противоположность ему в мультиномиальной модели вероятность $\hat{P}(t|c)$ оценивается как *доля лексем*, или *доля словоупотреблений* в документах класса *c*, содержащих термин *t* (равенство (13.7)). При классификации тестового документа на основе модели Бернулли используется бинарная информация о появлении термина, которая игнорирует количество вхождений этого термина, в то время как мультиномиальная модель отслеживает многократные употребления термина в документе. В результате при классификации длинных документов модель Бернулли, как правило, допускает много ошибок. Например, она может отнести к классу *Chine* целую книгу из-за единственного упоминания термина *China*.

Эти модели различаются также тем, как используются термины, не появляющиеся в документе. В мультиномиальной модели эта информация никак не влияет на решение, а в модели Бернулли вероятность отсутствия термина при вычислении вероятности $P(c|d)$ учитывается (рис. 13.3, `ApplyBernoulliNB`, строка 7). Это объясняется тем, что только модели Бернулли учитывают информацию об отсутствии термина явным образом.

Пример 13.2. Применив модель Бернулли к данным, приведенным в табл. 13.1, мы получили оценки $\hat{P}(c)$ =3/4 и $\hat{P}(\overline{c})$ =1/4. Условные вероятности принимают следующие значения.

$\hat{P}$ (Chinese|*c*) = (3 + 1)/(3 + 2) = 4/5,

$\hat{P}$ (Japan|*c*) = $\hat{P}$ (Tokio|*c*) = (0 + 1)/(3 + 2) = 1/5,

$\hat{P}$ (Beijing|*c*) = $\hat{P}$ (Macao|*c*) = $\hat{P}$ (Shanghai|*c*) = (1 + 1)/(3 + 2) = 2/5,

$\hat{P}$ (Chinese|$\overline{c}$) = (1 + 1)/(1 + 2) = 2/3,

$\hat{P}$ (Japan|$\overline{c}$) = $\hat{P}$ (Tokyo|$\overline{c}$) = (1 + 1)/(1 + 2) = 2/3,

$\hat{P}$ (Beijing|$\overline{c}$) = $\hat{P}$ (Macao|$\overline{c}$) = $\hat{P}$ (Shanghai|$\overline{c}$) = (0 + 1)/(1 + 2) = 1/3.

Знаменатели равны (3 + 2) и (1 + 2), поскольку в классе *c* существуют три документа, в классе $\overline{c}$ — один документ, а константа *B* в равенстве (13.7) равна двум: для каждого термина существуют два варианта — он или входит в документ, или нет.

Вероятности принадлежности тестового документа к каждому из классов.

$$\hat{P}(c|d_5) \propto \hat{P}(c)\cdot\hat{P}(\text{Chinese}|c)\cdot\hat{P}(\text{Japan}|c)\cdot\hat{P}(\text{Tokyo}|c)\cdot$$
$$\cdot(1-\hat{P}(\text{Beijing}|c))\cdot(1-\hat{P}(\text{Shanghai}|c))\cdot(1-\hat{P}(\text{Makao}|c)) =$$
$$= 3/4\cdot 4/5\cdot 1/5\cdot 1/5\cdot(1-2/5)\cdot(1-2/5)\cdot(1-2/5)\approx 0{,}005.$$

Аналогично

$$\hat{P}(\overline{c}|d_5) \propto 1/4\cdot 2/3\cdot 2/3\cdot 2/3\cdot(1-1/3)\cdot(1-1/3)\cdot(1-1/3)\approx 0{,}022.$$

Следовательно, классификатор отнесет тестовый документ к классу $\overline{c}$ = *not-China*. Если учитывать бинарный индикатор вхождения термина в документ, а не его частоту, то индикаторы терминов Japan и Tokyo являются индикаторами класса $\overline{c}$ (2/3 > 1/5), а условные вероятности термина Chinese для классов *c* и $\overline{c}$ недостаточно сильно отличаются друг от друга (4/5 от 2/3), чтобы повлиять на результат классификации.

13.4. Свойства наивной байесовской модели

Для того чтобы лучше понять эти две модели, а также предположения, на которых они основаны, вернемся назад и проверим, как мы вывели правила классификации в главах 11 и 12. Напомним, что решение о принадлежности к классу принимается путем определения класса с максимальной апостериорной вероятностью (раздел 11.3.2).

$$c_{map} = \arg\max_{c \in \mathbb{C}} \hat{P}(c|d) = \tag{13.9}$$

$$= \arg\max_{c \in \mathbb{C}} \frac{P(d|c)P(c)}{P(d)} =$$

$$= \arg\max_{c \in \mathbb{C}} P(d|c)P(c). \tag{13.10}$$

Применяя к выражению (13.9) правило Байеса (11.4), можно отбросить знаменатель, поскольку он является постоянным для всех классов и не влияет на величину argmax.

Выражение (13.10) можно интерпретировать как описание порождающего процесса, характерного для байесовской классификации текстов. Для того чтобы сгенерировать документ, мы сначала выбираем класс c с вероятностью $P(c)$ (верхние узлы на рис. 13.4 и 13.5). Эти две модели отличаются способом формализации второго этапа, т.е. порождения документа по заданному классу в соответствии с условным распределением $P(d|c)$.

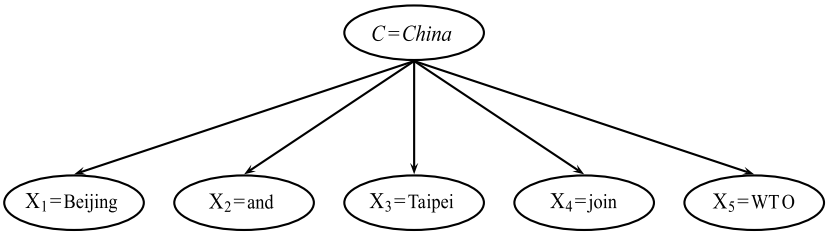


Рис. 13.4. Мультиномиальная наивная байесовская модель

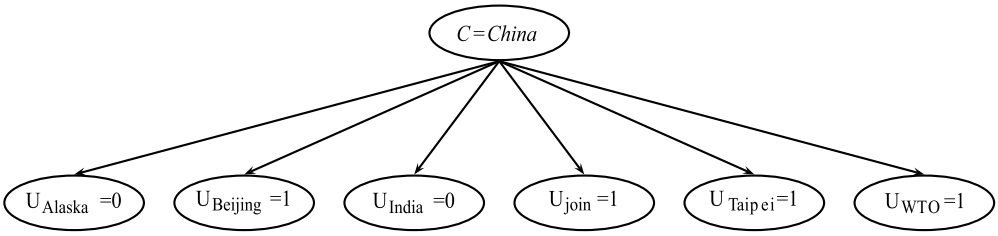


Рис. 13.5. Наивная байесовская модель Бернулли

Мультиномиальная модель $P(d|c) = P(\langle t_1,...,t_k,...,t_{n_d} \rangle | c)$ (13.11)

Бернуллиевская модель $P(d|c) = P(\langle e_1,...,e_i...,e_M \rangle | c)$ (13.12)

Здесь $\langle t_1,...,t_k,...,t_{n_d} \rangle$ — последовательность терминов, появляющихся в документе d (за исключением терминов, исключенных из лексикона), а $\langle e_1,...,e_i...,e_M \rangle$ — бинарный вектор, состоящий из M индикаторов присутствия каждого термина в документе d.

Теперь читателям должно бы понятнее, почему в равенстве (13.1), описывая задачу классификации текстов, мы ввели пространство документов $\mathbb{X}$. Критический шаг в решении задачи классификации текстов — выбор представления документа. Например, последовательности $\langle t_1,...,t_k,...,t_{n_d}\rangle$ и $\langle e_1,...,e_i...,e_M\rangle$ являются различными представлениями документа. В первом варианте пространство $\mathbb{X}$ является множеством всех последовательностей терминов (или, точнее, последовательностей лексем терминов). Во втором варианте пространство $\mathbb{X}$ представляет собой множество $[0,1]^M$.

Равенства (13.11) и (13.12) невозможно непосредственно применить для классификации текстов. Например, используя модель Бернулли, мы были бы вынуждены оценить $2^M|\mathbb{C}|$ разных параметров, по одному для каждого возможного из M сочетаний значений e_i и класса. Количество параметров в мультиномиальной модели примерно такого же порядка. [4] Поскольку количество параметров в обоих случаях очень велико, их оценка становится практически неразрешимой проблемой.

Для того чтобы уменьшить количество параметров, сформулируем *предположение об условной независимости* (conditional independence assumption). Будем считать, что, если класс задан, значения атрибутов не зависят друг от друга.

$$\textbf{Мультиномиальная модель}\quad P(d|c)=P\left(\langle t_1,...,t_{n_d}\rangle|c\right)=\prod_{1\le k\le n_d}P(X_k=t_k|c) \qquad (13.13)$$

$$\textbf{Бернуллиевская модель}\quad P(d|c)=P\left(\langle e_1,...,e_M\rangle|c\right)=\prod_{1\le i\le M}P(U_i=e_i|c) \qquad (13.14)$$

Здесь мы ввели две случайные величины, чтобы явным образом различать разные порождающие модели. Переменная X_k — это случайная величина для координаты k в документе, принимающая в качестве значений термины из лексикона, а $P(X_k = t|c)$ — вероятность того, что в документе из класса c термин t появится в координате k. Переменная U_i — это случайная величина для термина i из лексикона коллекции, принимающая значения 0 (термин отсутствует) и 1 (термин присутствует). Вероятность $P(U_i = 1|c)$ — это вероятность того, что в документе из класса c появится термин t_i — в любой координате и, возможно, многократно.

Предположение об условной независимости проиллюстрировано на рис. 13.4 и 13.5. На этих рисунках показано, что класс *China* порождает значения для каждого из пяти атрибутов (в мультиномиальной модели) или шести бинарных атрибутов термина (в модели Бернулли) с определенной вероятностью, не зависящей от значений других атрибутов. Иначе говоря, вероятность того, что документ в классе *China* содержит термин `Taipei`, никак не зависит от того, содержится ли в нем термин `Beijing`.

На практике предположение об условной независимости для текстовых данных не выполняется. Термины *зависят* друг от друга. Однако, как будет скоро показано, несмотря на это наивные байесовские модели работают достаточно хорошо.

Даже если предположение об условной независимости выполняется, то при условии, что каждая координата k в документе имеет собственное распределение вероятностей, в мультиномиальной модели по-прежнему остается слишком много параметров. Координата термина в документе сама по себе не содержит информации о классе. Несмотря на разницу между документами *China sues France* и *France sues China*, появление термина *China* в первой, а не в третьей координате в документе ничего не дает для классифи-

[4] Фактически, если длины документов не ограничены, количество параметров в мультиномиальной модели является бесконечным.

кации в рамках наивной байесовской модели, поскольку в ней все термины рассматриваются изолированно друг от друга. Предположение об условной независимости вынуждает нас обрабатывать термины именно так.

Кроме того, если бы мы предположили разные распределения терминов для каждой координаты k, то мы должны были бы оценивать разные множества параметров для каждой координаты k. Вероятность того, что термин `bean` появится как первый термин в документе *coffie*, должна отличаться от вероятности того, что он окажется вторым, и т.д. Это снова порождает проблемы с оценками на разреженных данных.

По этим причинам мы делаем второе предположение о независимости в мультиномиальной модели — *предположение о позиционной независимости* (positional independence): условные вероятности появления термина одинаковы независимо от его координаты в документе, т.е.

$$P\left(X_{k_1} = t \middle| c\right) = P\left(X_{k_2} = t \middle| c\right)$$

для всех координат k_1 и k_2, терминов t и классов c. Таким образом, в модели возникает единственное распределение терминов для всех координат k_i. Будем обозначать это распределение буквой X. [5] Предположение о позиционной независимости эквивалентно модели "мешка слов", описанной в контексте поиска по произвольному запросу в главе 6.

Приняв предположения об условной и позиционной независимости, задачу можно свести к оценке всего $\Theta(M|\mathbb{C}|)$ параметров $P(t_k|c)$ в мультиномиальной модели или $P(e_i|c)$ в модели Бернулли по одному для каждой комбинации "термин–класс", а не к оценке огромного количества параметров, пропорционального степени размера лексикона M.

Подведем итог. В мультиномиальной модели (см. рис. 13.4) мы генерируем документ, сначала выбирая класс $C = c$ с вероятностью $P(c)$, где C — случайная величина, принимающая значения из множества $\mathbb{C}$. Затем мы генерируем термин t_k в координате k с вероятностью $P(X_k = t_k|c)$ для каждой из n_d координат документа. Все величины X_k имеют одно и то же распределение над терминами в фиксированном классе c. В примере, продемонстрированном на рис. 13.4, мы показали схему порождения последовательности $<t_1, t_2, t_3, t_4, t_5>$ = <`Beijing`, `and`, `Taipei`, `join`, `WTO`>, соответствующей документу, содержащему одно предложение — *Beijing and Taipei join WTO*.

Для того чтобы модель порождения документа стала полностью определенной, необходимо определить распределение вероятностей $P(n_d|c)$ по длинам. Без этого мы получим модель порождения лексем, а не документа.

В модели Бернулли (см. рис. 13.5) мы генерируем сначала документ, выбирая класс $C = c$ с вероятностью $P(c)$, а затем — бинарный индикатор e_i для каждого термина t_i из лексикона ($1 \le i \le M$). В примере, продемонстрированном на рис. 13.5, показан процесс порождения последовательности бинарных индикаторов $<e_1, e_2, e_3, e_4, e_5, e_6>$ = = <0, 1, 0, 1, 1, 1, 1>, соответствующей, как и ранее, документу, содержащему предложение *Beijing and Taipei join WTO*, в предположении, что `and` является стоп-словом.

Результаты сравнения этих двух моделей приведены в табл. 13.3, включая формулы для вычисления оценок и решающих правил.

---

[5] Это нестандартная терминология. Случайная величина X является категориальной, а не мультиномиальной, поэтому соответствующую наивную байесовскую модель следовало бы назвать моделью последовательности (sequence model). Мы отождествили модель последовательности и мультиномиальную модель из раздела 13.4.1 потому, что с вычислительной точки зрения они идентичны.

Таблица 13.3. Сравнение мультиномиальной модели и модели Бернулли

	Мультиномиальная модель	*Модель Бернулли*
Событие	Порождение лексемы	Порождение документа
Случайные переменные	X = t тогда и только тогда, когда термин t встретился в заданной координате	U_t = 1 тогда и только тогда, когда термин t встретился в документе
Представление документа	$d = \langle t_1, t_2, ..., t_k, ..., t_{n_d} \rangle, t_k \in V$	$d = \langle e_1, e_2, ..., e_i, ..., e_M \rangle, e_i \in [0,1]$
Оценка параметра	$\hat{P}(X = t \mid c)$	$\hat{P}(U_i = e \mid c)$
Решающее правило: максимум	$\hat{P}(c) \prod_{1 \le k \le n_d} \hat{P}(X = t_k \mid c)$	$\hat{P}(c) \prod_{t_i \in V} \hat{P}(U_i = e_i \mid c)$
Термин встречается в документе несколько раз	Учитывается	Игнорируется
Длина документа	Справляется с длинными документами	Лучше всего работает с короткими документами
Количество признаков	Справляется с большим количеством признаков	Лучше работает с небольшим количеством признаков
Оценка термина *the*	$\hat{P}(X = \text{the} \mid c) \approx 0{,}05$	$\hat{P}(U_{\text{the}} = 1 \mid c) \approx 1{,}0$

Рассматриваемая модель называется наивной байесовской, поскольку сделанные нами предположения о независимости являются действительно наивными для модели естественного языка. Предположение об условной независимости утверждает, что для заданного класса признаки являются независимыми друг от друга. Это предположение редко выполняется для терминов в документах. Во многих случаях верно противоположное утверждение. Примерами сильно зависимых терминов являются пары `hong` и `kong`, `london` и `english`, указанные на рис. 13.7. Кроме того, мультиномиальная модель основана на предположении о позиционной независимости. Модель Бернулли игнорирует координаты в документе, поскольку в ней учитывается только присутствие или отсутствие термина. Эта модель "мешка слов" отбрасывает всю информацию, которую порождает порядок слов в предложениях естественного языка. Чем же объяснить эффективность наивной байесовской модели, если модель естественного языка так сильно упрощена?

Ответ заключается в том, что, несмотря на низкое качество *оценок вероятности* в наивной байесовской модели, *решения о классификации* удивительно точны. Рассмотрим документ d с истинными вероятностями $P(c_1|d) = 0{,}6$ и $P(c_2|d) = 0{,}4$, приведенными в табл. 13.4. Предположим, что в документе d много терминов, свидетельствующих в пользу класса c_1, и много терминов, свидетельствующих против принадлежности классу c_2. Таким образом, в мультиномиальной модели (13.13) оценка $\hat{P}(c_1) \prod_{1 \le k \le n_d} \hat{P}(t_k \mid c_1)$ будет намного больше, чем $\hat{P}(c_2) \prod_{1 \le k \le n_d} \hat{P}(t_k \mid c_2)$ (в табл. 13.4 0,00099 > 0,00001). После деления на 0,001 (для нормировки вероятности) приходим к выводу, что одна оценка близка к 1,0, а вторая — к 0,0. Обобщим сказанное: результирующий класс в наивной байесовской модели обычно имеет намного бо́льшую вероятность, чем другие классы, хотя оценки и сильно смещены от истинных вероятностей. Однако окончательное решение основано на

том, какой из классов получает наибольшую оценку вероятности. При этом совершенно не важно, насколько точными являются оценки. Несмотря на неточность, наивные байесовские оценки приписывают классу c_1 большую вероятность и, следовательно, относят документ d к правильному классу. *Правильная оценка означает точный прогноз, но точный прогноз не означает правильную оценку.* Наивные байесовские классификаторы опираются на неточные оценки, но позволяют достичь высокой точности классификации.

Таблица 13.4. Правильная оценка означает правильный прогноз, но правильный прогноз не всегда означает правильную оценку

	c_1	c_2	*Выбранный класс*
Истинная вероятность $P(c\|d)$	0,6	0,4	c_1
$\hat{P}(c)\prod_{1\le k\le n_d}\hat{P}(t_k\|c)$ (формула (13))	0,00099	0,00001	
Оценка NB $\hat{P}(c\|d)$	0,99	0,01	c_1

Даже если наивный байесовский метод не обладает наибольшей точностью классификации текстов, он имеет множество преимуществ. Если существует много одинаково важных признаков, совместно влияющих на окончательный вывод, то наивный байесовский метод оказывается лучше других. Кроме того, иногда он довольно устойчив к шумовым признакам (рассматриваются в следующем разделе) и *дрейфу понятий* (concept drift), т.е. к постепенному изменению понятия, лежащего в основе класса, например в случае класса *US president* — от Билла Клинтона к Джорджу Бушу (раздел 13.7). Классификаторы, основанные на методе kNN (раздел 14.3), можно точно настроить на особенности конкретного периода времени. Однако эти настройки могут оказаться неверными, когда с течением времени понятия слегка изменятся.

Модель Бернулли особенно устойчива к дрейфу понятий. Как будет показано на рис. 13.8, она может достигать приемлемой точности, используя меньше дюжины терминов. Изменение наиболее важных индикаторов класса маловероятно. Следовательно, вероятность того, что модель, опирающаяся только на такие индикаторы, будет сохранять определенный уровень точности при смещении понятий, достаточно высока.

Основное преимущество наивного байесовского метода заключается в его высокой производительности: обучение и классификацию можно провести за один проход. Поскольку этот метод имеет как высокую производительность, так и приемлемую правильность, его часто используют как основу при классификации текстов. Его выбирают, в первую очередь, в ситуациях, когда 1) “выжимание” пары дополнительных процентов правильности не стоит затраченного труда, 2) имеется большой объем обучающих данных, благодаря чему более высокую правильность можно получить за счет объема, а не за счет более качественного классификатора на небольшом объеме данных и 3) нужно использовать устойчивость метода к дрейфу понятий.

В нашей книге мы рассматриваем наивный байесовский метод применительно к классификации текстов. Предположение о независимости для текстов не выполняется. Однако можно показать, что этот метод является *оптимальным классификатором* (в смысле минимального уровня ошибок, наблюдаемого на новых данных) для данных, относительно которых предположение о независимости выполняется.

13.4.1. Вариант мультиномиальной модели

Альтернативная формализация мультиномиальной модели предусматривает представление каждого документа d как M-мерного вектора частот $\langle tf_{t_1,d},...,tf_{t_M,d}\rangle$, где $tf_{t_M,d}$ — частота термина t_i в документе d. В таком случае вероятность $P(d|c)$ вычисляется следующим образом (ср. с формулой (12.8)).

$$P(d|c) = P\left(\langle tf_{t_1,d},...,tf_{t_M,d}\rangle \middle| c\right) \propto \prod_{1\le i\le M} P\left(X = t_i|c\right)^{tf_{t_i,d}} \tag{13.15}$$

Обратите внимание на то, что мы опустили мультиномиальный множитель.

Формула (13.15) эквивалентна модели последовательности (13.2), поскольку для терминов, не встречающихся в документе d ($tf_{t_i,d} = 0$), выполняется равенство $P\left(X = t_i|c\right)^{tf_{t_i,d}} = 1$, а термины, встречающиеся в документе ($tf_{t_i,d} \ge 1$), порождают один и тот же множитель $tf_{t_i,d}$ как в формуле (13.2), так и в формуле (13.15).

? ***Упражнение 13.2*** **[*].** Какие из документов, приведенных в табл. 13.5, имеют представление, совпадающее и не совпадающее с представлением “мешка слов”, 1) в модели Бернулли и 2) в мультиномиальной модели? Если различия есть, опишите их.

Таблица 13.5. Набор документов, для которых выполнение предположений о независимости проблематично

1	He moved from London, Ontario, to London, England
2	He moved from London, England, to London, Ontario
3	He moved from England to London, Ontario

Упражнение 13.3. Факт, что термин появляется в документе в координате k, не несет полезной информации. Именно это позволяет выдвинуть предположения о независимости терминов. Найдите исключения из этого правила. Рассмотрите шаблонные документы с фиксированной структурой.

Упражнение 13.4. В табл. 13.3 приведены оценки для слова the, полученные в модели Бернулли и в мультиномиальной модели. Объясните разницу между ними.

13.5. Выбор признаков

Выбор признаков (feature selection) — это выбор подмножества терминов, встречающихся в обучающем множестве, и использование только этого набора признаков при классификации текстов. Выбор признаков преследует две основные цели. Во-первых, это повышает эффективность обучения и применения классификатора за счет уменьшения размера лексикона. Данное обстоятельство особенно важно для классификаторов, обучение которых, в отличие от наивного байесовского метода, очень затратно. Во-вторых, выбор признаков часто повышает точность классификации благодаря исключению шумовых признаков. *Шумовой признак* (noise feature) — это признак, при добавлении которого к представлению документа ошибка классификации на новых данных возрастает. Допустим, что редко встречающийся термин, например `arachnocentric`, не несет

информации о классе, скажем, *China*, но все экземпляры слова `arachnocentric` содержатся в документах из класса *China*, включенных в обучающее множество. Тогда метод обучения может создать классификатор, который по ошибке отнесет все документы, содержащие слово `arachnocentric`, к классу *China*. Такое неверное обобщение на основе случайного свойства обучающего множества называется *переобучением* (overfitting).

Выбор признаков можно рассматривать как метод замены сложного классификатора (использующего все признаки) более простым (использующим подмножество признаков). На первый взгляд, кажется нелогичным, что более слабый классификатор имеет преимущество в статистической классификации текстов, но при обсуждении компромисса между смещением и дисперсией в разделе 14.6 мы увидим, что на ограниченных обучающих множествах более слабые модели часто являются более предпочтительными.

Основной алгоритм выбора признаков приведен на рис. 13.6. Для заданного класса c мы вычисляем меру полезности $A(t, c)$ каждого термина из лексикона и выбираем k терминов, имеющих наибольшие значения $A(t, c)$. Все другие термины отбрасываются и в классификации не участвуют. В этом разделе рассматриваются три меры полезности: *взаимная информация* (mutual information) $A(t, c) = I(U_t; C_c)$, критерий χ^2, в котором $A(t, c) = X^2(t, c)$, и частота $A(t, c) = N(t, c)$.

```
SelectFeatures(𝔻, c, k)
1  V ← ExtractVocabulary(𝔻)
2  L ← []
3  for each t ∈ V
4  do A(t, c) ← ComputeFeatureUtility(𝔻, t, c)
5     Append(L, <A(t, c), t>)
6  return FeaturesWithLargestValues(L, k)
```

Рис. 13.6. Основной алгоритм выбора k наилучших признаков

Из двух наивных байесовских моделей модель Бернулли особенно чувствительна к шумовым признакам. Для нее в какой-то форме необходим выбор признаков, иначе правильность классификации снижается.

В этом разделе мы рассмотрим выбор признаков для бинарной классификации, например *China* и *not-China*. Оптимизация систем для задач с бо́льшим количеством классов кратко обсуждается в разделе 13.5.5.

13.5.1. Взаимная информация

Распространенный метод выбора признаков основан на подсчете величины $A(t, c)$ — ожидаемой *взаимной информации* о термине t и классе c (Mutual Information — MI).[6] Этот показатель измеряет количество информации о принадлежности классу c, которое несет наличие/отсутствие термина.

[6] Не следует путать ожидаемую взаимную информацию (expected mutual information) с поточечной взаимной информацией (pointwise mutual information), равной $\log N_{11}/E_{11}$, где величины N_{11} и E_{11} определяются формулой (13.17). Эти два показателя имеют разные свойства (см. раздел 13.7).

$$I(U;C)=\sum_{e_t\in\{1,0\}}\sum_{e_c\in\{1,0\}}P(U=e_t,C=e_c)\log_2\frac{P(U=e_t,C=e_c)}{P(U=e_t)P(C=e_c)}. \qquad (13.16)$$

Здесь U — случайная величина, принимающая значения $e_t = 1$ (документ содержит термин t) и $e_t = 0$ (документ не содержит термин t), а переменная C — случайная величина, принимающая значения $e_c = 1$ (документ принадлежит классу c) и $e_c = 0$ (документ не принадлежит классу c). Если из контекста неясно, о каком термине t и классе c идет речь, мы будем писать U_t и C_c.

При использовании оценок максимального правдоподобия равенство (13.16) становится эквивалентным равенству (13.17).

$$I(U;C)=\frac{N_{11}}{N}\log_2\frac{NN_{11}}{N_{1.}N_{.1}}+\frac{N_{01}}{N}\log_2\frac{NN_{01}}{N_{0.}N_{.1}}+\frac{N_{10}}{N}\log_2\frac{NN_{10}}{N_{1.}N_{.0}}+\frac{N_{00}}{N}\log_2\frac{NN_{00}}{N_{0.}N_{.0}} \qquad (13.17)$$

Здесь N — количество документов, а индексы соответствуют значениям переменных e_t и e_c. Например, N_{10} — количество документов, содержащих термин t (т.е. $e_t = 1$) и не принадлежащих классу c (т.е. $e_c = 0$). Число $N_{.1} = N_{10} + N_{11}$ — это количество документов, содержащих термин t (т.е. $e_t = 1$) независимо от принадлежности классу c (т.е. $e_c \in \{0,1\}$). Число $N = N_{11} + N_{01} + N_{10} + N_{00}$ — общее количество документов. Примером одной из оценок максимального правдоподобия, преобразующих равенство (13.16) в равенство (13.17), является оценка $P(U = 1, C = 1) = N_{11}/N$.

Пример 13.3. Рассмотрите класс *poultry* и термин export в коллекции Reuters-RCV1. Количество документов с четырьмя возможными комбинациями индикаторов указано ниже.

	$e_c = e_{poultry} = 1$	$e_c = e_{poultry} = 0$
$e_t = e_{export} = 1$	$N_{11} = 49$	$N_{10} = 27\,652$
$e_t = e_{export} = 0$	$N_{01} = 141$	$N_{00} = 774\,106$

Подставляя эти числа в равенство (13.17), получим следующий результат.

$$\begin{aligned}I(U;C)=&\frac{49}{801\,948}\log_2\frac{801\,948\cdot 49}{(49+27\,652)(49+141)}+\\&+\frac{141}{801\,948}\log_2\frac{801\,948\cdot 141}{(141+774\,106)(49+141)}+\\&+\frac{27\,652}{801\,948}\log_2\frac{801\,948\cdot 27\,652}{(49+27\,652)(27\,652+774\,106)}+\\&+\frac{774\,106}{801\,948}\log_2\frac{801\,948\cdot 774\,106}{(141+774\,106)(27\,652+774\,106)}\\\approx&\,0{,}0001105\end{aligned}$$

Для того чтобы выбрать k терминов $t_1, ..., t_k$ для заданного класса, используем алгоритм выбора признаков, описанный на рис. 13.6. Для этого вычислим меру полезности $A(t, c) = I(U_t, C_c)$ и выберем k терминов с наибольшими значениями этой меры.

Мера взаимной информации оценивает, сколько информации о классе — в теоретико-информационном смысле — содержит термин. Если распределение термина в классе совпадает с распределением термина во всей коллекции, то $I(U;C) = 0$. Мера взаимной информации достигает своего максимума, если термин является идеальным индикатором

для класса, т.е. если термин присутствует в документе тогда и только тогда, когда документ принадлежит классу.

На рис. 13.7 показаны термины с высокими показателями взаимной информации относительно шести классов, продемонстрированных на рис. 13.1.[7] Совершенно очевидно, что выбранные термины (например, `london`, `uk`, `british` для класса *UK*) весьма полезны для классификации своих классов. В конце списка терминов, полезных для идентификации класса *UK*, находятся термины наподобие `peripherals` и `tonight` (не приведенные на рисунке), которые, очевидно, не могут быть полезными при решении о принадлежности документа этому классу. Как и следовало ожидать, ориентация на информативные термины и игнорирование неинформативных позволяют уменьшить шум и повысить правильность классификации.

UK

london	0,1925
uk	0,0755
british	0,0596
stg	0,0555
britain	0,0469
plc	0,0357
england	0,0238
pence	0,0212
pounds	0,0149
english	0,0126

China

china	0,0997
chinese	0,0523
beijing	0,0444
yuan	0,0344
shanghai	0,0292
hong	0,0198
kong	0,0195
xinhua	0,0155
province	0,0117
taiwan	0,0108

poultry

poultry	0,0013
meat	0,0008
chicken	0,0006
agriculture	0,0005
avian	0,0004
broiler	0,0003
veterinary	0,0003
birds	0,0003
inspection	0,0003
pathogenetic	0,0003

coffee

coffee	0,0111
bags	0,0042
growers	0,0025
kg	0,0019
columbia	0,0018
brazil	0,0016
export	0,0014
exporters	0,0013
exports	0,0013
crop	0,0012

elections

election	0,1519
elections	0,0342
polls	0,0339
voters	0,0315
party	0,0303
vote	0,0299
poll	0,0225
candidate	0,0202
campaign	0,0202
democratic	0,0198

sports

soccer	0,0681
cup	0,0515
match	0,0441
matches	0,0408
played	0,0388
league	0,0386
beat	0,0301
game	0,0299
games	0,0284
team	0,0264

Рис. 13.7. Признаки с высокими показателями взаимной информации для шести классов из коллекции Reuters-RCV1

Такое изменение правильности можно обнаружить на рис. 13.8, на котором показана зависимость меры F_1 от количества признаков, выбранных из коллекции Reuters-RCV1.[8] Сравнивая меру F_1 при 132 776 признаках (что соответствует выбору всех признаков) и

[7] Показатели были подсчитаны по первым 100 тысячам документов, за исключением класса *poultry*, редкого класса, для которого было использовано 800 тысяч документов. Мы исключили числа и другие специальные слова из десятки.

[8] Классификаторы обучены по первым 100 тысячам документов, а мера F_1 вычислена по следующим 100 тысячам документов. На рисунке показаны графики средних значений F_1 по пяти классам.

при 10–100 признаках, мы видим, что выбор признаков на основе взаимной информации увеличивает показатель F_1 примерно на 0,1 в мультиномиальной модели и более чем на 0,2 — в модели Бернулли. Для модели Бернулли показатель F_1 быстро достигает максимума уже при десяти выбранных признаках. В этой точке модель Бернулли лучше, чем мультиномиальная. Если решение о классификации основывается лишь на нескольких признаках, то для повышения надежности следует учитывать только бинарный признак присутствия терминов. Для мультиномиальной модели (при выборе признаков на основе взаимной информации) пик достигается позднее — на сотне признаков, причем эффективность классификации несколько возрастает в конце, при выборе всех признаков. Причина этого явления заключается в том, что мультиномиальная модель учитывает при оценке параметров и классификации количество вхождений термина и поэтому при большем количестве признаков работает лучше модели Бернулли. Несмотря на различия между этими моделями, использование тщательно выбранного подмножества признаков обеспечивает намного более высокую эффективность, чем использование всех признаков.

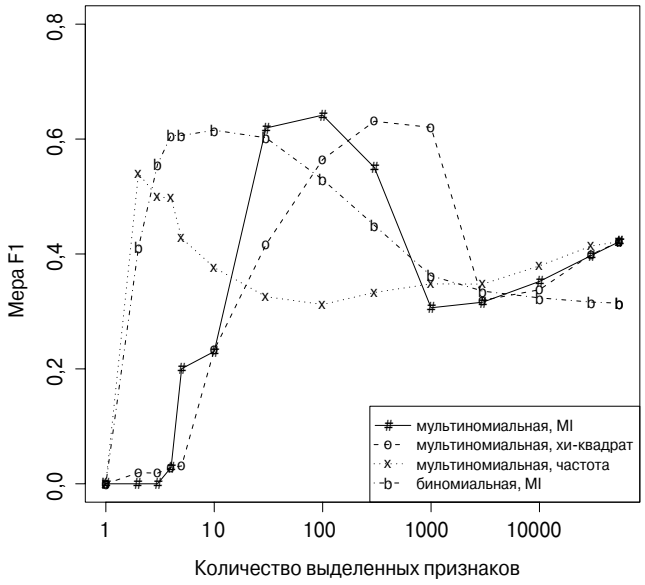


Рис. 13.8. Влияние количества используемых признаков на правильность классификации для мультиномиальной модели и модели Бернулли

13.5.2. Выбор признаков на основе критерия хи-квадрат

Другим распространенным методом выбора признаков является критерий χ^2 (хи-квадрат). В статистике критерий χ^2 используется для проверки независимости двух случайных событий, где события A и B считаются независимыми, если $P(AB) = P(A)P(B)$ или, что эквивалентно, $P(A|B) = P(A)$ и $P(B|A) = P(B)$. При выборе признаков двумя событиями являются появление термина и появление класса. После этого термины ранжируются в соответствии со следующей величиной.

$$X^2(\mathbb{D},t,c)=\sum_{e_t\in\{0,1\}}\sum_{e_c\in\{0,1\}}\frac{\left(N_{e_te_c}-E_{e_te_c}\right)^2}{E_{e_te_c}} \tag{13.18}$$

Здесь e_t и e_c — величины, определенные в равенстве (13.16), N — *наблюдаемая* частота в множестве $\mathbb{D}$, а E — ожидаемая частота. Например, E_{11} — ожидаемая частота одновременного появления термина t в документе и документа в классе c при условии, что термин и класс являются независимыми.

Пример 13.4. Вычислим величину E_{11} по данным из примера 13.3.

$$E_{11}=N\times P(t)\times P(c)=N\times\frac{N_{11}+N_{10}}{N}\times\frac{N_{11}+N_{01}}{N}=N\times\frac{49+141}{N}\times\frac{49+27\,652}{N}\approx 6{,}6.$$

Здесь N — общее количество документов.

Величины $E_{e_te_c}$ вычисляются аналогично.

	$e_c = e_{poultry} = 1$		$e_c = e_{poultry} = 0$	
$e_t = e_{export} = 1$	$N_{11} = 49$	$E_{11} \approx 6{,}6$	$N_{10} = 141$	$E_{10} \approx 27\,694{,}4$
$e_t = e_{export} = 0$	$N_{01} = 27\,652$	$E_{01} \approx 183{,}4$	$N_{00} = 774\,106$	$E_{01} \approx 774\,063{,}6$

Подставляя эти числа в равенство (13.18), приходим к выводу, что величина X^2 равна 284.

$$X^2(\mathbb{D},t,c)=\sum_{e_t\in\{0,1\}}\sum_{e_c\in\{0,1\}}\frac{\left(N_{e_te_c}-E_{e_te_c}\right)^2}{E_{e_te_c}}\approx 284$$

Величина X^2 позволяет судить о том, насколько сильно ожидаемая частота E и наблюдаемая частота N отклоняются друг от друга. Большое значение X^2 означает, что гипотеза о независимости, из которой следует близость ожидаемой и наблюдаемой частот, не выполняется. В нашем примере $X^2 \approx 284 > 10{,}83$. Основываясь на данных табл. 13.6, мы можем отвергнуть гипотезу о том, что класс *poultry* и термин `export` являются независимыми, причем шансы, что мы при этом ошибемся, равны лишь 0,001.[9] Другими словами, результат $X^2 \approx 284 > 10{,}83$ статистически значим на уровне 0,001. Если данные два события зависят друг от друга, то появление термина влечет за собой появление документа в классе с большей (или меньшей) вероятностью. Именно этот факт лежит в основе применения критерия χ^2 для выбора признаков.

Таблица 13.6. Критические значения статистики χ^2 с одной степенью свободы. Например, если два события являются независимыми, то $P(X^2>6{,}63) < 0{,}01$. Следовательно, при $X^2 > 6{,}63$ предположение о независимости можно отклонить с доверительным уровнем, равным 99%

P	Критическое значение χ^2
0,1	2,71
0,05	3,84
0,01	6,63

[9] Мы можем сделать такой вывод, поскольку, если два события независимы, X^2 имеет распределение χ^2 (Rice, 2006).

Окончание табл. 13.6

P	Критическое значение X^2
0,005	7,88
0,001	10,83

Статистику X^2 можно вычислить проще.

$$X^2(\mathbb{D},t,c)=\frac{(N_{11}+N_{10}+N_{01}+N_{00})\times(N_{11}N_{00}-N_{10}N_{01})^2}{(N_{11}+N_{01})\times(N_{11}+N_{10})\times(N_{10}+N_{00})\times(N_{01}+N_{00})} \quad (13.19)$$

Эта формула эквивалентна формуле (13.18) (упражнение 13.14).

Оценка критерия хи-квадрат как метода выбора признаков

Со статистической точки зрения применение критерия хи-квадрат для выбора признаков является проблематичным. Для критерия с одной степенью свободы необходимо использовать так называемую *поправку Йейтса* (Yates correction) (раздел 13.7), которая усложняет достижение статистической значимости. Кроме того, если статистический критерий применяется много раз, то вероятность сделать хотя бы одну ошибку возрастает. Если отклонить одну тысячу гипотез, каждую с вероятностью ошибки 0,05, то в среднем в $0{,}05 \times 1000 = 50$ случаях критерий будет давать неверный ответ. Однако в области классификации текстов включение или исключение нескольких признаков не имеет большого значения. Важную роль играет лишь *относительная* важность признака. Поскольку критерий хи-квадрат позволяет ранжировать признаки только по степени их полезности и не используется для выводов о статистической зависимости или независимости случайных величин, нас не должен очень волновать тот факт, что обоснованность его применения с точки зрения теории статистики является сомнительной.

13.5.3. Выбор признаков на основе частоты

Третий метод выбора признаков основан на использовании *частоты*, т.е. осуществляется отбор терминов, которые чаще всех встречаются в классе. В качестве частоты может использоваться как документная частота (количество документов в классе c, содержащих термин t), так и частота в коллекции (количество лексем термина t, встречающихся в документах класса c). Для модели Бернулли более приемлемой является документная частота, а для мультиномиальной модели — частота в коллекции.

Выбор признаков на основе частоты (frequency-based feature selection) может привести к выбору частотных терминов, не содержащих информации о классе, например дней недели (`Monday`, `Tuesday` и т.д.), которые могут встретиться в любых новостях. При выборе многих тысяч признаков использование частоты является оправданным. Следовательно, если субоптимальная правильность приемлема, необходимо применять метод выбора признаков на основе частоты, а не более сложные методы. Однако рис. 13.8 демонстрирует случай, когда метод выбора признаков на основе частоты работает хуже, чем метод взаимной информации или критерий хи-квадрат, и не должен использоваться.

13.5.4. Выбор признаков для нескольких классификаторов

В системах с большим количеством классификаторов желательно использовать одно множество признаков, а не отдельные множества для каждого классификатора. Для этого

можно вычислить статистику X^2 и заполнить таблицу $n \times 2$, в которой столбцы содержат индикаторы наличия или отсутствия термина, а строки соответствуют одному из классов. Затем мы выбираем k терминов с наибольшими статистиками X^2.

Чаще статистики для выбора признаков сначала вычисляются отдельно для каждого класса, т.е. для задачи бинарной классификации c и $\bar{c}$, а затем комбинируются. В одном из таких комбинационных методов сначала вычисляется показатель качества для каждого признака, например на основе усреднения значения $A(t, c)$ для признака t, а затем выбираются k признаков с лучшими показателями качества. В другом комбинационном методе сначала выбираются k/n признаков для каждого из n классификаторов, а затем эти n множеств интегрируются в одно глобальное множество признаков.

При выборе k общих признаков в системе, состоящей из n классификаторов, правильность классификации часто ниже, чем при выборе n разных множеств размера k. Однако и в этом случае выигрыш в эффективности за счет единообразного представления документов часто превышает проигрыш в правильности.

13.5.5. Сравнение методов выбора признаков

Взаимная информация и критерий хи-квадрат сильно отличаются друг от друга. Гипотезу о независимости термина t и класса c иногда можно с уверенностью отклонить, даже если термин t содержит мало информации о принадлежности документа классу c. Это особенно характерно для редких терминов. Если термин в большой коллекции встречается только один раз и именно в классе *poultry*, то этот факт является статистически значимым. Однако с точки зрения теории информации единственное появление термина в классе нельзя считать очень информативным. Поскольку критерий хи-квадрат является критерием значимости, он чаще выбирает редкие термины (которые часто являются ненадежными индикаторами), чем метод выбора признаков на основе взаимной информации. Однако метод выбора признаков на основе взаимной информации также не всегда выбирает признаки, позволяющие достичь максимальной точности классификации.

Несмотря на различия между этими методами, точность классификации на основе признаков, выбранных с их помощью, различается незначительно. В большинстве задач классификации текстов существует мало сильных индикаторов и много слабых индикаторов. Если в множество выбранных индикаторов попадают все сильные и много слабых индикаторов, правильность классификации должна быть хорошей. Оба метода позволяют это сделать.

На рис. 13.8 показаны результаты сравнения метода выбора признаков на основе взаимной информации и критерия хи-квадрат в рамках мультиномиальной модели. Пики эффективности для обоих методов практически совпадают. Эффективность метода, основанного на критерии хи-квадрат, достигается позднее — при 300 признаках, — вероятно, из-за того, что редко встречающиеся, но очень значимые признаки, которые он выбирает в начале, не охватывают все документы из класса. Однако признаки, выбранные позднее (от 100 до 300), обладают более высоким качеством, чем признаки, выбранные по взаимной информации.

Все три метода — основанные на взаимной информации, статистике хи-квадрат и частоте — являются *жадными* (greedy). Они могут выбирать признаки, не несущие дополнительной информации по сравнению с ранее выбранными. На рис. 13.7 показано, что термин `kong` был выбран семнадцатым, хотя он сильно коррелирует с ранее выбранным термином `hong`, а значит, является избыточным. Несмотря на то что эта избыточность может отрицательно влиять на правильность классификации, нежадные методы

(см. ссылки в разделе 13.7) редко используются для классификации текстов из-за их вычислительной сложности.

? ***Упражнение 13.5.*** Проанализируйте следующие частоты четырех терминов в классе *coffee* в первых 100 тысячах документов коллекции Reuters-RCV1.

Термин	N_{00}	N_{01}	N_{10}	N_{11}
brazil	98 012	102	1835	51
council	96 322	133	3525	20
producers	98 524	119	1118	34
roasted	99 824	143	23	10

Выберите два из четырех терминов, используя 1) статистику хи-квадрат, 2) взаимную информацию, 3) частоту.

13.6. Оценка классификации текстов

Исторически основным тестом для оценки методов классификации текстов являлась коллекция Reuters-21578. Она состоит из 21 578 новостных сообщений, собранных и размеченных компаниями Carnegie Group, Inc. и Reuters, Ltd. в ходе разработки системы классификации текстов CONSTRUE. Эта коллекция намного меньше более поздней коллекции Reuters-RCV1. Ее статьи распределены по классам, охватывающим 118 тематических категорий. Документ может принадлежать как нескольким классам, так и ни одному из них, но чаще всего документы принадлежат только одному классу (документы, относящиеся хотя бы к одному классу, в среднем принадлежат 1,24 классам). Стандартный подход к решению такой задачи многозначной классификации заключается в создании 118 классификаторов, по одному на каждый класс, причем *бинарный классификатор* для класса c — это классификатор для класса c и его дополнения $\overline{c}$.

Для каждого из этих классификаторов можно измерить полноту (recall), правильность (accuracy) и точность (precision). В работах, опубликованных недавно, исследователи практически постоянно использовали подмножество *ModApte split*, состоящее из 9 603 обучающих и 3 299 тестовых документов, просмотренных и размеченных вручную. Распределение документов по классам очень неравномерное, и поэтому в некоторых работах оценка работы системы производилась по десяти самым большим классам. Они перечислены в табл. 13.7. Типичный документ с указанием тем продемонстрирован на рис. 13.9.

Таблица 13.7. Десять самых больших классов в коллекции Reuters-21578, а также количество обучающих и тестовых документов

Класс	Количество обучающих документов	Количество тестовых документов	Класс	Количество обучающих документов	Количество тестовых документов
earn	2877	1087	*trade*	369	119
acquisitions	1650	179	*interest*	347	131
money-fx	538	179	*ship*	197	89
grain	433	149	*wheat*	212	71
crude	389	189	*corn*	182	56

```
<REUTERS TOPICS="YES" LEWISSPLIT="TRAIN"
CGISPLIT="TRAINING-SET" OLDID="12981" NEWID="798">
<DATE> 2-MAR-1987 16:51:43.42</DATE>
<TOPICS><D>livestock</D><D>hog</D></TOPICS>
<TITLE>AMERICAN PORK CONGRESS KICKS OFF TOMORROW</TITLE>
<DATELINE> CHICAGO, March 2 - </DATELINE><BODY>The American Pork
Congress kicks off tomorrow, March 3, in Indianapolis with 160
of the nations pork producers from 44 member states determining
industry positions on a number of issues, according to the
National Pork Producers Council, NPPC.
Delegates to the three day Congress will be considering 26
resolutions concerning various issues, including the future
direction of farm policy and the tax law as it applies to the
agriculture sector. The delegates will also debate whether to
endorse concepts of a national PRV (pseudorabies virus) control
and eradication program, the NPPC said. A large
trade show, in conjunction with the congress, will feature
the latest in technology in all areas of the industry, the NPPC
added. Reuter
&#3;</BODY></TEXT></REUTERS>
```

Рис. 13.9. Пример документа из коллекции Reuters-21578

Как указано в разделе 13.1, цель классификации текстов — минимизация ошибки классификации на тестовых данных. Ошибка классификации равна разности между единицей и правильностью классификации, т.е. доле правильных решений (см. раздел 8.3). Этот показатель является приемлемым, если доля документов класса велика, т.е. больше 10%. Однако, как сказано в разделе 8.3, правильность не подходит для "небольших" классов, поскольку в этом случае высокой степени правильности можно достичь, просто всегда отвечая "нет". Если относительная частота класса в коллекции составляет 1%, то классификатор "всегда нет" гарантирует 99%-ную правильность. Для небольших классов лучше использовать точность, полноту и меру F_1.

В качестве показателя, обобщающего понятие качества классификации, включая точность, полноту, меру F_1 и правильность, мы будем использовать *эффективность* (effectiveness). *Производительность* (performance) — это вычислительная эффективность классификации и систем информационного поиска. Однако многие исследователи, используя термин "производительность", подразумевают скорость, а не результативность (efficiency).

Обрабатывая коллекцию с помощью нескольких бинарных классификаторов (например, коллекцию Reuters-21578 с ее 118 классами), мы часто хотим вычислить единственный агрегированный показатель, объединяющий показатели отдельных классификаторов. Для этого есть два метода. При *макроусреднении* (macroaveraging) вычисляется обычное среднее значение по классам. При *микроусреднении* (microaveraging) объединяются все решения на уровне документов по всем классам, а затем вычисляется мера эффективности по объединенной факторной таблице (см. пример в табл. 13.8).

Таблица 13.8. Микро- и макроусреднение. "Truth" — это истинный класс, а "Call" — это решение классификатора. Например, макроусредненная точность равна [10/(10+10)+90/(10+90)]/2=(0,5+0,9)/2=0,7, а микроусредненная — 100/(100+20)≈ 0,83

Класс 1			Класс 2			Объединенная таблица		
	Truth: да	Truth: нет		Truth: да	Truth: нет		Truth: да	Truth: нет
Call: да	10	10	Call: да	90	10	Call: да	100	20
Call: нет	10	970	Call: нет	10	890	Call: нет	20	1860

Разница между этими двумя методами может быть велика. Макроусреднение приписывает решениям классификатора для каждого класса одинаковые веса, в то время как микроусреднение приписывает одинаковые веса решениям классификатора на каждом документе[10]. Поскольку мера F_1 игнорирует истинно отрицательные, а ее величина определяется в основном количеством истинно положительных, при микроусреднении большие классы доминируют над малыми. Например, точность, полученная при микроусреднении (0,83), намного ближе к точности по классу c_2 (0,9), чем по классу c_1 (0,5), поскольку класс c_2 в пять раз больше класса c_1. Следовательно, результаты применения микроусреднения на самом деле оценивают качество системы на крупных классах из тестовой коллекции. Для того чтобы правильнее оценить ее качество на малых классах, следует применить макроусреднение.

В однозначной классификации (раздел 14.5) микроусредненная мера F_1 совпадает с правильностью (упражнение 13.6).

В табл. 13.9 приведены микро- и макроусредненные значения эффективности наивного байесовского метода на подмножестве ModApte коллекции Reuters-21578. Для того чтобы понять смысл относительной эффективности наивного байесовского метода (NB), мы сравнили его с линейным *методом опорных векторов* (Support Vector Machine — SVM) (последний столбец; см. главу 15) — одним из наиболее эффективных классификаторов, трудоемкость обучения которого однако больше, чем у наивного байесовского метода. Микроусредненная мера F_1 для наивного байесовского метода равна 80%, что на 9% меньше, чем для метода SVM (89%), что соответствует относительному уменьшению на 10%. Итак, за простоту и эффективность наивного байесовского метода мы заплатили удивительно небольшую цену. Однако на небольших классах, содержащих порядка десяти примеров в обучающих множествах, наивный байесовский метод работает намного хуже. Его макроусредненная мера F_1 на 13% меньше, чем для метода SVM, т.е. происходит снижение на 22%.

Таблица 13.9. Сравнение показателей эффективности классификации текстов на коллекции Reuters-21578 по мере F1 (в процентах). Результаты цитируются по работам Ли и Янга (Li and Young, 2003) (а), а также Йоахимса (Joachims, 1998) (kNN, б) и Дюмэ и др. (Dumais et al., 1998) (наивный байесовский подход, метод Роккио, деревья решения, метод опорных векторов, б)

а	Наивный байесовский метод	Метод Роккио	kNN	Метод опорных векторов
Микроусредненная мера F_1 (90 классов)	80	85	86	89
Макроусредненная мера F_1	47	59	60	60

б	Наивный байесовский метод	Метод Роккио	kNN	Деревья решений	Метод опорных векторов
Earn	96	93	97	98	98
Acq	88	65	92	90	94
money-fx	57	47	78	66	75
grain	79	68	82	85	95

[10] То есть классы с бо́льшим числом документов и решений по ним вносят больший вес в микроусреднение. — *Примеч. ред.*

Окончание табл. 13.9

б	Наивный байесовский метод	Метод Роккио	kNN	Деревья решений	Метод опорных векторов
crude	80	70	86	85	89
trade	64	65	77	73	76
interest	65	63	74	67	78
ship	85	49	79	74	86
wheat	70	69	77	93	92
corn	65	48	78	92	90
Микроусредненная мера F_1 (10 крупнейших классов)	82	65	82	88	92
Микроусредненная мера F_1 (118 классов)	75	62	n/a	n/a	87

В таблице также сравниваются метод NB и другие классификаторы, описанные в книге: алгоритмы Роккио (Rocchio) и kNN. Кроме того, мы привели показатели для метода *деревьев решений* (decision trees), важный метод классификации, который в книге не освещается. В нижней части таблицы наблюдается сильная вариация показателей от класса к классу. Например, метод NB эффективнее метода kNN на классе *ship*, но намного хуже на классе *money-fx*.

Сравнивая части таблицы *а* и *б*, следует обратить внимание на то, насколько сильно отличаются результаты. Это частично объясняется тем, что числа в части *б* являются сбалансированными показателями, усредненными по 118 классам, в то время как в части *а* они являются истинными значениями меры F_1, вычисленными без какой-либо информации о тестовом множестве и усредненными по 90 классам. К сожалению, эта ситуация типична для сравнения разных результатов классификации текстов. Экспериментальные параметры и методы оценки часто сильно отличаются друг от друга, что усложняет интерпретацию результатов.

Эти и другие результаты показали, что средняя эффективность наивного байесовского метода не может конкурировать с другими методами, например с методом опорных векторов, если обучающие и тестовые данные являются *независимыми и одинаково распределенными* (independent and identically distributed — i.i.d.). Однако в реальном мире, в котором обучающие выборки извлекаются из подмножества данных, к которым классификатор будет применяться в дальнейшем, природа данных изменяется во времени (ср. дрейф понятий), а данные содержат ошибки (помимо всего прочего), разница становится не такой существенной и даже может свидетельствовать в пользу простого подхода. В некоторых реальных приложениях не удавалось создать изощренный классификатор, который был существенно лучше, чем наивный байесовский метод.

Из табл. 13.9 следует, что несмотря на то, что некоторые исследователи считают метод SVM лучше метода kNN, а метод kNN — лучше метода NB, ранжирование классификаторов в конце концов зависит от класса, коллекции документов и условий эксперимента. Как будет показано в разделе 15.3, в области классификации текстов сам по себе метод не является определяющим фактором.

Проводя оценку, аналогичную проиллюстрированной в табл. 13.9, важно строго разделять обучающее и тестовое множества. Имея даже частичную информацию о тестовом

множестве, например о том, что некий термин является хорошим предсказателем (даже несмотря на то, что это неверно по отношению к обучающему множеству), мы можем с легкостью получить хорошие результаты классификации. Более тонким примером использования знаний о тестовом множестве является попытка проверить большое количество разных значений параметра (например, количества выбранных признаков) и выбрать наилучший для данного тестового множества. Как правило, правильность на новых данных, т.е. на данных, которые нужно будет обрабатывать в рабочем режиме, намного ниже, чем правильность на тестовом множестве, на которое настроен классификатор[11]. Аналогичную проблему мы уже обсуждали, когда изучали поиск по произвольному запросу (см. раздел 8.1).

В лабораторных статистических экспериментах по классификации текстов никогда не следует запускать программу на тестовых данных и даже смотреть на данные в ходе разработки системы. Вместо этого следует выбрать *рабочее множество* (development set), на котором будет отлаживаться ваш метод. Именно это множество должно помочь вам определить наилучший параметр, например количество выбранных признаков, поэтому такое множество называется *отложенными данными* (held-out data). Проведите обучение классификатора на остальной части обучающего множества с разными значениями параметров и выберите то значение, которое дает лучшие результаты на отложенной части обучающего множества. В идеальном случае после настройки всех параметров и завершения разработки метода проводится окончательный эксперимент на тестовом множестве и публикуются его результаты. Поскольку при разработке классификатора никакая информация о тестовом множестве не использовалась, результаты такого эксперимента следует считать достоверными свидетельствами реального качества системы.

Этот идеал часто недостижим. Исследователи, как правило, одновременно оценивают несколько систем на одном и том же тестовом множестве на протяжении нескольких лет. Тем не менее крайне важно не заглядывать в тестовые данные и запускать систему на них как можно реже. Новички часто нарушают это правило, и их результаты оказываются некорректными, потому что они неявно настраивают свою систему на тестовые данные, просто запуская на них разные варианты системы и выбирая затем наилучший.

? ***Упражнение 13.6* [**].** Представьте ситуацию, в которой каждый документ тестовой коллекции относится только к одному классу, а классификатор также относит один документ к одному классу. Такая схема называется однозначной классификацией (раздел 14.5). Покажите, что в этой схеме 1) общее количество ложно позитивных решений равно общему количеству ложно отрицательных и 2) микроусредненная мера F_1 и правильность совпадают.

Упражнение 13.7. Априорные оценки в алгоритме, представленном на рис. 13.2, были вычислены как доля *документов* в классе, а не как доля *лексем* в классе. Почему?

[11] Эта мысль очень важна: по сути, она ставит под сомнение классическое применение тестового множества на практике: зачем выбрасывать из обучения примеры и терять в обучающей способности, если никакого выигрыша от независимости тестового набора мы не получаем, так как полная независимость на практике недостижима. — *Примеч. ред.*

Упражнение 13.8. Функция ApplyMultinomialNB на рис. 13.2 имеет временную сложность $\Theta(L_a + |\mathbb{C}|L_a)$. Как изменить эту функцию, чтобы ее временная сложность стала равной $\Theta(L_a + |\mathbb{C}|M_a)$?

Упражнение 13.9. На основе данных из табл. 13.10 1) оцените наивный байесовский мультиномиальный классификатор, 2) примените этот классификатор к тестовому документу, 3) оцените наивный байесовский классификатор Бернулли и 4) примените этот классификатор к тестовому документу. Не следует оценивать параметры, которые не требуются для классификации тестового документа.

Таблица 13.10. Данные для упражнения по оценке параметров

	docID	Слова в документе	c = China?
Обучающее множество	1	Taipei Taiwan	Да
	2	Macao Taiwan Shanghai	Да
	3	Japan Sapporo	Нет
	4	Sapporo Osaka Taiwan	Нет
Тестовое множество	5	Taiwan Taiwan Sapporo	?

Упражнение 13.10. Классифицируйте слова на английские и не английские. Слова генерируются источником, имеющим следующее распределение.

Событие	Слово	Английское?	Вероятность
1	Ozb	Нет	4/9
2	Uzu	Нет	4/9
3	zoo	Да	1/18
4	bun	Да	1/18

1) Вычислите параметры (априорные и условные) мультиномиального наивного байесовского классификатора, в котором в качестве выбранных признаков используются буквы b, n, o, u и z. Предположим, что обучающее множество идеально повторяет распределение источника. Сделайте те же предположения о независимости, которые обычно выдвигаются при работе с мультиномиальным классификатором, использующим термины в качестве признаков для классификации текстов. Вычислите параметры, используя сглаживание, в котором нулевые вероятности заменяются величиной 0,01, а ненулевые вероятности остаются неизменными. (При таком упрощенном сглаживании может возникнуть неравенство $P(A)+P(\bar{A})>1$. Исправлять эту ситуацию в данном упражнении не нужно.) 2) Как этот классификатор классифицирует слово zoo? 3) Классифицируйте слово zoo с помощью мультиномиального классификатора, не выдвигая предположение о позиционной независимости. Иначе говоря, оцените отдельные параметры для каждой координаты в слове. Вычислять следует только те параметры, которые необходимы для классификации слова zoo.

Упражнение 13.11. Чему равны статистики $I(U_t; C_c)$ и $X^2(\mathbb{D}, t, c)$, если термин и класс полностью независимы? Чему они равны, если термин и класс полностью зависимы?

Упражнение 13.12. Метод выбора признаков в равенстве (13.16) лучше всего подходит для модели Бернулли. Почему? Как модифицировать его для мультиномиальной модели?

Упражнение 13.13. Признаки можно выбрать с помощью *прироста информации* (information gain), который вычисляется следующим образом.

$$IG(\mathbb{D},t,c) = H(p_{\mathbb{D}}) - \sum_{x\in\{\mathbb{D}_{t^+},\mathbb{D}_{t^-}\}} \frac{|x|}{|\mathbb{D}|} H(p_x)$$

Здесь H — энтропия, $\mathbb{D}$ — обучающее множество, $\mathbb{D}_{t^+}$ и $\mathbb{D}_{t^-}$ — подмножества множества $\mathbb{D}$, содержащие и не содержащие термин t соответственно, $p_{A\_}$ распределение класса в (под)коллекции A, например если четверть документов из коллекции A принадлежит классу c, то $p_A(c) = 0{,}25$, $p_A(\bar{c}) = 0{,}75$.

Покажите, что взаимная информация и прирост информации эквивалентны.

Упражнение 13.14. Покажите, что формулы (13.18) и (13.19) для вычисления статистики X^2 эквивалентны.

Упражнение 13.15. В примере 13.4 фигурировало равенство $|N_{11} - E_{11}| = |N_{10} - E_{10}| = |N_{01} - E_{01}| = |N_{00} - E_{00}|$. Покажите, что эти равенства выполняются и в общем случае.

Упражнение 13.16. Статистика χ^2 и взаимная информация не позволяют различить положительно и отрицательно коррелированные признаки. Поскольку большинство хороших признаков положительно коррелируют друг с другом (т.е. чаще встречаются в классе c, чем в классе $\bar{c}$), можно сформулировать явное правило для выявления отрицательно коррелированных признаков. Как это сделать?

13.7. Библиография и рекомендации для дальнейшего чтения

Общее введение в статистическую классификацию и машинное обучение, включая многие важные методы (например, деревья решений и бустинг, которые мы не упоминали), изложены в работах Хасти и др. (Hastie et al., 2001), Митчела (Mitchell, 1997) и Дуда и др. (Duda et al., 2000). Всесторонний обзор методов классификации текстов и результатов, полученных в этой области, приведен в работе Себастиани (Sebastiani, 2002). Доступное введение в классификацию текстов, включая деревья решений, персептроны и модели максимальной энтропии, содержится в работе Маннинга и Шютце (Manning and Schütze, 1999). Более подробную информацию о суперлинейной временной сложности методов обучения, более точных, чем наивный байесовский метод, можно найти в работах Перкинса и др. (Perkins et al., 2003), а также Йоахимса (Joachins, 2006*a*).

Один из первых наивных байесовских классификаторов текста описали Марон и Кунс (Maron and Kuhns, 1960). Льюис (Lewis, 1998) сосредоточил внимание на истории этого

метода. Бернуллиевские и мультиномиальные модели, а также их точность для разных коллекций рассмотрены в работе Маккалума и Нигэма (McCallum and Nigam, 1998). Эйхераменди и др. (Eyheramendy et al., 2003) предложили новые наивные байесовские модели. Домингос и Паццани (Domingos and Pazzani, 1997), Фридман (Friedman, 1997), а также Хэнд и Ю (Hand and Yu, 2001) проанализировали, почему наивный байесовский метод работает достаточно хорошо, хотя его оценки вероятностей неточны. В первой из этих работ обсуждается также оптимальность наивного байесовского метода в ситуации, когда относительно данных выполняется гипотеза о независимости. Павлов и др. (Pavlov et al., 2004) предложили модифицированное представление документов, которое частично решает проблему некорректного предположения о независимости. Беннет (Bennet, 2000) предположил, что близость наивных байесовских оценок вероятностей к нулю или единице объясняется длиной документов. Нг и Джордан (Ng and Jordan, 2001) показали, что наивный байесовский метод иногда (довольно редко) превосходит дискриминантные методы, поскольку он быстро достигает оптимального уровня ошибок. Эффективность основной наивной байесовской модели, описанной в этой главе, можно повысить (Rennie et al., 2003; Kołcz and Yih, 2007). Проблема дрейфа понятий и другие причины, по которым современные классификаторы не всегда хороши на практике, описаны в работах Формана (Forman, 2006) и Хэнда (Hand, 2006).

Впервые взаимная информация и статистика χ^2 для выбора признаков в классификации текстов была применена Льюисом и Рингеттом (Lewis and Ringuette, 1994), а также Шютце и др. (Schütze et al., 1995) соответственно. Янг и Педерсен (Yang and Pedersen, 1997) сделали обзор методов выбора признаков и оценки их влияния на эффективность классификации. Они выяснили, что *поточечная взаимная информация* (pointwise mutual information) не может конкурировать с другими методами. Янг и Педерсен называли ожидаемую взаимную информацию (см. формулу (13.16)) *приростом информации* (см. упражнение 13.13). Хорошим источником сведений о статистике χ^2, а также о поправке Йейтса на непрерывность для таблиц 2×2 является книга Снедекора и Кохрана (Snedecor and Cochran, 1989). Даннинг (Dunning, 1993) обсудил проблемы применения критерия хи-квадрат, когда частоты малы. Нежадные методы выбора признаков описаны в работе Хасти и др. (Hastie et al., 2001). Коэн (Cohen, 1995) указал на ловушки, связанные с использованием нескольких критериев значимости, и описал, как их избежать. Формен (Forman, 2004) оценил несколько методов выбора признаков для нескольких классификаторов.

Дэвид Льюис (David D. Lewis) описал коллекцию ModApte по адресу `www.daviddlewis.com/resources/testcollections/reuters21578/readme.txt`, основываясь на работе Апте и др. (Apté et al., 1994). Льюис (Lewis, 1995) описал *показатели полезности* для оценки систем классификации текстов. Янг и Лиу (Uang and Liu, 1999) применили критерии значимости для оценки методов классификации текстов.

Льюис и др. (Lewis et al., 2004) выяснили, что метод опорных векторов (глава 15) на коллекции Reuters-RCV1 работает лучше, чем алгоритмы kNN и Роккио (глава 14).

Глава 14

Классификация в векторном пространстве

В наивной байесовской модели документ представляется в виде последовательности терминов или бинарного вектора $\{e_1, e_2, ..., e_M\} \in \{0,1\}^{|V|}$. В этой главе мы используем другое представление документа для классификации текстов — векторную модель, разработанную в главе 6. В ней каждый документ рассматривается как вектор, состоящий из действительных чисел, как правило — из весов tf–idf каждого термина. Таким образом, пространство документов $\mathbb{X}$, т.е. область определения функции классификации γ, совпадает с пространством $\mathbb{R}^{|V|}$. В этой главе описываются методы классификации, которые оперируют векторами действительных чисел.

В основе использования модели векторного пространства для классификации лежит *гипотеза компактности* (contiguity hypothesis).

> **Гипотеза компактности.** Документы, принадлежащие одному и тому же классу, образуют компактную область, причем области, соответствующие разным классам, не пересекаются.

Существует много задач классификации, в частности задачи, рассмотренные в главе 13, в которых классы отличаются употреблением слов. Например, документы в классе *China*, скорее всего, имеют большие значения на осях, соответствующих терминам `Chinese`, `Beijing` и `Mao`, в то время как документы из класса *UK* — большие значения на осях, соответствующих терминам `London`, `British` и `Queen`. Следовательно, документы из двух классов образуют разные непрерывные области, как показано на рис. 14.1. Между этими областями можно провести границы и классифицировать новые документы. Именно это является темой данной главы.

Заполняет ли множество документов непрерывную область, зависит от конкретного выбора представления документа: типа взвешивания, списка стоп-слов и т.д. Для того чтобы убедиться, что представление документа играет очень важную роль, рассмотрим два класса (документов): *написанные группой авторов* и *написанные отдельным человеком*. Высокая частота местоимения первого лица *I* (*я*), очевидно, является признаком второго класса. Однако эта информация, скорее всего, будет удалена из представления документа, если используется список стоп-слов. Если представление документа выбрано неудачно, то гипотеза компактности не будет выполняться и классификация в векторном пространстве станет невозможной.

В данном случае можно повторить те же рассуждения, которые привели нас к взвешенным представлениям, в частности — к нормализованным по длине представлениям tf–idf (см. главы 6 и 7). Например, термин, пять раз встречающийся в документе, должен иметь больший вес, чем термин, который встречается только один раз, но приписывать такому термину в пять раз больший вес означает придавать ему слишком большое зна-

чение. В векторной модели классификации не следует применять невзвешенные и ненормализованные частоты.

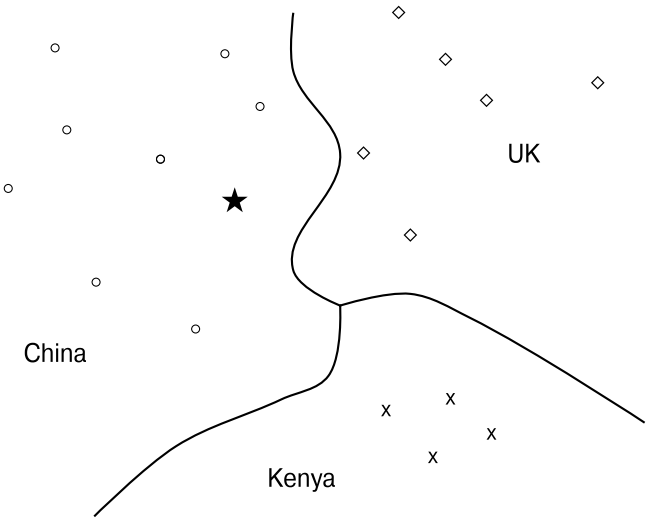


Рис. 14.1. Классификация на три класса в векторном пространстве

В этой главе рассматриваются две модели векторной классификации: Роккио (Rocchio) и kNN (k nearest neighours — k ближайших соседей). Классификация Роккио (раздел 14.2) разделяет векторное пространство на области, окружающие центроиды, или *прототипы*, по одному для каждого класса. Эти центроиды представляют собой центры масс всех документов в классе. Классификация Роккио проста в реализации и эффективна по скорости работы, но неточна, если классы далеки от сфер с примерно одинаковыми радиусами.

Метод kNN, или классификация по *k* ближайшим соседям (k nearest neighbor), описанная в разделе 14.3, относит тестовый документ к классу, которому принадлежат *k* его ближайших соседей. Метод kNN не требует явного обучения и допускает использование обучающего множества в процессе классификации без предварительной обработки. Он имеет большую временную сложность, чем другие методы классификации документов. Если обучающее множество велико, то метод kNN лучше справляется с несферическими и другими сложными классами, чем метод Роккио.

Многие классификаторы текстов можно рассматривать как линейные классификаторы, т.е. классификаторы, основанные на простой линейной комбинации признаков (раздел 14.4). Такие классификаторы разбивают пространство признаков на области с помощью разделяющих гиперплоскостей (decision hyperplanes). Из-за компромисса между смещением и дисперсией (раздел 14.6) более сложные нелинейные модели не всегда лучше линейных. Нелинейные модели имеют много параметров, которые следует подогнать на ограниченном объеме данных для обучения, и при этом для небольших и зашумленных наборов данных возрастает вероятность ошибок.

Применяя бинарные классификаторы для решения задач с несколькими классами, мы интерпретируем их либо как задачи *однозначной классификации* (one-of), т.е. документ должен быть отнесен только к одному из нескольких взаимно исключающих классов, либо как задачи *многозначной классификации* (any-of), т.е. документ может быть припи-

сан любому количеству классов (раздел 14.5). Бинарные классификаторы решают задачу многозначной классификации, а их комбинации можно использовать для решения задач однозначной классификации.

14.1. Представление документов и меры близости в векторном пространстве

Как и в главе 6, будем представлять документы в виде векторов из пространства $\mathbb{R}^{|V|}$. Для того чтобы проиллюстрировать свойства векторов документов в задачах векторной классификации, представим их в виде точек на плоскости (см. рис. 14.1). В действительности векторы документов являются нормализованными по длине единичными векторами, координаты которых лежат на поверхности гиперсферы. Мы можем рассматривать двумерные плоскости на наших рисунках как проекции поверхности гиперсферы на плоскость (рис. 14.2). Расстояния между точками на поверхности сферы и между точками на ее проекции примерно совпадают, если области на поверхности малы, а проекция выбрана корректно (упражнение 14.1).

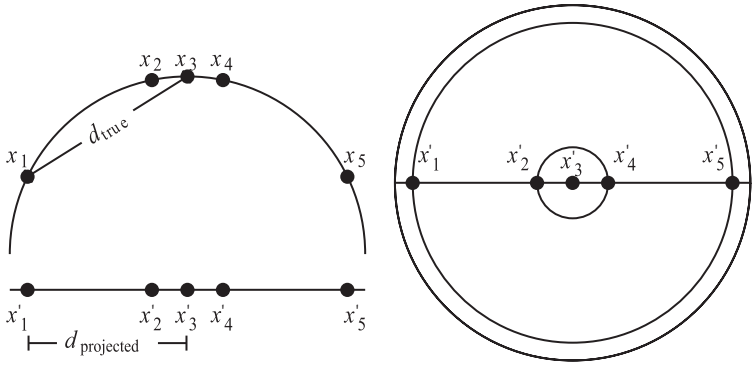


Рис. 14.2. Проекции небольших областей единичной сферы сохраняют расстояния. Слева: проекция двумерной полуокружности на числовую ось. Для точек x_1, x_2, x_3, x_4 и x_5 с x-координатами –0,9, –0,2, 0, 0,2 и 0,9 расстояние $|x_2x_3|$≈0,210 лишь на 0,5% отличается от расстояния $|x'_2x'_3| = 0,2$, но отношение $\frac{|x_1x_3|}{|x'_1x'_3|} = \frac{d_{true}}{d_{projected}} \approx \frac{1,06}{0,9} \approx 1,18$ представляет собой пример крупного искажения (18%), возникающего при проектировании больших областей. Справа: соответствующая проекция трехмерной полусферы на плоскость

Решения векторных классификаторов основаны на понятии расстояния, например на вычислении ближайших соседей в методе kNN. В этой главе в качестве основной меры близости выбрано евклидово расстояние. Как указывалось ранее (см. упражнение 6.18), между косинусной мерой сходства и расстоянием для векторов, нормализованных по длине, существует прямое соответствие. В векторной классификации очень редко имеет значение, как выражается близость двух документов — через меру сходства или расстояние.

Однако, кроме документов, в теории векторной классификации большую роль играют центроиды, или усредненные векторы. Центроиды не являются нормализованными по длине. Для таких векторов скалярное произведение, косинусная мера сходства и евклидово расстояние, в принципе, ведут себя по-разному (упражнение 14.4). В основном при определении сходства между документами и центроидом нас будут интересовать небольшие области, причем чем меньше область, тем более схожими становятся свойства всех трех мер близости.

? ***Упражнение 14.1.*** Для небольших областей расстояния на поверхности гиперсферы хорошо аппроксимируются расстояниями между их проекциями (см. рис. 14.2), поскольку для небольших углов $\alpha \approx \sin\alpha$. При какой величине угла искажение $\alpha/\sin\alpha$ равно 1) 1,01, 2) 1,05 и 3) 1,1?

14.2. Метод Роккио

На рис. 14.1 показаны три класса на двумерной плоскости: *China*, *UK* и *Kenya*. Документы отмечены кружочками, ромбиками и крестиками. *Разделяющие границы* (decision boundaries) на рисунке выбраны таким образом, чтобы разделить три класса, но в остальном их свойства произвольны. Для классификации нового документа, отмеченного на рисунке звездочкой, мы определяем область, в которую он попал, а затем класс, соответствующий этой области (в данном случае это класс *China*). Векторная классификация сводится к разработке алгоритма, вычисляющего “хорошие” границы, где термин “хорошие” означает высокую точность классификации на данных, не использованных в ходе обучения.

Для векторной классификации документов необходимо определить границы между классами, поскольку именно они определяют результат классификации. Вероятно, наиболее известным методом определения этих границ является *метод Роккио* (Rocchio classification), в котором для идентификации границ используются центроиды. Центроид класса c вычисляется как усредненный вектор, или центр масс членов класса.

$$\vec{\mu}(c) = \frac{1}{|D_c|} \sum_{d \in D_c} \vec{v}(d) \tag{14.1}$$

Здесь D_c — множество документов из пространства $\mathbb{D}$, принадлежащих классу c: $D_c = \{d: <d, c> \in \mathbb{D}\}$, $\vec{v}(d)$ — нормализованный вектор документа d (см. формулу (6.11)). На рис. 14.3 показаны три центроида, отмеченные черными кружочками.

Граница между двумя классами в методе Роккио представляет собой множество точек, равноудаленных от двух центроидов. Например, $|a_1| = |a_2|$, $|b_1| = |b_2|$ и $|c_1| = |c_2|$. Это множество точек всегда образует линию. Обобщением этой линии в M-мерном пространстве является гиперплоскость, которая представляет собой множество точек $\vec{x}$, удовлетворяющих условию

$$\vec{w}^T \vec{x} = b\,. \tag{14.2}$$

Здесь $\vec{w}$ — M-мерный *вектор нормали* (normal vector) [1] к гиперплоскости, а b — константа. Это определение гиперплоскостей включает в себя линии (любая линия на дву-

[1] Напомним, что $(\vec{v}, \vec{w}) = \vec{v}^T \vec{w}$, т.е. скалярное произведение векторов $\vec{v}$ и $\vec{w}$, равно произведению транспонированного вектора $\vec{v}$ и вектора $\vec{w}$.

мерной плоскости описывается уравнением $w_1x_1 + w_2x_2 = b$) и двумерные плоскости (любая плоскость в трехмерном пространстве определяется уравнением $w_1x_1 + w_2x_2 + w_3x_3 = b$). Линия разделяет плоскость на две полуплоскости, плоскость разделяет трехмерное пространство на два подпространства и гиперплоскость разделяет пространство более высокой размерности на два подпространства.

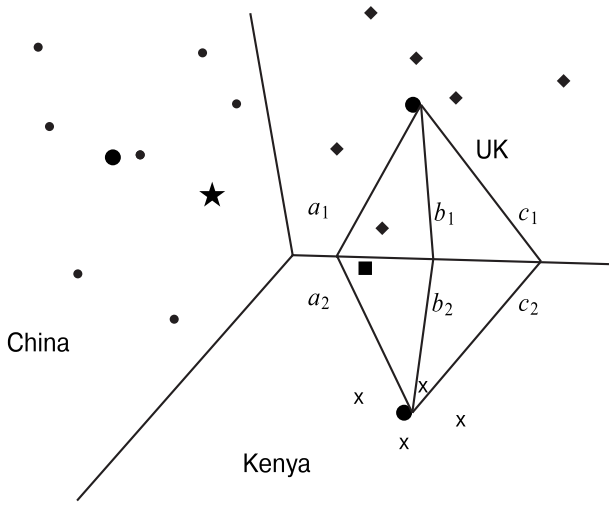


Рис. 14.3. Классификация Роккио

Таким образом, границами областей классов в методе Роккио являются гиперплоскости. Правило классификации в алгоритме Роккио заключается в определении области, в которую попадает точка. Это эквивалентно поиску центроида $\vec{\mu}(c)$, к которому точка лежит ближе, чем к другим центроидам, и приписыванию этой точки к классу c. В качестве примера рассмотрим звездочку на рис. 14.3. Она лежит в области *China*, поэтому алгоритм Роккио относит ее к классу *China*. Псевдокод алгоритма Роккио представлен на рис. 14.4.

`TrainRocchio(`$\mathbb{C}$`,` $\mathbb{D}$`)`

1 **for each** $c_j \in \mathbb{C}$

2 **do** $D_j \leftarrow \{d: \langle d, c_j \rangle \in \mathbb{D}\}$

3 $\vec{\mu}_j \leftarrow \frac{1}{|D_j|} \sum_{d \in D_j} \vec{v}(d)$

4 **return** $\{\vec{\mu}_1, \vec{\mu}_2, \ldots, \vec{\mu}_j\}$

`ApplyRocchio(`$\mathbb{C}$`,` $\mathbb{D}$`)`

1 **return** $\arg\min_j |\vec{\mu}_j - \vec{v}(d)|$

Рис. 14.4. Классификация Роккио. Обучение и тестирование

Пример 14.1. В табл. 14.1 приведены векторные представления с весами tf–idf пяти документов из табл. 13.1, полученных с помощью формулы $(1 + \log_{10}\mathrm{tf}_{t,d})\times$ $\times\log_{10}(4/\mathrm{df}_t)$, если $\mathrm{tf}_{t,d} > 0$ (см. формулу (6.14)). Центроидами классов являются два вектора: $\vec{\mu}_c = \frac{1}{3}\left(\vec{d}_1 + \vec{d}_2 + \vec{d}_3\right)$ и $\vec{\mu}_{\bar{c}} = \frac{1}{1}\vec{d}_4$. Расстояния тестового документа от центроидов равны $\left|\vec{\mu}_c - \vec{d}_5\right| \approx 1{,}15$ и $\left|\vec{\mu}_{\bar{c}} - \vec{d}_5\right| \approx 0{,}0$. Следовательно, алгоритм Роккио отнесет документ $\vec{d}_5$ к классу *c*.

Разделяющая гиперплоскость в этом случае описывается следующими параметрами.

$\vec{w} \approx (0 \quad -0{,}71 \quad -0{,}71 \quad 1/3 \quad 1/3 \quad 1/3)^T$,

$b = -1/3$

Вычисление вектора $\vec{w}$ и константы *b* описано в упражнении 14.15. Легко проверить, что эта гиперплоскость является искомой разделяющей границей: $\vec{w}^T\vec{d}_1 \approx 0\cdot 0 + (-0{,}71)\cdot 0 + (-0{,}71)\cdot 0 + \frac{1}{3}\cdot 0 + \frac{1}{3}\cdot 1{,}0 + \frac{1}{3}\cdot 0 = \frac{1}{3} > b$ (аналогично $\vec{w}^T\vec{d}_i > b$ для $2 \le i \le 3$) и $\vec{w}^T\vec{d}_4 = -1 < b$. Таким образом, документы из класса *c* лежат выше полуплоскости ($\vec{w}^T\vec{d} > b$), а документы из класса $\bar{c}$ — ниже полуплоскости ($\vec{w}^T\vec{d} < b$).

Таблица 14.1. Векторы и центроиды классов для данных из табл. 13.1

	Веса терминов					
Вектор	Chinese	Japan	Tokyo	Macao	Beijing	Shanghai
$\vec{d}_1$	0	0	0	0	1,0	0
$\vec{d}_2$	0	0	0	0	0	1,0
$\vec{d}_3$	0	0	0	1,0	0	0
$\vec{d}_4$	0	0,71	0,71	0	0	0
$\vec{d}_5$	0	0,71	0,71	0	0	0
$\vec{\mu}_c$	0	0	0	0,33	0,33	0,33
$\vec{\mu}_{\bar{c}}$	0	0,71	0,71	0	0	0

Критерием классификации на рис. 14.4 является евклидово расстояние. В качестве альтернативы можно использовать косинусную меру сходства.

$$\textit{Документ } d \textit{ принадлежит классу } c = \arg\max_{c'} \cos\left(\vec{\mu}(c'), \vec{v}(d)\right)$$

Как указано в разделе 14.1, два критерия классификации иногда приводят к разным результатам. Здесь мы привели вариант алгоритма Роккио, основанный на использовании евклидова расстояния, поскольку оно подчеркивает тесную связь с методом кластеризации *K* средних (K-means clustering), который будет описан в разделе 16.4.

Классификация Роккио является частным случаем метода Роккио для обратной связи по релевантности (см. раздел 9.1.1). Средним релевантных документов в соответствии с

наиболее важным компонентом вектора Роккио в методе обратной связи по релевантности (см. формулу (9.3)) является центроид "класса" релевантных документов. При описании классификации Роккио мы пропустили компонент запроса в формуле Роккио для обратной связи, поскольку в классификации текстов нет никаких запросов. Классификацию Роккио можно применять к $J > 2$ классам, в то время как метод Роккио для обратной связи по релевантности разработан для различения только двух классов, релевантного и нерелевантного.

Кроме соответствия гипотезе компактности, классы в классификации Роккио должны иметь приблизительную форму сфер с примерно одинаковыми радиусами. На рис. 14.3 черный квадрат непосредственно под границей между классами *UK* и *Kenya* больше подходит для класса *UK*, поскольку класс *UK* имеет более широкий разброс, чем класс *Kenya*. Однако алгоритм Роккио относит его к классу *Kenya*, потому что при классификации он игнорирует особенности распределения точек в классе и использует только расстояния до центроида.

Предположение о сферичности на рис. 14.5 также не выполняется. Класс "*a*" невозможно хорошо описать с помощью единственного прототипа, поскольку он состоит из двух кластеров. Алгоритм Роккио часто неверно распознает *мультимодальные классы* такого типа. Примером мультимодального класса в задачах текстовой классификации являются страны наподобие Бирмы, которая с 1989 года стала называться Мьянмой. Два кластера до и после смены названия не обязательно расположены близко друг к другу в векторном пространстве. Проблема мультимодальности возникает также в задачах обратной связи по релевантности (см. раздел 9.1.2).

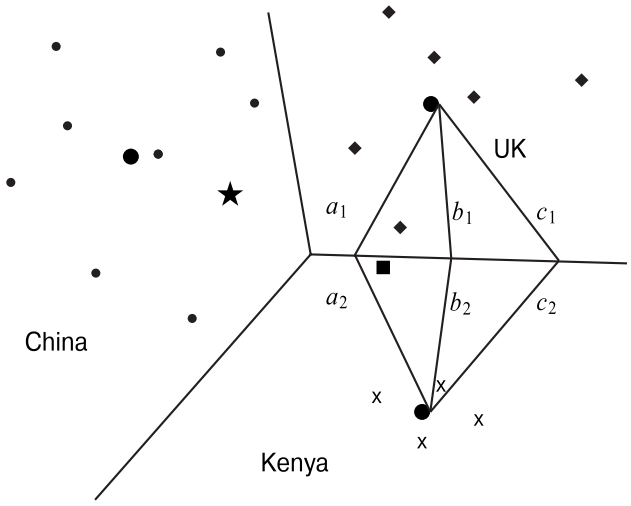


Рис. 14.5. Мультимодальный класс "a" состоит из двух разных кластеров (небольшие круги в верхней части рисунка с центрами в точках X). Классификация Роккио неправильно отнесет термин "o" к классу терминов "a", потому что он ближе к центроиду A класса "a", а не к центроиду B класса "b"

Еще одним примером, в котором классы редко представляют собой сферы с одинаковыми радиусами, является задача бинарной классификации. Большинство бинарных

классификаторов различают класс наподобие *China*, занимающий небольшую область в пространстве, и его дополнение, разбросанное по всему пространству. Предположение о равенстве радиусов привело бы к большому количеству ложно положительных решений. Следовательно, для большинства задач бинарной классификации необходимо уточнить решающее правило.

Документ d принадлежит классу c тогда и только тогда, когда $|\vec{\mu}(c)-\vec{v}(d)|<|\vec{\mu}(\bar{c})-\vec{v}(d)|-b$

Здесь b — положительная константа. Как и в алгоритме Роккио для обратной связи по релевантности, центроид "отрицательных" документов можно не использовать вообще, поэтому решающее правило можно упростить до $|\vec{\mu}(c)-\vec{v}(d)|<b'$, где b' — положительная константа.

В табл. 14.2 приведены оценки временной сложности классификации Роккио.[2] Сложность суммирования всех документов при вычислении вектора суммы равна $\Theta(|\mathbb{D}|L_{ave})$, а не $\Theta(|\mathbb{D}||V|)$, поскольку учитывать следует только ненулевые элементы. Сложность деления каждой суммы векторов на размер класса при вычислении центроида равна $\Theta(|V|)$. В целом время обучения линейно зависит от размера коллекции (см. упражнение 13.1). Таким образом, классификация Роккио и наивный байесовский метод имеют одинаковую временную сложность обучения.

Таблица 14.2. Оценки временной сложности обучения и тестирования для классификации Роккио (L_{ave} — среднее количество лексем в документе, L_a и M_a — количество лексем и типов в тестовом документе соответственно; временная сложность вычисления евклидова расстояния между центроидами классов и документов равна $\Theta(|\mathbb{C}|M_a)$)

Операция	*Временная сложность*
Обучение	$\Theta(\|\mathbb{D}\|L_{ave}+\|\mathbb{C}\|\|V\|)$
Тестирование	$\Theta(L_a+\|\mathbb{C}\|M_a)=\Theta(\|\mathbb{C}\|M_a)$

В следующем разделе мы опишем другой метод векторной классификации kNN, который лучше работает с несферическими и несвязными классами, а также с имеющими другие "неправильности".

? ***Упражнение 14.2*** **[*].** Покажите, что классификация Роккио может приписать документу метку, отличающуюся от его метки в обучающем классе.

14.3. Метод *k* ближайших соседей

В отличие от метода Роккио, *метод k ближайших соседей* (k nearest neighbor), или *kNN-классификация*, определяет разделяющие границы локально. В варианте 1NN каждый документ относится к определенному классу в зависимости от информации о его ближайшем соседе. В варианте kNN каждый документ относится к преобладающему классу ближайших соседей, где k —параметр метода. В основе метода kNN лежит факт,

[2] Оценка $\Theta(T)$ заменяется оценкой $\Theta(|\mathbb{D}|L_{ave})$, а длина тестовых документов ограничена.

что в соответствии с гипотезой компактности мы ожидаем, что тестовый документ d будет иметь такую же метку, как и обучающие документы в локальной области, окружающей документ d.

Разделяющие границы в методе 1NN представляют собой смежные сегменты *диаграммы Вороного* (Voronoi tesselation), показанной на рис. 14.6. Диаграмма Вороного для множества объектов разделяет пространство на ячейки, состоящие из точек, которые ближе к данному объекту, чем к другим. В нашем случае объектами являются документы. Диаграмма Вороного разделяет плоскость на $|\mathbb{D}|$ выпуклых многоугольников, каждый из который содержит соответствующий документ (и не содержит других), как показано на рис. 14.6, на котором выпуклый многоугольник есть выпуклая область двумерного пространства, ограниченная линиями.

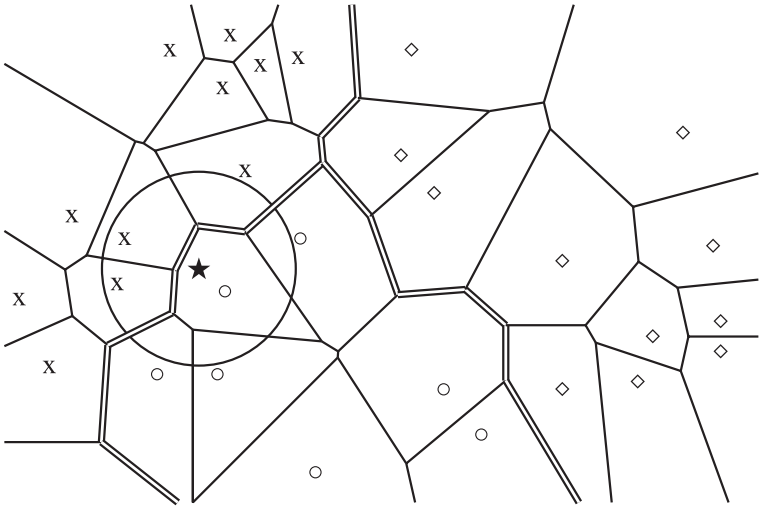


Рис. 14.6. Диаграмма Вороного и разделяющие границы (двойные линии) в методе 1NN. Показаны три класса: крестики, кружочки и ромбики

Для произвольного параметра $k \in \mathbb{N}$ в методе kNN рассмотрим область пространства, для которой множество k ближайших соседей остается одинаковым. Эта область также представляет собой выпуклый многоугольник, а пространство оказывается разделенным на выпуклые многоугольники, внутри каждого из которых множество k ближайших соседей является инвариантным (упражнение 14.11). [3]

Метод 1NN не очень устойчив. Классификация каждого текстового документа зависит от класса, к которому относится отдельный обучающий документ, который может иметь неверную метку или вообще быть нетипичным. Метод kNN при $k > 1$ является бо-

[3] Обобщением многоугольника на пространства более высоких размерностей является *многогранник*, т.е. область M-мерного пространства, ограниченная $(M-1)$-мерными гиперплоскостями. В M-мерных пространствах границы решений в методе kNN состоят из сегментов $(M-1)$-мерных полуплоскостей, образующих диаграмму Вороного, состоящую из выпуклых многогранников, построенных по обучающему множеству документов. Критерий отнесения документа к преобладающему среди его k ближайших соседей классу одинаково применим к M=2 (разбиение на многоугольники) и M>2 (разбиение на многогранники).

лее устойчивым. Он приписывает документы к превалирующему классу по k ближайшим соседям, случайным образом разрывая связи между ними.

Существует вероятностный вариант метода kNN. Можно оценить вероятность того, что документ принадлежит классу c, как долю k ближайших соседей в классе c. На рис. 14.6 приведен пример классификации при $k = 3$. Оценки вероятностей того, что документ, отмеченный звездочкой, принадлежит трем классам, таковы: (класс кружочков|звездочка) = 1/3, (класс крестиков|звездочка) = 2/3, (класс ромбиков|звездочка) = 0. Оценки метода 3NN ($\hat{P}_1$(класс кружочков|звездочка) = 1/3) и метода 1NN ($\hat{P}_1$(класс кружочков|звездочка) = 1) отличаются, поэтому метод 3NN отдает предпочтение классу крестиков, а метод 1NN — классу кружочков.

Параметр k в методе kNN часто выбирается на основании опыта или знаний о решаемой задаче классификации. Желательно, чтобы параметр k был нечетным, чтобы уменьшить вероятность "ничьей". Чаще всего выбираются значения $k = 3$ и $k = 5$, но используются и большие значения, между 50 и 100. В качестве альтернативы параметр k можно выбрать так, чтобы он гарантировал наилучшие результаты на отложенной части обучающего множества.

Можно также приписать веса "голосам" k ближайших соседей, используя их косинусную меру сходства. В этой схеме ранги классов вычисляются так.

$$\textit{ранг}(c,d) = \sum_{d' \in S_k} I_c(d') \cos\left(\vec{v}(d'), \vec{v}(d)\right)$$

Здесь S_k — множество k ближайших соседей документа d' и $I_c(d') = 1$ тогда и только тогда, когда документ d' принадлежит классу c и $I_c(d') = 0$ — в противном случае. Документ приписывается к классу с наибольшим рангом. Взвешивание с помощью меры сходства часто оказывается более точным, чем простое голосование. Например, если два класса имеют одинаковое количество соседей, то выбирается класс с более близкими соседями.

Алгоритм kNN представлен на рис. 14.7.

```
Train-kNN(ℂ, 𝔻)
1   𝔻′ ← Preprocess(𝔻)
2   k ← Select-k{ℂ,𝔻′}
3   return 𝔻′, k
Apply-kNN(ℂ,𝔻′, k, d)
1   S_k ← ComputeNearestNeighbors(𝔻′, k, d)
2   for each c_j ∈ ℂ
3   do  p_j ← |S_k ∩ c_j|/k
4   return arg max_j p_j
```

Рис. 14.7. Обучение (с предварительной обработкой) и тестирование по методу kNN. Величина p_j — это оценка вероятности $P(c_j|S_k) = P(c_j|d)$, а c_j — множество всех документов в классе c_j

Пример 14.2. Расстояния между тестовым документом и четырьмя обучающими документами в табл. 14.1 равны $\left|\vec{d}_1-\vec{d}_5\right|=\left|\vec{d}_2-\vec{d}_5\right|=\left|\vec{d}_3-\vec{d}_5\right|\approx 1{,}41$ и $\left|\vec{d}_4-\vec{d}_5\right|=0{,}0$. Следовательно, ближайшим соседом документа d_5 является документ d_4, и метод 1NN припишет документ d_5 к классу документа d_4, т.е. к классу $\overline{c}$.

14.3.1. Временная сложность и оптимальность метода kNN

Оценки временной сложности метода kNN приведены в табл. 14.3. Метод kNN довольно сильно отличается от других алгоритмов классификации. Обучение метода kNN сводится к простому определению параметра k и предварительной обработке документов. Фактически, если параметр k выбран заранее, а предварительная обработка документов не предусмотрена, в методе kNN вообще отсутствует фаза обучения. На практике предварительная обработка, например разбиение на лексемы (tokenization), является обязательной. Ее целесообразно провести один раз на этапе обучения, а не повторять каждый раз при классификации нового документа.

Таблица 14.3. Оценки временной сложности обучения и тестирования для классификации kNN (M_{ave} — средний размер лексикона коллекции)

kNN с предварительной обработкой обучающего множества	
Обучение	$\Theta(\lvert\mathbb{D}\rvert L_{ave})$
Тестирование	$\Theta(L_a+\lvert\mathbb{D}\rvert M_{ave}M_a)=\Theta(\lvert\mathbb{D}\rvert M_{ave}M_a)$
kNN без предварительной обработки обучающего множества	
Обучение	$\Theta(1)$
Тестирование	$\Theta(L_a+\lvert\mathbb{D}\rvert L_{ave}M_a)=\Theta(\lvert\mathbb{D}\rvert L_{ave}M_a)$

Временная сложность тестирования в методе kNN равна $\Theta(\lvert\mathbb{D}\rvert M_{ave}M_a)$. Она линейно зависит от размера обучающего множества, поскольку при классификации необходимо вычислить расстояние от тестового документа до каждого документа из обучающего множества. Продолжительность тестирования не зависит от количества классов J. Следовательно, при большом количестве классов J метод kNN имеет потенциальные преимущества.

В классификации по методу kNN не производится оценка ни одного параметра, как в методе Роккио (центроиды) или в наивном байесовском методе (априорные и условные вероятности). В методе kNN все экземпляры из обучающего множества просто запоминаются, а затем сравниваются с тестовым документом. По этой причине метод kNN также называют *обучением на основе запоминания* (memory-based learning) и *обучением на основе запоминания прецедентов* (instance-based learning). Для машинного обучения обычно желательно иметь как можно больше обучающих данных. Однако в методе kNN большие обучающие множества приводят к значительному снижению эффективности классификации.

Можно ли провести тестирование по методу kNN быстрее, чем $\Theta(|\mathbb{D}|M_{ave}M_a)$, или, игнорируя длину документов, эффективнее, чем $\Theta(|\mathbb{D}|)$. При небольших размерностях M существуют быстрые алгоритмы kNN (упражнение 14.12). При больших размерностях M разработаны приближенные методы, гарантирующие определенный диапазон ошибок при заданном уровне эффективности (раздел 14.7). Эти приближенные методы не прошли широкой апробации на задачах классификации текстов, поэтому пока неясно, могут ли они достичь эффективности, большей, чем $\Theta(|\mathbb{D}|)$, без значительной потери точности.

Читатели, возможно, заметили сходство между задачей поиска ближайшего соседа тестового документа и задачей поиска по произвольному запросу, целью которой являются документы, имеющие наибольшую схожесть с запросом (см. раздел 6.3.2). Фактически эти две проблемы представляют собой разные варианты задачи о k ближайших соседях и отличаются лишь относительной плотностью вектора тестового документа (десятки или сотни ненулевых элементов в методе kNN) и разреженностью вектора запроса (как правило, в задаче поиска по произвольному запросу количество ненулевых элементов меньше десяти). Для обеспечения эффективности произвольного поиска мы ввели обратный индекс (см. раздел 1.1). Можно ли создать эффективный метод kNN, используя аналогичный инвертированный индекс?

Инвертированный индекс ограничивает поиск только теми документами, которые имеют хотя бы один общий термин с запросом. Таким образом, в контексте метода kNN инвертированный индекс окажется эффективным, если тестовый документ не имеет общих терминов с большим количеством обучающих документов. Так ли это, зависит от конкретной задачи классификации. Если документы длинные, а список стоп-слов не используется, то экономия времени будет незначительной. Однако при коротких документах и длинном списке стоп-слов обратный индекс может сократить среднее время тестирования в десять и более раз.

Время поиска при использовании обратного индекса зависит от длины списка словопозиций терминов из запроса. Длина списков словопозиций сублинейно растет при увеличении размера коллекции, поскольку лексикон увеличивается в соответствии с законом Хипса (Heaps' law): если вероятность появления одних терминов возрастает, то вероятность появления других должна убывать. Однако большинство новых терминов не являются распространенными. Следовательно, сложность поиска при использовании инвертированного индекса остается равной $\Theta(T)$ (см. раздел 2.4.2), и если средняя длина документа со временем не изменяется, то $\Theta(T) = \Theta(|\mathbb{D}|)$.

Как будет показано в главе 15, качество классификации метода kNN близко к качеству наиболее точных методов обучения в классификации текстов (табл. 15.2). Мерой качества метода обучения является *уровень байесовской ошибки* (Bayes error rate), т.е. средний уровень ошибок классификаторов, обученных с помощью этого метода для решения конкретной задачи. Метод kNN не оптимален для проблем с ненулевой байесовской ошибкой, т.е. для проблем, при решении которых даже самый лучший из всех возможных классификаторов имеет ненулевую ошибку классификации. Ошибка метода 1NN асимптотически (при увеличивающемся обучающем множестве) ограничена удвоенным уровнем байесовской ошибки. Иначе говоря, если оптимальный классификатор имеет уровень ошибок, равный x, то асимптотический уровень ошибок метода 1NN равен $2x$. Это объясняется наличием шума — в разделе 13.5 приведен пример одной из

форм шума (шумовые признаки), хотя шум может принимать и другие формы. Источниками шума являются два компонента метода kNN: тестовый документ и ближайший обучающий документ. Эти источники являются аддитивными, поэтому суммарная ошибка метода 1NN вдвое превышает оптимальный уровень ошибки. Для задач с нулевым байесовским уровнем ошибки уровень ошибки метода 1NN при увеличении размера обучающего множества стремится к нулю.

? ***Упражнение 14.3.*** Объясните, почему метод kNN с мультимодальными классами работает лучше, чем метод Роккио.

14.4. Линейные и нелинейные классификаторы

В этом разделе мы покажем, что два метода обучения — наивный байесовский и Роккио — являются примерами линейных классификаторов, вероятно, наиболее важной группы классификаторов текстов, и противопоставим им нелинейные классификаторы. Для простоты рассмотрим лишь бинарные классификаторы и будем называть *линейным классификатором* (linear classifier) бинарный классификатор, принимающий решение о принадлежности документа классу с помощью сравнения линейной комбинации признаков с определенным пороговым значением.

На плоскости линейным классификатором является линия. На рис. 14.8 приведены примеры пяти линейных классификаторов. Общий функциональный вид этих классификаторов таков: $w_1x_1 + w_2x_2 = b$. Правило классификации с помощью линейного классификатора заключается в том, что документ относится к классу c, если $w_1x_1 + w_2x_2 > b$, и к классу $\overline{c}$, если $w_1x_1 + w_2x_2 \leq b$. Здесь $(x_1, x_2)^T$ — двумерное векторное представление документа, а $(w_1, w_2)^T$ — вектор параметров, которые вместе с числом b определяют разделяющую границу. Альтернативная геометрическая интерпретация линейного классификатора приведена на рис. 15.7.

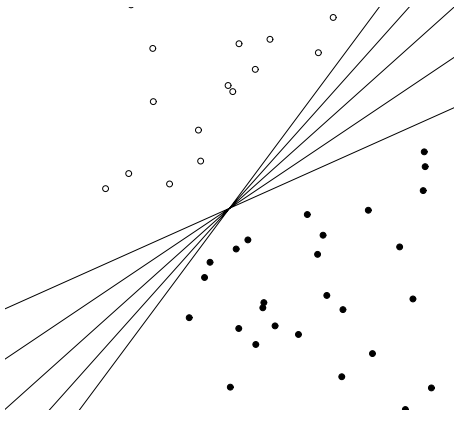


Рис. 14.8. Два линейно разделимых класса можно отделить друг от друга с помощью бесконечно большого количества гиперплоскостей

Двумерный линейный классификатор можно обобщить для пространств более высокой размерности, определив гиперплоскость с помощью формулы (14.2).

$$\vec{w}^T\vec{x} = b \tag{14.3}$$

Критерий решения имеет следующий вид: если $\vec{w}^T\vec{x} > b$, то класс c, а если $\vec{w}^T\vec{x} \le b$, то класс $\overline{c}$. Гиперплоскость, используемая в линейном классификаторе, называется *разделяющей гиперплоскостью* (decision hyperplane).

Соответствующий алгоритм линейной классификации в M-мерном пространстве показан на рис. 14.9. На первый взгляд, линейная классификация выглядит тривиальной. Однако обучение линейного классификатора связано с определенными трудностями, например, при определении параметров $\vec{w}$ и $\vec{b}$ по обучающему множеству. Если в качестве показателя качества метода обучения использовать эффективность обученного линейного классификатора на новых данных, то одни методы вычисляют намного более точные параметры, чем другие.

```
ApplyLinearClassifier(w⃗, b⃗, x⃗)
1    score ← Σ_{i=1}^{M} w_i x_i
2    if score > b
3      then return 1
4      else return 0
```

Рис. 14.9. Алгоритм линейной классификации

Теперь покажем, что алгоритм Роккио и наивный байесовский метод являются линейными классификаторами. Заметим, что в алгоритме Роккио вектор $\vec{x}$ лежит на разделяющей границе, если он лежит на одинаковом расстоянии от центроидов двух классов.

$$\left|\vec{\mu}(c_1) - \vec{x}\right| = \left|\vec{\mu}(c_2) - \vec{x}\right| \tag{14.4}$$

Простые алгебраические вычисления показывают, что это соответствует линейному классификатору с вектором нормали $\vec{w} = \vec{\mu}(c_1) - \vec{\mu}(c_2)$ и $b = 0{,}5\left(\left|\vec{\mu}(c_1)\right|^2 - \left|\vec{\mu}(c_2)\right|^2\right)$ (упражнение 14.15).

Линейность наивного байесовского метода следует из его решающего правила, в соответствии с которым выбирается класс c с наибольшим значением $\hat{P}(c|d)$ (см. рис. 13.2).

$$\hat{P}(c|d) \propto \hat{P}(c) \prod_{1 \le k \le n_d} \hat{P}(t_k|c)$$

Здесь n_d — количество лексем в документе, входящих в лексикон. Обозначив дополнение через $\overline{c}$, получим логарифм отношения шансов (log odds).

$$\log \frac{\hat{P}(c|d)}{\hat{P}(\overline{c}|d)} = \log \frac{\hat{P}(c)}{\hat{P}(\overline{c})} + \sum_{1 \le k \le n_d} \frac{\hat{P}(t_k|c)}{\hat{P}(t_k|\overline{c})} \tag{14.5}$$

Мы выбираем класс c, если отношение вероятностей больше единицы или, что эквивалентно, если логарифм отношения вероятностей (шанса) больше нуля. Легко видеть, что равенство (14.5) — это вариант равенства (14.3), если $w_i = \log\left[\hat{P}(t_i|c)/\hat{P}(t_i|\overline{c})\right]$, x_i — количество вхождений термина t_i в документ d и $b = -\log\left[\hat{P}(c)/\hat{P}(\overline{c})\right]$. Здесь индекс i,

изменяющийся от 1 до M, нумерует термины лексикона (а не координаты в документе d в отличие от индекса k (см. раздел 13.4.1)), а $\vec{x}$ и $\vec{w}$ — M-мерные векторы. Таким образом, в пространстве логарифмов наивный байесовский метод является линейным классификатором.

Пример 14.3. В табл. 14.4 описан линейный классификатор для категории *interest* в коллекции Reuters-21578 (см. раздел 13.6). Документ $\vec{d}_1$ "rate discount dlrs world" относится к классу *interest*, поскольку $\vec{w}^T\vec{d}_1 = 0{,}67\cdot 1+(-0{,}71)\cdot 1+(-0{,}35)\cdot 1 = 0{,}07 > 0 = b$. Документ $\vec{d}_2$ "prime dlrs" относится к дополнительному классу (не *interest*), поскольку $\vec{w}^T\vec{d}_2 = -0{,}01 \le b$. Для простоты мы используем бинарное векторное представление документа: если термин встречается в документе, вектор содержит единицу, а если нет — нуль.

Таблица 14.4. Линейный классификатор. Термины t_i и параметры w_i линейного классификатора для класса *interest* (в смысле interest rate **— процентная ставка) в коллекции Reuters-21578. Пороговое значение $b = 0$. Термины типа** dlr **и** world **имеют негативные веса, поскольку являются индикаторами конкурирующего класса *currency* (валюта)**

t_i	w_i	d_{1i}	d_{2i}	t_i	w_i	d_{1i}	d_{2i}
prime	0,70	0	1	dlrs	–0,71	1	1
rate	0,67	1	0	world	–0,35	1	0
interest	0,63	0	0	sees	–0,33	0	0
rates	0,60	0	0	year	–0,25	0	0
discount	0,46	1	0	group	–0,24	0	0
bundesbank	0,43	0	0	dlr	–0,24	0	0

На рис. 14.10 продемонстрирован пример *линейной задачи* (linear problem), которая по определению характеризуется тем, что базовые распределения $P(d|c)$ и $P(d|\overline{c})$ двух классов разделены линией. Эта линия называется *границей между классами* (class boundary). Граница между классами является "истинной" и отличается от разделяющей границы, которую метод обучения вычисляет в качестве ее приближения.

Как это обычно и бывает в классификации текстов, на рис. 14.10 есть *шумовые документы* (noise documents), отмеченные стрелочками. Эти документы недостаточно хорошо соответствуют общему распределению классов. В разделе 13.5 мы назвали шумовым признаком такой признак, включение которого в представление документа в среднем повышает ошибку классификации. Аналогично шумовым называется такой документ, включение которого в обучающее множество приводит к увеличению ошибки классификации. Интуитивно понятно, что базовое распределение разделяет пространство представления документов в основном на области с однородными метками классов. Документ, не соответствующий доминантному классу в определенной области, является шумовым.

Шумовые документы — это одна из причин, по которым обучение линейного классификатора представляет собой непростую задачу. Если при выборе разделяющей гиперплоскости уделить слишком большое внимание шумовым документам, то классификатор станет неточным на новых данных. И что еще более важно, обычно трудно опре-

делить, какие документы являются шумовыми и, следовательно, потенциально могут снизить точность классификации.

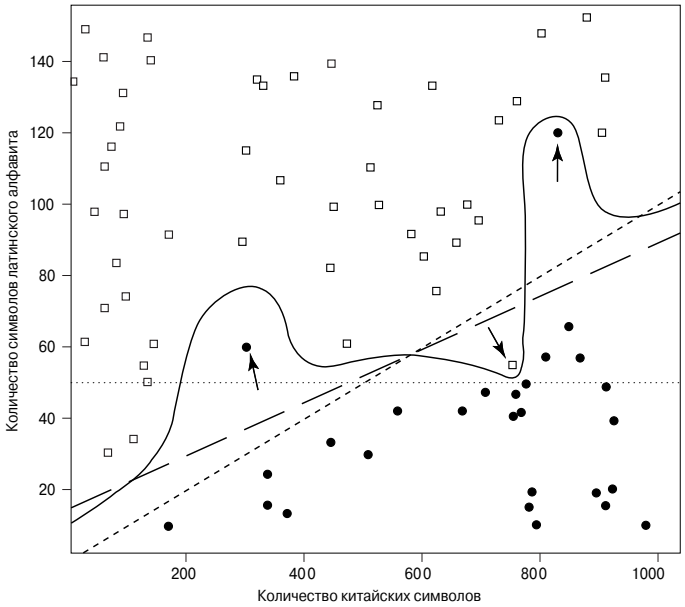


Рис. 14.10. Линейная задача с шумом. В этом гипотетическом сценарии классификации веб-страниц черными кружочками отмечены веб-страницы исключительно на китайском языке, а квадратиками — веб-страницы на смеси китайского и английского языков. Эти два класса разделены прямой линией (пунктирной), хотя существуют три шумовых документа (отмечены стрелочками)

Если существует гиперплоскость, идеально точно разделяющая два класса, то такие классы называются *линейно разделимыми* (linearly separable). На самом деле, если существует линейная разделимость, то количество линейных разделителей бесконечно, как показано на рис. 14.8 и в упражнении 14.4.

На рис. 14.8 продемонстрирована еще одна проблема, связанная с обучением линейного классификатора. Если задача является линейно разделимой, то необходим критерий для выбора разделяющей гиперплоскости среди многих гиперплоскостей, которые идеально разделяют обучающие данные. В принципе, одни из таких гиперплоскостей будут хорошо разделять новые данные, а другие — нет.

Примером нелинейного классификатора является метод kNN. Нелинейность классификатора kNN становится интуитивно ясной, если посмотреть на рис. 14.6. Разделяющая граница в методе kNN состоит из локально линейных сегментов, но в целом она имеет сложную форму, не совпадающую ни с линией на плоскости, ни с гиперплоскостью в пространствах более высокой размерности.

На рис. 14.11 показан еще один пример нелинейной задачи. Между распределениями $P(d|c)$ и $P(d|\overline{c})$ нет хорошего линейного разделения, поскольку в левом верхнем углу рисунка есть “круглый” анклав. Линейные классификаторы неверно классифицируют этот

анклав, в то время как метод kNN решает такие задачи с высокой точностью, если обучающее множество достаточно велико.

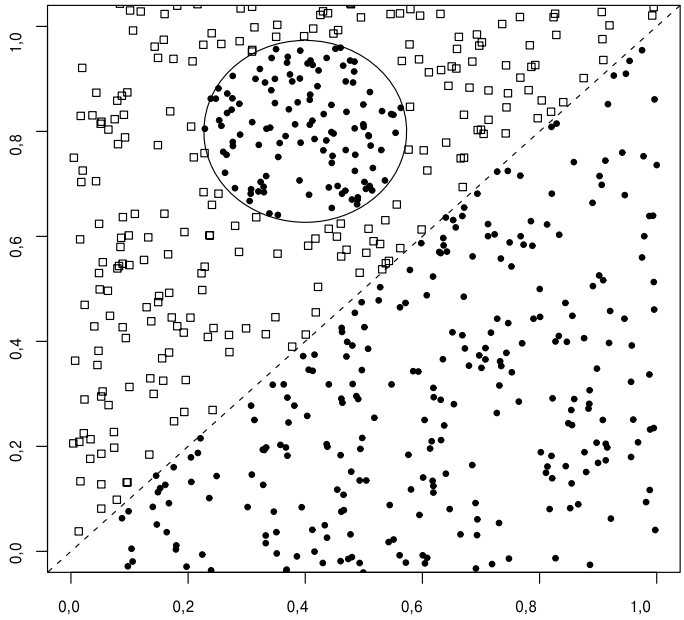


Рис. 14.11. Нелинейная задача

Если проблема носит нелинейный характер, а ее границы между классами невозможно хорошо аппроксимировать с помощью гиперплоскостей, то нелинейные классификаторы обычно оказываются точнее линейных. Если же задача является линейной, то лучше использовать более простой линейный классификатор.

? ***Упражнение 14.4.*** Докажите, что количество линейных разделителей двух классов либо бесконечно, либо равно нулю.

14.5. Классификация с несколькими классами

Бинарные линейные классификаторы можно расширить на вариант $J > 2$ классов. Выбор метода классификации в этом случае зависит от того, являются ли классы взаимоисключающими.

Классификация для классов, которые не являются взаимоисключающими, называется *многозначной* (any-of, multilabel, multivalue classification). В этом случае документ может принадлежать нескольким классам одновременно, одному классу или не принадлежать ни одному классу. Решение относительно одного класса не исключает решений относительно других классов. Иногда говорят, что классы *не зависят* друг от друга, но это неправильно; классы редко являются статистически независимыми. В терминах формальной постановки задачи классификации (формула (13.1)) можно сказать, что в задаче многозначной классификации происходит обучение J разных классификаторов γ_j, причем

каждый из классификаторов возвращает либо метку класса c_j, либо метку класса $\overline{c}_j$, т.е. $\gamma_j(d) \in \{c_j, \overline{c}_j\}$.

Решение задачи многозначной классификации с помощью линейных классификаторов очевидно.

1. Строим классификаторы для каждого класса, при этом обучающее множество состоит из набора документов, принадлежащих классу (положительные метки), и его дополнения (отрицательные метки).
2. Имея тестовый документ, применяем к нему каждый классификатор по отдельности. Решение одного классификатора не влияет на решение другого.

Еще одной разновидностью классификации с несколькими классами является *однозначная классификация* (one-of classification). В этом случае классы не пересекаются. Каждый документ должен принадлежать только одному из классов. Однозначная классификация называется также *мультиномиальной* (multinomial), *политомической* (polytomous) [4], *многоклассовой* (multiclass) или *классификацией с одной меткой* (single-label classification). С формальной точки зрения в этой классификации существует единственная функция классификации γ, областью значений которой является пространство $\mathbb{C}$, т.е. $\gamma(d) \in \{c_1, ..., c_J\}$. Например, метод kNN является (нелинейным) однозначным классификатором.

Истинно однозначные задачи классификации текстов встречаются реже, чем многозначные. Документы из классов *UK*, *China*, *poultry* и *coffee* могут быть релевантными нескольким темам одновременно, например если премьер-министр Великобритании (класс *UK*) посетил Китай (класс *China*), чтобы провести переговоры о торговле *кофе* (класс *coffee*) и *домашней птицей* (класс *poultry*).

Тем не менее мы часто будет принимать предположение об однозначной классификации, как показано на рис. 14.1, даже если классы на самом деле не являются взаимоисключающими. Для определения языка документа предположение об однозначной классификации является вполне обоснованным, поскольку большинство документов написано только на одном языке. В таких случаях наложение условия однозначности может повысить эффективность классификации, поскольку при этом исключаются ошибки, возникающие из-за того, что документ приписывается нескольким классам одновременно или ни одному классу вообще.

Как показано на рис. 14.12, J гиперплоскостей не разделяют пространство $\mathbb{R}^{|V|}$ на J непересекающихся областей. Следовательно, для решения задачи однозначной классификации с помощью бинарных линейных классификаторов мы должны применять их комбинацию. Проще всего ранжировать классы, а затем выбрать класс с наибольшим рангом. С геометрической точки зрения ранжирование сводится к вычислению расстояний от J разделяющих гиперплоскостей. Вероятность неправильно классифицировать документ, лежащий ближе к границе класса, выше, поэтому чем дальше документ от границы, тем выше вероятность, что он классифицирован правильно. В качестве альтернативы можно непосредственно вычислить доверительную оценку для ранга класса, например вероятность принадлежности классу. Этот алгоритм однозначной классификации с помощью линейных классификаторов можно описать следующим образом.

[4] Синонимом слова “политомический” является слово “полихотомический” (polychotomous).

1. Строим классификатор для каждого класса, причем обучающее множество состоит из набора документов, принадлежащих классу (положительные метки), и его дополнения (отрицательные метки).
2. К заданному тестовому документу применяем каждый классификатор по отдельности.
3. Относим тестовый документ к классу, имеющему
   - максимальный ранг или
   - максимальный доверительный уровень, или
   - максимальную вероятность.

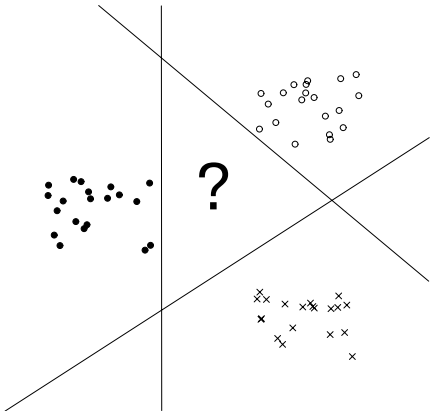


Рис. 14.12. J гиперплоскостей не разделяют пространство $\mathbb{R}^{|V|}$ на J непересекающихся областей

Важным инструментом анализа производительности классификатора при $J > 2$ классах является *матрица неточностей* (confusion matrix). Для каждой пары классов $<c_1, c_2>$ она показывает, сколько документов из класса c_1 ошибочно отнесены к классу c_2. В табл. 14.5 показаны результаты решения задачи, в которой классификатор должен был отличить три класса, посвященных финансам (*money-fx*, *trade* и *interest*), от трех классов сельскохозяйственной тематики (*wheat*, *corn* и *grain*), но сделал много ошибок внутри этих двух групп. Матрица неточностей помогает выявить возможности для повышения правильности работы системы. Например, чтобы устранить вторую по величине ошибку в табл. 14.5, можно попытаться ввести признаки, отличающие документы класса *wheat* от документов класса *grain*.

Таблица 14.5. Матрица неточностей для коллекции Reuters-21578. Например, четырнадцать документов из класса *grain* были ошибочно отнесены к классу *wheat* (Picca et al., 2006)

Приписанный класс / Истинный класс	*money-fx*	*trade*	*Interest*	*Wheat*	*corn*	*grain*
money-fx	95	0	10	0	0	0
trade	1	1	90	0	1	0
interest	13	0	0	0	0	0

Окончание табл. 14.5

Приписанный класс / Истинный класс	*money-fx*	*trade*	*Interest*	*Wheat*	*corn*	*grain*
wheat	0	0	1	34	3	7
corn	1	0	2	13	26	5
grain	0	0	2	14	5	10

? ***Упражнение 14.5.*** Создайте обучающее множество, состоящее из 300 документов, по 100 для каждого языка (например, английского, французского и испанского). Точно так же создайте тестовое множество, добавив в него 100 документов на четвертом языке. Проведите обучение однозначного и многозначного классификаторов на этом обучающем множестве и оцените их по тестовому множеству. Существуют ли какие-либо интересные различия между поведением этих классификаторов при решении поставленной задачи?

14.6. Компромисс между смещением и дисперсией

Нелинейные классификаторы мощнее линейных. Для некоторых задач существуют нелинейные классификаторы с нулевой ошибкой классификации, но нет аналогичных линейных классификаторов. Значит ли это, что для достижения оптимальной эффективности статистической классификации текстов всегда следует использовать нелинейные классификаторы?

Для ответа на этот вопрос в данном разделе описывается компромисс между смещением и дисперсией — одна из наиболее важных концепций в теории машинного обучения. Этот компромисс помогает объяснить, почему не существует оптимального метода обучения. Следовательно, выбор подходящего метода обучения является неотъемлемой частью решения задачи классификации текстов.

На протяжении этого раздела мы используем линейные и нелинейные классификаторы в качестве прототипов “менее мощного” и “более мощного” методов обучения соответственно. Такой подход является упрощением по нескольким причинам. Во-первых, многие нелинейные модели в частных случаях сводятся к линейным. Например, нелинейный метод обучения, такой как kNN, в некоторых случаях порождает линейный классификатор. Во-вторых, существуют нелинейные модели, которые проще линейных. Например, квадратный многочлен с двумя параметрами проще, чем 10 000-мерный линейный классификатор. В-третьих, сложность обучения на самом деле не является свойством классификатора, поскольку существует много факторов обучения (например, выбор признаков (см. раздел 13.5), регуляризация и ограничения, как, например, максимизация зазора между классами в главе 15), которые повышают или понижают точность метода обучения независимо от типа классификатора, т.е. окончательный результат обучения зависит не только от того, является ли классификатор линейным или нелинейным. Описание компромисса между смещением и дисперсией с учетом указанных факторов изложено в публикациях, перечисленных в разделе 14.7. В этом разделе линейные и нелинейные классификаторы служат в качестве примера менее мощного и более мощного методов обучения в задачах классификации текстов.

Сначала следует уточнить цель классификации текстов. В разделе 13.1 мы сказали, что хотим минимизировать ошибку классификации на тестовом множестве. При этом

было сделано неявное предположение, что обучающие и тестовые документы генерируются в соответствии с одним и тем же распределением. Обозначим это распределение как $P(<d, c>)$, где d — это документ, а c — метка его класса. Примеры порождающих моделей, представляющих вероятность $P(<d, c>)$ в виде произведения двух вероятностей, $P(c)$ и $P(d|c)$, приведены на рис. 13.4 и 13.5. Порождающие модели также были продемонстрированы на рис. 14.10 и 14.11 для пар $<d, c>$, где $d \in \mathbb{R}^2$ и $c \in$ {квадрат, черный кружок}.

В этом разделе вместо количества правильно классифицированных документов (или, что эквивалентно, уровня ошибок на тестовых документах) в качестве показателя качества классификации мы будем использовать оценку, учитывающую неустранимую неопределенность классификации. Во многих задачах классификации текстов одно и то же представление документа можно получить из документов, принадлежащих разным классам. Это происходит потому, что документы из разных классов могут отображаться в одно и то же представление документа. Например, документы, состоящие из предложений *China sues France* (Китай подал судебный иск к Франции) и *France sues China* (Франция подала судебный иск к Китаю), в модели "мешка слов" отображаются в одно и то же представление d' = {`China`, `France`, `sues`}. Однако классу c' = *юридические действия, предпринятые Францией*, который может быть определен, например, по запросу какого-нибудь специалиста по международной торговле, релевантен только второй документ.

Для упрощения вычислений при оценке классификатора мы не будем подсчитывать количество ошибок на тестовом множестве, а обратим внимание на то, насколько хорошо классификатор оценивает условную вероятность $P(c|d)$ для документа из класса. В приведенном выше примере $P(c'|d') = 0{,}5$.

Цель классификации текстов теперь заключается в поиске классификатора γ, который после усреднения по всем документам d гарантировал бы оценку $\gamma(d)$, как можно более близкую к вероятности $P(c|d)$. Этот показатель мы будем измерять с помощью среднеквадратической ошибки.

$$MSE(\gamma) = E_d\left[\gamma(d) - P(c|d)\right]^2 \tag{14.6}$$

Здесь E_d — математическое ожидание $P(d)$. Среднеквадратическая ошибка приписывает частичное доверие решениям классификатора γ, которые близки к правильным, даже если не полностью совпадают с ними.

Классификатор γ называется *оптимальным* относительно распределения $P(<d, c>)$, если он минимизирует среднеквадратическую ошибку MSE(γ).

Минимизация среднеквадратической ошибки — желательное свойство *классификатора*. Кроме того, нам необходим критерий для *метода обучения*. Напомним, что методом обучения Γ называется функция, аргументом которой является размеченное обучающее множество $\mathbb{D}$, а значением — классификатор γ.

Нашей целью является такой метод обучения Γ, который в среднем по всем обучающим множествам создает классификатор γ с минимальной среднеквадратической ошибкой (MSE). Это требование можно формализовать с помощью *ошибки обучения* (learning error).

$$\text{Ошибка обучения } (\Gamma) = E_{\mathbb{D}}[\text{MSE}(\Gamma(\mathbb{D}))] \tag{14.7}$$

Здесь $E_{\mathbb{D}}$ — математическое ожидание на размеченных обучающих множествах. Для простоты предположим, что обучающие множества имеют фиксированный размер, тогда распределение $P(<d, c>)$ определяет распределение $P(\mathbb{D})$ по обучающим множествам.

Выбор метода обучения в задаче статистической классификации текстов можно осуществлять на основе ошибки обучения. Метод обучения Γ является *оптимальным* для распределения $P(\mathbb{D})$, если он минимизирует ошибку обучения.

$$
\begin{aligned}
&E[x-\alpha]^2 = Ex^2 - 2Ex\alpha + \alpha^2 = \\
&= (Ex)^2 - 2Ex\alpha + \alpha^2 + Ex^2 - 2(Ex)^2 + (Ex)^2 = \\
&= [Ex-\alpha]^2 - E2x(Ex) + E(Ex)^2 = \\
&= [Ex-\alpha]^2 + E[x-Ex]^2 \qquad (14.8)
\end{aligned}
$$

$$
\begin{aligned}
&E_{\mathbb{D}}E_d[\Gamma_{\mathbb{D}}(d) - P(c|d)]^2 = E_dE_{\mathbb{D}}[\Gamma_{\mathbb{D}}(d) - P(c|d)]^2 = \\
&= E_d[[E_{\mathbb{D}}\Gamma_{\mathbb{D}}(d) - P(c|d)]^2 + E_{\mathbb{D}}[\Gamma_{\mathbb{D}}(d) - E_{\mathbb{D}}\Gamma_{\mathbb{D}}(d)]^2] \qquad (14.9)
\end{aligned}
$$

Рис. 14.13. Арифметические преобразования при разложении "смещение–дисперсия". При выводе формулы (14.9) мы ввели обозначения $\alpha = P(c|d)$ и $x = \Gamma_{\mathbb{D}}(d)$ в формуле (14.8)

Для упрощения записи заменим символы $\Gamma(\mathbb{D})$ обозначением $\Gamma_{\mathbb{D}}$. Тогда формулу (14.7) можно переписать следующим образом.

$$
\begin{aligned}
&\text{Ошибка обучения } (\Gamma) = E_{\mathbb{D}}[\mathrm{MSE}(\Gamma(\mathbb{D}))] = \\
&\qquad = E_{\mathbb{D}}E_d[\Gamma_{\mathbb{D}}(d) - P(c|d)]^2 + \qquad (14.10)\\
&\qquad = E_d[\text{смещение}(\Gamma, d) + \text{дисперсия}(\Gamma, d)], \qquad (14.11)\\
&\text{смещение}(\Gamma, d) = [P(c|d) - E_{\mathbb{D}}\Gamma_{\mathbb{D}}(d)]^2, \qquad (14.12)\\
&\text{дисперсия}(\Gamma, d) = E_{\mathbb{D}}[\Gamma_{\mathbb{D}}(d) - E_{\mathbb{D}}\Gamma_{\mathbb{D}}(d)]^2 \qquad (14.13)
\end{aligned}
$$

Здесь эквивалентность формул (14.10) и (14.11) следует из формулы (14.9), показанной на рис. 14.13. Обратите внимание на то, что документ d и множество $\mathbb{D}$ не зависят друг от друга. В принципе, для случайного документа d и случайного обучающего множества $\mathbb{D}$ множество $\mathbb{D}$ может не содержать помеченный экземпляр документа d.

Смещение (bias) — это квадрат разности между истинной условной вероятностью $P(c|d)$ того, что документ d принадлежит классу c, и $\Gamma_{\mathbb{D}}(d)$ — прогнозной оценкой, полученной с помощью обученного классификатора и усредненной по всем обучающим множествам. Если метод обучения порождает плохие классификаторы, то смещение велико. Если же 1) классификаторы хорошие или 2) разные обучающие множества порождают ошибки на разных документах или 3) разные обучающие множества порождают положительные и отрицательные ошибки на одних и тех же документах, но их среднее равно нулю, то смещение мало. Если выполняется одно из перечисленных выше условий, то $E_{\mathbb{D}}\Gamma_{\mathbb{D}}(d)$, т.е. математическое ожидание по всем обучающим множествам, близко к вероятности $P(c|d)$.

Линейные методы, такие как метод Роккио и наивный байесовский метод, на нелинейных задачах имеют большое смещение, поскольку они могут моделировать только один вид границ между классами — гиперплоскости. Если порождающая модель $P(<d, c>)$ имеет сложную нелинейную границу между классами, то смещение в равенстве (14.11) будет велико, поскольку большое количество точек будет классифицировано неправильно. Например, круглый анклав на рис. 14.11 не учитывается линейной моделью и может породить большое количество ошибок.

Смещение можно интерпретировать как результат знаний о предметной области (или их недостаток), встроенных в классификатор. Если известно, что истинная граница между классами является линейной, то, вероятнее всего, именно метод обучения, порождающий линейные классификаторы, будет успешнее, чем нелинейный метод. Однако если истинные границы между классами являются нелинейными и мы по ошибке выбрали линейный классификатор, то средняя точность классификации будет низкой.

Нелинейные методы, такие как kNN, имеют небольшое смещение. На рис. 14.6 показано, что границы между классами в методе kNN весьма изменчивы и зависят от распределения документов в обучающем множестве. В результате каждый документ имеет шанс быть классифицированным правильно на некотором обучающем множестве. Следовательно, средняя оценка $E_{\mathbb{D}}\Gamma_{\mathbb{D}}(d)$ близка к вероятности $P(c|d)$, а смещение меньше, чем у линейного метода обучения.

Дисперсия (variance) — это вариация оценок линейных классификаторов, квадрат разностей между оценкой $\Gamma_{\mathbb{D}}(d)$ и ее средним значением $E_{\mathbb{D}}\Gamma_{\mathbb{D}}(d)$, усредненный по всем обучающим множествам. Если разные обучающие множества $\mathbb{D}$ порождают совершенно разные классификаторы $\Gamma_{\mathbb{D}}$, то дисперсия велика. Если же обучающее множество мало влияет на классификацию $\Gamma_{\mathbb{D}}$, независимо от того, правильной она является или неправильной, то дисперсия мала. Дисперсия оценивает, насколько противоречивыми являются решения, независимо от того, правильные они или неправильные.

Линейные методы обучения имеют небольшую дисперсию, поскольку большинство случайно выбранных обучающих множеств порождает близкие разделяющие гиперплоскости. Разделяющие линии, порожденные линейными методами обучения на рис. 14.10 и 14.11, мало отклоняются от истинных границ между классами при изменении обучающих множеств, но на классификацию подавляющего большинства документов (за исключением документов, близких к границам между классами) это не влияет. Анклав в форме круга на рис. 14.11 будет постоянно классифицироваться неправильно.

Нелинейные методы, такие как kNN, имеют высокую дисперсию. Рис. 14.6 наглядно показывает, что метод kNN может моделировать очень сложные границы между двумя классами. По этой причине он очень чувствителен к шумовым документам, например, показанным на рис. 14.10. В результате дисперсия в равенстве (14.11) для метода kNN оказывается большой. Тестовые документы иногда классифицируются неверно (если он находится недалеко от шумовых документов), а иногда верно (если в его окрестности нет шумовых документов). Это приводит к большой вариации результатов при переходе от одного обучающего множества к другому.

Методы обучения с большой дисперсией подвержены *переобучению* (overfitting) по обучающим выборкам. Цель классификации — настроить классификатор на обучающих выборках, чтобы учесть основные свойства базового распределения $P(<d, c>)$. Однако если происходит переобучение, то результат включает в себя шумовую информацию. Переобучение увеличивает среднеквадратическую ошибку и часто порождает проблемы при использовании методов обучения с большой дисперсией.

Дисперсию можно также интерпретировать как *меру сложности модели* (model complexity), или *емкость запоминания* (memory capacity) метода обучения, т.е. насколько хорошо он запоминает характеристики обучающего множества, а затем применяет их к новым данным. Эта емкость соответствует количеству независимых параметров, доступных для подгонки к обучающему множеству. Каждое множество соседей S_k в методе kNN порождает отдельное независимое решение о классификации документа. В этом

случае параметром является оценка $\hat{P}(c|S_k)$, как показано на рис. 14.7. Таким образом, емкость метода kNN ограничена только объемом обучающего множества. Он может запоминать произвольно большие обучающие множества. В противоположность этому количество параметров в методе Роккио фиксировано — J параметров по каждой размерности, один для каждого центроида — и не зависит от размера обучающего множества. Классификатор Роккио (т.е. определяющие его центроиды) не может "запомнить" тонкие детали распределения документов в обучающем множестве.

В соответствии с равенством (14.7) наша цель — выбрать метод обучения, минимизирующий ошибку обучения. Основную идею, которая отражена в формуле (14.11), можно кратко сформулировать следующим образом: ошибка обучения представляет собой сумму смещения и дисперсии, т.е. состоит из двух компонентов, которые невозможно минимизировать одновременно. При сравнении двух методов обучения, Γ_1 и Γ_2, в большинстве случаев оказывается, что у одного из них больше смещение и меньше дисперсия, а у другого — меньше смещение и больше дисперсия. Таким образом, выбор метода обучения не сводится к поиску метода, устойчиво порождающего качественные классификаторы по обучающим множествам (небольшая дисперсия) или настраивающего их на решение задач со сложными границами между классами (небольшое смещение). Вместо этого необходимо взвесить относительные преимущества смещения и дисперсии для нашей прикладной задачи и на основании этой информации принять решение. Этот компромисс называется *компромиссом между смещением и дисперсией* (bias-variance tradeoff).

На рис. 14.10 приведена иллюстрация, хотя и несколько искусственная, но полезная для понимания природы этого компромисса. Допустим, что некий текст на китайском языке содержит английские слова, набранные буквами латинского алфавита, например `CPU`, `ONLINE` и `GPS`. Рассмотрим задачу, в которой необходимо отличить веб-страницы, созданные исключительно на китайском языке, от страниц, содержащих смесь китайских и английских слов. Поисковая система предложит пользователям, не владеющим английским (но понимающим смысл таких заимствований, как `CPU`), возможность отфильтровать документы на смеси языков. Для решения этой задачи предлагается использовать два признака: количество букв латинского алфавита и количество китайских символов на веб-странице. Как указывалось ранее, в соответствии с распределением $P(<d, c>)$ порождающей модели большинство сгенерированных смешанных (соответственно, китайских) документов лежит выше (соответственно, ниже) пунктирной линии, но существует несколько шумовых документов. На рис. 14.10 показаны три классификатора.

- **Классификатор по одному признаку.** Ему соответствует горизонтальная линия, состоящая из точек. Этот классификатор использует только один признак — количество букв латинского алфавита. Если метод обучения минимизирует количество неправильных ответов в обучающем множестве, то положение горизонтальной разделяющей границы слабо зависит от различий между обучающими выборками (т.е. от шумовых документов). Метод обучения, порождающий такой тип классификатора, имеет небольшую дисперсию, но его смещение велико, так как он последовательно будет неправильно классифицировать квадратики в левом нижнем углу и черные кружочки (документы, содержащие более 50 латинских букв).
- **Линейный классификатор.** Ему соответствует пунктирная линия с длинными черточками. Этот метод обучения имеет небольшое смещение; неправильно

будут классифицированы только шумовые документы и, возможно, несколько документов, прилегающих к границе между двумя классами. Дисперсия этого метода выше, чем дисперсия у классификатора по одному признаку, но остается небольшой. Пунктирная линия с длинными черточками ненамного отклоняется от истинной границы между двумя классами, как практически все линейные разделяющие границы, построенные по обучающим множествам. Таким образом, только несколько документов (прилегающих к границе) будут классифицированы неправильно.

- **Классификатор с идеальной подгонкой к обучающему множеству** ("fit-training-set-perfectly" classifier). Ему соответствует сплошная линия. В данном случае обучающий метод построил разделяющую границу, идеально разделяющую классы в обучающем множестве. Этот метод имеет наименьшее смещение, поскольку ни один документ не классифицирован неправильно — классификатор иногда правильно классифицирует даже шумовые документы в тестовом множестве. Однако дисперсия этого метода велика. Поскольку шумовые документы могут произвольно переходить через разделяющую границу, тестовые документы, близкие к шумовым в обучающем множестве, будут классифицированы неправильно, что нехарактерно для линейного метода обучения.

Возможно, читатели удивятся, узнав, что многие хорошо известные методы классификации текстов являются линейными. Некоторые из этих методов, в частности линейный метод опорных векторов (linear SVM), регуляризированная логистическая регрессия и регуляризированная линейная регрессия, относятся к категории наиболее эффективных методов. Понять причины их успеха позволяет компромисс между смещением и дисперсией. Типичные классы в задачах классификации текстов имеют сложную структуру, и маловероятно, что их границы являются линейными. Однако в пространствах высокой размерности, которые характерны для классификации текстов, эти интуитивные представления оказываются неверными. По мере увеличения размерности вероятность линейной разделимости быстро возрастает (упражнение 14.17). Таким образом, линейные модели в пространствах большой размерности являются довольно мощными, несмотря на свою линейность. Несмотря на то что более мощные нелинейные методы обучения способны моделировать разделяющие границы, сложность которых намного превышает сложность гиперплоскости, они более чувствительны к шумам, содержащимся в обучающих данных. Иногда, когда объем обучающих данных велик, нелинейные методы обучения работают лучше линейных, но далеко не во всех случаях.

? ***Упражнение 14.6.*** Какой из трех векторов, изображенных на рис. 14.14 ($\vec{a}$, $\vec{b}$ или $\vec{c}$), 1) больше других похож на вектор $\vec{x}$ в соответствии с мерой сходства на основе скалярного произведения, 2) больше других похож на вектор $\vec{x}$ в соответствии с косинусной мерой сходства и 3) ближе других к вектору $\vec{x}$ в соответствии с евклидовым расстоянием.

Упражнение 14.7. Загрузите коллекцию Reuters-21578, а затем обучите и протестируйте классификаторы Роккио и kNN на классах *aquisition*, *corn*, *crude*, *earn*, *grain*, *interest*, *money-fx*, *ship*, *trade* и *wheat*. Используйте подмножество ModApte. Можете воспользоваться одним из программных пакетов, реализующих методы Роккио и kNN, например Bow toolkit (McCallum, 1996).

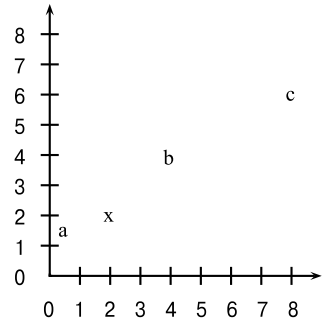


Рис. 14.14. Пример различий между евклидовым расстоянием, схожестью по скалярному произведению и косинусной мерой сходства. Векторы: $\vec{a} = (0{,}5 \quad 1{,}5)^T$, $\vec{x} = (2 \quad 2)^T$, $\vec{b} = (4 \quad 4)^T$ *и* $\vec{a} = (8 \quad 6)^T$

Упражнение 14.8. Загрузите коллекцию 20 Newsgroups (ссылка на раздел), а затем обучите и протестируйте классификаторы Роккио и kNN на этих 20 классах.

Упражнение 14.9. Покажите, что разделяющие границы в классификации Роккио, как и в методе kNN, соответствуют диаграмме Вороного.

***Упражнение 14.10* [*].** Временная сложность вычисления расстояния между "плотным" центроидом и разреженным вектором в наивной реализации, при которой выполняется итерация по всем M размерностям, составляет $\Theta(M)$. Основываясь на равенстве $\sum(x_i - \mu_i)^2 = 1{,}0 + \sum\mu_i^2 - 2\sum x_i\mu_i$ и предполагая, что $\sum\mu_i^2$ можно вычислить заранее, напишите алгоритм, временная сложность которого составляет $\Theta(M_a)$, где M_a — количество разных терминов в тестовом документе.

***Упражнение 14.11* [***].** Докажите, что область плоскости, состоящая из всех точек, имеющих одних и тех же k соседей, является выпуклым многоугольником.

Упражнение 14.12. Разработайте алгоритм, выполняющий эффективный поиск по методу 1NN по одной размерности (эффективность оценивается относительно количества документов N). Какова временная сложность этого алгоритма?

***Упражнение 14.13* [***].** Разработайте алгоритм, выполняющий эффективный поиск по методу 1NN по двум размерностям за полиномиальное по N время.

***Упражнение 14.14* [***].** Можно ли разработать точный эффективный алгоритм по методу 1NN при очень большом M, следуя идеям, использованным при выполнении предыдущего упражнения?

Упражнение 14.15. Покажите, что равенство (14.4) определяет гиперплоскость с параметрами $\vec{w}(c_1) - \vec{\mu}(c_2)$ и $b = 0{,}5 * \left(\left|\vec{\mu}(c_1)\right|^2 + \left|\vec{\mu}(c_2)\right|^2\right)$.

Упражнение 14.16. Можно легко построить неразделимые множества данных в пространстве большой размерности, просто погрузив одно множество в другое, как показано на рис. 14.15. Рассмотрите этот вариант в трехмерном пространстве, а затем измените положение точек на небольшую величину в случайном направле-

нии. Можно ли ожидать, что полученная конфигурация окажется линейно разделимой? Насколько вероятным является неразделимое множество, состоящее из $m<<M$ точек в M-мерном пространстве?

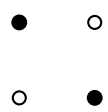


Рис. 14.15. Простое неразделимое множество точек

Упражнение 14.17. Представьте себе два класса и покажите, что доля неразделимых вершин гиперкуба при увеличении M уменьшается, например при $M = 1$ доля неразделимых вершин равна нулю, при $M = 2$ — 2/16. Один из вариантов неразделимых вершин при M = 2 показан на рис. 14.15. Другой вариант можно построить, применив зеркальное отражение. Решите эту задачу аналитически или с помощью моделирования.

Упражнение 14.18. Несмотря на схожесть наивного байесовского метода и линейных классификаторов на основе модели векторного пространства, представлять вектор частот (представление документа в наивном байесовском методе) в непрерывном векторном пространстве нецелесообразно. При этом существует формализация наивного байесовского метода, которая аналогична методу Роккио. Покажите, что наивный байесовский метод приписывает документ к классу (представленному в виде вектора параметров), имеющему наименьшую дивергенцию Кульбака–Лейблера (раздел 12.4) с документом (представленным в виде вектора частот как в разделе 13.4.1, значения нормированы таким образом, чтобы в сумме давать единицу).

14.7. Библиография и рекомендации для дальнейшего чтения

Как указывалось в главе 9, обратная связь по релевантности была предложена Роккио (Rocchio, 1971). Йоахимс (Joachims, 1997) провел вероятностный анализ этого метода. В 1990-х годах классификация Роккио широко использовалась в проекте TREC (Buckey et al., 1994*a*, *b*; Voorhees and Harman, 2005). Изначально этот метод использовался для поиска по постоянному запросу (routing). При таком типе поиска документы просто получают ранги в соответствии со своей релевантностью классу, но не приписываются к нему. Первые работы по *фильтрации* (filtering), представляющей собой вариант истинной классификации, в ходе которой принимается решение о приписывании документа к определенному классу, были опубликованы Иттнером и др. (Ittner et al., 1995) и Шапире и др. (Schapire et al., 1998). Понятие поиска по постоянному запросу, использованное здесь, не должно вводить читателей в заблуждение. Под этим также понимают распределение электронных документов подписчикам, т.е. так называемую *пассивная модель получения обновлений* (push model). В противоположность ей *в активной модели получения обновлений* (pull model) каждая передача документа пользователю инициируется самим пользователем, например, с помощью поиска или выбора из списка документов на новостном веб-сайте.

Некоторые авторы ограничивают термин *классификация Роккио* только задачами с двумя классами, а варианты метода Роккио с несколькими классами называют *классифика-*

цией, основанной на кластерах (cluster-based) (Iwayama and Tokunaga, 1995), или *классификацией, основанной на центроидах* (centroid-based classification)(Han and Karypis, 2000; Tan and Cheng, 2007).

Более подробное описание метода kNN можно найти в работе Хасти и др. (Hastie et al., 2001), в которой, помимо прочего, описаны способы настройки параметра *k*. Примером приближенного ускоренного алгоритма kNN является метод локального хеширования (Andoni et al., 2006). Кляйнберг (Kleinberg, 1997) разработал приближенный kNN-алгоритм, временная сложность которого составляла $\Theta((M\log^2 M)(M+\log N))$, где M — размерность пространства, а N — количество точек. Однако эта скорость была достигнута за счет экспоненциальных требований к памяти: $\Theta((N\log^2 M)^{2M})$. Обзор методов kNN в пространствах большой размерности можно найти в работе Индыка (Indyk, 2004). Ранние работы по применению метода kNN для классификации текстов мотивировались доступностью параллельной аппаратной архитектуры (Creecy et al., 1992). Для ускорения классификации kNN Янг (Yang, 1994) использовал инвертированный индекс. Оптимальность метода 1NN (асимптотическая оценка, вдвое превышающая байесовский уровень ошибок) доказана Кавером и Хартом (Cover and Hart, 1967).

Эффективность классификации Роккио и метода kNN сильно зависит от тщательной настройки параметров (в частности, параметра b' в методе Роккио и параметра k в методе kNN), конструирования признаков (раздел 15.3) и выбора признаков (раздел 13.5). Бакли и Солтон (Buckley and Salton, 1995), Шапире и др. (Schapire et al., 1998), Янг и Кисиль (Yang and Kisiel, 2003), а также Москитти (Moschitti, 2003) обращались к этой проблеме для метода Роккио, а Аулт и Янг (Ault and Yang, 2002) — для метода kNN. Заврель и др. (Zavrel et al., 2000) сравнили разные методы выбора признаков для метода kNN.

Компромисс между смещением и дисперсией был введен Германом и др. (German et al., 1992). В разделе 14.6 показан вывод оценки среднеквадратической ошибки $\mathrm{MSE}(\gamma)$, но этот компромисс применим ко многим функциям потерь (см. Friedman (1997) и Dimingos (2000)). Шютце и др. (Schütze et al., 1995), а также Льюис и др. (Lews et al., 1996) обсудили применение линейных классификаторов к текстам, а Хасти и др. (Hastie et al., 2001) — в целом. Читатели, заинтересовавшиеся алгоритмами, упомянутыми, но не описанными в этой главе, могут обратиться к книгам Бишопа (Bishop, 2006) по нейронным сетям, Хасти и др. (Hastie et al., 2001) — по линейной и логистической регрессии, а также Мински и Пейперта (Minsky and Papert, 1988) — по персептронам. Анагностопулос и др. (Anagnostopulos et al., 2006) показали, что с помощью инвертированного индекса можно повысить эффективность любого линейного классификатора, при условии, что этот классификатор останется эффективным при обучении на небольшом количестве признаков, полученном в результате выбора признаков (feature selection).

Мы представили лишь простейший метод сочетания бинарных классификаторов для создания однозначного классификатора. Другой важный метод основывается на использовании кодов исправления ошибок (erro-correcting codes), в которых для каждого документа конструируется отдельный вектор решений разных бинарных классификаторов. Вектор решения для тестового документа затем “корректируется” на основе распределения векторов решения в обучающем множестве. При выработке окончательного решения эта процедура учитывает информацию обо всех бинарных классификаторах и их корреляциях (Dietterich and Bakiri, 1995). Камрави и Маккалум (Chamrawi and McCalum, 2005) также использовали зависимости между классами для разработки метода многозначной классификации. Общий подход к сочетанию бинарных классификаторов предложили Олвейн и др. (Allweiun et al., 2000).

Глава 15

Метод опорных векторов и машинное обучение на документах

В результате двадцатилетних интенсивных научных исследований в области машинного обучения, направленных на повышение качества классификаторов, появилось новое поколение методов, в частности — методы опорных векторов (Support Vector Machines — SVM), бустинга решающих деревьев решений (boosted decision trees), регуляризированной логистической регрессии (regularized logistic regression), нейронных сетей (neural networks) и случайных лесов (random forests). Многие из этих методов, включая метод опорных векторов, описанный в этой главе, с успехом применялись для решения задач информационного поиска, в частности при классификации текстов. Метод опорных векторов представляет собой разновидность классификатора "с широким зазором": он относится к методам машинного обучения, основанным на модели векторного пространства, цель которых — найти разделяющие поверхности между классами, максимально удаленные от всех точек обучающего множества (возможно, проигнорировав некоторые точки как выбросы или шум).

Сначала мы опишем вариант метода опорных векторов для случая двух классов, допускающих разделение с помощью линейного классификатора (раздел 15.1), а затем расширим эту модель на неразделимые данные, задачи с несколькими классами и нелинейные задачи, а также приведем некоторые факты, касающиеся эффективности этого метода (раздел 15.2). После этого мы перейдем к практической разработке классификаторов текстов (раздел 15.3) и попробуем показать, какие классификаторы подходят для решения тех или иных задач и как использовать свойства текстов, характерные для конкретных предметных областей, для их классификации. В заключение мы покажем, как использовать технологии машинного обучения, построенные нами для решения задач классификации текстов, для ранжирования документов при поиске по произвольному запросу (раздел 15.4). Несмотря на то что эту задачу можно решить несколькими методами, наиболее распространенным оказался метод опорных векторов. Он не всегда работает лучше других методов машинного обучения (возможно, за исключением ситуаций с небольшим количеством обучающих данных), но его эффективность, теоретические и практические преимущества весьма велики.

15.1. Метод опорных векторов: случай линейно разделимых классов

Если обучающее множество содержит два класса данных, допускающих линейное разделение, как показано на рис. 14.8, то существует большое количество линейных классификаторов, с помощью которых можно разделить эти данные. Интуитивно ясно, что разделяющая поверхность, проходящая через середину полосы, разделяющей два класса, лучше, чем разделяющая поверхность, лежащая очень близко к экземплярам од-

ного или обоих классов. В то время как одни методы обучения, такие как персептрон (см. ссылки в разделе 14.7), позволяют найти хотя бы один линейный разделитель, другие методы, такие как наивный байесовский метод, находят наилучший линейный разделитель, используя определенный критерий. В частности, метод опорных векторов ищет разделяющую поверхность, максимально удаленную от любых точек данных. Расстояние между этой поверхностью и ближайшей точкой данных называется *зазором классификатора* (margin). В методе опорных векторов обязательно подразумевается, что решающая функция полностью определяется (обычно малым) подмножеством данных, влияющих на положение разделителя. Эти точки называются *опорными векторами* (support vectors) (в векторном пространстве точку можно рассматривать как вектор между началом координат и этой точкой). Зазор и опорные векторы для простой задачи показаны на рис. 15.1. Другие точки данных не влияют на выбор разделяющей поверхности.

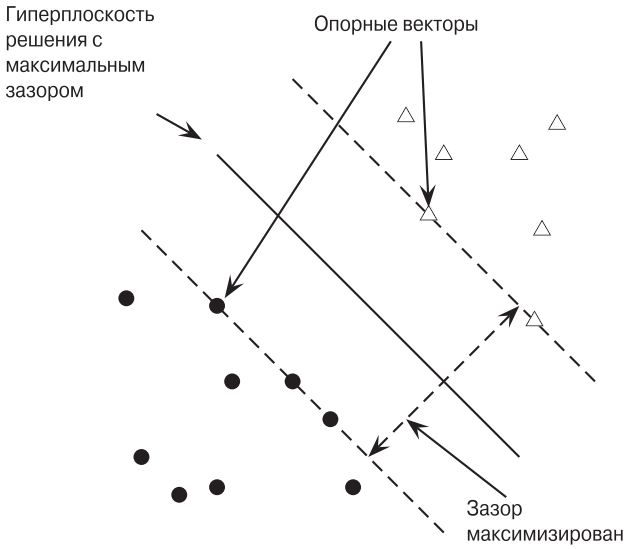


Рис. 15.1. Опорные векторы: пять точек, лежащих на границе зазора

Максимизация зазора выглядит хорошей идеей, поскольку точки, лежащие вблизи разделяющей поверхности, порождают большую неопределенность; с вероятностью 50% классификатор может принять любое из двух решений. Классификатор с большим зазором снижает неопределенность решения. Тем самым он создает определенный запас надежности: небольшая ошибка измерения или небольшое изменение документа не приведет к неправильной классификации. Другое интуитивное обоснование метода опорных векторов продемонстрировано на рис. 15.2. По своей конструкции классификатор SVM требует, чтобы вокруг разделяющей поверхности был широкий зазор. Если попытаться поместить между классами широкую полосу, то диапазон углов, при котором это можно сделать, окажется намного меньшим, чем для гиперплоскости. В результате емкость запоминания модели уменьшается, и можно ожидать, что способность модели правильно обобщать тестовые данные увеличивается (см. обсуждение компромисса между смещением и дисперсией в главе 14).

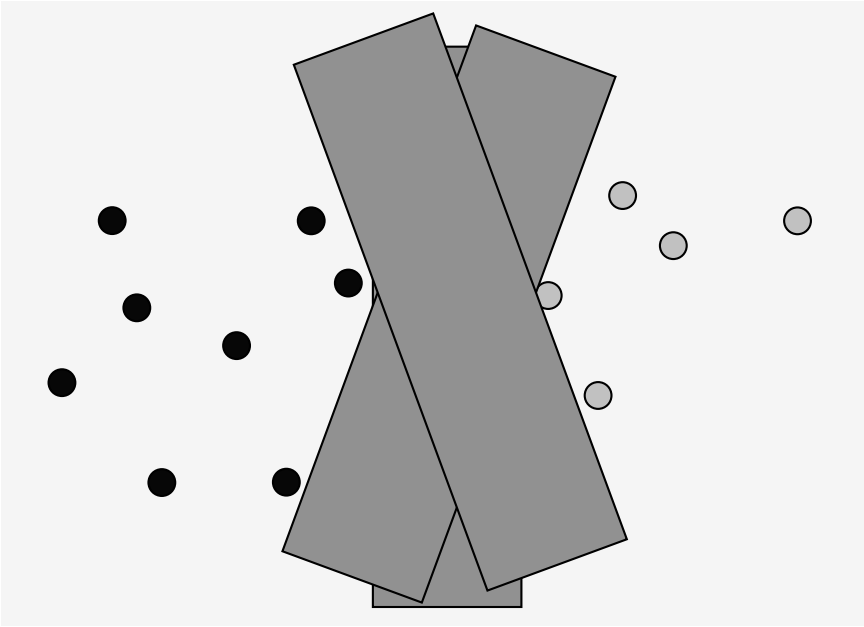


Рис. 15.2. Интуитивно понятное обоснование классификации с широким зазором. Стремление к широкому зазору уменьшает объем запоминания модели: диапазон углов, при котором между двумя множествами можно поместить широкую разделяющую полосу, меньше, чем для разделяющей гиперплоскости

Приведем формальное алгебраическое описание метода опорных векторов. Разделяющая гиперплоскость задается параметром сдвига *b* (точкой пересечения с осью *x*) и вектором $\vec{w}$ нормали к разделяющей гиперплоскости $\vec{w}$. В литературе по методам машинного обучения этот вектор обычно называется *вектором весов* (weight vector). Для того чтобы среди всех гиперплоскостей, перпендикулярных вектору нормали, выбрать одну нужную гиперплоскость, используется параметр *b*. Поскольку разделяющая гиперплоскость перпендикулярна вектору нормали, все точки $\vec{x}$ на гиперплоскости удовлетворяют уравнению $\vec{w}^T\vec{x} = -b$. Теперь допустим, что у нас есть обучающее множество $\mathbb{D} = \{(\vec{x}_i, y_i)\}$, в котором каждый элемент представляет собой пару, состоящую из точки $\vec{x}$ и соответствующей метки класса y_i. [1] В методе опорных векторов два класса всегда называются +1 и –1 (а не 1 и 0), а *параметр сдвига* (intercept term) всегда явно обозначается буквой *b* (а не включается в вектор $\vec{w}$ в качестве константного слагаемого). Благодаря этому математические выкладки становятся намного яснее. В этом случае линейный классификатор описывается следующей формулой.

$$f(\vec{x}) = sign(\vec{w}^T\vec{x} + b) \tag{15.1}$$

Значение –1 обозначает один класс, а +1 — другой.

[1] Как указывалось в разделе 14.1, мы рассматриваем точки в векторном пространстве в самом общем случае, но если эти точки представляют собой векторы документов, нормализованные по длине, то все операции выполняются на поверхности сферы, и разделяющая поверхность ее пересекает.

Классификация точки не вызывает сомнений, если она лежит далеко от разделяющей поверхности. Для заданной совокупности данных и разделяющей гиперплоскости *функциональным зазором* (functional margin) *i*-го экземпляра $\vec{x}_i$ по отношению к гиперплоскости $\langle \vec{w}, b \rangle$ называется величина $y_i\left(\vec{w}^T\vec{x}_i + b\right)$. В таком случае функциональный зазор совокупности данных относительно разделяющей поверхности вдвое больше функционального зазора любой из точек из совокупности данных с минимальным функциональным зазором (множитель 2 возникает за счет измерения всей ширины зазора, как показано на рис. 15.3). Однако с использованием этого определения связана одна проблема: значение недостаточно ограничено, поскольку функциональный зазор можно сделать сколь угодно большим, просто масштабируя параметры $\vec{w}$ и b. Например, если заменить вектор $\vec{w}$ вектором $5\vec{w}$, а параметр b — параметром $5b$, то функциональный зазор увеличится в пять раз: $y_i\left(5\vec{w}^T\vec{x}_i + 5b\right)$. Следовательно, необходимо каким-то образом ограничить величину вектора $\vec{w}$. Для этого необходимо вспомнить курс геометрии.

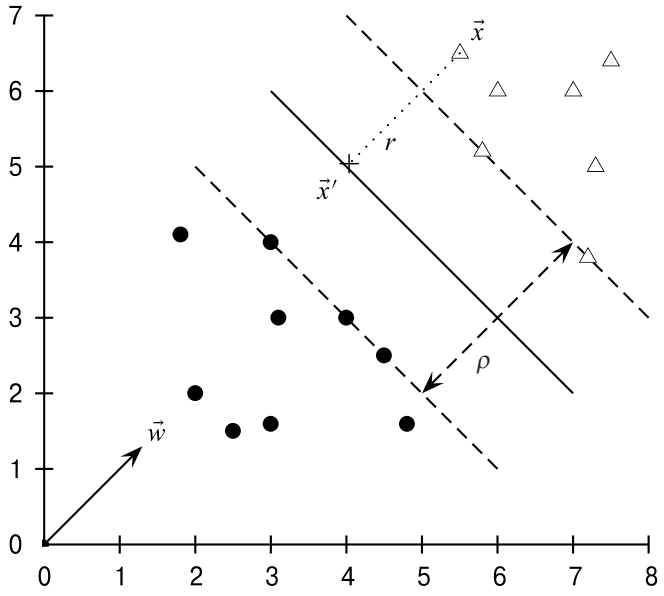


Рис. 15.3. Геометрический зазор точки r и разделяющей поверхности ρ

Что представляет собой евклидово расстояние между точкой $\vec{x}_i$ и разделяющей поверхностью? На рис. 15.3 оно обозначено символом *r*. Как известно, кратчайшее расстояние между точкой и гиперплоскостью определяется перпендикуляром к плоскости, который, естественно, параллелен вектору $\vec{w}$. Единичный вектор в этом направлении имеет вид $\vec{w}/|\vec{w}|$. Пунктирная линия на диаграмме представляет собой параллельный перенос вектора $r\vec{w}/|\vec{w}|$. Обозначим точку, лежащую на гиперплоскости, ближайшую к вектору $\vec{x}$, через $\vec{x}'$. Таким образом,

$$\vec{x}' = \vec{x} - yr\frac{\vec{w}}{|\vec{w}|}. \qquad (15.2)$$

Здесь умножение на число *y* просто изменяет знак для двух положений вектора $\vec{x}$ по разные стороны разделяющей поверхности. Более того, точка $\vec{x}'$ лежит на поверхности, а значит, удовлетворяет уравнению $\vec{w}^T\vec{x}'+b=0$. Следовательно,

$$\vec{w}^T\left(\vec{x}-yr\frac{\vec{w}}{|\vec{w}|}\right)+b=0. \tag{15.3}$$

Решая это уравнение относительно *r*, получим следующее решение [2].

$$r=y\frac{\vec{w}^T\vec{x}+b}{|\vec{w}|} \tag{15.4}$$

Точки, ближайшие к разделяющей гиперплоскости, как и прежде, являются опорными векторами. *Геометрический зазор* (geometric margin) — это максимальная ширина полосы, которую можно провести между опорными векторами двух классов. Иначе говоря, это значение, вдвое превышающее минимальное значение *r*, вычисленное по формуле (15.4), или, что эквивалентно, максимальная ширина одной из разделительных полос, показанных на рис. 15.2. Совершенно очевидно, что геометрический зазор не зависит от масштабирования: замена параметров $\vec{w}$ на $5\vec{w}$ и *b* на 5*b* не приводит к изменению геометрического зазора, поскольку он нормализуется длиной $\vec{w}$. Это значит, что мы можем наложить на вектор $\vec{w}$ любые ограничения по масштабу, не влияя на геометрический зазор. Например, можно установить ограничение $|\vec{w}|=1$. В этом случае геометрический зазор совпадает с функциональным.

Поскольку функциональный зазор можно произвольно масштабировать, стремясь к удобству решения крупных задач с помощью метода опорных векторов, потребуем, чтобы функциональный зазор всех точек данных был не меньше единицы и равнялся единице хотя бы на одном векторе данных. Иначе говоря, для всех точек должно выполняться неравенство

$$y_i\left(\vec{w}^T\vec{x}_i+b\right)\geq 1, \tag{15.5}$$

и должны существовать опорные векторы, на которых это неравенство превращается в равенство. Поскольку расстояние от точки $\vec{x}_i$ до гиперплоскости равно $r_i=y_i\left(\vec{w}^T\vec{x}_i+b\right)/|\vec{w}|$, геометрический зазор равен $\rho=\frac{2}{|\vec{w}|}$. Наша цель — максимизировать геометрический зазор. Иначе говоря, требуется найти параметры $\vec{w}$ и *b*, удовлетворяющие следующим условиям.

- Величина $\rho=\frac{2}{|\vec{w}|}$ достигает максимума.
- При всех $(\vec{x}_i,y_i)\in\mathbb{D}$, $y_i\left(\vec{w}^T\vec{x}_i+b\right)\geq 1$.

Максимизация величины $\frac{2}{|\vec{w}|}$ эквивалентна минимизации величины $\frac{|\vec{w}|}{2}$. Это приводит нас к окончательной стандартной формулировке задачи минимизации в методе опорных векторов.

Найти параметры $\vec{w}$ и *b*, удовлетворяющие следующим условиям. (15.6)

---

[2] Напомним, что $|\vec{w}|=\sqrt{\vec{w}^T\vec{w}}$.

- Величина $\frac{1}{2}\vec{w}^T\vec{w}$ достигает минимума.
- При всех $(\vec{x}_i, y_i) \in \mathbb{D}$ выполняется неравенство $y_i\left(\vec{w}^T\vec{x}_i + b\right) \geq 1$.

Итак, необходимо минимизировать квадратичную функцию при линейных ограничениях. Задача *квадратичной оптимизации* (quadratic optimization) — это хорошо изученная математическая задача оптимизации, для решения которой разработано множество алгоритмов. В принципе, реализовать метод опорных векторов можно с помощью стандартных библиотек квадратичного программирования (Quadratic Programming — QP), но в последнее время появилось много работ, которые предлагают специализированные методы квадратичного программирования для реализации метода опорных векторов. В результате разработаны более сложные, но более быстрые и масштабируемые библиотеки, которые широко используются для построения моделей. Описание этих алгоритмов не входит в нашу задачу.

Однако для понимания сущности метода опорных векторов полезно привести следующую информацию о решении поставленной оптимизационной задачи. Для того чтобы найти это решение, необходимо сформулировать двойственную задачу, в которой с каждым ограничением $y_i\left(\vec{w}^T\vec{x}_i + b\right) \geq 1$ прямой задачи связан соответствующий множитель Лагранжа α_i.

Найти $\alpha_1, \alpha_2, ..., \alpha_N$, при которых величина $\sum\alpha_i - \frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j y_i y_j \vec{x}_i^T\vec{x}_j$ достигает максимума (15.7)

- $\sum_i \alpha_i y_i = 0$,
- $\alpha_i \geq 0$ при всех $1 \leq i \leq N$.

Решение этой задачи имеет следующий вид.

$$\vec{w} = \sum \alpha_i y_i \vec{x}_i \quad (15.8)$$

$b = y_k - \vec{w}^T\vec{x}_k$ при любых $\vec{x}_k$, таких что $\alpha_k \neq 0$

В этом решении большинство параметров α_i равно нулю. Каждое ненулевое значение α_i означает, что соответствующий вектор $\vec{x}_i$ является опорным. Таким образом, функция классификации имеет следующий вид.

$$f(\vec{x}) = sign\left(\sum_i \alpha_i y_i \vec{x}_i^T\vec{x} + b\right) \quad (15.9)$$

Выражение, которое необходимо максимизировать в двойственной задаче, как и функция классификации, содержит скалярное произведение пар точек ($\vec{x}$ и $\vec{x}_i$ или $\vec{x}_i$ и $\vec{x}_j$), и это единственный способ использования данных. Значение этого факта станет ясным позднее.

Итак, мы начинаем с обучающего множества. Эта совокупность данных однозначно определяет наилучшую разделяющую гиперплоскость, которая является результатом решения задачи квадратичной оптимизации. Если новая точка $\vec{x}$ подлежит классификации, то функция классификации $f(\vec{x})$, определенная либо равенством (15.1), либо равенством (15.9), вычисляет проекцию этой точки на нормаль гиперплоскости. Знак этой функции определяет класс, которому принадлежит точка. Если точка лежит внутри зазо-

ра классификатора (или другой доверительной полосы t, которую мы установим для ошибок классификации), то классификатор отвечает "не знаю" и не выбирает ни один из классов. Значение функции $f(\vec{x})$ можно преобразовать в вероятность классификации; как правило, для этого подбирают подходящий сигмоид (Platt, 2000). Кроме того, зазор является постоянным, поэтому, если модель включает размерности разного происхождения, может понадобиться тщательное масштабирование. Однако это не проблема, если наши документы (точки) лежат на единичной гиперсфере.

Пример 15.1. Рассмотрим процесс создания классификатора по методу опорных векторов на основе (очень маленького) множества данных, показанного на рис. 15.4. С геометрической точки зрения весовой вектор, максимизирующий зазор, параллелен кратчайшей линии, соединяющей точки из двух классов, т.е. линии, проходящей через точки (1, 1) и (2, 3), что дает нам весовой вектор (1, 2). Оптимальная разделяющая поверхность ортогональна этой линии и пересекает ее посередине. Следовательно, она проходит через точку (1,5; 2). Следовательно, разделяющая поверхность по методу опорных векторов имеет вид

$$y = x_1 + 2x_2 - 5{,}5.$$

С алгебраической точки зрения мы должны минимизировать функцию $|\vec{w}|$ при условии $y_i(\vec{w}^T\vec{x}_i + b) \geq 1$. Это произойдет, если данное ограничение превратится в равенство на двух опорных векторах. Кроме того, известно, что решение имеет вид $\vec{w} = (a, 2a)$ при некотором a. Итак, необходимо решить систему уравнений

$$a + 2a + b = -1$$
$$2a + 6a + b = 1.$$

Следовательно, $a = 2/5$ и $b = -11/5$. Таким образом, оптимальная полуплоскость определяется параметрами $\vec{w} = (2/5; 4/5)$ и $b = -11/5$.

Зазор ρ равен $2/|\vec{w}| = 2/(4/25 + 15/25) = 2/(2\sqrt{5}/5) = \sqrt{5}$. Этот ответ можно подтвердить геометрически, проанализировав рис. 15.4.

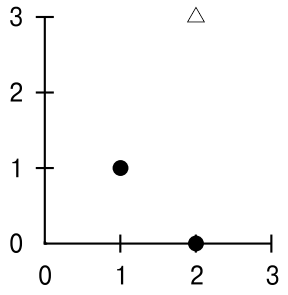


Рис. 15.4. Трехточечное обучающее множество данных для метода опорных векторов

Упражнение 15.1 **[*].** Какое минимальное количество опорных векторов может иметь набор данных, содержащий экземпляры каждого класса?

***Упражнение 15.2* [**].** Возможность использования ядер в методе опорных векторов (раздел 15.2.3) возникает благодаря тому, что функцию классификации можно записать в виде (15.9), где для больших задач почти все числа α_i равны нулю. Покажите, как можно записать функцию классификации в этом виде для данных из упражнения 15.1. Иначе говоря, запишите *f* как функцию, в которую входят точки данных, и единственной переменной является вектор $\vec{x}$.

***Упражнение 15.3* [**].** Инсталлируйте какой-нибудь программный пакет, реализующий метод опорных векторов, например SVMlight (`http://svmlight.joachims.org/`), и постройте классификатор для набора данных, описанного в примере 15.1. Убедитесь, что программа приводит к тем же самым результатам, которые указаны в тексте. Файл может иметь следующий вид, типичный для таких пакетов.

```
+1 1:2 2:3
–1 1:2 2:0
–1 1:1 2:1
```

Команда на обучение в пакете SVMlight имеет следующий вид.

```
svm_learn -c 1 -a alphas.dat train.dat model.dat
```

Опция `-c 1` необходима для того, чтобы отключить использование фиктивных переменных, которые мы обсудим в разделе 15.2.1. Убедитесь, что норма весового вектора согласуется с результатами, полученными при выполнении упражнения 15.1. Проверьте файл alphas.dat, содержащий значения α_i, и убедитесь, что они совпадают с ответами, полученными при выполнении упражнения 15.2.

15.2. Расширения модели опорных векторов

15.2.1. Классификация с мягким зазором

В задачах очень большой размерности, типичных для классификации текстов, данные иногда допускают линейное разделение. Однако в общем случае это предположение не выполняется, а если и выполняется, то предпочтение все равно следует отдать решению, которое лучше разделяет основную массу данных, игнорируя небольшое количество необычных шумовых документов.

Если обучающее множество $\mathbb{D}$ не является линейно разделимым, то обычно при построении широкого разделительного зазора допускается несколько ошибок (некоторые точки — выбросы или шумовые экземпляры — могут лежать внутри зазора или на неверной стороне). За каждый неверно классифицированный экземпляр накладывается штраф, который зависит от того, насколько сильно нарушаются условия (15.5), наложенные на зазор. Для этого в задачу вводятся *фиктивные переменные* (slack variables) ξ_i. Ненулевое значение переменной ξ_i позволяет вектору $\vec{x}_i$ нарушать требования, предъявляемые к зазору, но за это накладывается штраф, пропорциональный величине ξ_i (рис. 15.5).

Формулировка задачи оптимизации в методе опорных векторов с фиктивными переменными выглядит так.

Найти параметры $\vec{w}$, b и $\xi_i \geq 0$, при которых (15.10)

- функция $\frac{1}{2}\vec{w}^T\vec{w} + C\sum_i \xi_i$ достигает минимума и
- при всех $\{(\vec{x}_i, y_i)\}$ выполняется неравенство $y_i\left(\vec{w}^T\vec{x}_i + b\right) \geq 1 - \xi_i$.

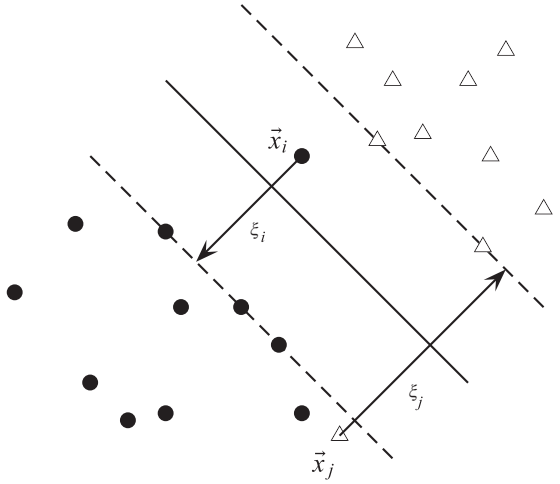


Рис. 15.5. Классификация с широким зазором и фиктивными переменными

Решение оптимизационной задачи носит компромиссный характер: оно устанавливает баланс между шириной зазора и количеством точек, которые пришлось бы переместить, для того чтобы обеспечить эту ширину. Установив фиктивное значение $\xi_i > 0$, ширину зазора для точки $\vec{x}_i$ можно сделать меньше единицы, но за это придется заплатить штраф $C\xi_i$. Сумма величин ξ_i определяет верхнюю границу количества ошибок при обучении. Метод опорных векторов с мягким зазором (soft-margin SVM) минимизирует количество ошибок при обучении за счет ширины зазора. Параметр C называется *параметром регуляризации* (regularization term). Он позволяет управлять переобучением: если параметр C становится большим, то нежелательно игнорировать данные за счет уменьшения геометрического зазора; если параметр C небольшой, то с помощью фиктивных переменных можно легко учесть некоторые точки и получить зазор, моделирующий основную массу данных.

Двойственная задача классификации с мягким зазором формулируется так.

Найти параметры $\alpha_1, \alpha_2, ..., \alpha_N$, максимизирующие функцию (15.11)

$$\sum \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j \vec{x}_i^T \vec{x}_j$$ при условиях

- $\sum_i \alpha_i y_i = 0$
- $0 \le \alpha_i \le C$ при всех $1 \le i \le N$.

В двойственной задаче не появляются ни фиктивные переменные ξ_i, ни множитель Лагранжа для них. В ней остается только константа C, ограничивающая возможную величину множителя Лагранжа для опорных векторов. Как и прежде, опорными векторами являются точки $\vec{x}_i$ с ненулевыми значениями α_i. Решение двойственной задачи имеет следующий вид.

$$\vec{w} = \sum \alpha y_i \vec{x}_i \tag{15.12}$$

$$b = y_k (1 - \xi_k) - \vec{w}^T \vec{x}_k \text{ для } k = \arg\max_k \alpha_k$$

Как и прежде, вектор $\vec{w}$ в явном виде для классификации не нужен. Классификацию можно осуществить с помощью вычисления скалярного произведения, как в формуле (15.9).

Как правило, опорные векторы составляют небольшую часть обучающего множества. Однако если задача не обладает свойством линейной разделимости или зазор мал, то каждая неверно классифицированная точка или точка, лежащая внутри зазора, будет иметь ненулевое значение α_i. Если множество таких точек становится большим, то в нелинейном случае, который будет рассмотрен в разделе 15.2.3, это может оказаться основным фактором снижения замедления метода опорных векторов на этапе тестирования.

Сложность обучения и тестирования с помощью линейного метода опорных векторов показана в табл. 15.1.[3] Время обучения в методе опорных векторов в основном определяется временем решения соответствующей задачи квадратичного программирования, поэтому теоретическая и эмпирическая сложность зависит от способа решения этой задачи. Считается, что временная сложность стандартного решения задачи квадратичного программирования пропорциональна кубу объема набора данных (Kozlov et al., 1979). Все недавние работы по методу опорных векторов направлены на снижение этой сложности, причем довольно часто это происходит за счет того, что точное решение заменяется приближенным. Как правило, эмпирическая сложность этих методов составляет $O(|\mathbb{D}|^{1,7})$ (Joachims, 2006*a*). Тем не менее сверхлинейная сложность традиционных алгоритмов опорных векторов затрудняет и даже иногда делает невозможным их применение к крупным наборам обучающих данных. Альтернативные алгоритмы опорных векторов, сложность которых линейно зависит от количества обучающих выборок, плохо масштабируются для большого количества признаков, что является характерным для задач классификации текстов. Однако новые многообещающие алгоритмы обучения, основанные на методе отсекающих плоскостей, линейно зависят от количества обучающих примеров и количества ненулевых признаков в них (Joachims, 2006*a*). И все же реальная скорость квадратичной оптимизации намного ниже скорости простого подсчета терминов в наивной байесовской модели. Замена линейного метода опорных векторов нелинейным, как будет показано в следующем разделе, обычно приводит к повышению временно*й* сложности обучения в $|\mathbb{D}|$ раз (поскольку необходимо вычислять скалярное произведение элементов обучающего множества), что совершенно неприемлемо. На практике чаще дешевле создать признаки более высокого порядка и провести обучение с помощью линейного метода опорных векторов.[4]

---

[3] Вместо $\Theta(T)$ мы пишем $\Theta(|\mathbb{D}|L_{ave})$ и предполагаем, что длина тестовых документов ограничена.

[4] Приблизительно так: набор признаков расширяется за счет признаков более высокого порядка на основе первоначальных признаков и их комбинации. Например, (x1, x2, x3) → (x1, x2, x3, x1*x1, x1*x2). Создание признаков означает непосредственное вычисление признаков более высокого порядка и учет их взаимодействия с их последующим включением в линейную модель.

Таблица 15.1. Сложность обучения и тестирования разных классификаторов, включая метод опорных векторов. Время обучения — это время, которое метод обучения затрачивает на настройку классификатора с помощью множества $\mathbb{D}$, а время тестирования — это время, которое классификатор затрачивает на классификацию одного документа. Для метода опорных векторов предполагается, что классификация по нескольким классам производится с помощью совокупности $|\mathbb{C}|$ бинарных (один класс — все остальные) классификаторов. L_{ave} — это среднее количество лексем на документ, а M_{ave} — средний размер лексикона документа (количество ненулевых признаков). L_a и M_a — количество лексем и разных терминов в тестовом документе соответственно

Классификатор	*Вид*	*Метод*	*Временная сложность*
Наивный байесовский	Обучение		$\theta(\|\mathbb{D}\|L_{ave} + \|\mathbb{C}\|\|V\|)$
Наивный байесовский	Тестирование		$\theta(\|\mathbb{C}\|M_a)$
Роккио	Обучение		$\theta(\|\mathbb{D}\|L_{ave} + \|\mathbb{C}\|\|V\|)$
Роккио	Тестирование		$\theta(\|\mathbb{C}\|M_a)$
kNN	Обучение	Предварительная обработка	$\theta(\|\mathbb{D}\|L_{ave})$
kNN	Тестирование	Предварительная обработка	$\theta(\|\mathbb{D}\|M_{ave}M_a)$
kNN	Обучение	Без предварительной обработки	$\theta(1)$
kNN	Тестирование	Без предварительной обработки	$\theta(\|\mathbb{D}\|L_{ave}M_{ave})$
Метод опорных векторов	Обучение	Обычный	$O(\|\mathbb{C}\|\|\mathbb{D}\|^3 M_{ave}) \approx O(\|\mathbb{C}\|\|\mathbb{D}\|^{1,7} M_{ave})$, эмпирически
Метод опорных векторов	Обучение	Отсекающие плоскости	$O(\|\mathbb{C}\|\|\mathbb{D}\|L_{ave}M_{ave})$
Метод опорных векторов	Тестирование		$O(\|\mathbb{C}\|M_a)$

15.2.2. Метод опорных векторов с несколькими классами

Стандартный метод опорных векторов предназначен для классификации по двум классам. Традиционно для классификации по нескольким классам с помощью метода опорных векторов используется один из методов, описанных в разделе 14.5. В частности, чаще всего на практике создается $|\mathbb{C}|$ классификаторов, работающих по принципу “один против остальных” (иногда этот принцип называется “один против всех” (One-Versus-All — OVA)), а затем выбирается класс, на котором тестовый документ отстоит дальше всего от разделяющей поверхности. Другая стратегия заключается в построении совокупности классификаторов, работающих по принципу “один против одного”, а затем выбирается класс, предложенный большинством классификаторов. Несмотря на то что эта процедура предусматривает создание $|\mathbb{C}|(|\mathbb{C}|-1)/2$ классификаторов, время на их обучение

на деле может снизиться, поскольку обучающее множество для каждого классификатора намного меньше.

Однако все эти стратегии не очень элегантны. Намного лучше разработать классификатор для нескольких классов, построив бинарный классификатор по вектору признаков $\Phi(\vec{x}, y)$, построенному по парам, состоящим из входных признаков и соответствующего класса. На этапе тестирования классификатор выбирает класс $y = \arg\max_{y'} \vec{w}^T \Phi(\vec{x}, y')$. Зазором на этапе обучения является разница между значениями, соответствующими правильному и ближайшему неправильному классам, поэтому задача квадратичного программирования содержит следующее условие: $\forall i \, \forall y \neq y_i \; \vec{w}^T \Phi(\vec{x}_i, y_i) - \vec{w}^T \Phi(\vec{x}_i, y) \geq 1 - \xi_i$. С помощью этого общего метода можно сформулировать задачу многоклассовой классификации для различных линейных классификаторов. Кроме того, это пример простого обобщения классификации, в котором классы представляют собой не просто множество независимых категориальных меток, а могут быть произвольно структурированными объектами с взаимными зависимостями. Такие варианты метода опорных векторов называются *структурными* (structural SVM). Мы вернемся к ним в разделе 15.4.2.

15.2.3. Нелинейный метод опорных векторов

До сих пор мы рассматривали случаи, когда наборы данных допускали линейное разделение (возможно, с небольшими исключениями или шумами). А что же делать, если совокупность данных не позволяет применять линейные классификаторы? Рассмотрим одномерный случай. Данные, приведенные в верхней части рис. 15.6, легко распознаются линейным классификатором, а данные в средний части — нет. Для того чтобы распознать данные в средней части, нужно выделить интервал. Один из способов решения этой проблемы заключается в отображении данных в пространство более высокой размерности с последующим применением линейного классификатора в этом пространстве. Например, в нижней части рис. 15.6 показано, что линейный классификатор легко распознает данные, если для отображения данных в двумерную плоскость используется квадратичная функция (другим вариантом являются полярные координаты). Основная идея заключается в отображении исходного пространства признаков в пространство признаков более высокой размерности, в котором обучающее множество оказывается линейно разделимым. Разумеется, при этом желательно сохранить релевантную размерность отношений между точками данных, так, чтобы полученный классификатор обобщал исходные данные.

Метод опорных векторов, как и ряд других линейных классификаторов, позволяет легко и эффективно осуществлять такое отображение данных в пространство более высокой размерности. Этот прием называется *переходом к ядру* (kernel trick). Линейный классификатор, построенный по методу опорных векторов, основан на вычислении скалярного произведения между векторами, соответствующими данным. Введем следующее обозначение: $K(\vec{x}_i, \vec{x}_j) = \vec{x}_i \vec{x}_j^T$. В таком случае уже известный нам классификатор можно переписать иначе.

$$f(\vec{x}) = sign\left(\sum_i \alpha_i y_i K(\vec{x}_i, \vec{x}) + b\right) \tag{15.13}$$

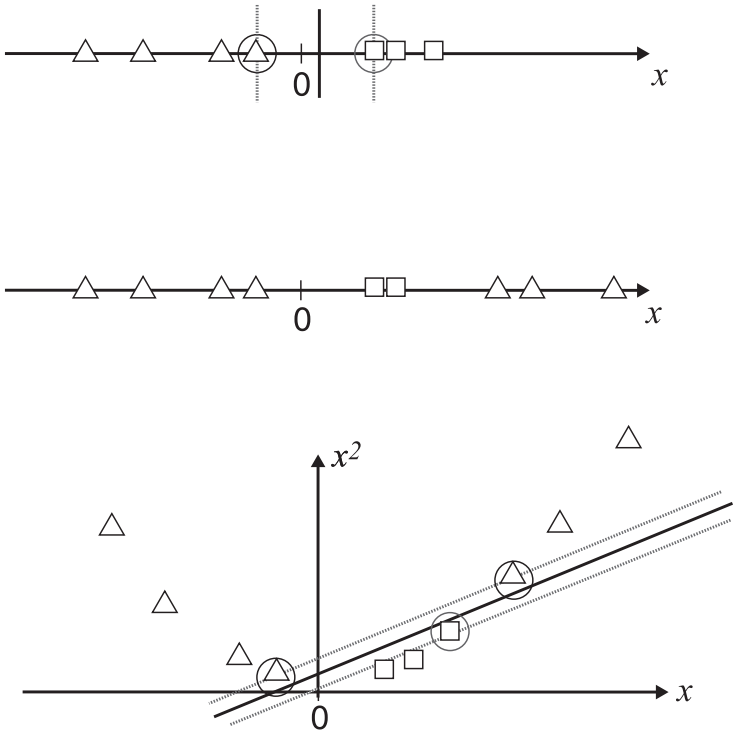


Рис. 15.6. Проекция данных, не допускающих линейного разделения, в пространство более высокой размерности, в котором линейная разделимость существует

Предположим теперь, что мы решили отобразить каждую точку в пространство более высокой размерности, используя функцию $\Phi: \vec{x} \mapsto \phi(\vec{x})$. В таком случае скалярное произведение превращается в произведение $\phi(\vec{x}_i)^T \phi(\vec{x}_j)$. Если окажется, что это скалярное произведение (которое представляет собой действительное число) можно вычислить относительно просто и эффективно по исходным точкам, то отображение $\vec{x} \mapsto \phi(\vec{x})$ на самом деле осуществлять необязательно. Вместо этого можно просто вычислить величину $K(\vec{x}_i, \vec{x}_j) = \phi(\vec{x}_i)^T \phi(\vec{x}_j)$, а затем использовать значение функции в формуле (15.13). *Ядро* K — это функция, соответствующая скалярному произведению в некотором расширенном пространстве признаков.

Пример 15.2. **Квадратичное ядро на плоскости.** Для двумерных векторов $\vec{u} = (u_1 \quad u_2)$ и $\vec{v} = (v_1 \quad v_2)$ рассмотрим функцию $K(\vec{u}, \vec{v}) = (1 + \vec{u}^T \vec{v})^2$. Покажем, что она является ядром, т.е. $K(\vec{u}, \vec{v}) = \phi(\vec{u})^T \phi(\vec{v})$ при некотором ϕ. Рассмотрим вектор $\phi(u) = (1 \quad u_1^2 \quad \sqrt{2}u_1u_2 \quad u_2^2 \quad \sqrt{2}u_1 \quad \sqrt{2}u_2)$. В таком случае

$$
\begin{aligned}
K(\vec{u},\vec{v}) &= \left(1+\vec{u}^T\vec{v}\right)^2 = \\
&= 1+u_1^2v_1^2+2u_1v_1u_2v_2+u_2^2v_2^2+2u_1v_1+2u_2v_2 = \\
&= \begin{pmatrix}1 & u_1^2 & \sqrt{2}u_1u_2 & u_2^2 & \sqrt{2}u_1 & \sqrt{2}u_2\end{pmatrix}^T = \begin{pmatrix}1 & v_1^2 & \sqrt{2}v_1v_2 & v_2^2 & \sqrt{2}v_1 & \sqrt{2}v_2\end{pmatrix} = \\
&= \phi(\vec{u})^T\phi(\vec{v}).
\end{aligned}
\tag{15.14}
$$

Используя терминологию функционального анализа, зададим вопрос “Какие функции могут быть *ядрами* (kernels)?” Ядра иногда точнее называть *ядрами Мерсера* (Mercer kernels), потому что они должны удовлетворять условию Мерсера: при любой функции $g(\vec{x})$, такой что интеграл $\int g^2(\vec{x})d\vec{x}$ конечен, должно выполняться условие

$$\int K(\vec{x},\vec{z})g(\vec{x})g(\vec{z})d\vec{x}d\vec{z} \geq 0\,. \tag{15.15}$$

Ядро *K* должно быть непрерывным, симметричным, а также иметь положительно определенную матрицу Грама (Gram). Эти условия гарантируют, что существует отображение в воспроизводящее ядро гильбертова пространства (гильбертово пространство — это векторное пространство, полное относительно скалярного произведения [5]), т.е. пространство, скалярное произведение в котором совпадает со значением функции *K*. Если ядро не удовлетворяет условию Мерсера, то соответствующая задача квадратичного программирования может не иметь решения. Для того чтобы лучше понять эти проблемы, рекомендуем обратиться к книгам, перечисленным в разделе 15.5. В противном случае можете удовлетвориться тем, что в 90% работ, посвященных нелинейному методу опорных векторов, используется одно из двух семейств функций двух векторов, которые будут рассмотрены ниже и которые являются допустимыми ядрами.

Наиболее распространенными семействами ядер являются полиномиальные ядра и функции радиального базиса. Полиномиальные ядра (polynomial kernels) имеют вид $K(\vec{x},\vec{z}) = \left(1+\vec{x}^T\vec{z}\right)^d$. При $d = 1$ ядро является линейным. Именно с таким ядром мы работали до этого раздела (константа 1 просто изменяет порог). При $d = 2$ возникает квадратичное ядро, которое также широко используется. Квадратичное ядро описано в примере 15.2.

Наиболее распространенной формой функции радиального базиса является функция плотности гауссова распределения

$$K(\vec{x},\vec{z}) = e^{-(\vec{x}-\vec{z})/(2\sigma^2)}\,. \tag{15.16}$$

Функция радиального базиса эквивалентна отображению данных в бесконечномерное гильбертово пространство, поэтому функцию радиального базиса невозможно проиллюстрировать так же конкретно, как квадратичное ядро. Кроме этих двух семейств, существуют интересные работы, посвященные разработке других ядер (некоторые из них позволяют получить многообещающие результаты в области классификации текстов). В частности, существуют исследования по строковым ядрам (string kernels) (см. раздел 15.5).

В теории метода опорных векторов используется особый язык, который несколько отличается от терминологии машинного обучения. Корни этой терминологии уходят глубоко в математику, но это не должно приводить читателей в благоговейный ужас. Действительно, мы говорим о достаточно простых вещах. Полиномиальные ядра позво-

[5] Строго говоря, гильбертово пространство — это векторное пространство, в котором введено скалярное произведение и которое является полным относительно нормы, порожденной этим скалярным произведением. — *Примеч. ред.*

ляют моделировать конъюнкции признаков (вплоть до порядка полинома). Иначе говоря, если мы хотим моделировать появление пар слов, которые дают больше информации о теме, чем отдельные слова, например *operating* `AND` *system* или *ethnic* `AND` *cleansing*, то следует применять квадратичное ядро. Если полезную информацию дают триплеты, то необходимо использовать кубическое ядро. Одновременно в задаче появляются степени исходных признаков, хотя в большинстве приложений в области классификации текстов это, возможно, и не требуется. Однако это является следствием математических вычислений и обычно не приносит вреда. Функции радиального базиса позволяют использовать признаки, выделяющие круги (гиперсферы), хотя при взаимодействии признаков разделяющей поверхности они могут оказаться намного сложнее. Строковые ядра позволяют работать с признаками, являющимися символьными подстроками терминов. Все это — вполне очевидные понятия, которые уже упоминались в других местах под другими именами.

15.2.4. Экспериментальные результаты

Результаты, изложенные в разделе 13.6, показывают, что метод опорных векторов является очень эффективным классификатором текстов. Результаты Дюма и др. (Dumais et al., 1998), приведенные в табл. 13.9, явно демонстрируют, что метод опорных векторов наилучший. Благодаря таким работам метод опорных векторов завоевал отличную репутацию. Другая пионерская работа по масштабированию и оценке метода опорных векторов для задачи классификации текстов была выполнена Йоахимсом (Joachims, 1998). Некоторые из результатов его более поздней работы (Joachims, 2002*a*) приведены в табл. 15.2.[6] Йоахимс использовал большое количество признаков, в противоположность Дюма и др. (Dumais et al., 1998), которые для построения классификатора использовали взаимную информацию (см. раздел 13.5.1) и в результате уменьшили количество признаков. Успех метода опорных векторов отражает результаты, касающиеся других линейных классификаторов (см. раздел 14.6). На первый взгляд, работа с простыми признаками может завести довольно далеко. Следует подчеркнуть, насколько различаются результаты, полученные в разных работах, посвященных одному и тому же методу машинного обучения. В частности, основываясь на повторных исследованиях, проведенных другими исследователями, результаты, характеризующие наивный байесовский метод в работе Йоахимса (Joachims, 1998), следует признать очень слабыми, а результаты, представленные в табл. 13.9, — репрезентативными.

---

[6] Эти результаты выражены через равновесный показатель F_1 (см. раздел 8.4). Многие исследователи критикуют использование данного показателя для оценки классификации текстов, так как его вычисление может быть связано с интерполированием, а не с реальными параметрами системы, и поэтому неясно, почему следует использовать именно его, а не максимум показателя F_1 или другую точку на кривой “точность/полнота”. Ранние работы (Joachims, 1998) давали основание предполагать, что использование полиномиальных ядер высокого порядка или функций радиального базиса дает большие преимущества при решении описываемой задачи. Эти предположения оправдались только для метода опорных векторов с жестким зазором. Среди методов опорных векторов с мягким зазором простой линейный метод опорных векторов, в котором по умолчанию $C = 1$, работает лучше.

Таблица 15.2. Равновесное значение показателя F_1, характеризующего эффективность метода опорных векторов (Joachims, 2002*a*). Результаты приведены для десяти самых больших категорий, а также микроусредненное значение по всем 90 категориям коллекции Reuters-21578

					Линейный SVM		*SVM с ядром радиального базиса*
	NB	*Роккио*	*Деревья решений*	*kNN*	*C = 0,5*	*C = 1,0*	*σ ≈ 0,7*
earn	96,0	96,1	96,1	97,8	98,0	98,2	98,1
acq	90,7	92,1	85,3	91,8	95,5	95,6	94,7
money-fx	59,6	67,6	69,4	75,4	78,8	78,5	74,3
grain	69,8	79,5	89,1	82,6	91,9	93,1	93,4
crude	81,2	81,5	75,5	85,8	89,4	89,4	88,7
trade	52,2	77,4	59,2	77,9	79,2	79,2	76,6
interest	57,6	72,5	49,1	76,7	75,6	74,8	69,1
ship	80,9	83,1	80,9	79,8	87,4	86,5	85,8
wheat	63,4	79,4	85,5	72,9	86,6	86,6	82,4
corn	45,2	62,2	87,7	71,4	87,5	87,7	84,6
Микросреднее	72,3	79,9	79,4	82,6	86,7	87,5	86,4

15.3. Проблемы, связанные с классификацией текстовых документов

Существует масса коммерческих приложений классификации текстов; возможно, самым распространенным приложением стала фильтрация электронной почты от спама. Джексон и Мулинье (Jackson and Moulinier, 2002) пишут: “Коммерческая ценность автоматической классификации документов по содержанию несомненна. Существуют мириады потенциальных приложений для корпоративных сетей, правительственных учреждений и поставщиков контента в Интернете”.

При обсуждении классификации наше внимание было сосредоточено на методах машинного обучения, а не на конкретных признаках текстовых документов, подходящих для классификации. Это вполне допустимо для учебников, но неуместно с точки зрения специалистов-практиков. Очень часто более высокой производительности можно достичь за счет использования признаков, характерных для предметной области, а не благодаря замене одного метода машинного обучения другим. Джексон и Мулинье (Jackson and Moulinier, 2002) предположили, что “понимание данных — один из ключей к успешной категоризации, хотя именно в этой области большинство предлагаемых инструментов категоризации чрезвычайно слабы. Многие из “универсальных” инструментов, предлагаемых на рынке, не прошли тестирование на широком спектре разнотипных текстов”. В этом разделе мы рассмотрим приложения классификации текстов, пространство возможных решений и полезность прикладных эвристик.

15.3.1. Выбор типа классификатора

Когда возникает необходимость построить классификатор текстов, то первый вопрос, на который следует ответить, — "Какой объем данных для обучения доступен в настоящий момент? Никаких данных? Очень мало? Довольно много? Огромное количество, возрастающее ежедневно?" Часто одной из главных задач при создании реальных классификаторов, основанных на машинном обучении, является создание или получение достаточного количества данных для обучения. Многие задачи для построения высокоэффективного классификатора часто требуют сотни и тысячи обучающих примеров для каждого класса, к тому же реальные приложения могут содержать большой набор категорий. Мы будем изначально предполагать, что классификатор нужен как можно скорее; если же для его реализации выделено много времени, то значительную его часть можно потратить на сбор данных.

Если размеченных данных для обучения нет, и особенно если существует опытный персонал, знающий предметную область, то никогда не следует забывать о решении задачи с помощью правил, написанных вручную. Иначе говоря, следует написать постоянные запросы, упомянутые в главе 13. Например,

```
IF (wheat OR grain) AND NOT (whole OR bread) THEN c = grain.
```

На практике правила намного сложнее, чем приведенное выше, и формулируются с помощью более сложных языков запросов, чем булевы выражения, включая использование числовых весов. При тщательном подходе (т.е. если люди настраивают правила на данных для обучения) точность этих правил может быть очень высокой. Якобс и Рау (Jacobs and Rau, 1990) сообщили, что идентифицировали статьи о поглощении компаний с точностью, равной 92%, и полнотой, равной 88,5%. Хейес и Вайнштейн (Hayes and Weinstein, 1990) утверждают, что достигли 94%-ной полноты и 84%-ной точности на 675 категориях коллекции новостных сообщений Reuters. Тем не менее объем работ, связанных с разработкой тонко настроенных правил, очень велик. Как правило, на один класс затрачивается два дня, а на уточнение правил также уходит дополнительное время, поскольку содержание документов в классах со временем "дрейфует" (о "дрейфе понятий" речь идет в разделе 13.4).

Если данных мало и вы хотите разработать классификатор с учителем, то примите к сведению следующее: теория машинного обучения утверждает, что необходимо выбирать классификатор с большим смещением (см. раздел 14.6). Например, существуют теоретические и эмпирические результаты, свидетельствующие о том, что наивный байесовский метод в таких ситуациях работает хорошо (Ng and Jordan, 2001; Forman and Cohen, 2004), хотя на практике при работе с регуляризированными моделями на текстовых данных этот эффект может и не проявиться (Klein and Manning, 2002). В любом случае модели с очень маленьким смещением, такие как модель ближайшего соседа, вероятно, в таких ситуациях противопоказана. В любом случае при небольшом объеме обучающих выборок качество модели снижается.

С теоретической точки зрения в такой ситуации интересно попытаться применить *методы обучения с частичным привлечением учителя* (semisupervised training methods). К ним относятся *самонастройка* (bootstrapping) и EM-*алгоритм* (expectation-maximization algorithm), которые будут описаны в разделе 16.5. При обучении с помощью этих методов система получает сначала некоторое количество размеченных документов, а затем — большое количество неразмеченных документов, на которых пытается пройти обучение. Одно из серьезных преимуществ наивного байесовского метода заключается в том, что

он легко распространяется на алгоритмы частичного обучения. У метода опорных векторов также существует вариант с частичным обучением, который называется *трансдуктивным методом опорных векторов* (transductive SVMs). Ссылки на литературу приведены в соответствующем разделе.

Практичное решение проблемы — постараться получить дополнительные размеченные данные как можно быстрее. Лучше всего самому включиться в процесс, в ходе которого люди будут рассматривать разметку документов для вас, как естественную часть своей работы. Например, во многих ситуациях люди сортируют или распределяют электронные сообщения в соответствии со своими задачами, и эти действия позволяют выявить информацию о классах. Альтернативный подход, когда документы размечаются специально для построения классификатора, обычно сложнее организовать, к тому же такие данные имеют более низкое качество, потому что люди решают довольно отвлеченную задачу. Вместо того чтобы поручать людям размечать все или случайное подмножество документов, можно провести *активное обучение* (active learning), которому посвящено довольно много работ. В ходе активного обучения создается отдельная система, решающая, какой документ должен разметить человек. Обычно это именно те документы, относительно которых классификатор выдает неопределенный ответ. Это позволяет сократить стоимость разметки в 2–4 раза. Однако существует проблема: документы, размечаемые для обучения классификатора одного типа, часто не подходят для обучения классификатора другого типа.

Если объем обучающих данных вполне достаточен, то можно применять все методы, описанные ранее. Например, можно использовать метод опорных векторов. Однако если вы разрабатываете линейный классификатор, например, на основе метода опорных векторов, возможно, в приложении вам стоит совместить его с классификатором на основе булевых правил. Пользователи часто хотят исправить недостаточно точные ответы, и если руководство дает указание исправить классификацию конкретного документа прямо сейчас, то это намного проще сделать с помощью правила, написанного вручную, чем уточняя веса в методе опорных векторов и рискуя потерять общую точность классификации. Это одна из причин популярности методов машинного обучения, таких как деревья решений, которые порождают легко интерпретируемые человеком логические модели.

Если количество данных огромно, то выбор классификатора, скорее всего, слабо повлияет на результаты классификации, и наилучший выбор останется неясным (Banko and Brill, 2001). Возможно, лучше всего выбирать классификатор, ориентируясь на масштабирование фазы обучения или даже на производительность вычислений на этапе классификации. Однако для этого необходим огромный объем данных. Эмпирическое правило гласит: удвоение объема обучающих данных приводит к линейному увеличению производительности классификатора, но при огромном объеме данных для обучения улучшение становится сублинейным.

15.3.2. Повышение производительности классификатора

В любом практическом приложении обычно есть запас для повышения эффективности классификатора за счет использования признаков, характерных для конкретной предметной области или коллекции документов. Часто документы содержат зоны, особенно полезные для классификации. Кроме того, могут существовать специфические подмножества лексикона, которые требуют специальной обработки для достижения оптимальной эффективности классификатора.

Крупные и сложные категориальные таксономии

Если задача классификации текстов состоит из небольшого количества хорошо разделимых категорий, то многие алгоритмы классификации работают хорошо. Однако многие реальные задачи классификации состоят из очень большого количества часто весьма похожих друг на друга категорий. Можно вспомнить о веб-каталогах (например, Yahoo! Directory или Open Directory Project), библиотечных схемах классификации — десятичной классификации Дьюи и классификации библиотека Конгресса (США) (Dewey Decimal или Library of Congress) — или схемах классификации юридической либо медицинской информации. Например, каталог Yahoo! Directory состоит более чем из 200 тысяч категорий, образующих глубокую иерархию. Точная классификация на большом множестве тесно связанных классов — весьма трудная задача.

Большинство крупных множеств категорий образуют иерархическую структуру, и попытка использовать эту информацию с помощью *иерархической классификации* (hierarchical classification) выглядит многообещающе. Однако в настоящее время повышение эффективности за счет иерархической классификации, а не благодаря работе непосредственно с классами-листьями, еще недостаточно велико. [7] Тем не менее иерархическая классификация может оказаться очень полезной для повышения масштабируемости классификаторов на больших иерархиях категорий. Другой простой способ улучшения масштабируемости классификаторов на крупных иерархиях заключается в агрессивном отборе признаков. Ссылки на работы, посвященные иерархической классификации, приведены в разделе 15.5.

Общий вывод, который можно сделать, изучив методы машинного обучения, гласит: всегда можно добиться небольшого увеличения качества классификации за счет сочетания нескольких классификаторов при условии, что ошибки, которые они делают, более или менее независимы. В настоящее время опубликовано много работ о таких методах, как *голосование* (voting), *бэггинг* (bagging) и *бустинг* (boosting). Библиографические ссылки на эти методы приведены в конце главы. Тем не менее в конце концов для получения удовлетворительной правильности классификации может понадобиться гибридное полуавтоматическое решение. В таких ситуациях сначала запускается классификатор, а затем принимаются все его решения, имеющие высокий доверительный уровень. В то же время решения классификатора с невысоким уровнем доверия направляются на проверку человеку. Такой процесс автоматически приводит к появлению новых обучающих данных, которые можно использовать в будущих вариантах классификатора. Однако следует отметить, что в этом случае данные для обучения *не* являются случайной выборкой из пространства документов.

Признаки текста

Как при поиске по запросу, так и при классификации текстов в качестве признаков по умолчанию используются термины. Однако при классификации текстов можно получить большой выигрыш за счет разработки специальных признаков, характерных для конкретной задачи. В отличие от языков запросов для информационного поиска, при классификации текстов нет никаких препятствий для передачи этих признаков конечному

[7] В небольшом дереве категорий на рис. 13.1 листовыми классами являются категории *poultry* и *coffee*, а класс *industries* находится на более высоком уровне иерархии.

пользователю, поскольку они относятся к “внутреннему устройству” классификатора. Этот процесс называется *конструированием признаков* (feature engineering). В настоящее время конструирование признаков остается привилегией человека, а не решается методами машинного обучения. Качественное конструирование признаков часто позволяет заметно повысить эффективность классификаторов. Это особенно выгодно в отдельных приложениях классификации текстов, таких как фильтрация спама и порнографических материалов.

Задачи классификации часто содержат большое количество терминов, которые удобно сгруппировать и считать эквивалентными. Типичными примерами являются упоминания календарных годов, строки, состоящие из восклицательных знаков, а также специальные лексемы, такие как коды ISBN и химические формулы. Часто их непосредственное использование приводит к значительному увеличению лексикона, но не повышает мощность классификатора, помимо того, что теперь известно, что текст содержит некую химическую формулу. В таких случаях количество признаков и их разреженность можно уменьшить, сопоставляя их с регулярными выражениями и конвертируя их в самостоятельные лексемы. Благодаря этому эффективность и скорость работы классификатора обычно повышаются. Иногда все числа преобразуются в один признак, но чаще некоторое значение можно различить по разным видам представления; скажем, четырехзначные числа (как правило — годы) можно отличить от остальных чисел, а также от действительных чисел с десятичной точкой. Аналогичные методы можно применить к датам, кодам ISBN, результатам спортивных соревнований и т.д.

Двигаясь в другом направлении, часто полезно увеличивать количество признаков, сравнивая части слов, а также информативные последовательности слов. Части слов часто сравниваются по символьным *k*-граммным признакам. Такие признаки особенно полезны при классификации текстов с неизвестными словами. Например, неизвестное слово с окончанием *-rase*, скорее всего, означает фермент, даже если слово не встречается в данных для обучения. Качественные многословные шаблоны можно найти, просматривая часто встречающиеся пары слов (возможно, с помощью критерия взаимной информации между словами, по аналогии с его использованием в разделе 13.5.1 для выбора признаков), а затем использовать методы выбора признаков. Это полезно в ситуациях, когда компоненты словосочетания, взятые по отдельности, могут привести к неверной классификации. Например, слово *ethnic* (этнический) является хорошим признаком категорий *food* (еда) и *arts* (искусство), а слово *cleansing* (уборка, чистка) характерно для категории *home* (дом), но сочетание *ethnic cleansing* (этническая чистка) указывает на категорию *world news* (мировые новости). Некоторые классификаторы текстов используют признаки, полученные на основе распознавания именованных объектов в текстах.

Могут ли такие методы, как стемминг и перевод в нижний регистр (см. раздел 2.2), помочь в решении задач классификации текстов? Как всегда, окончательный ответ можно дать, только проведя эмпирическую оценку на подходящей тестовой коллекции. Тем не менее стоит отметить, что эти методы имеют мало шансов улучшить классификацию текстов. Для информационного поиска часто необходимо склеивать формы слов, например *oxygenate* и *oxygenation*, поскольку присутствие хотя бы одного из них в документе является хорошим индикатором того, что данный документ релевантен запросу `oxygenation`. В то же время при обширных обучающих данных стемминг ничего не привносит в классификацию текстов. Если несколько форм с общей основой несут схожую информацию, то соответствующие признаки будут иметь близкие веса. Такие методы, как стемминг, позволяют лишь компенсировать разреженность данных. Это может ока-

заться полезным (как отмечалось в начале раздела), но часто разные формы слова могут быть признаками разных классов документов. Слишком агрессивный стемминг легко может снизить эффективность классификации.

Зоны документа

Как указывалось в разделе 6.1, документы обычно имеют зоны, например заголовок сообщения электронной почты имеет поля “Тема” и “Автор”, а научная статья — заголовок и ключевые слова. Классификаторы текстов, как правило, получают преимущество за счет использования этих зон в ходе обучения и классификации.

Завышение весов зон документа. Решая задачи классификации текстов, часто можно значительно повысить эффективность классификатора за счет приписывания разным зонам документа разных весов. Например, часто можно достичь высокой эффективности, приписав высокий вес словам из заголовка (Cohen and Singer, 1999). Эмпирическое правило гласит: вес слов из заголовков следует удваивать. Кроме того, можно приписывать повышенные веса определенным частям текста, которые не являются четко выраженными зонами, но структура документа или их содержание подсказывают, что они имеют большое значение. Мурата и др. (Murata et al., 2000) предположили, что эффективность поиска по запросу также можно повысить за счет приписывания повышенного веса первому предложению новостного документа.

Отдельные пространства признаков для зон документов. Существуют две стратегии работы с зонами документов. Выше мы приписывали повышенные веса словам, появляющимся в определенных зонах. Это значит, что мы используем те же самые признаки (т.е. параметры во всех зонах документов “связаны”), но в определенных зонах уделяем больше внимания их появлению. Альтернативная стратегия заключается в том, чтобы слова в разных зонах имели разные множества признаков и соответствующих параметров. В принципе, это более мощный подход: одно и то же слово в заголовке может указывать на тему *Middle East* (Ближний Восток), а в теле документа — на тему *Commodities* (Товары). Однако на практике связывание параметров чаще приводит к успеху. Разделение признаков означает увеличение количества параметров в несколько раз, причем многие из этих параметров в обучающих выборках могут встречаться редко. Это ухудшает оценки, в то время как завышение весов не оказывает такого эффекта. Более того, слова редко имеют разный смысл в разных зонах; скорее, можно вести речь об их весе. Тем не менее все это лишь предположения, зависящие от природы и количества обучающих данных.

Связь с реферированием текстов. В разделе 8.7 мы упоминали о реферировании текстов, а также о том, что большинство работ в этой области посвящено составлению реферата из фрагментов оригинального текста. Важные фрагменты (обычно предложения) выбираются на основании содержания и позиции в тексте. Методы автоматического реферирования можно использовать, чтобы выделить фрагменты текста, полезные для классификации. Например, Кольч и др. (Kołcz et al., 2000) рассмотрели разновидность выбора признаков, в которой документы классифицировались лишь по словам, находящимся в определенных зонах. Основываясь на исследованиях в области реферирования текстов, они использовали 1) только заглавие, 2) только первый абзац, 3) только абзац, содержащий наибольшее количество слов из заглавия или ключевых слов, 4) первые два абзаца или первый и последний абзацы и 5) все предложения, содержащие минимальное количество слов из заглавия или ключевых слов. В целом координатные методы выбора при-

знаков привели к таким же хорошим результатам, как и метод, использующий взаимную информацию (см. раздел 13.5.1), и позволили получить вполне конкурентоспособный классификатор. Ко и др. (Ko et al., 2004) также применили принципы автоматического реферирования и повышали веса предложений, содержащих слова из заглавия, или слов, отражающих смысл документа. Это позволило повысить правильность классификации почти на 1%. Вероятно, успех этого метода объясняется тем, что большинство таких предложений так или иначе отражают основной смысл документа.

? ***Упражнение 15.4* [**].** При рассылке спама по электронной почте применяются различные методы маскировки, препятствующие его фильтрации. Один из них заключается в многократном повторении или замене символов, чтобы сбить с толку классификаторы текстов. Например, в спамовом сообщении могут встретиться следующие слова.

Rep1icaRolex bonmus Viiiaaaagra pi11z
PHARlbdMACY [LEV]i[IT]l[RA] se^xual C1AfL1S

Подумайте, как сконструировать признаки, чтобы противостоять такой стратегии.

***Упражнение 15.5* [**].** Другая стратегия, используемая распространителями почтового спама, заключается в присоединении к пересылаемому сообщению (например, к предложению купить дешевые акции или что-нибудь еще) абзаца текста безвредного содержания (например, фрагмента статьи). Почему эта стратегия оказывается эффективной? Как решить данную проблему с помощью классификатора текстов?

***Упражнение 15.6* [*].** Какие еще типы признаков могли бы оказаться полезными для классификации почтового спама?

15.4. Методы машинного обучения для поиска по запросу

Вместо того чтобы подбирать функции взвешивания терминов и документов, как в главе 6, можно взглянуть на разные источники сигналов о релевантности (косинусная мера, совпадение заглавий и т.д.) как на признаки в задаче обучения. Классификатор, получивший примеры релевантных и нерелевантных документов для каждого запроса из некоторого множества, может вычислить относительный вес этих сигналов. Если сформулировать задачу с помощью пар "документ–запрос", которым назначена оценка "*релевантный*" или "*нерелевантный*", то ее также можно интерпретировать как задачу классификации текстов. Такой подход не всегда оказывается самым лучшим, и его альтернатива будет описана в разделе 15.4.2. Тем не менее в рамках уже изложенного материала проще всего подойти к этой задаче с точки зрения классификации текстов, упорядочивая документы в соответствии с доверительным уровнем решения, принятого бинарным классификатором (два класса — релевантный и нерелеватный). Может быть, это не вполне корректно дидактически, но именно такой подход иногда используется на практике.

15.4.1. Простой пример ранжирования на основе машинного обучения

В этом разделе мы обобщим методологию, изложенную в разделе 6.1.2, для задачи *машинного обучения* функции ранжирования документа. В разделе 6.1.2 мы описали

случай, когда следует комбинировать булевы индикаторы релевантности; здесь мы рассматриваем более общие факторы, чтобы развить идею *релевантности на основе машинного обучения* (machine-learned relevance). В частности, факторы, которые мы рассмотрим, выходят за рамки булевых функций присутствия термина запроса в зонах документа, как в разделе 6.1.2.

Будем считать, что функция ранжирования представляет собой линейную комбинацию двух факторов: 1) косинусной меры сходства между запросом и документом в векторном пространстве и 2) минимальной ширины окна ω, внутри которого лежат термины запроса. Как указывалось в разделе 7.2.2, близость терминов запроса (в документе) является хорошим индикатором релевантности, особенно в случае длинных документов и веб-страниц. Кроме того, этот показатель позволяет учесть скрытые словосочетания. Таким образом, у нас есть один фактор, зависящий от статистических характеристик терминов запроса в документе, рассматриваемом как "мешок слов", и другой фактор, зависящий от близости терминов запроса в документе. Мы ограничимся только этими двумя факторами, поскольку это упростит изложение, хотя данный метод можно обобщить на случай произвольного количества факторов.

Как и в разделе 6.1.2, мы работаем с *обучающими примерами* (training examples), каждый из которых представляет собой пару, состоящую из запроса и документа, а также оценку релевантности этого документа данному запросу: *релевантный* или *нерелевантный*. Для каждого такого примера можно вычислить косинусную меру сходства, а также ширину окна ω. В результате возникнет обучающее множество (табл. 15.3, напоминающая рис. 6.5).

Здесь два признака (косинусная мера сходства α и ширина окна ω) представляют собой действительные числа. Если обозначить оценку *релевантный* единицей, а оценку *нерелевантный* — нулем, то задачу можно сформулировать так: найти функцию ранжирования, комбинирующую значения признаков и принимающую значения, близкие к нулю или единице. Эта функция должна быть как можно лучше согласована с совокупностью обучающих примеров. Не ограничивая общности, будем считать, что искомый линейный классификатор имеет следующий вид.

$$Score(d, q) = Score(\alpha, \omega) = a\alpha + b\omega + c \qquad (15.17)$$

Здесь коэффициенты a, b и c вычисляются в ходе обучения. Хотя эту задачу можно сформулировать как задачу минимизации ошибки (см. раздел 6.1.2), с дидактической точки зрения полезнее проиллюстрировать геометрический смысл равенства (15.17). Примеры, перечисленные в табл. 15.3, можно изобразить на плоскости с осями, соответствующими косинусной схожести α и ширине окна ω (рис. 15.7).

Таблица 15.3. Обучающие примеры для ранжирования на основе машинного обучения

Пример	*DocID*	*Запрос*	*Косинусная схожесть*	ω	*Оценка*
Φ_1	37	linux operating system	0,032	3	Релевантный
Φ_2	37	penguin logo	0,020	4	Нерелевантный
Φ_3	238	operating system	0,043	2	Релевантный
Φ_4	238	runtime environment	0,004	2	Нерелевантный
Φ_5	1741	kernel layer	0,022	3	Релевантный

Окончание табл. 15.3

Пример	*DocID*	*Запрос*	*Косинусная схожесть*	ω	*Оценка*
Φ_6	2094	device driver	0,030	2	Релевантный
Φ_7	3191	device driver	0,027	5	Нерелевантный
...	...	...	...	..	...

В этой задаче функция *Score*(α, ω) из равенства (15.17) определяет плоскость, "висящую" над рис. 15.7. В идеальном случае эта плоскость в направлении, перпендикулярном странице, содержит значения функции, близкие к единице над точками с меткой R и близкие к нулю над точками с меткой N. Поскольку маловероятно, что эта плоскость будет содержать только значения, близкие к нулю и единице, над точками обучающей выборки, следует использовать *пороговое значение* θ. Для любого запроса и документа, релевантность которого нужно определить: если *Score*(α, ω) > θ, то документ объявляется *релевантным*, а иначе он объявляется *нерелевантным*. Как показывает анализ рис. 14.8, все точки, удовлетворяющие условию *Score*(α, ω) = θ, лежат на линии (пунктирная линия на рис. 15.7). Следовательно, мы построили линейный классификатор, разделяющий релевантные и нерелевантные экземпляры. С геометрической точки зрения разделяющую линию можно найти следующим образом. Рассмотрим линию, лежащую на плоскости *Score*(α, ω) на высоте θ над рис. 15.7. Спроецируем ее на рис. 15.7. В результате получим пунктирную линию, изображенную на рис. 15.7. Любая следующая пара "документ–запрос", расположенная ниже пунктирной линии, считается *нерелевантной*, а пара, лежащая выше пунктирной линии, — *релевантной*.

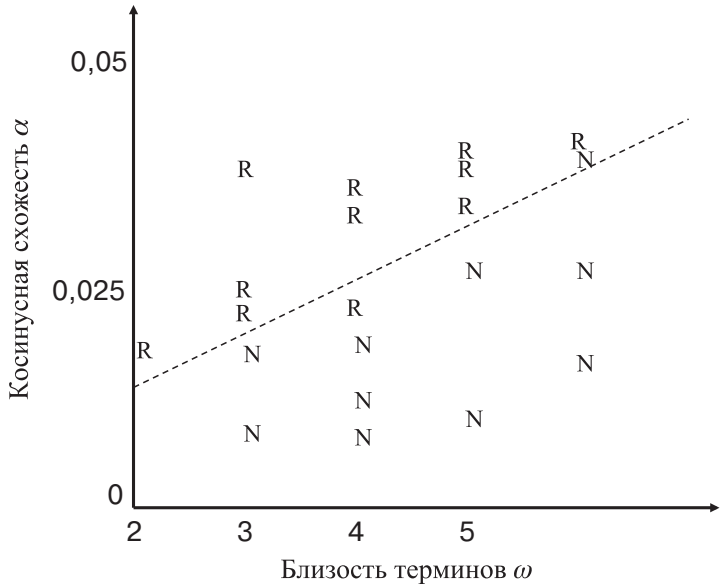


Рис. 15.7. Коллекция обучающих примеров. Каждая буква R означает обучающий пример, помеченный как релевантный, *а каждая буква N — обучающий пример, помеченный как* нерелевантный

Таким образом, проблема бинарной разметки *релевантных* и *нерелевантных* документов при наличии обучающей выборки, как показано выше, сводится к построению линии, отделяющей *релевантные* обучающие примеры от *нерелевантных*. Поскольку разделяющая линия лежит в плоскости α–ω, ее можно описать уравнением, зависящим от переменных α и ω, а также от двух коэффициентов — углового коэффициента и точки пересечения с осью O. Эту линию можно найти, используя любой из методов, описанных в главах 13–15. Если коллекция обучающих примеров достаточно велика, то можно полностью избежать ручной настройки функции (как мы это делали в разделе 7.2.3). Разумеется, узким местом этого подхода является репрезентативность множества обучающих примеров, релевантность которых должны оценивать эксперты.

15.4.2. Ранжирование результатов с помощью машинного обучения

Изложенные выше идеи можно легко обобщить на функции, зависящие от нескольких переменных. Существует множество других количественных показателей, с помощью которых можно оценивать релевантность документа запросу, например статические показатели качества документа (показатели наподобие PageRank, рассматриваемые в главе 21), возраст документа, вклады различных зон, длина документа и т.д. Если предположить, что эти показатели можно вычислить для коллекции обучающих примеров, то для настройки классификатора на основе машинного обучения можно использовать любое количество таких показателей. Например, можно настроить метод опорных векторов на основе бинарных оценок релевантности и упорядочивать документы по вероятности их релевантности, которая монотонно изменяется по мере изменения расстояния (с учетом знака) от документа до разделяющей поверхности.

Однако такой подход к ранжированию результатов информационного поиска не всегда обоснован. Статистики обычно различают задачи *классификации* (в которых предсказывается категориальная переменная) и *регрессионные* задачи (в которых предсказывается действительное число). Между этими полюсами расположена специальная область *порядковой регрессии* (ordinal regression), в которой предсказываются ранги (ранжирование). Машинное обучение для поиска по запросу следует, прежде всего, рассматривать как задачу ординальной регрессии, цель которой — ранжировать совокупность документов по отношению к запросу на основе обучающих примеров того же рода. Эта формулировка является более сильной, поскольку в соответствии с ней релевантность документов можно оценивать относительно других документов-кандидатов для этого запроса, а не по абсолютной шкале, что упрощает задачу: ведь нам требуется ранжировать документы, а не определить их абсолютную меру релевантности. Вопросы, связанные с ранжированием, особенно уместны при поиске документов в вебе, когда ранжирование верхушки списка результатов намного важнее оценок релевантности отдельных документов. Такого рода задачи ставились и решались в рамках структурного метода опорных векторов, упомянутого в разделе 15.2.2. В данном методе искомым классом является ранжированное множество документов. Однако здесь мы опишем более простой ранжирующий метод опорных векторов.

Конструирование *ранжирующего метода опорных векторов* (ranking SVM) осуществляется следующим образом. Начнем с набора запросов. Для каждого обучающего запроса q у нас есть совокупность документов, полностью упорядоченная по релевантности запросу в соответствии с оценками человека. Для каждой пары “документ–запрос”, используя признаки, описанные в разделе 15.4.1, и многие другие, построим вектор при-

знаков $\psi_j = \psi(d_j, q)$. Затем для двух документов, d_i и d_j, сформируем вектор разностей между признаками.

$$\Phi(d_i, d_j, q) = \psi(d_i, q) - \psi(d_j, q) \tag{15.18}$$

В соответствии с гипотезой один из документов, d_i или d_j, должен быть более релевантным. Если документ d_i, по мнению эксперта, более релевантен, чем документ d_j, — этот факт обозначается как $d_i \prec d_j$ (документ d_i должен предшествовать документу d_j в упорядоченном списке результатов), — то вектор $\Phi(d_i, d_j, q)$ присваивается классу $y_{ijq} = +1$, в противном случае — классу $y_{ijq} = -1$. Наша цель — построить классификатор, возвращающий результат

$$\vec{w}^T \Phi\left(d_i, d_j, q\right) > 0 \text{ тогда и только тогда, когда } d_i \prec d_j. \tag{15.19}$$

Эту задачу обучения метода опорных векторов можно сформулировать точно так же, как раньше.

Найти вектор $\vec{w}$ и числа $\xi_{ij} \geq 0$, удовлетворяющие следующим условиям. (15.20)

- Величина $\frac{1}{2}\vec{w}^T\vec{w} + C\sum_{i,j}\xi_{ij}$ достигает минимума.
- При всех $\{\Phi(d_i, d_j, q): d_i \prec d_j,\ \vec{w}^T\Phi\left(d_i, d_j, q\right) \geq 1 - \xi_{ij}$.

Классы y_{ijq} в ограничениях можно не упоминать. Необходимо лишь рассмотреть ограничение на пары документов, упорядоченных в одном направлении, поскольку отношение $\prec$ является антисимметричным. Решение этой задачи приводит к линейному классификатору, который может ранжировать пары документов. Этот подход был применен для создания ранжирующих функций, эффективность которых на стандартных наборах данных превышает эффективность стандартных созданных вручную функций ранжирования. Статьи по этой теме перечислены в конце главы.

В обоих методах, рассмотренных выше, используется линейное взвешивание признаков документов, играющих роль индикаторов релевантности. Это очень распространенный прием. Интересно, что большинство схем взвешивания в теории информационного поиска основано на *нелинейном* масштабировании основных показателей (например, логарифмическое взвешивание частоты терминов или idf). В настоящее время машинное обучение является очень эффективным способом получения оптимальных весов для признаков в линейной комбинации (или в аналогичных ограниченных классах моделей), но при нелинейном масштабировании основных показателей его эффективность снижается. По этой причине в этих задачах по-прежнему используется конструирование признаков с помощью экспертов.

Идея построения функций ранжирования с помощью обучения возникла много лет назад, но лишь недавно этот метод стал реализуемым и привлекательным благодаря накопленным знаниям в области машинного обучения, коллекциям документов для обучения и вычислительным мощностям. Таким образом, сейчас еще очень рано говорить что-либо определенное о машинном обучении для ранжирования результатов информационного поиска, но есть основания надеяться, что со временем этот подход будет применяться более широко.[8] Несмотря на то что опытные эксперты легко могут создавать очень качественные функции ранжирования, ручная настройка — сложный процесс, и

[8] К моменту выхода перевода книги появилась обширная литература по данному вопросу: см. материалы конкурсов и семинаров LETOR, IMAT, LR4IR, статью по Learning To Rank в Википедии и т.д. — *Примеч. ред.*

для каждой новой коллекции документов и нового класса пользователей его приходится повторять заново.

Упражнение 15.7. Постройте график в плоскости α–ω по первым семи строкам в табл. 15.3, ориентируясь на рис. 15.7.

Упражнение 15.8. Запишите уравнение линии в плоскости α–ω, отделяющей релевантные документы от нерелевантных.

Упражнение 15.9. Приведите обучающий пример (состоящий из значений α, ω и выводов о релевантности), который после добавления в обучающее множество делает невозможным разделение релевантных и нерелевантных документов с помощью линии на плоскости α–ω.

15.5. Библиография и рекомендации для дальнейшего чтения

Несколько вычурное название *метод опорных векторов* (буквально — машины опорных векторов) взято из научных работ по нейронным сетям, в которых обучающие алгоритмы рассматриваются как элементы архитектуры, т.е. как "машины". Отличительной чертой этой модели является то, что разделяющая поверхность полностью определена несколькими обучающими точками ("опирается" на эти точки), т.е. опорными векторами.

Более подробное описание метода опорных векторов изложено в работе Бурже (Burges, 1998). Чен и др. (Chen et al., 2005) ввели так называемый метод ν-SVM, обеспечивающий альтернативную параметризацию для решения неразделимых задач, где вместо введения штрафа C указывается параметр ν, ограничивающий количество примеров, которые могут появляться на неверной стороне разделяющей поверхности. В настоящее время издано несколько книг по методу опорных векторов, обучению с широком зазором и ядрами: как с точки зрения математической теории (Cristiani ans Shawe-Taylor, 2000; Schülkopf and Smola, 2001), так и с практической точки зрения (Shawe-Taylor and Cristiani, 2004). Основы метода изложены в книге его изобретателя Вапника (Vapnik, 1998) [9]. Недавние монографии, посвященные обучению с помощью статистика, например Хасте и др. (Haste et al., 2001), содержат подробное описание метода опорных векторов.

Конструирование *многоклассовых методов опорных векторов* (multiclass SVMs) обсуждается в работах Уестона и Уоткинса (Westin and Watkins, 1999), Краммера и Зингера (Crammer and Singer, 2001), а также Цочантаридиса и др. (Tsochantaridis et al., 2005). В последней работе изложено введение в структурный метод опорных векторов.

Переход к ядру впервые представлен в работе Айзермана и др. (Aizerman et al., 1964). Строковые ядра и другие ядра для структурированных данных описаны в книгах Лодхи и др. (Lodhi et al., 2002) и Гартнера и др. (Gaertner et al., 2002). Основным местом для обнародования результатов теоретических работ по машинному обучению, например по методу опорных векторов, стали конференции *Advances in Neural Information Processing* (NIPS). Другие конференции, такие как SIGIR, уделяют больше внимания экспериментальным исследованиям и использованию специфических признаков текста для улучшения эффективности классификаторов.

---

[9] См. Вапник В.Н. Восстановление зависимостей по эмпирическим данным. — М.: Наука, 1979. — 448 с. — *Примеч. ред.*

Результаты недавнего сравнения современных классификаторов, основанных на машинном обучении (впрочем, на задачах, которые отличаются от типичных задач классификации текстов), приведены в работе Каруана и Никулеску-Мизил (Caruana and Niculescu-Mizil, 2006). Наиболее свежие результаты сравнительной оценки классификаторов на основе машинного обучения при решении задач классификации текстов обсуждаются в работе Ли и Янга (Li and Yang, 2003), процитированной в разделе 13.6. Более ранние результаты сравнения классификаторов на примере классификации текстов приведены в работах Янга, Янда, Ли, Дюмэ и др. (Yang, 1999; Yand and Li, 1999; Dumais et al., 1998). Йоахимс (Joachims, 2002*a*) опубликовал детальное описание применения метода опорных векторов для решения задач обработки текстов. Чанг и Олес (Zhang and Oles, 2001) в своей работе провели глубокое сравнение наивного байесовского метода, регуляризированной логистической регрессии и метода опорных векторов.

Йоахимс (Joachims, 1999) описал возможности повысить практическую ценность метода опорных векторов на больших наборах данных, а затем развил эти результаты (Joachims, 2006).

Для решения задач, в которых классы образуют иерархическую структуру, предложено много методов (Koller and Sahami, 1997; McCallum et al., 1998; Weigend et al., 1999; Dumais and Chen, 2000). В недавнем крупном исследовании проблем масштабирования метода опорных векторов на весь каталог Yahoo! Лиу и др. (Liu et al., 2005) пришли к выводу, что иерархическая классификация достойна внимания, но, как и ранее, ненамного превышает эффективность обычной (“плоской”) классификации. Эффективность классификации по-прежнему ограничивается очень небольшим количеством обучающих документов для многих классов. Более общий подход, который можно применить для моделирования произвольных, а не только иерархических отношений между классами, описан в работе Цочантаридиса и др. (Tsochantaridis et al., 2005).

Москитти и Базили (Moschitti and Basili, 2004) исследовали использование сложных имен существительных, имен собственных и смыслов слов в качестве признаков текстовой классификации.

Диттерих (Dietterich, 2002) опубликовал обзор методов на основе ансамблей классификаторов, а Шапире (Schapire, 2003) сосредоточился на методе бустинга (Schapire and Singer, 2000).

Книга Шапеля и др. (Chapelle et al., 2006) содержит введение в методы с частичным обучением, включая главы о применении EM-алгоритма к задачам классификации текстов (Nigam et al., 2006) и трансдуктивном методе опорных векторов (Joachims, 2006*b*). Синдхвани и Кеерти (Sindhwani and Keerthi, 2006) описывают более эффективную реализацию трансдуктивного метода опорных векторов для крупных наборов данных.

Тонг и Коллер (Tong and Koller, 2001) исследовали активное обучение с использованием метода опорных векторов применительно к задачам классификации текстов; Белдридж и Осборн (Beldridge and Osborne, 2004) указали, что примеры, выбранные для разметки одним классификатором в контексте активного обучения, могут оказаться не лучше случайных, если их использовать для обучения другого классификатора.

Машинное обучение для ранжирования результатов поиска по запросу впервые было предложено в работах Вонга, Фура и Гея (Wong et al., 1988; Fuhr, 1992; Gey, 1994). Однако ограниченность обучающих данных и слабые методы машинного обучения не позволили получить результаты, превышающие средний уровень, и в то время не оказали влияния на методы информационного поиска.

Тейлор и др. (Taylor et al., 2006) применили методы машинного обучения для настройки параметров функций ранжирования из семейства BM25 (см. раздел 11.4.3) и максимизации показателя NDCG (см. раздел 8.4). Подходы на основе машинного обучения к порядковой регрессии описаны в работах Хербриха и др. (Herbrich et al., 2000), Бурже и др. (Burges et al., 2005) и были применены к данным о кликах пользователей в работе Йоахимса (Joachims, 2002*b*). Као и др. (Cao et al., 2006) изучили, насколько эффективным может оказаться этот подход в задачах информационного поиска, а Куин и др. (Qin et al., 2007) предложили их расширение с помощью нескольких гиперплоскостей. Юе и др. (Yue et al., 2007) исследовали ранжирование с помощью структурного метода опорных векторов и, в частности, показали, что эту конструкцию можно эффективно использовать для непосредственной оптимизации показателя MAP (см. раздел 8.4) вместо применения заменителей, таких как правильность или площадь фигуры под кривой ROC. Генг и др. (Geng et al., 2007) исследовали выбор признаков в задаче ранжирования.

Другие подходы к обучению с целью ранжирования также оказались эффективными для веб-поиска (Burges et al., 2005; Richardson et al., 2006).

Глава 16

Плоская кластеризация

Алгоритмы кластеризации разделяют совокупность документов на подмножества, или *кластеры* (clusters). Цель этих алгоритмов — создать кластеры, однородные внутри, но четко отличающиеся друг от друга. Иначе говоря, документы внутри кластера должны быть максимально похожими друг на друга, но в то же время максимально отличаться от документов другого кластера.

Кластеризация (clustering) — это наиболее распространенная форма *обучения без учителя* (unsupervised learning). Отсутствие учителя означает, что в алгоритме не предусмотрено участие эксперта, присваивающего документы классам. В задачах кластеризации распределение и структура данных определяют принадлежность к кластеру. Простой пример кластеризации приведен на рис. 16.1. На нем невооруженным глазом видны три отдельных кластера точек. В этой и следующей главах описываются алгоритмы, с помощью которых можно найти такие кластеры, не прибегая к обучающим примерам.

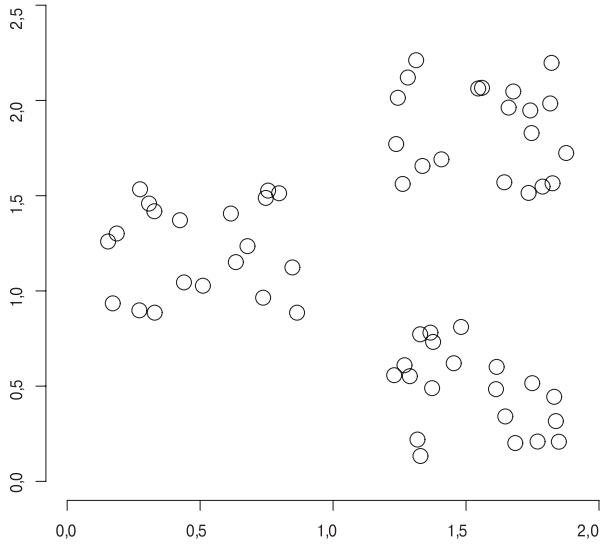


Рис. 16.1. Пример данных с четкой кластерной структурой

Разница между кластеризацией и классификацией, на первый взгляд, может показаться незначительной. В конце концов, в обоих случаях множество документов разделяется на группы. Однако вскоре мы убедимся, что между этими задачами существует фундаментальная разница. Классификация является разновидностью обучения с учителем (см. главу 13). Ее цель — воспроизвести разделение данных по категориям, установленное экспертом. В обучении без учителя, наиболее важным примером которого является кластеризация, учителя нет.

Основной входной информацией для алгоритма кластеризации является метрика. На рис. 16.1 в качестве метрики используется расстояние между точками на плоскости. Эта метрика позволяет выделить три разных кластера, показанных на рисунке. При кластеризации документов часто используется евклидово расстояние. Изменение метрики может повлиять на результаты кластеризации. Таким образом, метрика — важный инструмент, с помощью которого можно изменять результаты кластеризации.

Плоская кластеризация (flat clustering) порождает совокупность кластеров, не имеющих явных взаимосвязей. *Иерархическая кластеризация* (hierarchical clustering) создает иерархию кластеров (глава 17). В главе 17 также рассматривается сложная задача автоматического назначения кластерам меток.

Так же важно различать жесткую и мягкую кластеризацию. *Жесткая кластеризация* (hard clustering) вычисляет *жесткое присваивание* — каждый документ может быть элементом только одного кластера. Присваивание в алгоритмах *мягкой кластеризации* (soft clustering) является мягким, принадлежность документа распределена по всем кластерам. При мягком присваивании документ может частично принадлежать нескольким кластерам. Примером алгоритма мягкой кластеризации является латентное семантическое индексирование — одна из форм снижения размерности, описанная в главе 18.

В этой главе на многочисленных примерах обосновывается применение кластеризации в задачах информационного поиска (раздел 16.1), формулируется задача, которая решается в ходе кластеризации (раздел 16.2), и обсуждаются показатели качества кластеров (раздел 16.3). Затем описываются два алгоритма плоской кластеризации: метод *K*-средних (раздел 16.4), представляющий собой алгоритм жесткой кластеризации, и метод *expectation-maximization*, или EM-алгоритм (раздел 16.5), относящийся к мягкой кластеризации. Благодаря простоте и эффективности метод *K*-средних, вероятно, используется чаще других алгоритмов плоской кластеризации. EM-алгоритм — это обобщение метода *K*-средних. Его можно применять к широкому спектру представлений и распределений документов.

16.1. Кластеризация в информационном поиске

Кластерная гипотеза (cluster hypothesis) — это основное предположение, позволяющее применять кластеризацию для информационного поиска.

> **Кластерная гипотеза.** Документы, принадлежащие одному и тому же кластеру, являются примерно одинаково релевантными по отношению к информационным потребностям.

Эта гипотеза утверждает, что если документ принадлежит кластеру и является релевантным по отношению к запросу, то вполне вероятно, что и другие документы этого кластера релевантны. Это объясняется тем, что кластеризация объединяет документы, содержащие много общих терминов. Кластерная гипотеза по существу является гипотезой о смежности, сформулированной в главе 14. В обоих случаях мы постулируем, что похожие документы обладают примерно одинаковыми свойствами релевантности.

В табл. 16.1 перечислены основные приложения кластеризации в области информационного поиска. Они отличаются по множествам документов, которые подвергаются кластеризации (результаты поиска, коллекция или подмножества коллекции), и по аспектам информационно-поисковой системы, которые при этом улучшаются (качество или производительность поиска). При этом все они опираются на основную гипотезу о кластерах.

Таблица 16.1. Некоторые приложения кластеризации в области информационного поиска

Приложение	*Что подвергается кластеризации*	*Выгода*	*Пример*
Кластеризация результатов поиска	Результаты поиска	Более качественное представление информации	Рис. 16.2
Разбиение и объединение	Подмножества коллекций	Альтернативный пользовательский интерфейс: "поиск без ввода слов"	Рис. 16.3
Кластеризация коллекции	Коллекция	Более качественное представление информации для навигации пользователей	McKeown et al. (2002), `http://news.google.com`
Языковые модели	Коллекция	Повышение точности и/или полноты	Liu and Croft (2004)
Кластерный поиск	Коллекция	Повышение производительности: скорости поиска	Salton (1971a)

Первой в табл. 16.1 упоминается *кластеризация результатов поиска*, в которой под результатами поиска подразумеваются документы, возвращенные в ответ на запрос. По умолчанию результаты поиска в информационно-поисковых системах представляются в виде простого списка. Пользователи просматривают этот список сверху вниз, пока не найдут нужную им информацию. В то же время после кластеризации результатов поиска похожие документы будут указаны в списке недалеко друг от друга. Иногда легче просматривать группы однородных документов, а не множество отдельных документов. Это особенно полезно, если термин запроса имеет несколько значений. Например, на рис. 16.2 показан запрос jaguar. Три наиболее распространенных значения этого слова —

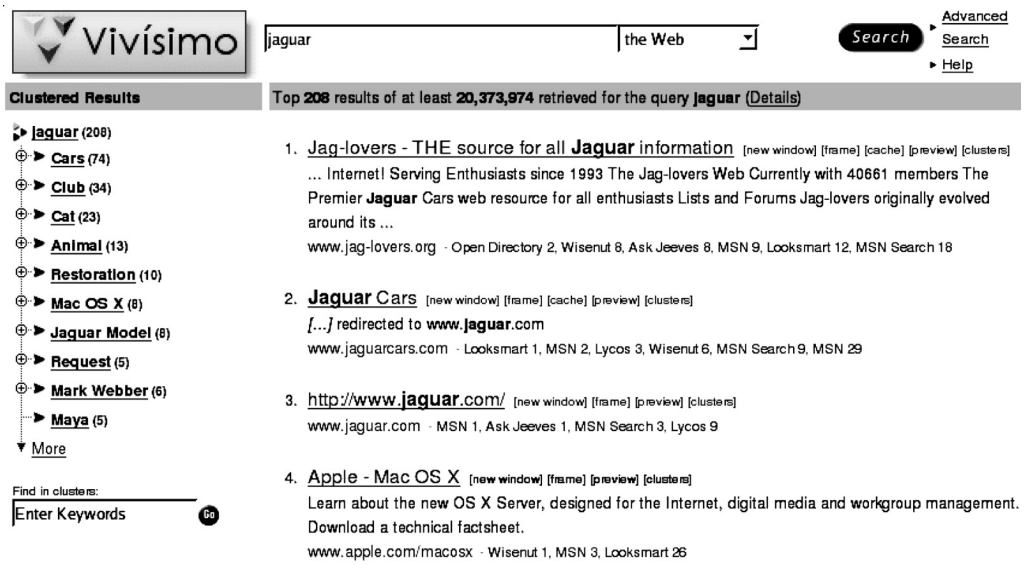


Рис. 16.2. Кластеризация результатов поиска повышает его полноту. Ни один из первых ответов не соответствует смыслу "животное" слова jaguar*, однако, щелкнув на кластере* Cat *панели* Clustered Results *(третья стрелка сверху), можно легко найти это значение*

автомобиль, животное и операционная система компьютера Apple. Панель *Clustered Results* в поисковой системе Vivisimo (`http://vivisimo.com`) представляет собой более рациональный пользовательский интерфейс, позволяющий лучше понять содержание результатов поиска по сравнению с обычным списком документов.

Более удобный пользовательский интерфейс можно также создать с помощью *разбиения и объединения* (scatter-gather). Это приложение указано в табл. 16.1 вторым. Метод разбиения и объединения *разбивает* (scatter) всю коллекцию на кластеры, чтобы получить группы документов, которые пользователь может выбрать или *объединить* (gather). Выбранные группы объединяются, а совокупность результатов вновь разделяется на кластеры. Этот процесс повторяется до тех пор, пока не будет найден искомый кластер. Пример показан на рис. 16.3.

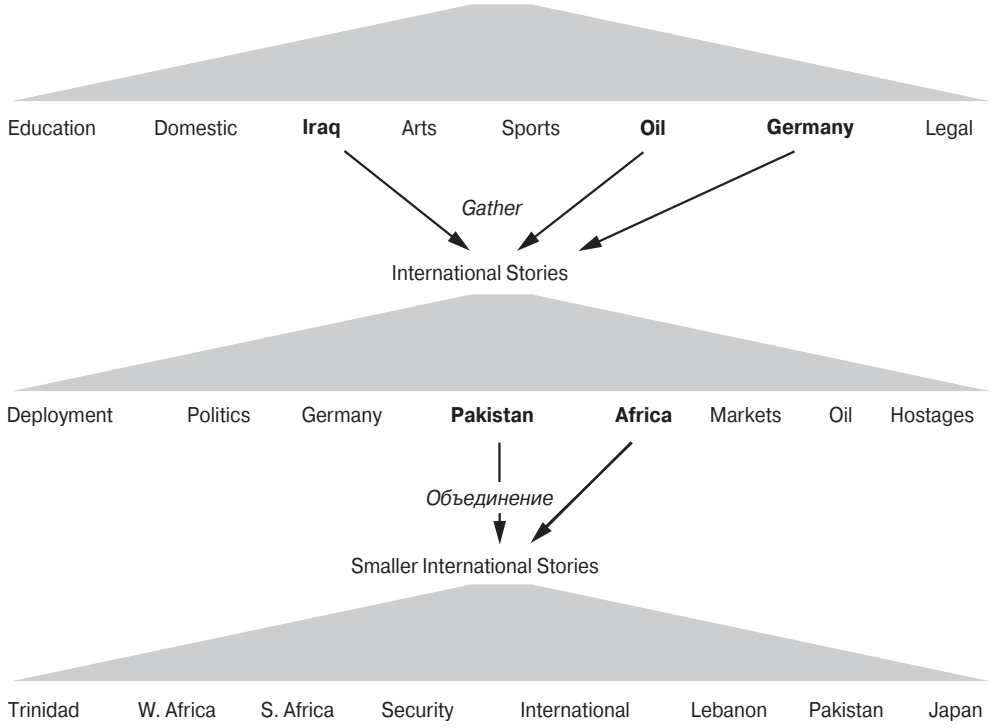


Рис. 16.3. Пример работы пользователя по методу разбиения и объединения. Коллекция новостей, опубликованных в газете New York Times*, разбивается на восемь кластеров (верхняя строка). Пользователь вручную* собирает *три из них в меньшую коллекцию* International Stories *и снова выполняет разбиение. Этот процесс повторяется до тех пор, пока не будет найден небольшой кластер с* релевантными *документами (например,* Trinidad*)*

Автоматически сгенерированные кластеры, такие как на рис. 16.3, не так аккуратно организованы, как вручную сконструированное иерархическое дерево, подобное каталогу Open Directory (`http://dmoz.org`). Кроме того, автоматический поиск меток, описывающих кластеры, представляет собой сложную проблему (раздел 17.7). Однако навигация по кластерам является интересной альтернативой стандартному поиску по ключе-

вым словам. Это особенно верно в ситуациях, когда пользователь предпочитает запросам переходы по ссылкам, поскольку не уверен в выборе терминов запроса.

В качестве альтернативы итеративной кластеризации, управляемой пользователем, в методе разбиения и объединения можно осуществить статическую иерархическую кластеризацию коллекции без участия пользователя (в табл. 16.1 это приложение называется "Кластеризация коллекции"). Примерами такого подхода являются новостная поисковая система Google News и ее предшественница — система Columbia NewsBlaster. В случае новостей мы должны проводить кластеризацию часто, чтобы пользователь имел доступ к самым свежим новостям. Кластеризация хорошо подходит работы с коллекциями новостей, поскольку чтение новостей на самом деле является не поиском, а процессом выбора подмножества сообщений о недавних событиях.

Четвертое приложение кластеризации использует гипотезу о кластерах напрямую для улучшения результатов поиска за счет кластеризации всей коллекции. Для идентификации начального множества документов, соответствующих запросу, используется стандартный инвертированный индекс. Затем к этому множеству добавляются другие документы из того же кластера, даже если они слабо связаны с запросом. Например, если поступил запрос car, то будет извлечено несколько документов из кластера, посвященного автомобилям, а затем к ним будут добавлены документы из этого кластера, в которых используются другие термины (automobile, vehicle и т.д.). Это повышает полноту поиска; группа документов с высокой взаимной схожестью часто образует группу релевантных документов.

Недавно эта идея была использована для построения языковых моделей (language modeling). Из формулы (12.10) следует, что для решения проблемы разреженных данных, возникающей при построении языковых моделей для информационного поиска, модель документа d можно интерполировать с моделью коллекции. Однако эта коллекция содержит много документов, термины в которых не характерны для документа d. Заменяя модель коллекции моделью, построенной по кластеру, которому принадлежит документ d, можно получить более точные оценки вероятностей появления терминов в документе d.

Кластеризация может также ускорить поиск. Как показано в разделе 6.3.2, поиск с помощью модели векторного пространства равнозначен поиску ближайших соседей запроса. Инвертированный индекс обеспечивает быстрый поиск ближайшего соседа при стандартных условиях информационного поиска. Однако иногда инвертированный индекс невозможно использовать эффективно, например, при латентном семантическом индексировании (глава 18). В таких ситуациях можно было бы оценить сходство запроса с каждым документом, но это слишком медленный процесс. Кластерная гипотеза предлагает альтернативу: найти кластер, ближайший к запросу, и рассматривать только документы из данного кластера. Для этого (намного меньшего) множества можно вычислить сходство запроса с каждым документом и ранжировать документы обычным образом. Поскольку кластеров намного меньше, чем документов, поиск ближайшего кластера выполняется быстро, а поскольку документы, соответствующие запросу, похожи друг на друга, они, как правило, принадлежат одному и тому же кластеру. Хотя этот алгоритм неточен, ожидаемое снижение качества поиска незначительно. Это использование кластеризации описано в разделе 7.1.6.

? ***Упражнение 16.1.*** Будем считать документы похожими, если они оба содержат по крайней мере два имени собственных, например Clinton и Sarkozy. Приведите пример информационной потребности и двух документов, для которых кластерная гипотеза при таких условиях схожести *не* выполняется.

Упражнение 16.2. Разработайте простой одномерный пример (т.е. точки на линии) с двумя кластерами, демонстрирующий неточность кластерного информационного поиска. В вашем примере поиск кластера, ближайшего к запросу, должен быть хуже, чем непосредственный поиск ближайшего соседа.

16.2. Формулировка задачи

Цель плоской жесткой кластеризации формулируется следующим образом. Дано: 1) множество документов $D = \{d_1, d_2, ..., d_N\}$, 2) желательное количество кластеров K, 3) *целевая функция* (objective function), оценивающая качество кластеризации. Необходимо вычислить присваивание γ: $D \rightarrow \{1, ..., K\}$, минимизирующую (или максимизирующую) целевую функцию. В большинстве случаев также требуется, чтобы функция γ была сюръективной, т.е. ни один из K кластеров не должен быть пустым.

Целевая функция часто определяется в терминах сходства или расстояния между документами. Ниже будет показано, что целью кластеризации по методу K-средних является минимизация среднего расстояния между документами и их центроидами или, что эквивалентно, максимизация сходства между документами и их центроидами. Меры сходства и метрики, которые обсуждались в главе 14, также применимы для задач кластеризации. В этой главе, как и в главе 14, говоря о связи между документами, мы используем как сходство, так и расстояние между ними.

Сходство документов обычно выражается в виде одной из функций тематического сходства, или просто высоких значений на одних и тех же осях в модели векторного пространства.[1] Например, документы о Китае имеют большие значения на осях Chinese, Beijing и Mao, а документы о Великобритании — на осях London, Britain è Queen. Тематическое сходство документов выражается в виде косинусной меры сходства или евклидова расстояния в векторном пространстве (глава 6). Если пользователей интересует не тематическое, а другое сходство документов, например сходство языка, то можно выбрать другое представление. При вычислении тематического сходства стоп-слова можно совершенно спокойно игнорировать, но они играют важную роль при разделении кластеров, содержащих документы на английском языке (в которых артикль the появляется часто, а артикль la — редко) и французском языке (в которых артикль la появляется часто, а артикль the — редко).

Терминологические замечания. Альтернативное определение жесткой кластеризации заключается в том, что документ может быть полноценным элементом более чем одного кластера.[2] *Разделяющая кластеризация* (partitional clustering) всегда означает, что каждый документ принадлежит только одному классу. (Однако в разделяющей иерархической кластеризации (глава 17) все элементы кластера, разумеется, являются элементами его родительского кластера.) При альтернативном определении жесткой кластеризации, допускающем множественную принадлежность, разница между мягкой и жесткой кластеризациями заключается в том, что при жесткой кластеризации функция принад-

[1] В вебе, социальных сетях и библиометрии сходство документов может выражаться с помощью метрик, основанных на ссылках, таких как совместное цитирование (co-citation) или библиографическое сочетание (bibliographic coupling). — *Примеч. ред.*

[2] Поэтому мы не используем для перевода “hard clustering” термин “однозначная кластеризация”, который иногда встречается в отечественной литературе. — *Примеч. ред.*

лежности принимает значение либо 0, либо 1, в то время как при мягкой кластеризации она может принимать любое неотрицательное значение.

Некоторые исследователи проводят различие между *полной кластеризацией* (exhaustive clustering), в которой каждый документ присваивается какому-то кластеру, и *частичной кластеризацией* (nonexhaustive clustering), в которой некоторые документы могут не присваиваться ни одному кластеру. Частичная кластеризация, в которой каждый документ может принадлежать либо одному, либо ни одному кластеру, называется *исключающей* (exclusive). В этой книге мы всегда считаем кластеризацию полной.

16.2.1. Мощность кластеризации: количество кластеров

При кластеризации трудно определить количество кластеров, или *мощность кластеризации* (cardinality of a clustering), которую мы обозначаем буквой K. Часто число K — не более чем догадка, основанная на опыте или предметных знаниях. Однако в методе K-средних используется также эвристический метод для выбора числа K и предпринимается попытка включить выбор числа K в целевую функцию. Иногда приложения накладывают ограничения на диапазон числа K. Например, интерфейс, созданный по принципу "разбиение–объединение" и показанный на рис. 16.3, не может выводить на экран больше десяти кластеров на один слой. Это число продиктовано размерами и разрешением дисплеев компьютеров в начале 1990-х годов.

Поскольку наша цель — оптимизировать целевую функцию, кластеризация по существу является задачей поиска. Используя подход "полного перебора", можно было бы просто перенумеровать все возможные кластеры и выбрать наилучший. Однако количество возможных разбиений возрастает экспоненциально, поэтому данный подход на практике реализовать невозможно. [3] По этой причине большинство алгоритмов плоской кластеризации итеративно уточняют начальное разбиение. Если поиск начинается с неудачной отправной точки, то мы можем пропустить точку глобального оптимума. Следовательно, выбор подходящей начальной точки представляет собой другую важную задачу плоской кластеризации.

16.3. Оценивание кластеризации

Типичная целевая функция в кластеризации решает задачу максимизации сходства между документами внутри кластера (документы внутри кластера похожи друг на друга) и минимизации сходства между кластерами (документы из разных кластеров отличаются друг от друга). Так можно построить *внутренний критерий* (internal criterion) качества кластеризации. Однако хорошие показатели внутреннего критерия еще не означают высокого качества приложения. Альтернативной внутреннему критерию является непосредственная пользовательская оценка качества в интересующем нас приложении. Например, для оценки кластеризации результатов поиска можно измерить время, которое пользователь затрачивает на поиск ответа с помощью разных алгоритмов кластеризации. Это наиболее непосредственная оценка, но она требует больших затрат, особенно если нужны масштабные эксперименты с участием пользователей.

---

[3] Верхняя граница количества разбиений равна $K^N/K!$ Точное количество разных способов разбиения N документов на K кластеров равно числу Стирлинга второго рода (Stirling number). (См. `http://mathworld.wolfram.com/StirlingNumberoftheSecondKind.html` или Comtet (1974).)

В качестве замены пользовательских оценок можно использовать совокупность классов, образующих контрольный набор, или золотой стандарт (см. разделы 8.5 и 13.6). В идеальном случае золотой стандарт формулируется экспертами с хорошим уровнем взаимной согласованности (см. главу 8). Затем мы можем вычислить *внешний критерий* (external criterion), оценивающий, насколько кластеризация соответствует золотому стандарту. Например, оптимальная кластеризация результатов поиска по запросу jaguar, показанных на рис. 16.2, состоит из трех классов, соответствующих трем значениям этого слова: *car*, *animal* и *operating system*. При решении таких задач следует применять только идеальное разбиение на классы, а не метки классов.

В этом разделе введены четыре критерия качества кластеризации. *Чистота* (purity) — это простой для понимания и интуитивно понятный показатель. *Нормализованная взаимная информация* (normalized mutual information) хорошо интерпретируется с точки зрения теории информации. *Индекс Рэнда* (Rand Index) штрафует как ложно позитивные, так и ложно негативные решения в процессе кластеризации. *F-мера* (F-measure) дополнительно разрешает использовать разные веса для этих двух типов ошибок.

При вычислении *чистоты* сначала каждый кластер присваивается классу, с которым у кластера наибольшее перекрытие. После этого вычисляется правильность данного присваивания как количество верно присвоенных документов, деленное на общее число документов N. Соответствующая формула выглядит так.

$$purity(\Omega, \mathbb{C}) = \frac{1}{N}\sum_{k}\max_{j}\left(\omega_k \cap c_j\right) \tag{16.1}$$

Здесь $\Omega = \{\omega_1, \omega_2, ..., \omega_k\}$ — множество кластеров, $\mathbb{C}=\{c_1, c_2, ..., c_j\}$ — множество классов. Множество ω_k интерпретируется как совокупность документов, принадлежащих соответствующему кластеру, а c_j — как множество документов в соответствующем классе.

Пример вычисления чистоты приведен на рис. 16.4. [4] Чистота плохой кластеризации близка к нулю, а чистота идеальной кластеризации равна единице. Показатели качества сравниваются с тремя другими показателями в табл. 16.2.

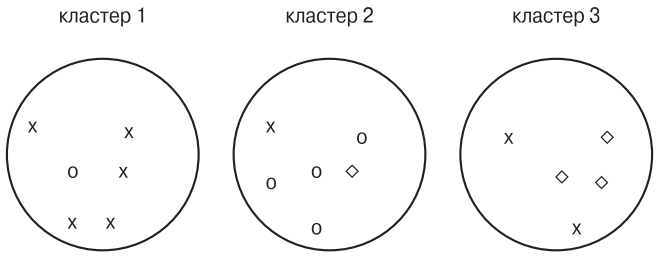


Рис. 16.4. Показатель качества как внешний критерий кластеризации. Основной класс и количество элементов основного класса в трех кластерах таковы: ×, 5 (кластер 1), o, 4 (кластер 2) и ◊, 3 (кластер 3). Показатель качества равен (1/17)×(5 + 4 + 3) ≈ 0,71

[4] Напомним о предупреждении, сделанном в примечании к рис. 14.2 и относящемся к этому и другим двумерным рисункам: эти иллюстрации могут ввести в заблуждение, поскольку двумерные проекции векторов, нормализованных по длине, искажают сходство и расстояния между точками.

Таблица 16.2. Четыре внешние оценки качества кластеризации, приведенной на рис. 16.4

	Качество	*NMI*	*RI*	F_5
Нижняя граница	0,0	0,0	0,0	0,0
Максимум	1,0	1,0	1,0	1,0
Значение на рис. 16.4	0,71	0,36	0,68	0,46

Если количество кластеров велико, то высокую чистоту обеспечить легко, в частности если каждый документ представляет собой отдельный кластер, то чистота равна единице. Следовательно, чистоту нельзя использовать для поиска баланса между качеством кластеризации и количеством кластеров.

Эту проблему можно решить с помощью *нормализованной взаимной информации* (NMI).

$$NMI(\Omega,\mathbb{C}) = \frac{I(\Omega;\mathbb{C})}{\left[H(\Omega)+H(\mathbb{C})\right]/2} \tag{16.2}$$

Здесь I — взаимная информация (см. главу 13).

$$I(\Omega;\mathbb{C}) = \sum_k \sum_j P\left(\omega_k \cap c_j\right) \log \frac{P\left(\omega_k \cap c_j\right)}{P(\omega_k)P\left(c_j\right)} = \tag{16.3}$$

$$= \sum_k \sum_j \frac{\left|\omega_k \cap c_j\right|}{N} \log \frac{N\left|\omega_k \cap c_j\right|}{\left|\omega_k\right|\left|c_j\right|}. \tag{16.4}$$

Здесь $P(\omega_k)$, $P(c_j)$ и $P(\omega_k \cap c_j)$ — вероятности того, что документ принадлежит кластеру ω_k, классу c_j и пересечению $\omega_k \cap c_j$ соответственно. Равенство (16.4) эквивалентно равенству (16.3) для оценки по максимуму правдоподобия (MLE) (т.е. оценкой каждой вероятности является соответствующая относительная частота).

H — это энтропия, определенная в главе 5.

$$H(\Omega) = -\sum_k P(\omega_k) \log P(\omega_k) = \tag{16.5}$$

$$= -\sum_k \frac{\left|\omega_k\right|}{N} \log \frac{\left|\omega_k\right|}{N}. \tag{16.6}$$

Здесь второе равенство основано на оценках по максимуму правдоподобия.

Показатель $I(\Omega;\mathbb{C})$ в равенстве (16.3) оценивает объем информации, на которую увеличивается знание о классе, когда мы узнаем, из каких документов состоят кластеры. Если классификация является случайной по отношению к принадлежности к классу, то $I(\Omega;\mathbb{C})$ принимает минимальное значение — нуль. В этом случае знание о том, что документ принадлежит конкретному кластеру, не дает никакой новой информации о самом классе. Максимальная взаимная информация достигается при кластеризации Ω_{exact}, которая точно воспроизводит классы, и если кластеры в разбиении Ω_{exact} продолжают подразделяться на более мелкие кластеры (упражнение 16.7). В частности, максимум взаимной информации достигается при кластеризации с $K = N$ кластерами, состоящими из одного документа. Таким образом, показатель MI порождает такую же проблему, как и чистота: он не накладывает штраф за большую мощность и, следовательно, не позволяет формализовать наше убеждение, что при прочих равных условиях, чем меньше кластеров, тем лучше.

Эту проблему можно решить путем нормализации с помощью знаменателя $[H(\Omega) + H(\mathbb{C})]/2$ в равенстве (16.2): при увеличении количества кластеров энтропия возрастает. Например, величина $H(\Omega)$ достигает максимума $\log N$ при $K = N$. Таким образом, при $K = N$ величина NMI будет небольшой. Поскольку показатель NMI нормализован, его можно использовать для сравнения кластеризации при разном количестве кластеров. Конкретный вид знаменателя был выбран потому, что величина $[H(\Omega) + H(\mathbb{C})]/2$ является точной верхней границей показателя $I(\Omega;\mathbb{C})$ (упражнение 16.8). Таким образом, показатель NMI всегда лежит в диапазоне от нуля до единицы.

Альтернативой этой теоретико-информационной интерпретации кластеризации является точка зрения на кластеризацию как на последовательность решений, по одному на каждую из $N(N-1)/2$ пар документов из коллекции. Два документа относятся к одному и тому же кластеру тогда и только тогда, когда они похожи. Истинно положительное решение (True-Positive — TP) относит два похожих документа к одному кластеру, истинно отрицательное решение (True-Negative — TN) относит два разных документа к разным кластерам. Возможны два типа ошибок. Ложно положительное решение (False-Positive — FP) относит два непохожих документа к одному и тому же кластеру. Ложно отрицательное решение (False-Negative — FN) относит два похожих документа к разным кластерам. *Коэффициент Рэнда* (Rand index) измеряет долю правильных решений. Иначе говоря, он представляет собой правильность (accuracy), рассмотренную в разделе 8.3.

$$RI = \frac{TP + TN}{TP + FP + FN + TN}$$

В качестве примера вычислим показатель RI по данным, представленным на рис. 16.4. Сначала вычислим сумму TP+FP. Три кластера содержат шесть, шесть и пять точек соответственно, поэтому общее количество положительных результатов, т.е. пар документов, принадлежащих одному и тому же кластеру, равно

$$TP + FP = C_6^2 + C_6^2 + C_5^2 = 40.$$

Среди них крестики в кластере 1, нулики в кластере 2, ромбики в кластере 3, а также крестики в кластере 3 являются истинно положительными результатами.

$$TP + FP = C_5^2 + C_4^2 + C_3^2 = 20$$

Следовательно, FP = 40 – 20 = 20.

Показатели FN и TN вычисляются аналогично. В результате получаем следующую факторную таблицу.

	Один и тот же кластер	*Разные кластеры*
Один и тот же класс	TP = 20	FN = 24
Разные классы	FP = 20	TN = 72

Таким образом, RI = (20 + 75)/(20 + 20 + 24 + 72) ≈ 0,68.

Показатель RI приписывает параметрам FP и FN одинаковые веса. Разделение похожих документов иногда хуже, чем включение непохожих документов в один и тот же кластер. Для того чтобы штрафовать за параметр FN строже, чем за FP, можно использовать *F-меру* (см. раздел 8.3), выбрав $\beta > 1$, тем самым придав полноте больший вес.

$$P = \frac{TP}{TP + FP},\ R = \frac{TP}{TP + FN},\ F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

Из данных, приведенных в факторной таблице, следует, что $P = 20/40 = 0{,}5$ и $R = 20/44 \approx 0{,}455$. Значит, $F_1 \approx 0{,}48$ при $\beta = 1$ и $F_5 \approx 0{,}456$ при $\beta = 5$. В области информационного поиска оценка кластеризации с помощью *F-меры* имеет преимущество, которое заключается в том, что эта мера уже известна исследователям.

? ***Упражнение 16.3.*** Замените каждую точку *d* на рис. 16.4 двумя идентичными копиями *d*, оставив их в том же классе.

1. Оцените, менее сложно, одинаково сложно или более сложно кластеризовать это множество, состоящее из 34 точек, по сравнению с множеством, состоящим из 17 точек?
2. Вычислите чистоту, NMI, RI и F_5 для кластеризации 34 точек. Какой показатель увеличивается, а какой остается таким же после удвоения количества точек?
3. На основании ответа к п. 1 и результатов выполнения п. 2 определите, какие показатели лучше подходят для сравнения качества двух разбиений?

16.4. Метод *K*-средних

Метод *K*-средних (K-means) является наиболее важным алгоритмом плоской кластеризации. Его цель — минимизировать среднеквадратичное евклидово расстояние (см. главу 6) между документами и центрами их кластеров, причем центр кластера ω определяется как *центроид* $\vec{\mu}$ его документов.

$$\vec{\mu}(\omega) = \frac{1}{|\omega|}\sum_{\vec{x}\in\omega}\vec{x}$$

В этом определении предполагается, что документы представлены в виде векторов, нормализованных по длине. Центроиды ранее использовались в алгоритме Роккио (см. главу 14). Здесь они играют аналогичную роль. Идеальным кластером в методе *K*-средних является сфера, центроид которой совпадает с ее центром тяжести. В идеальном случае кластеры не должны пересекаться. В методе Роккио требовалось то же самое. Разница между этими алгоритмами заключается в том, что при кластеризации нет размеченных документов, о которых известно, что они должны принадлежать одному и тому же кластеру.

Параметром, характеризующим, насколько хорошо центроиды представляют элементы своих кластеров, является *остаточная сумма квадратов* (Residual Sum of Squares — RSS), сумма квадратов расстояний от каждого из векторов до соответствующего центроида.

$$RSS_k = \sum_{\vec{x}\in\omega_k}\left|\vec{x} - \vec{\mu}(\omega_k)\right|^2,$$

$$RSS = \sum_{k=1}^{K} RSS_k \qquad (16.7)$$

Остаточная сумма квадратов RSS является целевой функцией в методе K-средних и должна быть минимизирована. Поскольку число *N* фиксировано, минимизация показате-

ля RSS эквивалентна минимизации среднеквадратического расстояния от центроидов до документов.

На первом шаге метода *K*-средних случайным образом выбираются *K* документов, играющих роль *начальных центров* кластеров, или *затравок* (seeds). После этого алгоритм перемещает центры кластеров по пространству, стремясь минимизировать показатель RSS. Как показано на рис. 16.5, это осуществляется путем итеративного повторения двух шагов, пока не будет выполнен критерий остановки: присваивание документов кластерам с ближайшими центроидами и вычисление центроидов на основе текущих элементов каждого кластера. На рис. 16.6 показаны результаты выполнения девяти итераций алгоритма *K*-средних для небольшого набора точек.

K-means($\{\vec{x}_1, ..., \vec{x}_N\}$, K)

1 $(\vec{s}_1, \vec{s}_2, ..., \vec{s}_K) \leftarrow$ SelectRandomSeeds($\{\vec{x}_1, ..., \vec{x}_N\}$, K)
2 **for** $k \leftarrow 1$ **to** K
3 **do** $\vec{\mu}_k \leftarrow \vec{s}_k$
4 **while** не выполняется критерий остановки
5 **do for** $k \leftarrow 1$ **to** K
6 **do** $\omega_k \leftarrow \{\}$
7 **for** $n \leftarrow 1$ **to** N
8 **do** $j \leftarrow \arg\min_{j'} |\vec{\mu}_{j'} - \vec{x}_n|$
9 $\omega_j \leftarrow \omega_j \cup \{\vec{x}_n\}$ (*присваивание векторов кластерам*)
10 **for** $k \leftarrow 1$ **to** K
11 **do** $\vec{\mu}_k \leftarrow \frac{1}{|\omega_k|} \sum_{\vec{x} \in \omega_k} \vec{x}$ (*вычисление центроидов кластеров*)
12 **return** $\{\vec{\mu}_1, ..., \vec{\mu}_K\}$

Рис. 16.5. Алгоритм K-средних. В большинстве приложений, связанных с информационным поиском, векторы $\vec{x}_n \in \mathbb{R}^M$ должны быть нормализованы по длине. Альтернативные методы выбора затравок обсуждаются ниже

В качестве условия остановки можно использовать одно из следующих.

- Выполнено фиксированное количество итераций *I*. Это условие ограничивает время выполнения алгоритма кластеризации, но в некоторых случаях из-за недостаточного количества итераций качество кластеризации может оказаться низким.
- Присваивание документов кластерам (функция разбиения γ) больше не изменяется. За исключением ситуации, когда алгоритм “застревает” на локальном минимуме, это условие позволяет получить хорошее разбиение, но время выполнения алгоритма может стать неприемлемо долгим.
- Центроиды $\vec{\mu}_k$ больше не изменяются. Это эквивалентно постоянству функции γ (упражнение 16.5).

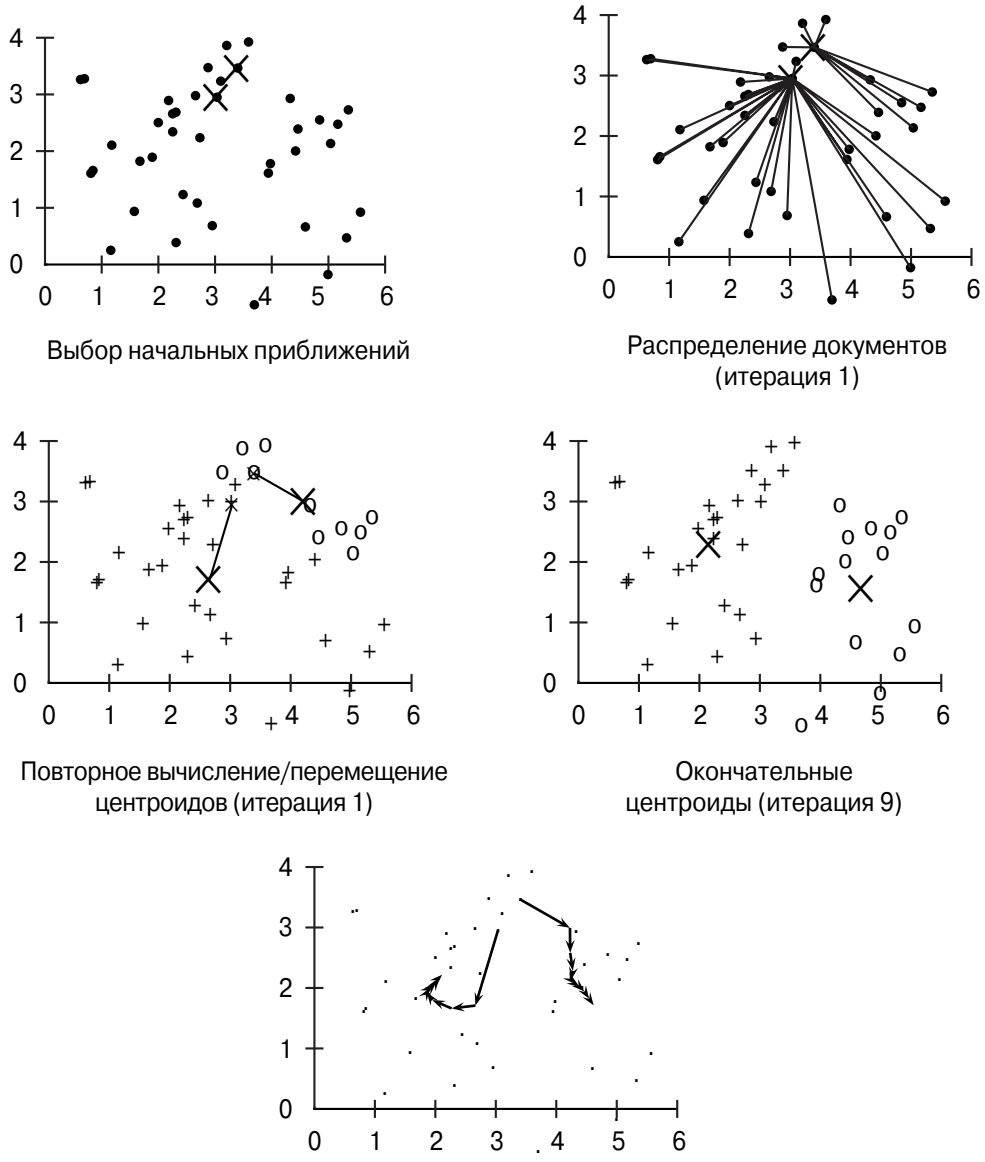


Рис. 16.6. Пример использования метода К-средних при К=2 в $\mathbb{R}^2$. Положение двух центроидов (указаны крупными крестиками) сходится после девяти итераций

- Значение RSS ниже порогового. Этот критерий гарантирует, что кластеризация после окончания работы алгоритма будет иметь требуемое качество. На практике его необходимо сочетать с ограничением количества итераций, чтобы гарантировать остановку работы.

- Изменение значения RSS ниже порогового θ. При малом значении θ это означает, что метод почти сошелся. Данное условие также следует сочетать с ограничением количества итераций, чтобы предотвратить чрезмерно долгую работу алгоритма.

Теперь покажем, что метод *K*-средних действительно сходится. Для этого докажем, что значение RSS на каждой итерации монотонно уменьшается. *Уменьшается* в данном случае означает *уменьшается* или *не изменяется*. Во-первых, значение RSS на шаге присваивания векторов кластерам уменьшается, поскольку каждый вектор присваивается кластеру с ближайшим центроидом, поэтому вклад, который они вносят в выражение для показателя RSS, уменьшается. Во-вторых, значение RSS уменьшается на шаге повторного вычисления центроидов, поскольку новый центроид — это вектор $\vec{v}$, на котором функция RSS_k достигает минимума.

$$RSS_k(\vec{v}) = \sum_{\vec{x}\in\omega_k} |\vec{v} - \vec{x}|^2 = \sum_{\vec{x}\in\omega_k} \sum_{m=1}^{M} (v_m - x_m)^2. \tag{16.8}$$

$$\frac{\partial RSS_k(\vec{v})}{\partial v_m} = \sum_{\vec{x}\in\omega_k} 2(v_m - x_m) \tag{16.9}$$

Здесь x_m и v_m — m-е компоненты соответствующих векторов. Приравнивая частную производную к нулю, получаем следующее уравнение.

$$v_m = \frac{1}{|\omega_k|} \sum_{\vec{x}\in\omega_k} x_m \tag{16.10}$$

Эта формула определяет компонентный состав центроида. Таким образом, когда старый центроид заменяется новым, функция RSS_k достигает минимума. RSS — сумма всех величин RSS_k — в ходе повторного вычисления также должна уменьшаться.

Поскольку количество возможных разбиений конечно, монотонно убывающая функция в конце концов достигнет (локального) минимума. Однако следует иметь в виду, что связи следует разрывать единообразно; например, если существует несколько центроидов, лежащих на одинаковом расстоянии от документа, то документ следует присваивать кластеру с наименьшим индексом. В противном случае алгоритм может зациклиться, ходя по кругу по разбиениям, имеющим одну и ту же ошибку.

Несмотря на то что это доказывает сходимость метода *K*-средних, к сожалению, нет никаких гарантий, что целевая функция достигнет *глобального минимума*. Эта проблема становится особенно острой, если множество документов содержит *выбросы* (outliers), т.е. документы, лежащие далеко от других документов и не относящиеся ни к одному кластеру. Если такой выброс взять в качестве начального центра кластера, то ему не будет присвоен ни один другой документ на всех последующих итерациях. Таким образом, возникает *одноэлементный кластер* (singleton cluster), даже если, возможно, существуют другие разбиения с меньшим значением RSS. Пример такого субоптимального разбиения, возникшего вследствие неудачного выбора начального приближения, показан на рис. 16.7.

Другой вид субоптимального разбиения, который возникает довольно часто, связан с пустыми кластерами (упражнение 16.11).

Эффективные эвристические правила выбора начального приближения формулируются так: 1) исключить из множества начальных центров выбросы, 2) проверить несколько начальных приближений и выбрать кластеризацию с наименьшей ошибкой, 3) получить начальные центры с помощью другого метода, например иерархической кластеризации. Поскольку детерминированный метод иерархической кластеризации

более предсказуем, чем метод К-средних, иерархическая кластеризация небольшой случайной выборки размером iK (например, $i = 5$ и $i = 10$) часто позволяет найти удачное начальное приближение.

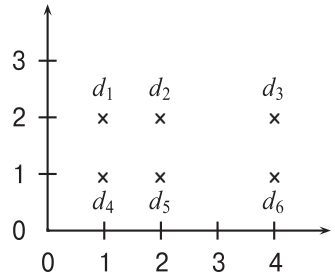


Рис. 16.7. Результат кластеризации по методу К-средних зависит от выбора начального приближения. При начальных центрах d_2 и d_5 алгоритм К-средних сходится к субоптимальному разбиению $\{\{d_1, d_2, d_3\}, \{d_4, d_5, d_6\}\}$. При начальных центрах d_2 и d_3 алгоритм К-средних сходится к глобальному оптимуму $\{\{d_1, d_2, d_4, d_5\}, \{d_3, d_6\}\}$ для $K = 2$

Другие методы инициализации вычисляют начальные центры, не выбирая их из кластеризуемых векторов. Например, устойчивый метод, хорошо работающий с широким диапазоном распределений документов, заключается в том, чтобы для каждого кластера выбрать i случайных векторов (например, $i = 10$) и использовать их центроид в качестве его начального приближения. Более сложные методы инициализации рассматриваются в разделе 16.6.

Какова временная сложность метода *К*-средних? Большая часть времени затрачивается на вычисление расстояний между векторами. Временная сложность одной такой операции составляет $\Theta(M)$. На шаге присваивания вычисляются KN расстояний, поэтому общая временная сложность составляет $\Theta(KNM)$. На шаге вычисления центроидов каждый вектор добавляется к сумме векторов один раз, поэтому сложность этого шага составляет $\Theta(NM)$. Следовательно, при фиксированном количестве итераций I общая сложность алгоритма составляет $\Theta(IKNM)$. Таким образом, алгоритм *К*-средних является линейным по всем задействованным факторам: по количеству итераций, по количеству кластеров, по количеству векторов и по размерности пространства. Это значит, что метод *К*-средних более эффективен, чем иерархические алгоритмы, описанные в главе 17. Однако зафиксировать количество итераций на практике довольно сложно. Тем не менее в большинстве случаев метод *К*-средних быстро либо полностью сходится, либо почти сходится. В последнем случае некоторые документы на дальнейших итерациях могли бы изменить свою принадлежность кластерам, но это мало влияет на общее качество кластеризации.

В изложенных выше аргументах есть один нюанс. Даже линейный алгоритм может работать медленно, если один из аргументов в оценке $\Theta(...)$ велик, а число M, как правило, велико. Высокая размерность не является проблемой при вычислении расстояния между двумя документами. Их векторы являются разреженными, поэтому только малая часть из теоретически возможных покомпонентных разностей участвует в вычислении. Однако центроиды образуют плотные векторы. Они учитывают все термины, появляю-

щиеся хотя бы в одном из документов кластера. В результате при наивной реализации алгоритма *K*-средних расстояния вычисляются медленно. Однако существуют простые и эффективные эвристические приемы, позволяющие оценивать сходство между центроидом и документом так же быстро, как и между документами. Усечение центроидов до наиболее значимых *k* терминов (например, $k = 1000$) редко ухудшает качество кластеров, но существенно ускоряет выполнение шага присваивания документов кластерам (см. ссылки в разделе 16.6).

Проблему производительности можно решить с помощью алгоритма *K-медоидов* (K-medoids), представляющего собой вариант алгоритма *K*-средних, в котором вместо центроидов вычисляются медоиды кластеров. *Медоидом* (medoid) кластера называется вектор документа, ближайший к центроиду. Поскольку медоиды являются разреженными векторами документов, вычисление расстояния между ними и документами осуществляется быстро.

16.4.1. Количество кластеров в методе K-средних

В разделе 16.2 указано, что количество кластеров *K* в большинстве алгоритмов плоской кластеризации задается заранее. А что делать, если у нас нет разумной догадки об этом числе?

В качестве наивного подхода можно было бы выбрать оптимальное значение *K* в соответствии с целевой функцией, т.е. выбрать число *K*, минимизирующее величину RSS. Определив величину $\mathrm{RSS}_{\min}(K)$ как минимальное значение RSS на всех разбиениях с *K* кластерами, мы обнаружим, что функция $\mathrm{RSS}_{\min}(K)$ монотонно убывает при возрастании величины *K* (упражнение 16.13) и достигает минимума, равного нулю, при $K = N$, где N — количество документов. В итоге каждый документ окажется в собственном кластере. Очевидно, что такая кластеризация не является оптимальной.

Эвристический метод, решающий эту проблему, заключается в вычислении величины $\mathrm{RSS}_{\min}(K)$ следующим образом. Сначала выполним *i* (например, $i = 10$) разбиений на *K* кластеров (каждую при другом начальном приближении) и вычислим величину RSS для каждой из них. Затем найдем минимум среди *i* значений RSS. Обозначим этот минимум как $\widehat{RSS}_{\min}(K)$. Теперь можно проверить, как изменяется величина $\widehat{RSS}_{\min}(K)$ при увеличении *K*, и найти точку перегиба кривой — точку, после которой дальнейшее уменьшение величины $\widehat{RSS}_{\min}(K)$ становится заметно меньше. На рис. 16.8 есть две такие точки: одна, $K = 4$, в которой угол наклона незначительно уменьшается, и вторая, $K = 9$, в которой угол наклона изменяется заметнее. Это типичная ситуация: редко существует единственная наилучшая оценка количества кластеров. Следовательно, необходимо установить дополнительное ограничение, чтобы выбрать одно из нескольких возможных значений *K* (в данном случае — четыре и девять).

Вторая разновидность критериев для выбора количества кластеров предусматривает введение штрафа за каждый новый кластер. В этом случае мы начинаем с одного кластера, содержащего все документы, а затем ищем оптимальное количество кластеров, постепенно увеличивая число *K*. Для того чтобы определить количество кластеров таким образом, целевая функция обобщается так, чтобы учесть два элемента: *искажение* (distortion) — показатель, характеризующий отклонение документов от прототипов из своих кластеров (например, RSS в методе *K*-средних), и меру *сложности модели* (model complexity). В данном случае мы рассматриваем кластеризацию (разбиение) как модель

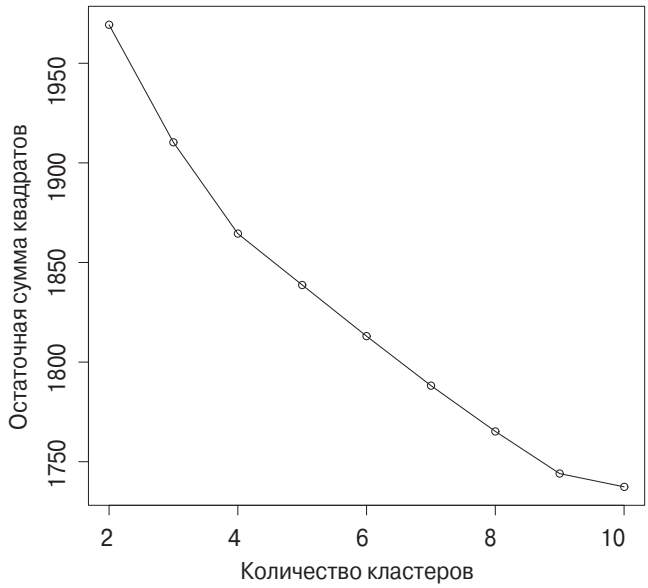


Рис. 16.8. Оценка минимальных остаточных сумм квадратов ($\widehat{RSS}_{\min}(K)$) как функция количества кластеров в методе K-средних. В этой кластеризации 1203 документов из коллекции Reuters-RCV1 существуют две точки, после которых кривая $\widehat{RSS}_{\min}(K)$ изменяет наклон: четыре и девять кластеров. Документы были выбраны из категорий China, Germany, Russia *и* Sports, *поэтому значение K = 4 точнее всего соответствует классификации коллекции Reuters-RCV1*

данных. Сложность модели обычно выражается как количество кластеров или функция, зависящая от этого показателя. Для метода *K*-средних получается следующий критерий выбора параметра *K*.

$$K = \arg\min_{K}\left[RSS_{\min}(K) + \lambda K\right] \tag{16.11}$$

Здесь λ — весовой фактор. Большое значение λ отдает предпочтение решениям с небольшим количеством кластеров. При λ = 0 штрафа за дополнительные кластеры нет, и наилучшим решением является выбор $K = N$.

Очевидная трудность, связанная с уравнением (16.11), заключается в необходимости определить параметр λ. Если его определить не легче, чем параметр *K*, то сложность возводится в квадрат. В некоторых случаях можно выбрать те значения λ, которые хорошо работали на аналогичных данных в прошлом. Например, если мы периодически проводим кластеризацию поступающих новостей, то, вероятно, зафиксируем параметр λ, который в каждой следующей кластеризации позволит найти правильное значение *K*. В этом приложении мы не сможем найти параметр *K*, основываясь на прошлом опыте, так как он изменяется.

Теоретическим обоснованием уравнения (16.11) является *информационный критерий Акаике* (Akaike Information Criterion — AIC). Он позволяет найти баланс между искаже-

нием и сложностью модели, исходя из положений теории информации. Общая форма информационного критерия Акаике выглядит следующим образом.

$$\mathbf{AIC}: K = \arg\min_{K}\left[-2L(K) + 2q(K)\right] \tag{16.12}$$

Здесь $-L(K)$ — отрицательный максимум логарифмической функции правдоподобия данных для K кластеров, характеризующий искажение, а $q(K)$ — количество параметров модели с K кластерами, характеризующее сложность модели. Мы не будем здесь выводить информационный критерий Акаике, но его легко понять интуитивно. Первое свойство хорошей модели данных заключается в том, что каждая точка хорошо приближается данной моделью. В этом заключается цель малых искажений. Однако модель также должна быть небольшой (т.е. иметь низкую сложность). Модель, просто описывающая данные (и, следовательно, имеющая нулевое искажение), бесполезна. Информационный критерий Акаике обеспечивает теоретическую основу для одно из методов взвешивания этих двух факторов — искажения и сложности модели — при выборе модели.

Информационный критерий Акаике для метода K-средних можно сформулировать следующим образом.

$$\mathbf{AIC}: K = \arg\min_{K}\left[RSS_{\min}(K) + 2MK\right] \tag{16.13}$$

Равенство (16.13) является частным случаем равенства (16.11) при $\lambda = 2M$.

Для вывода равенства (16.13) из равенства (16.12) заметим, что в методе K-средних $q(K) = KM$, поскольку каждый элемент K центроидов является параметром, который может изменяться независимо. Кроме того, если моделью метода K-средних является смесь нормальных распределений с жесткой кластеризацией, кластерами, имеющими одинаковые веса, и идентичными сферическими матрицами ковариации, то $L(K) = -(1/2)\mathrm{RSS}_{\min}(K)$ (по модулю константы) (упражнение 16.19).

Вывод информационного критерия Акаике основан на нескольких предположениях, например на предположении о том, что данные являются независимыми и одинаково распределенными случайными величинами. В практике информационного поиска эти предположения выполняются лишь приближенно. Вследствие этого критерий AIC редко применяется для кластеризации текстов без модификации. На рис. 16.8 размерность векторного пространства M равна приблизительно 50 000. Следовательно, слагаемое $2MK > 50\,000$ превышает слагаемое $\widehat{RSS}_{\min}(1) < 5000$ (не показано на рисунке), а минимум выражения достигается при $K = 1$. Однако, как мы знаем, выбор $K = 4$ лучше соответствует классам *China*, *Germany*, *Russia* и *Sports*, чем $K = 1$. На практике равенство (16.11) часто полезнее равенства (16.13), с оговоркой, касающейся оценки параметра λ.

Упражнение 16.4. Почему документы, не использующие один и тот же термин для обозначения понятия *car*, скорее всего, окажутся в одном и том же кластере, если кластеризация проводится по методу K-средних?

Упражнение 16.5. Остановка алгоритма K-средних произойдет, если 1) разбиение не изменилось, 2) центроиды не изменились. Вытекают ли эти условия одно из другого?

16.5. Кластеризация, основанная на моделях

В этом разделе рассматривается обобщение метода K-средних — EM-алгоритм. Его можно применять к более широкому диапазону представлений и распределений документов.

В методе *K*-средних мы пытаемся найти репрезентативные центроиды. Множество *K* центроидов можно рассматривать как модель, генерирующую данные. Порождение документов в этой модели сводится к выбору случайного центроида и добавлению шума. Если шум распределен нормально, то эта процедура создаст кластеры сферической формы. *Кластеризация, основанная на моделях* (model-based clustering), как бы предполагает, что данные были порождены моделью, и старается восстановить оригинальную модель по этим данным. Затем модель, которая восстанавливается на основе этих данных, определяет кластеры и присваивает документы этим кластерам.

Как правило, для оценки параметров модели используется метод максимума правдоподобия. В методе *K*-средних величина exp(-RSS) пропорциональна правдоподобию того, что именно эта конкретная модель (т.е. совокупность центроидов) породила данные. В методе *K*-средних максимум правдоподобия и минимум ошибки RSS являются эквивалентными критериями. Обозначим параметры модели символом Θ. В методе *K*-средних $\Theta = \{\vec{\mu}_1,...,\vec{\mu}_K\}$.

Критерий максимума правдоподобия заключается в выборе параметров Θ, максимизирующих логарифмическую функцию правдоподобия порождения данных *D*.

$$\Theta = \arg\max_{\Theta} L(D|\Theta) = \arg\max_{\Theta} \log \prod_{n=1}^{N} P(d_n|\Theta) = \arg\max_{\Theta} \log \sum_{n=1}^{N} \log P(d_n|\Theta)$$

Здесь $L(D|\Theta)$ — целевая функция, характеризующая качество кластеризации. Из двух разбиений на одинаковое количество кластеров следует предпочесть разбиение с бо́льшим значением $L(D|\Theta)$.

Этот же подход был принят в главе 12 для построения языковых моделей, а также в разделе 13.1 для классификации текстов. Решая задачу классификации текстов, мы выбирали класс, обеспечивающий максимум правдоподобия порождения конкретного документа. Теперь мы выбираем кластеризацию Θ, обеспечивающую максимум правдоподобия порождения заданного набора документов. Имея параметры Θ, мы можем вычислить вероятность присваивания $P(d|\omega_k; \Theta)$ для каждой пары "документ–кластер". Эти вероятности присваивания определяют мягкую кластеризацию.

Рассмотрим пример такого мягкого присваивания. Документы о китайских автомобилях могут иметь частичную принадлежность, равную 0,5, к каждому из двух кластеров: *China* и *automobiles*. Это отражает тот факт, что обе темы являются уместными. Жесткая кластеризация, как в методе *K*-средних, не может моделировать одновременную релевантность двух разных тем.

Кластеризация, основанная на моделях, позволяет использовать знания о предметной области. Метод *K*-средних и алгоритмы иерархической кластеризации, описанные в главе 17, делают достаточно негибкие предположения о данных. Например, в методе K-средних предполагается, что кластеры имеют сферическую форму. Кластеризация на основе моделирования обеспечивает бо́льшую гибкость. Модель кластеризации может адаптироваться к тому, что мы знаем о лежащем в основе распределении данных, будь то распределение Бернулли (как в примере из табл. 16.3) или нормальное распределение с несферической дисперсией (другая важная модель кластеризации документов), или к чему-нибудь другому.

Таблица 16.3. EM-алгоритм кластеризации. В таблице приведены а) совокупность документов и б) значения параметров для отдельных итераций в ходе кластеризации с помощью EM-алгоритма. Приведены параметры: априорная вероятность α_1, значения функции мягкого присваивания $r_{n,1}$ (только для первого кластера) и лексические параметры $q_{m,k}$ для нескольких терминов. Авторы изначально присвоили документ 6 кластеру 1, а документ 7 — кластеру 2 (итерация 0). EM-алгоритм сходится после 25 итераций. Для сглаживания параметры $r_{n,k}$ в уравнении (16.16) заменены числами $r_{n,k} + \varepsilon$, где $\varepsilon = 0{,}0001$

а)

DocID	*Текст документа*	*DocID*	*Текст документа*
1	hot chocolate cocoa beans	7	sweet sugar
2	cocoa ghana africa	8	sugar cane brazil
3	beans harvest ghana	9	sweet sugar brazil
4	cocoa butter	10	sweet cake icing
5	butter truffles	11	cake black forest
6	sweet chocolate		

б)

	Итерация							
Параметр	0	1	2	3	4	5	6	7
α_1		0,50	0,45	0,53	0,57	0,58	0,54	0,45
$r_{1,1}$		1,00	1,00	1,00	1,00	1,00	1,00	1,00
$r_{2,1}$		0,50	0,79	0,99	1,00	1,00	1,00	1,00
$r_{3,1}$		0,50	0,84	1,00	1,00	1,00	1,00	1,00
$r_{4,1}$		0,50	0,75	0,94	1,00	1,00	1,00	1,00
$r_{5,1}$		0,50	0,52	0,66	0,91	1,00	1,00	1,00
$r_{6,1}$	1,00	1,00	1,00	1,00	1,00	1,00	0,83	0,00
$r_{7,1}$	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
$r_{8,1}$		0,00	0,00	0,00	0,00	0,00	0,00	0,00
$r_{9,1}$		0,00	0,00	0,00	0,00	0,00	0,00	0,00
$r_{10,1}$		0,50	0,40	0,14	0,01	0,00	0,00	0,00
$r_{11,1}$		0,50	0,57	0,58	0,41	0,07	0,00	0,00
$q_{\text{africa},1}$		0,000	0,100	0,134	0,156	0,158	0,169	0,200
$q_{\text{africa},2}$		0,000	0,083	0,042	0,001	0,000	0,000	0,000
$q_{\text{brazil},1}$		0,000	0,000	0,000	0,000	0,000	0,000	0,000
$q_{\text{brazil},2}$		0,000	0,167	0,195	0,213	0,214	0,196	0,167
$q_{\text{cocoa},1}$		0,000	0,400	0,432	0,465	0,474	0,508	0,600
$q_{\text{cocoa},2}$		0,000	0,167	0,090	0,014	0,001	0,000	0,000

Окончание табл. 16.3

	Итерация							
Параметр	0	1	2	3	4	5	6	7
$q_{sugar,1}$		0,000	0,000	0,000	0,000	0,000	0,000	0,000
$q_{sugar,2}$		1,000	0,500	0,585	0,640	0,642	0,589	0,500
$q_{sweet,1}$		1,000	0,300	0,238	0,180	0,159	0,153	0,000
$q_{sweet,2}$		1,000	0,417	0,507	0,610	0,640	0,608	0,667

Наиболее широко используемым алгоритмом кластеризации на основе моделирования является *EM-алгоритм*. EM-кластеризация — это итеративный процесс поиска максимума функции $L(D|\Theta)$. EM-алгоритм можно применять для решения разнообразных задач вероятностного моделирования. В данном случае мы будем работать со смесью многомерных распределений Бернулли (см. разделы 11.3 и 13.3).

$$P(d|\omega_k;\Theta)=\left(\prod_{t_m\in d} q_{mk}\right)\left(\prod_{t_m\notin d}(1-q_{mk})\right) \tag{16.14}$$

Здесь $\Theta = \{\Theta_1, ..., \Theta_K\}$, $\Theta_k = (\alpha_k, q_{1k}, ..., q_{Mk})$ и $q_{mk} = P(U_m=1|\omega_k)$ — параметры модели.[5] $P(U_m=1|\omega_k)$ — вероятность того, что документ из кластера ω_k содержит термин t_m. Вероятность α_k — априорная вероятность кластера ω_k, т.е. вероятность того, что документ d принадлежит кластеру ω_k, если у нас нет никакой информации о документе d.

Смешанная модель выглядит следующим образом.

$$P(d|\Theta)=\sum_{k=1}^{K}\alpha_k\left(\prod_{t_m\in d} q_{mk}\right)\left(\prod_{t_m\notin d}(1-q_{mk})\right) \tag{16.15}$$

Порождение документа с помощью этой модели осуществляется следующим образом. Сначала с вероятностью α_k выбирается кластер ω_k, а затем в соответствии с параметрами q_{mk} генерируются термины документа. Напомним, что представление документа в многомерной модели Бернулли является вектором, состоящим из M булевых значений (а не действительных чисел).

Как с помощью EM-алгоритма определить параметры кластеризации по данным? Иначе говоря, как выбрать параметры Θ, при которых функция $L(D|\Theta)$ достигает максимума? EM-алгоритм похож на метод K-средних, поскольку также состоит из *E-шага* (expectation step), соответствующего шагу присваивания документов кластерам, и *M-шага* (maximization step), соответствующего повторному вычислению параметров модели. Параметрами в методе K-средних являются центроиды, а параметрами в EM-алгоритме — α_k и q_{mk}.

На M-шаге повторно вычисляются условные параметры q_{mk} и априорные вероятности α_k.

$$\textit{Этап максимизации}: q_{mk}=\frac{\sum_{n=1}^{N} r_{nk} I(t_m\in d_n)}{\sum_{n=1}^{N} r_{nk}},\ \alpha_k=\frac{\sum_{n=1}^{N} r_{nk}}{N}. \tag{16.16}$$

[5] U_m — случайная переменная, определенная в разделе 13.3 для бернуллиевской наивной байесовской модели. Она равна единице, если термин t_m встречается в документе, или нулю, если термин t_m в документе не встречается.

Здесь $I(t_m \in d_n)=1$, если $t_m \in d_n$, и $I(t_m \in d_n)=0$, если $t_m \notin d_n$, а r_{nk} — параметр мягкого присваивания документа d_n кластеру ω_k, вычисленный на предыдущей итерации. Эти параметры являются оценками по максимуму правдоподобия параметров многомерного распределения Бернулли (см. табл. 13.3), за исключением того, что документы здесь присвоены кластерам частично. Эти оценки по максимуму правдоподобия обеспечивают максимум правдоподобия данных в рамках указанной модели.

На E-шаге происходит мягкое присваивание документов кластерам при текущих параметрах α_k и q_{mk}, т.е. вычисление параметров r_{nk}.

$$\textit{Этап ожидания}: \quad r_{nk} = \frac{\alpha_k \left(\prod_{t_m \in d_n} q_{mk} \right) \left(\prod_{t_m \notin d_n} (1 - q_{mk}) \right)}{\sum_{k=1}^{K} \alpha_k \left(\prod_{t_m \in d_n} q_{mk} \right) \left(\prod_{t_m \notin d_n} (1 - q_{mk}) \right)}. \tag{16.17}$$

На E-шаге с помощью формул (16.14) и (16.15) вычисляется правдоподобие того, что кластер ω_k генерирует документ d_n. Эта процедура классификации в рамках многомерной модели Бернулли описана в табл. 13.3. Таким образом, E-шаг — это не что иное, как классификация с помощью бернуллиевского наивного байесовского подхода (включая нормализующий знаменатель для перевода функции классификации в вид распределения вероятностей по кластерам).

В табл. 16.3 одиннадцать документов кластеризованы на два кластера с помощью EM-алгоритма. Алгоритм сходится за 25 итераций. В итоге первые пять документов присвоены кластеру 1 ($r_{i,1} = 1{,}00$), а последние шесть — кластеру 2 ($r_{i,1} = 0{,}00$). Несколько неожиданно последнее присваивание оказалось жестким. EM-алгоритм обычно сходится к мягкому присваиванию. На итерации 25 априорная вероятность α_1 кластера 1 равна $5/11 \approx 0{,}45$, поскольку в первом кластере оказались пять из одиннадцати документов. Некоторые термины сразу связываются с одним кластером, так как начальное распределение может распространиться на них однозначным образом. Например, принадлежность кластеру 2 уже на первой итерации распространяется от седьмого документа к восьмому, поскольку оба эти документа содержат термин sugar (на первой итерации $r_{8,1} = 0$). Для параметров терминов, появляющихся в неоднозначных контекстах, требуется больше итераций. Например, шестой и седьмой документы содержат термин sweet. Для того чтобы этот термин был однозначно связан с кластером 2, потребовалось 25 итераций ($q_{\text{sweet},1} = 0$ на итерации 25).

Поиск хорошего начального приближения для EM-алгоритма еще важнее, чем для метода *K*-средних. EM-алгоритм имеет тенденцию попадать в локальный оптимум при неудачном выборе затравочных документов. Эта проблема характерна и для других приложений EM-алгоритма.[6] По этой причине, как и в методе K-средних, начальное присваивание документов кластерам часто вычисляется с помощью другого алгоритма. Например, начальное приближение можно получить с помощью жесткой кластеризации по методу *K*-средних, а затем “смягчить”, применив EM-алгоритм.

? ***Упражнение 16.6.*** Выше было показано, что временная сложность алгоритма *K*-средних составляет $\Theta(IKNM)$. Оцените временную сложность EM-алгоритма.

[6] Например, эта проблема возникает, когда EM-алгоритм используется для оценки параметров скрытых марковских моделей, вероятностных грамматик и моделей машинного перевода с одного естественного языка на другой (Manning and Schütze, 1999).

Упражнение 16.7. Пусть Ω — разбиение на кластеры, точно воспроизводящее структуру классов $\mathbb{C}$, а Ω' — разбиение на кластеры, в котором некоторые кластеры разбиения Ω разделены на части. Покажите, что $I(\Omega;\mathbb{C}) = I(\Omega';\mathbb{C})$.

Упражнение 16.8. Покажите, что $I(\Omega;\mathbb{C}) \leq [H(\Omega) + H(\mathbb{C})]/2$.

Упражнение 16.9. Взаимная информация является симметричной в том смысле, что ее значение не изменяется при перестановке кластеров и классов: $I(\Omega;\mathbb{C}) = I(\mathbb{C};\Omega)$. Какой еще из трех показателей качества кластеризации является симметричным?

Упражнение 16.10. Вычислите показатель RSS для двух разбиений, показанных на рис. 16.7.

Упражнение 16.11. 1) Приведите пример совокупности точек и трех начальных центроидов (не являющихся элементами этой совокупности), для которых метод 3-средних сходится к разбиению с пустым кластером. 2) Может ли разбиение с пустым кластером обеспечить глобальный оптимум по отношению к показателю RSS?

Упражнение 16.12. Загрузите коллекцию Reuters-21578. Отбросьте документы, не появляющиеся ни в одном из следующих десяти классов: *acquisitions*, *corn*, *crude*, *earn*, *grain*, *interest*, *mones-fx*, *ship*, *trade* и *wheat*. Отбросьте документы, появляющиеся в двух из этих классов. 1) Вычислите разбиение этого множества на десять кластеров по методу *K*-средних. Метод *K*-средних реализован во многих пакетах программ, например WEKA (Witten and Frank, 2005) и R (R Development Core Team, 2005). 2) Вычислите показатель качества, нормализованную взаимную информацию, меру F_1 и коэффициент Рэнда для данного разбиения на десять кластеров. 3) Вычислите матрицу неточностей (см. табл. 14.5) для десяти классов и десяти кластеров. Идентифицируйте классы, увеличивающие показатели FP и FN.

Упражнение 16.13. Докажите, что функция $\mathrm{RSS}_{\min}(K)$ монотонно убывает при увеличении параметра K.

Упражнение 16.14. Существует мягкий вариант метода *K*-средних, в котором вычисляется частичная принадлежность документа к кластеру, представляющая собой монотонно убывающую функцию, зависящую от расстояния Δ от центроида, например $e^{-\Delta}$. Модифицируйте шаги присваивания документов кластерам и вычисления центроидов кластеров жесткого метода *K*-средних в сторону мягкого варианта.

Упражнение 16.15. На последней итерации в табл. 16.3 шестой документ принадлежит второму кластеру, несмотря на то что он был начальным приближением для первого кластера. Почему этот документ оказался в другом кластере?

Упражнение 16.16. Значения параметров q_{mk} на итерации 25 в табл. 16.3 округлены. Укажите точные значения, к которым сходится EM-алгоритм.

Упражнение 16.17. Выполните кластеризацию по методу *K*-средних для документов, перечисленных в табл. 16.3. Сколько итераций потребуется, чтобы метод K-средних сошелся? Сравните этот результат с кластеризацией по EM-алгоритму в табл. 16.3 и укажите различия между ними.

Упражнение 16.18 **[***].** Модифицируйте E- и M-шаги EM-алгоритма для смеси нормальных распределений. На M-шаге вычисляются оценки по максимуму правдоподобия α_k, $\vec{\mu}_k$ и Σ_k для каждого кластера. На E-шаге для каждого вектора на

основании текущих параметров вычисляется мягкое присваивание кластерам. Запишите формулы для смеси нормальных распределений, соответствующие формулам (16.16) и (16.17).

***Упражнение 16.19* [***].** Покажите, что метод *K*-средних можно интерпретировать как предельный вариант EM-алгоритма для смеси нормальных распределений, если дисперсия очень мала и все коэффициенты ковариации равны нулю.

***Упражнение 16.20* [***].** *Внутрикластерный разброс* (within-point scatter) определяется по формуле $\sum_k \frac{1}{2} \sum_{\vec{x}_i \in \omega_k} \sum_{\vec{x}_j \in \omega_k} |\vec{x}_i - \vec{x}_j|^2$. Покажите, что минимизация ошибки RSS и минимизация внутрикластерного разброса эквивалентны.

***Упражнение 16.21* [***].** Выведите информационный критерий Акаике для смешанной модели Бернулли из формулы (16.12).

16.6. Библиография и рекомендации для дальнейшего чтения

Обширный обзор методов кластеризации с акцентом на масштабируемость опубликован в работе Берхина (Berkhin, 2006*b*). Классической монографией по кластеризации в теории распознавания образов, в которой описаны и метод *K*-средних, и EM-алгоритм, является книга Дуда и др. (Duda et al., 2000). Расмуссен (Rasmussen, 1992) изложил основы кластеризации с точки зрения информационного поиска. Андерберг (Anderberg, 1973) описал общие приложения кластеризации. Кроме евклидова расстояния и косинусной меры сходства, в качестве меры сходства между документами и кластерами используется расстояние Кульбака–Лейблера (Xu and Croft, 1999; Muresan and Harper, 2004; Kurland and Lee, 2004).

Кластерная гипотеза выдвинута Жарденом и ван Рийсбергеном (Jardine and van Rijsbergen, 1971), которые сформулировали ее следующим образом: *ассоциации между документами несут информацию о релевантности документов запросам*. Качество и производительность поиска на основе кластеризации исследовались в многочисленных работах (Salton, 1971*a*, 1975; Croft, 1978; Voorhees, 1985*a*; Can and Ozkarahan, 1990; Cacheda et al., 2003; Can et al., 2004; Singitham et al., 2004; Altingцvde et al., 2008). Несмотря на то что некоторые из этих исследований продемонстрировали повышение качества, производительности, а также и того, и другого, консенсуса по этим вопросам до сих пор нет. Языковые модели на основе кластеризации были впервые предложены Ли и Крофтом (Liu and Croft, 2004).

Существуют убедительные доказательства того, что кластеризация результатов поиска обогащает опыт пользователей и повышает качество результатов (Hearst and Pedersen, 1996; Zamir and Etzioni, 1999; Tombros et al., 2002; Käki, 2005; Toda and Kataoka, 2005), правда, не настолько сильно, как методы структуризации результатов поиска с помощью тщательно составленных иерархий категорий (Hearst, 2006). Интерфейс, работающий по принципу разбиения и объединения (scatter-gather), для просмотра коллекций разработан Каттингом и др. (Cutting et al., 1992). Теоретические основы для анализа свойств этого и других интерфейсов изложены Пиролли (Pirolli, 2007). Метод LSI (глава 18) и усеченное представление центроидов для повышения производительности кластеризации по методу *K*-средних описаны в работе Шютце и Сильверштайна (Schütze and Silverstein, 1997).

Иерархическая кластеризация (глава 17), позволившая ввести два уровня детализации новостей, использована в системе Columbia NewsBlaster (McKeown et al., 2002), предшественнице знаменитой и более совершенной системы Google News (`http://news.google.com`). Детали изложены в работах Хацивассилиоглу и др. (Hatzivassilioglu et al., 2000), Чена и Лина (Chen and Lin, 2000), а также Радева и др. (Radev et al., 2001). Другими приложениями кластеризации в области информационного поиска являются нахождение дубликатов (Yang and Callan, 2006), обнаружение новизны документов (см. раздел 17.9), а также выявление метаданных в семантическом вебе (semantic web) (Alonso et al., 2006).

Обсуждение внешней оценки качества частично основано на работах Штреля (Strehl, 2002). Дом (Dom, 2002) предположил, что показатель Q_0 лучше обоснован с теоретической точки зрения, чем показатель NMI. Показатель Q_0 — это количество битов, необходимое для передачи информации о принадлежности документа классу при условии, что принадлежность документов кластерам известна. Коэффициент Рэнда предложен Рэндом (Rand, 1971). Хубурт и Араби (Hubert and Arabie, 1985) предложили *скорректированный коэффициент Рэнда* (adjusted Rand index), лежащий в диапазоне от –1 до 1 и принимающий значение 0, если между кластерами и классами существует лишь случайное согласование (этот показатель похож на показатель κ из главы 8). Басу и др. (Basu et al., 2004) утверждают, что все три показателя — NMI, индекс Рэнда и F-мера — приводят к очень похожим результатам. Штейн и др. (Stein et al., 2003) предложили использовать в качестве внутреннего показателя качества кластеризации *ожидаемую плотность ребер* (expected edge density). Аксиоматические основы для сравнения кластеризаций изложены в работах Клейнберга (Kleinberg, 2002) и Мейла (Meilă, 2005).

Изобретение метода *K*-средних часто приписывается нескольким авторам — Ллойду (Lloyd, 1982)) (работа впервые опубликована в 1957 году), Боллу (Ball, 1965), Маккуину (MacQueen, 1967), Хартигану и Вонгу (Hartigan and Wong, 1979). Сложность метода *K*-средних в худшем случае исследована в работе Артура и Вассилвитски (Arthur and Vassilvitskii, 1998). Эмпирическое исследование сходимости метода *K*-средних и ее зависимость от выбора начального приближения провели Брэдли и Файад (Bradley and Fayyad, 1998), Пеллег и Мур (Pelleg and Moore, 1999) и Дэвидсон и Сатьянарараяна (Davidson and Satyanarayana, 2003). Диллон и Модха (Dhillon and Modha, 2001) сравнили кластеры, полученные с помощью метода *K*-средних, с кластерами, полученными с помощью метода SVD (Singular Value Decomposition), описанного в главе 18. Алгоритм *K*-медоидов предложен Кауфманом и Руссо (Kaufman and Rousseeuw, 1990). EM-алгоритм разработан Демпстером (Dempster et al., 1977). Глубокое исследование EM-алгоритма проведено в работе Маклахлена и Кришнана (McLachlan and Krishnan, 1996). Публикации о латентном анализе, который можно рассматривать как разновидность мягкой кластеризации, перечислены в разделе 18.5.

Информационный критерий Акаике (Akaike Information Criterion — AIC) сформулировал Хироцугу Акаике (Akaike, 1974) (см. также работу Бурхэма и Андерсона (Burnham and Anderson, 2002)). Альтернативой информационному критерию Акаике является байесовский информационный критерий (Bayesian Information Criterion — BIC), который можно интерпретировать как байесовскую модель процедуры выбора (Schwarz, 1978). Фрэли и Рафтери (Fraley and Raftery, 1998) показали, как выбрать оптимальное количество кластеров, основываясь на критерии BIC. Приложение критерия BIC к методу *K*-средних описано в работе Пеллега и Мура (Pelleg and Moore, 2000). Хамерли и Элькан (Hamerly and Elkan, 2003) предложили альтернативу критерию BIC, которая в их экспе-

риментах работала лучше. Другой важный байесовский подход к определению количества кластеров (одновременно с присваиванием документов кластерам) описан в работе Чиземан и Штуц (Cheeseman and Stutz, 1996). Два метода определения количества кластеров без внешних критериев описаны в работе Тибширани и др. (Tibshirani et al., 2001).

Мы описали лишь классическую кластеризацию без учителя. В настоящее время исследования направлены на использование априорного знания для управления кластеризацией (см., например, работу Джи и Ксу (Ji and Xu, 2006)), а также на использование интерактивной обратной связи в ходе кластеризации (см., например, работу Хуанга и Митчелла (Huang and Mitchell, 2006)). Файад и др. (Fayyad et al., 1998) предложили способ определения начального приближения в EM-кластеризации. Алгоритмы, способные разбить на кластеры очень большие объемы данных за один проход, описаны в работе Брэдли и др. (Bradly et al., 1998).

Все приложения, перечисленные в табл. 16.1, относятся к кластеризации документов. В других приложениях из области информационного поиска кластеризации подвергаются слова (Crouch, 1988), контексты слов (Schütze and Pedersen, 1995) или слова и документы одновременно (Tishby and Slonim, 2000; Dhillon, 2001; Zha et al., 2001). Одновременная кластеризация слов и документов является примером *совместной кластеризации* (co-clustering), или *бикластеризации* (biclustering).

Глава 17

Иерархическая кластеризация

Плоская кластеризация эффективна и проста, но, как мы видели в главе 16, имеет много недостатков. Алгоритмы, описанные в главе 16, создают простое неструктурированное множество кластеров, используя количество кластеров как входной параметр. Кроме того, эти алгоритмы являются недетерминированными. *Иерархическая кластеризация* (hierarchical clustering) создает иерархию, т.е. структурированное множество, которое является более информативным, чем неструктурированное множество кластеров, создаваемое в ходе плоской кластеризации. [1] Для иерархической кластеризации не требуется заранее задавать количество кластеров, и большинство иерархических алгоритмов, используемых для информационного поиска, являются детерминированными. Однако за эти преимущества алгоритмов иерархической кластеризации приходится расплачиваться более низкой производительностью. Сложность наиболее распространенных алгоритмов иерархической кластеризации является как минимум квадратичной по отношению к количеству документов, в то время как алгоритм *K*-средних и EM-алгоритм имеют линейную сложность (см. раздел 16.4).

В начале главы описывается *агломеративная иерархическая кластеризация* (раздел 17.1). Затем в разделах 17.2–17.4 рассматриваются четыре агломеративных алгоритма, отличающихся используемыми мерами сходства: метод одиночной связи (single-link), метод полной связи (complete-link), метод усреднения по группе (group-average) и метод центроидов (centroid similarity). После этого в разделе 17.5 обсуждаются условия оптимальности иерархической кластеризации. В разделе 17.6 описывается иерархическая *нисходящая*, или *разделяющая*, кластеризация (top-down, or divisive clustering). Раздел 17.7 посвящен автоматическому именованию кластеров, задаче, которую необходимо решать, если результаты кластеризации предъявляются человеку. Вопросы реализации обсуждаются в разделе 17.8. В разделе 17.9 содержатся библиография и рекомендации для дальнейшего чтения, включая публикации, в которых освещаются вопросы иерархической кластеризации, не затронутые в нашей книге.

Существуют определенные различия в применении плоской и иерархической кластеризации для информационного поиска. В частности, иерархическую кластеризацию можно применять для решения любой из задач, перечисленных в табл. 16.1. Фактически примеры кластеризации коллекций, описанные в главе 16, были иерархическими. В принципе, плоскую кластеризацию следует выбирать, когда важна эффективность, а иерархическую — когда возникает одна из проблем, присущих плоской кластеризации (недостаточная структурированность, заранее определенное количество кластеров, недетерминированный характер). Кроме того, многие исследователи считают, что иерархическая кластеризация точнее, чем плоская. Однако по этому вопросу консенсус еще не достигнут (раздел 17.9).

---

[1] Хотя в этой главе рассматриваются только бинарные деревья (рис. 17.1), иерархическую кластеризацию можно легко расширить на любую разновидность деревьев.

17.1. Агломеративная иерархическая кластеризация

Алгоритмы иерархической кластеризации являются либо нисходящими, либо восходящими. Восходящие алгоритмы на начальном этапе сначала рассматривают каждый документ как отдельный кластер, а затем последовательно объединяют (или *агломерируют*) пары кластеров, пока они не сольются в один кластер, содержащий все документы. По этой причине восходящая иерархическая кластеризация называется *агломеративной иерархической кластеризацией* (Hierarchical Agglomerative Clustering — HAC). Нисходящая иерархическая кластеризация основывается на разделении кластера. В ходе нисходящей кластеризации кластеры рекурсивно разделяются до тех пор, пока не будут расщеплены на отдельные документы (раздел 17.6). Восходящая иерархическая кластеризация используется для информационного поиска чаще, чем нисходящая. Именно она является предметом рассмотрения в данной главе.

Прежде чем перейти к анализу конкретных мер сходства, используемых в агломеративной иерархической кластеризации (разделы 17.2–17.4), рассмотрим метод ее графического описания и некоторые ключевые свойства, а затем представим простой вычислительный алгоритм, использующийся в ходе этой кластеризации.

Агломеративная иерархическая кластеризация обычно изображается с помощью *дендрограмм* (dendrogram), показанных на рис. 17.1. Каждое объединение представлено горизонтальной линией. Ордината этой горизонтальной линии представляет собой меру сходства между двумя объединяемыми кластерами (отдельные документы рассматриваются как одноэлементные кластеры). Эта мера сходства называется *комбинационной мерой сходства* (combination similarity) объединенного кластера. Например, комбинационная мера сходства кластера, состоящего из документов *Lloyd's CEO questioned* и *Lloyd's chief / U.S. grilling* на рис. 17.1, примерно равна 0,56. Комбинационная мера сходства одноэлементного кластера трактуется как сходство документа с самим собой, или самоподобие (при использовании косинусной меры сходства оно равно единице).

Переходя по дендрограмме снизу вверх, можно проследить процесс кластеризации. Например, как показано на рис. 17.1, два документа с заголовками *War hero Colin Powell* были объединены первыми, а последнее объединение произошло, когда к кластеру, состоящему из 29 документов, был добавлен документ *Ag trade reform*.

Основное предположение агломеративной иерархической кластеризации заключается в том, что операция объединения является *монотонной*. Это значит, что если $s_1, s_2, ..., s_{K-1}$ — комбинационные меры сходства последовательных объединений в агломеративной иерархической кластеризации, то выполняются неравенства $s_1 \geq s_2 \geq ... \geq s_{K-1}$. Немонотонная иерархическая кластеризация содержит хотя бы одну *инверсию* (inversion) $s_i < s_{i+1}$, противоречащую основному предположению, что на каждом этапе выбирается наилучшее объединение. Пример инверсии показан на рис. 17.12.

При иерархической кластеризации не нужно задавать количество кластеров заранее. Однако в некоторых приложениях необходимо точно такое же разбиение на непересекающиеся кластеры, как и при плоской кластеризации. В этих ситуациях иерархию в определенной точке следует отсечь. Для определения точки отсечения существует много критериев.

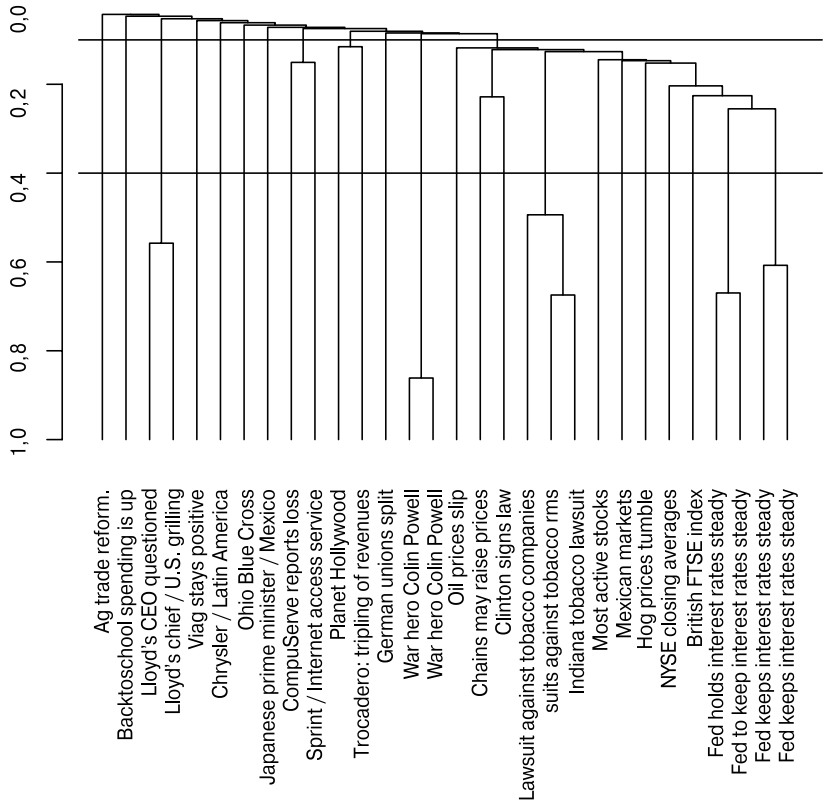


Рис. 17.1. Дендрограмма кластеризации с одиночной связью 30 документов из коллекции Reuters-RCV1. Показаны два возможных сечения дендрограммы: на уровне 0,4 — на 24 кластера и на уровне 0,1 — на 12 кластеров

- Отсечение на заранее указанном уровне сходства. Например, чтобы минимальное комбинационное сходство кластеров было равным 0,4, следует провести сечение дендрограммы на этом уровне. На рис. 17.1 показано, что сечение дендрограммы на уровне $y = 0{,}4$ порождает 24 кластера (группируются только документы с высоким сходством), а сечение дендрограммы на уровне $y = 0{,}1$ порождает 12 кластеров (один крупный кластер финансовых новостей и одиннадцать более мелких кластеров).
- Сечение дендрограммы в точке максимальной разницы между двумя последовательными комбинационными мерами сходства. Такие перепады являются признаком “естественной” кластеризации. Добавление еще одного кластера в этом случае значительно ухудшает качество кластеризации, поэтому желательно делать отсечение до того, как это произойдет. Данная стратегия напоминает поиск точки перегиба на графике *K*-средних (рис. 16.8).
- Применение формулы (16.11)

$$K = \arg\min_{K'} \left[RSS\left(K'\right) + \lambda K' \right],$$

где K' — уровень отсечения иерархии, порождающий K' кластеров, RSS — остаточная сумма квадратов, а λ — штраф за каждый дополнительный кластер. Вместо показателя RSS можно использовать другой показатель искажения.

- Как и в плоской кластеризации, в иерархической кластеризации можно заранее задавать количество кластеров K и выбирать точку отсечения так, чтобы в итоге получить K кластеров.

На рис. 17.2 продемонстрирован простой наивный алгоритм агломеративной иерархической кластеризации. В нем сначала вычисляется матрица сходства C размерности $N \times N$, а затем выполняются $N - 1$ этапов, на которых объединяются наиболее похожие друг на друга в текущий момент кластеры. На каждой итерации два наиболее похожих кластера объединяются, а строки и столбцы, соответствующие объединенному кластеру i в матрице C, вычисляются заново. [2] Разбиение хранится в виде списка объединений A. В массиве I хранится список кластеров, доступных для объединения. Функция Sim(i, m, j) вычисляет сходство кластера j с объединением кластеров i и m. В некоторых алгоритмах агломеративной иерархической кластеризации функция Sim(i, m, j) — это просто функция, зависящая от $C[j][i]$ и $C[j][m]$, например максимум среди этих двух значений в методе одиночной связи.

```
SimpleHAC(d_1, ..., d_N)
1   for n ← 1 to N
2   do for i ← 1 to N
3      do C[n][i] ← Sim(d_n, d_i)
4      I[n] ← 1 (отслеживаем активные кластеры)
5   A ← [] (сохраняем кластеризацию как последовательность объединений)
6   for k ← to N − 1
7   do <i, m> ← argmax_{{⟨i,m⟩: i≠m ∧ I[i]=1 ∧ I[m]=1}} C[i][m]
8      A.Append(<i, m>) (сохраняем объединение)
9      for j ← 1 to N
10     do C[i][j] ← Sim(i, m, j)
11        C[j][i] ← Sim(i, m, j)
12     I[m] ← 0 (деактивизируем кластер)
13  return A
```

Рис. 17.2. Простой, но неэффективный алгоритм агломеративной иерархической кластеризации

Далее мы уточним этот алгоритм для разных мер сходства при кластеризации методами одиночной и полной связи (раздел 17.2), а также для методов усреднения по группе и центроидов (разделы 17.3 и 17.4). Критерии объединения для этих четырех вариантов приведены на рис. 17.3.

[2] Предполагается, что мы используем детерминированный метод принятия решений при равных значениях меры сходства, так что всегда выбирается объединение, которое является первым кластером по отношению к полному упорядочению подмножеств множества документов D.

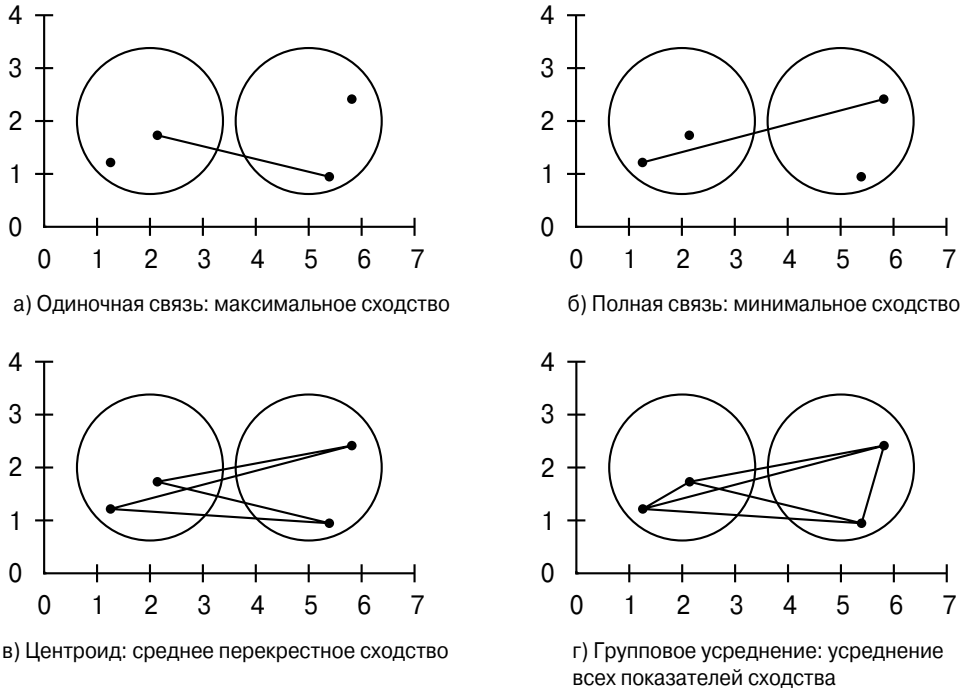


Рис. 17.3. Разные определения сходства кластеров, используемые в агломеративной иерархической кластеризации. Взаимная мера сходства — *это сходство между двумя документами из разных кластеров*

17.2. Кластеризация методами одиночной и полной связи

В *кластеризации методом одиночной связи* (single-link clustering, single-linkage clustering) сходство двух кластеров представляет собой сходство между их *наиболее похожими* элементами (см. рис. 17.3, *а*).[3] Критерий объединения в методе одиночной связи носит *локальный характер*. В этом алгоритме внимание уделяется исключительно области, в которой два кластера наиболее близки друг к другу. Другие, более удаленные, части кластера и его структура не учитываются.

В *кластеризации методом полной связи* (complete-link clustering, complete-linkage clustering) сходство двух кластеров представляет собой сходство между их *наиболее непохожими* элементами (см. рис. 17.3, *б*). Это эквивалентно выбору пары кластеров, объединение которых имеет наименьший диаметр. Критерий объединения в методе полной связи носит *нелокальный характер*: на решение об объединении кластеров может влиять вся структура кластеризации. Это приводит к преобладанию компактных кластеров с маленькими диаметрами над длинными растянутыми кластерами, но одновременно повышает чувствительность к выбросам. Отдельный документ, находящийся далеко от центра, может резко увеличить диаметр возможного объединения и полностью изменить окончательное разбиение.

[3] В этой главе сходство отождествляется с близостью на плоскости.

На рис. 17.4 продемонстрирован процесс кластеризации восьми документов методами одиночной и полной связи. На первых этапах оба метода формируют по четыре идентичных кластера, каждый из двух документов. Затем алгоритм метода одиночной связи объединяет верхние две пары (а после — и нижние). Поскольку в качестве меры сходства в данном алгоритме используется максимальное сходство между элементами, эти кластеры считаются ближайшими. Алгоритм метода полной связи объединяет две левые пары (а затем и две правые), поскольку эти пары ближе друг к другу в соответствии с определением сходства кластеров как минимального сходства их элементов. [4]

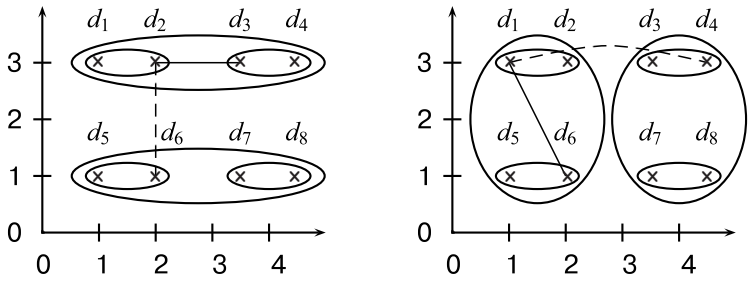


Рис. 17.4. Кластеризация восьми документов методами одиночной связи (слева) и полной связи (справа). Эллипсы соответствуют последовательным этапам кластеризации. Слева: сходство на основе одиночной связи между двумя двухточечными кластерами вверху равно показателю сходства между документами d_2 и d_3 (сплошная линия), которое превышает сходство на основе одиночной связи между двухточечными кластерами слева (пунктирная линия). Справа: сходство на основе полной связи двух двухточечных кластеров вверху равно показателю сходства между документами d_1 и d_4 (пунктирная линия), которая меньше, чем сходство на основе полной связи между двумя левыми двухточечными кластерами (сплошная линия)

Пример кластеризации множества документов с помощью метода одиночной связи приведен на рис. 17.1, а пример кластеризации с помощью метода полной связи того же множества — на рис. 17.5. Проведя отсечение последнего объединения на рис. 17.5, мы получим два кластера примерно одинакового размера (документы 1–16 от *NYSE closing averages* до *Lloyd's chief / U.S. grilling* и документы 17–30 от *Ohio Blue Cross* до *Clinton signs law*). На рис. 17.1 не существует такого сечения дендрограммы, которое приводило бы к разбиению на кластеры примерно одинакового размера.

Как кластеризацию методом одиночной связи, так и кластеризацию методом полной связи можно интерпретировать с помощью теории графов. Пусть s_{k-} комбинационная мера сходства между двумя кластерами, объединенными на этапе k, а $G(s_k)$ — граф, связывающий все точки, сходство между которыми не меньше, чем s_k. Тогда кластеры после этапа k в процессе кластеризации методом одиночной связи представляют собой связные компоненты графа $G(s_k)$, а кластеры после этапа k в процессе кластеризации методом полной связи представляют собой максимальные клики (*cliques*) графа $G(s_k)$. *Компонент связности* (connected component) — это максимальное множество вершин, соединенных

[4] Если вас беспокоит проблема неопределенности, можете считать, что документ d_1 имеет координаты $(1 + \varepsilon, 3 - \varepsilon)$, а все остальные документы имеют целочисленные координаты.

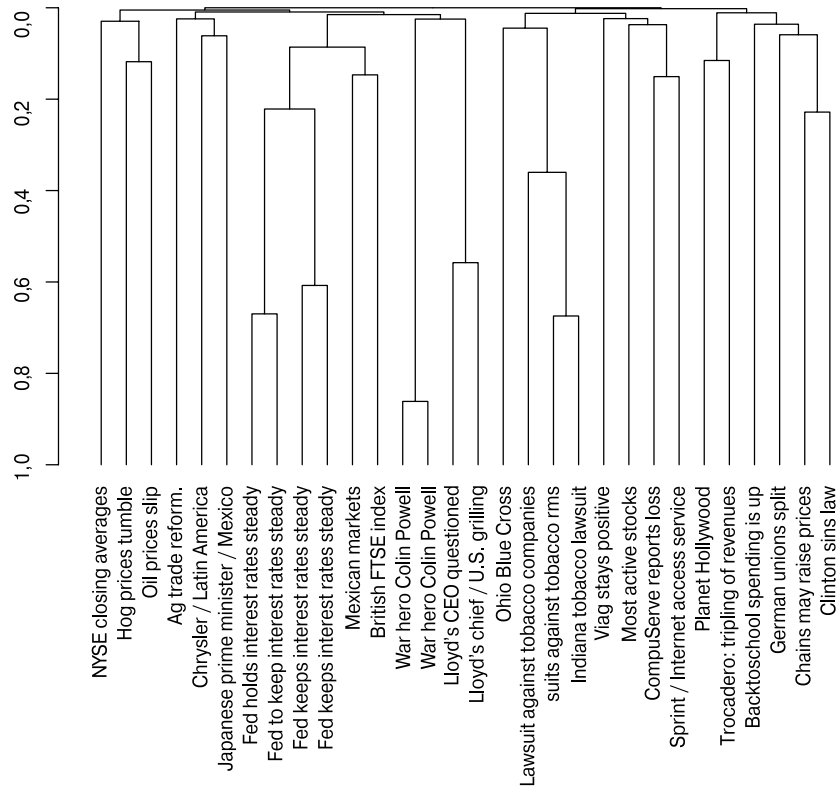


Рис. 17.5. Дендрограмма кластеризации по методу полной связи. Здесь показаны те же 30 документов, которые были кластеризованы с помощью метода одиночной связи на рис. 17.1

между собой так, что для каждой пары точек существует соединяющий их путь. *Клика* (clique) — это множество точек, образующих полный граф (т.е. любые две точки смежные).

Эти интерпретации объясняют названия методов: одиночной связи и полной связи. Кластеры, полученные методом одиночной связи на этапе k, — это максимальные множества точек, между которыми существует хотя бы одна связь по сходству: $s \geq s_k$. Кластеры, полученные методом полной связи на этапе k, — это максимальные множества точек, у каждой из которых есть связь по мере сходства со всеми другими: $s \geq s_k$.

Алгоритмы кластеризации методами одиночной и полной связи сводят задачу оценки качества кластера к оценке меры сходства между двумя документами: двумя наиболее похожими документами в алгоритме метода одиночной связи и двумя наиболее непохожими документами в алгоритме метода полной связи. Оценки сходства между двумя документами не отражают свойства распределения документов в кластере. По этой причине не удивительно, что оба алгоритма часто порождают нежелательные кластеры. Кластеризация методом одиночной связи может создать разбросанные кластеры, как показано на рис. 17.6. Поскольку критерий объединения в этом алгоритме носит строго локальный характер, цепочка пар может растянуться на большое расстояние без учета формы возникающего кластера. Этот эффект называется *сцеплением* (chaining).

Эффект сцепления виден и на рис. 17.1. Последние 11 объединений в алгоритме кластеризации методом одиночной связи (находящиеся над линией $y = 0{,}1$), которые добавляют единичный документ или пару документов, образуют цепочку. Кластеризация методом полной связи, продемонстрированная на рис. 17.5, позволяет избежать этого эффекта. Когда дендрограмма рассекается на этапе последнего объединения, документы разделяются на две группы примерно одинакового объема. В общем, это более полезная организация данных, чем сцепленные кластеры.

Однако кластеризация методом полной связи имеет другой недостаток. Она придает слишком большой вес выбросам, т.е. точкам, не вписывающимся в общую структуру кластера. В примере, показанном на рис. 17.7, четыре документа, d_2, d_3, d_4, d_5, не попали в один кластер из-за выброса d_1 (упражнение 17.1). Кластеризация методом полной связи в данном случае не способна создать наиболее естественную структуру кластеров.

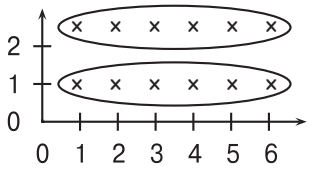


Рис. 17.6. Сцепление, возникающее при кластеризации методом одиночной связи. Локальный критерий в кластеризации методом одиночной связи может породить нежелательно вытянутые кластеры

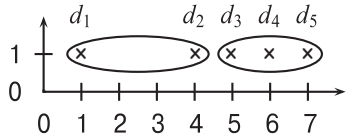


Рис. 17.7. Выбросы в кластеризации методом полной связи. Пять документов имеют координаты x, равные $1+2\varepsilon$, 4, $5+2\varepsilon$, 6 и $7-\varepsilon$. Кластеризация методом полной связи создает два кластера, показанных как эллипсы. Наиболее правильным с интуитивной точки зрения было бы разбиение $\{\{d_1\}, \{d_2, d_3, d_4, d_5\}\}$, но при кластеризации методом полной связи выброс d_1 разбивает кластер $\{d_2, d_3, d_4, d_5\}$ так, как показано на рисунке

17.2.1. Временная сложность

Сложность наивного алгоритма агломеративной иерархической кластеризации, показанного на рис. 17.2, составляет $\Theta(N^3)$, поскольку, чтобы найти элементы с наибольшим сходством на каждой из N - 1 итераций необходимо осуществить полный перебор элементов матрицы C, имеющей размерность $N \times N$.

Для четырех методов, рассматриваемых в этой главе, более эффективным является алгоритм, использующий очереди с приоритетом (рис. 17.8). Его временная сложность — $\Theta(N^2 \log N)$. Строки $C[k]$ матрицы сходства C, имеющей размерность $N \times N$, сортируются в порядке убывания сходства в очередях с приоритетами P. После этого функция $P[k]$.max() возвращает кластер в элементе $P[k]$, который в данный момент больше всех похож на кластер ω_k, где кластер ω_k — это k-й кластер, как в главе 16. После объединения кластеров ω_{k_1} и ω_{k_2} кластер ω_{k_1} используется в качестве представителя объединенного кластера. Функция Sim вычисляет меру сходства между потенциальными парами, подлежащими объединению: в методе одиночной связи — наибольшую, в методе полной связи — наименьшую, в методе GAAC — среднюю (раздел 17.3), а в методе центроидов — меру сходства между центроидами (раздел 17.4). Рассмотрим пример обработки строки матрицы C (рис. 17.8, *внизу*). Временная сложность обоих циклов верхнего

```
    EfficientHAC( d_1, ..., d_N )
1   for n ← 1 to N
2   do for i ← 1 to N
3      do C[n][i].sim ← ( d_n, d_1 )
4         C[n][i].index ← i
5      I[n] ← 1
6      P[n] ← очередь с приоритетами для C[n], упорядоченная по sim
7      P[n].Delete(C[n][n]) (самподобие нам не нужно)
8   A ← []
9   for k ← 1 to N – 1
10  do k_1 ← arg max_{k:I[k]=1} P[k].Max().sim
11     k_2 ← P[k_1].Max().index
12     A.Append(<k_1, k_2>)
13     I[k_1] ← 0
14     P[k_1] ← []
15     for each i with I[i] = 1 ∧ i ≠ k_1
16     do P[i].Delete(C[i][k_1])
17     P[i].Delete(C[i][k_2])
18     C[k_1][i].sim ← Sim(i, k_1, k_2)
19     P[i].Insert(C[i][k_1])
20     C[k_1][i].sim ← Sim(i, k_1, k_2)
21     P[k_1].Insert(C[k_1][i])
22  return A
```

Алгоритм кластеризации	sim(i, k_1, k_2)
Метод одиночной связи	max(sim(i, k_1), sim(i, k_2))
Метод полной связи	min(sim(i, k_1), sim(i, k_2))
Центроид	$\left(\left(\frac{1}{N_m}\vec{v}_m\right),\left(\frac{1}{N_i}\vec{v}_i\right)\right)$
Усреднение по группе	$\frac{1}{(N_m+N_i)(N_m+N_i-1)}\left[(\vec{v}_m+\vec{v}_i)^2-(N_m+N_i)\right]$

Вычисляем C[5]	1	2	3	4	5
	0,2	0,8	0,6	0,4	1,0
Создаем P[5]	2	3	4	1	
(с помощью сортировки)	0,8	0,6	0,4	0,2	
Объединяем 2 и 3, обновляем	2	4	1		
Сходство с 2, удаляем 3	0,3	0,4	0,2		
Удаляем и вновь вставляем 2	4	2	1		
	0,4	0,3	0,2		

Рис. 17.8. Алгоритм агломеративной иерархической кластеризации, использующий очередь с приоритетами. Вверху: алгоритм. В центре: четыре меры сходства. Внизу: пример обработки этапов 6 и 16–19 (искусственный пример, показывающий P[5] для матрицы C размерности 5×5)

уровня (строки 1–7 и 9–21) равна $\Theta(N^2 \log N)$ для реализации очереди с приоритетом, которая поддерживает удаление и вставку за время $\Theta(\log N)$. Следовательно, общая сложность алгоритма равна $\Theta(N^2 \log N)$. В определении функции Sim векторы $\vec{v}_m$ и $\vec{v}_i$ — это векторы сумм $\omega_{k_1} \cup \omega_{k_2}$ и ω_i соответственно, а N_m и N_i — количество документов в множествах $\omega_{k_1} \cup \omega_{k_2}$ и ω_i соответственно.

Аргументом функции EfficientHAC на рис. 17.8 является множество векторов (в отличие от множества типичных документов), поскольку агломеративная кластеризация на основе усреднения по группе и кластеризации с помощью сравнения центроидов (разделы 17.3 и 17.4) в качестве входной информации требуют векторов. Версию алгоритма EfficientHAC для метода полной связи также можно применить к документам, которые не представляются в виде векторов.

В версию кластеризации с одиночной связью с целью ее оптимизации можно ввести массив наилучших кандидатов на объединение (next-best-merge array — NBM), как показано на рис. 17.9. Массив NBM отслеживает наилучшее объединение для каждого кластера. Каждый из двух циклов верхнего уровня на рис. 17.9 имеет временную сложность $\Theta(N^2)$, поэтому общая сложность кластеризации методом одиночной связи составляет $\Theta(N^2)$.

SingleLinkClustering(d_1, ..., d_N)

1 **for** $n \leftarrow 1$ **to** N
2 **do for** $i \leftarrow 1$ **to** N
3 **do** $C[n][i]$.sim $\leftarrow$ SIM(d_n, d_i)
4 $C[n][i]$.index $\leftarrow i$
5 $I[n] \leftarrow n$
6 $NBM[n] \leftarrow \underset{\{X : C[n][i] = n \neq i\}}{\arg\max} X$.sim
7 $A \leftarrow []$
8 **for** $n \leftarrow 1$ **to** $N - 1$
9 **do** $i_1 \leftarrow \underset{\{i : I[i] = i\}}{\arg\max} NBM[i]$.sim
10 $i_2 \leftarrow I[NBM[i_1]$.index$]$
11 A.Append($\langle i_1, i_2 \rangle$)
12 **for** $i \leftarrow 1$ **to** N
13 **do if** $I[i] = i \wedge i \neq i_1 \wedge i \neq i_2$
14 **then** $C[i_1][i]$.sim $\leftarrow C[i][i_1]$.sim $\leftarrow \max(C[i_1][i]$.sim,$C[i_2][i]$.sim)
15 **if** $I[i] = i_2$
16 **then** $I[i] \leftarrow i_1$
17 $NBM[i_1] \leftarrow \underset{X \in \{C[i_1][i] : I[i] = i \wedge i \neq i_1\}}{\arg\max} X$.sim
18 **return** A

Рис. 17.9. Алгоритм агломеративной иерархической кластеризации методом одиночной связи, использующий массив NBM. После слияния двух кластеров, i_1 и i_2, индекс i_1 относится к объединенному кластеру. Если $I[i] = i$, то документ i является представителем текущего кластера. Если $I[i] \neq i$, то документ i включается в кластер, представленный документом $I[i]$, и, следовательно, игнорируется при обновлении вектора $NBM[i_1]$

Можно ли ускорить с помощью массива NBM три других метода агломеративной иерархической кластеризации? Нельзя, так как только кластеризация методом одиночной связи является *устойчивой по отношению к наилучшему объединению*. Допустим, что в кластеризации методом одиночной связи наилучшим кандидатом на объединение с кластером ω_k является кластер ω_j. Тогда после объединения кластера ω_j с третьим кластером $\omega_i \neq \omega_k$ объединение кластеров ω_i и ω_j будет наилучшим кандидатом на объединение с кластером ω_k (упражнение 17.6). Иначе говоря, в кластеризации методом одиночной связи, наилучшим кандидатом на объединение с уже объединенным кластером является один из двух наилучших кандидатов на объединение из его компонент. Это значит, что массив C на каждой итерации можно обновить за время $\Theta(N)$, просто вычислив максимум среди двух чисел в строке 14 на рис. 17.9 для каждого из оставшихся кластеров, количество которых не превышает N.

На рис. 17.10 показано, что устойчивость по отношению к наилучшему объединению не сохраняется в алгоритме кластеризации методом полной связи. Это значит, что мы не можем использовать массив NBM для ускорения кластеризации. После объединения кластера d_2, наилучшего кандидата на объединение с кластером d_3, с кластером d_1 отдельный кластер d_4 становится наилучшим кандидатом на объединение с кластером d_3. Это объясняется тем, что критерий полной связи является нелокальным и зависит от точек, находящихся на большом расстоянии от области соприкосновения двух кандидатов на объединение.

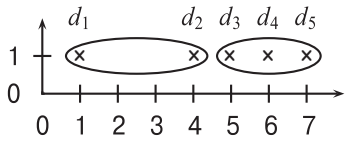


Рис. 17.10. Кластеризация методом полной связи не является устойчивой по отношению к наилучшему объединению. Сначала наилучшим кандидатом на объединение с кластером d_3 является документ d_2. Однако после объединения кластеров d_1 и d_2 наилучшим кандидатом на объединение является кластер d_4. В устойчивом алгоритме, таком как алгоритм метода одиночной связи, наилучшим кандидатом на объединение с d_3 являлся бы кластер $\{d_1, d_2\}$

На практике снижение эффективности алгоритма со сложностью $\Theta(N^2 \log N)$ по сравнению с алгоритмом метода одиночной связи, имеющим сложность $\Theta(N^2)$, невелико, поскольку вычисление меры сходства между двумя документами (например, в виде скалярного произведения) выполняется на порядок медленнее, чем сравнение двух чисел при сортировке. Все четыре алгоритма в этой главе имеют сложность $\Theta(N^2)$ с учетом вычисления меры сходства. Итак, различия по сложности на практике редко принимаются во внимание при выборе алгоритма.

? ***Упражнение 17.1.*** Покажите, что кластеризация с полной связью создает двухкластерное разбиение, показанное на рис. 17.7.

17.3. Агломеративная кластеризация на основе усреднения по группе

Агломеративная кластеризация на основе усреднения по группе (Group-Average Agglomerative Clustering — GAAC), упомянутая на рис. 17.3, *г*, оценивает качество кластера на основе вычисления меры сходства между *всеми* документами. Это позволяет избежать недостатков, свойственных кластеризации методом одиночной или полной связей, в которых сходство между двумя кластерами выражается через меру сходства между парой документов. Метод GAAC иногда называют *кластеризацией методом средней связи* (average-link clustering). В нем вычисляется среднее сходство SIM-GA всех пар документов, включая пары документов из одного кластера. Однако самоподобие в усреднении не учитывается.

$$\mathrm{SIM\text{-}GA}(\omega_i,\omega_j)=\frac{1}{(N_i+N_j)(N_i+N_j-1)}\sum_{d_m\in\omega_i\cup\omega_j}\sum_{d_n\in\omega_i\cup\omega_j,d_n\neq d_m}(\vec{d}_m,\vec{d}_n) \tag{17.1}$$

Здесь $\vec{d}$ — вектор документа d, нормализованный по длине, $(\cdot,\cdot)$ — скалярное произведение, а N_i и N_j — количество документов в кластерах ω_i и ω_j соответственно.

В основе метода GAAC лежат следующие соображения. Наша цель — выбрать два кластера, ω_i и ω_j, которые на следующем этапе иерархической кластеризации будут объединены так, чтобы образованный ими кластер $\omega_k = \omega_i \cup \omega_j$ был связным. Для того чтобы оценить связность кластера ω_k, необходимо оценить меру сходства всех пар документов внутри этого кластера, т.е. документов, принадлежащих ω_i и ω_j.

Среднюю меру сходства между документами можно эффективно вычислить, поскольку сумма мер сходства между отдельными векторами равна показателю сходства их сумм.

$$\sum_{d_m\in\omega_i}\sum_{d_n\in\omega_j}(\vec{d}_m,\vec{d}_n)=\left(\left(\sum_{d_m\in\omega_i}\vec{d}_m\right),\left(\sum_{d_n\in\omega_j}\vec{d}_n\right)\right) \tag{17.2}$$

Учитывая равенство (17.2), приходим к следующему равенству.

$$\mathrm{SIM\text{-}GA}(\omega_i,\omega_j)=\frac{1}{(N_i+N_j)(N_i+N_j-1)}\left[\left(\sum_{d_m\in\omega_i\cup\omega_j}\vec{d}_m\right)^2-(N_i+N_j)\right] \tag{17.3}$$

Слагаемое $(N_i + N_j)$ в правой части равенства представляет собой сумму $N_i + N_j$ самоподобий, каждое из которых равно единице. Используя этот прием, сходство между кластерами можно вычислить за постоянное время (при условии, что суммы векторов $\sum_{d_m\in\omega_i}\vec{d}_m$ и $\sum_{d_m\in\omega_j}\vec{d}_m$ нам уже известны), а не за время $\Theta(N_iN_j)$. Это важно, поскольку в алгоритме EfficientHAC (рис. 17.8) в строках 18 и 20 функцию Sim необходимо вычислять за постоянное время, чтобы алгоритм GAAC был реализован эффективно. Обратите внимание на то, что для двух одноэлементных кластеров равенство (17.3) эквивалентно скалярному произведению.

Равенство (17.2) основано на дистрибутивном законе, которому подчиняется скалярное произведение векторов по отношению к сложению векторов. Поскольку для эффективной реализации алгоритма GAAC этот факт очень важен, данный метод невозможно непосредственно применить для представлений документов, отличающихся от векторов действительных чисел. Кроме того, равенство (17.2) выполняется только для скалярного произведения. Несмотря на то что многие алгоритмы, описанные в книге, почти не изме-

няются, если в качестве меры сходства используется не скалярное произведение, а косинусная мера или евклидово расстояние (см. раздел 14.1), равенство (17.2) остается справедливым только для скалярного произведения. Это фундаментальное отличие метода GAAC от алгоритмов кластеризации методом одиночной или полной связи. Последние упомянутые алгоритмы в качестве входной информации используют лишь квадратную матрицу сходства и не зависят от того, как именно эта мера сходства была вычислена.

Подводя итоги, отметим, что алгоритм GAAC требует, чтобы 1) документы представлялись в виде векторов, 2) векторы были нормализованы по длине так, чтобы показатель самоподобия равнялся единице, 3) в качестве меры сходства между векторами и суммами векторов использовалось скалярное произведение.

Алгоритмы объединения в методе GAAC и методе полной связи совпадают, за исключением того, что в методе GAAC в качестве меры сходства на рис. 17.8 используется функция (17.3). Следовательно, общая сложность алгоритма GAAC не отличается от общей сложности кластеризации с полной связью и составляет $\Theta(N^2 \log N)$. Как и кластеризация методом полной связи, алгоритм GAAC не является устойчивым по отношению к наилучшему объединению (упражнение 17.6). Это значит, что для метода GAAC не существует алгоритма со сложностью $\Theta(N^2)$, в отличие от кластеризации методом одиночной связи, для которой такой алгоритм существует (рис. 17.9).

Кроме того, можно определить меру сходства, усредненную по группе, и не исключая самоподобия.

$$\text{SIM-GA}'(\omega_i, \omega_j) = \frac{1}{(N_i + N_j)^2}\left(\sum_{d_m \in \omega_i \cup \omega_j} \vec{d}_m\right)^2 = \frac{1}{(N_i + N_j)^2}\sum_{d_m \in \omega_i \cup \omega_j}\left(\vec{d}_m, \vec{\mu}(\omega_i \cup \omega_j)\right) \qquad (17.4)$$

Здесь центроид $\vec{\mu}(\omega)$ определяется по формуле (14.1). Это определение эквивалентно интуитивному определению качества кластера как среднего сходства документов $\vec{d}_m$ с центроидом кластера $\vec{\mu}$.

Показатель самоподобия всегда равен единице, т.е. максимально возможному значению показателя сходства между векторами, нормализованными по длине. Доля самоподобий для кластера, имеющего размер i, в формуле (17.4) равна $i/i^2 = 1/i$. Это дает нежелательное преимущество малым кластерам. Их доля самоподобий будет непропорционально высокой. Для документов d_1 и d_2 с показателем сходства s усредненная мера сходства, усредненная по группе, будет равна SIM-GA'(d_1, d_2) = (1 + s)/2. В противоположность этому SIM-GA(d_1, d_2) = $s \le (1 + s)/2$. Мера сходства SIM-GA(d_1, d_2) между двумя документами совпадает с мерами сходства между ними, полученными с помощью алгоритмов кластеризации методом одиночной и полной связи, а также кластеризации методом центроидов. Равенство (17.3), исключающее самоподобие из средней оценки, считается более предпочтительным, поскольку 1) нежелательно штрафовать крупные кластеры из-за меньшего вклада самоподобий и 2) желательно сохранять согласованность при вычислении меры сходства s между документами во всех четырех алгоритмах агломеративной иерархической кластеризации.

? ***Упражнение 17.2.*** Примените кластеризацию на основе усреднения по группе для точек, изображенных на рис. 17.6 и 17.7. Отобразите их на поверхность единичной сферы в трехмерном пространстве, чтобы получить векторы, нормализованные по длине. Отличается ли кластеризация на основе усреднения по группе от кластеризации методами одиночной и полной связей?

17.4. Кластеризация методом центроидов

В алгоритме кластеризации методом центроидов мера сходства между двумя кластерами определяется как мера сходства между их центроидами.

$$\text{SIM-CENT}(\omega_i,\omega_j)=(\vec{\mu}(\omega_i),\vec{\mu}(\omega_j))=\left(\left(\frac{1}{N_i}\sum_{d_m\in\omega_i}\vec{d}_m\right),\left(\frac{1}{N_j}\sum_{d_m\in\omega_j}\vec{d}_n\right)\right), \tag{17.5}$$

$$\text{SIM-CENT}(\omega_i,\omega_j)=\frac{1}{N_iN_j}\sum_{d_m\in\omega_i}\sum_{d_n\in\omega_j}(\vec{d}_m,\vec{d}_n) \tag{17.6}$$

Формула (17.5) задает меру сходства между центроидами. Формула (17.6) показывает, что мера сходства между центроидами эквивалентна средней мере сходства между всеми парами документов из *разных* классов. Таким образом, разница между методом GAAC и кластеризацией методом центроидов заключается в том, что в методе GAAC при вычислении средней попарной меры сходства рассматриваются все пары документов (см. рис. 17.3, *г*), а в методе кластеризации методом центроидов исключаются пары документов, принадлежащие одному кластеру (см. рис. 17.3, *в*).

На рис. 17.11 продемонстрированы первые три итерации по методу центроидов. Первые две итерации образуют кластер $\{d_5, d_6\}$ с центроидом μ_1 и кластер $\{d_1, d_2\}$ с центроидом μ_2, поскольку пары $\{d_5, d_6\}$ и $\{d_1, d_2\}$ имеют наибольшее сходство. На третьей итерации наибольшее сходство обнаруживается между центроидом μ_1 и документом d_4. В результате возникает кластер $\{d_4, d_5, d_6\}$ с центроидом μ_3.

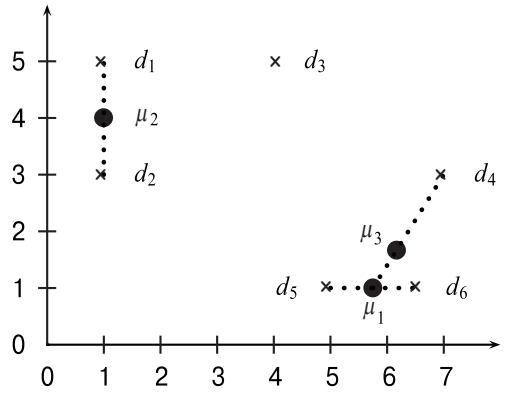


Рис. 17.11. Три итерации по методу центроидов. На каждой итерации происходит объединение двух кластеров с ближайшими центроидами

Как и метод GAAC, метод центроидов не является устойчивым по отношению к наилучшему объединению и, следовательно, имеет сложность $\Theta(N^2\log N)$ (упражнение 17.6).

В противоположность трем остальным методам агломеративной иерархической кластеризации метод центроидов не является монотонным. В ходе выполнения этого алгоритма могут возникать так называемые *инверсии* (inversions): сходство может возрастать, как на рис. 17.12, где показатель сходства мы полагаем равным расстоянию со знаком "минус". При первом объединении мера сходства между документами d_1 и d_2 равна $-(4-\varepsilon)$. При втором объединении мера сходства между центроидом документов d_1 и d_2

(кружочек) и документом d_3 равна приблизительно $-\cos(\pi/6)\times 4 \approx -\sqrt{3}/2\times 4 \approx -3{,}46 > -(4-\varepsilon)$. Это пример инверсии: мера сходства в данной последовательности итераций *возрастает*. В монотонном алгоритме агломеративной иерархической кластеризации мера сходства монотонно *убывает* по мере выполнения итераций.

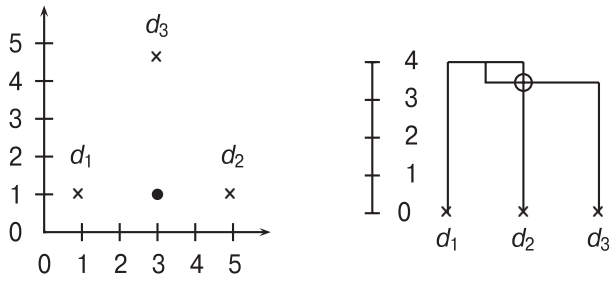


Рис. 17.12. Кластеризация методом центроидов не является монотонной. Документы d_1 в точке $(1+\varepsilon, 1)$, d_2 в точке $(5, 1)$ и d_3 в точке $(3, 1 + 2\sqrt{3})$ являются практически равноудаленными, причем документы d_1 и d_2 ближе друг к другу, чем к документу d_3. Немонотонная инверсия в иерархической кластеризации этих трех точек выражается в виде пересечения линии объединения на дендрограмме. Это пересечение обозначено кружочком

Возрастание меры сходства в последовательности итераций алгомеративной иерархической кластеризации противоречит основному предположению о том, что небольшие кластеры являются более связными, чем крупные. Инверсия в дендрограмме представляется в виде горизонтальной линии объединения, которая лежит *ниже* линии предыдущего объединения. Все линии объединения на рис. 17.1 и 17.5 лежат выше, чем их предшественницы, поскольку алгоритмы методом одиночной и полной связей являются монотонными.

Несмотря на немонотонность, кластеризация методом центроидов используется часто, поскольку ее мера сходства — между центроидами — концептуально проще, чем средняя попарная мера сходства в методе GAAC. Для того чтобы понять принцип кластеризации методом центроидов, достаточно просто посмотреть на рис. 17.11. Рисунок, иллюстрирующий принцип работы алгоритма GAAC, не настолько прост.

? ***Упражнение 17.3.*** Для фиксированной совокупности, состоящей из N документов, в методах одиночной и полной связей существует не более N^2 различных значений меры сходства между кластерами. Сколько разных значений меры сходства между кластерами возникает в алгоритмах GAAC и кластеризации методом центроидов?

17.5. Оптимальность агломеративной иерархической кластеризации

Для того чтобы точно сформулировать условия оптимальности иерархической кластеризации, сначала определим комбинационную меру сходства COMB-SIM разбиения $\Omega = \{\omega_1, ..., \omega_K\}$ как наименьшую комбинационную меру сходства любого из ее K кластеров.

$$\text{COMB-SIM}(\{\omega_1, ..., \omega_K\}) = \min_K \text{COMB-SIM}(\omega_k)$$

Напомним, что комбинационная мера сходства кластера ω, созданного путем слияния кластеров ω_1 и ω_2, — это мера сходства между самими кластерами ω_1 и ω_2.

Будем считать, что разбиение $\Omega = \{\omega_1, ..., \omega_K\}$ является *оптимальным*, если все разбиения Ω', состоящие из k кластеров ($k \leq K$), имеют меньшую комбинационную меру сходства.

$$|\Omega'| \leq |\Omega| \Rightarrow \text{COMB-SIM}(\Omega') \leq \text{COMB-SIM}(\Omega)$$

На рис. 17.12 показано, что кластеризация методом центроидов не является оптимальной. Комбинационная мера сходства разбиения $\{\{d_1, d_2\}, \{d_3\}\}$ (при $K = 2$) равна $-(4 - \varepsilon)$, а комбинационная мера сходства разбиения $\{\{d_1, d_2, d_3\}\}$ (при $K = 1$) равна $-3{,}46$. Таким образом, разбиение $\{\{d_1, d_2\}, \{d_3\}\}$, созданное в результате первого объединения, не является оптимальным, поскольку существует разбиение с меньшим количеством кластеров $\{\{d_1, d_2, d_3\}\}$, которое имеет более высокую комбинационную меру сходства. Кластеризация с помощью центроидов не является оптимальной, так как в ней могут появляться инверсии.

Приведенное выше определение оптимального разбиения было бы мало полезным, если бы для его определения нужно было знать всю предысторию объединений. Однако можно показать (упражнение 17.4), что *комбинационную меру сходства* разбиения в трех монотонных алгоритмах можно определить, даже не зная его предыстории. Приведем эти прямые определения.

- **Алгоритм одиночной связи.** Комбинационная мера сходства кластера ω в методе одиночной связи — это наименьшая мера сходства среди всех разбиений кластера надвое, где мера сходства среди разбиений надвое представляет собой наибольшую меру сходства между любыми двумя документами, принадлежащими разным частям кластера.

  $$\text{COMB-SIM}(\omega) = \min_{\{\omega':\omega' \subset \omega\}} \max_{d_i \in \omega'} \max_{d_i \in \omega - \omega'} \text{SIM}(d_i, d_j)$$

- Здесь каждая пара $<\omega', \omega - \omega'>$ является возможным разбиением кластера ω на две части.

- **Алгоритм полной связи.** Комбинационная мера сходства кластера ω — это наименьшая мера сходства между любыми двумя точками в кластере ω: $\min_{d_i \in \omega} \min_{d_j \in \omega} \text{SIM}(d_i, d_j)$.

- **Алгоритм GAAC.** Комбинационная мера сходства кластера ω — это средняя попарная мера сходства между всеми документами в кластере ω (без учета самоподобия) (см. формулу (17.3)).

При таком определении комбинационной меры сходства оптимальность является свойством совокупности кластеров, а не процесса, приводящего к ее созданию.

Теперь можно доказать оптимальность кластеризации методом одиночной связи с помощью индукции по количеству кластеров K. Для простоты изложения приведем доказательство для варианта, в котором нет двух пар документов с одинаковыми мерами сходства, однако это доказательство можно распространить на вариант с равными мерами сходства.

Базой индукции в нашем доказательстве является разбиение, содержащее $K = N$ одноэлементных кластеров и имеющее комбинационную меру сходства, равную единице. Это

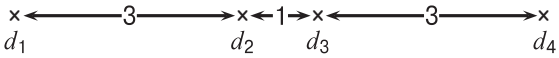
ее максимально возможное значение. Индуктивное предположение заключается в том, что разбиение Ω_K, полученное методом одиночной связи и состоящее из K кластеров, является оптимальным: COMB-SIM(Ω_K) > COMB-SIM(Ω'_K) для всех Ω'_K. Допустим от противного, что разбиение Ω_{K-1}, полученное в результате объединения двух наиболее схожих кластеров в разбиении Ω_K, не является оптимальным и что к оптимальному разбиению, состоящему из $K-1$ кластеров, ведет другая последовательность объединений Ω'_K, Ω'_{K-1}. Предположение о том, что разбиение Ω'_{K-1} является оптимальным, а разбиение Ω_{K-1} — нет, можно записать следующим образом: COMB-SIM(Ω'_{K-1}) > COMB-SIM(Ω_{K-1}).

Вариант 1. Два документа, связанные мерой сходства s = COMB-SIM(Ω'_{K-1}), принадлежат одному и тому же кластеру в разбиении Ω_K. Они могут принадлежать одному и тому же кластеру, только если в последовательности объединений, приведших к разбиению Ω_K, существует объединение с мерой сходства, меньшей, чем s. Отсюда следует, что s > COMB-SIM(Ω_K). Следовательно, COMB-SIM(Ω'_{K-1}) = s > COMB-SIM(Ω_K) > COMB-SIM(Ω'_K) > COMB-SIM(Ω'_{K-1}). Это противоречие.

Вариант 2. Два документа, связанные мерой сходства s = COMB-SIM(Ω'_{K-1}), не принадлежат одному и тому же кластеру в разбиении Ω_K. Однако s = COMB-SIM(Ω'_{K-1}) > COMB-SIM(Ω_{K-1}), поэтому по правилу объединения по методу одиночной связи в процессе создания разбиения Ω_K эти кластеры должны были быть объединены. Это противоречие.

Итак, разбиение Ω_{K-1} является оптимальным.

В отличие от кластеризации методом одиночной связи кластеризация методом полной связи и разбиение по алгоритму GAAC не являются оптимальными, что доказывает следующие пример.



Оба алгоритма сначала объединяют две точки, лежащие на расстоянии, равном единице (d_2 и d_3), а значит, не могут найти двухкластерное разбиение {{d_1, d_2}, {d_3, d_4}}. Однако это разбиение является оптимальным в соответствии с критерием оптимальности, принятым в алгоритме полной связи и алгоритме GAAC.

Однако критерии объединения, принятые в методах полной связи и GAAC, точнее аппроксимируют сферичность, чем критерий, принятый в алгоритме односвязной кластеризации. Во многих приложениях желательно работать со сферическими кластерами. Таким образом, даже если, на первый взгляд, кластеризация методом одиночной связи кажется предпочтительной благодаря своей оптимальности, следует помнить, что эта оптимальность соответствует неправильному критерию для многих приложений кластеризации документов.

Свойства четырех алгоритмов агломеративной иерархической кластеризации, описанных в этой главе, перечислены в табл. 17.1. Для кластеризации документов мы рекомендуем применять алгоритм GAAC, поскольку, как правило, он порождает разбиения, лучше других подходящие для многих приложений. У него нет эффекта сцепления и инверсий и он является нечувствительным к выбросам.

У этой рекомендации есть два исключения. Во-первых, метод GAAC невозможно применить, если документ не представляется в виде вектора действительных чисел. В таких случаях кластеризацию следует проводить с помощью метода полной связи.

Таблица 17.1. Сравнение алгоритмов агломеративной иерархической кластеризации

Метод	*Комбинационное сходство*	*Временная сложность*	*Является ли оптимальным*	*Комментарий*
Одиночной связи	Максимальная взаимная мера сходства любых двух документов	$\Theta(N^2)$	Да	Эффект сцепления
Полной связи	Минимальная взаимная мера сходства любых двух документов	$\Theta(N^2 \log N)$	Нет	Чувствительность к выбросам
Усреднения по группе	Усреднение всех значений меры сходства	$\Theta(N^2 \log N)$	Нет	Лучший выбор для большинства приложений
Центроидов	Усреднение всех значения взаимной меры сходства	$\Theta(N^2 \log N)$	Нет	Могут возникать инверсии

Во-вторых, в некоторых приложениях целью кластеризации является не создание полной иерархии или исчерпывающее разбиение всего множества документов. Например, задачей приложения может быть идентификация первого сообщения о событии в потоке новостей. Для решения этой задачи можно найти компактный кластер среди документов, поступивших на ленту новостей за короткий период времени и непохожих на предыдущие документы. Например, документы, отосланные по ленте новостей через несколько минут после атаки на Всемирный торговый центр 11 сентября 2001 года, образуют именно такой кластер. Для решения этой задачи хорошо подходят разные варианты кластеризации методом одиночной связи, поскольку в таком случае следует учитывать структуру небольших частей векторного пространства, а не ее глобальную структуру.

Аналогичный подход мы опишем в разделе 19.6, посвященном нахождению дубликатов в вебе, где кластеризация методом одиночной связи используется в форме алгоритма “объединение–поиск” (union-find algorithm). И снова на решение о том, является ли группа документов дубликатом другой группы, не влияют документы, расположенные далеко. В этом случае кластеризация методом одиночной связи является хорошим выбором.

? ***Упражнение 17.4.*** Докажите эквивалентность двух определений комбинационной меры сходства: динамическое определение из раздела 17.1 и статическое определение из раздела 17.5.

17.6. Нисходящая кластеризация

До сих пор мы рассматривали только агломеративную кластеризацию, однако разбиение на кластеры может осуществляться и сверху вниз. Такая иерархическая кластеризация называется *нисходящей* (top-down clustering), или *разделяющей (или дивизивной)* (divisive clustering). На первом ее этапе все документы принадлежат одному кластеру. Затем этот кластер разделяется на части с помощью алгоритма плоской кластеризации. Данная процедура выполняется рекурсивно, пока каждый документ не окажется в своем отдельном кластере.

Нисходящая кластеризация концептуально более сложна, чем восходящая, поскольку ее составной частью является дополнительный алгоритм плоской кластеризации. Тем не менее она может оказаться более эффективной, если не генерировать полную иерархию вплоть до отдельных документов. При фиксированном количестве верхних уровней

и использовании эффективного алгоритма плоской кластеризации, например метода *K*-средних, алгоритм нисходящей кластеризации является линейным по количеству документов и кластеров. По этой причине он работает намного быстрее, чем алгоритмы агломеративной иерархической кластеризации, сложность которых является как минимум квадратичной.

Существуют примеры того, что в некоторых ситуациях нисходящие алгоритмы создают более точную иерархию, чем восходящие. Соответствующая ссылка на биссекторный метод *K*-средних приведена в разделе 17.9. Алгоритмы восходящей кластеризации принимают решения о разделении на кластеры, исходя из информации о локальных образцах и не учитывая их общее распределение. Принятые на более ранних этапах решения отменить невозможно. Преимущества нисходящей кластеризации проявляются в учете полной информации о глобальном распределении уже на самых верхних уровнях разбиения.

17.7. Именование кластеров

Во многих приложениях плоской и иерархической кластеризации, особенно в задачах анализа данных и пользовательских интерфейсах (см. Табл. 16.1), люди непосредственно работают с кластерами. В таких ситуациях кластеры необходимо именовать так, чтобы пользователи могли понимать их содержание.

Дифференцированное именование кластеров (differential cluster labeling) выбирает метки кластеров, сравнивая распределение термов в одном кластере с распределением термов в других кластерах. При дифференцированном именовании кластеров можно использовать методы выбора признаков, описанные в разделе 13.5.[5] В частности, для нахождения отличительных меток кластеров можно использовать критерий взаимной информации (см. раздел 13.5.1), критерий прироста информации и критерий хи-квадрат (см. раздел 13.5.2). К наилучшим результатам часто приводит сочетание дифференцированного критерия со штрафом за низкую частоту, поскольку редко встречающиеся термины не всегда являются хорошими представителями кластера в целом.

В табл. 17.2 приведены результаты применения трех методов именования для кластеризации с помощью алгоритма *K*-средних. В этом примере между критерием взаимной информации и критерием хи-квадрат практически нет никакой разницы, поэтому мы не приводим данные для второго подхода.

Внутрикластерное именование (cluster-internal labeling) вычисляет метку, которая зависит исключительно от самого кластера. Одним из методов внутрикластерного именования является выделение заголовка документа, ближайшего к центроиду. Заголовки читать легче, чем список терминов. Полный заголовок может также содержать важный контекст, который может не попасть в список первых десяти терминов, отобранных с помощью метода взаимной информации. В вебе роль, аналогичную заголовку, может играть текст ссылок, поскольку он может служить краткой аннотацией страницы, на которую указывает.

---

[5] Выбор наиболее часто встречающихся терминов относится к недифференцированным методам выбора признаков, рассмотренным в разделе 13.5. Его можно использовать и для именования кластеров.

Таблица 17.2. Автоматически вычисленные метки трех из десяти кластеров (4, 9 и 10), полученных в ходе кластеризации первых 10 тысяч документов из коллекции Reuters-RCV1 с помощью метода *K*-средних. В последних трех столбцах приведены аннотации кластеров, полученные с помощью трех методов именования: центроидов (определения терминов с наибольшим весом в центроиде), взаимной информации и заголовка документа, ближайшего к центроиду кластера. Термины, полученные только одним из первых двух методов, выделены полужирным шрифтом

		Метод именования		
	Количество документов	Центроид	Взаимная информация	Заголовок
4	622	oil plant mexico production crude **power 000 refinery gas** bpd	plant oil production **barrels** crude bpd mexico **dolly capacity petroleum**	MEXICO: Hurricane Dolly heads for Mexico coast
9	1017	police security **russian** people military peace killed told **grozny court**	police killed military security peace told **troops forces rebels** people	RUSSIA: Russia's Lebed meets rebel chief in Chechnya
10	1259	00 000 tonnes traders futures wheat prices **cents september** tonne	**delivery** traders futures tonne tonnes **desk** wheat prices 000 00	USA: Export Business - Grain/oilseeds complex

В табл. 17.2 заголовок кластера 9 подсказывает, что многие его документы касаются конфликта в Чечне. Критерий взаимной информации этот факт никак не отражает. Однако один документ нельзя считать репрезентативным по отношению ко всем документам кластера. Например, заголовок для кластера 4 может ввести в заблуждение. Основная тема этого кластера — нефть. Статьи об урагане Долли появились в данном кластере лишь потому, что этот ураган повлиял на цены на нефть.

В качестве метки можно также использовать список терминов, имеющих наибольшие веса в центроиде кластера. Такие термины (а еще лучше — фразы, особенно — именные группы) намного точнее отражают тему кластера, чем несколько заголовков, даже если они не прошли фильтрацию с помощью дифференцированных методов. Однако чтение списка фраз отнимает у пользователя больше времени, чем чтение хорошего заголовка.

Внутрикластерные методы очень эффективны, но они не могут отличить термины, которые часто появляются во всей коллекции, от терминов, которые часто появляются лишь в пределах кластера. Такие термины, как year или Tuesday, могут часто встречаться в кластере, но это никак не поможет понять содержание кластера, имеющего специальную тему, например “нефть”.

В табл. 17.2 метод центроидов извлек больше неинформативных терминов (000, court, cents, september), чем метод взаимной информации (forces, desk). Тем не менее большинство терминов, выбранных с помощью этих методов, являются вполне информативными. Просматривая данные термины, можно ясно представить себе содержание документов кластера.

При именовании кластеров в ходе иерархической кластеризации возникают дополнительные сложности. Дело в том, что в иерархической кластеризации внутренний узел дерева необходимо отличать не только от других одноуровневых узлов, но и от его родительского и дочерних узлов. Документы в дочерних узлах по определению являются членами соответствующего родительского узла, поэтому для поиска меток, отличающих родительские узлы от дочерних, невозможно применить наивный дифференцированный

метод. Однако с помощью более сложного критерия, основанного на частоте термина во всей коллекции и его частоте в конкретном кластере, можно определить, является термин более информативным для дочернего узла или для родительского (см. раздел 17.9).

17.8. Вопросы реализации

Эффективность решения большинства задач, требующих вычисления большого количества скалярных произведений, можно повысить с помощью инвертированного индекса. Это относится и к агломеративной иерархической кластеризации. Экономия за счет инвертированного индекса велика, если существует много нулевых показателей сходства, — либо из-за того, что много документов не имеют общих терминов, либо из-за использования обширного списка стоп-слов.

В задачах более низкой размерности возможна более агрессивная оптимизация, которая позволяет избежать большей части вычислений попарных значений меры сходства (упражнение 17.10). Однако для задач высокой размерности такие алгоритмы неизвестны. Такая же ситуация характерна для метода k ближайших соседей (kNN), описанного в разделе 14.7.

Применяя алгоритм GAAC к крупной коллекции документов в пространстве большой размерности, следует избегать плотных центроидов, т.е. векторов, содержащих мало нулей. Из-за них временная сложность кластеризации составляет $\Theta(MN^2\log N)$, где M — размер словаря, тогда как временная сложность кластеризации методом полной связи составляет $\Theta(M_{ave}N^2\log N)$, где M_{ave} — средний размер словаря документа. Таким образом, для больших словарей кластеризация методом полной связи может оказаться более эффективной, чем реализация алгоритма GAAC без оптимизации. В главе 16 эта проблема уже обсуждалась в контексте метода K-средних. Тогда было предложено два решения: усечение центроидов (сохранение только терминов с большими весами) и представление кластеров с помощью разреженных медоидов вместо плотных центроидов. Эти решения можно применить и в алгоритме GAAC, и в методе центроидов.

Даже после таких усовершенствований алгоритмы агломеративной иерархической кластеризации имеют сложность $\Theta(N^2)$ или $\Theta(N^2\log N)$ и поэтому неприемлемы для крупных коллекций, состоящих из миллиона и более документов. К таким крупным коллекциям алгоритмы агломеративной иерархической кластеризации можно применять только в сочетании с алгоритмами плоской кластеризации, такими как метод K-средних. Напомним, что алгоритм K-средних требует, чтобы было задано начальное приближение (см. рис. 16.5). Если начальное приближение выбрано неудачно, итоговое разбиение будет иметь низкое качество. Мы можем использовать агломеративную иерархическую кластеризацию для выбора хорошего начального приближения. Если метод агломеративной иерархической кластеризации применяется к подмножеству документов, размер которого имеет порядок $\sqrt{N}$, то общая сложность комбинации метода K-средних и агломеративной иерархической кластеризации составит $\Theta(N)$. Это объясняется тем, что сложность квадратичного алгоритма, примененного к подмножеству, размер которого имеет порядок $\sqrt{N}$, составляет $\Theta(N)$. Для того чтобы гарантировать линейную сложность, такую же модификацию можно сделать для алгоритма, имеющего сложность $\Theta(N^2\log N)$. Этот алгоритм называется *алгоритмом картечи* (Buckshot algorithm). Он сочетает детерминированный характер и высокую надежность алгоритмов агломеративной иерархической кластеризации с эффективностью метода K-средних.

?

Упражнение 17.5. Разбиение методом одиночной связи можно вычислить по минимальному остовному дереву графа (minimum spanning tree). Минимальное остовное дерево графа соединяет все вершины, причем сумма весов ребер дерева минимальна. В нашем случае вес ребра — это расстояние между двумя документами. Покажите, что если $\Delta_{k-1} > \Delta_k > ... > \Delta_1$ — веса ребер минимального остовного дерева, то эти ребра соответствуют $k - 1$ объединениям в процессе кластеризации методом одиночной связи.

Упражнение 17.6. Покажите, что кластеризация методом одиночной связи является устойчивой по отношению к наилучшему объединению, а алгоритм GAAC и метод центроидов — нет.

Упражнение 17.7.

1. Рассмотрите процесс кластеризации коллекции документов на двух разных языках по методу 2-средних. Какого результата вы ожидаете?
2. Следует ли ожидать такого же результата от алгоритма агломеративной иерархической кластеризации?

Упражнение 17.8. Загрузите коллекцию Reuters-21758. Сохраните только документы, содержащие классы *crude*, *interest* и *grain*. Отбросьте документы, принадлежащие более чем одному из этих трех классов. Вычислите кластеризации методами 1) одиночной связи, 2) полной связи, 3) GAAC и 4) центроидов. Отсеките каждую дендрограмму на второй ветви сверху, чтобы получить $K = 3$ кластеров. Вычислите индекс Ранда для каждого из четырех разбиений. Какой метод кластеризации эффективнее других?

Упражнение 17.9. Предположим, что в ходе агломеративной иерархической кластеризации найдены $K = 7$ кластеров, удовлетворяющих заданному критерию качества. Является ли это разбиение наилучшим среди всех разбиений, состоящих из семи кластеров?

Упражнение 17.10. Рассмотрите задачу кластеризации N точек на линии методом одиночной связи.

Покажите, что достаточно вычислить сумму приблизительно N показателей сходства. Какова сложность кластеризации этого множества точек на линии методом одиночной связи?

Упражнение 17.11. Докажите, что кластеризация методами одиночной и полной связей, а также кластеризация с помощью усреднения по группе являются монотонными.

Упражнение 17.12. Для N точек существует не меньше N^k плоских разбиений на K кластеров (см. раздел 16.2). Чему равно количество разных иерархических разбиений (или дендрограмм) N документов? Чего больше — плоских или иерархических разбиений для заданных K и N?

17.9. Библиография и рекомендации для дальнейшего чтения

Отличный обзор методов кластеризации опубликован Джейн и др. (Jain et al., 1999). Первыми публикациями, посвященными конкретным алгоритмам агломеративной иерархической кластеризации, были работы Кинга (King, 1967) по односвязной кластеризации, Снита и Сокала (Sneath and Sokal, 1973) по полносвязной кластеризации и методам GAAC, а также Ланса и Уильямса (Lance and Williams, 1967) о разнообразных алгоритмах иерархической кластеризации. Односвязный алгоритм, показанный на рис. 17.9, похож на *алгоритм Крускала* (Kruskal's algorithm) для построения минимального остовного дерева. Доказательство правильности алгоритма Крускала, основанное на теории графов (аналогичное доказательству из раздела 17.5), изложено в книге Кормена и др. (Cormen et al., 1990). Связь между минимальным остовным деревом и односвязной кластеризацией устанавливается в упражнении 17.5.

Часто утверждают, что алгоритмы иерархической кластеризации порождают более качественные разбиения, чем алгоритмы плоской кластеризации (Jain and Dubes, 1988; Cutting et al., 1992; Larsen and Aone, 1999), хотя недавно были получены экспериментальные результаты, демонстрирующие обратное (Zhao and Karypis, 2002). Даже без консенсуса относительно свойств этих алгоритмов нет никаких сомнений в том, что результаты EM-алгоритма и метода *K*-средних очень изменчивы, поскольку они часто сходятся к локальному оптимуму. Алгоритмы агломеративной иерархической кластеризации, описанные в этой главе, являются детерминированными, а значит, более предсказуемыми.

В некоторых работах сложность кластеризации методом полной связи, а также кластеризации, полученной с помощью усреднения по группе или методом центроидов, полагается равной $\Theta(N^2)$ (Day and Edelsbrunner, 1984; Voorhees, 1985*b*; Murtagh, 1983), поскольку вычисление сходства документов на порядок сложнее, чем простое сравнение, которое является основной операцией, выполняемой на этапе объединения после вычисления матрицы сходства $N \times N$.

Описанный нами алгоритм центроидов предложен Ворхес (Voorhees, 1985*b*). Ворхес считала, что в информационном поиске методы полной связи и центроидов предпочтительнее, чем метод одиночной связи. Алгоритм "картечи" впервые был описан в работе Каттинга и др. (Cutting et al., 1993). Аллан и др. (Allan et al., 1998) применили кластеризацию методом одиночной связи для обнаружения новых тем в потоке новостей.

Важным, но не описанным в этой главе методом агломеративной иерархической кластеризации, является *метод Варда* (Ward's method) (Ward Jr., 1963; El-Hamdouchi and Willett, 1986). Он называется также *кластеризацией с минимальной дисперсией* (minimum variance clustering). На каждом этапе этот метод выбирает объединение с наименьшим значением RSS (см. главу 16). Критерий объединения в методе Варда (функция всех расстояний от центроида) тесно связан с критерием объединения в методе GAAC (функция всех мер сходства с центроидом).

Несмотря на прикладную важность метода, относительно мало публикаций посвящено именованию кластеров. Попескул и Унгар (Popescul and Ungar, 2000) получили хорошие результаты, скомбинировав критерий хи-квадрат и частоту термина в коллекции. Гловер и др. (Glover et al., 2002*b*) использовали для именования кластеров веб-страниц критерий информационной выгоды. Подход Штейна и Мейер цу Айссена (Stein and Meyer zu Eissen, 2004) основан на онтологии. Более сложная проблема именования узлов

в иерархии (требующая различения более общих меток для родительских узлов и более специфических меток для дочерних узлов) рассматривалась в работе Гловера и др. (Glover et al., 2002*a*) и Триратпитука и Каллана (Treeratpituk and Callan, 2006). Некоторые алгоритмы кластеризации пытаются сначала найти множество меток, а затем построить (часто перекрывающиеся) кластеры вокруг этих меток, тем самым избежав проблемы именования вообще (Zamir and Etzioni, 1999; Kaki, 2005; Osinski and Weiss, 2005). Исчерпывающее исследование, посвященное сравнению качества такой кластеризации с алгоритмами кластеризации, описанными в этой главе и в главе 16, еще не опубликовано. В принципе, работу о многодокументной аннотации (McKeown and Radev, 1995) также можно применить для решения задачи именования кластеров, но аннотации нескольких документов обычно длиннее, чем короткие фрагменты текстов, необходимые для именования кластеров (ср. раздел 8.7). Представление кластеров в виде, понятном для пользователей, сводится к проблеме пользовательского интерфейса. В качестве введение в этот вопрос мы рекомендуем книгу Баеза-Йейтса и Рибейро-Нето (Baeza-Yates and Riveiro-Neto, 1999) (см. также веб-страницу `http://www.searchuserinterfaces.com/`— *Примеч. ред.*).

Примером эффективного разделяющего алгоритма является биссекторный метод *K*-средних (Steinbach et al., 2000). Алгоритмы *спектральной кластеризации* (Kannan et al., 2000; Dhillon, 2001; Zha et al., 2001; Ng et al., 2001*a*), включая алгоритм *разбиения по главному направлению* (Principal Direction Divisive Partitioning — PDDP), в котором биссекторные решения основаны на сингулярном разложении (см. главу 18) (Boley, 1998; Savaresi and Boley, 2004), с вычислительной точки зрения сложнее, чем биссекторный метод *K*-средних, однако их преимуществом является детерминированный характер.

В отличие от метода *K*-средних и EM-алгоритмов, большинство алгоритмов кластеризации не имеют вероятностной интерпретации. Исключение составляет иерархическая кластеризация, основанная на моделировании (Vaithyanathan and Dom, 2000; Kamvar et al., 2002; Castro et al., 2004).

Методология оценок, описанная в разделе 16.3, применима и к иерархической кластеризации. Специальные способы оценки иерархий обсуждаются в работах Фоулкеса и Мэллоуза (Fowlkes and Mallows, 1983), Ларсена и Аона (Larsen and Aone, 1999) и Саху и др. (Sahoo et al., 2006).

Хорошим инструментом иерархической кластеризации является пакет программ R (R Development Core Team, 2005). Функция `hclust` из пакета R реализует кластеризацию методом одиночной связи, кластеризацию методом полной связи, кластеризацию методом усреднения по группе, кластеризацию методом центроидов, а также метод Варда. Другой вариант — функция `median`, представляющая каждый кластер с помощью медоида (см. главу 16). Поддержка кластеризации векторов в пространствах большой размерности реализовано в пакете программ CLUTO (`http://glaros.dtc.umn.edu/gkhome/views/cluto`).

Глава 18

Разложение матриц и латентно-семантическое индексирование[1]

В главе 6 было введено понятие *матрицы "термин–документ"* (term-document matrix) — матрицы C размерности $M \times N$, строки которой соответствовали терминам, а столбцы — документам из коллекции. Даже для коллекции небольшого размера матрица "термин–документ" C может содержать несколько десятков тысяч строк и столбцов. В разделе 18.1.1 описывается категория операций, известных в линейной алгебре как *разложение матриц* (matrix decomposition). В разделе 18.2 используется особая форма разложения матриц, позволяющая построить малоранговую (low-rank) аппроксимацию матрицы "термин–документ". В разделе 18.3 исследуется применение таких малоранговых аппроксимаций к индексированию и поиску документов. Эта процедура называется *латентно-семантическим индексированием* (latent semantic indexing). Несмотря на то что латентно-семантическое индексирование не оказало значительного влияния на вычисление весов и ранжирование в задачах информационного поиска, оно остается многообещающим подходом к кластеризации во многих областях, включая кластеризацию текстовых документов (см. раздел 16.6). Выявлением его полного потенциала активно занимаются многие исследователи.

Читатели, которым нет необходимости освежать знания по линейной алгебре, могут пропустить раздел 18.1. Тем не менее мы рекомендуем обратить внимание на пример 18.1, в котором освещаются свойства собственных значений, используемые на протяжении этой главы.

18.1. Обзор сведений из линейной алгебры

Рассмотрим кратко основные сведения из линейной алгебры, которые нам понадобятся в этой главе. Пусть C — матрица действительных чисел размерности $M \times N$. В матрице "термин–документ" все элементы являются неотрицательными. *Ранг* (rank) матрицы — это число ее линейно независимых строк (или столбцов). Таким образом, $rank(C) \leq \min\{M, N\}$. Квадратная матрица $r \times r$, все недиагональные элементы которой равны нулю, называется *диагональной* (diagonal matrix). Ее ранг равен количеству ненулевых элементов, стоящих на диагонали. Если все r диагональных элементов такой матрицы равны единице, то эта матрица называется единичной матрицей размерности r и обозначается как I_r.

Для квадратной матрицы C, имеющей размерность $M \times M$, и вектора $\vec{x}$, не все элементы которого равны нулю, числа λ, удовлетворяющие равенству

$$C\vec{x} = \lambda\vec{x}, \tag{18.1}$$

[1] Хотя в отечественной переводной литературе пока что доминирует именно такой вариант перевода, все чаще употребляется альтернативный вариант "скрытое семантическое индексирование". — *Примеч. ред.*

называются *собственными значениями* (eigenvalues) матрицы *C*. Вектор $\vec{x}$, состоящий из *N* элементов и удовлетворяющий равенству (18.1) при заданном собственном значении λ, называется *правым собственным вектором* (right eigenvector). Собственный вектор, соответствующий наибольшему собственному значению, называется *главным собственным вектором* (principal eigenvector). Аналогично *левым собственным вектором* (left eigenvector) матрицы *C* называется вектор $\vec{y}$, состоящий из *M* элементов и удовлетворяющий равенству

$$\vec{y}^T C = \lambda \vec{y}^T . \tag{18.2}$$

Количество ненулевых собственных значений матрицы *C* не превышает ранга матрицы *C*.

Собственные значения матрицы можно найти, решив *характеристическое уравнение* (characteristic equation), которое можно получить, переписав уравнение (18.1) в виде $(C - \lambda I_M)\vec{x} = 0$. Собственные значения матрицы *C* являются решениями уравнения $|(C - \lambda I_M)| = 0$, где $|S|$ — определитель матрицы *S*. Уравнение $|(C - \lambda I_M)| = 0$ представляет собой полиномиальное уравнение порядка *M* относительно λ и может иметь не более *M* решений, являющихся собственными значениями матрицы *C*. Эти собственные значения в общем случае могут быть комплексными, даже если все элементы матрицы *C* являются действительными.

Проанализируем теперь некоторые свойства собственных значений и собственных векторов, чтобы понять основную идею сингулярного разложения матриц, изложенную в разделе 18.2. Посмотрим на связь между умножением матрицы на вектор и собственными значениями.

Пример 18.1. Рассмотрим матрицу

$$S = \begin{pmatrix} 30 & 0 & 0 \\ 0 & 20 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Очевидно, что ранг этой матрицы равен трем, и она имеет три ненулевых собственных значения, $\lambda_1 = 30$, $\lambda_2 = 20$ и $\lambda_3 = 1$, которым соответствуют три собственных вектора.

$$\vec{x}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \ \vec{x}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \text{ и } \vec{x}_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Умножение матрицы *S* на собственный вектор действует так, будто мы умножаем на него единичную матрицу, причем результаты умножения для разных собственных векторов разные. Теперь рассмотрим произвольный вектор, например $\vec{v} = \begin{pmatrix} 2 \\ 4 \\ 6 \end{pmatrix}$. Его можно представить как линейную комбинацию трех собственных векторов матрицы *S*. В данном случае

$$\vec{v} = \begin{pmatrix} 2 \\ 4 \\ 6 \end{pmatrix} = 2\vec{x}_1 + 4\vec{x}_2 + 6\vec{x}_3 .$$

Допустим, что мы умножаем матрицу S на вектор $\vec{v}$.

$$\begin{aligned} S\vec{v} &= S\left(2\vec{x}_1 + 4\vec{x}_2 + 6\vec{x}_3\right) = \\ &= 2S\vec{x}_1 + 4S\vec{x}_2 + 6S\vec{x}_3 = \\ &= 2\lambda_1\vec{x}_1 + 4\lambda_2\vec{x}_2 + 6\lambda_3\vec{x}_3 = \\ &= 60\vec{x}_1 + 80\vec{x}_2 + 6\vec{x}_3 \end{aligned} \tag{18.3}$$

Пример 18.1 показывает, что даже если вектор $\vec{v}$ совершенно произвольный, умножение на него матрицы S можно выразить через собственные значения и собственные векторы. Более того, из равенства (18.3) интуитивно ясно, что на произведение $S\vec{v}$ слабо влияют слагаемые, соответствующие малым собственным значениям матрицы S. В нашем примере $\lambda_3 = 1$, поэтому вклад третьего слагаемого в правой части равенства (18.3) мал. Фактически, если полностью проигнорировать вклад этого слагаемого, произведение $S\vec{v}$ было бы равно $\begin{pmatrix} 60 \\ 80 \\ 0 \end{pmatrix}$, а не $\begin{pmatrix} 60 \\ 80 \\ 6 \end{pmatrix}$. Эти два вектора относительно близки друг другу в любой метрике (например, по длине разности между векторами).

Можно предположить, что влияние малых собственных значений (и соответствующих собственных векторов) на произведение матрицы на вектор мало. Мы уточним эту интуитивную догадку позднее, когда приступим к изучению разложений матриц и малоранговых аппроксимаций в разделе 18.2. Сначала рассмотрим собственные значения и собственные векторы специальных матриц, представляющих для нас особый интерес.

Для *симметричной* матрицы S собственные векторы, соответствующие разным собственным значениям, являются *ортогональными*. Более того, если матрица S состоит из действительных чисел и является симметричной, то все ее собственные значения являются действительными числами.

Пример 18.2. Рассмотрим действительную и симметричную матрицу

$$S = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}. \tag{18.4}$$

Из характеристического уравнения $|S - \lambda I| = 0$ получаем квадратное уравнение $(2 - \lambda)^2 = 0$, решениями которого являются собственные значения 3 и 1. Соответствующие собственные векторы $\begin{pmatrix} 1 \\ -1 \end{pmatrix}$ и $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ являются ортогональными.

18.1.1. Разложение матриц

В этом разделе рассматриваются способы *факторизации* квадратных матриц, т.е. представления таких матриц в виде произведения матриц, полученных с помощью собственных векторов исходной матрицы. Этот процесс называется *разложением матриц* (matrix decomposition). Разложение матриц, описанное в настоящем разделе, лежит в основе методов анализа текста, изложенных в разделе 18.3, в котором будут проанализированы разложения прямоугольных матриц “термин–документ”. Разложения квадратных матриц, описанные в данном разделе, проще и могут исследоваться с достаточной математической строгостью, что позволит читателям глубже понять этот процесс. Детальное

математическое описание более сложных матричных разложений в разделе 18.2 выходит за рамки нашей книги.

Начнем с двух теорем о представлении квадратной матрицы в виде произведения трех матриц специального вида. Теорема 18.1 касается разложения действительной матрицы на три множителя. Теорема 18.2 относится к квадратным симметричным матрицам. На ней основано сингулярное разложение, которому посвящена теорема 18.3.

Теорема 18.1 (теорема о диагонализации матрицы). *Пусть S — квадратная действительная матрица, имеющая размерность $M \times M$ и M линейно независимых собственных векторов. Тогда существует собственное разложение матрицы*

$$S = U\Lambda U^{-1}, \tag{18.5}$$

где столбцы матрицы U — собственные векторы матрицы S, а Λ — диагональная матрица, диагональные элементы которой — это собственные значения матрицы S, расположенные в убывающем порядке.

$$\begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ldots & \\ & & & \lambda_M \end{pmatrix}, \lambda_i \geq \lambda_{i+1} \tag{18.6}$$

Если все собственные значения различны, то разложение — единственное.

Для того чтобы понять смысл теоремы 18.1, отметим, что U — матрица, столбцы которой представляют собой собственные векторы матрицы S.

$$U = \begin{pmatrix} \vec{u}_1 & \vec{u}_2 & \ldots & \vec{u}_M \end{pmatrix} \tag{18.7}$$

Отсюда следует, что

$$\begin{aligned} SU &= S\begin{pmatrix} \vec{u}_1 & \vec{u}_2 & \ldots & \vec{u}_M \end{pmatrix} = \\ &= \begin{pmatrix} \lambda_1\vec{u}_1 & \lambda_2\vec{u}_2 & \ldots & \lambda_M\vec{u}_M \end{pmatrix} = \\ &= \begin{pmatrix} \vec{u}_1 & \vec{u}_2 & \ldots & \vec{u}_M \end{pmatrix} \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ldots & \\ & & & \lambda_M \end{pmatrix}. \end{aligned}$$

Таким образом, $SU = U\Lambda$, или $S = U\Lambda U^{-1}$.

Перейдем теперь к тесно связанному с этой теоремой разложению симметричной квадратной матрицы в виде произведения матриц, образованных их собственными векторами. Это позволит нам описать основной инструмент анализа текстов — сингулярное разложение (раздел 18.2).

Теорема 18.2 (теорема о симметричной диагонализации). *Пусть S — квадратная симметричная действительная матрица, имеющая размерность $M \times M$ и M линейно независимых собственных векторов. Тогда существует симметричное диагональное разложение этой матрицы*

$$S = Q\Lambda Q^T, \tag{18.8}$$

где столбцы матрицы Q являются ортогональными и нормализованными (имеют единичную длину и состоят из действительных чисел) собственными векторами матрицы S, а матрица Λ — диагональная матрица, элементами которой являются собственные значения матрицы S. Более того, все элементы матрицы Q являются действительными числами и $Q^{-1} = Q^T$.

Диагональное разложение матрицы позволяет найти малоранговые аппроксимации матриц "термин–документ".

Упражнение 18.1. Чему равен ранг приведенной ниже матрицы 3×3?

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 2 & 1 \end{pmatrix}$$

Упражнение 18.2. Покажите, что число $\lambda = 2$ является собственным значением матрицы

$$\begin{pmatrix} 6 & -2 \\ 4 & 0 \end{pmatrix},$$

и найдите соответствующий собственный вектор.

Упражнение 18.3. Вычислите единственное собственное разложение матрицы 2×2, заданной равенством (18.4).

18.2. Матрицы "термин–документ" и сингулярные разложения

Разложения, описанные выше, применялись к квадратным матрицам. Однако нас интересует матрица "термин–документ" C, имеющая размерность $M \times N$, где за редким исключением $M \neq N$. Более того, маловероятно, что матрица C симметрична. Сначала рассмотрим расширение симметричного диагонального разложения, которое называется *сингулярным* (singular value decomposition). Затем в разделе 18.3 мы покажем, как построить приближение матрицы C. Математическое обоснование сингулярного разложения матрицы выходит за рамки нашей книги. Отметим лишь, что связь между сингулярным разложением и симметричным диагональным разложением из раздела 18.1.1 устанавливается теоремой 18.3. Для данной матрицы C пусть U — матрица, имеющая размерность $M \times M$, столбцы которой представляют собой ортогональные собственные векторы матрицы CC^T, а матрица V — матрица, имеющая размерность $M \times N$, столбцы которой являются ортогональными векторами матрицы C^TC, где C^T — транспонированная матрица C.

Теорема 18.3. *Пусть r — ранг матрицы C, имеющей размерность $M \times N$. Тогда существует сингулярное разложение (Singular Value Decomposition — SVD) матрицы C*

$$C = U\Sigma V^T, \tag{18.9}$$

где

1) *собственные значения $\lambda_1, ..., \lambda_r$ матрицы CC^T совпадают с собственными значениями матрицы C^TC;*
2) *пусть $\sigma_i = \sqrt{\lambda_i}$*, где $\lambda_i \geq \lambda_{i+}1$ *при всех* $1 \leq i \leq r$; тогда *матрица* Σ *размерности* $M \times N$ *состоит из чисел* $\Sigma_{ii} = \sigma_i$ *при всех* $1 \leq i \leq r$ *и нулей*.

Числа σ_i называются *сингулярными значениями* (singular values) матрицы C. Вместо прямого доказательства теоремы 18.3, которое выходит за рамки книги, довольно поучительно исследовать связи между теоремами 18.2 и 18.3.

Умножив матрицу (18.9) на ее транспонированную матрицу, получим

$$CC^T = U\Sigma V^T V\Sigma U^T = U\Sigma^2 U^T. \qquad (18.10)$$

Теперь напомним, что в равенстве (18.10) левая часть представляет собой квадратную симметричную действительную матрицу, а правая часть является ее симметричным диагональным разложением (теорема 18.2). Что же представляет собой матрица CC^T? Это квадратная матрица, строки и столбцы которой соответствуют M терминам. Элемент (i, j) этой матрицы является мерой перекрытия между i- и j-м терминами, основанной на их совместном появлении в документах. Точный математический смысл этого элемента зависит от способа определения весов терминов, с помощью которого построена матрица C. Рассмотрим вариант, в котором матрица C представляет собой матрицу инцидентности (см. рис. 1.1). В этом случае элемент (i, j) матрицы CC^T — это количество документов, в которых одновременно встречаются i- и j-й термины.

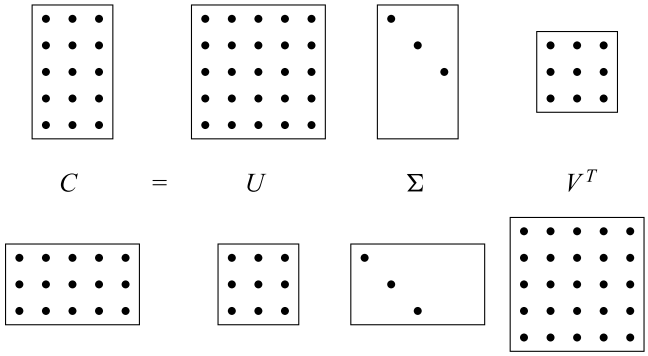


Рис. 18.1. Схематическая иллюстрация сингулярного разложения матрицы (18.9). Рассмотрены два варианта: в верхней части $M > N$, а в нижней — $M < N$

Записывая сингулярное разложение, удобно выразить матрицу Σ в виде матрицы $r \times r$ с сингулярными значениями на диагонали, поскольку все ее остальные элементы, в том числе диагональные при $i > r$, равны нулю. Соответственно, можно проигнорировать $M - r$ столбцов матрицы U, расположенных справа и соответствующих нулевым строками матрицы Σ. Аналогично можно проигнорировать правые $N - r$ столбцов матрицы V, поскольку в матрице V^T они соответствуют строкам, которые умножаются на $N - r$ нулевых столбцов матрицы Σ. Эту форму записи сингулярного разложения иногда называют *сокращенным сингулярным разложением* (reduced SVD) или *усеченным сингулярным разложением* (truncated SVD). Она рассматривается в упражнении 18.9. Все числовые примеры, описанные в этой главе, используют именно сокращенную форму записи сингулярного разложения.

Пример 18.3. Проиллюстрируем сингулярное разложение матрицы 4×2 с рангом 2. Сингулярными значениями этой матрицы являются числа $\Sigma_{11} = 2{,}236$ и $\Sigma_{22} = 1$.

$$C = \begin{pmatrix} 1 & -1 \\ 0 & 1 \\ 1 & 0 \\ -1 & 1 \end{pmatrix} = \begin{pmatrix} -0{,}632 & 0{,}000 \\ 0{,}316 & -0{,}707 \\ -0{,}316 & -0{,}707 \\ 0{,}632 & 0{,}000 \end{pmatrix} \begin{pmatrix} 2{,}236 & 0{,}000 \\ 0{,}000 & 1{,}000 \end{pmatrix} \begin{pmatrix} -0{,}707 & 0{,}707 \\ -0{,}707 & -0{,}707 \end{pmatrix}. \qquad (18.11)$$

Как и матричное разложение, описанное в разделе 18.1.1, сингулярное разложение матрицы можно вычислить, используя разные алгоритмы, многие из которых реализованы с помощью свободно распространяемого программного обеспечения (раздел 18.5).

Упражнение 18.4. Пусть матрица

$$C = \begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{18.12}$$

является матрицей "термин–документ", описывающей некую коллекцию. Вычислите матрицу совместной встречаемости CC^T. Что собой представляют диагональные элементы матрицы CC^T?

Упражнение 18.5. Убедитесь, что сингулярное разложение матрицы (18.12) выглядит следующим образом.

$$U = \begin{pmatrix} -0,816 & 0,000 \\ -0,408 & -0,707 \\ -0,408 & 0,707 \end{pmatrix},\ \Sigma = \begin{pmatrix} 1,732 & 0,000 \\ 0,000 & 1,000 \end{pmatrix},\ V^T = \begin{pmatrix} -0,707 & -0,707 \\ 0,707 & -0,707 \end{pmatrix} \tag{18.13}$$

Проверьте все условия теоремы 18.3.

Упражнение 18.6. Предположим, что C — матрица инцидентности "термин–документ". Что представляют собой элементы матрицы C^TC?

Упражнение 18.7. Пусть матрица

$$C = \begin{pmatrix} 0 & 2 & 1 \\ 0 & 3 & 0 \\ 2 & 1 & 0 \end{pmatrix} \tag{18.14}$$

является матрицей "термин–документ", элементами которой являются частоты терминов. Таким образом, термин 1 встречается в документе 2 дважды, а в документе 3 — один раз. Вычислите матрицу CC^T. Убедитесь, что ее наибольшие элементы соответствуют двум терминам, которые чаще других встречаются одновременно в одном и том же документе.

18.3. Малоранговые аппроксимации

Рассмотрим проблему аппроксимации матрицы, которая, на первый взгляд, слабо связана с информационным поиском. Опишем решение этой проблемы с помощью метода сингулярного разложения, а затем продемонстрируем ее приложение к информационному поиску.

Для заданных матрицы C, имеющей размерность $M \times N$, и положительного целого числа k необходимо найти матрицу C_k, имеющую размерность $M \times N$ и ранг, не превышающий число k, т.е. минимизировать *норму Фробениуса* (Frobenius norm) матрицы $X = C - C_k$, определенную формулой

$$\|X\|_F = \sqrt{\sum_{i=1}^{M}\sum_{j=1}^{N} X_{ij}^2}\ . \tag{18.15}$$

Таким образом, норма Фробениуса матрицы X оценивает различие между матрицами C_k и C. Наша цель — найти матрицу C_k, которая минимизирует это различие, а ее ранг не превышает числа k. Пусть r — ранг матрицы C. Очевидно, что в этом случае $C_r = C$, а норма Фробениуса разности между данными матрицами равна нулю. Если число k намного меньше r, то матрица C_r называется *малоранговой аппроксимацией* (low-rank approximation).

Проблему малоранговой аппроксимации можно решить с помощью сингулярного разложения. Для этого необходимо выполнить следующую трехэтапную процедуру.

1. Для заданной матрицы C построим ее сингулярное разложение в форме (18.9), т.е. $C = U\Sigma V^T$.
2. По матрице Σ построим матрицу Σ_k, заменив нулями $r - k$ наименьших сингулярных значений на диагонали матрицы Σ.
3. Вычислим матрицу $C_k = U\Sigma_k V^T$ — малоранговую аппроксимацию матрицы C.

Ранг матрицы C_k не превышает числа k. Это следует из того факта, что матрица Σ_k имеет не больше k ненулевых значений. Напомним пример 18.1: влияние малых собственных значений на произведение матриц мало. Следовательно, кажется правдоподобным, что замена этих малых собственных значений нулями незначительно изменит произведение матриц, которое будет по-прежнему близко к матрице C. Теорема Эккарта–Юнга (Eckart–Young theorem) утверждает, что описанная выше процедура порождает матрицу с рангом k и ошибкой, имеющей наименьшую норму Фробениуса.

Теорема 18.4 (теорема Эккарта–Юнга)

$$\min_{Z|rank(Z)=k} \|C - Z\|_F = \|C - C_k\|_F = \sqrt{\sum_{i=k+1}^{r} \sigma_i^2}. \tag{18.16}$$

Напомним, что сингулярные значения отсортированы в порядке убывания: $\sigma_1 \geq \sigma_2 \geq \ldots$ Следовательно, по теореме 18.4 матрица C_k является наилучшей аппроксимацией матрицы C среди матриц, ранг которых равен k. При этом ошибка (норма Фробениуса матрицы $C - C_k$) равна σ_{k+1}. Таким образом, чем больше число k, тем меньше эта ошибка (в частности, при $k = r$ эта ошибка равна нулю, поскольку $\Sigma_r = \Sigma$; если $r < M$ и $r < N$, то $\sigma_{r+1} = 0$, а значит, $C_r = C$).

Для того чтобы лучше понять, почему процесс отсечения наименьших $r - k$ сингулярных значений матрицы Σ приводит к аппроксимации ранга k с небольшой ошибкой, изучим структуру матрицы C_k.

$$C_k = U\Sigma_k V^T \tag{18.17}$$

$$= U \begin{pmatrix} \sigma_1 & 0 & 0 & 0 & 0 \\ 0 & \ldots & 0 & 0 & 0 \\ 0 & 0 & \sigma_k & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \ldots \end{pmatrix} = V^T = \tag{18.18}$$

$$= \sum_{i=1}^{k} \sigma_i \vec{u}_i \vec{v}_i^T. \tag{18.19}$$

Здесь $\vec{u}_i$ и $\vec{v}_i$ — i-е столбцы матриц U и V соответственно. Следовательно, $\vec{u}_i \vec{v}_i^T$ — матрица, ранг которой равен единице, поэтому достаточно представить матрицу C_k как сумму k матриц единичного ранга, каждая из которых умножена на соответствующее

сингулярное значение. По мере увеличения индекса i вес матриц $\vec{u}_i\vec{v}_i^T$ выражается все более уменьшающимися сингулярными значениями σ_i.

? ***Упражнение 18.8.*** Вычислите аппроксимацию матрицы C из примера 18.12 с рангом, равным единице, используя сингулярное разложение из упражнения 18.5. Чему равна норма Фробениуса ошибки этой аппроксимации?

Упражнение 18.9. Проанализируйте вычисления, выполненные в упражнении 18.8. Следуя схеме, изображенной на рис. 18.2, отметим, что аппроксимация, имеющая единичный ранг, представляет собой скаляр σ_1. Обозначим через U_1 первый столбец матрицы U, а через V_1 — первый столбец матрицы V. Покажите, что аппроксимацию матрицы C, имеющей единичный ранг, можно записать как $U_1\sigma_1V_1^T = \sigma_1U_1V_1^T$.

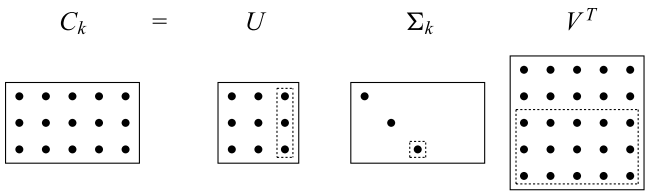


Рис. 18.2. Иллюстрация малоранговой аппроксимации с помощью сингулярного разложения. Прямоугольники, изображенные пунктиром, охватывают элементы матрицы, на которые влияет "обнуление" наименьших сингулярных значений

Упражнение 18.10. Упражнение 18.19 можно обобщить на аппроксимации, ранг которых равен k. Обозначим через U_k' и V_k' "усеченные" матрицы, образованные только первыми k столбцами матриц U и V соответственно. Следовательно, матрица U_k' имеет размерность $M \times k$, а матрица $V_k'^T$ — $k \times N$. Таким образом,

$$C_k = U_k'\Sigma_k'V_k'^T, \tag{18.20}$$

где Σ_k' — квадратная подматрица матрицы Σ_k, имеющая размерность $k \times k$ с сингулярными значениями σ_1, ..., σ_k на диагонали. Основное преимущество представления (18.20) состоит в том, что оно позволяет исключить большое количество излишних нулевых столбцов матриц U и V, а значит, избежать умножения столбцов, не влияющих на малоранговую аппроксимацию. Этот вариант сингулярного разложения, более простой с вычислительной точки зрения, иногда называется сокращенным, или усеченным. Запишите для матрицы C из упражнения 18.3 матрицы Σ_2 и Σ_2'.

18.4. Латентно-семантическое индексирование

Обсудим теперь приближение матрицы "термин–документ" C матрицей более малого ранга, построенной с помощью метода сингулярного разложения. Малоранговая аппроксимация матрицы C создает новое представление каждого документа в коллекции. Запросы также переводятся в малоранговое представление, что позволяет вычислять

сходство между запросами и документами. Этот процесс называется *латентно-семантическим индексированием* (Latent Semantic Indexing — LSI).

Однако сначала изложим цели такого представления. Напомним, что в главе 6 было введено векторное представление документов и запросов. Эта модель имеет много преимуществ, в том числе единообразное представление запросов и документов в виде векторов, ранжирование с помощью косинусной меры сходства, возможность дифференцированно взвешивать термины, а также применение в других задачах, например для кластеризации и классификации. Тем не менее векторная модель не лишена недостатков, поскольку не в состоянии решить две классические проблемы, связанные с естественными языками: синонимию и полисемию. *Синонимия* (synonymy) означает ситуацию, в которой два разных слова (например, `car` и `automobile`) имеют одно и то же значение. Векторное представление не позволяет учесть связь между синонимичными терминами, таким как `car` и `automobile`, поэтому каждый из них порождает отдельную координатную ось векторного пространства. Вследствие этого вычисленная мера сходства $\left(\vec{q},\vec{d}\right)$ между запросом $\vec{q}$ (скажем, `car`) и документом $\vec{d}$, содержащим как термин `car`, так и термин `automobile`, недооценивает истинное сходство между ними, которое ищет пользователь. *Полисемия* (polysemy) возникает, когда термин, например `charge`, имеет несколько значений. В этом случае вычисленная мера сходства $\left(\vec{q},\vec{d}\right)$ переоценивает истинное сходство между запросом и документом, которое ищет пользователь. Можем ли мы учитывать одновременное появление терминов (например, термина `charge` в документе, содержащем термин `steed`, и в документе, содержащем термин `electron`), чтобы распознать латентно-семантические ассоциации терминов и разрешить эти проблемы?

Даже если коллекция имеет небольшой размер, матрица "термин–документ" C, как правило, содержит несколько десятков тысяч строк и столбцов, а ранг измеряется десятками тысяч. При латентно-семантическом индексировании (которое иногда называется латентно-семантическим анализом (Latent Semantic Analysis — LSA)) с помощью сингулярного разложения создается приближение C_k матрицы "термин–документ", причем число k выбирается намного меньшим, чем ранг матрицы C. В экспериментальной работе, процитированной ниже, число k обычно выбирается равным нескольким сотням. Таким образом, мы отображаем каждую строку/столбец (соответствующую термину/документу) в k-мерное пространство. Это пространство определяется k главными собственными векторами (соответствующими наибольшим собственным значениям) матриц CC^T и C^TC. Обратите внимание на то, что матрица C_k остается матрицей размерности $M \times N$ независимо от числа k.

Новое k-мерное представление, полученное с помощью латентно-семантического индексирования, теперь можно использовать для вычисления меры сходства между векторами. Вектор запроса $\vec{q}$ отображается в соответствующее представление с помощью преобразования

$$\vec{q}_k = \Sigma_k^{-1} U_k^T \vec{q} \,. \tag{18.21}$$

Теперь между запросом и документом, между двумя документами и между двумя терминами можно вычислить косинусную меру сходства, описанную в главе 6. Подчеркнем, что равенство (18.21) не является "специализированным" для запроса, здесь мы просто оперируем векторами в пространстве терминов. Это значит, что если представление коллекции документов получено с помощью латентно-семантического индексирова-

ния, то новый документ, не принадлежащий коллекции, может быть переведен в новое представление с помощью формулы (18.21). Таким образом, существует возможность постепенного добавления документов в представление коллекции, полученное с помощью латентно-семантического индексирования. Разумеется, такое постепенное увеличение коллекции не позволяет учесть совместную встречаемость терминов в новых документах (и даже игнорирует любые новые термины, которые они содержат). Следовательно, по мере добавления новых документов представление, полученное с помощью латентно-семантического индексирования, постепенно деградирует и в итоге возникает необходимость вычислить его заново.

Точность аппроксимации матрицы C матрицей C_k позволяет надеяться на то, что соответствующие значения косинусной меры сходства будут сохранены: если запрос был близок к документу в исходном пространстве, то он останется близким к нему и в k-мерном пространстве. Однако сам по себе этот факт не очень интересен, особенно если разреженный вектор запроса $\vec{q}$ отображается в плотный вектор запроса $\vec{q}_k$ в пространстве более малой размерности. Это значительно повышает вычислительные затраты по сравнению с обработкой запроса $\vec{q}$ в его первоначальном виде.

Пример 18.4. Рассмотрим следующую матрицу “термин–документ”.

	d_1	d_2	d_3	d_4	d_5	d_6
ship	1	0	1	0	0	0
boat	0	1	0	0	0	0
ocean	1	1	0	0	0	0
voyage	1	0	0	1	1	0
trip	0	0	0	1	0	1

Ее сингулярное разложение представляет собой произведение следующих трех матриц. В первую очередь, матрица U.

	1	2	3	4	5
ship	–0,44	–0,30	0,57	0,58	0,25
boat	–0,13	–0,33	–0,59	0,00	0,73
ocean	–0,48	–0,51	–0,37	0,00	–0,61
voyage	–0,70	0,35	0,15	–0,58	0,16
trip	–0,26	0,65	–0,41	0,58	–0,09

Когда сингулярное разложение (SVD) применяется к матрице “термин–документ”, матрица U называется *сингулярной матрицей терминов* (SVD term matrix). Сингулярные значения таковы.

$$\Sigma = \begin{pmatrix} 2{,}16 & 0{,}00 & 0{,}00 & 0{,}00 & 0{,}00 \\ 0{,}00 & 1{,}59 & 0{,}00 & 0{,}00 & 0{,}00 \\ 0{,}00 & 0{,}00 & 1{,}28 & 0{,}00 & 0{,}00 \\ 0{,}00 & 0{,}00 & 0{,}00 & 1{,}00 & 0{,}00 \\ 0{,}00 & 0{,}00 & 0{,}00 & 0{,}00 & 0{,}39 \end{pmatrix}$$

В заключение матрица V^T, которая в контексте матрицы "термин–документ" называется *сингулярной матрицей документов* (SVD document matrix), имеет следующий вид.

	d_1	d_2	d_3	d_4	d_5	d_6
1	–0,75	–0,28	–0,20	–0,45	–0,33	–0,12
2	–0,29	–0,53	–0,19	0,63	0,22	0,41
3	0,28	–0,75	0,45	–0,20	0,12	–0,33
4	0,00	0,00	0,58	0,00	–0,58	0,58
5	–0,53	0,29	0,63	0,19	0,41	–0,22

Обнуляя все, кроме двух наибольших, сингулярные значения матрицы Σ, получим приведенную ниже матрицу.

$$\Sigma_2 = \begin{pmatrix} 2{,}16 & 0{,}00 & 0{,}00 & 0{,}00 & 0{,}00 \\ 0{,}00 & 1{,}59 & 0{,}00 & 0{,}00 & 0{,}00 \\ 0{,}00 & 0{,}00 & 0{,}00 & 0{,}00 & 0{,}00 \\ 0{,}00 & 0{,}00 & 0{,}00 & 0{,}00 & 0{,}00 \\ 0{,}00 & 0{,}00 & 0{,}00 & 0{,}00 & 0{,}00 \end{pmatrix}$$

Отсюда следует, что матрица C_2 имеет следующий вид.

	d_1	d_2	d_3	d_4	d_5	d_6
1	–1,62	–0,60	–0,44	–0,97	–0,70	–0,26
2	–0,46	–0,84	–0,30	1,00	0,35	0,65
3	0,00	0,00	0,00	0,00	0,00	0,00
4	0,00	0,00	0,00	0,00	0,00	0,00
5	0,00	0,00	0,00	0,00	0,00	0,00

Отметим, что малоранговая аппроксимация, в отличие от первоначальной матрицы C, может содержать отрицательные элементы.

Изучение матриц C_2 и Σ_2 в примере 18.4 показывает, что последние три строки в каждой из этих матриц содержат одни нули. Это значит, что факторизация $U\Sigma V^T$ в равенстве (18.18) может быть выполнена с учетом только двух строк матриц Σ_2 и V^T. Следователь-

но, эти матрицы можно заменить их усеченными аналогами Σ_2' и $(V')^T$. Например, усеченная сингулярная матрица документов $(V')^T$ в данном примере имеет следующий вид.

	d_1	d_2	d_3	d_4	d_5	d_6
1	–0,75	–0,28	–0,20	–0,45	–0,33	–0,12
2	–0,29	–0,53	–0,19	0,63	0,22	0,41

На рис. 18.3 документы из матрицы $(V')^T$ показаны в двух измерениях. Обратите внимание на то, что матрица C_2 плотнее, чем матрица C.

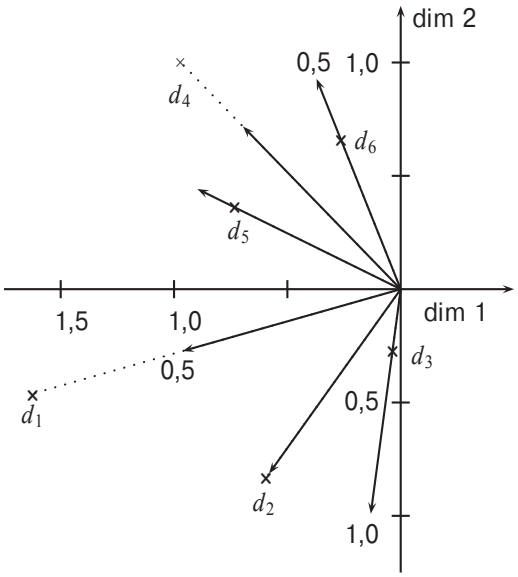


Рис. 18.3. Документы из примера 18.4 представлены в двумерном пространстве в матрице $(V')^T$

Малоранговую аппроксимацию матрицы C матрицей C_k можно рассматривать с более общей точки зрения как *задачу оптимизации с ограничениями*, в которой ограничением является требование, чтобы матрица C_k имела ранг, не превышающий k, а целью — представление терминов и документов с помощью матрицы C, минимизирующей норму Фробениуса ошибки $C - C_k$. При необходимости втиснуть термины/документы в k-мерное пространство латентно-семантическое индексирование должно сводить вместе термины с высокой совместной встречаемостью. Интуитивно ясно, что качество поиска при уменьшении размерности не должно сильно ухудшиться, но на практике может даже *улучшиться*.

Дюмэ (Dumais, 1993; 1995) провела эксперименты, в ходе которых было выполнено латентно-семантическое индексирование документов из коллекции TREC и для сингулярного разложения использовался широко известный алгоритм Ланцоша (Lanczos algorithm). В начале 1990-х годов латентно- семантическое индексирование десятков тысяч

документов на одном компьютере занимало приблизительно один день. В этих экспериментах была достигнута точность на уровне медианы точности участников TREC и выше. Примерно на 20% тем коллекции TREC система, разработанная Дюма, вышла на первое место; вариант с 350 размерностями был чуть лучше в среднем стандартной системы на основе векторного пространства. Ниже перечислены некоторые выводы этой работы, впоследствии подтвержденные многочисленными независимыми исследователями.

- Вычислительные затраты, связанные с сингулярным разложением матриц, значительны; на момент написания этой книги нам неизвестен ни один успешный эксперимент, в котором было бы обработано более миллиона документов. Именно это обстоятельство стало основным препятствием для широкого распространения латентно-семантического индексирования. Одним из способов обойти это препятствие было предложение создать представление LSI на случайно выбранном подмножестве документов, а оставшиеся документы добавить так, как описано в пояснении к равенству (18.21).
- По мере уменьшения числа k полнота поиска, как правило, увеличивается.
- Что наиболее удивительно, значение k, не превышающее нескольких сотен, на самом деле *увеличивает* точность поиска на некоторых запросах коллекции. Это означает, что при правильном выборе числа k латентно-семантическое индексирование частично решает проблему синонимии.
- Латентно-семантическое индексирование лучше всего работает в приложениях, в которых между запросами и документами наблюдается лишь небольшое перекрытие.[2]

В этих экспериментах были также исследованы некоторые модели, в которых латентно-семантическое индексирование проигрывает по эффективности традиционному индексированию. Заслуживает внимания (и возможно, очевидно) то, что латентно-семантическое индексирование унаследовало от векторной модели два ее основных недостатка: отсутствие хорошего способа выражения отрицания (например, найти документ, содержащий термин `german`, но не содержащий термина `shepherd`), а также способа описания логических условий.

Латентно-семантическое индексирование можно рассматривать как *мягкую кластеризацию* (soft clustering), если каждую размерность редуцированного пространства интерпретировать как кластер, а значение, которое документ принимает на координатной оси, — как частичную принадлежность кластеру.

? ***Упражнение 18.11.*** Предположим, что задана совокупность документов, каждый из которых написан либо на английском, либо на испанском языке. Эта коллекция изображена на рис. 18.4.

DocID	Текст документа
1	hello
2	open house
3	mi casa

[2] По сути это означает, что размер коллекции должен быть мал. — *Примеч. ред.*

4	hola Profesor
5	hola y bienvenido
6	hello and welcome

Рис. 18.4. Документы для упражнения 18.11

На рис. 18.5 приведен глоссарий испанских и английских слов. Он *недоступен* информационно-поисковой системе.

Испанский язык	Английский язык
mi	my
casa	house
hola	hello
professor	professor
y	and
bienvenido	welcome

Рис. 18.5. Глоссарий для упражнения 18.11

1. Постройте соответствующую матрицу "термин–документ" C для коллекции, состоящей их указанных документов. Для удобства используйте простую частоту терминов, а не нормализованные веса tf–idf. Убедитесь, что все размерности вашей матрицы ясно размечены.
2. Запишите матрицы U_2, Σ_2' и V_2, а затем постройте аппроксимацию второго ранга C_2.
3. Кратко опишите, что собой представляет элемент (i, j) матрицы C^TC.
4. Кратко опишите, что собой представляет элемент (i, j) матрицы $C_2^TC_2$ и чем он отличается от элемента (i, j) матрицы C^TC.

18.5. Библиография и рекомендации для дальнейшего чтения

Стрэнг (Ctrang, 1986) опубликовал превосходный обзор методов разложения матриц, включая сингулярное разложение. Теорема 18.4 доказана Эккартом и Юнгом (Eckart and Young, 1936). Связь между информационным поиском и малоранговой аппроксимацией матрицы "термин–документ" выявлена Деервестером и др. (Deerwester et al., 1990). Обзор последующих результатов опубликован в работе Берри и др. (Berry et al., 1995). Дюмэ (Dumais, 1993; 1995) описала эксперимент в рамках проекта TREC и показала, что по крайней мере на некоторых тестах латентно-семантическое индексирование позволяет достичь более высокой точности и полноты поиска, чем стандартный поиск в векторном пространстве. Исчерпывающий список научной литературы и программного обеспечения, относящихся к латентно-семантическому индексированию, можно найти на веб-

страницах `www.cs.utk.edu/~berry/lsi++/` и `http://lsi.agreehouse.com/lsi/LSIPapers.html`. Шютце и Сильверстайн (Schьtze and Silverstein, 1997) сравнили латентно-семантическое индексирование и усеченные представления центроидов для повышения эффективности метода *K*-средних (см. раздел 16.4). Баст и Муджумдар (Bast and Mujumdar, 2005) уточнили роль редуцированной размерности k в латентно-семантическом индексировании и показали, как разные пары терминов соединяются друг с другом при разных значениях k. Приложения латентно-семантического индексирования к *межъязыковому информационному поиску* (cross-language information retrieval), в котором индексируются документы на разных языках, и запросу, поданному на одном языке, могут соответствовать документы на разных языках, описываются в работах Берри и Янга (Berry and Young, 1995) и Литтмана (Littman et al., 1998). Латентно-семантическое индексирование (или латентно-семантический анализ) применялось к многочисленным задачам из области компьютерных наук — от моделирования памяти до компьютерного зрения.

Хофман (Hofmann, 1999*a*, 1999*b*) предложил первоначальное вероятностное расширение основных методов латентно-семантического индексирования (*pLSI — probabilistic LSI*). Более формальной основой вероятностной модели латентной переменной для снижения размерности является латентное размещение Дирихле (Latent Dirichlet Allocation — LDA), которое предложено в работе Блея и др. (Blei et al., 2003). Эта порождающая модель приписывает вероятности документам, не принадлежащим обучающему множеству. Она была расширена для иерархической кластеризации в работе Розен-Цви и др. (Rosen-Zvi et al., 2004). Вай и Крофт (Wei and Croft, 2006) провели первое крупномасштабное исследование модели LDA и продемонстрировали, что она значительно превосходит (языковую) модель правдоподобия запроса (раздел 12.2), но уступает модели релевантности, описанной в разделе 12.4 (хотя в модели релевантности, в отличие от модели LDA, предусмотрена предварительная обработка запроса). Те и др. (Teh et al., 2006) обобщили этот подход, предложив иерархические процессы Дирихле (Hierarchical Dirichlet processes) — вероятностную модель, позволяющую извлекать группу (в нашем случае — документ) из бесконечной смеси латентных тем, при этом одной теме может соответствовать много документов.

Глава 19

Основы поиска в вебе

В этой и следующих двух главах рассматриваются поисковые системы веба. Разделы 19.1–19.4 посвящены основам и истории. Эти разделы помогут читателям понять причины, по которым веб так хаотична, быстро меняется и (с точки зрения информационного поиска) сильно отличается от "традиционных" коллекций, которые до сих пор рассматривались в этой книге. Разделы 19.5 и 19.6 посвящены оценке количества документов, проиндексированных поисковыми системами веба, и удалению дубликатов из веб-индексов. Эти разделы служат основой для двух следующих глав.

19.1. Основы и история

Веб является беспрецедентным по многим параметрам: по масштабу, по практически полному отсутствию координации при его создании, а также по разнообразию его участников. Каждая из этих особенностей отличает поиск в вебе от поиска "обычных" документов (и делает его намного сложнее).

Изобретение гипертекста, предсказанного Ванневаром Бушем (Vannevar Bush) в 1940-х годах и впервые реализованного на практике в середине 1970-х годов, предвосхитило процесс формирования Всемирной паутины (World Wide Web, WWW, или веб) в 1990-х годах. Веб рос гигантскими темпами и достиг уровня, когда ее пользователями стала значительная часть всего человечества. При этом сеть основана на простой открытой архитектуре "клиент/сервер": 1) сервер взаимодействует с клиентом посредством простого протокола (*http* — hypertext transfer protocol, протокол передачи гипертекста), с помощью которого в асинхронном режиме передается разнообразная информация (текст, изображения, а со временем — аудио- и видеофайлы), закодированная с помощью простого языка разметки HTML (hypertext markup language); 2) клиент, как правило, *браузер* (browser) — приложение с графическим пользовательским интерфейсом, — может игнорировать то, чего не понимает. Каждое из этих, на первый взгляд, малозначащих свойств, стало важной причиной стремительного роста веба и поэтому заслуживает более внимательного рассмотрения.

Основная операция заключается в следующем: клиент (например, браузер) посылает *http-запрос* (http request) *веб-серверу* (web server). Браузер указывает *унифицированный указатель ресурса* (Universal Resource Locator — URL), например `http://www.stanford.edu/home/atoz/contact.html`. В данном примере URL-строка *http* означает протокол, в соответствии с которым передаются данные. Строка `www.stanford.edu` называется *доменом* (domain) и ссылается на корень иерархии веб-страниц (как правило, отражающей иерархию файловой системы, лежащей в основе веб-сервера). В данном примере строка `/home/atoz/contact.html` представляет собой путь в этой иерархии к файлу `contact.html`, содержащему информацию, которая будет возвращена веб-сервером `www.stanford.edu` в ответ на запрос. Файл `contact.html`, закодированный на языке HTML, содержит гиперссылки и определенную информацию (в дан-

ном случае — контактную информацию Стэнфордского университета), а также правила форматирования для отображения этой информации в браузере. Такой http-запрос позволяет извлечь содержимое страницы, т.е. то, что можно использовать для обхода веба и индексирования документов (глава 20).

Разработчики первых браузеров предусматривали простой способ просмотра исходного кода страницы — разметки HTML. Это предоставляло новым пользователям возможность создавать собственные страницы, размеченные с помощью HTML, без интенсивного обучения или накопления опыта; вместо этого они обучались на примерах, которые им нравились. Второе свойство браузеров, способствовавшее быстрому распространению навыков создания и использования веб-страниц, состояло в том, что браузеры игнорировали информацию, которую не понимали. Тем не менее это не привело, как можно было бы предположить, к появлению многочисленных несовместимых друг с другом диалектов языка HTML. Просто непрофессиональные разработчики веб-страниц могли свободно экспериментировать, не боясь появления синтаксических ошибок, которые “сломают систему”. Публикации веб-страниц стали массовым явлением, в которое оказались вовлеченными не небольшое количество специально обученных программистов, а десятки и сотни миллионов человек. Для большинства пользователей и большинства информационных потребностей веб быстро стала наилучшим средством предоставления и потребления информации о чем угодно: от сведений о редких заболеваниях до расписания движения поездов метрополитена.

Огромная часть информации, размещенной в вебе, совершенно бесполезна до тех пор, пока она не обнаружена и воспринята пользователями. Первые попытки сделать информацию, размещенную в вебе, “обнаруживаемой” разбиваются на две категории: 1) полнотекстовые поисковые системы, такие как Altavista, Excite и Infoseek, и 2) каталоги, систематизирующие веб-страницы по категориям, такие как Yahoo! Первые предлагали интерфейс поиска по ключевым словам и были основаны на инвертированных индексах и механизмах ранжирования, аналогичных описанным ранее в этой книге. Вторые предоставляли пользователю возможность перемещаться по иерархическому дереву тематических меток. Несмотря на удобство и интуитивную понятность этого интерфейса, он обладает серьезными недостатками. Во-первых, точная классификация веб-страниц по узлам таксономии требует ручной работы, что затрудняет масштабирование для веба. На это можно возразить, что нам нужны только качественные страницы в каталоге, чтобы в каждой категории находились только самые лучшие веб-страницы. Однако для того, чтобы выявить такие страницы и аккуратно их классифицировать по категориям, необходимо выполнить много ручной работы. Более того, чтобы эффективно находить веб-страницы, классифицированные по узлам таксономического дерева, представление пользователя о том, какое поддерево следует исследовать в поисках конкретной темы, должно совпадать с представлениями редактора, выполнившего классификацию. При увеличении количества категорий задача быстро становится трудноразрешимой. Таксономическое дерево каталога Yahoo! уже на начальном этапе содержало больше тысячи узлов. Из-за этих трудностей таксономии со временем утратили популярность, хотя возникли варианты этого подхода (например, сайт `About.com` и Open Directory Project), когда эксперты собирают и аннотируют веб-страницы для каждой категории.

Первые поисковые системы веба применяли классические методы поиска, описанные в предыдущих главах нашей книги, сосредоточившись на проблеме масштаба. Они должны были справляться с индексами, содержащими десятки миллионов документов. Размер этих коллекций на несколько порядков превышал размер любой из существовав-

ших на то время систем информационного поиска, ставших всеобщим достоянием. Индексирование, обработка запросов и ранжирование в этих масштабах требовали согласованной работы десятков компьютеров для обеспечения высокой готовности на таком уровне, который до этого времени еще не был реализован ни в одной из систем взаимодействия с клиентами. Первое поколение поисковых систем успешно решало эти проблемы, индексируя значительную часть веба и обрабатывая запросы за доли секунды. Однако качество и релевантность результатов поиска таких систем оставляла желать лучшего, что объяснялось особенностями информации в вебе, которые мы обсудим в разделе 19.2. Для того чтобы решить эту проблему, пришлось разработать новые методы ранжирования и фильтрации спама, гарантировавшие качество результатов поиска. Несмотря на то что классические методы информационного поиска (описанные в нашей книге ранее) остаются актуальными для поиска в вебе, их совершенно недостаточно. Ключевой момент (глава 21) состоит в том, что, в то время как классические методы оценивают релевантность документа по отношению к запросу, остается необходимость оценки *авторитетности* документа по таким признакам, как, например, хост, на котором он расположен.

19.2. Характеристики веба

Основное свойство, которое привело к лавинообразному росту веба (децентрализованная публикация информации практически без контроля авторства), стало главной проблемой поисковых систем при попытках индексировать и предъявить содержание. Пользователи создавали веб-страницы на десятках (естественных) языков и тысячах диалектов. Это требовало множества различных форм стемминга и других лингвистических операций. Поскольку публикация стала доступной десяткам миллионов человек, веб-страницы были чрезвычайно неоднородны по многим параметрам. Создание общедоступных страниц больше не было прерогативой обученных журналистов и редакторов. Несмотря на то, что это свидетельствовало о демократизации процесса распространения информации, это в то же время привело к огромному разнообразию грамматики и стиля (во многих случаях ни грамматику, ни стиль распознать не удается). Действительно, публикации в вебе, по существу, высвободили и лучшие, и худшие стороны традиционного издательского дела в планетарном масштабе, так что страницы запестрели диким разнообразием цветов, шрифтов и структур. Некоторые веб-страницы, включая созданные профессионалами веб-страницы крупных корпораций, состояли исключительно из изображений (которые после щелчка мышью вели к более содержательной текстовой информации), т.е. не были пригодны для индексирования.

Что можно сказать о содержании текстов, размещенных на веб-страницах? Демократизация процесса создания веб-страниц подразумевала новый уровень детализации *мнений* практически по каждому вопросу. Это значило, что веб содержал и правду, и ложь, и противоречия, и предположения в колоссальных масштабах. Возникает вопрос “Каким веб-страницам можно доверять?” В простейшем случае можно сказать, что некоторые издатели заслуживают доверия, а некоторые — нет. Однако остается вопрос “Как поисковая система может присвоить такую степень доверия каждому веб-сайту или веб-странице?”. В главе 21 мы исследуем подходы к решению этого вопроса. Здесь подчеркнем лишь, что не существует универсальной, не зависящей от пользователя понятия доверия. Веб-страница, содержание которой заслуживает доверия с точки зрения одного пользователя, может быть недостоверной в глазах другого. В традиционных (не элек-

тронных) средствах массовой информации эта проблема не возникает: пользователи сами выбирают источники, которым доверяют. Таким образом, один читатель будет считать, что заслуживает доверия газета *The New York Times*, а другой предпочтет газету *The Wall Street Journal*. Однако если поисковая система является единственным средством, с помощью которого пользователь может получать информацию, то проблема доверия становится очень важной.

Несмотря на то что на вопрос "Насколько велик веб?" ответить нелегко (раздел 19.5), вопрос "Сколько веб-страниц находится в индексе поисковой системы?" является более точным, хотя и на него можно ответить по-разному. К концу 1995 года создатели поисковой системы Altavista заявили, что она обошла и проиндексировала примерно 30 миллионов *статических веб-страниц*. Статической называется веб-страница, содержание которой не изменяется от одного запроса к другому. Например, профессор, еженедельно обновляющий свою домашнюю страницу, считается автором статической веб-страницы, а страница, содержащая информацию о состоянии авиарейсов, является динамической. *Динамические страницы* обычно автоматически генерируются сервером приложения в ответ на запрос к базе данных, как показано на рис. 19.1. Признаком такой страницы является то, что адрес URL содержит знак вопроса (символ "?"). Поскольку считалось, что в 1995 году количество статических страниц каждые несколько месяцев удваивалось, первые поисковые системы, такие как Altavista, вынуждены были постоянно наращивать оборудование и увеличивать пропускную способность сети для обхода и индексации веб-страниц.

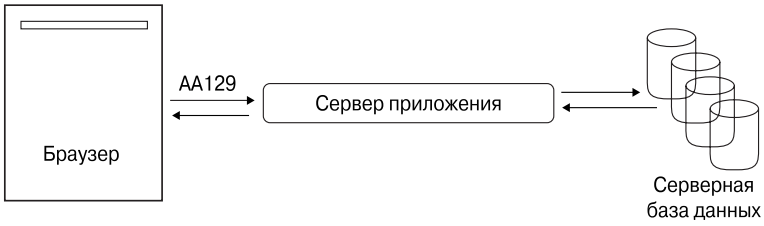


Рис. 19.1. Динамически созданная веб-страница. Браузер отсылает веб-приложению запрос на информацию об авиарейсе AA129. Веб-приложение извлекает информацию из серверной базы данных и создает динамическую веб-страницу, которую возвращает браузеру

19.2.1. Веб-граф

Статическую часть веба, состоящую из статических HTML-страниц и гиперссылок между ними, можно представить в виде направленного графа, в котором каждый узел является веб-страницей, а каждое направленное ребро — гиперссылкой.

На рис. 19.2 показаны два узла, A и B, принадлежащие графу. Каждый узел соответствует некоей веб-странице с гиперссылкой из A на B. Совокупность таких узлов и направленных ребер называется веб-графом. На рис. 19.2 показано также, что (как это обычно бывает на веб-страницах) источник гиперссылки на странице A окружен неким текстом. Этот текст, как правило, инкапсулируется в атрибуте `href` тега `<a>` (от английского *anchor*), кодирующего гиперссылку в HMTL-коде страницы A, и называется *текстом ссылки* (anchor text). Можно предположить, что этот направленный граф не является *сильно связанным* (strongly connected), поскольку существуют такие пары страниц,

что с одной из них невозможно перейти на другую, следуя по гиперссылкам. Гиперссылки, указывающие на страницу, называются *входящими* (in-link), а указывающие вовне — *исходящими* (out-link). Количество входящих ссылок на страницу (так называемая *полустепень захода* (in-degree)) в среднем колеблется от 8 до 15. Аналогично определяется *полустепень исхода* как количество исходящих ссылок. Эти понятия проиллюстрированы на рис. 19.3.

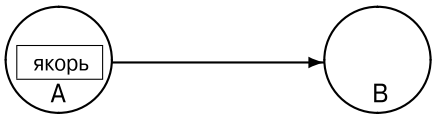


Рис. 19.2. Два узла веб-графа

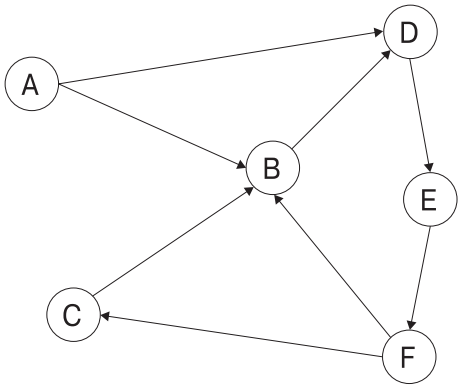


Рис. 19.3. Пример веб-графа: шесть страниц обозначены буквами от A до F. Полустепень захода страницы B равна трем, а полустепень исхода — единице. В этом примере граф не является сильно связанным, поскольку из страниц от B до F нет путей к странице A

Существуют достаточные свидетельства того, что эти ссылки не распределены случайным образом. Распределение количества входящих ссылок на веб-страницу не подчиняется закону Пуассона (Poisson), как можно было бы ожидать, если бы каждая веб-страница выбирала цели своих ссылок равномерно случайно. Скорее, это распределение подчиняется *степенному закону* (power low), согласно которому общее количество веб-страниц с полустепенью захода, равной i, пропорционально $1/i^{\alpha}$. Согласно некоторым исследованиям величина α равна 2,1. [1] Более того, некоторые исследователи утверждают, что направленный граф, соединяющий веб-страницы, имеет форму *галстука-бабочки* (bowtie): существуют три основные категории веб-страниц, которые иногда называют IN, OUT и SCC (strongly connected component). Пользователь может перейти с любой страницы категории IN на любую страницу категории SCC, следуя по гиперссылкам. Аналогично можно перейти со страницы категории SCC на любую страницу категории OUT. И наконец, можно перейти с любой страницы категории SCC на любую другую страницу в этой категории. Однако невозможно перейти со страницы категории SCC ни на одну страницу категории IN, а также со страницы категории OUT ни на одну страницу категории SCC (и, следовательно, категории IN). Примечательно, что в некоторых исследованиях категории IN и OUT оказались примерно равными по размеру, в то время как категория SCC — несколько крупнее. Большинство веб-страниц попадают в одну из трех категорий.

[1] Ср. с законом Ципфа, которому подчиняется распределение слов в тексте (см. главу 5). Этот закон также является степенным с $\alpha = 1$.

Остальные страницы образуют *туннели* (tubes), небольшие множества страниц вне категории SCC, ведущие непосредственно со страниц категории IN на страницы категории OUT, а также *усики* (tendrils), которые не ведут со страницы категории IN никуда или приходят ниоткуда на страницу категории OUT. Эта структура веба показана на рис. 19.4.

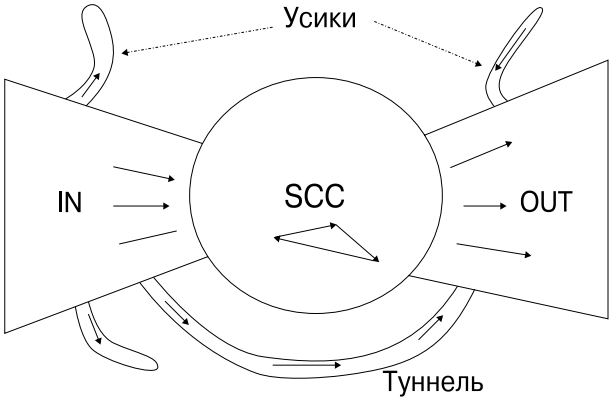


Рис. 19.4. Структура веба в виде галстука-бабочки. В данном случае продемонстрированы один туннель и три усика

19.2.2. Спам

В самом начале истории веб-поиска стало ясно, что поисковые системы являются важным каналом доставки рекламы потенциальным покупателям. Пользователь, задавший запрос `maui golf real estate`, ищет не просто новости или развлекательную информацию об учебных курсах по гольфу на острове Мауи, а, возможно, намеревается купить там недвижимость. Следовательно, продавцы такой недвижимости и их агенты сильно заинтересованы в создании веб-страниц, которые высоко ранжировались бы по этому запросу. В поисковой системе, в которой ранжирование основано на подсчете частоты термина, веб-страницы с большим количеством повторений слов `maui golf real estate` будут иметь более высокий ранг. Это привело к появлению первого поколения *спама*, который (в контексте веб-поиска) заключается в манипулировании содержанием веб-страниц с целью получения высоких позиций в результатах поиска для выбранных ключевых слов. Для того чтобы не вызывать у пользователя раздражения из-за повторяющихся терминов, более опытные *спамеры* прибегали к таким трюкам, как написание повторяющихся терминов цветом фона. Несмотря на то что эти слова остаются невидимыми для пользователей, индексаторы поисковых систем извлекают их из HTML-представления веб-страницы и индексируют так, будто они на самом деле видны на странице.

Причиной спама является неоднородность мотивов создания веб-страниц. В частности, многие создатели веб-страниц руководствуются коммерческими интересами и, следовательно, стремятся достичь выгоды за счет манипулирования результатами поиска. На это можно возразить, что спам ничем не отличается от ситуации, когда компании печатают номера своих телефонов крупным шрифтом на “желтых страницах” рекламных справочников. Однако это стоит компании бо́льших затрат и, следовательно, является более честным способом рекламирования. Возможно, более удачная аналогия — исполь-

зование в качестве названия компаний длинных строк, начинающихся с нескольких букв "А", чтобы попасть на первые места списка в соответствующей категории рекламного справочника. Фактически структура рекламных справочников, в которых компании, желающие выделиться за счет более крупного или более темного шрифта, должны заплатить больше, была воспроизведена и в веб-поиске. Многие поисковые системы допускают возможность дополнительной оплаты за включение веб-страницы в свой индекс.[2] Эта модель называется *платным включением* (paid inclusion). Разные поисковые системы придерживаются разных правил, касающихся оплаченного включения и влияния этой оплаты на ранжирование результатов.[3]

Вскоре поисковые системы стали достаточно сложными, чтобы распознавать спам по большому количеству повторений конкретных ключевых слов. Спаммеры ответили новыми трюками, самые известные из которых мы сейчас опишем. Первый из этих приемов называется *клоакингом* (cloaking) и проиллюстрирован на рис. 19.5. Здесь веб-сервер спамера возвращает разные страницы в зависимости от того, откуда поступил http-запрос — от робота поисковой системы (робот — это часть поисковой системы, собирающая веб-страницы; он будет описан в главе 20) или от браузера индивидуального пользователя. В первом случае веб-страница будет проиндексирована поисковой системой по ключевым словам, вводящим пользователя в заблуждение. Если пользователь будет искать эти ключевые слова и выберет для просмотра эту страницу, он получит веб-страницу с совершенно другим содержанием, которое отличается от содержания, проиндексированного поисковой системой. Такой обман поисковых индексаторов неизвестен в традиционных системах информационного поиска. Он появился из-за того, что авторы страниц и поисковые системы преследуют разные цели.

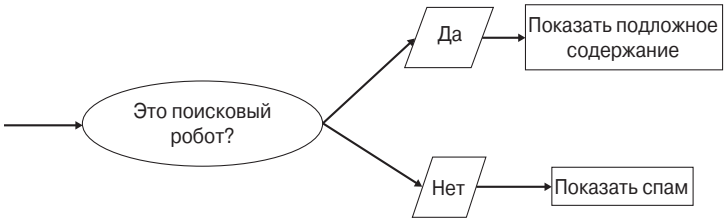


Рис. 19.5. Клоакинг

Дорвей (doorway page) содержит текст и метаданные, тщательно выбранные так, чтобы обеспечить высокий ранг по отношению к ключевым словам. Если браузер обращается к дорвею, его запрос переадресовывается на страницу, содержащую коммерческую информацию. Более сложные технологии спама манипулируют метаданными, включая ссылки на страницы (по причинам, которые будут рассмотрены в главе 21). Поскольку спам по своей природе носит коммерческий характер, вокруг него сложилась индустрия *поисковых оптимизаторов* (Search Engine Optimizers — SEO), оказывающих консультационные услуги клиентам, желающим поднять ранг своих веб-страниц по выбранным

[2] Наоборот, так делают весьма немногие поисковые системы. Например, ни Google, ни Bing, ни Яндекс этого не делают. — *Примеч. ред.*

[3] Те немногие поисковые системы, которые имеют такие программы, опираются на принцип невлияния платного включения на ранжирование (Yahoo) либо помечают такие результаты для пользователя (Baidu). — *Примеч. ред.*

ключевым словам. Поисковые системы неодобрительно относятся к этому виду деятельности по расшифровке и адаптации к их алгоритмам ранжирования, а также публикуют правила, касающиеся приемов поисковой оптимизации, которые считают недопустимыми (известны случаи, когда поисковые системы блокировали запросы, поступавшие от некоторых оптимизаторов, нарушающих установленные правила[4]). В конце концов, противостояние между поисковыми оптимизаторами (которые постепенно догадываются о методах ранжирования, реализованных поисковыми системами) и создателями поисковых систем (которые в ответ пытаются усовершенствовать свои методы) может продолжаться бесконечно. В ходе этой битвы даже появилось новое направление исследований — *информационный поиск в условиях противодействия* (adversarial information retrieval). Для того чтобы победить спамеров, манипулирующих текстами своих веб-страниц, необходимо использовать ссылочную структуру веба — метод, известный как *анализ ссылок* (link analysis). Первой поисковой системой, применившей крупномасштабный анализ ссылок (подробности приведены в главе 21), стал Google, хотя все поисковые системы в настоящее время используют этот метод (и, соответственно, спамеры предпринимают серьезные попытки противодействовать ему с помощью *ссылочного спама* (link spam)).

? ***Упражнение 19.1.*** Какова вероятность того, что случайно выбранная веб-страница будет иметь полустепень захода, равную единице, если количество веб-страниц с полустепенью захода i пропорционально $1/i^{2,1}$?

Упражнение 19.2. Чему равна средняя полустепень захода веб-страницы, если количество веб-страниц с полустепенью захода i пропорционально $1/i^{2,1}$?

Упражнение 19.3. Уменьшится, увеличится или останется неизменной доля страниц с внутренней степенью i при устремлении максимальной полустепени захода к бесконечности, если количество веб-страниц с полустепенью захода i пропорционально $1/i^{2,1}$? Как изменится ответ для показателей степени, отличных от 2,1?

Упражнение 19.4. Средняя полустепень захода всех узлов в веб-графе равна 9. Что можно сказать о средней полустепени исхода всех узлов в этом графе?

19.3. Реклама как экономическая модель

В первые годы существования веба компании размещали графическую баннерную рекламу на популярных веб-сайтах (на сайтах новостей и развлечений, таких как MSN, America Online, Yahoo! и CNN). Основной целью этих рекламных объявлений была *имиджевая реклама* (branding), т.е. стремление сформировать у пользователя, просматривающего веб-страницу, позитивное отношение к бренду компании, разместившей рекламу. Обычно эти рекламные объявления оплачивались, исходя из *цены за тысячу показов* (Cost Per Mil — CPM). Некоторые веб-сайты заключали контракты со своими рекламодателями, в которых рекламные объявления оценивалась не по количеству показов (или *тиражу* (impressions)), а по количеству *кликов* (clicked on) пользователей на объявлении. Эта модель ценообразования называется *ценой за клик* (Cost Per Click — CPC). В таких случаях клик (щелчок кнопкой мыши) на рекламном объявлении, сделанный

[4] Скорее всего, речь идет о блокировке большого потока автоматизированных запросов, а профессиональная ориентация их отправителей и их принадлежность к той или иной индустрии не имеют значения. — *Примеч. ред.*

пользователем, приводил его на веб-сайт рекламодателя, на котором пользователь мог сделать покупку. Цель такой рекламы заключается не в продвижении торговой марки, а в побуждении к покупке. Это различие между имиджевой рекламой и рекламой, ориентированной на совершение покупки, было устоявшимся в традиционных средствах массовой информации, таких как радио и печатные СМИ. Интерактивность веба сделала возможным автоматический биллинг в модели CPC: веб-сайт регистрирует клики пользователей, с тем чтобы выставить счет рекламодателю.

Первопроходцем в этом направлении стала компания Goto, изменившая свое название на Overture перед поглощением компанией Yahoo!. Система Goto не была поисковой системой в традиционном смысле: для каждого термина запроса q она принимала *заявки* (bids) от компаний, желающих показать свои веб-страницы в ответ на запрос q. В ответ на запрос q система Goto возвращала страницы всех рекламодателей, приславших заявки на запрос q, расположенные в порядке убывания предложенной цены. Далее, когда пользователь кликал на одном из возвращенных результатов, соответствующий рекламодатель должен был осуществить платеж компании Goto (в первоначальной реализации этот платеж был равен сумме заявки рекламодателя на запрос q).

Некоторые аспекты модели Goto заслуживают внимания. Во-первых, пользователь, вводя запрос q в интерфейсе Goto, выражал свой активный интерес и намерения, связанные с запросом q. Например, если пользователь набирал запрос `golf clubs` (клюшки для гольфа), то, более вероятно, что он собирается купить набор клюшек, а не просто ищет новости о гольфе. Во-вторых, компания Goto получала деньги, только если пользователь действительно проявлял интерес к рекламе, о чем свидетельствовал клик на рекламном объявлении. Эти факторы, вместе взятые, легли в основу мощного рекламного механизма. В результате годовой доход компании Goto/Overture быстро достиг сотен миллионов долларов. Этот стиль поиска называется *оплаченным поиском* (sponsored search) или *поисковой рекламой* (search advertising).

После появления двух основных типов поисковых систем (“чисто” поисковых систем, таких как Google и Altavista, с одной стороны, и систем оплаченного поиска — с другой) следующим логичным шагом стало их объединение. В настоящее время поисковые системы работают по этой же схеме: в качестве основного ответа на запрос пользователя они предоставляют результаты обычного *алгоритмического поиска* (algorithmic search), которые сопровождаются поисковой рекламой справа (сбоку, сверху или снизу) от основных результатов. Эта схема показана на рис. 19.6. Нахождение результатов оплаченного поиска и их ранжирование по отношению к запросу в настоящее время стали более изощренными, чем простая схема Goto. В этом процессе сочетаются идеи информационного поиска и микроэкономики, которые выходят за рамки нашей книги. Необходимость понимания того, как системы поиска ранжируют рекламные объявления и как планировать бюджет маркетинговых кампаний с учетом разных ключевых слов и рекламных площадок, привела к появлению новой области деятельности — *поискового маркетинга* (Search Engine Marketing — SEM).

Экономические мотивы, лежащие в основе поисковой рекламы, породили попытки некоторых участников использовать систему в своих интересах. Эти попытки принимают разные формы, например *кликовый спам* (click spam). В настоящее время не существует общепринятого определения этого явления. Как следует из его названия, оно подразумевает, что клики на рекламных объявлениях выполняет недобросовестный пользователь. Например, злонамеренный рекламодатель может попытаться исчерпать рекламный бюджет конкурента, постоянно кликая (с помощью специального “кликающего робота”)

на его рекламном объявлении. Поисковые системы столкнулись с необходимостью распознавать кликовый спам, чтобы не взимать деньги за такие клики с рекламодателей.

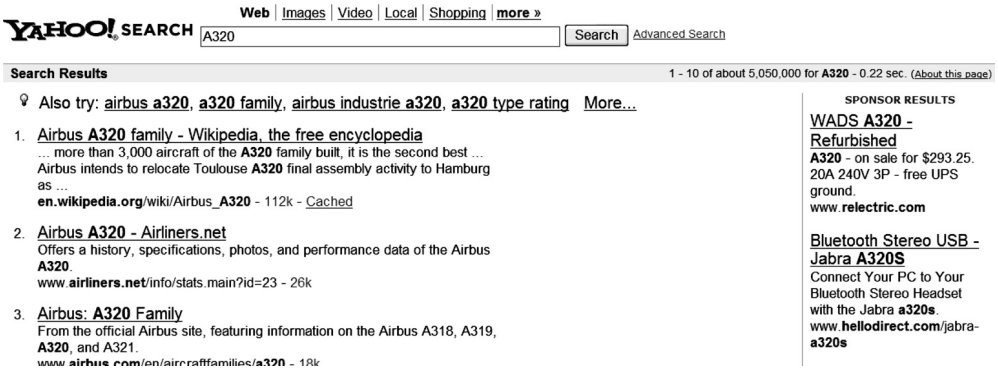


Рис. 19.6. Поисковая реклама, выданная в ответ на ключевые слова запроса. Здесь в ответ на запрос `A320` пользователь получил результаты алгоритмического поиска о самолете Airbus вместе с рекламой разных товаров, не связанных с самолетами, но имеющих A320 в названии и предлагаемых рекламодателями на рынке. Отсутствие рекламы самолета отражает тот факт, что мало кто из маркетологов пытается продавать самолет A320 через Интернет

? ***Упражнение 19.5.*** Метод Goto ранжировал рекламу, соответствующую запросу, по *заявкам*: рекламодатель, составивший наибольшую заявку, находился в начале списка, а остальные следовали за ним в порядке убывания. Что плохого в том, что рекламодатель, сделавший самую большую заявку, разместил рекламу, нерелевантную запросу. Почему рекламодатель с нерелевантным рекламным объявлением может стремиться попасть на первые места в списке?

Упражнение 19.6. Допустим, что, кроме предложений, у нас есть показатель CTR (click-through rate) для каждого рекламодателя, т.е. мы накапливаем отношение количества кликов пользователей к общему количеству показов рекламных объявлений этого рекламодателя. Предложите модификацию схемы Goto с использованием этих данных, позволяющую избежать проблемы, описанной в упражнении 19.5.

19.4. Опыт пользователей поисковых систем

Очень важно также понимать потребности пользователей поисковых систем. Это еще одно существенное отличие от традиционного информационного поиска, в ходе которого пользователи обычно обращались к поиску в рамках профессиональной деятельности, имели некоторый опыт составления запросов к тщательно организованным коллекциям, стиль и структуру которых они хорошо понимают. В противоположность этому пользователи поисковых систем не знают (и часто не хотят знать) о неоднородности содержания веба, синтаксисе языков запросов и искусстве составления запросов. Действительно, популярный инструмент (а поиск в вебе таким уже стал) не должен предъявлять слишком высоких требований к миллиардам людей. В результате многочисленных исследований выяснилось, что среднее количество ключевых слов в типичном запросе к поисковой системе колеблется от двух до трех. Синтаксические операторы (булевы связки, джокеры и т.д.)

используются редко. Это также является результатом того, что поиск в вебе ведут "нормальные" люди, а не ученые в области информатики.

Очевидно, что чем выше трафик поисковой системы, тем больше прибыли приносит поисковая реклама. Как же происходит дифференциация поисковых систем и что обеспечивает рост их трафика? Компания Google сформулировала два принципа, помогающих ей выигрывать конкурентную борьбу: 1) акцент на релевантности, особенно на точности, а не на полноте первых результатов, и 2) облегчение восприятия пользователями, т.е. упрощение страницы для задания запроса и страницы результатов поиска за счет использования преимущественно текстовой, а не графической информации. Благодаря первому принципу пользователи просто экономят время при поиске нужной информации. Второй принцип привлекает пользователей быстрой реакцией — им не приходится долго ждать, пока загрузятся страница со строкой запроса и страница результатов.[5]

19.4.1. Потребности пользователей

Обычные запросы на поиск в вебе можно разделить на три категории: 1) информационные, 2) навигационные и 3) транзакционные. Совершенно очевидно, что одни запросы попадают более чем в одну категорию, а другие не попадают ни в одну.

Информационные запросы (informational queries) имеют цель найти общую информацию по широкой теме, например о лейкемии или Провансе. Обычно не вся искомая информация содержится на одной веб-странице; пользователи, задающие информационные запросы, обычно пытаются использовать информацию нескольких веб-страниц.

Навигационные запросы (navigational queries) отражают желание найти конкретный веб-сайт или домашнюю страницу, например `Lufthansa airlines`. В таких случаях пользователь ожидает, что среди первых результатов должна быть домашняя страница компании Lufthansa. Пользователя не интересует огромное множество документов, содержащих термин `Lufthansa`. Для такого пользователя наилучшей мерой удовлетворения является точность на уровне один (precision at 1).

Транзакционные запросы (transactional query) предшествуют совершению транзакций в вебе, например покупке товара, загрузке файла или бронированию номера в гостинице. В таких случаях поисковая система должна возвращать в качестве результатов список сервисов, предоставляющих интерфейс для выполнения таких транзакций.

Распознать, к какой категории относится тот или иной запрос, иногда затруднительно. Категория запроса может влиять не только на результаты алгоритмического поиска, но и на пригодность запроса для показа рекламных объявлений (так как запрос может обнаружить намерение сделать покупку). Для навигационных запросов можно было бы потребовать, чтобы поисковая система возвращала только один результат или даже перенаправляла пользователя на требуемую веб-страницу. Так исторически сложилось, что поисковые системы постоянно соревнуются между собой, стремясь проиндексировать как можно больше страниц. Должно ли это интересовать пользователя? Вероятно, нет, но средства массовой информации приводят оценки (часто статистически недостоверные) размеров разных поисковых систем. Такие отчеты влияют на пользователей, и поисковые системы стараются следить за размерами своих индексов и сравнивать их с кон-

[5] Авторы здесь выступают скорее в роли авторов рекламного буклета, а не учебника по поиску. Такие же или похожие принципы и до, и после Google сформулировали для себя и другие поисковые системы. — *Примеч. ред.*

курентами. В случае информационных (и в меньшей степени — транзакционных) запросов пользователя интересует охват индекса поисковой системы.

На рис. 19.7 показаны составные части поисковой системы, включая поисковый робот, а также веб-страницу и рекламные индексы. Часть рисунка, отделенная извилистой пунктирной линией, иллюстрирует внутреннее устройство поисковой системы.

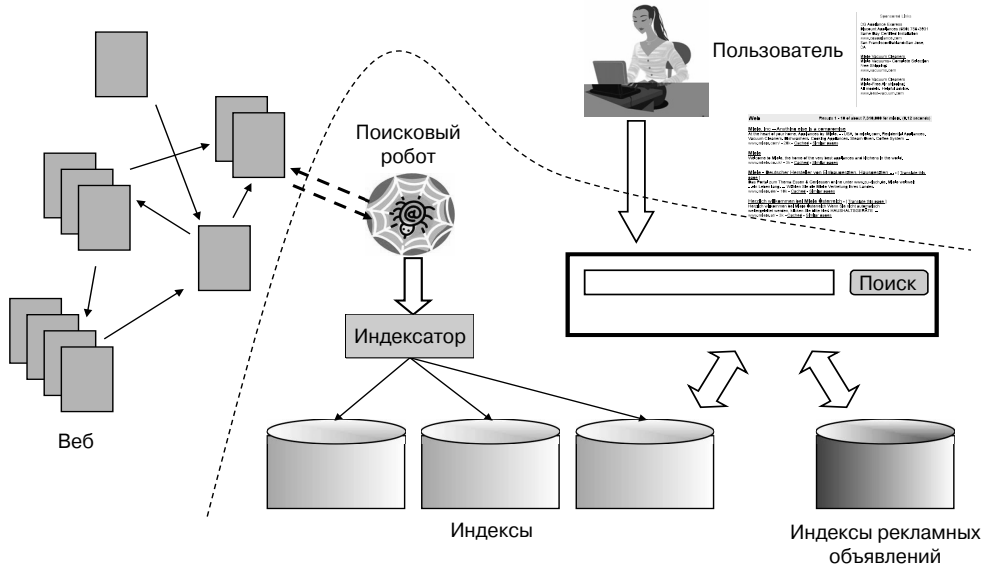


Рис. 19.7. Компоненты поисковой системы

19.5. Размер индекса и оценка его размера

На первый взгляд, при увеличении размера индекса охват информации должен расти, хотя это зависит от того, какие именно веб-страницы индексирует поисковая система: одни страницы являются более информативными, чем другие. Кроме того, трудно судить о доле веба, проиндексированной поисковой системой, поскольку в сети существует огромное количество динамических веб-страниц. Например, обращение к странице `http://www.yahoo.com/any_string` возвращает корректную HTML-страницу, а не ошибку, вежливо информируя пользователя о том, что такой страницы на сайте Yahoo! не существует. Такая “мягкая ошибка 404” представляет собой только один пример из многих, когда веб-серверы генерируют бесконечное количество корректных веб-страниц. Хотя некоторые из этих страниц являются злонамеренными ловушками, вынуждающими поисковый робот (компонент, систематически загружающий веб-страницы для индексации; см. главу 20) застревать на спамерском веб-сайте и индексировать его многочисленные страницы.

Мы можем задать более корректный вопрос: “Как соотносятся размеры индексов двух поисковых систем?” Но даже этот вопрос является неточным по следующим причинам.

1. В ответ на запросы поисковая система может вернуть веб-страницы, содержание которых не было (ни полностью, ни частично) проиндексировано. Поисковые системы, как правило, индексируют только несколько первых тысяч слов на веб-

странице. В некоторых случаях поисковая система знает о том, что страница *p связана* с проиндексированными страницами, но при этом не индексирует саму страницу *p*. Как будет показано в главе 21, в таком случае страница *p* может попасть в список результатов поиска.

2. Поисковые системы обычно разделяют свои индексы на несколько ярусов и сегментов, и не все они проверяются при каждом поиске (ярусные индексы описаны в разделе 7.2.1). Например, веб-страница, расположенная глубоко внутри веб-сайта, может быть проиндексирована, но не найдена в ходе обычного поиска. Однако, возможно, она будет возвращена в результате поиска с ограничением по конкретному сайту (этот вид поиска предлагается большинством поисковых систем).

Таким образом, индексы поисковых систем включают в себя различные классы проиндексированных страниц, и единого показателя, отражающего размер индекса, не существует. Тем не менее было предложено множество методов для грубой оценки отношения размеров индексов поисковых систем E_1 и E_2. Основная гипотеза, лежащая в их основе, заключается в том, что каждая поисковая система индексирует часть веба, выбранную независимо и случайным образом. При этом делаются довольно ненадежные предположения: 1) веб состоит из конечного количества страниц, из которых поисковая система выбирает подмножество; 2) каждая поисковая система выбирает свое подмножество независимо от другой и случайным образом. Как будет показано при обсуждении поисковых роботов в главе 20, эти предположения далеки от реальности. Однако, отталкиваясь от них, можно применить классический метод оценки, известный как *метод повторного захвата* (capture-recapture method).[6]

Предположим, что мы можем выбрать случайную страницу из индекса поисковой системы E_1 и проверить, принадлежит ли она индексу поисковой системы E_2, и наоборот. В результате этих экспериментов можно оценить долю x страниц из индекса поисковой системы E_1 в индексе поисковой системы E_2 и долю y страниц из индекса поисковой системы E_2 в индексе поисковой системы E_1. Обозначим через $|E_i|$ размер индекса поисковой системы E_i. Тогда

$$x|E_1| \approx y|E_2|.$$

Отсюда следует, что

$$\frac{|E_1|}{|E_2|} \approx \frac{y}{x}. \tag{19.1}$$

Если наши предположения о E_1 и E_2 как о независимых и случайных подмножествах веба были бы верны, а наш метод выбора страниц не был смещен, то оценка (19.1) давала бы несмещенную оценку отношения $|E_1|/|E_2|$. Здесь необходимо различать два сценария. Эта оценка получена либо кем-то, кто имеет доступ к индексу одной из поисковых систем (например, сотрудником компании E_1), либо независимым экспертом, не имеющим доступа к индексам поисковых систем. В первом случае мы можем просто случайным образом извлекать документы из индекса. Второй вариант сложнее: мы должны извлекать случайную страницу из индекса одной поисковой системы, *находясь вне этой системы*, а затем проверять, принадлежит ли эта страница индексу другой поисковой системы.

---

[6] Метод статистической биологии используется для оценки размеров популяции животных. — *Примеч. ред.*

Для того чтобы реализовать фазу выбора страницы, можно выбрать случайную страницу из всего (идеализированной, конечной) веба, а затем проверить, принадлежит ли она индексу каждой из поисковых систем E_1 и E_2. К сожалению, обеспечить случайность и равновероятность извлечения страницы из веба довольно трудно. Мы кратко опишем попытки решить эту проблему, указав на недостатки, присущие каждому подходу, а затем подробно опишем метод, на котором основано большинство подобных исследований.

1. *Случайные поиски* (random searches). Начнем с лога запросов веб-поиска; отправим случайный запрос из этого лога поисковой системе E_1 и возьмем случайную страницу из результатов поиска. Поскольку такие логи, как правило, недоступны извне поисковой системы, можно использовать захват всех поисковых запросов, исходящих от членов рабочей группы (например, ученых исследовательского центра), согласившихся на регистрацию всех их запросов. Этот подход имеет массу недостатков, в частности существует смещение типов поиска, выполненного рабочей группой. Кроме того, случайный документ из списка результатов E_1 — это не случайный документ индекса системы E_1.
2. *Случайные IP-адреса* (random IP-addresses). В рамках второго подхода генерируется случайный IP-адрес, отправляется запрос к веб-серверу, находящемуся по этому случайному адресу, а затем собираются все страницы этого сервера. Смещение такого подхода заключается в том, что многие хосты могут иметь один и тот же IP-адрес (эта практика называется виртуальным хостингом) или не принимать http-запросы от хоста, на котором проводится эксперимент. Более того, при таком подходе можно более вероятно попасть на многочисленные сайты с небольшим количеством страниц, что приводит к искажению распределения документов. Ситуацию можно исправить, если знать распределение количества страниц на веб-сайтах.
3. *Случайное блуждание* (random walks). Если бы веб-граф был сильно связанным направленным графом, то мы могли бы выполнить случайное блуждание, начав с произвольной веб-страницы. Это блуждание могло бы сойтись к стационарному распределению (см. раздел 21.2.1), из которого можно было бы извлечь веб-страницу с фиксированной вероятностью. Этот метод также имеет много недостатков. Во-первых, веб-граф не является сильно связанным, поэтому даже несмотря на корректирующие правила невозможно гарантировать сходимость такого блуждания к стационарному распределению при старте с произвольной страницы. Во-вторых, время, которое понадобится, чтобы случайное блуждание пришло в стационарное состояние, неизвестно и может превысить продолжительность эксперимента.

Очевидно, что каждый из этих подходов далек от идеала. Опишем четвертый вариант выбора страниц — *случайные запросы* (random queries). Этот подход заслуживает внимания по двум причинам: он был успешно реализован в серии постоянно уточняющихся оценок, и, наоборот, этот подход часто неверно интерпретируют и неаккуратно реализуют, что ведет к некорректным оценкам. Идея заключается в том, чтобы извлечь (почти) случайную страницу из индекса поисковой системы, отослав ей случайный запрос. Очевидно, что извлечение случайных терминов, например, из словаря *Webster's Dictionary* — не лучшая реализация этой идеи. С одной стороны, не все словарные термины встречаются одинаково часто, поэтому данный подход не приведет к извлечению документов из поисковой системы с одинаковой вероятностью. С другой стороны, в веб-документах есть

много терминов, которые не встречаются в стандартных словарях, таких как *Webster's Dictionary*. Для того чтобы решить проблему с терминами, не принадлежащими стандартным словарям, начнем с накопления выборочного веб-словаря. Для этого можно обойти ограниченную область веба или какое-либо репрезентативное подмножество страниц, составленное вручную, например сайт Yahoo! (именно так и поступали в первых экспериментах с использованием этого метода). Рассмотрим конъюнктивный (AND) запрос, содержащий несколько слов, случайно выбранных из данного словаря.

Выполним следующие действия. Отошлем случайный конъюнктивный запрос поисковой системе E_1 и случайным образом извлечем из первых ста результатов страницу p. Проверим наличие страницы p в индексе поисковой системы E_2: выберем на странице p от шести до восьми терминов с низкой частотой и составим из них конъюнктивный запрос для поисковой системы E_2. Полученную оценку можно улучшить, повторив эксперимент много раз. Как процесс извлечения страницы, так и процесс тестирования порождают ряд проблем.

1. Наша выборка смещена в сторону более длинных документов.
2. Извлечение страницы из первых ста результатов E_1 порождает смещение из-за алгоритма ранжирования E_1. Извлечение страницы из всех результатов E_1 замедляет эксперимент. Это обстоятельство объясняется тем, что большинство поисковых систем защищаются от интенсивных автоматических запросов.
3. На этапе проверки возникают дополнительные проблемы. Например, поисковая система E_2 может неправильно обрабатывать конъюнктивные запросы, состоящие из восьми слов.
4. Как система E_1, так и система E_2 могут отклонять тестовые запросы, рассматривая их как спам, а не как запросы пользователей-людей.
5. Могут возникать технические проблемы, например разрыв соединения из-за истечения времени соединения.

Для того чтобы устранить некоторые из этих препятствий, был проведен ряд исследований, однако идеальное решение найдено не было, хотя уровень статистических вычислений, направленных на уменьшение смещений, заметно повысился. Основная идея заключается в том, чтобы бороться со смещениями, оценивая их для каждого документа. Зная их величину, можно применять стандартные статистические методы и генерировать несмещенные выборки. Современные исследователи предлагают оказаться от использования конъюнктивных запросов на этапе проверки и применить фразовые и другие запросы, которые ведут себя лучше. Кроме того, в последних экспериментах используются и другие методы выбора, кроме случайных запросов. Наибольшую популярность приобрел метод *выбора документов в ходе случайного блуждания* (document random walk sampling), в котором документ выбирается в процессе случайного блуждания по виртуальному графу, построенному по документам. В этом графе узлами являются документы; два документа соединяются ребром, если содержат два или более одинаковых слов. Этот граф никогда не строится. Случайное блуждание от одного документа к другому осуществляется с помощью выбора пары ключевых слов из документа d, задания запроса поисковой системе и выбора случайного документа из полученных результатов. Детали этого процесса описаны работах, ссылки на которые приведены в разделе 19.7.

? ***Упражнение 19.7.*** Две поисковые системы, A и B, равномерно генерируют случайные страницы из своих индексов. Тридцать процентов страниц из индекса A принадлежат индексу B, и пятьдесят процентов страниц из индекса B встречаются в индексе A. Каково отношение количества страниц в индексе A к количеству страниц в индексе B?

19.6. Нечеткие дубликаты и алгоритм шинглов

Обсуждая размеры индексов в разделе 19.5, мы проигнорировали один аспект — *дубликаты* (duplicates). В вебе содержится много копий одного и того же контента. По некоторым оценкам около 40% страниц в вебе являются дубликатами других страниц. Многие из этих копий вполне легитимны; например, некоторые хранилища информации имеют зеркала для повышения надежности и скорости доступа за счет избыточности. Поисковые системы пытаются избежать многократного индексирования одного контента, чтобы сэкономить ресурсы на хранение и обработку данных.

Самый простой способ нахождения дубликатов — вычислить *контрольную сумму*[7] (fingerprint) (скажем, длиной 64 бит) последовательности символов каждой страницы. Тогда, если контрольные суммы двух веб-страниц равны, следует проверить, совпадают ли сами страницы, и выявить дубликаты. Однако этот подход не может справиться с явлением, получившим в вебе широкое распространение, — *полудубликатами* (near duplicates). Во многих случаях содержание одной веб-страницы идентично содержанию другой, за исключением нескольких символов, скажем, даты и времени последнего изменения страницы. Даже в таких случаях мы хотели бы рассматривать эти две страницы как дубликаты, чтобы индексировать только одну копию. Как найти и отфильтровать полудубликаты, не прибегая к полному перебору всех пар веб-страниц, что с учетом миллиардов страниц просто невозможно?

Опишем решение проблемы выявления приблизительных дубликатов в вебе. Ответом является метод, названный *шинглированием* (shingling) (рис. 19.8). При заданных положительном значении k и последовательности терминов в документе d будем называть k-шинглами документа d множество всех непрерывных последовательностей, состоящих из k терминов документа d. Рассмотрим в качестве примера следующий текст: `a rose is a rose is a rose`. Для данного текста 4-шинглами ($k = 4$ — типичное значение, используемое для выявления приблизительных дубликатов веб-страниц) являются `a rose is a`, `rose is a rose` и `is a rose is`. Первые два шингла встречаются в тексте дважды. Интуитивно ясно, что два документа являются приблизительными дубликатами, если множества порождаемых ими шинглов почти совпадают. Уточним это интуитивное представление и разработаем метод эффективного вычисления и сравнения множеств шинглов для всех веб-страниц.

Обозначим через $S(d_j)$ множество шинглов документа d_j. Напомним, что коэффициент Жаккара (Jaccard coefficient), измеряющий степень перекрытия множеств $S(d_1)$ и $S(d_2)$, равен $J(S(d_1), S(d_2)) = |S(d_1) \cap S(d_2)|/|S(d_1) \cup S(d_2)|$. Проверка приблизительных дубликатов между документами d_1 и d_2 сводится к вычислению коэффициента Жаккара. Если он превышает установленное пороговое значение (скажем, 0,9), мы объявляем их приблизительными дубликатами и исключаем один из них из процесса индексирования. Однако это не приводит к упрощению: коэффициенты Жаккара необходимо вычислить для всех пар.

[7] Иногда говорят “дактилограмму”. — *Примеч. ред.*

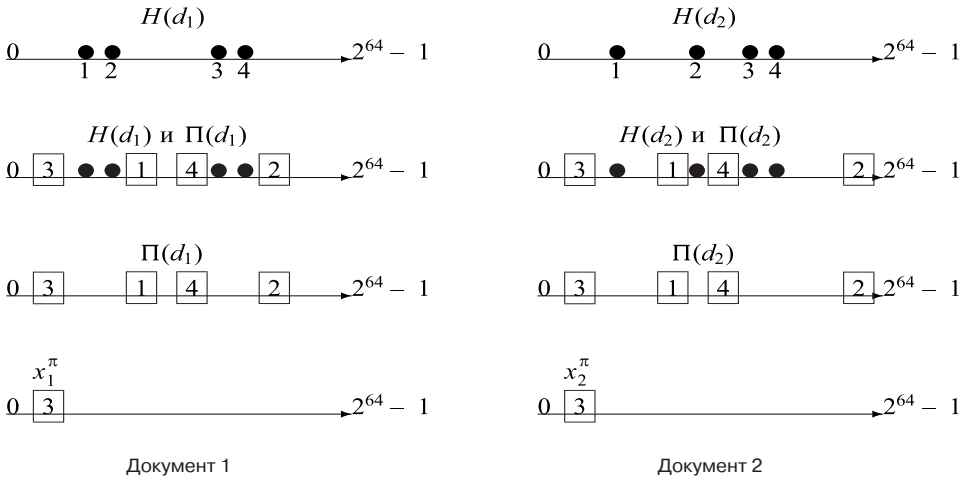


Рис. 19.8. Иллюстрация алгоритма шинглов. Два документа проходят четыре этапа шинглирования. На первом этапе (верхняя строка) к каждому шинглу из каждого документа применяется хеширование и вычисляются множества $H(d_1)$ и $H(d_2)$ (кружочки). Затем с помощью случайной перестановки величин из множеств $H(d_1)$ и $H(d_2)$ вычисляются множества $\Pi(d_1)$ и $\Pi(d_2)$ (квадратики). В третьей строке показаны только множества $\Pi(d_1)$ и $\Pi(d_2)$, а в нижней — минимальные значения x_1^π и x_2^π

Для того чтобы избежать этого, применим хеширование. Отобразим каждый шингл в хеш-значение в большом пространстве, например 64 бит. Пусть $H(d_j)$ образуют соответствующее множество 64-битовых хеш-значений, полученных из множества $S(d_j)$ ($j = 1, 2$). Теперь для того, чтобы выявить пары документов, множества H которых сильно перекрываются, применим следующий прием. Пусть π — случайная перестановка, преобразовывающая 64-битовое целое число в другое 64-битовое целое число. Обозначим через $\Pi(d_j)$ множество переставленных хеш-значений множества $H(d_j)$, Таким образом, для каждого $h \in H(d_j)$ существует соответствующее число $\pi(h) \in \Pi(d_j)$.

Пусть x_j^π — наименьшее целое число в множестве $\Pi(d_j)$. Справедлива следующая теорема.

Теорема 19.1

$$J(S(d_1), S(d_2)) = P(x_1^\pi = x_2^\pi)$$

Доказательство. Приведем доказательство при немного более общих условиях. Рассмотрим семейство множеств, элементы которых извлечены из общей генеральной совокупности. Будем рассматривать эти множества как столбцы матрицы A, в которых каждому элементу генеральной совокупности соответствует отдельная строка. Элемент a_{ij} равен единице, если элемент i принадлежит множеству S_j, которому соответствует j-й столбец.

Пусть Π — случайная перестановка строк матрицы A. Обозначим через $\Pi(S_j)$ столбец, получающийся в результате применения перестановки Π к j-му столбцу. В заключение пусть x_j^π — индекс первой строки, в которой столбец $\Pi(S_j)$ содержит единицу. Докажем, что для любых столбцов j_1 и j_2 выполняется равенство

$$P\left(x_{j_1}^\pi = x_{j_2}^\pi\right) = J(S_{j_1}, S_{j_2}).$$

Теорема следует из этого равенства.

Рассмотрим столбцы j_1 и j_2, показанные на рис. 1919.9. Упорядоченные пары элементов множеств $S(j_1)$ и $S(j_2)$ разделяют строки на четыре типа: нули в обоих столбцах, нуль в столбце $S(j_1)$ и единица в столбце $S(j_2)$, единица в столбце $S(j_1)$ и нуль в столбце $S(j_2)$, а также единицы в обоих столбцах. Действительно, первые четыре строки на рис. 19.9 демонстрируют все четыре типа строк. Обозначим через C_{00} количество строк первого типа, через C_{01} — второго типа, через C_{10} — третьего и через C_{11} — четвертого типа.

S_{j_1}	S_{j_2}
0	1
1	0
1	1
0	0
1	1
0	1

Рис. 19.9. Множества $S(j_1)$ и $S(j_2)$; их коэффициент Жаккара равен 2/5

В таком случае

$$J\left(S_{j_1}, S_{j_2}\right) = \frac{C_{11}}{C_{01} + C_{10} + C_{11}}. \tag{19.2}$$

Для того чтобы завершить доказательство, покажем, что правая часть в уравнении (19.2) равна $P\left(x_{j_1}^{\pi} = x_{j_2}^{\pi}\right)$. Для этого рассмотрим столбцы j_1 и j_2; будем увеличивать индекс строки, пока в каком-либо из столбцов не встретится первый ненулевой элемент. Поскольку П — случайная перестановка, вероятность того, что эта наименьшая строка содержит единицу в обоих столбцах, равна выражению в правой части равенства (19.2).

Таким образом, вычисление коэффициента Жаккара на множестве шинглов является вероятностным: мы сравниваем вычисленные значения[8] x_i^{π} из разных документов. Если пары совпадают, то они являются вероятными дубликатами. Повторим этот процесс независимо для 200 перестановок π (как рекомендуется в работах на эту тему). Назовем множество, состоящее из 200 значений x_i^{π}, *эскизом* (sketch) $\psi(d_i)$ документа d_i. Мы можем оценить коэффициент Жаккара для любой пары документов d_i, d_j: $|\psi_i \cap \psi_j|/200$. Если коэффициент Жаккара превышает установленный пороговый уровень, то документы d_i и d_j считаются полудубликатами.

Как быстро вычислить величину $|\psi_i \cap \psi_j|/200$ для всех пар i, j? Действительно, как представить все пары одинаковых документов, не прибегая к квадратичному алгоритму сравнения огромного количества документов? Во-первых, с помощью контрольных сумм мы удаляем все, кроме одной, копии *идентичных* документов. Кроме того, при вычисле-

[8] Напоминаем, что под "вычислением" подразумевается нахождение минимального хеш-значения для каждой перестановки. — *Примеч. ред.*

нии шинглов можно не учитывать теги HTML и целые числа, чтобы исключить шинглы, которые часто встречаются во многих документах и не содержат полезной информации для выявления дубликатов. Затем с помощью *алгоритма объединения–поиска* (union-find algorithm) создадим кластеры, содержащие похожие документы. Для этого необходимо выполнить очень важный этап: перейти от множества *эскизов* к множеству пар *i*, *j*, таких, что d_i и d_j — похожие документы.

Для этого вычислим количество шинглов, общих для любой пары документов, *эскизы* которых имеют общие члены. Начнем со списка пар $\langle x_i^\pi, d_i \rangle$, упорядоченных по величине x_i^π. Для каждого значения x_i^π теперь можно сгенерировать пары *i*, *j*, для которых величина x_i^π присутствует в обоих эскизах. Это позволяет вычислить для каждой пары *i*, *j* с ненулевым пересечением эскизов количество значений x_i^π, принадлежащих обоим эскизам. Применяя предустановленный порог, мы узнаем пары *i*, *j*, имеющие сильное пересечение эскизов. Например, если пороговое значение равно 80%, то для каждой пары *i*, *j* мы должны насчитать как минимум 160 общих шинглов. Идентифицировав такие пары, применим алгоритм объединения-поиска, чтобы сгруппировать полудубликаты в “синтаксические кластеры”. По сути это вариант односвязного алгоритма кластеризации, описанного в разделе 17.2.

Заключительный прием сокращает размер пространства, необходимого для вычисления коэффициента Жаккара $|\psi_i \cap \psi_j|/200$ для пар *i*, *j*, а это вычисление в принципе все еще требует размера пространства, квадратичного к количеству документов. Для исключения из рассмотрения пар *i*, *j*, пересечение эскизов которых содержит немного шинглов, обработаем эскиз каждого документа следующим образом. Упорядочим значения x_i^π в эскизе и применим к этой последовательности шинглирование, чтобы сгенерировать множество *супершинглов* (super-shingles) для каждого документа[9]. Если два документа имеют общий супершингл, то вычисляется точное значение $|\psi_i \cap \psi_j|/200$. Это по-прежнему лишь эвристический прием, но он позволяет существенно сократить количество пар *i*, *j*, для которых подсчитывается величина пересечения эскизов.

? ***Упражнение 19.8.*** Каждая из поисковых систем A и B обходит случайное подмножество веба одинакового размера. Некоторые из загруженных страниц являются дубликатами — точными текстовыми копиями с разными адресами URL. Допустим, что дубликаты равномерно распределены среди страниц, выбранных поисковыми системами A и B. Кроме того, допустим, что дубликат — это страница, имеющая ровно две копии, и ни одна страница не имеет больше двух копий. Поисковая система A индексирует страницы без исключения дубликатов, а поисковая система B индексирует только одну копию каждого документа. До исключения дубликатов два случайных подмножества имеют одинаковые размеры. Какая доля веба состоит из страниц, не имеющих дубликатов, если 45% адресов URL, проиндексированных системой A, встречаются в индексе системы B, и 50% адресов URL, проиндексированных системой B, встречаются в индексе системы A?

[9] Размер множества (т.е. число) неперекрывающихся супершинглов определяется максимально допустимым числом отличающихся шинглов. Например, если мы ищем не более 4,5% отличий в эскизе (не более 9 разных шинглов), мы должны разбить эскиз (из 200 шинглов) на 10 равных групп. Тем самым мы гарантируем, что хотя бы один супершингл для всех таких документов будет обязан совпасть. — *Примеч. ред.*

Упражнение 19.9. Вместо процесса, показанного на рис. 19.8, рассмотрите следующую оценку коэффициента Жаккара для пересечения множеств S_1 и S_2. Извлечем случайное подмножество элементов из генеральной совокупности, которой принадлежат множества S_1 и S_2. Это соответствует извлечению случайного подмножества строк матрицы A в доказательстве теоремы. Вычислим коэффициент Жаккара, перебрав все элементы этих случайных подмножеств. Почему эта оценка является несмещенной оценкой коэффициента Жаккара для множеств S_1 и S_2?

Упражнение 19.10. Объясните, почему эту оценку трудно применить на практике.

19.7. Библиография и рекомендации для дальнейшего чтения

Буш (Bush, 1945), описывая систему управления информацией, которую он назвал *memex*, предсказал появление веба. Бернерс-Ли и др. (Berners-Lee et al., 1992) описали одно из первых воплощений веба. Кумар и др. (Kumar et al., 2000) и Бродер и др. (Broder et al., 2000) провели тщательные исследования веба как графа. Использование текста ссылок впервые описано Макбрайаном (McBryan, 1994). Таксономия поисковых запросов в вебе, описанная в разделе 19.4, предложена Бродером (Broder, 2002). Наблюдение степенного закона с показателем 2,1, описанного в разделе 19.2.1, сделано Кумаром и др. (Kumar et al., 1999). Книга Чакрабарти (Chakrabarti, 2002) представляет собой хороший справочник по многим аспектам поиска и анализа информации в вебе.

Оценка размеров индексов поисковых систем имеет долгую историю, которая прослеживается в работах Бхарата и Бродера, Лоуренса и Джайлза, Хензингера и др., Бар-Йозефа и Гуревича (Bharat and Broder, 1998; Lawrence and Giles, 1998; Rusmevichientong et al., 2001; Lawrence and Giles, 1999; Henzinger et al., 2000; Bar-Yoseff and Gurevich, 2006). Современное состояние исследований отражает работа Бар-Йоссефа и Гуревича (Bar-Yossef and Gurevich, 2006), включая некоторые методы устранения смещения, описанные в конце раздела 19.5. Шинглирование разработано Бродером и др. (Broder et al., 1997) и использовано для нахождения идентичных веб-сайтов (а не просто веб-страниц) Бхаратом и др. (Bharat et al., 2000).

Глава 20

Обход и индексирование веба

20.1. Обзор

Обход веба (web crawling) — это сбор страниц в вебе для их дальнейшего индексирования и поддержки функционирования поисковой системы. Цель обхода — быстро и эффективно собрать как можно больше полезных веб-страниц вместе со ссылками, которые их объединяют. В главе 19 мы говорили о сложности веба, причина которой в том, что в создании сети принимают участие миллионы независимых индивидуумов. В настоящей главе мы исследуем проблемы, возникающие при обходе веба. В центре внимания здесь находится компонент, изображенный на рис. 19.7, — *поисковый робот* (web crawler or spider). [1]

Цель данной главы не сводится к описанию устройства поискового робота полномасштабной коммерческой поисковой системы. Вместо этого мы сосредоточимся на спектре проблем, возникающих при обходе веба, начиная с уровня студенческих работ и заканчивая серьезными исследовательскими проектами. В разделе 20.1 перечислены требования к поисковым роботам, а в разделе 20.2 изложены подходы к их реализации. В оставшейся части описаны архитектура и некоторые детали реализации распределенных поисковых роботов, удовлетворяющих указанным требованиям. В разделе 20.3 обсуждаются индексы, распределенные по многим машинам для реализации в масштабе веба.

20.1.1. Обязательные свойства поискового робота

Требования к поисковым роботам разделяются на две категории: *обязательные* и *желательные*.

Устойчивость. В вебе существуют серверы, создающие *ловушки для поисковых роботов* (spider traps). Они генерируют веб-страницы, вынуждающие поисковых роботов зацикливаться на бесконечном количестве веб-страниц в определенном домене. Поисковые роботы должны быть устойчивыми к таким ловушкам. Не все ловушки созданы злонамеренно; некоторые из них возникают как побочные эффекты небрежной разработки веб-сайта.

Вежливость. Веб-серверы могут иметь явные и неявные правила, регулирующие частоту обращений поисковых роботов. Следует соблюдать эти правила.

---

[1] В англоязычной литературе при описании веба приняты “зоологические” термины: “crawler” — “пресмыкающееся”, “spider” — “паук”, “web” — “паутина”. В русскоязычной литературе принято заменять их техническими терминами: “crawler” и “spider” — “поисковый робот” и “web” — “сеть веб”. — *Примеч. ред.*

20.1.2. Желательные свойства поискового робота

Распределенность. Желательно, чтобы поисковый робот имел возможность функционировать в распределенном режиме на многих машинах.

Масштабируемость. Желательно, чтобы архитектура поискового робота допускала повышение производительности обхода путем добавления новых машин и расширения полосы пропускания.

Производительность и эффективность. Желательно, чтобы поисковый робот эффективно использовал разнообразные ресурсы поисковой системы, включая процессор, память и полосу пропускания компьютерной сети.

Качество. Учитывая, что значительная часть веб-страниц плохо удовлетворяет информационные потребности пользователей, желательно, чтобы поисковый робот при обходе отдавал преимущество, в первую очередь, полезным страницам.

Свежесть. Во многих приложениях желательно, чтобы поисковый робот работал непрерывно, получая свежие копии ранее загруженных страниц. Например, поисковый робот поисковой системы может гарантировать, что ее индекс содержит самые свежие представления каждой из проиндексированных веб-страниц. Для обеспечения такого непрерывного обхода желательно, чтобы поисковый робот мог обходить страницу с частотой, примерно равной частоте ее обновления.

Расширяемость. Поисковые роботы следует разрабатывать так, чтобы их можно было расширять по разным направлениям — учитывать новые форматы данных, новые протоколы передачи данных и т.д. Для этого необходимо, чтобы архитектура поискового робота была модульной.

20.2. Обход веба

Основное предназначение любого гипертекстового поискового робота (в вебе, корпоративной сети или другой коллекции гипертекстовых документов) заключается в следующем. Поисковый робот начинает работу с одного или нескольких URL, образующих *начальное множество* (seed set). Он извлекает URL из этого множества, а затем загружает по нему веб-страницу. После этого выбранная страница обрабатывается и из нее извлекаются текст и ссылки (каждая из них указывает на другой URL). Извлеченный текст передается индексатору (описанному в главах 4 и 5). Извлеченные ссылки (URL) передаются *очереди на скачивание URL* (URL frontier), состоящей из URL, соответствующих страницам, которые должны быть загружены поисковым роботом. В самом начале очередь на скачивание URL содержит начальное множество; как только страница выбрана, ее адрес удаляется из очереди на скачивание URL. В целом этот процесс можно рассматривать как обход веб-графа (см. главу 19). При непрерывном обходе URL выбранной страницы добавляется обратно в очередь URL, чтобы загрузить ее еще раз в будущем.

Этот внешне простой рекурсивный обход веб-графа усложняется многочисленными требованиями, предъявляемыми к поисковым роботам: поисковый робот должен быть распределенным, масштабируемым, корректным, устойчивым и расширяемым, причем выбранные страницы должны иметь высокое качество. Обсудим влияние каждого из перечисленных факторов. При изложении мы следуем устройству поискового робота *Mercator*, который стал основой нескольких исследовательских и коммерческих поисковых роботов. Для того чтобы подчеркнуть сложность задачи, отметим, что для загрузки миллиарда страниц (в настоящее время это лишь небольшая часть статичного веба) в течение

месяца необходимо выбирать несколько сотен страниц в секунду. В дальнейшем мы покажем, как многопотоковая архитектура позволяет устранить узкие места в поисковом роботе и достичь необходимой скорости обработки страниц.

Прежде чем приступить к подробному изложению, повторим для читателей, собирающихся создать собственные поисковые роботы, основные требования к любому роботу, в том числе к непрофессиональному.

1. В каждый момент времени с одним хостом должно быть установлено только одно соединение.
2. Время ожидания между последовательными запросами к узлу сети должно составлять несколько секунд.[2]
3. Следует соблюдать правила доступа к веб-сайтам, упомянутые в разделе 20.2.1.

20.2.1. Архитектура обхода веба поисковым роботом

Простая схема обхода, обрисованная выше, требует наличия нескольких модулей, которые вместе составляют поисковый робот (рис. 20.1).

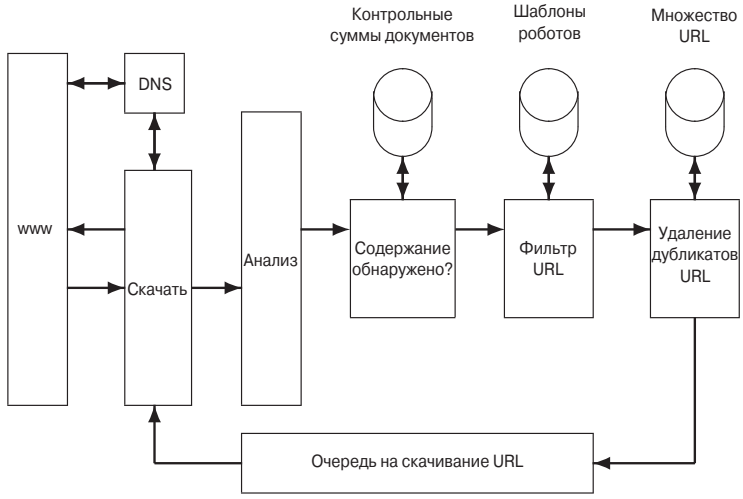


Рис. 20.1. Основная архитектура поискового робота

1. Очередь на скачивание URL, содержащая URL, которые должны быть выбраны в рамках текущего цикла обхода (в случае непрерывного обхода URL может быть уже "обойден", но снова помещен во очередь для повторного обхода). Этот компонент будет описан в разделе 20.2.3.
2. *Модуль разрешения доменных имен* (Domain Name System — DNS) определяет веб-сервер, с которого будет выбрана страница с указанным URL. Этот модуль будет описан в разделе 20.2.2.

[2] Количество секунд должно быть достаточным для того, чтобы не создавать проблем сайтам, однако для больших высоконагруженных сайтов это могут быть и доли секунды. — *Примеч. ред.*

3. Модуль загрузки, использующий протокол http для скачивания веб-страницы по ее URL.
4. Модуль парсирования, извлекающий текст и ссылки из загруженной страницы.
5. Модуль удаления дубликатов, детектирующей такие факты, как, например, есть ли уже данный URL в очереди на скачивание или URL был недавно скачан.

Обход происходит в несколько потоков (количество потоков может достигать сотен), а каждый поток следует логическому циклу, изображенному на рис. 20.1. Эти потоки могут запускаться в одном процессе или разделяться между несколькими процессами, которые выполняются на разных узлах распределенной системы. Отложим описание реализации очереди на скачивание URL до раздела 20.2.3, а пока предположим, что она уже есть в наличии и не пуста. Пройдем по циклу отдельного URL, подлежащего скачиванию, проходящего через различные проверки и фильтры и (при непрерывном обходе) возвращающегося в очередь на скачивание URL.

Процесс обхода начинается с извлечения URL из очереди и выбора веб-страницы, как правило, с помощью протокола http. Извлеченная страница затем записывается во временное хранилище, где над ней производится много различных операций. Далее страница анализируется, и из нее извлекаются текст и ссылки. Текст (с информацией о тегах, например термины, выделенные полужирным шрифтом) передается индексатору. Информация о ссылках, включая текст ссылок (анкор-тексты), также передается индексатору для использования при ранжировании (глава 21). Кроме того, каждая извлеченная ссылка проходит ряд тестов, чтобы определить, должна ли она добавляться к очереди на скачивание URL.

Сначала поток проверяет, встречалась ли веб-страница с идентичным содержанием по другому адресу. Простейшая реализация этой проверки — использование контрольной суммы (помещенной в хранилище с именем “Контрольные суммы документов” на рис. 20.1). Более “продвинутый” тест может быть реализован на основе шинглов, а не контрольных сумм (см. главу 19).

Далее используется *фильтр URL*, определяющий, следует ли исключить URL из очереди на скачивание по результатам одного из нескольких тестов. Например, может быть поставлено условие не обходить определенные домены (например, все URL вида `.com`). В этом случае тест отбрасывает адреса из домена `.com`. Такой тест можно применять для включения, а не исключения URL. Многие веб-сайты устанавливают ограничения на обход своих страниц поисковыми роботами с помощью стандартного средства — *протокола исключений для роботов* (robots exclusive protocol). Для этого в корне иерархии URL, соответствующей конкретному сайту, размещают файл с именем robots.txt. В этом примере файл robots.txt указывает на то, что ни один робот не должен посещать страницу, URL которой в данной иерархии начинается с префикса `/yoursite/temp/`, кроме робота с именем `searchengine`.

```
User-agent: *
Disallow: /yoursite/temp/

User-agent: searchengine
Disallow:
```

Файл robots.txt должен быть извлечен из веб-сайта, чтобы проверить, удовлетворяет ли URL, проходящий проверку, ограничениям, наложенным для роботов, и может ли он быть добавлен в очередь на скачивание URL. Чтобы не загружать файл для каждого про-

веряемого адреса, для хранения файлов robots.txt можно использовать кэш. Это особенно важно, поскольку многие ссылки, извлеченные из страницы, указывают на страницы на этом же хосте, а значит, должны проверяться одним файлом robots.txt. Таким образом, выполняя фильтрацию на этапе извлечения ссылок, мы получим высокий процент удачных попаданий в кэш robots.txt за счет высокой локальности ссылок. К сожалению, это не соответствует ожиданиям веб-мастеров. URL (особенно ссылающийся на документ низкого качества или редко изменяющийся документ) может находиться в очереди на скачивание несколько дней или даже недель. Если фильтрация роботов проводится *до* добавления URL в очередь на скачивание, то его файл robots.txt ко времени, когда URL будет удален из очереди на скачивание и загружен, может измениться. Следовательно, фильтрацию роботов следует проводить непосредственно перед попыткой выбора веб-страницы. Тем не менее поддержка кэша для файлов robots.txt остается очень эффективной, так как локальность сохраняется даже среди адресов, извлекаемых из очереди на скачивание.[3]

Далее URL следует *нормализовать* в следующем смысле: ссылки с веб-страницы *p* могут быть заданы в такой форме, что их URL (относительный) обозначает цель этой ссылки не в абсолютной форме, а по отношению к странице *p*. То есть на HTML-странице могут существовать относительные ссылки. Например, на странице `en.wikipedia.org/wiki/Main_Page` ссылка

>
> Disclaimers

указывает на URL `http://en. wikipedia.org/wiki/Wikipedia:general_disclaimer`.

В заключение URL проходит проверку для исключения дубликатов. Если такой URL в очереди на скачивание уже есть или (при обходе циклами) уже скачан, то в очередь он не добавляется. Если URL все же добавляется в очередь на скачивание, то ему присваивается приоритет, в зависимости от которого он в конце концов удаляется из очереди обхода. Детали назначения приоритета описаны в разделе 20.2.3.

Обычно специальный поток выполняет определенную “уборку”. Этот поток, как правило, бездействует, но включается каждые несколько секунд для того, чтобы записать в лог статистические показатели процесса обхода (скачанные URL, размер очереди и т.д.) и решить, прекратить ли обход или (каждые несколько часов) создать контрольную точку. Процесс *резервного копирования в контрольной точке обхода* (checkpointing) представляет собой запись текущего состояния поискового робота (например, очереди URL) на диск. При сбое поискового робота его работа возобновляется с контрольной точки.

Распределенный поисковый робот

Как уже говорилось, потоки задач в поисковом роботе можно разделить на несколько процессов, каждый из которых выполняется на отдельном узле распределенной системы обхода. Такое распределение имеет большое значение для масштабирования системы; его можно также использовать для географического распределения системы обхода,

---

[3] В последние годы большое распространение получили расширения протокола Robot Exclusion Protocol, в частности появился стандарт SiteMaps, позволяющий явно перечислять URL, которые веб-мастер считает полезными для скачивания. Кроме того, поисковые системы активно предлагают разнообразные средства управления индексацией сайта через центры веб-мастера, в рамках которых владелец подтверждает владение сайтом. — *Примеч. ред.*

в которой каждый узел обходит “близлежащие” хосты. Распределение хостов, подлежащих обходу, по узлам поискового робота можно выполнить с помощью хеш-функции или другой, более тонкой стратегии. Например, можно разместить узел поискового робота в Европе, чтобы сосредоточить его внимание на европейских доменах, хотя это ненадежная стратегия по следующим причинам: маршруты, по которым пакеты передаются в Интернете, не всегда отражают географическую близость, и в любом случае домен хоста не всегда отражает его физическое местоположение.

Как разные узлы распределенного поискового робота взаимодействуют и совместно обрабатывают URL? Идея заключается в репликации потока данных, изображенного на рис. 20.1, на каждом узле с одним существенным отличием: после фильтра URL расположен *разделитель хостов* (host splitter), который распределяет поступающие URL по узлам робота. Таким образом, множество хостов, подлежащих обходу, распределяется между узлами поискового робота. Этот модифицированный поток продемонстрирован на рис. 20.2. Выход распределителя хостов поступает в модули исключения дубликатов URL, расположенные на всех остальных узлах распределенного поискового робота.

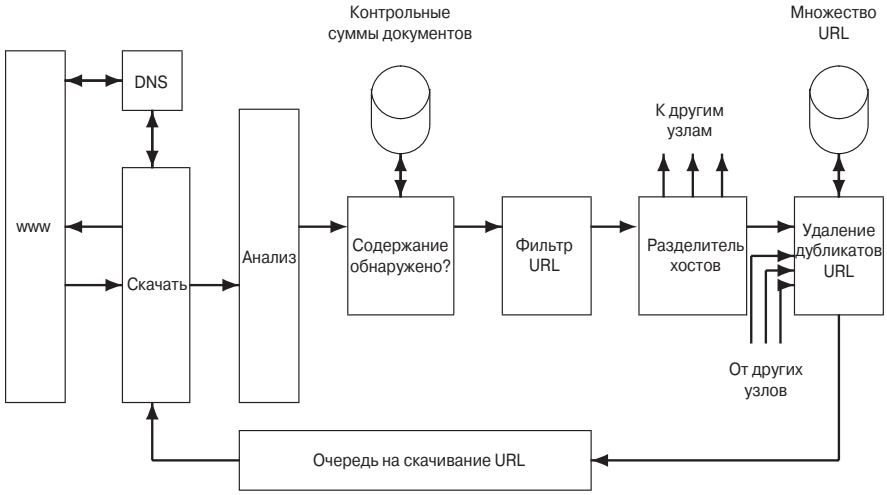


Рис. 20.2. Распределенная архитектура поискового робота

Работу модуля определения дубликатов, являющегося элементом распределенной системы, показанной на рис. 20.2, осложняют несколько факторов.

1. В отличие от очереди на скачивание URL и модуля исключения дубликатов URL, контрольные суммы и шинглы невозможно разделить, основываясь на имени хоста. Ничто не мешает тому же самому (или почти тому же самому) содержимому появляться на разных веб-серверах. Следовательно, множество контрольных сумм/шинглов должно распределяться по узлам на основе их свойств (скажем, результата деления контрольной суммы по модулю, равному количеству узлов). Вследствие такого нарушения локальности в большинстве случаев модуль “Содержание обнаружено?” вызывает удаленную процедуру (хотя существует возможность группировать запросы на поиск дублей).

2. Локальность не проявляется в потоке контрольных сумм/шинглов документов. Таким образом, кэширование популярных контрольных сумм ничего не дает (поскольку популярных контрольных сумм не существует).
3. Документы со временем изменяются и поэтому в контексте непрерывного обхода необходимо иметь возможность удалять устаревшие контрольные суммы и шинглы из множества уже просмотренного содержания. Для этого необходимо хранить контрольные суммы и шинглы документа в очереди на скачивание URL вместе с URL.

20.2.2. Разрешение доменных имен

Каждый веб-сервер (да и любой хост, подсоединенный к Интернету) имеет уникальный *IP-адрес* (IP-address): последовательность четырех байтов, обычно записанную в виде четырех целых чисел, разделенных точками; например, 207.142.131.248 — это числовой IP-адрес, связанный с сайтом `www.wikipedia.org`. Процесс перевода текстового представления URL, например `www.wikipedia.org`, в соответствующий IP-адрес (в данном случае — в адрес 207.142.131.248) называется *разрешением DNS* (DNS resolution) или *DNS-поиском* (DNS lookup). Здесь аббревиатура DNS означает *служба доменных имен* (domain name service). В процессе разрешения DNS программа, желающая выполнить это преобразование (в нашем случае — компонент поискового робота), обращается к *серверу DNS* (DNS-server), который возвращает ей преобразованный IP-адрес. (На практике весь процесс преобразования может выполняться не на одном сервере DNS — сервер DNS, к которому обратился компонент поискового робота, может рекурсивно вызывать другие серверы DNS, чтобы выполнить эту операцию). Для более сложных URL, таких как `en.wikipedia.org/wiki/Domain_Name_System`, компонент поискового робота, отвечающий за разрешение DNS, извлекает имя хоста (в данном случае — `en.wikipedia.org`) и ищет IP-адрес этого хоста.

Разрешение DNS — известное узкое место обхода веба. Из-за распределенного характера службы доменных имен процесс разрешения DNS может включать в себя большое количество запросов и пересылок по Интернету и продолжаться несколько секунд (а иногда еще дольше). Это противоречит нашей цели — извлекать несколько сотен документов за секунду. Стандартное средство решения этой проблемы предусматривает использование кэша: URL, которые недавно прошли процедуру разрешения DNS, с большой вероятностью хранятся в кэше. Это позволяет избежать обращения к серверам DNS в Интернете, однако требование сохранять толерантность по отношению к правилам посещения веб-сайтов (раздел 20.2.3) снижает число удачных попаданий в кэш (cache hit rate).

С разрешением DNS связана еще одна сложность: реализации DNS-поиска (DNS lookup) в стандартных библиотеках (которые чаще всего используются разработчиками поисковых роботов) являются синхронными. Это значит, что как только запрос отсылается в службу доменных имен, другие потоки поискового робота, которые выполняются на данном узле, блокируются, пока первый запрос не будет выполнен. Для того чтобы обойти это препятствие, большинство поисковых роботов реализуют собственные модули разрешения DNS в виде отдельных компонентов. Поток *i*, выполняющий код разрешения, отсылает сообщение на сервер DNS, а затем ожидает: работа возобновляется, когда приходит сигнал от другого потока или истекает время ожидания. Отдельный поток DNS постоянно проверяет стандартный DNS-порт (port 53), ожидая входящих ответных пакетов от сервера имен. Получив ответ, он сигнализирует соответствующему потоку

поискового робота (в данном случае — потоку *i*) и отсылает ему ответный пакет, если поток *i* еще не возобновил свою работу из-за просрочки времени ожидания. Поток поискового робота, возобновивший свою работу из-за истечения времени ожидания, постоянно выполняет фиксированное количество попыток, отсылая новое сообщение серверу DNS и переходя в режим ожидания. Разработчики системы Mercator рекомендуют выполнять примерно пять попыток. После каждой из них время ожидания экспоненциально возрастает. Система Mercator начинает с одной секунды и заканчивает примерно 90 секундами, с учетом того, что существуют имена хостов, для разрешения которых необходимо затратить десятки секунд.

20.2.3. Очередь на скачивание URL

Очередь на скачивание URL, расположенная на узле, получает URL от собственного процесса обхода (или через распределитель по хостам от другого процесса обхода). Она хранит URL и выдает их в определенном порядке, когда какому-нибудь потоку поискового робота нужен новый URL. Порядок, в котором очередь возвращает URL, определяется двумя обстоятельствами. Прежде всего, часто изменяющиеся высококачественные страницы должны загружаться чаще. Следовательно, приоритет страницы должен зависеть от скорости ее обновления и качества (для этого нужна подходящая оценка качества). Это сочетание необходимо, поскольку большое количество страниц со спамом полностью изменяется при каждом новом обходе.

Второе условие — вежливость: нежелательно делать много запросов к одному хосту в течение короткого промежутка времени. Вероятность такой ситуации отягощается локальностью ссылок: многие страницы ссылаются на другие страницы того же хоста. В результате очередь на скачивание URL, реализованная как простая очередь с приоритетами, может породить шквал запросов к одному хосту. Это может произойти, даже если ограничить поисковый робот так, чтобы в любой момент к любому хосту могло обращаться не больше одного потока. На практике между последовательными запросами на скачивание, отосланными к одному хосту, вставляются задержки, величина которых должна быть на порядок больше, чем время, затраченное в ходе обработки последнего запроса к данному хосту.

На рис. 20.3 продемонстрирована реализация правил вежливости и установки приоритетов с помощью очереди на скачивание URL, которая гарантирует, что 1) в каждый момент времени открыто только одно соединение с хостом, 2) между последовательными запросами к хосту проходит несколько секунд, 3) предпочтение будет отдано веб-страницам с высоким приоритетом.

В этой схеме основную роль играют два модуля: *F фронтальных очередей* (F front queues) в верхней части рисунка и *B тыльных очередей* (B back queues) в нижней части рисунка. Все эти очереди подчиняются принципу FIFO (First In First Out — FIFO). Фронтальные очереди реализуют распределение приоритетов, а тыльные — правил вежливости. Поток URL, поступающий в очередь на скачивание, проходит через фронтальную и тыльную очереди. При этом *модуль назначения приоритетов* (prioritizer) сначала назначает данному URL целочисленный приоритет *i*, лежащий в диапазоне от 1 до *F*, с учетом истории скачиваний (т.е. с учетом скорости изменения веб-страниц с данным URL между последовательными циклами обхода). Например, документ с высокой частотой изменения получит более высокий приоритет. Другие эвристики могут зависеть от вида приложения и задаваться явно, например URL новостных сайтов могут иметь более высокий приоритет. Получив приоритет *i*, URL окажется в *i*-й фронтальной очереди.

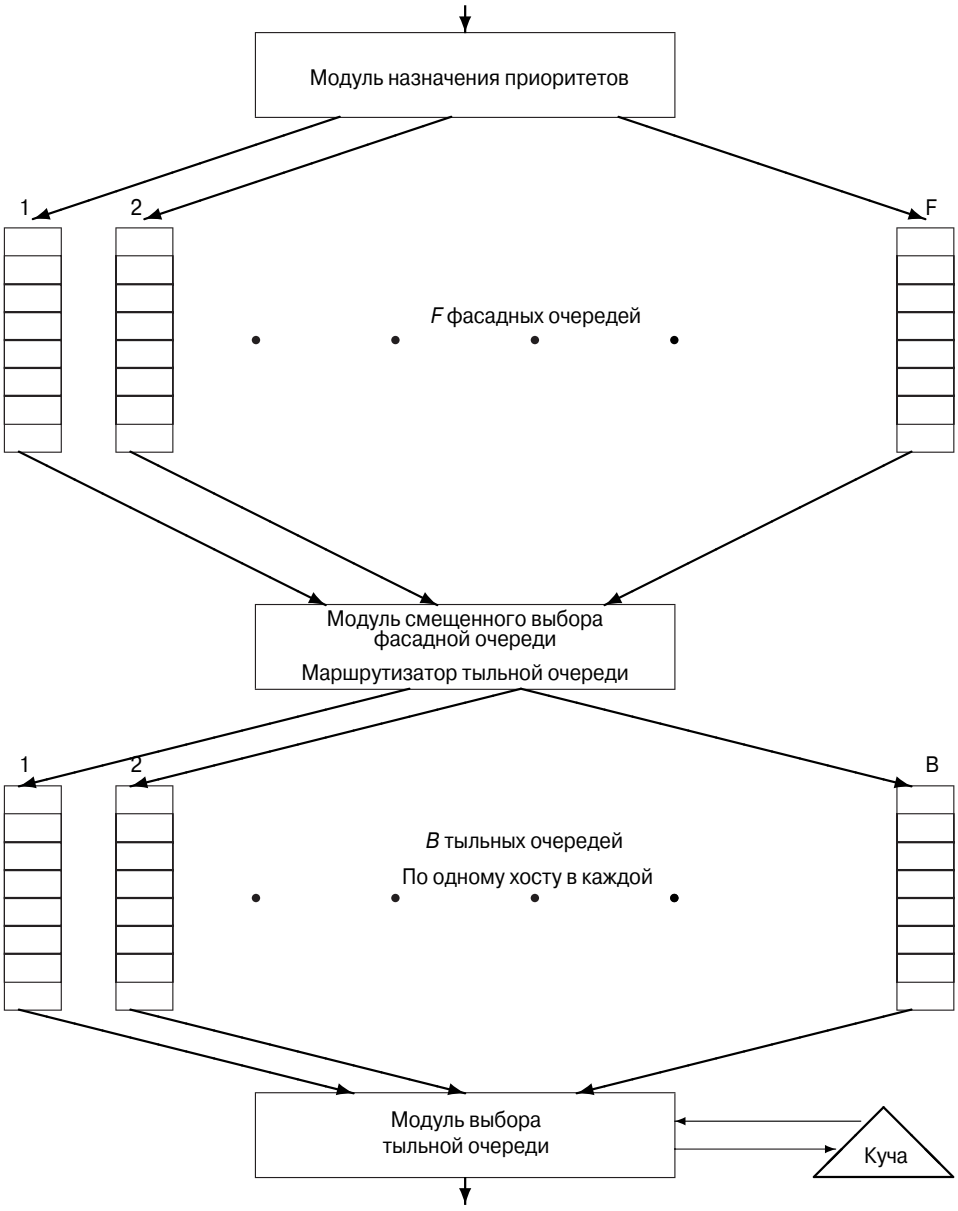


Рис. 20.3. Очередь на скачивание URL. В верхнюю часть рисунка поступают URL, извлеченные из потока уже обойденных страниц. Поток поискового робота, запрашивающий URL, извлекает его из нижней части рисунка. В промежутке между этими частями рисунка потоки URL проходят через одну из фронтальных очередей *(front queues), управляющих приоритетами обхода, и одну из* тыльных очередей *(back queues), управляющих правилами вежливости*

Каждая из *B* тыльных очередей удовлетворяет следующим неизменным условиям (инвариантам): 1) она не пуста, пока обход не закончен, 2) она содержит только URL,

полученные от одного хоста.[4] Для реализации отображения хостов в тыльные очереди используется вспомогательная таблица T (рис. 20.4). Как только тыльная очередь опустеет и начнет заполняться заново из фронтальной очереди, таблица T должна соответственно обновиться.

Хост	Тыльная очередь
stanford.edu	23
microsoft.org	47
acm.org	12

Рис. 20.4. Пример вспомогательной таблицы "тыльная очередь–хост"

Кроме того, система создает кучу (heap), в которой каждый элемент соответствует отдельной тыльной очереди. Этот элемент представляет собой самое раннее время t_e, при наступлении которого можно вновь установить связь с хостом, соответствующим данной очереди.

Поток поискового робота, запрашивающий URL из очереди на скачивание, извлекает вершину этой кучи и (при необходимости) ожидает, пока не наступит соответствующий момент t_e. Затем он берет URL-адрес u из начала тыльной очереди j в соответствии с извлеченным из кучи значением и загружает страницу, расположенную по адресу u. После обработки URL u вызывающий поток проверяет, пуста ли очередь j. Если пуста, он обращается к фронтальной очереди и извлекает из ее начала URL v. Выбор фронтальной очереди является смещенным (как правило, это случайный процесс) в сторону очередей с более высоким приоритетом. Это гарантирует, что URL с высоким приоритетом быстрее "перетекают" в тыльные очереди. Далее проверяется, существует ли тыльная очередь, в которой стоят URL хоста v. Если существует, то URL v добавляется в эту очередь, и нам следует вернуться к фронтальной очереди, чтобы найти другого кандидата на вставку в пустующую очередь j. Этот процесс продолжается, пока очередь j вновь не заполнится. В любом случае поток вставляет в кучу элемент, соответствующий очереди j, с новым наиболее ранним моментом времени t_e, зависящим от свойств URL в очереди j, обработанного последним (например, когда было последнее обращение к этому хосту, сколько времени затрачено на скачивание). Например, новый элемент t_e может представлять собой текущее время, к которому добавлено десятикратное время последнего скачивания.

Количество фронтальных очередей вместе с правилами установки приоритетов и выбора очередей определяют свойства приоритетов, которые следует встроить в систему. Количество тыльных очередей определяет степень, в которой можно загружать потоки заданий поискового робота, не нарушая правил вежливости. Разработчики системы Mercator рекомендуют эмпирическое правило: количество тыльных очередей должно в три раза превышать количество потоков заданий поискового робота.

При обходе в глобальном масштабе очередь на скачивание URL может увеличиться до таких размеров, что потребует больше памяти, чем доступно на узле. Для решения этой задачи бо́льшую часть очереди на скачивание URL можно хранить на диске. Часть каждой очереди хранится в оперативной памяти, и по мере продвижения очереди новые порции загружаются с диска.

[4] Предполагается, что количество хостов намного больше B.

? ***Упражнение 20.1.*** Почему лучше разделять хосты (а не отдельные URL) между узлами распределенного поискового робота?

Упражнение 20.2. Почему разбиение хостов должно происходить до исключения дубликатов URL?

Упражнение 20.3 **[***].** Выше рекомендовались две “жесткие константы”: время t_e должно увеличиваться в `десять` раз по сравнению с временем последнего скачивания, а количество тыльных очередей должно быть в `три` раза больше количества потоков заданий поискового робота. Как эти константы связаны друг с другом?

20.3. Распределение индексов

В разделе 4.4 мы описали распределенное индексирование. Теперь рассмотрим распределение индекса по крупному компьютерному кластеру[5], поддерживающему запросы. Напрашиваются две альтернативные реализации индекса: *разбиение по терминам* (partitioning by terms), известное также как глобальная организация индекса, и *разбиение по документам* (partitioning by documents), известное как локальная организация индекса. В первом случае лексикон индекса разделяется на подмножества, каждое из которых хранится на отдельном узле. Вместе с терминами на узле хранятся словопозиции. Запрос направляется узлам, соответствующим его терминам. В принципе, это позволяет шире применять параллельный режим работы, поскольку поток запросов с разными терминами можно направлять на разные множества машин.

На практике разбиение индексов по терминам лексикона оказывается нетривиальной задачей. Многословные запросы требуют пересылки длинных списков словопозиций между узлами для их объединения. Стоимость этих пересылок перевешивает выгоду от параллельной работы. Балансировка загрузки при таком разбиении определяется не при априорном анализе относительной частоты терминов, а по распределению терминов запроса и их совместной встречаемости, которое со временем смещается или демонстрирует внезапные скачки. Удачное разбиение зависит от совместного появления терминов запроса и накладывает требования к кластеризации терминов, которую нужно оптимизировать в сторону достижения цели, плохо поддающейся количественной оценке. Кроме того, данная стратегия усложняет реализацию динамической индексации.

Более распространенным является разбиение по документам. При этом каждый узел содержит индекс подмножества всех документов. Каждый запрос распределяется по всем узлам, а результаты, полученные от разных узлов, перед представлением пользователю сливаются. Эта стратегия связана с более частыми обращениями к дискам и менее интенсивными пересылками информации между узлами. Единственная трудность, присущая этому подходу, заключается в том, что глобальную статистику, используемую при ранжировании, например частоту idf, необходимо вычислять по всей коллекции документов, в то время как индекс, расположенный на одном узле, содержит лишь часть коллекции. Эти показатели вычисляются с помощью распределенных “фоновых” процессов, которые периодически обновляют статистические показатели в индексах, расположенных в узлах.[6]

---

[5] Пожалуйста, учтите, что в разных главах слово *кластер* имеет разный смысл (см. главы 16 и 17).

[6] Впрочем, глобальную статистику частот терминов можно сделать доступной и на распределяющих/сливающих узлах, тогда ей будет необходимо дополнять каждый запрос, отправляемый на узлы с индексом. — *Примеч. ред.*

Как распределить документы по узлам? Используя идеи, лежащие в основе архитектуры поискового робота, описанной в разделе 20.2.1, можно предложить просто назначать все страницы с одного хоста одному узлу. Это разбиение может соответствовать разбиению хостов по узлам поискового робота. Недостаток такого разбиения заключается в том, что для многих запросов подавляющее число результатов может порождаться документами, расположенными на немногих хостах (а значит, на немногих индексных узлах).

Хеширование каждого URL в пространство индексных узлов порождает более равномерное распределение времени обработки запроса по узлам.[7] В ходе обработки запрос передается каждому из узлов, а k первых результатов, полученных от каждого узла, объединяются в единый список, чтобы определить k первых результатов, соответствующих запросу. Обычно коллекция документов разделяется на индексы документов, имеющих потенциально более высокий ранг по отношению к большинству запросов (например, с помощью методов, описанных в главе 21), и индексы оставшихся документов, имеющих потенциально более низкие ранги. Если среди индексов высокого ранга обнаруживается слишком мало совпадений с запросом, поиск производится в индексах низкого ранга, как описано в разделе 7.2.1.

20.4. Серверы проверки ссылочной связности

По причинам, которые станут ясными в главе 21, поисковые системы нуждаются в *серверах проверки ссылочной связности* (connectivity servers), допускающих быструю обработку *запросов для проверки ссылочной связности* (connectivity queries) на веб-графе. Типичными запросами для проверки ссылочной связности являются запросы *Какие адреса URL имеют ссылки на заданный URL?* и *На какие адреса URL ссылается заданный URL?* Для их обработки необходимо хранить в памяти отображения URL в исходящие ссылки и во входящие ссылки. Приложениями таких запросов являются управление обходом, анализ веб-графа, изощренная оптимизация обхода и *анализ ссылок* (глава 21).

Допустим, что сеть веб состоит из четырех миллиардов страниц, каждая из которых содержит десять ссылок на другие страницы. В простейшем случае для описания каждой ссылки (источника и цели) требуется 32 бит или 4 байт. Следовательно, в целом нужна память размером

$$4 \times 10^9 \times 10 \times 8 = 3{,}2 \times 10^{11} \text{ байт.}$$

Можно использовать некоторые основные свойства веб-графа, чтобы объем данных не превышал 10% этой оценки. На первый взгляд, возникает проблема сжатия данных, которую можно решить одним из стандартных методов. Однако наша цель — не просто сжать веб-граф, чтобы он поместился в памяти, а сделать это так, чтобы эффективно обрабатывать коммуникативные запросы. Эта проблема напоминает задачу сжатия индекса (см. главу 5).

Предположим, что каждая веб-страница представлена единственным целым числом. Конкретная схема присвоения таких целых чисел описана ниже. Построим *таблицу смежности* (adjacency table), напоминающую инвертированный индекс. Каждой веб-странице соответствует строка, а сами строки расположены в порядке возрастания соответствующих целых чисел. Строка, соответствующая произвольной странице p, содер-

[7] Если информация о хосте документа используется при ранжировании или отображении результатов, на каждом узле с индексом должна храниться информация практически о каждом хосте. Это делает неприемлемым такой простой подход для коллекций с сотнями миллионов хостов.

жит упорядоченный список целых чисел, каждое из которых соответствует веб-странице, ссылающейся на страницу *p*. Эта таблица позволяет ответить на вопросы вида *Какие страницы ссылаются на страницу p*? Аналогично строится таблица, элементами которой являются страницы, на которые ссылается страница *p*.

Это табличное представление уменьшает на 50% размер пространства, необходимого для наивного способа (в котором каждая ссылка явно описывается двумя конечными точками, для кодирования каждой из которых требуется 32 бит). Приведенное ниже описание касается таблицы ссылок, *исходящих из* каждой страницы. Совершенно очевидно, что этот метод точно так же можно применить к ссылкам *на* каждую таблицу. Для того чтобы еще больше сэкономить память для хранения таблицы, используем несколько идей.

1. **Сходство между списками.** Многие строки таблицы имеют много общих элементов. Таким образом, если мы явно зададим прототип строки для нескольких похожих друг на друга строк, то оставшиеся строки можно будет сжато выразить через строку-прототип.
2. **Локальность.** Многие ссылки со страницы указывают на "близлежащие" страницы, например на страницы, расположенные на том же хосте. Это значит, что для кодировки целевого адреса ссылки часто можно использовать небольшие целые числа и таким образом экономить память.
3. **Использование смещения в упорядоченных списках.** Вместо целевого адреса каждой ссылки можно хранить смещение относительно предыдущего элемента в строке.

Раскроем смысл каждого из перечисленных методов.

URL можно рассматривать как буквенно-цифровые строки и располагать их в *лексикографическом порядке*. На рис. 20.5 показан фрагмент упорядоченного таким образом списка адресов URL. При истинно лексикографическом порядке следования веб-страниц доменная часть имени должна быть инвертирована; например, строка `www.stanford.edu` принимает вид `edu.stanford.www`, но здесь это необязательно, поскольку нас интересуют в основном локальные ссылки для отдельного хоста.

```
1: www.stanford.edu/alchemy
2: www.stanford.edu/biology
3: www.stanford.edu/biology/plant
4: www.stanford.edu/biology/plant/copyright
5: www.stanford.edu/biology/plant/people
6: www.stanford.edu/chemistry
```

Рис. 20.5. Лексикографически упорядоченные URL

Для каждого URL закодируем его координату в списке с помощью уникального целочисленного идентификатора. Пример такой нумерации и полученная таблица приведены на рис. 20.6. В этом примере URL `www.stanford.edu/biology` присвоено число 2, поскольку он является вторым в последовательности.

```
1: 1, 2, 4, 8, 16, 32, 64
2: 1, 4, 9, 16, 25, 36, 49, 64
3: 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144
4: 1, 4, 8, 16, 25, 36, 49, 64
```

Рис. 20.6. Четыре строки таблицы ссылок

Далее мы используем свойство, проистекающее из того, как большинство веб-сайтов структурировано. Оно позволит нам получить искомые “сходство” и “локальность”. Большинство веб-сайтов имеют единый шаблон страниц, так что с каждой страницы некоторое количество ссылок ведет на фиксированное множество страниц этого же сайта (например, уведомление об авторском праве, условия использования информации и т.д.). В этом случае строки, соответствующие страницам, расположенным на одном веб-сайте, будут иметь много общих элементов таблицы. Более того, при лексикографическом упорядочении URL очень вероятно, что страницам одного веб-сайта будут соответствовать соседние строки таблицы.

Будем придерживаться следующей стратегии: перемещаясь по таблице вниз, кодируем каждую строку с помощью семи предыдущих строк. В примере, показанном на рис. 20.6, четвертую строку можно закодировать как “совпадающая со строкой, имеющей смещение, равное двум (т.е. на две строки выше), в которой число 9 заменено числом 8”. Для этого необходимо дать определения смещения, удаляемых целых чисел (в данном случае — девятки) и добавляемых целых чисел (в данном случае — восьмерки). Использование только семи предшествующих строк имеет два преимущества: 1) смещение можно выразить с помощью только трех битов; это число оптимально и найдено эмпирическим способом (вопросу, почему для кодировки используются семь, а не восемь строк, посвящено упражнение 20.4); 2) фиксация максимального смещения на низком уровне (например, равном семи) позволяет избежать слишком интенсивного поиска среди многих кандидатов на роль прототипа.

А если ни одна из предшествующих семи строк не станет хорошим прототипом для кодировки текущей строки? Это может случиться, например, на границах между разными веб-сайтами, которые мы пересекаем, спускаясь по строкам таблицы. В этом случае мы просто выразим строку как начинающуюся с пустого множества и будем добавлять к ней очередное целое число, встречающееся в этой строке. Используя вместо самих чисел интервалы (разности) между ними и кодируя интервалы строго в соответствии с их распределением, мы достигнем еще большей экономии памяти. В экспериментах, упомянутых в разделе 20.5, ряд методов, описанных здесь, позволил в среднем использовать три бита на ссылку. Это впечатляющая экономия по сравнению с наивным 64-битовым представлением.

Несмотря на то что указанные методы позволяют разместить веб-граф полностью в памяти, нам по-прежнему необходимо поддерживать запросы на проверку ссылочной связности. Что произойдет при поиске исходящих ссылок со страницы с использованием такого представления. Сначала необходимо хешировать адрес URL в номер строки таблицы, а затем — реконструировать эти элементы, которые могут быть закодированы с помощью элементов, стоящих в других строках. Для этого нужно пройти по смещению и восстановить эти строки — процесс, который в принципе может продолжаться довольно долго. Однако на практике это случается редко. Для того чтобы этого избежать, следует учитывать структуру таблицы. Проверяя, можно ли с помощью предыдущих семи строк моделировать текущую строку, необходимо установить пороговое значение сходства между текущей строкой и кандидатом на роль прототипа. Это пороговое значение следует выбирать осторожно. Если оно будет слишком высоким, мы редко будем использовать прототипы. Если же пороговое значение окажется слишком низким, то большинство строк будет выражено через прототипы, поэтому при обработке запроса реконструкция строк приведет к появлению многочисленных промежуточных звеньев, ведущих через предыдущие прототипы.

? ***Упражнение 20.4.*** Как отмечалось выше, выражение строк через одну из семи предыдущих строк позволяет использовать не более трех битов для указания того, какая из строк является прототипом. Почему при этом используются семь, а не восемь строк? (*Подсказка*: *рассмотрите вариант, в котором ни одна из семи предыдущих строк не может использоваться как хороший прототип*.)

Упражнение 20.5. Как указано в разделе 20.4, декодирование ссылок на URL может привести к появлению многочисленных промежуточных звеньев. Придумайте пример, в котором количество таких звеньев линейно возрастает при увеличении количества URL.

20.5. Библиография и рекомендации для дальнейшего чтения

Первый поисковый робот Wanderer был написан Мэтью Греем (Matthew Gray) весной 1993 года. Поисковый робот Mercator создан Найорком и Хейдоном (Najork and Heydon, 2001, 2002). При изложении материала данной главы мы следовали этим работам. Другие ранние классические описания обхода веба приведены в работах Бернера (Burner, 1997), Брина и Пейджа (Brin and Page, 1998), Чо и др. (Cho et al., 1998), а также в работах создателей системы Webbase Стэнфордского университета (Hirai et al., 2000). Чо и Гарсиа-Молина (Cho and Garsia-Molina, 2002) предложили классификацию и провели сравнительный анализ различных режимов взаимодействия узлов распределенного поискового робота. Стандарт Robots Exclusion Protocol описан на веб-странице `www.robotstxt.org/wc/exclusion.html`. Работы Болди и др. (Boldi at el., 2002), а также Шкапенюка и Суэля (Shkapenyuk and Suel, 2002) содержат более актуальную информацию о реализации крупномасштабных распределенных поисковых роботов.

В рассмотрении процесса DNS-разрешения (см. раздел 20.2.2) использованы современные соглашения об интернет-адресах, известные как протокол IPv4 (Internet Protocol version 4). Согласно этому протоколу каждый IP-адрес является последовательностью четырех байтов. В будущем соглашение об адресах (известное как *адресное пространство* Интернета), скорее всего, будет использовать новый протокол IPv6 (`www.ipv6.org/`).

Ключевые работы, посвященные сравнению разбиения распределенных индексов по терминам и по документам, опубликованы Томасиком и Гарсиа-Молина (Tomasic and Garsia-Molina, 1993), а также Йеонгом и Омичинским (Jeong and Omiecinski, 1995). Оказалось, что разбиение по документам является лучшим решением, по крайней мере когда распределение терминов является асимметричным, что соответствует реальному положению дел. Эти результаты были подтверждены более поздней работой (MacFarlane et al., 2000). Однако выводы зависят от деталей устройства распределенной системы. По крайней мере, авторы некоторых работ (Ribeiro-Neto and Barbosa, 1998; Badue et al., 2001) пришли к противоположным выводам. Сорнил (Sornil, 2001) предложил гибридную схему разбиения распределенного индекса по терминам и документам. Баррозо и др. (Barroso et al., 2003) описали методы распределения, принятые в системе Google. Первая реализация сервера связи описана в работе Бхарата и др. (Bharat et al., 1998). Схема, рассмотренная в этой главе и в настоящее время считающаяся лучшей среди опубликованных (в которой ссылка в среднем кодируется тремя битами), описана в ряде статей Болди и Винья (Boldi and Vigna, 2004*a*, 2004*b*).

Глава 21

Анализ ссылок

В развитии веб-поиска важную роль играет анализ гиперссылок и структуры веб-графа. Данная глава посвящена использованию гиперссылок для ранжирования результатов веб-поиска. Такой анализ ссылок является одним из многих факторов, учитываемых поисковыми машинами при ранжировании веб-страницы по запросу. Начнем с обзора основ представления веба в виде графа (раздел 21.1), а затем перейдем к технической разработке элементов анализа ссылок с целью ранжирования.

Ранним предшественником анализа ссылок в вебе можно считать анализ цитирования научных работ, который использовался в библиометрии. Целью этих методов была количественная оценка авторитетности научных статей путем анализа ссылок между ними. По аналогии с научными статьями в контексте веба ссылка рассматривается как признание авторитетности одной страницы по мнению другой. Очевидно, что не любое цитирование или гиперссылка является "положительным отзывом", поэтому простое измерение качества веб-страницы по количеству входящих ссылок (цитат с других страниц) не вполне надежно. Например, можно создать множество веб-страниц, ссылающихся на целевую веб-страницу, чтобы искусственно повысить количество ее входящих ссылок. Это явление называется *ссылочным спамом* (link spam). Тем не менее цитирование является общепринятым и заслуживающим доверия источником полезного сигнала для поисковых машин, позволяющим лучше ранжировать веб-страницы. Анализ ссылок также помогает определить порядок обхода веба, в этом случае анализ ссылок можно использовать для определения приоритетов очереди на скачивание URL (см. главу 20).

В разделе 21.1 изложены идеи, лежащие в основе использования веб-графа для анализа ссылок. Разделы 21.2. и 21.3 посвящены методам анализа ссылок PageRank и HITS.

21.1. Веб как граф

Вспомните понятие веб-графа, введенное в разделе 19.2.1, и особенно рис. 19.2. Изложение методов анализа ссылок основывается на двух интуитивных предположениях.

1. Текст ссылки, указывающей на страницу B, является хорошим описанием страницы B.
2. Гиперссылка со страницы A на страницу B представляет собой признание авторитетности страницы B со стороны создателя страницы A. Это не всегда так; например, многие ссылки со страницы на страницу внутри одного веб-сайта обязаны своим появлением общему шаблону сайта. Например, большинство корпоративных веб-сайтов имеют на каждой странице ссылки на уведомление об авторском праве. Совершенно ясно, что это не является свидетельством одобрения. Соответственно, алгоритмы анализа ссылок учитывают такие ссылки с меньшим весом.

21.1.1. Текст ссылки и веб-граф

Следующий фрагмент HTML-кода веб-страницы демонстрирует гиперссылку на домашнюю страницу журнала *Journal of the ACM*.

```
<a href="http://www.acm.org/jacm/">Journal of the ACM.</a>
```

В данном случае ссылка указывает на страницу `www.acm.org/jacm/`, а текстом ссылки является строка *Journal of the ACM*. Очевидно, в этом примере текст ссылки является описанием целевой страницы. Однако целевая страница (B = `http://www.acm.org/jacm/`) сама содержит такое же описание и дополнительную информацию о журнале. Зачем же нужен текст ссылки?

В вебе имеется множество примеров, когда страница B не содержит точного описания самой себя. Часто это зависит от того, как создатель страницы B хочет представить самого себя. Особенно это характерно для корпоративных веб-сайтов, когда присутствие в вебе подчинено требованиям маркетинга. Например, на момент написания этой книги домашняя страница корпорации IBM (`www.ibm.com`) не содержала термина `computer` нигде в HTML-коде страницы, несмотря на то что компания IBM является крупнейшим производителем компьютеров. Аналогично HTML-код домашней страницы Yahoo! (`www.yahoo.com`) в это же время не содержал слова `portal`.

Итак, довольно часто между терминами на веб-странице и тем, как описали бы эту страницу пользователи, существует несогласованность. Следовательно, при поиске в вебе нет необходимости запрашивать термины, находящиеся на самой странице. Кроме того, многие веб-страницы заполнены графическими изображениями и/или текстом, встроенным в это изображение. При анализе таких HTML-страниц невозможно извлечь полезный для индексирования текст. Решение этой задачи в рамках “классического” информационного поиска предполагает использование методов, изложенных в главе 9 и в разделе 12.4. В качестве альтернативы этим методам можно использовать текст ссылок, воспользовавшись знаниями сообщества авторов веб-страниц.

Тот факт, что текст многих ссылок, указывающих на веб-страницу `www.ibm.com`, содержит слово `computer`, можно использовать в поисковых машинах. Например, терминами текста ссылки можно индексировать целевую веб-страницу. Таким образом, список позиций термина `computer` должен включать документ `www.ibm.com`, а список позиций термина `portal` — документ `www.yahoo.com`. При этом используется специальный индикатор, показывающий, что данные термины встречаются в тексте ссылки (а не на самой странице). Как и в случае терминов, встречающихся на странице, термины текста ссылки (анкор-текста) обычно имеют веса, зависящие от частоты их встречаемости. Термины, встречающиеся слишком часто (наиболее распространенными терминами в текстах ссылок являются слова `Click` и `here`), штрафуются с помощью методов, очень напоминающих метод idf. Реальные веса терминов определяются с помощью взвешивания на основе метода машинного обучения (см. раздел 15.4.1). По-видимому, современные поисковые машины приписывают существенный вес терминам ссылок.

Применение текста ссылок имеет интересный побочный эффект. Поиск фразы `big blue` с помощью большинства поисковых машин возвращает домашнюю страницу корпорации IBM в качестве первого ответа; что соответствует популярному неофициальному прозвищу корпорации IBM. С другой стороны, существует много примеров, когда оскорбительный текст ссылок, например `evil empire`, приводит к неожиданным результатам поиска по этим словам. Это явление использовалось для организации кампаний, направленных против конкретных сайтов. Такие тексты ссылок могут быть спамом; веб-

сайт может создавать дезориентирующие тексты ссылок, указывающих на него же, чтобы завысить свой ранг по отношению к определенным терминам запроса. Идентификация и устранение такого систематического злоупотребления текстом ссылок представляет собой еще одну форму борьбы со спамом, которую ведут поисковые машины.

Текстовое окружение ссылки (иногда его называют *расширенным текстом ссылки*) часто используется так же, как и сам текст ссылки, например фрагмент страницы `there is good discussion of vedic scripture <a>here</a>`. Расширение текста ссылок рассматривалось в разных работах с целью определения оптимальной ширины текстового окружения (см. раздел 21.4).

? ***Упражнение 21.1.*** Всегда ли можно проследовать по ориентированному ребру (гиперссылкам) в веб-графе из любого узла (веб-страницы) в любой другой узел? Аргументируйте свой ответ.

Упражнение 21.2. Приведите пример дезориентирующего текста ссылки в вебе.

Упражнение 21.3. Представьте себе коллекцию фраз из текстов ссылок на веб-страницу *x*. Предложите эвристическое правило для выбора одного термина (или фразы) из такой коллекции, который лучше всего описывает страницу *x*.

Упражнение 21.4. Учитывает ли эвристическое правило, предложенное вами в предыдущем упражнении, отдельный домен *D*, повторяющий текст ссылки на страницу *x* на многих своих страницах?

21.2. Метод PageRank

Рассмотрим теперь методы вычисления весов и ранжирования, вытекающие исключительно из структуры ссылок. Первый метод анализа ссылок присваивает каждому узлу веб-графа вес в диапазоне от нуля до единицы, известный как *PageRank*. Вес узла Page Rank зависит от структуры ссылок в веб-графе. По заданному запросу поисковая машина вычисляет итоговый вес каждой веб-страницы, учитывающий сотни разных признаков, например косинусную меру близости (см. раздел 6.3) и близость терминов (см. раздел 7.7.2) в сочетании с весом PageRank. Этот составной вес, вычисленный с помощью методов из раздела 15.4.1, используется для создания списка ранжированных результатов поиска по запросу.

Рассмотрим случайное блуждание по сети, начинающееся с какой-нибудь веб-страницы (узла веб-графа) и подчиняющееся следующим правилам. На каждом шаге по времени "путешественник" переходит с текущей страницы A на случайно выбранную веб-страницу, на которую на странице A есть гиперссылка. На рис. 21.1 показано, что "путешественник", осуществляющий случайное блуждание по веб-графу, находится в узле A и на следующем шаге с одинаковой вероятностью, равной 1/3, может перейти на один из узлов B, C и D.

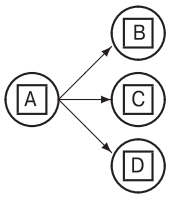


Рис. 21.1. "Путешественник", осуществляющий случайное блуждание по веб-графу, в узле A с вероятностью, равной 1/3, может перейти в узлы B, C и D

В ходе случайного блуждания с узла на узел "путешественник" посещает одни узлы чаще других; интуитивно ясно, что на эти узлы имеется много ссылок, находящихся на других часто посещаемых узлах. Идея, лежащая в основе алгоритма PageRank, заключа-

ется в том, что страницы, чаще других посещаемые в ходе данного случайного блуждания, имеют бо́льшую важность.

Что если в узле A нет исходящих ссылок? Для решения этой проблемы "путешественник", случайно блуждающий по сети, может выполнить дополнительную операцию *телепортации* (teleport). При ее выполнении "путешественник" перепрыгивает с текущего узла на любой другой узел веб-графа. Это может произойти, если наш "путешественник" набирает адрес в строке URL своего браузера. Цель телепортации определяется простым случайным выбором среди всех страниц веба. Иначе говоря, если N — общее количество узлов веб-графа[1], операция телепортации переносит "путешественника" в любой узел с вероятностью $1/N$. Кроме того, "путешественник" может заново попасть в свою прежнюю позицию с вероятностью $1/N$.

Присваивая вес PageRank каждому узлу веб-графа, мы используем операцию телепортации двояко: 1) если узел не имеет исходящих ссылок, то "путешественник" вызывает операцию телепортации; 2) в любом узле, имеющем исходящие ссылки, "путешественник" вызывает операцию телепортации с вероятностью $0 < \alpha < 1$ или продолжает случайное блуждание (следуя по исходящим ссылкам, определенным путем простого случайного выбора) с вероятностью $1 - \alpha$, где α — фиксированный параметр, выбранный заранее. Как правило, $\alpha = 0{,}1$.

В разделе 21.2.1 мы применим теорию марковских цепей и покажем, что когда "путешественник" следует этому комбинированному процессу (случайное блуждание плюс телепортация), он проводит на каждом узле v веб-графа фиксированную долю времени $\pi(v)$, которая зависит от 1) структуры веб-графа и 2) от значения α. Назовем величину $\pi(v)$ весом PageRank узла v. Способ его вычисления приведен в разделе 21.2.2.

21.2.1. Марковские цепи

Марковская цепь (Markov chain) — это дискретный стохастический процесс, представляющий собой последовательность временны́х шагов, на каждом из которых происходит случайный выбор. Марковская цепь состоит из N состояний. Каждая веб-страница соответствует состоянию в создаваемой нами марковской цепи.

Марковская цепь характеризуется *матрицей вероятностей переходов* P, имеющей размерность $N \times N$. Каждый элемент этой матрицы лежит в диапазоне от нуля до единицы. Сумма элементов в строке матрицы переходов равна единице. На каждом временно́м шаге марковская цепь может пребывать в одном из N состояний. Следовательно, элемент P_{ij} представляет собой вероятность, что на следующем временно́м шаге цепь будет находиться в j-м состоянии, при условии, что в данный момент она находится в i-м состоянии. Каждый элемент P_{ij} называется вероятностью перехода и зависит только от текущего состояния. Это свойство называется марковским. Итак,

$$\forall\, i, j\ P_{ij} \in [0, 1]$$

и

$$\forall i \sum_{j=1}^{N} P_{ij} = 1\,. \tag{21.1}$$

Матрица с неотрицательными элементами, удовлетворяющая условию (21.1), называется *стохастической*. Ключевое свойство стохастической матрицы заключается в том,

[1] Это соответствует общему количеству документов в коллекции.

что она имеет *главный левый собственный вектор*, соответствующий максимальному собственному значению, которое равно единице.

В марковской цепи распределение вероятностей следующих состояний зависит от текущего состояния, а от предыстории не зависит. На рис. 21.2 показана простая марковская цепь с тремя состояниями. Из среднего состояния A можно с одинаковой вероятностью, равной 0,5, перейти как в состояние B, так и в состояние C. Из состояний B и C мы с вероятностью, равной единице, вернемся в состояние A. В таком случае матрица вероятностей переходов для этой марковской цепи имеет следующий вид.

$$\begin{pmatrix} 0 & 0{,}5 & 0{,}5 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

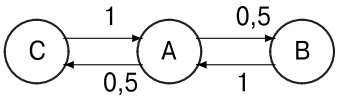


Рис. 21.2. Простая марковская цепь с тремя состояниями; числа над стрелками означают вероятности перехода

Распределение вероятностей марковской цепи можно рассматривать как *вектор вероятностей* (probability vector), элементы которого лежат в интервале от нуля до единицы, а их сумма равна единице. *N*-мерный вектор вероятностей, элементы которого соответствуют состояниям марковской цепи, можно рассматривать как распределение вероятностей по состояниям. Для простой марковской цепи, изображенной на рис. 21.2, вектор вероятностей имеет три компонента, сумма которых равна единице.

"Путешественника", случайно блуждающего по вебу, можно рассматривать как марковскую цепь. Каждая страница — это состояние цепи, а каждая вероятность перехода соответствует вероятности перехода с одной страницы на другую. Операция телепортации учитывается вероятностью перехода. Матрица смежности A веб-графа определяется следующим образом: если существует гиперссылка со страницы i на страницу j, то $A_{ij} = 1$, в противном случае $A_{ij} = 0$. По матрице смежности A, имеющей размер $N \times N$, можно легко построить матрицу вероятностей переходов P для заданной марковской цепи. Для всех строк, содержащих единицы, следует выполнить такие операции.

1. Разделить каждую единицу в матрице A на количество единиц в данной строке. Таким образом, если в строке есть три единицы, то каждая из них будет заменена значением 1/3.
2. Умножить полученную матрицу на $1 - \alpha$.
3. Добавить α/N к каждому элементу результирующей матрицы.

Распределение вероятностей положения "путешественника" в любой момент времени можно задать с помощью вектора вероятностей $\vec{x}$. В момент $t = 0$ блуждание начинается с состояния, которому соответствует единица в векторе $\vec{x}$, а все остальные элементы равны нулю. По определению распределение вероятностей состояния "путешественника" в момент $t = 1$ задается вектором вероятностей $\vec{x}P$; в момент $t = 2$ — вектором $(\vec{x}P)P = \vec{x}P^2$ и т.д. Более подробно этот процесс описан в разделе 21.2.2. Таким образом, мы

можем вычислить распределение вероятностей состояния путешественника в любой момент времени, зная лишь начальное распределение и матрицу вероятностей переходов P.

Если марковская цепь допускает многократные шаги, то цепь будет переходить в каждое состояние с определенной (разной) частотой, зависящей от структуры марковской цепи. Возвращаясь к нашей аналогии, "путешественник" будет посещать одни веб-страницы (например, главные страницы популярных новостных сайтов) чаще, чем другие. Теперь уточним наши интуитивные предположения, установив условия, при которых частота перехода в состояние стремится к фиксированной стационарной величине. Следуя этому подходу, установим вес PageRank каждого узла v равным этой стационарной частоте посещения и покажем, как его вычислить.

Определение. Марковская цепь называется *эргодической* (ergodic), если существует положительное целое число T_0, такое, что для всех пар состояний i, j в марковской цепи, стартующей в момент $t = 0$ из состояния i, для всех $t > T_0$ вероятность нахождения в момент времени t в состоянии j больше нуля.

Для того чтобы марковская цепь была эргодической, должны быть выполнены два технических условия, касающихся ее состояний и ненулевых вероятностей переходов. Эти условия называются *неразложимостью* (irreducibility) и *апериодичностью* (aperiodicity). Говоря неформально, первое условие гарантирует, что существует последовательность переходов с ненулевой вероятностью из любого состояния в любое другое, а второе гарантирует, что состояния не делятся на такие множества, что все переходы осуществляются циклически из состояний одного множества в состояния другого множества.

Теорема 21.1. *Для любой эргодической цепи Маркова существует единственный вектор стационарного распределения вероятностей* $\vec{\pi}$, *являющийся главным левым собственным вектором матрицы* P, *и если* $\eta(i, t)$ *— количество переходов в состояние* i *за* t *шагов, то*

$$\lim_{t\to\infty}\frac{\eta(i,t)}{t}=\pi(i),$$

где $\pi(i) > 0$ *— стационарная вероятность состояния* i.

Из теоремы 21.1 следует, что случайное блуждание с телепортацией приводит к единственному стационарному распределению вероятностей по состояниям в индуцированной марковской цепи. Эта стационарная вероятность состояния равна весу PageRank соответствующей веб-страницы.

21.2.2. Вычисление веса PageRank

Как вычислить вес PageRank? По определению (18.2) левые собственные векторы $\vec{\pi}$ матрицы вероятностей переходов P состоят из N чисел и удовлетворяют условию

$$\vec{\pi}P = \lambda\vec{\pi}. \qquad (21.2)$$

N элементов главного собственного вектора $\vec{\pi}$ представляют собой стационарные вероятности случайного блуждания с телепортацией, т.е. веса PageRank соответствующих веб-страниц. Уравнение (21.2) можно интерпретировать следующим образом: если $\vec{\pi}$ — распределение вероятностей нахождения "путешественника" на веб-страницах, то оно является стационарным. Если $\vec{\pi}$ — стационарное распределение, то $\vec{\pi}P = 1\vec{\pi}$, поэтому единица является собственным значением матрицы P. Таким образом, вычислив главный левый собственный вектор матрицы P (соответствующий собственному значению, равному единице), мы найдем веса PageRank.

Левые собственные векторы можно вычислить с помощью многих алгоритмов (соответствующие ссылки приведены в конце главы 18 и этой главы). Здесь мы приведем довольно простой способ — *степенной метод* (power iteration). Если $\vec{x}$ — начальное распределение по состояниям, то распределение в момент t равно $\vec{x}P^t$. При увеличении переменной t можно ожидать, что распределение $\vec{x}P^t$ будет очень похоже на распределение $\vec{x}P^{t+1}$. При больших значениях t можно рассчитывать на то, что марковская цепь достигнет стационарного состояния.[2] По теореме 21.1 оно не зависит от начального распределения $\vec{x}$. Степенной метод имитирует случайное блуждание: начав с определенного состояния и выполнив большое количество шагов t, мы отслеживаем частоты посещения каждого состояния. После большого количества шагов t эти частоты "устанавливаются", так что разница между вычисленными частотами опускается ниже заданного порогового значения. Эти частоты объявляются весами PageRank.

Рассмотрим веб-граф, упомянутый в упражнении 21.6, при $\alpha = 0{,}5$. Матрица вероятностей переходов для случайного блуждания с телепортацией примет следующий вид.

$$P = \begin{pmatrix} 1/6 & 2/3 & 1/6 \\ 5/12 & 1/6 & 5/12 \\ 1/6 & 2/3 & 1/6 \end{pmatrix} \tag{21.3}$$

Представим, что "путешественник" отправляется из состояния 1, соответствующего начальному вектору распределения $\vec{x}_0 = (1 \quad 0 \quad 0)$. Тогда через один шаг распределение изменится:

$$\vec{x}_0 P = \left(1/6 \quad 2/3 \quad 1/6\right) = \vec{x}_1 . \tag{21.4}$$

Через два шага получим

$$\vec{x}_1 P = \left(1/6 \quad 2/3 \quad 1/6\right)\begin{pmatrix} 1/6 & 2/3 & 1/6 \\ 5/12 & 1/6 & 5/12 \\ 1/6 & 2/3 & 1/6 \end{pmatrix} = \left(1/3 \quad 1/3 \quad 1/3\right) = \vec{x}_2 . \tag{21.5}$$

Продолжая так далее, получим векторы вероятностей, приведенные на рис. 21.3.

$\vec{x}_0$	1	0	0
$\vec{x}_1$	1/6	2/3	1/6
$\vec{x}_3$	1/3	1/3	1/3
$\vec{x}_4$	1/4	1/2	1/4
$\vec{x}_5$	7/24	5/12	7/24
...	...	...	...
$\vec{x}$	5/18	4/9	5/18

Рис. 21.3. Последовательность векторов вероятностей

[2] Обратите внимание, что P^t обозначает P, возведенную в степень t, а не транспонирование P (эта операция обозначается P^T).

Сделав несколько шагов, мы убедимся, что распределения сходятся к стационарному распределению $\vec{x} = (5/18 \quad 4/9 \quad 5/18)$. В этом простом примере стационарное распределение можно просто вычислить, учитывая симметрию данной марковской цепи: состояния 1 и 3 являются симметричными, поскольку первая и третья строки матрицы вероятностей переходов (21.3) идентичны. Оба эти состояния имеют одинаковые стационарные вероятности p. В таком случае стационарное распределение вероятностей равно $\vec{\pi} = (p \quad 1-2p \quad p)$. Итак, используя тождество $\vec{\pi} = \vec{\pi}P$, мы решим простое линейное уравнение и получим $p = 5/18$. Следовательно, $\vec{\pi} = (5/18 \quad 4/9 \quad 5/18)$.

Значения весов PageRank (и неявный порядок страниц) не зависят от запроса пользователя. Таким образом, вес PageRank является мерой статического качества веб-страницы, не зависящей от запросов пользователей (аналогичные статические меры качества были введены в разделе 7.1.4). С другой стороны, интуитивно ясно, что относительный порядок страниц должен зависеть от обрабатываемого запроса. По этой причине поисковые машины используют статические показатели, такие как вес PageRank, наряду со многими другими рангами веб-страницы. Действительно, относительный вклад веса PageRank в общий ранг страницы определяется с помощью метода машинного обучения, описанного в разделе 15.4.1.

Пример 21.1. Проанализируйте граф, изображенный на рис. 21.4. При коэффициенте телепортации, равном 0,14, стохастическая матрица вероятностей переходов имеет следующий вид.

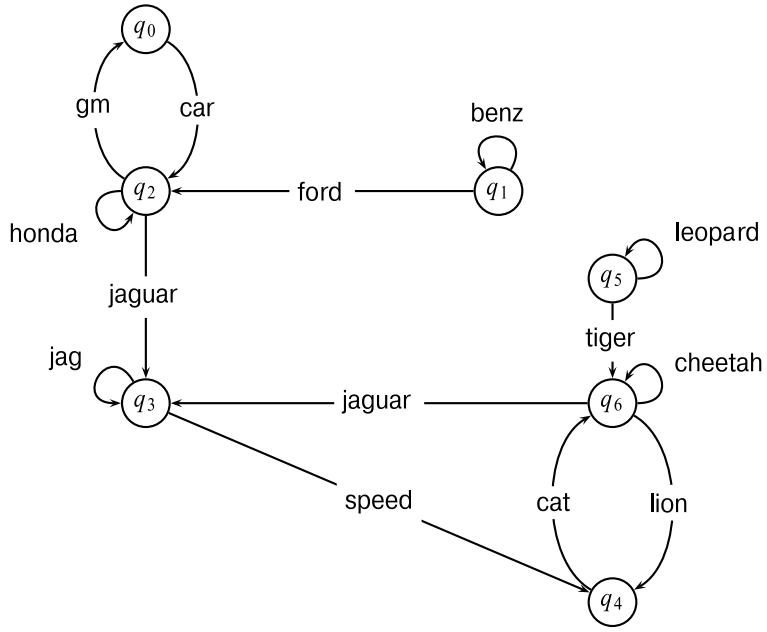


Рис. 21.4. Небольшой веб-граф. Дуги обозначены словами, встречающимися в якорном тексте соответствующей ссылки

0,02	0,02	0,88	0,02	0,02	0,02	0,02
0,02	0,45	0,45	0,02	0,02	0,02	0,02
0,02	0,02	0,31	0,31	0,02	0,02	0,02
0,02	0,02	0,02	0,45	0,45	0,02	0,02
0,02	0,02	0,02	0,02	0,02	0,02	0,88
0,02	0,02	0,02	0,02	0,02	0,45	0,45
0,02	0,02	0,02	0,31	0,31	0,02	0,31

Вектор весов PageRank для этой матрицы имеет следующий вид.

$$\vec{x} = (0,05 \quad 0,04 \quad 0,11 \quad 0,25 \quad 0,21 \quad 0,04 \quad 0,31) \tag{21.6}$$

На рис. 21.4 q_2, q_3, q_4 и q_6 — узлы, имеющие по крайней мере две входящие ссылки. Из них узел q_2 имеет наименьший вес PageRank, поскольку случайное блуждание смещается к нижней части графа, "путешественник" может вернуться в верхнюю только в результате телепортации.

21.2.3. Тематические веса PageRank

До сих пор мы рассматривали вычисление веса PageRank с помощью операции телепортации, позволяющей "путешественнику" перепрыгивать на равномерно случайно выбранную веб-страницу. Теперь рассмотрим телепортацию на случайную веб-страницу, выбранную *неравномерно*. Это позволяет настраивать веса PageRank, соответствующие конкретным темам. Например, спортивный болельщик хотел бы, чтобы страницы со спортивными новостями имели более высокие ранги, чем другие. Допустим, что веб-страницы, посвященные спорту, расположены "близко" на веб-графе. В таком случае "путешественник", часто заходящий на случайно выбранные спортивные веб-страницы, скорее всего, б*о*льшую часть времени проведет именно там. Следовательно, стационарные вероятности спортивных страниц возрастают.

Допустим, что "путешественник" совершает телепортацию на *случайную спортивную веб-страницу*, а не на равномерно выбранную случайную веб-страницу. Мы не будем останавливаться на способе определения подмножества всех спортивных веб-страниц. Фактически нам необходимо только непустое подмножество S веб-страниц, связанных со спортом, на которые можно "телепортировать" пользователя. Такое подмножество можно получить, например, с помощью каталога спортивных страниц, созданного вручную, скажем, в рамках Open Directory Project (`www.dmoz.org/`) или каталога Yahoo.

Если множество S спортивных веб-страниц не пусто, то существует случайное непустое множество $Y \supseteq S$, блуждание на котором имеет стационарное распределение. Обозначим такое распределение *весов PageRank спортивных страниц* через $\vec{\pi}_s$. Для веб-страниц, не принадлежащих Y, вес PageRank полагается равным нулю. Назовем распределение $\vec{\pi}_s$ распределением *тематических весов PageRank* спортивных страниц.

Мы не требуем, чтобы телепортация происходила равномерно по всем спортивным страницам. Распределение по целям телепортации S фактически может быть произвольным.

Аналогичным образом можно ввести распределение весов PageRank для любой другой темы, например науки, религии, политики и т.д. Каждое из этих распределений приписывает каждой веб-странице вес PageRank из интервала [0, 1). Для пользователя, интересующегося только одной из этих тем, при ранжировании результатов поиска можно

применять соответствующее распределение весов PageRank. Это позволяет поисковой машине учитывать тему, которая интересует пользователя. Для этого либо пользователь должен явно указать свои интересы, либо система должна обучаться на основе поведения отдельных пользователей.

А что если пользователя интересует много тем? Например, пользователь может на 60% интересоваться спортом, а на 40% — политикой. Такое распределение интересов называется *профилем* (profile). Можно ли вычислить *персональный вес PageRank* (personalized PageRank) для этого пользователя? На первый взгляд, вычислить распределения весов PageRank для каждого профиля пользователя (которых может быть громадное количество) слишком сложно. Справиться с этой проблемой можно, предположив, что индивидуальные интересы пользователя хорошо аппроксимируются линейной комбинацией небольшого количества тематических распределений. Пользователь со смешанными интересами может совершать телепортацию следующим образом. Сначала он определяет, куда телепортироваться — на одну из спортивных или политических страниц. Выбор производится случайным образом: спортивные страницы выбираются в 60% случаев, а политические — в 40% (рис. 21.5).

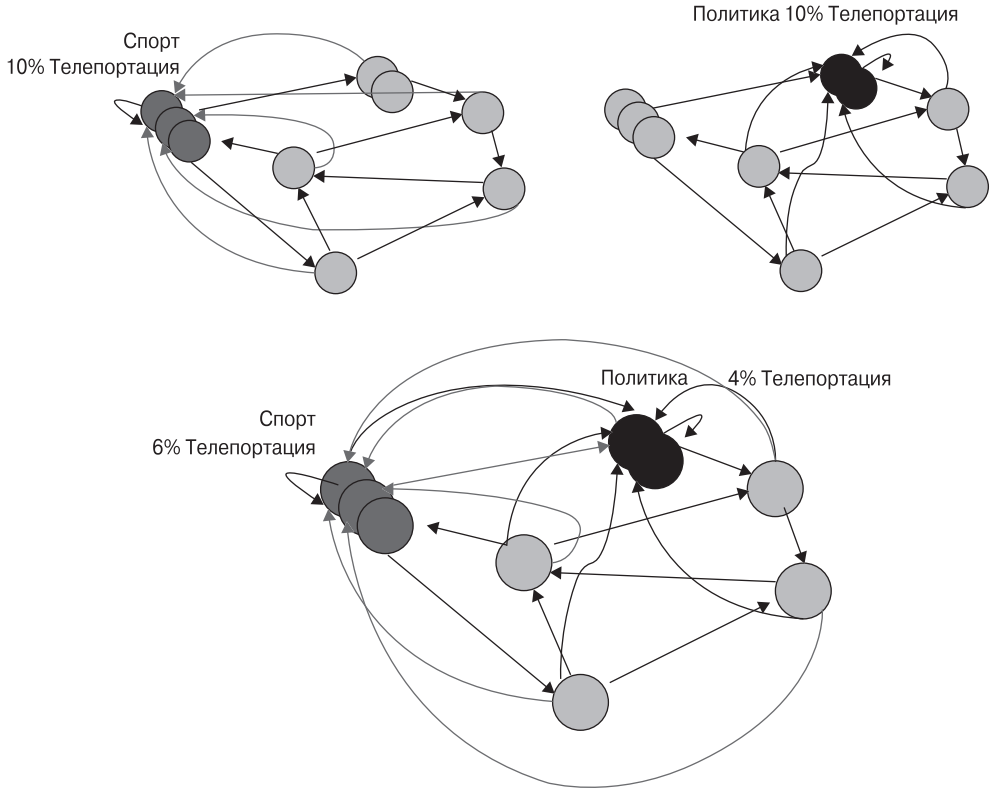


Рис. 21.5. Тематические веса PageRank. В этом примере пользователя на 60% интересует спорт и на 40% — политика. Если вероятность телепортации равна 10%, то вероятность телепортации на спортивную страницу равна 6%, а на страницу о политике — 4%

Выбрав множество (скажем, спортивных страниц), "путешественник" равномерным образом выбирает случайную страницу из этого множества. Это, в свою очередь, приводит к эргодической марковской цепи со стационарным распределением, учитывающим персональные интересы пользователя (упражнение 21.16).

Хотя идея интуитивно понятна, ее реализация выглядит сложной. На первый взгляд, матрицу вероятностей переходов и соответствующее стационарное распределение необходимо вычислять для каждого пользователя. Ситуацию упрощает тот факт, что эволюцию распределения вероятностей по состояниям марковской цепи можно рассматривать как линейную систему. В упражнении 21.6 мы покажем, что вычислять вектор весов PageRank для каждой комбинации тематических интересов пользователя необязательно. Персональный вектор весов PageRank любого пользователя можно выразить в виде линейной комбинации базовых тематических весов PageRank. Например, персональный вектор весов PageRank для пользователя, интересы которого разделены между спортом и политикой в пропорции 60:40, можно вычислить так.

$$0,6\vec{\pi}_s + 0,4\vec{\pi}_p \tag{21.7}$$

Здесь $\vec{\pi}_s$ и $\vec{\pi}_p$ — тематические веса PageRank для спортивных и политических страниц соответственно.

? ***Упражнение 21.5.*** Запишите матрицу вероятностей переходов для примера, изображенного на рис. 21.2.

Упражнение 21.6. Рассмотрите веб-граф с узлами 1, 2 и 3. Связи в этом графе таковы: $1 \to 2$, $3 \to 2$, $2 \to 1$ и $2 \to 3$. Запишите матрицы вероятностей переходов для случайного блуждания с телепортацией при трех значениях вероятности телепортации: 1) $\alpha = 0$, 2) $\alpha = 0{,}5$ и 3) $\alpha = 1$.

Упражнение 21.7. Пользователь браузера в дополнение к кликам по гиперссылкам на странице *x*, которую он в данный момент просматривает, может перейти по кнопке Назад и вернуться на предыдущую страницу. Можно ли эту ситуацию описать с помощью марковской цепи? Как смоделировать повторяющиеся щелчки на кнопке Назад?

Упражнение 21.8. Рассмотрите марковскую цепь с состояниями A, B и C и следующими вероятностями перехода. Из состояния A в состояние B цепь переходит с вероятностью, равной единице. Из состояния B переход в состояние A осуществляется с вероятностью p_A, а в состояние C — с вероятностью $1 - p_A$. Из состояния C цепь переходит в состояние A с вероятностью, равной единице. При каких значениях $p_A \in [0,1]$ эта марковская цепь является эргодической?

Упражнение 21.9. Покажите, что для любого ориентированного графа марковская цепь, индуцированная случайным блужданием с операцией телепортации, является эргодической.

Упражнение 21.10. Покажите, что вес PageRank каждой страницы не меньше α/N. Какой вывод следует из этого относительно разницы значений весов PageRank (для разных страниц), если вероятность α близка к единице?

Упражнение 21.11. Для данных из упражнения 21.1 напишите небольшую программу или используйте научный калькулятор и вычислите значения весов Page Rank по формуле (21.6).

Упражнение 21.12. Предположим, что веб-граф хранится на диске в виде списка смежности так, что по запросу можно получать список соседей (страницы, на которые есть ссылки с данной) в том порядке, в котором они хранятся. Граф нельзя полностью загрузить в оперативную память, но можно делать много запросов по всему графу. Напишите алгоритм вычисления весов PageRank при данных условиях.

Упражнение 21.13. Вспомните о множествах S и Y, введенных в начале раздела 21.2.3. Как множество Y связано с множеством S?

Упражнение 21.14. Всегда ли множество Y содержит все веб-страницы? Аргументируйте свой ответ.

Упражнение 21.15 **[***].** Превышает ли спортивный вес PageRank любой страницы из множества S ее обычный вес PageRank?

Упражнение 21.16 **[***].** Рассмотрим два тематических веса PageRank для каждой веб-страницы: вектор спортивных весов PageRank $\vec{\pi}_s$ и вектор политологических весов PageRank $\vec{\pi}_p$. Пусть α — общая вероятность телепортации, используемая при вычислении обоих тематических весов. Для $q \in [0, 1]$ представим себе пользователя, интересы которого разделены между спортом и политикой в пропорции q к $1 - q$. Покажите, что вектор персональных весов PageRank данного пользователя представляет собой стационарное распределение случайного блуждания, в ходе которого на этапе телепортации пользователь переходит на спортивную страницу с вероятностью q и на политическую страницу — с вероятностью $1 - q$.

Упражнение 21.17. Покажите, что марковская цепь, соответствующая блужданию, описанному в упражнении 21.16, является эргодической, и, следовательно, персональный вектор весов PageRank можно найти, вычислив стационарное распределение марковской цепи.

Упражнение 21.18. Покажите, что в стационарном распределении из упражнения 21.17 стационарная вероятность для любой веб-страницы i равна $q\pi_s(i) + (1-q)\pi_p(i)$.

21.3. Порталы и авторитетные источники

Разработаем схему, в которой по заданному запросу каждая веб-страница получает *два* показателя. Один из них называется *показателем портальности* (hub score), а второй — *показателем авторитетности* (authority score). В этой схеме для любого запроса вычисляются два ранжированных списка результатов, а не один. Первый список ранжируется в соответствии с показателем портальности, а второй — с показателем авторитетности.

Этот подход возник в результате анализа процесса создания веб-страниц: существуют две основные разновидности веб-страниц, полезных при *широком поиске* (broad-topic search). Под широким поиском мы понимаем информационные запросы вида “Я хочу знать о лейкемии”. Существуют авторитетные источники информации на эту тему, в частности веб-страница Национального института рака (National Cancer Institute), посвященная лейкемии. Такие страницы мы будем называть *авторитетными источниками* (authorities). В описанных ниже вычислениях такие веб-страницы должны получить высокий показатель авторитетности.

С другой стороны, многие страницы в вебе представляют собой списки ссылок на авторитетные веб-страницы, посвященные конкретным темам. Эти страницы — *порталы* (hub) — сами по себе авторитетными источниками не являются, но содержат информацию, собранную

и организованную заинтересованными людьми. В рамках описываемого подхода порталы используются для поиска авторитетных источников. В вычислениях, которые мы сейчас опишем, порталами являются веб-страницы, имеющие высокий портальный показатель.

Хорошая портальная страница ссылается на многие авторитетные источники. Хорошей авторитетной страницей считается веб-страница, на которую ссылаются многие хорошие портальные страницы. Следовательно, определения портальной и авторитетной страниц связаны друг с другом, и мы можем найти веса страниц с помощью итеративных вычислений. Предположим, что у нас есть подмножество веб-страниц, содержащее хорошие портальные и авторитетные страницы вместе со ссылками. Мы итеративно вычислим портальный показатель и показатель авторитетности каждой веб-страницы из этого множества, отложив обсуждение выбора этого подмножества до раздела 21.3.1.

Портальный показатель веб-страницы v в нашем подмножестве будем обозначать как $h(v)$, а показатель авторитетности — как $a(v)$. В начале $h(v) = a(v) = 1$ для всех узлов v. Кроме того, обозначим как $v \mapsto y$ гиперссылку со страницы v на страницу y. В основе итеративного алгоритма лежит обновление портального показателя и показателя авторитетности для всех страниц по формуле (21.8). Это отражает интуитивные представления о том, что хороший портал ссылается на хорошие авторитетные страницы, а на хорошие авторитетные страницы ссылаются хорошие порталы.

$$\begin{aligned} h(v) &\leftarrow \sum_{v \mapsto y} a(y), \\ a(v) &\leftarrow \sum_{y \mapsto v} h(y) \end{aligned} \tag{21.8}$$

Таким образом, первая строка в формуле (21.8) определяет показатель портальности страницы v как сумму показателей авторитетности страниц, на которые она ссылается. Иначе говоря, если страница v ссылается на страницы с высокими показателями авторитетности, то ее показатель портальности увеличивается. Вторая строка имеет противоположный смысл. Если на страницу v ссылаются хорошие порталы, то ее показатель авторитетности увеличивается.

Что происходит, когда мы выполняем эти вычисления итеративно, заново вычисляя сначала показатели портальности, а затем — показатели авторитетности, основанные на обновленных показателях портальности, и т.д.? Преобразуем формулу (21.8) в матричный вид. Обозначим через $\vec{h}$ и $\vec{a}$ векторы всех показателей портальности и авторитетности для страниц, принадлежащих нашему подмножеству. Пусть A — квадратная матрица смежности подмножества веб-графа, в которой каждой странице соответствуют одна строка и один столбец. Элемент A_{ij} равен единице, если существует гиперссылка со страницы i на страницу j, и нулю — в противном случае. Следовательно, формулы (21.8) можно переписать следующим образом.

$$\begin{aligned} \vec{h} &\leftarrow A\vec{a}, \\ \vec{a} &\leftarrow A^T\vec{h} \end{aligned} \tag{21.9}$$

Здесь A^T — транспонированная матрица A. Теперь правая часть каждой строки в формуле (21.9) представляет собой вектор, расположенный в левой части другой строки формулы (21.9). Путем взаимной подстановки формулы (21.9) можно переписать в следующем виде.

$$\begin{aligned} \vec{h} &\leftarrow AA^T\vec{h}, \\ \vec{a} &\leftarrow A^TA^T\vec{a} \end{aligned} \tag{21.10}$$

Теперь формула (21.10) очень похожа на пару уравнений для вычисления собственных векторов (см. раздел 18.1). Действительно, если заменить символы ← символами = и ввести неизвестное собственное значение, то первая строка в формуле (21.10) примет вид уравнения для собственных векторов матрицы AA^T, а вторая — вид уравнения для собственных векторов матрицы A^TA.

$$\begin{aligned} \vec{h} &= \frac{1}{\lambda_h} AA^T \vec{h}, \\ \vec{a} &\leftarrow \frac{1}{\lambda_a} A^T A^T \vec{a} \end{aligned} \tag{21.11}$$

Здесь λ_h — собственное значение матрицы AA^T, а λ_a — собственное значение матрицы A^TA.

Отсюда следуют важные следствия.

1. Итеративные вычисления формулы (21.8) (или, что эквивалентно, формулы (21.9)), умноженные на соответствующие собственные значения, эквивалентны степенному методу для вычисления собственных значений матриц AA^T и A^TA. Поскольку главное собственное значение матрицы AA^T единственно, итеративно вычисленные элементы векторов $\vec{h}$ и $\vec{a}$ стремятся к стационарным значениям, определяемым матрицей A, а значит структурой связей в графе.
2. При вычислении этих собственных векторов мы не ограничены использованием степенным методом. Действительно, для вычисления главного собственного вектора стохастической матрицы можно применить любой быстрый метод.

Итак, вычисления принимают следующий вид.

1. Выбираем целевое подмножество веб-страниц, создаем веб-граф, индуцированный гиперссылками, и вычисляем матрицы AA^T и A^TA.
2. Вычисляем главные собственные векторы матриц AA^T и A^TA и создаем вектор показателей портальности $\vec{h}$ и показателей авторитетности $\vec{a}$.
3. Возвращаем веб-страницы, имеющие высокий показатель портальности и высокий показатель авторитетности.

Описанный метод анализа ссылок называется *методом HITS* (Hyperlink-Induced Topic Search).

Пример 21.2. Предположим, что пользователь отправил запрос jaguar, а ссылки, текст которых содержит слово запроса, имеют двойной вес. Матрица *A* на рис. 21.4 имеет следующий вид.

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 2 & 1 & 0 & 1 \end{pmatrix}$$

Векторы показателей портальности и авторитетности выглядят так.

$$\vec{h} = (0{,}03 \quad 0{,}04 \quad 0{,}33 \quad 0{,}18 \quad 0{,}04 \quad 0{,}04 \quad 0{,}35)$$

$$\vec{a} = (0{,}10 \quad 0{,}01 \quad 0{,}12 \quad 0{,}47 \quad 0{,}16 \quad 0{,}01 \quad 0{,}13)$$

Здесь страница q_3 имеет самый высокий авторитет — два портала (q_2 и q_6) имеют на нее ссылки с высоким весом, содержащими слово jaguar.

Поскольку итеративный метод, описанный выше, формализует интуитивные представления о хороших порталах и авторитетных источниках, страницы с высоким показателем, выведенные в списке результатов, являются хорошими порталами и авторитетными источниками из целевого подмножества веб-страниц. Остальные детали будут описаны в разделе 21.3.1 (в частности, как выбрать целевое подмножество веб-страниц по теме `leukemia`).

21.3.1. Выбор подмножества веб-страниц

Выбирая подмножество веб-страниц по теме `leukemia`, следует учитывать тот факт, что хорошие авторитетные страницы могут не содержать термина запроса `leukemia`. Как указано в разделе 21.1.1, это особенно ярко проявляется тогда, когда авторитетные страницы используют свое присутствие в вебе для поддержания определенного маркетингового имиджа. Например, многие страницы веб-сайта корпорации IBM являются авторитетными источниками информации о компьютерном аппаратном обеспечении, несмотря на то что эти страницы могут не содержать терминов `computer` и `hardware`. Однако порталы, хранящие ссылки на информационные ресурсы, посвященные аппаратному обеспечению, скорее всего, используют эти термины и в то же время содержат ссылки на релевантные страницы веб-сайта компании IBM.

Опираясь на приведенные выше рассуждения, опишем процедуру выбора подмножества страниц из веба, на основе которого будут вычисляться показатели портальности и авторитетности.

1. По заданному запросу (например, `leukemia`) с помощью текстового индекса находим страницы, содержащие слово `leukemia`. Назовем это множество страниц *корневым* (root set).
2. Построим *базовое множество* (base set) страниц, включающее в себя корневое множество, а также любую веб-страницу, которая либо сама ссылается на страницу из корневого множества, либо на нее ссылается какая-то страница из корневого множества.

Теперь будем использовать базовое множество для вычисления показателей портальности и показателей авторитетности. Базовое множество создается на основе следующих соображений.

1. Хорошая авторитетная страница может не содержать текст запроса (например, `computer hardware`).
2. Если текстовый запрос приводит к включению хорошей портальной страницы v_h в корневое множество, то включение всех страниц, на которые ссылаются страницы корневого множества, приведет к включению всех хороших авторитетных страниц, на которые ссылается страница v_h, в базовое множество.
3. И наоборот, если текстовый запрос приводит к включению хорошей авторитетной страницы v_a в корневое множество, то включение всех страниц, ссылающихся на v_a, приведет к включению всех хороших портальных страниц, связанных со

страницей v_h, в базовое множество. Иначе говоря, "расширение" корневого множества в базовое множество обогащает общий пул хороших портальных и авторитетных страниц.

Применение метода HITS к множеству запросов приводит к интересным выводам относительно анализа ссылок. Часто документы, находящиеся в начале списка портальных и авторитетных страниц, содержат фразы, написанные на языке, отличающемся от языка запроса. Эти страницы были предположительно включены в базовое множество из-за особенностей корневого множества. Таким образом, очевидны некоторые элементы *межъязыкового поиска* (в котором язык запроса отличается от языка документа). Интересно, что этот межъязыковой эффект является следствием исключительно анализа ссылок, без применения машинного перевода.

Завершим раздел некоторыми замечаниями о реализации этого алгоритма. Корневое множество состоит из всех страниц, соответствующих текстовому запросу; фактически большинство вариантов реализации этого алгоритма (раздел 21.4) предполагает, что для корневого множества достаточно около 200 веб-страниц, и не требуется включать в корневое множество все страницы, соответствующие текстовому запросу. Для вычисления векторов показателей портальности и авторитетности подходит любой алгоритм вычисления собственных векторов. В принципе, точные значения этих показателей вычислять необязательно. Достаточно лишь знать относительные величины этих показателей, чтобы идентифицировать первые элементы списка портальных и авторитетных страниц. Для этого иногда достаточно выполнить небольшое количество итераций степенного метода. Эксперименты показывают, что для получения достаточно хороших результатов достаточно примерно пяти итераций по формуле (21.8). Более того, из-за разреженности веб-графа (средняя веб-страница связана примерно с десятью другими), алгоритм выполняется не путем умножения матрицы на вектор, а с помощью последовательного применения формул (21.8).

На рис. 21.6 показаны результаты применения метода HITS к запросу `japan elementary schools`. На рисунке показаны первые портальные и авторитетные страницы. В каждой строке списка содержится тег `title` соответствующей HTML-страницы. Поскольку итоговая строка необязательно состоит из латинских символов, некоторые результаты нечитабельны (в данном случае, скорее всего, они соответствуют страницам на японском языке). Кроме того, в списке оказались страницы, созданные на других языках, что удивительно с учетом того, что запрос был сформулирован на английском. Этот факт является довольно характерным для метода HITS — после формирования корневого множества строка запроса (на английском языке) игнорируется. Базовое множество часто содержит документы, созданные на других языках, например англоязычная портальная страница ссылается на японские домашние страницы японских средних школ. Поскольку последующее вычисление первых элементов списка портальных и авторитетных страниц основано исключительно на анализе ссылок, среди них появятся страницы, созданные на иностранных языках.

Упражнение 21.19. Если все показатели портальности и авторитетности инициализированы единицей, чему они будут равны через одну итерацию?

Упражнение 21.20. Как интерпретировать элементы матриц AA^T и A^TA? Как они связаны с матрицей одновременного появления терминов CC^T из главы 18?

Упражнение 21.21. Чему равны главные собственные значения матриц AA^T и A^TA?

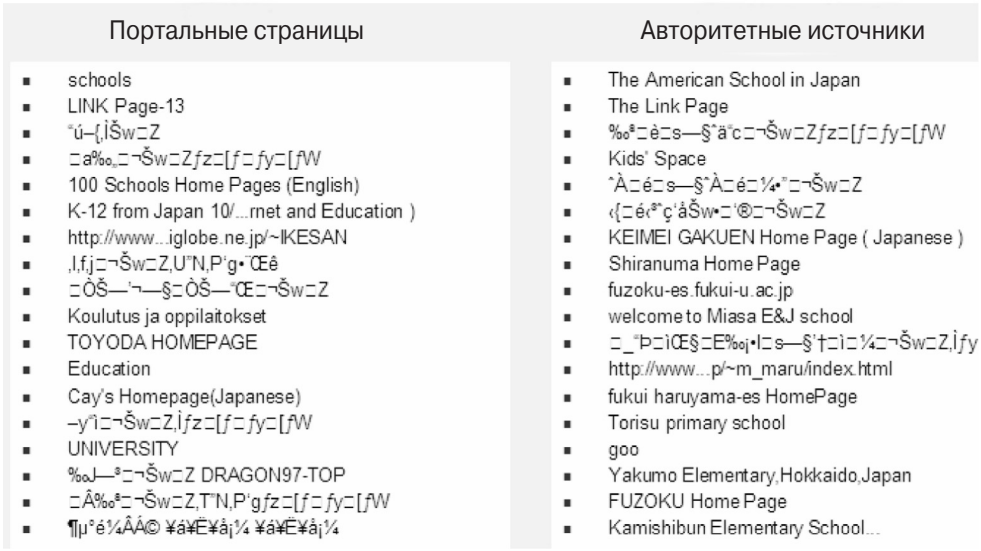


Рис. 21.6. Демонстрация метода HITS на примере запроса `japan elementary schools`

Упражнение 21.22. Вычислите вес PageRank, а также показатели портальности и авторитетности для каждой из трех страниц, показанных на рис. 21.7. Каков относительный порядок этих узлов в соответствии с тремя показателями (укажите равные значения).

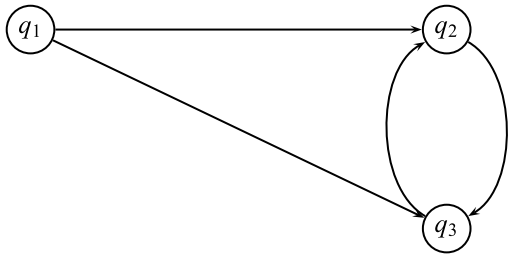


Рис. 21.7. Веб-граф для упражнения 21.22

PageRank: предположим, что на каждом этапе вычисления весов PageRank с помощью случайного блуждания мы перепрыгиваем на случайную страницу с вероятностью, равной 0,1, причем распределение страниц при телепортации является равномерным.

Порталы/авторитетные источники: нормализуем показатели портальности и авторитетности так, чтобы максимальный показатель портальности или авторитетности был равен единице.

Подсказка 1. При использовании свойства симметричности решение линейных уравнений может оказаться проще, чем при использовании итеративных методов.

Подсказка 2. Приведите относительный порядок всех трех узлов относительно всех трех показателей (указав на равные значения).

21.4. Библиография и рекомендации для дальнейшего чтения

Основы анализа цитирования заложил Гарфилд (Garfield, 1955). Впоследствии Пински и Нарин (Pinski and Narin, 1976) разработали *фактор влиятельности журнала* (journal influence weight), определение которого очень похоже на вес PageRank.

Использование якорного текста для поиска и ранжирования предложено в работе Макбрайана (McBryan, 1994). В его работе неявно использовался расширенный якорный текст. Результаты систематических экспериментов с якорным текстом приведены в статье Чакрабарти и др. (Chakrabarti et al., 1998).

Классическая книга по марковским цепям написана Кемени и Снелл (Kemeny and Snell, 1976). Вес PageRank был разработан Брином и Пейджем (Brin and Page, 1998). Он также описан в отчете Пейджа и др. (Page et al., 1998). Множество методов быстрого вычисления весов PageRank описано в работах Берхина (Berkhin, 2005) и Лангвилля и Мейера (Langville and Meyer, 2006). Во второй книге также показано, почему вычисление вектора весов PageRank можно свести к решению системы линейных уравнений, одному из решений упражнения (21.16). Влияние вероятности телепортации α исследовано в работах Баеза–Йейтса и др. (Baeza-Yates, 2005) и Болди и др. (Boldi et al., 2005). Тематический вес PageRank и его варианты описаны в работах Хавеливала (Haveliwala, 2002, 2003), а также Йе и Уидома (Jeh and Widom, 2003). Берхин (Berkhin, 2006*a*) разработал альтернативную точку зрения на тематический вес PageRank.

Нг и др. (Ng et al., 2001*b*) предположили, что оценка веса PageRank надежнее, чем оценки по методу HITS, в том смысле, что эти веса менее чувствительны к небольшим изменениям топологии графа. Однако они также заметили, что операция телепортации в этом смысле придает оценкам PageRank больше устойчивости. Методы PageRank и HITS могут подвергнуться атаке спама путем организованной вставки ссылок на веб-граф. Действительно, в вебе существуют *ссылочные фермы* (link farms), участвующие в сговоре для повышения показателей определенных страниц, вычисленных с помощью разных алгоритмов анализа ссылок.

Алгоритм HITS изобретен Клейнбергом (Kleinberg, 1999). Чакрабарти и др. (Chakrabarti et al., 1998) разработали варианты, которые взвешивают ссылки в итеративных вычислениях в зависимости от присутствия терминов запроса на страницах, и сравнили полученные результаты с результатами работы нескольких поисковых машин. Бхарат и Хенцингер (Bharat and Henzinger, 1998) усовершенствовали эти и другие эвристические приемы, показав, что некоторые их сочетания превосходят результаты базового алгоритма HITS. Бородин и др. (Borodin et al., 2001) выполнили систематическое исследование вариантов алгоритма HITS. Нг и др. (Ng et al., 2001*b*) ввели понятие *устойчивости* анализа ссылок, аргументируя это тем, что небольшие изменения топологии графа не должны приводить к значительным изменениям ранжированного списка результатов поиска по запросу. Многие авторы предложили свои варианты метода HITS. Вероятно, наиболее известен метод SALSA (Lempel and Moran, 2000).

Библиография

В библиографии используются следующие аббревиатуры названий журналов и конференций.

CACM	Communications of the Association for Computing Machinery (“Известия ACM”)
IP&M	Information Processing and Management (журнал “Обработка информации и информационное обслуживание”)
IR	Information Retrieval (журнал “Информационный поиск”)
JACM	Journal of the Association for Computing Machinery (журнал “ACM”)
JASIS	Journal of the American Society for Information Science (журнал Американского общества информатики)
JASIST	Journal of the American Society for Information Science and Technology (журнал Американского общества информатики и технологии)
JMLR	Journal of Machine Learning Research (журнал “Исследования в машинном обучениии”)
TOIS	ACM Transactions on Information Systems (журнал Ассоциации вычислительных машин “Успехи информационных систем”)
Proc. ACL	Proceedings of the Annual Meeting of the Association for Computational Linguistics (Материалы ежегодной встречи Ассоциации компьютерной лингвистики) Доступен по адресу `http://www.aclweb.org/anthology-index/`
Proc.CIKM	Proceedings of the Conference on Information and Knowledge Management (Труды конференции по информационному обслуживанию и управлению знаниями)
Proc.ECIR	Proceedings of the European Conference on Information Retrieval (Труды Европейской конференции по информационному поиску)
Proc. ECML	Proceedings of the European Conference on Machine Learning (Труды Европейской конференции по машинному обучению)
Proc. ICML	Proceedings of the International Conference on Machine Learning (Труды Международной конференции по машинному обучению)
Proc. IJCAI	Proceedings of the International Joint Conference on Artificial Intelligence (Труды Международной конференции по искусственному интеллекту)
Proc. INEX	Proceedings of the Initiative for the Evaluation of XML Retrieval (Материалы инициативы по оценке XML-поиска — INEX)
Proc. KDD	Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (Труды Международной конференции по выявлению знаний и извлечению данных)
Proc. NIPS	Proceedings of the Neural Information Processing Systems Conference (Труды конференции по нейронным системам обработки информации)
Proc. PODS	Proceedings of the ACM Conference on Principles of Database Systems (Труды конференции по принципам систем баз данных)

Proc. SDAIR Proceedings of the Annual Symposium on Document Analysis and Information Retrieval (Труды ежегодного симпозиума по анализу документов и информационному поиску)

Proc. SIGIR Proceedings of the Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval (Труды конференции по исследованиям и разработке в информационном поиске)
Доступен по адресу
`http://www.sigir.org/proceedings/Proc-Browse.html`

Proc. SPIRE Proceedings of the Symposium on String Processing and Information Retrieval (Труды конференции по обработке строк и информационному поиску)

Proc. TREC Proceedings of the Text Retrieval Conference (Труды конференции по поиску текстов)

Proc. UAI Proceedings of the Conference on Uncertainty in Artificial Intelligence (Труды конференции по проблеме неопределенности в искусственном интеллекте)

Proc. VLDB Proceedings of the Very Large Data Bases Conference (Труды конференции по сверхбольшим базам данных)

Proc. WWW Proceedings of the International WorldWide Web Conference (Труды Международной конференции WWW)

Proc. RCDL Proceedings of the Russian Conference on Research and Advanced Technology for Digital Libraries (Труды Всероссийской научной конференции по электронным библиотекам: перспективные методы и технологии, электронные коллекции)

Aberer, Karl. 2001. P-grid: A self-organizing access structure for P2P information systems. In *Proc. International Conference on Cooperative Information Systems*, pp. 179–194. Springer.

Aizerman, Mark A., Emmanuel M. Braverman, and Lev I. Rozonoer. 1964. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control* 25:821–837.

Akaike, Hirotugu. 1974. A new look at the statistical model identification. *IEEE Transactions on automatic control* 19(6):716–723.

Allan, James. 2005. HARD track overview in TREC 2005: High accuracy retrieval from documents. In *Proc. TREC*.

Allan, James, Ron Papka, and Victor Lavrenko. 1998. On-line new event detection and tracking. In *Proc. SIGIR*, pp. 37–45. ACM Press. doi: `http://doi.acm.org/10.1145/290941.290954`.

Allwein, Erin L., Robert E. Schapire, and Yoram Singer. 2000. Reducing multiclass to binary: A unifying approach for margin classifiers. *JMLR* 1:113–141. URL: `www.jmlr.org/papers/volume1/allwein00a/allwein00a.pdf`.

Alonso, Omar, Sandeepan Banerjee, and Mark Drake. 2006. GIO: A semantic web application using the information grid framework. In *Proc. WWW*, pp. 857–858. ACM Press. doi: `http://doi.acm.org/10.1145/1135777.1135913`.

Altingövde, Ismail Sengör, Engin Demir, Fazli Can, and Özgür Ulusoy. 2008. Incremental cluster-based retrieval using compressed cluster-skipping inverted files. *TOIS*. To appear.

Amer-Yahia, Sihem, Chavdar Botev, Jochen Dörre, and Jayavel Shanmugasundaram. 2006. XQuery full-text extensions explained. *IBM Systems Journal* 45(2):335–352.

Amer-Yahia, Sihem, Pat Case, Thomas Rölleke, Jayavel Shanmugasundaram, and Gerhard Weikum. 2005. Report on the DB/IR panel at SIGMOD 2005. *SIGMOD Record* 34(4):71–74. doi: `http://doi.acm.org/10.1145/1107499.1107514`.

Amer-Yahia, Sihem, and Mounia Lalmas. 2006. XML search: Languages, INEX and scoring. *SIGMOD Record* 35(4):16–23. doi: `http://doi.acm.org/10.1145/1228268.1228271`.

Anagnostopoulos, Aris, Andrei Z. Broder, and Kunal Punera. 2006. Effective and efficient classification on a search-engine model. In *Proc. CIKM*, pp. 208–217. ACM Press. doi: `http://doi.acm.org/10.1145/1183614.1183648`.

Anderberg, Michael R. 1973. *Cluster analysis for applications*. Academic Press.

Andoni, Alexandr, Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab Mirrokni. 2006. Locality-sensitive hashing using stable distributions. In *Nearest Neighbor Methods in Learning and Vision: Theory and Practice*. MIT Press.

Anh, Vo Ngoc, Owen de Kretser, and Alistair Moffat. 2001. Vector-space ranking with effective early termination. In *Proc. SIGIR*, pp. 35–42. ACM Press.

Anh, Vo Ngoc, and Alistair Moffat. 2005. Inverted index compression using word-aligned binary codes. *IR* 8(1):151–166. doi: `http://dx.doi.org/10.1023/B:INRT.0000048490.99518.5c`.

Anh, Vo Ngoc, and Alistair Moffat. 2006*a*. Improved word-aligned binary compression for text indexing. *IEEE Transactions on Knowledge and Data Engineering* 18(6):857–861.

Anh, Vo Ngoc, and Alistair Moffat. 2006*b*. Pruned query evaluation using precomputed impacts. In *Proc. SIGIR*, pp. 372–379. ACM Press. doi: `http://doi.acm.org/10.1145/1148170.1148235`.

Anh, Vo Ngoc, and Alistair Moffat. 2006c. Structured index organizations for highthroughput text querying. In *Proc. SPIRE*, pp. 304–315. Springer.

Apté, Chidanand, Fred Damerau, and Sholom M. Weiss. 1994. Automated learning of decision rules for text categorization. *TOIS* 12(1):233–251.

Arthur, David, and Sergei Vassilvitskii. 2006. How slow is the *k*-means method? In *Proc. Symposium on Computational Geometry*, pp. 144–153.

Arvola, Paavo, Marko Junkkari, and Jaana Kekäläinen. 2005. Generalized contextualization method for XML information retrieval. In *Proc. CIKM*, pp. 20–27. ACM Press.

Aslam, Javed A., and Emine Yilmaz. 2005. A geometric interpretation and analysis of R-precision. In *Proc. CIKM*, pp. 664–671. ACM Press.

Ault, Thomas Galen, and Yiming Yang. 2002. Information filtering in TREC-9 and TDT-3: A comparative analysis. *IR* 5(2–3):159–187.

Badue, Claudine Santos, Ricardo A. Baeza-Yates, Berthier Ribeiro-Neto, and Nivio Ziviani. 2001. Distributed query processing using partitioned inverted files. In *Proc. SPIRE*, pp. 10–20.

Baeza-Yates, Ricardo, Paolo Boldi, and Carlos Castillo. 2005. The choice of a damping function for propagating importance in link-based ranking. Technical report, Dipartimento di Scienze dell'Informazione, Universit`a degli Studi diMilano.

Baeza-Yates, Ricardo, and Berthier Ribeiro-Neto. 1999. *Modern Information Retrieval*. Addison-Wesley.

Bahle, Dirk, Hugh E.Williams, and Justin Zobel. 2002. Efficient phrase querying withan auxiliary index. In *Proc. SIGIR*, pp. 215–221. ACM Press.

Baldridge, Jason, and Miles Osborne. 2004. Active learning and the total cost of annotation. In *Proc. EMNLP*, pp. 9–16.

Ball, G. H. 1965. Data analysis in the social sciences: What about the details? In *Proc. Fall Joint Computer Conference*, pp. 533–560. Spartan Books.

Banko, Michele, and Eric Brill. 2001. Scaling to very very large corpora for natural language disambiguation. In *Proc. ACL*.

Bar-Ilan, Judit, and Tatyana Gutman. 2005. How do search engines respond to some non-English queries? *Journal of Information Science* 31(1):13–28.

Bar-Yossef, Ziv, and Maxim Gurevich. 2006. Random sampling from a search engine's index. In *Proc.WWW*, pp. 367–376. ACM Press. doi: `http://doi.acm.org/10.1145/1135777.1135833`.

Barroso, Luiz André, Jeffrey Dean, and Urs Hölzle. 2003. Web search for a planet: The Google cluster architecture. *IEEE Micro* 23(2):22–28. `http://dx.doi.org/10.1109/MM.2003.1196112`.

Bartell, Brian Theodore. 1994. *Optimizing ranking functions: A connectionist approach to adaptive information retrieval*. PhD thesis, University of California at San Diego, La Jolla, CA.

Bartell, Brian T., Garrison W. Cottrell, and Richard K. Belew. 1998. Optimizing similarity using multi-query relevance feedback. *JASIS* 49(8):742–761.

Barzilay, Regina, and Michael Elhadad. 1997. Using lexical chains for text summarization. In *Workshop on Intelligent Scalable Text Summarization*, pp. 10–17.

Bast, Holger, and Debapriyo Majumdar. 2005. Why spectral retrieval works. In *Proc. SIGIR*, pp. 11–18. ACM Press. doi: `http://doi.acm.org/10.1145/1076034.1076040`.

Basu, Sugato, Arindam Banerjee, and Raymond J. Mooney. 2004. Active semisupervision for pairwise constrained clustering. In *Proc. SIAM International Conference on Data Mining*, pp. 333–344.

Beesley, Kenneth R. 1998. Language identifier: A computer program for automatic natural-language identification of on-line text. In *Languages at Crossroads: Proceedings of the Annual Conference of the American Translators Association*, pp. 47–54.

Beesley, Kenneth R., and Lauri Karttunen. 2003. *Finite State Morphology*. CSLI Publications.

Bennett, Paul N. 2000. *Assessing the calibration of naive Bayes' posterior estimates.* Technical Report CMU-CS-00-155, School of Computer Science, Carnegie Mellon University.

Berger, Adam, and John Lafferty. 1999. Information retrieval as statistical translation. In *Proc. SIGIR*, pp. 222–229. ACM Press.

Berkhin, Pavel. 2005. A survey on pagerank computing. *Internet Mathematics* 2(1):73–120.

Berkhin, Pavel. 2006*a*. Bookmark-coloring algorithm for personalized pagerank computing. *Internet Mathematics* 3(1):41–62.

Berkhin, Pavel. 2006b. A survey of clustering datamining techniques. In Jacob Kogan, Charles Nicholas, andMarc Teboulle (eds.), *Grouping Multidimensional Data: Recent Advances in Clustering*, pp. 25–71. Springer.

Berners-Lee, Tim, Robert Cailliau, Jean-Francois Groff, and Bernd Pollermann. 1992. World-Wide Web: The information universe. *Electronic Networking: Research, Applications and Policy* 1(2):74–82. URL: `citeseer.ist.psu.edu/article/bernerslee92worldwide.html`.

Berry, Michael, and Paul Young. 1995. Using latent semantic indexing for multilanguage information retrieval. *Computers and the Humanities* 29(6):413–429.

Berry, Michael W., Susan T. Dumais, and GavinW. O'Brien. 1995. Using linear algebra for intelligent information retrieval. *SIAM Review* 37(4):573–595.

Betsi, Stamatina, Mounia Lalmas, Anastasios Tombros, and Theodora Tsikrika. 2006.
User expectations from XML element retrieval. In *Proc. SIGIR*, pp. 611–612. ACM Press.

Bharat, Krishna, and Andrei Broder. 1998. A technique for measuring the relative size and overlap of public web search engines. *Computer Networks and ISDN Systems* 30 (1–7): 379–388. doi: `http://dx.doi.org/10.1016/S0169-7552(98)00127-5`.

Bharat, Krishna, Andrei Broder, Monika Rauch Henzinger, Puneet Kumar, and Suresh Venkatasubramanian. 1998. The connectivity server: Fast access to linkage information on the web. In *Proc. WWW*, pp. 469–477.

Bharat, Krishna, Andrei Z. Broder, Jeffrey Dean, and Monika Rauch Henzinger. 2000. A comparison of techniques to find mirrored hosts on the WWW. *JASIS* 51(12): 1114–1122. URL: `citeseer.ist.psu.edu/bharat99comparison.html`.

Bharat, Krishna, and Monika Rauch Henzinger. 1998. Improved algorithms for topic distillation in a hyperlinked environment. In *Proc. SIGIR*, pp. 104–111. ACM Press. URL: `citeseer.ist.psu.edu/bharat98improved.html`.

Bishop, Christopher M. 2006. *Pattern Recognition and Machine Learning*. Springer.

Blair, David C., and M. E. Maron. 1985. An evaluation of retrieval effectiveness for a full-text document-retrieval system. *CACM* 28(3):289–299.

Blanco, Roi, and Alvaro Barreiro. 2006. TSP and cluster-based solutions to the reassignment of document identifiers. *IR* 9(4):499–517.

Blanco, Roi, and Alvaro Barreiro. 2007. Boosting static pruning of inverted files. In *Proc. SIGIR*. ACM Press.

Blandford, Dan, and Guy Blelloch. 2002. Index compression through document reordering. In *Proc. Data Compression Conference*, p. 342. IEEE Computer Society.

Blei, David M., AndrewY. Ng, and Michael I. Jordan. 2003. Latent Dirichlet allocation. *JMLR* 3:993–1022.

Boldi, Paolo, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna. 2002. Ubicrawler: A scalable fully distributed web crawler. In *Proc. Australian World Wide Web Conference*. URL: `citeseer.ist.psu.edu/article/boldi03ubicrawler.html`.

Boldi, Paolo, Massimo Santini, and Sebastiano Vigna. 2005. PageRank as a function of the damping factor. In *Proc. WWW*. URL: `citeseer.ist.psu.edu/boldi05pagerank.html`.

Boldi, Paolo, and Sebastiano Vigna. 2004a. Codes for the World-Wide Web. *Internet Mathematics* 2(4):405–427.

Boldi, Paolo, and Sebastiano Vigna. 2004b. TheWebGraph framework I: Compression techniques. In *Proc. WWW*, pp. 595–601. ACM Press.

Boldi, Paolo, and Sebastiano Vigna. 2005. Compressed perfect embedded skip lists for quick inverted-index lookups. In *Proc. SPIRE*. Springer.

Boley, Daniel. 1998. Principal direction divisive partitioning. *Data Mining and Knowledge Discovery* 2(4):325–344. doi: `http://dx.doi.org/10.1023/A:1009740529316`.

Borodin, Allan, Gareth O. Roberts, Jeffrey S. Rosenthal, and Panayiotis Tsaparas. 2001. Finding authorities and hubs from link structures on theWorldWideWeb. In *Proc. WWW*, pp. 415–429.

Bourne, Charles P., and Donald F. Ford. 1961. A study of methods for systematically abbreviating English, words and names. *JACM* 8(4):538–552. doi: http://doi.acm.org/10.1145/321088.321094.

Bradley, Paul S., and UsamaM. Fayyad. 1998. Refining initial points for k-means clustering. In *Proc. ICML*, pp. 91–99.

Bradley, Paul S., Usama M. Fayyad, and Cory Reina. 1998. Scaling clustering algorithms to large databases. In *Proc. KDD*, pp. 9–15.

Brill, Eric, and Robert C. Moore. 2000. An improved error model for noisy channel spelling correction. In *Proc. ACL*, pp. 286–293.

Brin, Sergey, and Lawrence Page. 1998. The anatomy of a large-scale hypertextual web search engine. In *Proc. WWW*, pp. 107–117.

Brisaboa, Nieves R., Antonio Fariña, Gonzalo Navarro, and José R. Paramá. 2007. Lightweight natural language text compression. *IR* 10(1):1–33.

Broder, Andrei. 2002. A taxonomy of web search. *SIGIR Forum* 36(2):3–10. doi: http://doi.acm.org/10.1145/792550.792552.

Broder, Andrei, S. Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. 2000. Graph structure in the web. *Computer Networks* 33(1):309–320.

Broder, Andrei Z., Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. 1997. Syntactic clustering of the web. In *Proc. WWW*, pp. 391–404.

Brown, EricW. 1995. *Execution Performance Issues in Full-Text Information Retrieval*. PhD thesis, University of Massachusets, Amherst. [137]

Buckley, Chris, James Allan, and Gerard Salton. 1994*a*. Automatic routing and ad-hoc retrieval using SMART: TREC 2. In *Proc. TREC*, pp. 45–55.

Buckley, Chris, and Gerard Salton. 1995. Optimization of relevance feedback weights. In *Proc. SIGIR*, pp. 351–357. ACM Press. doi: http://doi.acm.org/10.1145/215206.215383.

Buckley, Chris, Gerard Salton, and James Allan. 1994*b*. The effect of adding relevance information in a relevance feedback environment. In *Proc. SIGIR*, pp. 292–300. ACM Press.

Buckley, Chris, Amit Singhal, and Mandar Mitra. 1995. New retrieval approaches using SMART: TREC 4. In *Proc. TREC*.

Buckley, Chris, and EllenM. Voorhees. 2000. Evaluating evaluation measure stability. In *Proc. SIGIR*, pp. 33–40.

Burges, Chris, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to rank using gradient descent. In *Proc. ICML*.

Burges, Christopher J. C. 1998. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery* 2(2):121–167.

Burner, Mike. 1997. Crawling towards eternity: Building an archive of the World Wide Web. *Web Techniques Magazine* 2(5).

Burnham, Kenneth P., and David Anderson. 2002. *Model Selection and Multi-Model Inference*. Springer.

Bush, Vannevar. 1945. As we may think. *The Atlantic Monthly*. URL: www.theatlantic.com/doc/194507/bush.

Büttcher, Stefan, and Charles L. A. Clarke. 2005*a*. Indexing time vs. query time: Tradeoffs in dynamic information retrieval systems. In *Proc. CIKM*, pp. 317–318. ACM Press. doi: http://doi.acm.org/10.1145/1099554.1099645.

Büttcher, Stefan, and Charles L. A. Clarke. 2005*b*. A security model for full-text file system search in multi-user environments. In *FAST*. URL: www.usenix.org/events/fast05/tech/buettcher.html.

Büttcher, Stefan, and Charles L. A. Clarke. 2006. A document-centric approach to static index pruning in text retrieval systems. In *Proc. CIKM*, pp. 182–189. ACM Press. doi: http://doi.acm.org/10.1145/1183614.1183644.

Büttcher, Stefan, Charles L. A. Clarke, and Brad Lushman. 2006. Hybrid index maintenance for growing text collections. In *Proc. SIGIR*, pp. 356–363. ACM Press. doi: http://doi.acm.org/10.1145/1148170.1148233.

Cacheda, Fidel, Victor Carneiro, Carmen Guerrero, and Ángel Viña. 2003. Optimization of restricted searches in web directories using hybrid data structures. In *Proc. ECIR*, pp. 436–451.

Callan, Jamie. 2000. Distributed information retrieval. In W. Bruce Croft (ed.), *Advances in information retrieval*, pp. 127–150. Kluwer.

Can, Fazli, Ismail Sengör Altingövde, and Engin Demir. 2004. Efficiency and effectiveness of query processing in cluster-based retrieval. *Information Systems* 29(8): 697–717. doi: http://dx.doi.org/10.1016/S0306-4379(03)00062-0.

Can, Fazli, and Esen A. Ozkarahan. 1990. Concepts and effectiveness of the covercoefficient-based clustering methodology for text databases. *ACM Trans. Database Syst.* 15(4):483–517.

Cao, Guihong, Jian-Yun Nie, and Jing Bai. 2005. Integrating word relationships into language models. In *Proc. SIGIR*, pp. 298–305. ACM Press.

Cao, Yunbo, Jun Xu, Tie-Yan Liu, Hang Li, Yalou Huang, and Hsiao-Wuen Hon. 2006. Adapting Ranking SVM to document retrieval. In *Proc. SIGIR*. ACM Press.

Carbonell, Jaime, and Jade Goldstein. 1998. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proc. SIGIR*, pp. 335–336. ACM Press. doi: http://doi.acm.org/10.1145/290941.291025.

Carletta, Jean. 1996. Assessing agreement on classification tasks: The kappa statistic. *Computational Linguistics* 22:249–254.

Carmel, David, Doron Cohen, Ronald Fagin, Eitan Farchi, Michael Herscovici, Yoelle S. Maarek, and Aya Soffer. 2001. Static index pruning for information retrieval systems. In *Proc. SIGIR*, pp. 43–50. ACM Press. doi: http://doi.acm.org/10.1145/383952.383958.

Carmel, David, Yoelle S. Maarek, Matan Mandelbrod, Yosi Mass, and Aya Soffer. 2003. Searching XML documents via XML fragments. In *Proc. SIGIR*, pp. 151–158. ACM Press. doi: http://doi.acm.org/10.1145/860435.860464.

Caruana, Rich, and Alexandru Niculescu-Mizil. 2006. An empirical comparison of supervised learning algorithms. In *Proc. ICML*.

Castro, R. M., M. J. Coates, and R. D. Nowak. 2004. Likelihood based hierarchical clustering. *IEEE Transactions in Signal Processing* 52(8):2308–2321.

Cavnar, William B., and John M. Trenkle. 1994. N-gram-based text categorization. In *Proc. SDAIR*, pp. 161–175. [43]

Chakrabarti, Soumen. 2002. *Mining the Web: Analysis of Hypertext and Semi Structured Data*. Morgan Kaufman. [404]

Chakrabarti, Soumen, Byron Dom, David Gibson, Jon Kleinberg, Prabhakar Raghavan, and Sridhar Rajagopalan. 1998. Automatic resource list compilation by analyzing hyperlink structure and associated text. In *Proc. WWW*. URL: citeseer.ist.psu.edu/chakrabarti98automatic.html.

Chapelle, Olivier, Bernhard SchOlkopf, and Alexander Zien (eds.). 2006. *Semi-Supervised Learning*. MIT Press.

Chaudhuri, Surajit, Gautam Das, Vagelis Hristidis, and Gerhard Weikum. 2006. Probabilistic information retrieval approach for ranking of database query results. *ACM Trans. Database Syst.* 31(3):1134–1168. doi: http://doi.acm.org/10.1145/1166074.1166085.

Cheeseman, Peter, and John Stutz. 1996. Bayesian classification (AutoClass): Theory and results. In *Advances in Knowledge Discovery and Data Mining*, pp. 153–180. MIT Press.

Chen, Hsin-Hsi, and Chuan-Jie Lin. 2000. A multilingual news summarizer. In *Proc. COLING*, pp. 159–165.

Chen, Pai-Hsuen, Chih-Jen Lin, and Bernhard Schölkopf. 2005. A tutorial on ν-support vector machines. *Applied Stochastic Models in Business and Industry* 21:111–136.

Chiaramella, Yves, Philippe Mulhem, and Franck Fourel. 1996. A model for multimedia information retrieval. Technical Report 4–96, University of Glasgow.

Chierichetti, Flavio, Alessandro Panconesi, Prabhakar Raghavan, Mauro Sozio, Alessandro Tiberi, and Eli Upfal. 2007. Finding near neighbors through cluster pruning. In *Proc. PODS*. [137]

Cho, Junghoo, and Hector Garcia-Molina. 2002. Parallel crawlers. In *Proc. WWW*, pp. 124–135. ACM Press. doi: http://doi.acm.org/10.1145/511446.511464.

Cho, Junghoo, Hector Garcia-Molina, and Lawrence Page. 1998. Efficient crawling through URL ordering. In *Proc. WWW*, pp. 161–172.

Chu-Carroll, Jennifer, John Prager, Krzysztof Czuba, David Ferrucci, and Pablo Duboue. 2006. Semantic search via XML fragments: A high-precision approach to IR. In *Proc. SIGIR*, pp. 445–452. ACM Press. doi: http://doi.acm.org/10.1145/1148170.1148247.

Clarke, Charles L.A., Gordon V. Cormack, and Elizabeth A. Tudhope. 2000. Relevance ranking for one to three term queries. *IP&M* 36:291–311.

Cleverdon, Cyril W. 1991. The significance of the Cranfield tests on index languages. In *Proc. SIGIR*, pp. 3–12. ACM Press.

Coden, Anni R., Eric W. Brown, and Savitha Srinivasan (eds.). 2002. *Information Retrieval Techniques for Speech Applications*. Springer.

Cohen, Paul R. 1995. *Empirical Methods for Artificial Intelligence*. MIT Press.

Cohen, William W. 1998. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *Proc. ACM SIGMOD International Conference on Management of Data*, pp. 201–212. ACM Press.

Cohen, William W., Robert E. Schapire, and Yoram Singer. 1998. Learning to order things. In *Proc. NIPS*. The MIT Press. URL: citeseer.ist.psu.edu/article/cohen98learning.html.

Cohen, William W., and Yoram Singer. 1999. Context-sensitive learning methods for text categorization. *TOIS* 17(2):141–173.

Comtet, Louis. 1974. *Advanced Combinatorics*. Reidel.

Cooper, William S., Aitao Chen, and Fredric C. Gey. 1994. Full text retrieval based on probabilistic equations with coefficients fitted by logistic regression. In *Proc. TREC*, pp. 57–66. [138]

Cormen, Thomas H., Charles Eric Leiserson, and Ronald L. Rivest. 1990. *Introduction to Algorithms*. MIT Press.

Cover, Thomas M., and Peter E. Hart. 1967. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory* 13(1):21–27.

Cover, Thomas M., and Joy A. Thomas. 1991. *Elements of Information Theory*. Wiley.

Crammer, Koby, and Yoram Singer. 2001. On the algorithmic implementation of multiclass kernel-based machines. *JMLR* 2:265–292.

Creecy, Robert H., Brij M. Masand, Stephen J. Smith, and David L. Waltz. 1992.
Trading MIPS and memory for knowledge engineering. *CACM* 35(8):48–64. doi: `http://doi.acm.org/10.1145/135226.135228`.

Crestani, Fabio, Mounia Lalmas, Cornelis J. Van Rijsbergen, and Iain Campbell. 1998. Is this document relevant?. . . probably: A survey of probabilistic models in information retrieval. *ACM Computing Surveys* 30(4):528–552. doi: `http://doi.acm.org/10.1145/299917.299920`.

Cristianini, Nello, and John Shawe-Taylor. 2000. *Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge.

Croft, W. Bruce. 1978. A file organization for cluster-based retrieval. In *Proc. SIGIR*, pp. 65–82. ACM Press.

Croft, W. Bruce, and David J. Harper. 1979. Using probabilistic models of document retrieval without relevance information. *Journal of Documentation* 35(4):285–295.

Croft, W. Bruce, and John Lafferty (eds.). 2003. *Language Modeling for Information Retrieval*. Springer.

Crouch, Carolyn J. 1988. A cluster-based approach to thesaurus construction. In *Proc. SIGIR*, pp. 309–320. ACM Press. doi: `http://doi.acm.org/10.1145/62437.62467`.

Cucerzan, Silviu, and Eric Brill. 2004. Spelling correction as an iterative process that exploits the collective knowledge ofweb users. In *Proc. Empirical Methods in Natural Language Processing*.

Cutting, Douglas R., David R. Karger, and Jan O. Pedersen. 1993. Constant interaction-time Scatter/Gather browsing of very large document collections. In *Proc. SIGIR*, pp. 126–134. ACM Press.

Cutting, Douglas R., Jan O. Pedersen, David Karger, and John W. Tukey. 1992. Scatter/Gather: A cluster-based approach to browsing large document collections. In *Proc. SIGIR*, pp. 318–329. ACM Press.

Damerau, Fred J. 1964. A technique for computer detection and correction of spelling errors. *CACM* 7(3):171–176. doi: `http://doi.acm.org/10.1145/363958.363994`.

Davidson, Ian, and Ashwin Satyanarayana. 2003. Speeding up k-means clustering by bootstrap averaging. In *ICDM 2003 Workshop on Clustering Large Data Sets*.

Day, William H., and Herbert Edelsbrunner. 1984. Efficient algorithms for agglomerative hierarchical clustering methods. *Journal of Classification* 1:1–24.

de Moura, Edleno Silva, Gonzalo Navarro, Nivio Ziviani, and Ricardo Baeza-Yates. 2000. Fast and flexible word searching on compressed text. *TOIS* 18(2):113–139. doi:`http://doi.acm.org/10.1145/348751.348754`.

Dean, Jeffrey, and Sanjay Ghemawat. 2004. MapReduce: Simplified data processing on large clusters. In *Symposium on Operating System Design and Implementation*.

Deerwester, Scott, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *JASIS* 41(6):391–407.

del Bimbo, Alberto. 1999. *Visual Information Retrieval*. Morgan Kaufmann.

Dempster, A.P., N.M. Laird, and D.B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society Series B* 39:1–38.

Dhillon, Inderjit S. 2001. Co-clustering documents and words using bipartite spectralgraph partitioning. In *Proc. KDD*, pp. 269–274.

Dhillon, Inderjit S., and Dharmendra S. Modha. 2001. Concept decompositions for large sparse text data using clustering. *Machine Learning* 42(1/2):143–175. doi: http://dx.doi.org/10.1023/A:1007612920971.

Di Eugenio, Barbara, and Michael Glass. 2004. The kappa statistic: A second look. *Computational Linguistics* 30(1):95–101. doi: http://dx.doi.org/10.1162/089120104773633402.

Dietterich, Thomas G. 2002. Ensemble learning. In Michael A. Arbib (ed.), *The Handbook of Brain Theory and Neural Networks*. 2nd edition. MIT Press. [319]

Dietterich, Thomas G., and Ghulum Bakiri. 1995. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research* 2:263–286. [292]

Dom, Byron E. 2002. An information-theoretic external cluster-validity measure. In *Proc. UAI*.

Domingos, Pedro. 2000. A unified bias-variance decomposition for zero-one and squared loss. In *Proc. National Conference on Artificial Intelligence and Proc. Conference Innovative Applications of Artificial Intelligence*, pp. 564–569. AAAI Press/The MIT Press.

Domingos, Pedro, and Michael J. Pazzani. 1997. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning* 29(2-3):103–130. URL: citeseer.ist.psu.edu/domingos97optimality.html.

Downie, J. Stephen. 2006. The Music Information Retrieval Evaluation eXchange (MIREX). *D-Lib Magazine* 12(12).

Duda, Richard O., Peter E. Hart, and David G. Stork. 2000. *Pattern Classification*, 2nd edition.Wiley-Interscience. [264, 343]

Dumais, Susan, John Platt, David Heckerman, and Mehran Sahami. 1998. Inductive learning algorithms and representations for text categorization. In *Proc. CIKM*, pp. 148–155. ACM Press. doi: http://doi.acm.org/10.1145/288627.288651.

Dumais, Susan T. 1993. Latent semantic indexing (LSI) and TREC-2. In *Proc. TREC*, pp. 105–115.

Dumais, Susan T. 1995. Latent semantic indexing (LSI): TREC-3 report. In *Proc. TREC*, pp. 219–230.

Dumais, Susan T., and Hao Chen. 2000. Hierarchical classification of Web content. In *Proc. SIGIR*, pp. 256–263. ACM Press.

Dunning, Ted. 1993. Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics* 19(1):61–74.

Dunning, Ted. 1994. *Statistical identification of language*. Technical Report 94-273, Computing Research Laboratory, New Mexico State University.

Eckart, Carl, and Gale Young. 1936. The approximation of a matrix by another of lower rank. *Psychometrika* 1:211–218.

El-Hamdouchi, Abdelmoula, and PeterWillett. 1986. Hierarchic document classification usingWard's clustering method. In *Proc. SIGIR*, pp. 149–156. ACM Press. doi: http://doi.acm.org/10.1145/253168.253200.

Elias, Peter. 1975. Universal code word sets and representations of the integers. *IEEE Transactions on Information Theory* 21(2):194–203.

Eyheramendy, Susana, David Lewis, and David Madigan. 2003. On the Naive Bayes model for text categorization. In *Proc. International Workshop on Artificial Intelligence and Statistics*. Society for Artificial Intelligence and Statistics.

Fallows, Deborah, 2004. The internet and daily life. URL: www.pewinternet.org/pdfs/PIPInternet and Daily Life.pdf. Pew/Internet and American Life Project.

Fayyad, Usama M., Cory Reina, and Paul S. Bradley. 1998. Initialization of iterative refinement clustering algorithms. In *Proc. KDD*, pp. 194–198.

Fellbaum, Christiane D. 1998. *WordNet — An Electronic Lexical Database*. MIT Press.

Ferragina, Paolo, and Rossano Venturini. 2007. Compressed permuterm indexes. In *Proc. SIGIR*. ACM Press. [59]

Forman, George. 2004. A pitfall and solution in multi-class feature selection for text classification. In *Proc. ICML*.

Forman, George. 2006. Tackling concept drift by temporal inductive transfer. In *Proc. SIGIR*, pp. 252–259. ACM Press. doi: `http://doi.acm.org/10.1145/1148170.1148216`.

Forman, George, and Ira Cohen. 2004. Learning from little: Comparison of classifiers given little training. In *PKDD*, pp. 161–172.

Fowlkes, Edward B., and Colin L. Mallows. 1983. A method for comparing two hierarchical clusterings. *Journal of the American Statistical Association* 78(383):553–569. URL: `www.jstor.org/view/01621459/di985957/98p0926l/0`.

Fox, Edward A., and Whay C. Lee. 1991. *FAST-INV: A fast algorithm for building large inverted files*. Technical report, Virginia Polytechnic Institute & State University, Blacksburg, VA, USA.

Fraenkel, Aviezri S., and Shmuel T. Klein. 1985. Novel compression of sparse bit-strings — Preliminary report. In *Combinatorial Algorithms on Words, NATO ASI Series Vol F12*, pp. 169–183. Springer.

Frakes, William B., and Ricardo Baeza-Yates (eds.). 1992. *Information Retrieval: Data Structures and Algorithms*. Prentice-Hall.

Fraley, Chris, and Adrian E. Raftery. 1998. How many clusters? Which clustering method? Answers via model-based cluster analysis. *Computer Journal* 41(8):578–588.

Friedl, Jeffrey E. F. 2006. *Mastering Regular Expressions*, 3rd edition. O'Reilly.

Friedman, Jerome H. 1997. On bias, variance, 0/1–loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery* 1(1):55–77.

Friedman, Nir, and Moises Goldszmidt. 1996. Building classifiers using bayesian networks. In *Proc. National Conference on Artificial Intelligence*, pp. 1277–1284.

Fuhr, Norbert. 1989. Optimum polynomial retrieval functions based on the probability ranking principle. *TOIS* 7(3):183–204.

Fuhr, Norbert. 1992. Probabilistic models in information retrieval. *Computer Journal* 35 (3):243–255.

Fuhr, Norbert, Norbert Gövert, Gabriella Kazai, and Mounia Lalmas (eds.). 2003*a*. *INitiative for the Evaluation of XML Retrieval (INEX). Proc. First INEX Workshop*. ERCIM.

Fuhr, Norbert, and Kai Großjohann. 2004. XIRQL: An XML query language based on information retrieval concepts. *TOIS* 22(2):313–356. URL: `http://doi.acm.org/10.1145/984321.984326`.

Fuhr, Norbert, and Mounia Lalmas. 2007. Advances in XML retrieval: The INEX initiative. In *Proc. International Workshop on Research Issues in Digital Libraries*.

Fuhr, Norbert, Mounia Lalmas, Saadia Malik, and Gabriella Kazai (eds.). 2006. *Advances in XML Information Retrieval and Evaluation, 4th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2005*. Springer.

Fuhr, Norbert, Mounia Lalmas, Saadia Malik, and Zoltán Szlávik (eds.). 2005. *Advances in XML Information Retrieval, Third International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2004*. Springer.

Fuhr, Norbert, Mounia Lalmas, and Andrew Trotman (eds.). 2007. *Comparative Evaluation of XML Information Retrieval Systems, 5th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2006*. Springer.

Fuhr, Norbert, Saadia Malik, and Mounia Lalmas (eds.). 2003b. *INEX 2003 Workshop Proceedings*. URL: `http://inex.is.informatik.uni-duisburg.de:2003/proceedings.pdf`.

Fuhr, Norbert, and Ulrich Pfeifer. 1994. Probabilistic information retrieval as a combination of abstraction, inductive learning, and probabilistic assumptions. *TOIS* 12 (1):92–115. doi: `http://doi.acm.org/10.1145/174608.174612`.

Fuhr, Norbert, and Thomas Rölleke. 1997. A probabilistic relational algebra for the integration of information retrieval and database systems. *TOIS* 15(1):32–66. doi: `http://doi.acm.org/10.1145/239041.239045`.

Gaertner, Thomas, John W. Lloyd, and Peter A. Flach. 2002. Kernels for structured data. In *International Conference on Inductive Logic Programming*, pp. 66–83.

Gao, Jianfeng, Mu Li, Chang-Ning Huang, and Andi Wu. 2005. Chinese word segmentation and named entity recognition: A pragmatic approach. *Computational Linguistics* 31(4):531–574.

Gao, Jianfeng, Jian-Yun Nie, Guangyuan Wu, and Guihong Cao. 2004. Dependence language model for information retrieval. In *Proc. SIGIR*, pp. 170–177. ACM Press.

Garcia, Steven, Hugh E.Williams, and Adam Cannane. 2004. Access-ordered indexes. In *Proc. Australasian conference on Computer science*, pp. 7–14.

Garcia-Molina, Hector, Jennifer Widom, and Jeffrey D. Ullman. 1999. *Database System Implementation*. Prentice-Hall.

Garfield, Eugene. 1955. Citation indexes to science: A new dimension in documentation through association of ideas. *Science* 122:108–111.

Garfield, Eugene. 1976. The permuterm subject index: An autobiographic review. *JASIS* 27(5–6):288–291.

Geman, Stuart, Elie Bienenstock, and René Doursat. 1992. Neural networks and the bias/variance dilemma. *Neural Computation* 4(1):1–58.

Geng, Xiubo, Tie-Yan Liu, Tao Qin, and Hang Li. 2007. Feature selection for ranking. In *Proc. SIGIR*, pp. 407–414. ACM Press.

Gerrand, Peter. 2007. Estimating linguistic diversity on the internet: A taxonomy to avoid pitfalls and paradoxes. *Journal of Computer-Mediated Communication* 12(4). URL: `http://jcmc.indiana.edu/vol12/issue4/gerrand.html`. article 8.

Gey, Fredric C. 1994. Inferring probability of relevance using the method of logistic regression. In *Proc. SIGIR*, pp. 222–231. ACM Press.

Ghamrawi, Nadia, and Andrew McCallum. 2005. Collective multi-label classification. In *Proc. CIKM*, pp. 195–200. ACM Press. doi: `http://doi.acm.org/10.1145/1099554.1099591`.

Glover, Eric, David M. Pennock, Steve Lawrence, and Robert Krovetz. 2002*a*. Inferring hierarchical descriptions. In *Proc. CIKM*, pp. 507–514. ACM Press. doi: `http://doi.acm.org/10.1145/584792.584876`.

Glover, Eric J., Kostas Tsioutsiouliklis, Steve Lawrence, David M. Pennock, and Gary W. Flake. 2002*b*. Using web structure for classifying and describing web pages. In *Proc. WWW*, pp. 562–569. ACM Press. doi: `http://doi.acm.org/10.1145/511446.511520`.

Gövert, Norbert, and Gabriella Kazai. 2003. Overview of the INitiative for the Evaluation of XML retrieval (INEX) 2002. In Fuhr et al. (2003b), pp. 1–17. URL: `http://inex.is.informatik.uni-duisburg.de:2003/proceedings.pdf`.

Grabs, Torsten, and Hans-Jörg Schek. 2002. Generating vector spaces on-the-fly for flexible XML retrieval. In *XML and Information Retrieval Workshop at SIGIR 2002*.

Greiff, Warren R. 1998. A theory of term weighting based on exploratory data analysis. In *Proc. SIGIR*, pp. 11–19. ACM Press.

Grinstead, Charles M., and J. Laurie Snell. 1997. *Introduction to Probability*, 2nd edition. American Mathematical Society. URL: `www.dartmouth.edu/.chance/teachingaids/booksarticles/probability book/amsbook.mac.pdf`.

Grossman, David A., and Ophir Frieder. 2004. *Information Retrieval: Algorithms and Heuristics*, 2nd edition. Springer.

Gusfield, Dan. 1997. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press. (Русский перевод: Гасфилд Д. Строки, деревья и последовательности в алгоритмах. Информатика и вычислительная биология. — СПб.: Невский диалект, БХВ-Петербург, 2003. — 656 с.)

Hamerly, Greg, and Charles Elkan. 2003. Learning the *k* in *k*-means. In *NIPS*. URL: `http://books.nips.cc/papers/files/nips16/NIPS2003AA36.pdf`.

Han, Eui-Hong, and George Karypis. 2000. Centroid-based document classification: Analysis and experimental results. In *PKDD*, pp. 424–431.

Hand, David J. 2006. Classifier technology and the illusion of progress. *Statistical Science* 21:1–14.

Hand, David J., and Keming Yu. 2001. Idiot's Bayes: Not so stupid after all. *International Statistical Review* 69(3):385–398.

Harman, Donna. 1991. How effective is suffixing? *JASIS* 42:7–15.

Harman, Donna. 1992. Relevance feedback revisited. In *Proc. SIGIR*, pp. 1–10. ACM Press.

Harman, Donna, Ricardo Baeza-Yates, Edward Fox, and W. Lee. 1992. Inverted files. In Frakes and Baeza-Yates (1992), pp. 28–43.

Harman, Donna, and Gerald Candela. 1990. Retrieving records from a gigabyte of text on a minicomputer using statistical ranking. *JASIS* 41(8):581–589.

Harold, Elliotte Rusty, and Scott W. Means. 2004. *XML in a Nutshell*, 3rd edition. O'Reilly.

Harter, Stephen P. 1998. Variations in relevance assessments and the measurement of retrieval effectiveness. *JASIS* 47:37–49.

Hartigan, J. A., and M. A.Wong. 1979. A K-means clustering algorithm. *Applied Statistics* 28:100–108.

Hastie, Trevor, Robert Tibshirani, and Jerome H. Friedman. 2001. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer.

Hatzivassiloglou, Vasileios, Luis Gravano, and Ankineedu Maganti. 2000. An investigation of linguistic features and clustering algorithms for topical document clustering. In *Proc. SIGIR*, pp. 224–231. ACM Press. doi: `http://doi.acm.org/10.1145/345508.345582`.

Haveliwala, Taher. 2003. Topic-sensitive PageRank: A context-sensitive ranking algorithm for web search. *IEEE Transactions on Knowledge and Data Engineering* 15(4): 784–796. URL: `citeseer.ist.psu.edu/article/haveliwala03topicsensitive.html`.

Haveliwala, Taher H. 2002. Topic-sensitive PageRank. In *Proc. WWW*. URL: `citeseer.ist.psu.edu/haveliwala02topicsensitive.html`.

Hayes, Philip J., and Steven P. Weinstein. 1990. CONSTRUE/TIS: A system for content-based indexing of a database of news stories. In *Proc. Conference on Innovative Applications of Artificial Intelligence*, pp. 49–66.

Heaps, Harold S. 1978. *Information Retrieval: Computational and Theoretical Aspects*. Academic Press.

Hearst, Marti A. 1997. TextTiling: Segmenting text into multi-paragraph subtopic passages. *Computational Linguistics* 23(1):33–64.

Hearst, Marti A. 2006. Clustering versus faceted categories for information exploration. *CACM* 49(4):59–61. doi: `http://doi.acm.org/10.1145/1121949.1121983`.

Hearst, Marti A., and Jan O. Pedersen. 1996. Reexamining the cluster hypothesis. In *Proc. SIGIR*, pp. 76–84. ACM Press.

Hearst, Marti A., and Christian Plaunt. 1993. Subtopic structuring for full-length document access. In *Proc. SIGIR*, pp. 59–68. ACM Press. doi: `http://doi.acm.org/10.1145/160688.160695`.

Heinz, Steffen, and Justin Zobel. 2003. Efficient single-pass index construction for text databases. *JASIST* 54(8):713–729. doi: `http://dx.doi.org/10.1002/asi.10268`.

Heinz, Steffen, Justin Zobel, and Hugh E. Williams. 2002. Burst tries: A fast, efficient data structure for string keys. *TOIS* 20(2):192–223. doi: `http://doi.acm.org/10.1145/506309.506312`.

Henzinger, Monika Rauch, Allan Heydon, Michael Mitzenmacher, and Marc Najork. 2000. On near-uniform URL sampling. In *Proc. WWW*, pp. 295–308. North-Holland. doi: `http://dx.doi.org/10.1016/S1389-1286(00)00055-4`.

Herbrich, Ralf, Thore Graepel, and Klaus Obermayer. 2000. Large margin rank boundaries for ordinal regression. In *Advances in Large Margin Classifiers*, pp. 115–132. MIT Press.

Hersh, William, Chris Buckley, T. J. Leone, and David Hickam. 1994. OHSUMED: An interactive retrieval evaluation and new large test collection for research. In *Proc. SIGIR*, pp. 192–201. ACM Press.

Hersh, William R., Andrew Turpin, Susan Price, Benjamin Chan, Dale Kraemer, Lynetta Sacherek, and Daniel Olson. 2000a. Do batch and user evaluation give the same results? In *Proc. SIGIR*, pp. 17–24.

Hersh, William R., Andrew Turpin, Susan Price, Dale Kraemer, Daniel Olson, Benjamin Chan, and Lynetta Sacherek. 2001. Challenging conventional assumptions of automated information retrieval with real users: Boolean searching and batch retrieval evaluations. *IP&M* 37(3):383–402.

Hersh, William R., Andrew Turpin, Lynetta Sacherek, Daniel Olson, Susan Price, Benjamin Chan, and Dale Kraemer. 2000b. Further analysis of whether batch and user evaluations give the same results with a question-answering task. In *Proc. TREC*. Hiemstra, Djoerd. 1998. A linguistically motivated probabilistic model of information retrieval. In *Proc. ECDL*, pp. 569–584.

Hiemstra, Djoerd. 2000. A probabilistic justification for using tf.idf term weighting in information retrieval. *International Journal on Digital Libraries* 3(2):131–139.

Hiemstra, Djoerd, and Wessel Kraaij. 2005. A language-modeling approach to TREC. In Voorhees and Harman (2005), pp. 373–395.

Hirai, Jun, Sriram Raghavan, Hector Garcia-Molina, and Andreas Paepcke. 2000. WebBase: A repository of web pages. In *Proc. WWW*, pp. 277–293.

Hofmann, Thomas. 1999a. Probabilistic Latent Semantic Indexing. In *UAI*. URL: `citeseer.ist.psu.edu/hofmann99probabilistic.html`.

Hofmann, Thomas. 1999b. Probabilistic Latent Semantic Indexing. In *Proc. SIGIR*, pp. 50–57. ACM Press. URL: citeseer.ist.psu.edu/article/hofmann99probabilistic.html.

Hollink, Vera, Jaap Kamps, Christof Monz, and Maarten de Rijke. 2004. Monolingual document retrieval for European languages. *IR* 7(1):33–52.

Hopcroft, John E., Rajeev Motwani, and Jeffrey D. Ullman. 2000. *Introduction to Automata Theory, Languages, and Computation*, 2nd edition. Addison Wesley.

Huang, Yifen, and Tom M. Mitchell. 2006. Text clustering with extended user feedback. In *Proc. SIGIR*, pp. 413–420. ACM Press. doi: http://doi.acm.org/10.1145/1148170.1148242.

Hubert, Lawrence, and Phipps Arabie. 1985. Comparing partitions. *Journal of Classification* 2:193–218.

Hughes, Baden, Timothy Baldwin, Steven Bird, Jeremy Nicholson, and Andrew MacKinlay. 2006. Reconsidering language identification for written language resources. In *International Conference on Language Resources and Evaluation*, pp. 485–488.

Hull, David. 1993. Using statistical testing in the evaluation of retrieval performance. In *Proc. SIGIR*, pp. 329–338. ACM Press.

Hull, David. 1996. Stemming algorithms — A case study for detailed evaluation. *JASIS* 47(1):70–84.

Ide, E. 1971. New experiments in relevance feedback. In Salton (1971b), pp. 337–354.

Indyk, Piotr. 2004. Nearest neighbors in high-dimensional spaces. In J. E. Goodman and J. O'Rourke (eds.), *Handbook of Discrete and Computational Geometry*, 2nd edition. pp. 877–892. Chapman and Hall/CRC Press.

Ingwersen, Peter, and Kalervo Järvelin. 2005. *The Turn: Integration of Information Seeking and Retrieval in Context*. Springer.

Ittner, David J., David D. Lewis, and David D. Ahn. 1995. Text categorization of low quality images. In *Proc. Annual Symposium on Document Analysis and Information Retrieval*, pp. 301–315.

Iwayama, Makoto, and Takenobu Tokunaga. 1995. Cluster-based text categorization: A comparison of category search strategies. In *Proc. SIGIR*, pp. 273–280. ACM Press.

Jackson, Peter, and Isabelle Moulinier. 2002. *Natural Language Processing for Online Applications: Text Retrieval, Extraction and Categorization*. John Benjamins.

Jacobs, Paul S., and Lisa F. Rau. 1990. SCISOR: Extracting information from on-line news. *CACM* 33:88–97.

Jain, Anil, M. Narasimha Murty, and Patrick Flynn. 1999. Data clustering: A review. *ACM Computing Surveys* 31(3):264–323.

Jain, Anil K., and Richard C. Dubes. 1988. *Algorithms for Clustering Data*. Prentice-Hall.

Jardine, N., and C. J. van Rijsbergen. 1971. The use of hierarchic clustering in information retrieval. *Information Storage and Retrieval* 7:217–240.

Järvelin, Kalervo, and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *TOIS* 20(4):422–446.

Jeh, Glen, and Jennifer Widom. 2003. Scaling personalized web search. In *Proc.WWW*, pp. 271–279. ACM Press.

Jensen, Finn V., and Finn B. Jensen. 2001. *Bayesian Networks and Decision Graphs*. Springer. [215]

Jeong, Byeong-Soo, and Edward Omiecinski. 1995. Inverted file partitioning schemes in multiple disk systems. *IEEE Transactions on Parallel and Distributed Systems* 6(2): 142–153.

Ji, Xiang, and Wei Xu. 2006. Document clustering with prior knowledge. In *Proc. SIGIR*, pp. 405–412. ACM Press. doi: http://doi.acm.org/10.1145/1148170.1148241.

Jing, Hongyan. 2000. Sentence reduction for automatic text summarization. In *Proc. Conference on applied natural language processing*, pp. 310–315.

Joachims, Thorsten. 1997. A probabilistic analysis of the Rocchio algorithm with tfidf for text categorization. In *Proc. ICML*, pp. 143–151. Morgan Kaufmann.

Joachims, Thorsten. 1998. Text categorization with support vector machines: Learning with many relevant features. In *Proc. ECML*, pp. 137–142. Springer.

Joachims, Thorsten. 1999. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola (eds.), *Advances in Kernel Methods–Support Vector Learning*. MIT Press.

Joachims, Thorsten. 2002a. *Learning to Classify Text Using Support Vector Machines*. Kluwer.

Joachims, Thorsten. 2002b. Optimizing search engines using clickthrough data. In *Proc. KDD*, pp. 133–142.

Joachims, Thorsten. 2006a. Training linear SVMs in linear time. In *Proc. KDD*, pp. 217–226. ACM Press. doi: http://doi.acm.org/10.1145/1150402.1150429.

Joachims, Thorsten. 2006b. Transductive support vector machines. In Chapelle et al. (2006), pp. 105–118.

Joachims, Thorsten, Laura Granka, Bing Pan, Helene Hembrooke, and Geri Gay. 2005. Accurately interpreting clickthrough data as implicit feedback. In *Proc. SIGIR*, pp. 154–161. ACM Press.

Johnson, David, Vishv Malhotra, and Peter Vamplew. 2006. More effective web search using bigrams and trigrams. *Webology* 3(4). URL: www.webology.ir/2006/v3n4/a35.html.

Jurafsky, Dan, and James H.Martin. 2008. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*, 2nd edition. Prentice-Hall.

Käki, Mika. 2005. Findex: Search result categories help users when document ranking fails. In *Proc. SIGCHI Conference on Human Factors in Computing Systems*, pp. 131–140. ACM Press. doi: http://doi.acm.org/10.1145/1054972.1054991.

Kammenhuber, Nils, Julia Luxenburger, Anja Feldmann, and Gerhard Weikum. 2006. Web search clickstreams. In *ACM SIGCOMM on Internet Measurement*, pp. 245–250. ACM Press.

Kamps, Jaap, Maarten de Rijke, and Börkur Sigurbjörnsson. 2004. Length normalization in XML retrieval. In *Proc. SIGIR*, pp. 80–87. ACM Press. doi: http://doi.acm.org/10.1145/1008992.1009009.

Kamps, Jaap, Maarten Marx, Maarten de Rijke, and Börkur Sigurbjörnsson. 2006. Articulating information needs in XML query languages. *TOIS* 24(4):407–436. doi: http://doi.acm.org/10.1145/1185877.1185879.

Kamvar, Sepandar D., Dan Klein, and Christopher D. Manning. 2002. Interpreting and extending classical agglomerative clustering algorithms using a model-based approach. In *Proc. ICML*, pp. 283–290. Morgan Kaufmann.

Kannan, Ravi, Santosh Vempala, and Adrian Vetta. 2000. On clusterings — Good, bad and spectral. In *Proc. Annual Symposium on Foundations of Computer Science*, p. 367. IEEE Computer Society.

Kaszkiel, Marcin, and Justin Zobel. 1997. Passage retrieval revisited. In *Proc. SIGIR*, pp. 178–185. ACM Press. doi: http://doi.acm.org/10.1145/258525.258561.

Kaufman, Leonard, and Peter J. Rousseeuw. 1990. *Finding groups in data*. Wiley.

Kazai, Gabriella, and Mounia Lalmas. 2006. eXtended cumulated gain measures for the evaluation of content-oriented XML retrieval. *TOIS* 24(4):503–542. doi: http://doi.acm.org/10.1145/1185883.

Kekäläinen, Jaana. 2005. Binary and graded relevance in IR evaluations — Comparison of the effects on ranking of IR systems. *IP&M* 41:1019–1033.

Kekäläinen, Jaana, and Kalervo Järvelin. 2002. Using graded relevance assessments in IR evaluation. *JASIST* 53(13):1120–1129.

Kemeny, John G., and J. Laurie Snell. 1976. *Finite Markov Chains*. Springer.

Kent, Allen, Madeline M. Berry, Fred U. Luehrs, Jr., and J. W. Perry. 1955. Machine literature searching VIII. Operational criteria for designing information retrieval systems. *American Documentation* 6(2):93–101.

Kernighan, Mark D., Kenneth W. Church, and William A. Gale. 1990. A spelling correction program based on a noisy channel model. In *Proc. ACL*, pp. 205–210.

King, Benjamin. 1967. Step-wise clustering procedures. *Journal of the American Statistical Association* 69:86–101.

Kishida, Kazuaki, Kuang Hua Chen, Sukhoon Lee, Kazuko Kuriyama, Noriko Kando, Hsin-Hsi Chen, and Sung Hyon Myaeng. 2005. Overview of CLIR task at the fifth NTCIR workshop. In *Proc. NTCIR Workshop Meeting on Evaluation of Information Access Technologies: Information Retrieval, Question Answering and Cross-Lingual Information Access*. National Institute of Informatics.

Klein, Dan, and Christopher D. Manning. 2002. Conditional structure versus conditional estimation in NLP models. In *Proc. Empirical Methods in Natural Language Processing*, pp. 9–16.

Kleinberg, Jon M. 1997. Two algorithms for nearest-neighbor search in high dimensions. In *Proc. Annual ACM Symposium on Theory of Computing*, pp. 599–608. ACM Press. doi: http://doi.acm.org/10.1145/258533.258653.

Kleinberg, Jon M. 1999. Authoritative sources in a hyperlinked environment. *JACM* 46(5):604–632. URL: citeseer.ist.psu.edu/article/kleinberg98authoritative.html.

Kleinberg, Jon M. 2002. An impossibility theorem for clustering. In *Proc. NIPS*.

Knuth, Donald E. 1997. *The Art of Computer Programming, Volume 3: Sorting and Searching*, 3rd edition. Addison-Wesley. (Русский перевод: Кнут Д. Искусство программирования. Т. 3. Сортировка и поиск. 2-е изд. — М.: “Вильямс”, 2007. — 824 с.)

Ko, Youngjoong, Jinwoo Park, and Jungyun Seo. 2004. Improving text categorization using the importance of sentences. *IP&M* 40(1):65–79.

Koenemann, Jürgen, and Nicholas J. Belkin. 1996. A case for interaction: A study of interactive information retrieval behavior and effectiveness. In *Proc. SIGCHI Conference on Human Factors in Computing Systems*, pp. 205–212. ACM Press. doi: http://doi.acm.org/10.1145/238386.238487.

Kołcz, Aleksander, Vidya Prabakarmurthi, and Jugal Kalita. 2000. Summarization as feature selection for text categorization. In *Proc. CIKM*, pp. 365–370. ACM Press.

Kołcz, Aleksander, and Wen-Tau Yih. 2007. Raising the baseline for high-precision text classifiers. In *Proc. KDD*.

Koller, Daphne, and Mehran Sahami. 1997. Hierarchically classifying documents using very few words. In *Proc. ICML*, pp. 170–178.

Konheim, Alan G. 1981. *Cryptography: A Primer*. JohnWiley & Sons.

Korfhage, Robert R. 1997. *Information Storage and Retrieval*.Wiley.

Kozlov, M. K., S. P. Tarasov, and L. G. Khachiyan. 1979. Polynomial solvability of convex quadratic programming. *Soviet Mathematics Doklady* 20:1108–1111. Translated from original in *Doklady Akademiia Nauk SSR*, 228 (1979). (Оригинал: Козлов М.А., Тарасов С.П., Хачиян Л.Г. Полиномиальная разрешимость выпуклого квадратичного программирования // Журнал вычислительной математики и математической физики. — 1980. — Т. 20, № 5. — С. 1319–1323.).

Kraaij, Wessel, and Martijn Spitters. 2003. Language models for topic tracking. In W. B. Croft and J. Lafferty (eds.), *Language Modeling for Information Retrieval*, pp. 95–124. Kluwer.

Kraaij, Wessel, Thijs Westerveld, and Djoerd Hiemstra. 2002. The importance of prior probabilities for entry page search. In *Proc. SIGIR*, pp. 27–34. ACM Press.

Krippendorff, Klaus. 2003. *Content Analysis: An Introduction to its Methodology*. Sage.

Krovetz, Bob. 1995. *Word sense disambiguation for large text databases*. PhD thesis, University of Massachusetts Amherst.

Kukich, Karen. 1992. Techniques for automatically correcting words in text. *ACM Computing Surveys* 24(4):377–439. doi: `http://doi.acm.org/10.1145/146370.146380`.

Kumar, Ravi, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. 1999. Trawling the Web for emerging cyber-communities. *Computer Networks* 31(11–16): 1481–1493. URL: `citeseer.ist.psu.edu/kumar99trawling.html`.

Kumar, S. Ravi, Prabhakar Raghavan, Sridhar Rajagopalan, Dandapani Sivakumar, Andrew Tomkins, and Eli Upfal. 2000. The Web as a graph. In *Proc. PODS*, pp. 1–10. ACM Press. URL: `citeseer.ist.psu.edu/article/kumar00web.html`.

Kupiec, Julian, Jan Pedersen, and Francine Chen. 1995. A trainable document summarizer. In *Proc. SIGIR*, pp. 68–73. ACM Press.

KURLand, Oren, and Lillian Lee. 2004. Corpus structure, language models, and ad hoc information retrieval. In *Proc. SIGIR*, pp. 194–201. ACM Press. doi: `http://doi.acm.org/10.1145/1008992.1009027`.

Lafferty, John, and Chengxiang Zhai. 2001. Document language models, query models, and risk minimization for information retrieval. In *Proc. SIGIR*, pp. 111–119. ACM Press.

Lafferty, John, and Chengxiang Zhai. 2003. Probabilistic relevance models based on document and query generation. In W. Bruce Croft and John Lafferty (eds.), *Language Modeling and Information Retrieval*. Kluwer.

Lalmas, Mounia, Gabriella Kazai, Jaap Kamps, Jovan Pehcevski, Benjamin Piwowarski, and Stephen E. Robertson. 2007. INEX 2006 evaluation measures. In Fuhr et al. (2007), pp. 20–34.

Lalmas, Mounia, and Anastasios Tombros. 2007. Evaluating XML retrieval effectiveness at INEX. *SIGIR Forum* 41(1):40–57. doi: `http://doi.acm.org/10.1145/1273221.1273225`.

Lance, G. N., and W. T. Williams. 1967. A general theory of classificatory sorting strategies 1. Hierarchical systems. *Computer Journal* 9(4):373–380.

Langville, Amy, and Carl Meyer. 2006. *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton University Press.

Larsen, Bjornar, and Chinatsu Aone. 1999. Fast and effective text mining using linear-time document clustering. In *Proc. KDD*, pp. 16–22. ACM Press. doi: http://doi.acm.org/10.1145/312129.312186.

Larson, Ray R. 2005. A fusion approach to XML structured document retrieval. *IR* 8 (4):601–629. doi: http://dx.doi.org/10.1007/s10791-005-0749-0.

Lavrenko, Victor, and W. Bruce Croft. 2001. Relevance-based language models. In *Proc. SIGIR*, pp. 120–127. ACM Press.

Lawrence, Steve, and C. Lee Giles. 1998. Searching the World Wide Web. *Science* 280 (5360):98–100. URL: citeseer.ist.psu.edu/lawrence98searching.html.

Lawrence, Steve, and C. Lee Giles. 1999. Accessibility of information on the web. *Nature* 500:107–109.

Lee, Whay C., and Edward A. Fox. 1988. *Experimental comparison of schemes for interpreting Boolean queries*. Technical Report TR-88-27, Computer Science, Virginia Polytechnic Institute and State University.

Lempel, Ronny, and Shlomo Moran. 2000. The stochastic approach for link-structure analysis (SALSA) and the TKC effect. *Computer Networks* 33(1–6):387–401. URL: citeseer.ist.psu.edu/lempel00stochastic.html.

Lesk, Michael. 1988. Grab — Inverted indexes with low storage overhead. *Computing Systems* 1:207–220.

Lesk, Michael. 2004. *Understanding Digital Libraries*, 2nd edition. Morgan Kaufmann.

Lester, Nicholas, Alistair Moffat, and Justin Zobel. 2005. Fast on-line index construction by geometric partitioning. In *Proc. CIKM*, pp. 776–783. ACM Press. doi: http://doi.acm.org/10.1145/1099554.1099739.

Lester, Nicholas, Justin Zobel, and Hugh E. Williams. 2006. Efficient online index maintenance for contiguous inverted lists. *IP&M* 42(4):916–933. doi: http://dx.doi.org/10.1016/j.ipm.2005.09.005.

Levenshtein, Vladimir I. 1965. Binary codes capable of correcting spurious insertions and deletions of ones. *Problems of Information Transmission* 1:8–17. (Оригинал: Левенштейн В. И. Двоичные коды с исправлением выпадений, вставок и замещений символов // Доклады Академии Íŕóę ŃŃŃĐ. — 1965. — 163, ą4. — Ń. 845-848.)

Lew, Michael S. 2001. *Principles of Visual Information Retrieval*. Springer.

Lewis, David D. 1995. Evaluating and optimizing autonomous text classification systems. In *Proc. SIGIR*. ACM Press.

Lewis, David D. 1998. Naive (Bayes) at forty: The independence assumption in information retrieval. In *ECML*, pp. 4–15. Springer.

Lewis, David D., and Karen Spärck Jones. 1996. Natural language processing for information retrieval. *CACM* 39(1):92–101. doi: http://doi.acm.org/10.1145/234173.234210.

Lewis, David D., and Marc Ringuette. 1994. A comparison of two learning algorithms for text categorization. In *SDAIR*, pp. 81–93.

Lewis, David D., Robert E. Schapire, James P. Callan, and Ron Papka. 1996. Training algorithms for linear text classifiers. In *Proc. SIGIR*, pp. 298–306. ACM Press. doi: http://doi.acm.org/10.1145/243199.243277.

Lewis, David D., Yiming Yang, Tony G. Rose, and Fan Li. 2004. RCV1: A new benchmark collection for text categorization research. *JMLR* 5:361–397.

Li, Fan, and Yiming Yang. 2003. A loss function analysis for classification methods in text categorization. In *Proc. ICML*, pp. 472–479.

Liddy, Elizabeth D. 2005. Automatic document retrieval. In *Encyclopedia of Language and Linguistics*, 2nd edition. Elsevier.

List, Johan, Vojkan Mihajlovic, Georgina Ramírez, Arjen P. Vries, Djoerd Hiemstra, and Henk Ernst Blok. 2005. TIJAH: Embracing IR methods in XML databases. *IR* 8 (4):547–570. doi: `http://dx.doi.org/10.1007/s10791-005-0747-2`.

Lita, Lucian Vlad, Abe Ittycheriah, Salim Roukos, and Nanda Kambhatla. 2003. tRuEcasIng. In *Proc. ACL*, pp. 152–159.

Littman, Michael L., Susan T. Dumais, and Thomas K. Landauer. 1998. Automatic cross-language information retrieval using latent semantic indexing. In Gregory Grefenstette (ed.), *Cross Language Information Retrieval*. Kluwer. URL: `citeseer.ist.psu.edu/littman98automatic.html`.

Liu, Tie-Yan, Yiming Yang, Hao Wan, Hua-Jun Zeng, Zheng Chen, and Wei-Ying Ma. 2005. Support vector machines classification with very large scale taxonomy. *ACM SIGKDD Explorations* 7(1):36–43. [319]

Liu, Xiaoyong, and W. Bruce Croft. 2004. Cluster-based retrieval using language models. In *Proc. SIGIR*, pp. 186–193. ACM Press. doi: `http://doi.acm.org/10.1145/1008992.1009026`.

Lloyd, Stuart P. 1982. Least squares quantization in PCM. *IEEE Transactions on Information Theory* 28(2):129–136.

Lodhi, Huma, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. 2002. Text classification using string kernels. *JMLR* 2:419–444.

Lombard, Matthew, Cheryl C. Bracken, and Jennifer Snyder-Duch. 2002. Content analysis in mass communication: Assessment and reporting of intercoder reliability. *Human Communication Research* 28:587–604.

Long, Xiaohui, and Torsten Suel. 2003. Optimized query execution in large search engines with global page ordering. In *Proc. VLDB*. URL: `citeseer.ist.psu.edu/long03optimized.html`.

Lovins, Julie Beth. 1968. Development of a stemming algorithm. *Translation and Computational Linguistics* 11(1):22–31.

Lu, Wei, Stephen E. Robertson, and Andrew MacFarlane. 2007. CISR at INEX 2006. In Fuhr et al. (2007), pp. 57–63.

Luhn, Hans Peter. 1957. A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of Research and Development* 1(4):309–317.

Luhn, Hans Peter. 1958. The automatic creation of literature abstracts. *IBM Journal of Research and Development* 2(2):159–165, 317.

Luk, Robert W. P., and Kui-Lam Kwok. 2002. A comparison of Chinese document indexing strategies and retrieval models. *ACM Transactions on Asian Language Information Processing* 1(3):225–268.

Lunde, Ken. 1998. *CJKV Information Processing*. O'Reilly.

MacFarlane, A., J.A. McCann, and S.E. Robertson. 2000. Parallel search using partitioned inverted files. In *Proc. SPIRE*, pp. 209–220.

MacQueen, James B. 1967. Some methods for classification and analysis of multivariate observations. In *Proc. Berkeley Symposium on Mathematics, Statistics and Probability*, pp. 281–297. University of California Press.

Manning, Christopher D., and Hinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press.

Maron, M. E., and J. L. Kuhns. 1960. On relevance, probabilistic indexing, and information retrieval. *JACM* 7(3):216–244.

Mass, Yosi, Matan Mandelbrod, Einat Amitay, David Carmel, Yoëlle S. Maarek, and Aya Soffer. 2003. JuruXML — An XML retrieval system at INEX'02. In Fuhr et al. (2003b), pp. 73–80. URL: `http://inex.is.informatik.uni-duisburg.de:2003/proceedings.pdf`.

McBryan, Oliver A. 1994. GENVL and WWWW: Tools for Taming the Web. In *Proc. WWW*. URL: `citeseer.ist.psu.edu/mcbryan94genvl.html`.

McCallum, Andrew, and Kamal Nigam. 1998. A comparison of event models for Naive Bayes text classification. In *Working Notes of the 1998 AAAI/ICML Workshop on Learning for Text Categorization*, pp. 41–48.

McCallum, Andrew, Ronald Rosenfeld, Tom M. Mitchell, and Andrew Y. Ng. 1998. Improving text classification by shrinkage in a hierarchy of classes. In *Proc. ICML*, pp. 359–367. Morgan Kaufmann.

McCallum, Andrew Kachites. 1996. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. URL: `www.cs.cmu.edu/.mccallum/bow`.

McKeown, Kathleen, and Dragomir R. Radev. 1995. Generating summaries of multiple news articles. In *Proc. SIGIR*, pp. 74–82. ACM Press. doi: `http://doi.acm.org/10.1145/215206.215334`.

McKeown, Kathleen R., Regina Barzilay, David Evans, Vasileios Hatzivassiloglou, Judith L. Klavans, Ani Nenkova, Carl Sable, Barry Schiffman, and Sergey Sigelman. 2002. Tracking and summarizing news on a daily basis with Columbia's Newsblaster. In *Proc. Human Language Technology Conference*.

McLachlan, Geoffrey J., and Thiriyambakam Krishnan. 1996. *The EM Algorithm and Extensions*. John Wiley & Sons.

Meadow, Charles T., Donald H. Kraft, and Bert R. Boyce. 1999. *Text Information Retrieval Systems*. Academic Press.

Meilă, Marina. 2005. Comparing clusterings — An axiomatic view. In *Proc. ICML*.

Melnik, Sergey, Sriram Raghavan, Beverly Yang, and Hector Garcia-Molina. 2001. Building a distributed full-text index for the web. In *Proc.WWW*, pp. 396–406. ACM Press. doi: `http://doi.acm.org/10.1145/371920.372095`.

Mihajlović, Vojkan, Henk Ernst Blok, Djoerd Hiemstra, and Peter M. G. Apers. 2005. Score region algebra: Building a transparent XML-R database. In *Proc. CIKM*, pp. 12–19. ACM Press. doi: `http://doi.acm.org/10.1145/1099554.1099560`.

Miller, David R. H., Tim Leek, and Richard M. Schwartz. 1999. A hidden Markov model information retrieval system. In *Proc. SIGIR*, pp. 214–221. ACM Press.

Minsky, Marvin Lee, and Seymour Papert (eds.). 1988. *Perceptrons: An Introduction to Computational Geometry*. MIT Press. Expanded edition.

Mitchell, Tom M. 1997. *Machine Learning*. McGraw-Hill.

Moffat, Alistair, and Timothy A. H. Bell. 1995. In situ generation of compressed inverted files. *JASIS* 46(7):537–550.

Moffat, Alistair, and Lang Stuiver. 1996. Exploiting clustering in inverted file compression. In *Proc. Conference on Data Compression*, pp. 82–91. IEEE Computer Society.

Moffat, Alistair, and Justin Zobel. 1992. Parameterised compression for sparse bitmaps. In *Proc. SIGIR*, pp. 274–285. ACM Press. doi: http://doi.acm.org/10.1145/133160.133210.

Moffat, Alistair, and Justin Zobel. 1996. Self-indexing inverted files for fast text retrieval. *TOIS* 14(4):349–379. [44]

Moffat, Alistair, and Justin Zobel. 1998. Exploring the similarity space. *SIGIR Forum* 32(1).

Mooers, Calvin. 1961. From a point of view of mathematical etc. techniques. In R.A. Fairthorne (ed.), *Towards Information Retrieval*, pp. xvii–xxiii. Butterworths.

Mooers, Calvin E. 1950. Coding, information retrieval, and the rapid selector. *American Documentation* 1(4):225–229.

Moschitti, Alessandro. 2003. A study on optimal parameter tuning for Rocchio text classifier. In *Proc. ECIR*, pp. 420–435.

Moschitti, Alessandro, and Roberto Basili. 2004. Complex linguistic features for text classification: A comprehensive study. In *Proc. ECIR*, pp. 181–196.

Murata, Masaki, Qing Ma, Kiyotaka Uchimoto, Hiromi Ozaku, Masao Utiyama, and Hitoshi Isahara. 2000. Japanese probabilistic information retrieval using location and category information. In *Proc. International Workshop on Information Retrieval With Asian Languages*, pp. 81–88. URL: http://portal.acm.org/citation.cfm?doid=355214.355226.

Muresan, Gheorghe, and David J. Harper. 2004. Topic modeling for mediated access to very large document collections. *JASIST* 55(10):892–910. doi: http://dx.doi.org/10.1002/asi.20034.

Murtagh, Fionn. 1983. A survey of recent advances in hierarchical clustering algorithms. *Computer Journal* 26(4):354–359.

Najork, Marc, and Allan Heydon. 2001. *High-performance web crawling*. Technical Report 173, Compaq Systems Research Center.

Najork, Marc, and Allan Heydon. 2002. High-performance web crawling. In Panos Pardalos, James Abello and Mauricio Resende (eds.), *Handbook of Massive Data Sets*, chapter 2. Kluwer.

Navarro, Gonzalo, and Ricardo Baeza-Yates. 1997. Proximal nodes: A model to query document databases by content and structure. *TOIS* 15(4):400–435. doi: http://doi.acm.org/10.1145/263479.263482.

Newsam, Shawn, Sitaram Bhagavathy, and B. S. Manjunath. 2001. Category-based image retrieval. In *IEEE International Conference on Image Processing, Special Session on Multimedia Indexing, Browsing and Retrieval*, pp. 596–599.

Ng, Andrew Y., and Michael I. Jordan. 2001. On discriminative vs. generative classifiers: A comparison of logistic regression and Naive Bayes. In *NIPS*, pp. 841–848. URL: www-2.cs.cmu.edu/Groups/NIPS/NIPS2001/papers/psgz/AA28.ps.gz.

Ng, Andrew Y., Michael I. Jordan, and YairWeiss. 2001a. On spectral clustering: Analysis and an algorithm. In *Proc. NIPS*, pp. 849–856.

Ng, Andrew Y., Alice X. Zheng, and Michael I. Jordan. 2001b. Link analysis, eigenvectors and stability. In *Proc. IJCAI*, pp. 903–910. URL: citeseer.ist.psu.edu/ng01link.html.

Nigam, Kamal, Andrew McCallum, and Tom Mitchell. 2006. Semi-supervised text classification using EM. In Chapelle et al. (2006), pp. 33–56.

Ntoulas, Alexandros, and Junghoo Cho. 2007. Pruning policies for two-tiered inverted index with correctness guarantee. In *Proc. SIGIR*, pp. 191–198. ACM Press.

Oard, DouglasW., and Bonnie J. Dorr. 1996. *A survey of multilingual text retrieval*. Technical Report UMIACS-TR-96-19, Institute for Advanced Computer Studies, University of Maryland, College Park, MD, USA.

Ogilvie, Paul, and Jamie Callan. 2005. Parameter estimation for a simple hierarchical generative model for XML retrieval. In *Proc. INEX*, pp. 211–224. doi: `http://dx.doi.org/10.1007/11766278_16`.

O'Keefe, Richard A., and Andrew Trotman. 2004. The simplest query language that could possibly work. In Fuhr et al. (2005), pp. 167–174.

Osiński, Stanisław, and Dawid Weiss. 2005. A concept-driven algorithm for clustering search results. *IEEE Intelligent Systems* 20(3):48–54.

Page, Lawrence, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1998. The Page-Rank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project. URL: `citeseer.ist.psu.edu/page98pagerank.html`.

Paice, Chris D. 1990. Another stemmer. *SIGIR Forum* 24(3):56–61.

Papineni, Kishore. 2001. Why inverse document frequency? In *North American Chapter of the Association for Computational Linguistics*, pp. 1–8.

Pavlov, Dmitry, Ramnath Balasubramanyan, Byron Dom, Shyam Kapur, and Jignashu Parikh. 2004. Document preprocessing for naive Bayes classification and clustering with mixture of multinomials. In *Proc. KDD*, pp. 829–834.

Pelleg, Dan, and Andrew Moore. 1999. Accelerating exact k-means algorithms with geometric reasoning. In *Proc. KDD*, pp. 277–281. ACM Press. doi: `http://doi.acm.org/10.1145/312129.312248`.

Pelleg, Dan, and Andrew Moore. 2000. X-means: Extending k-means with efficient estimation of the number of clusters. In *Proc. ICML*, pp. 727–734. Morgan Kaufmann.

Perkins, Simon, Kevin Lacker, and James Theiler. 2003. Grafting: Fast, incremental feature selection by gradient descent in function space. *JMLR* 3:1333–1356.

Persin, Michael. 1994. Document filtering for fast ranking. In *Proc. SIGIR*, pp. 339-348. ACM Press.

Persin, Michael, Justin Zobel, and Ron Sacks-Davis. 1996. Filtered document retrieval with frequency-sorted indexes. *JASIS* 47(10):749–764.

Peterson, James L. 1980. Computer programs for detecting and correcting spelling errors. *CACM* 23(12):676–687. doi: `http://doi.acm.org/10.1145/359038.359041`.

Picca, Davide, Benoît Curdy, and Fran**C**ois Bavaud. 2006. Non-linear correspondence analysis in text retrieval: A kernel view. In *Proc. JADT*.

Pinski, Gabriel, and Francis Narin. 1976. Citation influence for journal aggregates of scientific publications: Theory, with application to the literature of Physics. *IP&M* 12:297–326.

Pirolli, Peter L. T. 2007. *Information Foraging Theory: Adaptive Interaction With Information*. Oxford University Press.

Platt, John. 2000. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In A.J. Smola, P.L. Bartlett, B. Sch**O**lkopf, and D. Schuurmans (eds.), *Advances in Large Margin Classifiers*, pp. 61–74. MIT Press.

Ponte, Jay M., and W. Bruce Croft. 1998. A language modeling approach to information retrieval. In *Proc. SIGIR*, pp. 275–281. ACM Press.

Popescul, Alexandrin, and Lyle H. Ungar. 2000. Automatic labeling of document clusters. Unpublished.

Porter, Martin F. 1980. An algorithm for suffix stripping. *Program* 14(3):130–137.

Pugh, William. 1990. Skip lists: A probabilistic alternative to balanced trees. *CACM* 33(6):668–676.

Qin, Tao, Tie-Yan Liu, Wei Lai, Xu-Dong Zhang, De-Sheng Wang, and Hang Li. 2007. Ranking with multiple hyperplanes. In *Proc. SIGIR*. ACM Press.

Qiu, Yonggang, and H.P. Frei. 1993. Concept based query expansion. In *Proc. SIGIR*, pp. 160–169. ACM Press.

R Development Core Team. 2005. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria. URL: `www.Rproject.org`. ISBN 3-900051-07-0.

Radev, Dragomir R., Sasha Blair-Goldensohn, Zhu Zhang, and Revathi Sundara Raghavan. 2001. Interactive, domain-independent identification and summarization of topically related news articles. In *Proc. European Conference on Research and Advanced Technology for Digital Libraries*, pp. 225–238.

Rahm, Erhard, and Philip A. Bernstein. 2001. A survey of approaches to automatic schema matching. *VLDB Journal* 10(4):334–350. URL: `citeseer.ist.psu.edu/rahm01survey.html`.

Rand, William M. 1971. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association* 66(336):846–850.

Rasmussen, Edie. 1992. Clustering algorithms. In Frakes and Baeza-Yates (1992), pp. 419–442.

Rennie, Jason D., Lawrence Shih, Jaime Teevan, and David R. Karger. 2003. Tackling the poor assumptions of naive Bayes text classifiers. In *Proc. ICML*, pp. 616–623.

Ribeiro-Neto, Berthier, Edleno S. Moura, Marden S. Neubert, and Nivio Ziviani. 1999. Efficient distributed algorithms to build inverted files. In *Proc. SIGIR*, pp. 105–112. ACM Press. doi: `http://doi.acm.org/10.1145/312624.312663`.

Ribeiro-Neto, Berthier A., and Ramurti A. Barbosa. 1998. Query performance for tightly coupled distributed digital libraries. In *ACM Conference on Digital Libraries*, pp. 182–190.

Rice, John A. 2006. *Mathematical Statistics and Data Analysis*. Duxbury Press.

Richardson, M., A. Prakash, and E. Brill. 2006. Beyond PageRank: machine learning for static ranking. In *Proc. WWW*, pp. 707–715.

Riezler, Stefan, Alexander Vasserman, Ioannis Tsochantaridis, Vibhu Mittal, and Yi Liu. 2007. Statistical machine translation for query expansion in answer retrieval. In *Proc. ACL*, pp. 464–471. Association for Computational Linguistics. URL: `www.aclweb.org/anthology/P/P07/P07-1059`.

Ripley, B. D. 1996. *Pattern Recognition and Neural Networks*. Cambridge University Press. Robertson, Stephen. 2005. How Okapi came to TREC. In Voorhees and Harman (2005), pp. 287–299.

Robertson, Stephen, Hugo Zaragoza, and Michael Taylor. 2004. Simple BM25 extension to multiple weighted fields. In *Proc. CIKM*, pp. 42–49. ACM Press. doi: `http://doi.acm.org/10.1145/1031171.1031181`.

Robertson, Stephen E., and Karen Spärck Jones. 1976. Relevance weighting of search terms. *JASIS* 27:129–146.

Rocchio, J. J. 1971. Relevance feedback in information retrieval. In Salton (1971b), pp. 313–323.

Roget, P. M. 1946. *Roget's International Thesaurus*. Thomas Y. Crowell.

Rosen-Zvi, Michal, Thomas Griffiths, Mark Steyvers, and Padhraic Smyth. 2004. The author-topic model for authors and documents. In *Proc. UAI*, pp. 487–494. AUAI Press.

Ross, Sheldon. 2006. *A First Course in Probability*. Pearson Prentice-Hall.

Rusmevichientong, Paat, David M. Pennock, Steve Lawrence, and C. Lee Giles. 2001. Methods for sampling pages uniformly from the world wide web. In *Proc. AAAI Fall Symposium on Using Uncertainty Within Computation*, pp. 121–128. URL: `citeseer.ist.psu.edu/rusmevichientong01methods.html`.

Ruthven, Ian, and Mounia Lalmas. 2003. A survey on the use of relevance feedback for information access systems. *Knowledge Engineering Review* 18(1).

Sahoo, Nachiketa, Jamie Callan, Ramayya Krishnan, George Duncan, and Rema Padman. 2006. Incremental hierarchical clustering of text documents. In *Proc. CIKM*, pp. 357–366. ACM Press. doi: `http://doi.acm.org/10.1145/1183614.1183667`.

Sakai, Tetsuya. 2007. On the reliability of information retrieval metrics based on graded relevance. *IP&M* 43(2):531–548.

Salton, Gerard. 1971a. Cluster search strategies and the optimization of retrieval effectiveness. In *The SMART Retrieval System — Experiments in Automatic Document Processing* Salton (1971b), pp. 223–242.

Salton, Gerard (ed.). 1971b. *The SMART Retrieval System — Experiments in Automatic Document Processing*. Prentice-Hall.

Salton, Gerard. 1975. *Dynamic information and library processing*. Prentice-Hall. (Русский перевод: Сэлтон Дж. Динамические библиотечно-поисковые системы. М.: — Мир, 1979).

Salton, Gerard. 1989. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. AddisonWesley. (*Русский перевод*: Сэлтон Г. Автоматическая обработка, хранение и поиск информации. — М.: Советское радио, 1973.)

Salton, Gerard. 1991. The Smart project in automatic document retrieval. In *Proc. SIGIR*, pp. 356–358. ACM Press.

Salton, Gerard, James Allan, and Chris Buckley. 1993. Approaches to passage retrieval in full text information systems. In *Proc. SIGIR*, pp. 49–58. ACM Press. doi: `http://doi.acm.org/10.1145/160688.160693`.

Salton, Gerard, and Chris Buckley. 1987. *Term weighting approaches in automatic text retrieval*. Technical report, Cornell University, Ithaca, NY.

Salton, Gerard, and Christopher Buckley. 1988. Term-weighting approaches in automatic text retrieval. *IP&M* 24(5):513–523.

Salton, Gerard, and Chris Buckley. 1990. Improving retrieval performance by relevance feedback. *JASIS* 41(4):288–297.

Saracevic, Tefko, and Paul Kantor. 1988. A study of information seeking and retrieving. II: Users, questions and effectiveness. *JASIS* 39:177–196.

Saracevic, Tefko, and Paul Kantor. 1996. A study of information seeking and retrieving. III: Searchers, searches, overlap. *JASIS* 39(3):197–216.

Savaresi, Sergio M., and Daniel Boley. 2004. A comparative analysis on the bisecting K-means and the PDDP clustering algorithms. *Intelligent Data Analysis* 8(4):345–362.

Schamber, Linda, Michael Eisenberg, and Michael S. Nilan. 1990. A re-examination of relevance: toward a dynamic, situational definition. *IP&M* 26(6):755–776.

Schapire, Robert E. 2003. The boosting approach to machine learning: An overview. In D. D. Denison, M. H. Hansen, C. Holmes, B. Mallick, and B. Yu (eds.), *Nonlinear Estimation and Classification*. Springer.

Schapire, Robert E., and Yoram Singer. 2000. Boostexter: A boosting-based system for text categorization. *Machine Learning* 39(2/3):135–168.

Schapire, Robert E., Yoram Singer, and Amit Singhal. 1998. Boosting and Rocchio applied to text filtering. In *Proc. SIGIR*, pp. 215–223. ACM Press.

Schlieder, Torsten, and Holger Meuss. 2002. Querying and ranking XML documents. *JASIST* 53(6):489–503. doi: `http://dx.doi.org/10.1002/asi.10060`.

Scholer, Falk, Hugh E.Williams, John Yiannis, and Justin Zobel. 2002. Compression of inverted indexes for fast query evaluation. In *Proc. SIGIR*, pp. 222–229. ACM Press. doi: `http://doi.acm.org/10.1145/564376.564416`.

Schölkopf, Bernhard, and Alexander J. Smola. 2001. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press.

Schütze, Hinrich. 1998. Automatic word sense discrimination. *Computational Linguistics* 24(1):97–124.

Schütze, Hinrich, David A. Hull, and Jan O. Pedersen. 1995. A comparison of classifiers and document representations for the routing problem. In *Proc. SIGIR*, pp. 229–237. ACM Press.

Schütze, Hinrich, and Jan O. Pedersen. 1995. Information retrieval based on word senses. In *Proc. SDAIR*, pp. 161–175.

Schütze, Hinrich, and Craig Silverstein. 1997. Projections for efficient document clustering. In *Proc. SIGIR*, pp. 74–81. ACM Press.

Schwarz, Gideon. 1978. Estimating the dimension of a model. *Annals of Statistics* 6(2):461–464.

Sebastiani, Fabrizio. 2002. Machine learning in automated text categorization. *ACM Computing Surveys* 34(1):1–47.

Shawe-Taylor, John, and Nello Cristianini. 2004. *Kernel Methods for Pattern Analysis*. Cambridge University Press.

Shkapenyuk, Vladislav, and Torsten Suel. 2002. Design and implementation of a high performance distributed web crawler. In *Proc. International Conference on Data Engineering*. URL: `citeseer.ist.psu.edu/shkapenyuk02design.html`.

Siegel, Sidney, and N. John Castellan, Jr. 1988. *Nonparametric Statistics for the Behavioral Sciences*, 2nd edition. McGraw-Hill.

Sifry, Dave, 2007. The state of the Live Web, April 2007. URL: `http://technorati.com/weblog/2007/04/328.html`.

Sigurbjörnsson, Börkur, Jaap Kamps, and Maarten de Rijke. 2004. Mixture models, overlap, and structural hints in XML element retrieval. In *Proc. INEX*, pp. 196–210.

Silverstein, Craig, Monika Rauch Henzinger, Hannes Marais, and Michael Moricz. 1999. Analysis of a very large web search engine query log. *SIGIR Forum* 33(1):6–12.

Silvestri, Fabrizio. 2007. Sorting out the document identifier assignment problem. In *Proc. ECIR*, pp. 101–112.

Silvestri, Fabrizio, Raffaele Perego, and Salvatore Orlando. 2004. Assigning document identifiers to enhance compressibility of web search engines indexes. In *Proc. ACM Symposium on Applied Computing*, pp. 600–605.

Sindhwani, V., and S. S. Keerthi. 2006. Large scale semi-supervised linear SVMs. In *Proc. SIGIR*, pp. 477–484.

Singhal, Amit, Chris Buckley, and Mandar Mitra. 1996a. Pivoted document length normalization. In *Proc. SIGIR*, pp. 21–29. ACM Press. URL: `citeseer.ist.psu.edu/singhal96pivoted.html`.

Singhal, Amit, Mandar Mitra, and Chris Buckley. 1997. Learning routing queries in a query zone. In *Proc. SIGIR*, pp. 25–32. ACM Press.

Singhal, Amit, Gerard Salton, and Chris Buckley. 1995. *Length normalization in degraded text collections*. Technical report, Cornell University, Ithaca, NY.

Singhal, Amit, Gerard Salton, and Chris Buckley. 1996*b*. Length normalization in degraded text collections. In *Proc. SDAIR*, pp. 149–162.

Singitham, Pavan Kumar C., Mahathi S. Mahabhashyam, and Prabhakar Raghavan. 2004. Efficiency-quality tradeoffs for vector score aggregation. In *Proc. VLDB*, pp. 624–635. URL: `http://www.vldb.org/conf/2004/RS17P1.PDF`.

Smeulders, Arnold W. M., Marcel Worring, Simone Santini, Amarnath Gupta, and Ramesh Jain. 2000. Content-based image retrieval at the end of the early years. *IEEE Trans. Pattern Anal. Mach. Intell.* 22(12):1349–1380. doi: `http://dx.doi.org/10.1109/34.895972`

Sneath, Peter H.A., and Robert R. Sokal. 1973. *Numerical Taxonomy: The Principles and Practice of Numerical Classification*.W.H. Freeman.

Snedecor, George Waddel, and William G. Cochran. 1989. *Statistical Methods*. Iowa State University Press.

Somogyi, Zoltan. 1990. *The Melbourne University bibliography system*. Technical Report 90/3, Melbourne University, Parkville, Victoria, Australia.

Song, Ruihua, Ji-Rong Wen, and Wei-Ying Ma. 2005. *Viewing term proximity from a different perspective*. Technical Report MSR-TR-2005-69, Microsoft Research.

Sornil, Ohm. 2001. *Parallel Inverted Index for Large-Scale, Dynamic Digital Libraries*. PhD thesis, Virginia Tech. URL: `http://scholar.lib.vt.edu/theses/available/etd-02062001-114915/`.

Spärck Jones, Karen. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation* 28(1):11–21.

Spärck Jones, Karen. 2004. Language modelling's generative model: Is it rational? MS, Computer Laboratory, University of Cambridge. URL: `http://www.cl.cam.ac.uk/.ksj21/langmodnote4.pdf`.

Spärck Jones, Karen, S.Walker, and Stephen E. Robertson. 2000. A probabilistic model of information retrieval: Development and comparative experiments. *IP&M* 36(6):779–808, 809–840.

Spink, Amanda, and Charles Cole (eds.). 2005. *New Directions in Cognitive Information Retrieval*. Springer.

Spink, Amanda, Bernard J. Jansen, and H. Cenk Ozmultu. 2000. Use of query reformulation and relevance feedback by Excite users. *Internet Research: Electronic Networking Applications and Policy* 10(4):317–328. URL: `http://ist.psu.edu/facultypages/jjansen/academic/pubs/internetresearch2000.pdf`.

Sproat, Richard, and Thomas Emerson. 2003. The first international Chinese word segmentation bakeoff. In *SIGHAN Workshop on Chinese Language Processing*.

Sproat, Richard, William Gale, Chilin Shih, and Nancy Chang. 1996. A stochastic finite-state word-segmentation algorithm for Chinese. *Computational Linguistics* 22 (3):377–404.

Sproat, Richard William. 1992. *Morphology and Computation*. MIT Press.

Stein, Benno, and Sven Meyer zu Eissen. 2004. Topic identification: Framework and application. In *Proc. International Conference on Knowledge Management*.

Stein, Benno, Sven Meyer zu Eissen, and Frank Wißbrock. 2003. On cluster validity and the information need of users. In *Proc. Artificial Intelligence and Applications*.

Steinbach, Michael, George Karypis, and Vipin Kumar. 2000. A comparison of document clustering techniques. In *KDD Workshop on Text Mining*.

Strang, Gilbert (ed.). 1986. *Introduction to Applied Mathematics*. Wellesley-Cambridge Press.

Strehl, Alexander. 2002. *Relationship-based Clustering and Cluster Ensembles for Highdimensional Data Mining*. PhD thesis, The University of Texas at Austin.

Strohman, Trevor, and W. Bruce Croft. 2007. Efficient document retrieval in main memory. In *Proc. SIGIR*, pp. 175–182. ACM Press.

Swanson, Don R. 1988. Historical note: Information retrieval and the future of an illusion. *JASIS* 39(2):92–98.

Tague-Sutcliffe, Jean, and James Blustein. 1995. A statistical analysis of the TREC-3 data. In *Proc. TREC*, pp. 385–398.

Tan, Songbo, and Xueqi Cheng. 2007. Using hypothesis margin to boost centroid text classifier. In *Proc. ACM Symposium on Applied Computing*, pp. 398–403. ACM Press. doi: http://doi.acm.org/10.1145/1244002.1244096.

Tannier, Xavier, and Shlomo Geva. 2005. XML retrieval with a natural language interface. In *Proc. SPIRE*, pp. 29–40.

Tao, Tao, Xuanhui Wang, Qiaozhu Mei, and Cheng Xiang Zhai. 2006. Language model information retrieval with document expansion. In *Proc. Human Language Technology Conference/North American Chapter of the Association for Computational Linguistics*, pp. 407–414.

Taube, Mortimer, and Harold Wooster (eds.). 1958. *Information Storage and Retrieval: Theory, Systems, and Devices*. Columbia University Press.

Taylor, Michael, Hugo Zaragoza, Nick Craswell, Stephen Robertson, and Chris Burges. 2006. Optimisation methods for ranking functions with multiple parameters. In *Proc. CIKM*. ACM Press.

Teh, Yee Whye, Michael I. Jordan, Matthew J. Beal, and David M. Blei. 2006. Hierarchical Dirichlet processes. *Journal of the American Statistical Association* 101(476):1566–1581.

Theobald, Martin, Holger Bast, Debapriyo Majumdar, Ralf Schenkel, and Gerhard Weikum. 2008. TopX: Efficient and versatile top-*k* query processing for semistructured data. *VLDB Journal* 17(1):81–115.

Theobald, Martin, Ralf Schenkel, and Gerhard Weikum. 2005. An efficient and versatile query engine for TopX search. In *Proc. VLDB*, pp. 625–636. VLDB Endowment.

Tibshirani, Robert, Guenther Walther, and Trevor Hastie. 2001. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society Series B* 63:411–423.

Tishby, Naftali, and Noam Slonim. 2000. Data clustering by Markovian relaxation and the information bottleneck method. In *Proc. NIPS*, pp. 640–646.

Toda, Hiroyuki, and Ryoji Kataoka. 2005. A search result clustering method using informatively named entities. In *Proc. Annual ACM International Workshop on Web Information and Data Management*, pp. 81–86. ACM Press. doi: http://doi.acm.org/10.1145/1097047.1097063.

Tomasic, Anthony, and Hector Garcia-Molina. 1993. Query processing and inverted indices in shared-nothing document information retrieval systems. *VLDB Journal* 2(3):243–275. [419]

Tombros, Anastasios, and Mark Sanderson. 1998. Advantages of query biased summaries in information retrieval. In *Proc. SIGIR*, pp. 2–10. ACM Press. doi: http://doi.acm.org/10.1145/290941.290947.

Tombros, Anastasios, Robert Villa, and C. J. van Rijsbergen. 2002. The effectiveness of query-specific hierarchic clustering in information retrieval. *IP&M* 38(4):559-582. doi: http://dx.doi.org/10.1016/S0306-4573(01)00048-6.

Tomlinson, Stephen. 2003. Lexical and algorithmic stemming compared for 9 European languages with Hummingbird Searchserver at CLEF 2003. In *Proc. Cross-Language Evaluation Forum*, pp. 286–300.

Tong, Simon, and Daphne Koller. 2001. Support vector machine active learning with applications to text classification. *JMLR* 2:45–66.

Toutanova, Kristina, and Robert C. Moore. 2002. Pronunciation modeling for improved spelling correction. In *Proc. ACL*, pp. 144–151.

Treeratpituk, Pucktada, and Jamie Callan. 2006. An experimental study on automatically labeling hierarchical clusters using statistical features. In *Proc. SIGIR*, pp. 707–708. ACM Press. doi: `http://doi.acm.org/10.1145/1148170.1148328`.

Trotman, Andrew. 2003. Compressing inverted files. *IR* 6(1):5–19. doi: `http://dx.doi.org/10.1023/A:1022949613039`.

Trotman, Andrew, and Shlomo Geva. 2006. Passage retrieval and other XML-retrieval tasks. In *SIGIR 2006 Workshop on XML Element Retrieval Methodology*, pp. 43–50.

Trotman, Andrew, Shlomo Geva, and Jaap Kamps (eds.). 2007. *Proc. SIGIR 2007 Workshop on Focused Retrieval*. University of Otago, Dunedin, New Zealand.

Trotman, Andrew, Nils Pharo, and Miro Lehtonen. 2006. XML-IR users and use cases. In *Proc. INEX*, pp. 400–412.

Trotman, Andrew, and Börkur Sigurbjörnsson. 2004. Narrowed Extended XPath I (NEXI). In Fuhr et al. (2005), pp. 16–40. doi: `http://dx.doi.org/10.1007/114245502`.

Tseng, Huihsin, Pichuan Chang, Galen Andrew, Daniel Jurafsky, and Christopher Manning. 2005. A conditional random field word segmenter. In *SIGHAN Workshop on Chinese Language Processing*.

Tsochantaridis, Ioannis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. 2005. Large margin methods for structured and interdependent output variables. *JMLR* 6:1453–1484.

Turpin, Andrew, and William R. Hersh. 2001. Why batch and user evaluations do not give the same results. In *Proc. SIGIR*, pp. 225–231.

Turpin, Andrew, and William R. Hersh. 2002. User interface effects in past batch versus user experiments. In *Proc. SIGIR*, pp. 431–432.

Turpin, Andrew, Yohannes Tsegay, David Hawking, and Hugh E.Williams. 2007. Fast generation of result snippets in web search. In *Proc. SIGIR*, pp. 127–134. ACM Press.

Turtle, Howard. 1994. Natural language vs. Boolean query evaluation: A comparison of retrieval performance. In *Proc. SIGIR*, pp. 212–220. ACM Press.

Turtle, Howard, and W. Bruce Croft. 1989. Inference networks for document retrieval. In *Proc. SIGIR*, pp. 1–24. ACM Press.

Turtle, Howard, and W. Bruce Croft. 1991. Evaluation of an inference network-based retrieval model. *TOIS* 9(3):187–222.

Turtle, Howard, and James Flood. 1995. Query evaluation: strategies and optimizations. *IP&M* 31(6):831–850. doi: `http://dx.doi.org/10.1016/0306-4573(95)00020-H`.

Vaithyanathan, Shivakumar, and Byron Dom. 2000. Model-based hierarchical clustering. In *Proc. UAI*, pp. 599–608. Morgan Kaufmann.

van Rijsbergen, C. J. 1979. *Information Retrieval*, 2nd edition. Butterworths.

van Rijsbergen, C. J. 1989. Towards an information logic. In *SIGIR*, pp. 77–86. ACM Press. doi: `http://doi.acm.org/10.1145/75334.75344`.

van Zwol, Roelof, Jeroen Baas, Herre van Oostendorp, and Frans Wiering. 2006. Bricks: The building blocks to tackle query formulation in structured document retrieval. In *Proc. ECIR*, pp. 314–325.

Vapnik, Vladimir N. 1998. *Statistical Learning Theory*. Wiley-Interscience. (См. также Вапник В. Н., Червоненкис А. Я. Теория распознавания образов — М.: Наука, 1974 и Вапник В. Н. Восстановление зависимостей по эмпирическим данным. — М.: Наука, 1979.)

Vittaut, Jean-Noël, and Patrick Gallinari. 2006. Machine learning ranking for structured information retrieval. In *Proc. ECIR*, pp. 338–349.

Voorhees, Ellen M. 1985a. The cluster hypothesis revisited. In *Proc. SIGIR*, pp. 188–196. ACM Press.

Voorhees, Ellen M. 1985b. *The effectiveness and efficiency of agglomerative hierarchic clustering in document retrieval*. Technical Report TR 85–705, Cornell.

Voorhees, Ellen M. 2000. Variations in relevance judgments and the measurement of retrieval effectiveness. *IP&M* 36:697–716.

Voorhees, Ellen M., and Donna Harman (eds.). 2005. *TREC: Experiment and Evaluation in Information Retrieval*. MIT Press.

Wagner, Robert A., and Michael J. Fischer. 1974. The string-to-string correction problem. *JACM* 21(1):168–173. doi: `http://doi.acm.org/10.1145/321796.321811`.

Ward Jr., J. H. 1963. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association* 58:236–244.

Wei, Xing, and W. Bruce Croft. 2006. LDA-based document models for ad-hoc retrieval. In *Proc. SIGIR*, pp. 178–185. ACM Press. doi: `http://doi.acm.org/10.1145/1148170.1148204`.

Weigend, Andreas S., Erik D. Wiener, and Jan O. Pedersen. 1999. Exploiting hierarchy in text categorization. *IR* 1(3):193–216.

Weston, Jason, and Chris Watkins. 1999. Support vector machines for multi-class pattern recognition. In *Proc. European Symposium on Artificial Neural Networks*, pp. 219–224.

Williams, Hugh E., and Justin Zobel. 2005. Searchable words on the web. *International Journal on Digital Libraries* 5(2):99–105. doi: `http://dx.doi.org/10.1007/s00799-003-0050-z`.

Williams, Hugh E., Justin Zobel, and Dirk Bahle. 2004. Fast phrase querying with combined indexes. *TOIS* 22(4):573–594.

Witten, Ian H., and Timothy C. Bell. 1990. Source models for natural language text. *International Journal Man-Machine Studies* 32(5):545–579.

Witten, Ian H., and Eibe Frank. 2005. *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd edition. Morgan Kaufmann.

Witten, Ian H., Alistair Moffat, and Timothy C. Bell. 1999. *Managing Gigabytes: Compressing and Indexing Documents and Images*, 2nd edition. Morgan Kaufmann.

Wong, S. K. Michael, Yiyu Yao, and Peter Bollmann. 1988. Linear structure in information retrieval. In *Proc. SIGIR*, pp. 219–232. ACM Press.

Woodley, Alan, and Shlomo Geva. 2006. NLPX at INEX 2006. In *Proc. INEX*, pp. 302–311.

Xu, Jinxi, and W. Bruce Croft. 1996. Query expansion using local and global document analysis. In *Proc. SIGIR*, pp. 4–11. ACM Press.

Xu, Jinxi, and W. Bruce Croft. 1999. Cluster-based language models for distributed retrieval. In *Proc. SIGIR*, pp. 254–261. ACM Press. doi: `http://doi.acm.org/10.1145/312624.312687`.

Yang, Hui, and Jamie Callan. 2006. Near-duplicate detection by instance-level constrained clustering. In *Proc. SIGIR*, pp. 421–428. ACM Press. doi: http://doi.acm.org/10.1145/1148170.1148243.

Yang, Yiming. 1994. Expert network: Effective and efficient learning from human decisions in text categorization and retrieval. In *Proc. SIGIR*, pp. 13–22. ACM Press.

Yang, Yiming. 1999. An evaluation of statistical approaches to text categorization. *IR* 1:69–90.

Yang, Yiming. 2001. A study of thresholding strategies for text categorization. In *Proc. SIGIR*, pp. 137–145. ACM Press. doi: http://doi.acm.org/10.1145/383952.383975.

Yang, Yiming, and Bryan Kisiel. 2003. Margin-based local regression for adaptive filtering. In *Proc. CIKM*, pp. 191–198. ACM Press. doi: http://doi.acm.org/10.1145/956863.956902.

Yang, Yiming, and Xin Liu. 1999. A re-examination of text categorization methods. In *Proc. SIGIR*, pp. 42–49. ACM Press.

Yang, Yiming, and Jan Pedersen. 1997. Feature selection in statistical learning of text categorization. In *Proc. ICML*.

Yue, Yisong, Thomas Finley, Filip Radlinski, and Thorsten Joachims. 2007. A support vector method for optimizing average precision. In *Proc. SIGIR*. ACM Press.

Zamir, Oren, and Oren Etzioni. 1999. Grouper: A dynamic clustering interface to web search results. In *Proc. WWW*, pp. 1361–1374. Elsevier North-Holland. doi: http://dx.doi.org/10.1016/S1389-1286(99)00054-7.

Zaragoza, Hugo, Djoerd Hiemstra, Michael Tipping, and Stephen Robertson. 2003. Bayesian extension to the language model for ad hoc information retrieval. In *Proc. SIGIR*, pp. 4–9. ACM Press.

Zavrel, Jakub, Peter Berck, and Willem Lavrijssen. 2000. Information extraction by text classification: Corpus mining for features. In *Proc.Workshop Information Extraction meets Corpus Linguistics*. URL: http://www.cnts.ua.ac.be/Publications/2000/ZBL00. Held in conjunction with LREC-2000.

Zha, Hongyuan, Xiaofeng He, Chris H. Q. Ding, Ming Gu, and Horst D. Simon. 2001. Bipartite graph partitioning and data clustering. In *Proc. CIKM*, pp. 25–32. ACM Press.

Zhai, Chengxiang, and John Lafferty. 2001*a*. Model-based feedback in the language modeling approach to information retrieval. In *Proc. CIKM*. ACM Press.

Zhai, Chengxiang, and John Lafferty. 2001b. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proc. SIGIR*, pp. 334–342. ACM Press.

Zhai, Cheng Xiang, and John Lafferty. 2002. Two-stage language models for information retrieval. In *Proc. SIGIR*, pp. 49–56. ACM Press. doi: http://doi.acm.org/10.1145/564376.564387.

Zhang, Jiangong, Xiaohui Long, and Torsten Suel. 2007. Performance of compressed inverted list caching in search engines. In *Proc. CIKM*. ACM Press.

Zhang, Tong, and Frank J. Oles. 2001. Text categorization based on regularized linear classification methods. *IR* 4(1):5–31. URL: citeseer.ist.psu.edu/zhang00text.html.

Zhao, Ying, and George Karypis. 2002. Evaluation of hierarchical clustering algorithms for document datasets. In *Proc. CIKM*, pp. 515–524. ACM Press. doi: http://doi.acm.org/10.1145/584792.584877.

Zipf, George Kingsley. 1949. *Human Behavior and the Principle of Least Effort*. Addison-Wesley.

Zobel, Justin. 1998. How reliable are the results of large-scale information retrieval experiments? In *Proc. SIGIR*, pp. 307–314.

Zobel, Justin, and Philip Dart. 1995. Finding approximate matches in large lexicons. *Software Practice and Experience* 25(3):331–345. URL: citeseer.ifi.unizh.ch/zobel95finding.html.

Zobel, Justin, and Philip Dart. 1996. Phonetic string matching: Lessons from information retrieval. In *Proc. SIGIR*, pp. 166–173. ACM Press.

Zobel, Justin, and Alistair Moffat. 2006. Inverted files for text search engines. *ACM Computing Surveys* 38(2).

Zobel, Justin, Alistair Moffat, Ross Wilkinson, and Ron Sacks-Davis. 1995. Efficient retrieval of partial documents. *IP&M* 31(3):361–377. doi: http://dx.doi.org/10.1016/0306-4573(94)00052-5.

Zukowski, Marcin, Sandor Heman, Niels Nes, and Peter Boncz. 2006. Super-scalar RAM-CPU cache compression. In *Proc. International Conference on Data Engineering*, p. 59. IEEE Computer Society. doi: http://dx.doi.org/10.1109/ICDE.2006.150.

Библиографические ссылки, добавленные при переводе

К главе 2 (инвертированные списки, 2.4.1. Двусловные индексы)

Губин М.В. Изучение статистики встречаемости терминов и пар терминов в текстах для выбора методов сжатия инвертированного файла // Труды RCDL. — 2002. — Т. 2. — С. 26–38.

К главе 4 (построение индекса, 4.5. Динамическое индексирование)

Губин М.В. Электронная библиотека многоверсионных текстовых документов // Труды RCDL. — 2004. — С. 169–174.

К главе 8 (оценка качества, 8.6.3 Настройка работающей системы)

Agichtein, Eugene, Eric Brill, Susan Dumais, Robert Ragno. 2006. Learning User Interaction Models for Predicting Web Search Result Preferences // In: *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, 2006. — P. 3–10.

К главе 15 (машинное обучение, 15.4. Методы машинного обучения для поиска по запросу)

Liu, Tie-Yan. 2009. Learning to Rank for Information Retrieval. A tutorial at WWW2009. URL: http://www2009.org/pdf/T7A-LEARNING%20TO%20RANK%20TUTORIAL.pdf

К главе 19 (основы веб-поиска, 19.6. Нечеткие дубликаты и алгоритм шинглов)

Henzinger, Monika Rauch. 2006. Finding near-duplicate web pages: a large-scale evaluation of algorithms // In: *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, 2006. — P. 284–291.

Зеленков Ю. Г., Сегалович И.В. Сравнительный анализ методов определения нечетких дубликатов для web-документов // Труды 9-й Всероссийской научной конференции “Электронные библиотеки: перспективные методы и технологии, электронные коллекции” RCDL’2007: Сб. работ участников конкурса / Переславль-Залесский, Россия, 2007, Т.1 — С. 166–174.

Manku, Gurmeet Singh, Arvind Jain, and Anish Das Sarma. Detecting Near-Duplicates for Web Crawling // In *Proceedings of Proceedings of the 16th international conference on World Wide Web, 2007*. — P. 141–140.

К главе 20 (обход веба, 20.1.2. Желательные свойства поискового робота)

Abiteboul, Serge, Mihai Preda, Gregory Cobena. Adaptive On-Line Page Importance Computation // In: *Proceedings of the 12th international conference on World Wide Web*, 2003. — P. 280–290.

К главе 21 (веб-граф. 21.4. Библиография и рекомендации для дальнейшего чтения)

Gibson, David, Ravi Kumar, Andrew Tomkins. Discovering Large Dense Subgraphs in Massive Graphs // In: *Proceedings of the 31st international conference on Very large data bases*, VLDB'05, 2005. — P. 721–732.

Предметный указатель

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

Б

В

Г

Д

Е

З

И

К

Л

М

Н

О

П

P

C

T

У

Ф

Х

Ц

Ч

Ш

Э

Я