Patri Friedman
CS 224N
Final Project

Automated Web Search Classification and Refinement

Motivation

Searches of the entire World Wide Web using search engines such as Google have become an extremely common way of locating information. Searches are usually keyword based, which has advantages and disadvantages. The challenge for the user is to come up with a set of search terms which is neither too large (making the search too specific and resulting in many false negatives) or too small (making the search too general and resulting in many false positives). This process can sometimes be time-consuming, and might benefit from automation.

A particular type of search motivated this project, namely a "vanity" search where people look for mentions of themselves on webpages. While first/last name combinations can be fairly unique ("patri friedman"), one may wish to find mentions by only first name, and first names tend to be very non-unique. The techniques used are completely general, however. The important thing is the *characteristics* of the search, in this case the desire to locate all pages from a certain class which contain a certain base term which often occurs outside the class as well. It is the desire of the search to be **complete** (find all true positives) and **precise** (find few false positives) which makes the methods described here worth using. If one just wants to find a few true positives, they are unnecessary. Another type of search which would be useful is one which is periodically repeated, and thus it is worth generating an accurate search because it will be used many times. An example might be searching for a particular type of news story.

Proposed Solution

Statistical NLP is of course the answer. The user provides the base term and classifies search results for that term alone. This generates a training corpus. The software helps the user to refine their search with both positive and negative terms in order to efficiently find more true positives, which may be quite sparse in searches for the base term only. When enough data has been gathered, a search string is constructing using the corpus, and the results are classified based on the corpus, with only positives shown.

?? Disclaimer:

Disclaimer: Chose to focus on creating a practical, usable package. Thus some time was devoted to UI and automation that was not available to do experimentation with different methods. The report will not have as many comparisons of different experiments with different techniques as might be in a more scientific setting. Generally when a problem is encountered, a single reasonable solution is found, when it would be better to experiment with several reasonable solutions and compare them. However the package is usable, deployed, and works. I urge you to experiment with it yourself to see its functionality, I

put a lot of work into it and am concerned that this may not show if you just read the report :). This may just be because I am a perfectionist, and the amount of work done is adequate. I dunno.

Methods

Software

The package was implemented in Java. The user interface is done through a Java servlet. WebMacro was used as a templating engine. Google provides a Java object to access their SOAP API, which made searching from code quite easily (yet another way that Google enriches our lives). A free Java html parser from Quiotix was used, and the tokenizing code was based on classes from Lucene, a search engine hosted by the Apache Jakarta project. The source code is available at:

http://patrifriedman.com/projects/code/patri-224n-final-src.tar.gz

and a demo will be running at:

http://clevername.net/classify/

For a couple weeks. (If it doesn't work, please email me)

It is important to note here the cost function under which we are operating. The Google API allows only 1000 searches per day for each registrant, where getting 10 results for a query counts as 1 search (so getting 100 results for a query counts as 10 searches against the quota). In order to classify webpages, they have to be downloaded, which is moderately costly. This is why search refinement is important as well as classification (and ended up being our focus, due to time considerations).

For example, consider the following approach. Find and download the top 1000 hits for "patri" on Google. Show 10 at a time, getting classifications, until there is at least one positive. Build a classifier, and rank the remaining pages according to likelihood of being positives. Let the user continue to classify them, periodically re-building the classifier and re-evaluating it on the data set. When there are no unseen pages which are classified as positive remaining, download another large batch, and keep downloading until a decent number of probably positives are found.

This would eliminate the problem of refining searches and leave it as a pure classification problem. Refinement would simply be unnecessary, since all desired matches appear somewhere among the 230,000 pages which match "patri". The bias introduced by searching for auxiliary terms would be eliminated. Unfortunately, this is simply too costly. We would have to perform many Google searches, and we would have to download many webpages – most of which are negatives.

We must note, however, that neither of these costs would apply if the search engine itself were performing the algorithm! It has the web in RAM, it can do many searches cheaply,

and has fast access to cached versions of all the webpages it has indexed. This means that it can perform this procedure more effectively than we can. It also has the most to gain from improving its own searches.

Overview & User Inteface

So we have two stages – obtaining data and using the model. First the user enters the base term and an initial number of hits. There should be enough hits that there will be some positive instances found. The algorithm could be improved to deal with very rare positive instances by allowing auxiliary terms to be guessed by the user from the beginning, but this was not implemented.

The google search is performed, and the results are listed, with radio buttons "Dunno, Yes, No" for classification. They are initially all set to "No". Each result has a summary listed, and a link to the page. The links pop up in a separate window, so that they can be examined while classifying without having to go forward and back. When the user has classified this data set, he clicks a submit button.

The webapp then stores these classifications and downloads all of the webpages, converts them to html, and tokenizes them. It computes unigrams on the positive and negative set, and uses the information gain metric to calculate the best features, writing them to a file with the unigram, IG, and probability log difference (used for classifying via naïve bayes). It then presents the user with the top 20 positive features, including their score and information gain.

The user can choose any of these features for search refinement by clicking a button (optionally entering the number of results). The webapp performs a google search on the base term plus the feature, filters out pages that have already been classified, and each hit is downloaded, texted, tokenized, and scored. The results are listed, again with radio buttons for classification. To make things easier for the user and demonstrate how classification is working with the current corpus, we set the radio buttons based on how our model classifies these webpages, as well as giving the score. The score is equivalent to the log of the probability ratio that the document is positive vs. negative, so a positive score indicates a positive document and a negative score a negative one.

When we are refining a search, we do not filter out pages we classify as negative, because it is good to get more negative examples in our corpus as well as positive ones.
The list of pages while refining the search is sorted from least to most certain (ie by absolute value of score, from lowest). If classification is working well, this lets the user focus their attention on evaluating the first listed results, and skim the remaining ones.

The user can also choose to do a negative refinement. In this case, the algorithm looks for features which have no true positives (never occur in positive training instances), and sorts them by the number of true negatives (negative webpages which contain the feature). This metric identifies terms that we can exclude and will increase the proportion of true positives while eliminating few or zero true positives (our goal when doing a complete search). The top 20 are shown, and a search can be performed with these removed.

To make negative feature use more accurate, it might be worth doing a search on that feature **included** and scoring the top results to see if any result in a positive classification. That is, the webapp might search on “patri gloria” and see if any of the top 20 pages appear to be positive. This would help us to not eliminate true positives from the search space while not imposing a further classification burden on the user.

Finally, once the user has used positive and negative refinement to widen the training corpus, he can do a full search, optionally including or excluding already classified pages. In the full search, a google query is generated that includes the base term, the negative terms (excluded), and a boolean OR of the top positive features. The model is applied on the results, and they are listed by score.

Converting html to text

This is done with the Quiotix open-source html parser, and a routine I developed during some consulting work for extracting the text. We remove javascript. We ignore tags, exept that we treat them as delimiters, since they act as lexical separators. We treat the title as part of body, since it is likely to provide key tokens (it could be a separate context). Bad html sometimes screws up our parser, causing tags to sometimes be treated as text, which is unfortunate. The impact of this was minimized because we used binary features, and by adding the most common tags to the stop-words list.

Tokenizing

The tokenizing code was based on some classes for the Lucene search engine, modified for this particular use. We first split on whitespace plus “.”, “-”, and “/”. We then lowercase all tokens. We filter out escaped html such as “ ”, filter bad chars (unicode indicating non-ascii characters, anything that isn't a letter, digit, or one of a few allowed punctuation characters). Finally we apply a stop-word filter to remove common characters, including several specialized for this use such as “com, org, http, href”. We also remove 1 & 2 character long tokens. Porter stemming was not used because we want to be able to search for our tokens, and stemming modifies them.

The package keeps track, for each document, of the search used to find it, and removes the search tokens when tokenizing. When computing probabilities, this is taken into account in computing the number of potential matches. So if “patri” yields 10 positives, 5 of which contain “friedman”, and the next search is for “patri friedman” and yields 30 positives, “friedman” is still considered as occuring in 5/10 positive documents. Otherwise friedman would be biased much too high (35/40) or much too low (5/40).

It is important to remember that every data set is biased by the search used to find it. That is, we should expect that $P(S \mid R) \mathrel{!=} P(S \mid R(w))$, where S is some logical proposition or statistic of the document, R is a random web document, and R is a random web document which contains a certain word and is high up in the listing when searching for that word. The bias introduced by our base term, we ignore, because it is part of every search we make, and so does not affect us. The bias introduced by various auxiliary terms used in different searches, we also ignore, because it would be difficult to

deal with (ie perhaps our corpus is biased because of the particular auxiliary terms used to generate it) . The exception is the bias toward its own presence introduced by an auxiliary term, which is so pronounced that we must deal with it, and also quite easy to deal with by simply not considering such documents as being relevant to the frequency with which that term occurs.

Because we use a binary metric on tokens, webpages which have several copies of the same page at different URL's would have a high impact on results. A feature to analyze pages and eliminate identical ones would be useful. (Google may do this already).

Smoothing

Smoothing is not as much of an issue in this context as in some others, although it still matters. The reason is that all our calculations are on relative probabilities between two corpora. So if a token does not appear in **either** corpora, we have the luxury of not needing to smooth it at all. When we encounter it, it indicates nothing about which class this document belongs in, so we ignore it. We only need to smooth when a token appears in one set, but not the other. Since our initial classes are often highly skewed (towards the negative), this is something we need to worry about.

First, we throw out any tokens that did not occur at least 3 times in either corpora as indicating too little information to be useful. (2 was used instead of 3 when there were less than 10 url's in the class, and no minimum with less than 4). When calculating the score to use for a feature, which is a function of the probability of the feature occurring in each corpora, we pretend that unseen features were seen 0.1 times. This was an arbitrary choice, and should be done with something better, ideally Good-Turing smoothing. Experimentation suggested that modest variations of this value did not make a big difference on the results.

Features

The only feature we used was unigrams in the entire context of the webpage. We counted them by binary presence in each document. Originally we used frequency across each corpora, but this caused problems when one long doc dominated (especially when the long doc had bad HTML). Binary features should work particularly well with multiple context windows and backoff smoothing, as discussed in a moment.

We selected top features, both when trimming down the number to improve computation speed, and when presenting features for auxiliary searches, via the Information Gain metric. It measures the amount by which this feature reduces entropy. Originally the frequency difference was used, but it did not perform as well.

Binary features did cause some problems. Since the goal is completeness, small areas of the search space are still desirable. But when there are large clusters of positives, they tend to drown out the smaller features. For example, “juggling”, which is feature #10 after the initial base search, and empirically results in an extremely precision when used as an auxilliary term, is ranked in the low 30's after a refinement on “patri friedman”.

Refinement is likely to have positive feedback, continuing to explore the same part of the search space. This is discussed further under results.

Some other potential features were: title (which we just included in the body), internet domain, words in pages this page links to, text between anchor tags linking **to** this document (Riboni).

The most effective extension of the basic algorithm which I did not have time to implement, and would really like to, is multiple context windows. Phrases like “gloria patri”, “tom patri” and “in nomine patri” are all clear non-matches which would be quickly and effectively learned by an algorithm. The proposed method is that when tokenizing a document, rather than throwing all tokens into one pool, we create multiple pools based on distance from the base term. The sizes of the pool would initially be about phrase length (1 word each way), sentence length, and paragraph length, but could be anything, and should be studied empirically.

As the data for smaller windows would be quite sparse, backoff smoothing would be used. That is, when smoothing the phrase context, the probability mass assigned to unseen words is allocated according to the words probability in larger contexts (probably the smallest such larger context in which it occurs). This should be a very effective smoothing method.

It is unclear how exactly to combine the scores from the various context pools. We could pretend that the conditional probabilities are independent, and add the scores, which would be equivalent to multiplying the probability ratios P(+ | context 1) * P (+ | context 2) ... / [P(- | context 1) * P(- | context 2) ...]. Or we could do some linear combination, perhaps based on empirical testing.

Note that these multiple pools allows for some html-specific features to be easily incorporated into the model. The title of the document might be a seperate pool, and the URL as well (with a custom tokenizer). It seems likely that enhancing features in this way would contribute substantially to classification success, and it is quite unfortunate that there was not enough time to implement it before turning in this report. However, I will likely implement it anyway, and if you wait until 6/7 or 6/8 to check out the live demo webapp, this feature should be added.

Scoring / Classification

The naïve bayes model was used. The score for a feature was $\log(w_+) - \log(w\_)$, where w_+ = frequency of word in the positive classification and w_ = frequency in negative classification. (If w was used as a search term, the documents found that way were not counted for calculating this frequency). If w_+ or w_ was zero, smoothing was used. The score for a document was then the sum of the scores for each feature (each token that appeared at least once). This represents (given the naïve bayes assumption of independence between tokens) log (p (doc is +) / p(doc is -)).

Full Search

Full search is simply done by creating the query “base_term (aux1 OR aux2 OR ...) -neg1 – neg2 ...”. The results are filtered for positive scores and presented.

Results

A search for webpages about Patri will be used as the example. We'll start out by giving many of the preliminary results which motivated improvements in the algorithm, sort of a narrative journey through the iterative process of development. Finally we'll give an example of a run with the final version of the code, including some statistics on accuracy.

Initially when we searched on“patri” - of 100 pages, only 3 are true positives. Here we see the top features by probability difference across the two corpora:

Feature PD 1 For feature "the" posProb=0.462 negProb=0.105 delta=0.358
Feature PD 2 For feature "nbsp" posProb=0.6 negProb=0.257 delta=0.-250
Feature PD 3 For feature "is" posProb=0.237 negProb=0.46 delta=0.191
Feature PD 4 For feature "juggling" posProb=0.179 negProb=0.0 delta=0.179
Feature PD 5 For feature "of" posProb=0.232 negProb=0.78 delta=0.154
Feature PD 6 For feature "it" posProb=0.153 negProb=0.15 delta=0.138
Feature PD 7 For feature "contact" posProb=0.137 negProb=0.4 delta=0.133
Feature PD 8 For feature "cj" posProb=0.116 negProb=0.0 delta=0.116
Feature PD 9 For feature "ball" posProb=0.110 negProb=0.0 delta=0.110
Feature PD 10 For feature "this" posProb=0.122 negProb=0.14 delta=0.108

Only 4 of these (juggling, contact, cj, ball) can be considered non-trivial. Short words and the html escaped   have a high presence. Because of this, I next tried implementing the Information Gain metric, as suggested by Forman. Its rankings were much more intuitive, with the top 10:

Feature IG 1(4) For feature "juggling" Information Gain=2.2545200270019175E-4
Feature IG 2(8) For feature "cj" Information Gain=1.4568313314596536E-4
Feature IG 3(9) For feature "ball" Information Gain=1.3750992893684331E-4
Feature IG 4(20) For feature "balls" Information Gain=7.255939700433027E-5
Feature IG 5(7) For feature "contact" Information Gain=6.521192711235815E-5
Feature IG 6(1) For feature "the" Information Gain=6.408801499430648E-5
Feature IG 7(2) For feature "nbsp" Information Gain=4.395513208715722E-5
Feature IG 8(6) For feature "it" Information Gain=4.1554376177593486E-5
Feature IG 9(3) For feature "is" Information Gain=3.787776410260971E-5
Feature IG 10(59) For feature "moschen" Information Gain=3.559755442507882E-5

The ranks in parenthesis are the ranks of each feature under the PD metric. While some of the same extraneous terms appear, the top 5 are all nontrivial tokens. "friedman", the most obvious auxiliary term to use (and one which empirically results in high recall and high accuracy) is 126 under PD and 20 under IG. Thus IG was used as the metric.

Note that this list is very biased by the particular positive pages in the search set, because of the extreme class skew (3 / 100), and the fact that one of the 3 positive cases is patri's page on Contact Juggling, which is much longer than the other 2. This motivated moving to binary features.

Here we note that a refinement on “patri juggling” was extremely successful in terms of

precision - the top 20 hits were all positive cases. However it explored a very limited portion of the search space, essentially all pages of links which point at patri's CJ page. We should expect that some auxiliary terms (“friedman”) will result in a broad exploration of the search space, while others (“juggling”) will explore a very narrow area. There is unfortunately likely to be positive feedback where exploring a homogenous cluster reinforces the weighting of a small set of terms. Some mechanism is necessary to avoid this.

Currently, allowing the user to choose which positive terms to refine on is our crude first step. Ideally, some sort of correlation analysis would be done among the features with high information gain. Rather than just using IG as the metric, some combination of IG and low correlation with terms already used for refinement would be superior. Similarly, when constructing a final query, if we just go by IG, then the auxiliary terms in our boolean OR may all be highly correlated, and we are wasting limited query length. We would rather use a representative term from each cluster, ie a set of OR'd terms with minimal cross-correlation. Doing correlation analysis should not be difficult.

Next we added the stop word filter to remove common and short words. This was very effective, as the top 10 positive features (note that some of these lists are of all features, and some are of positive features only) became:

1 juggling 421.2728570655938
2 ball 241.3907729952175
3 contact 177.50234227200124
4 balls 143.83793520058308
5 friedman 80.8110926280775
6 page 72.55664779973469
7 moschen 63.30863667187159
8 has 60.17382662006909
9 pictures 59.08822384161905
10 patris 53.95503966521831

Next we classified the top 20 results for “patri juggling” (all of which were positive). As desired, our best feature has extremely high precision. After only a single classification set with only 3 positive matches, we've found a token which results in perfect recall (at least on the top 20). This indicates the potential strength of the search refinement technique. Unfortunately, the top positive features became the following:

1 hrefindex 1130.5996612123836
2 targetblank 1130.5996612123836
3 return 1026.4946964279598
4 onmouseoverfwww 889.9460562221461
5 abr 751.0712342837955
6 contact 338.5392464982928
7 abra 337.5193619661876
8 hrefhttp 275.23817274430763
9 circus 271.0851430505645
10 page 261.6249467832121

Upon investigation, it was found that one of the matches was a long page with html that was misinterpreted by the parser, resulting in many tags and javascript being treated as

text. Because the page was long compared to most positive examples, it had a major impact on the results. It was here that binary features were switched to. Also the search was made english-only, which gave 8 positives in the initial 100 rather than 3. Next we see the top features by information gain. The (1.0) indicates a positive feature, as opposed to (-1.0) which is a negative one. Because positive pages are more homogenous, the best features at this point are all positive.

Feature IG 1 For feature "friedman" Information Gain=7.487199599933159E-4 (1.0)
Feature IG 2 For feature "patris" Information Gain=4.6937111114525587E-4 (1.0)
Feature IG 3 For feature "patriizzy" Information Gain=3.8152754660264465E-4 (1.0)
Feature IG 4 For feature "modified" Information Gain=3.393186945496729E-4 (1.0)
Feature IG 5 For feature "bio" Information Gain=3.175213154777423E-4 (1.0)
Feature IG 6 For feature "url" Information Gain=3.175213154777423E-4 (1.0)
Feature IG 7 For feature "webpage" Information Gain=3.175213154777423E-4 (1.0)
Feature IG 8 For feature "pictures" Information Gain=2.7792063930554E-4 (1.0)
Feature IG 9 For feature "moving" Information Gain=2.474545506258585E-4 (1.0)
Feature IG 10 For feature "plans" Information Gain=2.474545506258585E-4 (1.0)

We can see that this list is much less juggling-centric than the previous one, since Patri's Contact Juggling Page was much longer than the few other pages in the positive set, it had dominated the results. Also the english-only brought in other webpages which were not about juggling topics. Finally we have the token "friedman" at the top of the list, which intuitively is the best auxiliary token.

Next we refine the search using "friedman" to widen our class of positive results. All 36 new results in the 50 results fetched were positive cases. As desired, the algorithm has, after a single classification set created from the base token, found a token which results in extremely high precision. And this is without even using multiple context windows. The best features after adding these results were:

Feature IG 1 For feature "poker" Information Gain=2.5788835274320743E-4 (1.0)
Feature IG 2 For feature "jack" Information Gain=2.2940212665478743E-4 (1.0)
Feature IG 3 For feature "father" Information Gain=2.21952379735879 1E-4 (-1.0)
Feature IG 4 For feature "reserved" Information Gain=2.107464855823915E-4 (-1.0)
Feature IG 5 For feature "mike" Information Gain=2.0507558220417277E-4 (1.0)
Feature IG 6 For feature "players" Information Gain=2.0507558220417277E-4 (1.0)
Feature IG 7 For feature "tournament" Information Gain=2.0507558220417277E-4 (1.0)
Feature IG 8 For feature "gloria" Information Gain=2.00702772042205E-4 (-1.0)
Feature IG 9 For feature "play" Information Gain=1.9162400538619284E-4 (1.0)
Feature IG 10 For feature "played" Information Gain=1.8494425951887106E-4 (1.0)

The presence of names on this list is rather strange. It appears that lists of names are more likely to be positive cases than negative ones, and thus some common names are positive features. The absence of “friedman” on this list was due to a bug in how I was counting frequency of tokens which were included in search results.

At this point I wondered if a way to address the problem of skew due to auxiliary terms was to use negative features rather than (or as well as) positive ones. In particular, features with some false positives and no true positives seemed ideal, as their elimination would remove undesired pages without removing desired ones. Thus a "refine by negatives" option was added. "refining by positives" is still useful to increase the number

of positive cases when it is very small. Negative refinement's effectiveness would depend on the homogeneity of the negative class, and the class skew. But it should introduce less bias. It will also be less precise, resulting in many more negative cases than we saw with “juggling” and “friedman”, but during the refinement phase we don't mind getting some more negative cases as long as we get a decent number of positive ones as well.

Another option would be to generate a query using a boolean "OR" of many positive features, and this was in fact used for the full search

Here are the top negative features selected. All have zero true positives, and the score listed is the number of false positives:

Neg feature 1 for feature artists fp-tp=11.0
Neg feature 2 for feature historic fp-tp=10.0
Neg feature 3 for feature graphic fp-tp=7.0
Neg feature 4 for feature spiritual fp-tp=7.0
Neg feature 5 for feature architects fp-tp=6.0
Neg feature 6 for feature awards fp-tp=6.0
Neg feature 7 for feature declared fp-tp=6.0
Neg feature 8 for feature flags fp-tp=6.0
Neg feature 9 for feature prayer fp-tp=6.0
Neg feature 10 for feature pugliese fp-tp=6.0

There is an architect named patri, as well as several artists. I was named after Patri Pugliese, a friend of the family, and while its possible that a webpage might mention both of us, the current set does not include any such. A manual search for "patri pugliese friedman" found no positive matches, which suggests yet another improvement I don't have time to make (as mentioned earlier).

When considering negative features, a search could be constructed with the base term, the negative term, and a number of positive terms. Results would be evaluated according to the model. If any results which appeared positive were found, the negative feature would not be used. If there were a small number of anomalous positive results, they could be stored in a special list, and the negative feature used anyway.

A manual search for "patri friedman artists" does reveal some positive matches however, which is undesirable. Having a large pool of positive instances is probably important in making sure that negative features are chosen well (that they don't result in false negatives).

After increasing the size of the pool by doing another positive refinement (“patri friedman” with a larger search size) a negative refinement test was done with these terms:

Neg feature 1 for feature artists fp-tp=15.0
Neg feature 2 for feature shipping fp-tp=12.0
Neg feature 3 for feature ancient fp-tp=11.0
Neg feature 4 for feature graphic fp-tp=9.0
Neg feature 5 for feature historic fp-tp=9.0
Neg feature 6 for feature meetings fp-tp=9.0

Neg feature 7 for feature registration fp-tp=9.0
Neg feature 8 for feature warm fp-tp=9.0
Neg feature 9 for feature delivered fp-tp=8.0
Neg feature 10 for feature landscape fp-tp=8.0

On the downside, almost all of the pages found with this negative refinement which had not been seen already were negative hits. Because we are searching on only the base term as a positive term, it is still tough to find positive hits (especially other than the ones we've already found). However there is less skew to what we are finding. This suggests some combination of positive and negative features is necessary for effective search. Another note is that some mostly-negative features (notable "gloria") have a few positive appearances, yet are strong negative indicators. Multiple contexts or bigrams would be extremely effective here, as "gloria patri" is a common and entirely negative phrase.

In the final version of the software (at least at report-due time), we have:

Feature 1"friedman" Info Gain=0.19826772736979761 weight 3.5409593240373143
Feature 2"patris" Info Gain=0.11311575838694837 weight 1.9315214116032138
Feature 3"patriizzy" Info Gain=0.11174016117468066 weight 5.84354441703136
Feature 4"modified" Info Gain=0.07625006362441283 weight 2.2192034840549946
Feature 5"bio" Info Gain=0.0710330240062142 weight 2.847812143477369
Feature 6"webpage" Info Gain=0.0710330240062142 weight 2.847812143477369
Feature 7"carpet" Info Gain=0.07082938453078258 weight 5.438079308923196
Feature 8"fascinating" Info Gain=0.07082938453078258 weight 5.438079308923196
Feature 9"hypnotic" Info Gain=0.07082938453078258 weight 5.438079308923196
Feature 10"juggling" Info Gain=0.07082938453078258 weight 5.438079308923196
Feature 11"listed" Info Gain=0.07082938453078258 weight 5.438079308923196
Feature 12"poker" Info Gain=0.07082938453078258 weight 5.438079308923196
Feature 13"pool" Info Gain=0.07082938453078258 weight 5.438079308923196
Feature 14"slowly" Info Gain=0.07082938453078258 weight 5.438079308923196
Feature 15"survival" Info Gain=0.07082938453078258 weight 5.438079308923196
Feature 16"trackback" Info Gain=0.07082938453078258 weight 5.438079308923196
Feature 17"tub" Info Gain=0.07082938453078258 weight 5.438079308923196
Feature 18"pictures" Info Gain=0.06036482792975578 weight 2.4423470353692043
Feature 19"home" Info Gain=0.05311399062727856 weight -3.523799703754487
Feature 20"moving" Info Gain=0.05219175549459576 weight 2.1546649629174235

Refining on “friedman”, 60 results, 45 of which were new and downloadable. Of these 45, 43 were positive, demonstrating the effectiveness of information gain at finding precise auxiliary terms. However the two negative cases were mis-classified as positive.

Feature 1"poker" Info Gain=0.205825869269065 weight 5.804426000435889
Feature 2"friedman" Info Gain=0.19826772736979761 weight 3.5409593240373143
Feature 3"mike" Info Gain=0.15417020834130213 weight 5.553111572154982
Feature 4"play" Info Gain=0.15063551387418372 weight 1.9552038962467728
Feature 5"players" Info Gain=0.15063158812956068 weight 3.3840578717854592
Feature 6"steve" Info Gain=0.11767768411008084 weight 2.6263721700879428
Feature 7"tournament" Info Gain=0.11767768411008084 weight 2.6263721700879428
Feature 8"chips" Info Gain=0.11753387291696271 weight 5.311949515338094
Feature 9"holdem" Info Gain=0.11753387291696271 weight 5.311949515338094
Feature 10"jack" Info Gain=0.1148215360237671 weight 3.176418507007215

Feature 11"limit" Info Gain=0.1148215360237671 weight 3.176418507007215
Feature 12"played" Info Gain=0.1148215360237671 weight 3.176418507007215
Feature 13"player" Info Gain=0.1148215360237671 weight 3.176418507007215
Feature 14"cards" Info Gain=0.10644069654237465 weight 2.557379298600991
Feature 15"bets" Info Gain=0.1056932697625299 weight 5.2166393355337695
Feature 16"big" Info Gain=0.09856455404246711 weight 1.997763510665569
Feature 17"win" Info Gain=0.09856455404246711 weight 1.997763510665569
Feature 18"david" Info Gain=0.09588713955577444 weight 1.7100814382137879
Feature 19"last" Info Gain=0.09531463477252189 weight 1.0533019018247176
Feature 20"casino" Info Gain=0.09403150835488516 weight 5.111278819875944

Clearly the mentions of Patri in the context of poker make up a large proportion of the hits for “patri friedman”. Next, we refine on “juggling” from the first list, fetching 40 search items. 34 were new and downloadable, and 28 of these were positive cases.

eature 1"contactjuggling" Info Gain=0.14016653244314103 weight 5.534039700069107
Feature 2"clubs" Info Gain=0.13226528367236778 weight 5.482746405681556
Feature 3"videos" Info Gain=0.13226528367236778 weight 5.482746405681556
Feature 4"circus" Info Gain=0.12445018900051219 weight 5.42867918441128
Feature 5"spinning" Info Gain=0.12445018900051219 weight 5.42867918441128
Feature 6"ball" Info Gain=0.11622072153340401 weight 3.2314546070750607
Feature 7"tricks" Info Gain=0.10517981710640312 weight 2.27260426078211
Feature 8"links" Info Gain=0.1020978826638046 weight 1.2342511636321971
Feature 9"yahoo" Info Gain=0.0981332605987767 weight 2.2281524982112764
Feature 10"bowers" Info Gain=0.0940203951027776 weight 5.177364756130374
Feature 11"jugglers" Info Gain=0.0940203951027776 weight 5.177364756130374
Feature 12"notation" Info Gain=0.0940203951027776 weight 5.177364756130374
Feature 13"few" Info Gain=0.09260619163914119 weight 1.3343346221891794
Feature 14"jack" Info Gain=0.08687764160139333 weight 3.0083110557608506
Feature 15"instructions" Info Gain=0.08661395847584552 weight 5.1032567839766525
Feature 16"izzy" Info Gain=0.08661395847584552 weight 5.1032567839766525
Feature 17"pentix" Info Gain=0.08661395847584552 weight 5.1032567839766525
Feature 18"club" Info Gain=0.08458658790492102 weight 1.6628386891612157
Feature 19"poker" Info Gain=0.08439646144953428 weight 2.4329469108572894
Feature 20"gloria" Info Gain=0.08105027449247293 weight -4.900076103529193

Since this search results in very homogenous webpages, it has almost overwhelmed the feature list, although “poker” is still on there (and others are farther down). Notice that “gloria” has made it on as a negative feature.

Next, a negative refinement for 100 search hits was performed:

Neg feature 1 for feature gloria fp-tp=17.0
Neg feature 2 for feature spirit fp-tp=10.0
Neg feature 3 for feature description fp-tp=9.0
Neg feature 4 for feature voice fp-tp=9.0
Neg feature 5 for feature fax fp-tp=8.0
Neg feature 6 for feature base fp-tp=7.0
Neg feature 7 for feature charge fp-tp=7.0

Neg feature 8 for feature established fp-tp=7.0
Neg feature 9 for feature heaven fp-tp=7.0
Neg feature 10 for feature historic fp-tp=7.0
Neg feature 11 for feature joined fp-tp=7.0
Neg feature 12 for feature languages fp-tp=7.0
Neg feature 13 for feature nevertheless fp-tp=7.0
Neg feature 14 for feature photography fp-tp=7.0
Neg feature 15 for feature prayer fp-tp=7.0
Neg feature 16 for feature presented fp-tp=7.0
Neg feature 17 for feature themselves fp-tp=7.0
Neg feature 18 for feature wedding fp-tp=7.0
Neg feature 19 for feature advance fp-tp=6.0
Neg feature 20 for feature bachelor fp-tp=6.0

Bachelor though Patri is, apparently it is a negative term and not a positive one in association with his name on the web. Of the 100 pages, 40 were new and downloadable., and all were negative instances. This suggests that while negative terms may help make our searches more efficient (which is important in our context), they are not sufficient to locate positive occurrences when they are sparse on searches for only the base term. 36 of the 40 were classified as positive. There is something broken about the scoring system, probably a bug, but I am out of time to fix it. The positive feature list is quite similar to last time:

Feature 1"contactjuggling" Info Gain=0.14250713066844267 weight 5.87051193669032
Feature 2"circus" Info Gain=0.12671969086481527 weight 5.765151421032493
Feature 3"spinning" Info Gain=0.12671969086481527 weight 5.765151421032493
Feature 4"tricks" Info Gain=0.11627172238472783 weight 2.609076497403323
Feature 5"videos" Info Gain=0.11465785748376245 weight 3.5166335493087235
Feature 6"links" Info Gain=0.11170676589647743 weight 1.3765673858124523
Feature 7"yahoo" Info Gain=0.10898665630532489 weight 2.5646247348324893
Feature 8"ball" Info Gain=0.1069371569722698 weight 2.874779663136329
Feature 9"few" Info Gain=0.10284501683993352 weight 1.5037527741472263
Feature 10"patris" Info Gain=0.10272557015612227 weight 1.6082865018290584
Feature 11"clubs" Info Gain=0.09967681026292452 weight 2.823486368748778
Feature 12"bowers" Info Gain=0.09606163751255603 weight 5.513836992751587
Feature 13"jugglers" Info Gain=0.09606163751255603 weight 5.513836992751587
Feature 14"notation" Info Gain=0.09606163751255603 weight 5.513836992751587
Feature 15"poker" Info Gain=0.09252605406905134 weight 2.7694191474785024
Feature 16"instructions" Info Gain=0.08858113579630844 weight 5.439729020597865
Feature 17"izzy" Info Gain=0.08858113579630844 weight 5.439729020597865
Feature 18"pentix" Info Gain=0.08858113579630844 weight 5.439729020597865
Feature 19"juggle" Info Gain=0.0851737051528545 weight 3.2802447712444933
Feature 20"play" Info Gain=0.083760685270593 weight 1.5654463431525665

Finally, a full search was performed, with the top 20 positive aux terms OR'd and the top 20 negative terms excluded. 100 search hits were retrieved, 58 of which were new. Only 8 of these were positive examples. Apparently OR'ing together many terms at once degrades the results greatly, which is interesting but makes sense. Separate searches should be done, and the results agglomerated.

Further Work

This method should be easily extensible to Usenet discussion groups. Because of their

dynamic nature as continuous discussions, having specific searches that one periodically repeats is natural to the domain. Thus the time spent training a good classifier would be worthwhile in order to have many effective later searches.

The scoring function is broken. If it is not easy to fix, it may turn out that offsetting it by a constant amount is effective. One idea would be to use the refinement data to do this. That is, when the user does a refinement, they are classifying scored pages. By observing the classification of various scores, we might see that shifting the boundary would result in better performance. Also, we could score the initial corpus (from the first search on the base term only).

Performance Measure: Lots more work on evaluating the performance of the classifier, and experimenting with ways to get it to perform better is clearly necessary.

Full search could be improved by reducing correlation between OR terms without even calculating correlation through this simple procedure. Form a list of positive auxiliary terms with significant information gain. Do a search on “base aux1 – neg...”, and get the top 10 or 20 hits. Filter them for positive results. Now, for each remaining aux term, count how many times it appears in those hits, and make that its score. Order aux terms lexically, first by score then by IG. Use the aux term with the lowest score, breaking ties by highest information gain. Iterate until you have the requested number of positive hits. This would implicitly take cross-correlation into account. I may add this feature over the next few days.

While the basic framework is in place, in some ways it seems as though most of this project is further work. In terms of the effort : reward curve, the current codebase is still on the steep upslope, in fact it has just reached it (the curve looks like a sigmoid, I suspect). I am unfortunately left feeling as though I've mostly done the annoying and less rewarding work. This suggests that further work would be effective. It would be optimal if someone would pay me to turn this initial prototype into a late-stage prototype, but that seems unlikely. Another option would be to open-source the code and see if anyone else wants to finish it. It would certainly be desirable if there was more open-source java code for doing NLP. For example a Good-Turing smoothing routine would have been nice.

Conclusion

Certain types of web searches benefit greatly from refinement, and statistical NLP can be used to perform the refinement. There are a number of pitfalls and subtleties involved, but there is clearly potential to navigate them successfully. While the existence of some open-source Java packages makes life much easier, a project like this is still a great deal of work for one person. The package I have created could be easily and substantially improved.

References

Forman, George: An Extensive Empirical Study of Feature Selection Metrics for Text

Classification. (Note – from the name, this guy could use my project!)

Manning, Christopher and Schutze, Hinrich: Foundations of Statistical Natural Language Processing.

Riboni, Daniele: Feature Selection for Web Page Classification.