CS224N Final Project
Summarizing Movie Reviews

Dan Fingal, Jeff Michels, Jamie Nicolson

June 3, 2004 (1 late day)

1 Abstract

Text summarization is a classic problem in natural language processing. Given a body of text, is there an automated way to generate a few sentences that sum up its content? Using movie reviews downloaded from RottenTomatoes.com, along with summary sentences provided by the site, we attempt to find statistical machine learning methods to find acceptable summary sentences in previously unseen movie reviews. The task is inherently difficult due to the relatively unstructured nature of online movie reviews, the large variability in writing styles, and the presence of many possible "good" sentences among which only one will be tagged as the correct "RottenTomatoes" choice. Our best system first classifies each review as either positive or negative in opinion and then uses a unigram language model along with a ranking support vector machine to guess the correct sentence with 26% precision. Human precision on this task was tested to be 40%. In addition, many of the "incorrect" sentences that the system returns are subjectively plausible summaries.

2 Research Questions

2.1 Text Summarization

One approach to the problem of text summarization is to search the document for sentences that use language that normally occurs in statements of broad description, judgement, or reflection. Our aim is to build a system that, given a list of the sentences in the document, can detect which of these qualify as providing a summary of that document's content. We attempt to do this within a machine learning context, in which the system distills the relevant features through training on a large corpus of tagged data, as opposed to a heuristic method, in which hand engineered detectors provide the discriminating factors.

Kupiec et al. [3] extract summaries from technical documents using a very small feature space. Their features are sentence length; the presence of certain fixed phrases, such as "In conclusion"; the position of the sentence within

a paragraph; and the presence of capitalized words. When their algorithm is told how many summary sentences there are, it achieves 42% accuracy on test data. Technical documents, however, are far more structured than online movie reviews. Kupiec uses fixed phrases such as "Therefore," "In conclusion," etc. along with section titles to help determine whether or not a sentence would make a good summary. Movie reviews typically do not have labeled subsections or catch phrases such as these. In addition, technical articles more often are divided into paragraphs that have introductory and concluding sentences, so a lot of mileage can be gotten from simply looking at the position of the sentence within the paragraph. Movie reviews, on the other hand, often contain misleading information. A review may begin with a description of the reviewer's expectations, only to later go on to describe how the movie did not live up to them. These factors combine to make movie review summarization a difficult task.

2.2 Reviews

Product reviews provide an excellent category on which to direct our system of summary sentence detection. Generally, reviews will describe the product under consideration, its positive and negative qualities, and then provide the overall sentiment of the reviewer in one or two lines. It would be useful, therefore, for a seeker of product information to be able to cut right to the chase and get this statement of sentiment for a large number of reviews. Such sentences are more useful to the consumer than simple positive or negative categorization, as they not only clearly provide a more nuanced measure of the reviewer's rating of the product, but also distill down the essential reasons that the reviewer felt the way he or she did about the product.

2.3 RottenTomatoes.com

RottenTomatoes.com is a movie review megasite. For each movie that has ever seen widespread release, RottenTomatoes maintains a page with a variety of information and links related to that movie. Central to these pages are the vast quantities of links to reviews by RottenTomato approved critics. For each of these links, RottenTomatoes displays a rating (fresh or rotten) and a quote drawn from the review that characterizes its general sentiment. These quotes are of the same essential nature as described above; they convey to the reader the intensity which which the reviewer classifies the movie, and give the reader a justifying reason for the expressed opinion.

This structure provides us with an exellent bank of tagged data with which we can test the systems that we implement. For every review linked to by the site, there is a single target summary sentence hand labeled by RottenTomatoes. The central focus of our project was therefore to build a system which, after being trained to recognize sentences tagged as summary or non-summary, would be able to properly classify sentences in reviews it had not seen before. We made the simplifying assumption that any given review contains exactly one "best"

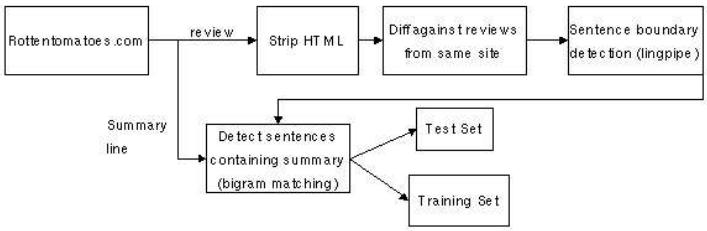

Figure 1: Data Preparation Stage

summary sentence as labeled by RottenTomatoes.com, as this reduced the subjectivity inherent in the summarization task. This did, naturally, likely result in apparently lower performance as a "pretty good" sentence that happened not to be the one that RT.com chose was labeled as an incorrect response. It was our hope that the classifier we built would fare decently well in the strict task of finding the tagged summary sentence, and in the process select sentences that overall were reasonable, even if those sentences were not tagged as the target in our corpus.

3 Algorithms and Implementation

3.1 Data Collection

To set about acheiving these goals, we first had to acquire the data to train and test on. We created a spider (`getRottenTomatoes.py`) to traverse the RottenTomatoes.com website, acquiring every review referenced by the site on its portal for each movie as well as the blurb RottenTomatoes.com selected to summarize the review. These HTML pages were of little use in their raw form, however. We needed to scrub the data down to the core features we would be using in our learning algorithm.

The process of taking a webpage and turning into data was four steps long. In the first step (`htmlstrip.cpp`), we eliminated any html or scripting code, leaving only plaintext.

In the second step (`dodiffs.pl`), we attempted to eliminate plaintext that occurs on all pages originating from the same site. To do this, we sorted the files into originating sites (`consolidateByURL.py`), then randomly `diff`ed the files against several others from the same site. `diff` outputs the portion of each file that is different from the other. We took the smallest set of diffs to be the normative version of the file.

3.1.1 Sentence Boundary Detection

In the third step (`runSentenceBoundary.sh`), we attempted to add information about the location of the sentences in the text. To do this we employed the program lingpipe. Lingpipe, among other things, is able to read through text and add xml tags surrounding each sentence indicating the boundaries. This performs well on normal text, although it has difficulties when it encounters swaths of garbage text that were not diffed out for one reason or another.

3.1.2 Detecting the Blurb

At the last stage of this processing (`detectquote.cpp`), we were ready to turn out usable data. The purpose of this step was to output the canonicalized form of the data: one sentence per line, with the target sentence (that is, the RottenTomatoes blurb) tagged. However, the detection of the target sentence could not occur by simple string matching with the blurb; oftentimes RottenTomatoes.com would modify a sentence from the review to use as its blurb, by taking a subsequence, leaving words out, or changing tense, among other things.

This left us with something of a disambiguation problem. To try to detect the target sentence, we iterated through the sentences of the text, assigning a score to each based on how well it matched the blurb. At first our metric was simple unigram matching; everytime we encountered a unigram that also appeared in the blurb, that sentence gained a point. Whichever sentence had the highest ratio of score to size then won.

However, this approach was biased toward choosing smaller sentences with many common words, and performance was inadequate. We attempted to augment this by ignoring words under a certain length, but this too proved unsatisfactory. We thus switched over to a bigram model. A sentence received a point every time one of its bigrams also appeared in the blurb. Again, we selected the sentence with the highest ratio of score to size. This performed extremely well, and allowed us to prune away faulty data (like unexpected reviews that say "Please login here" and the like) by rejecting data whose maximal sentence score ratio was under a certain threshold.

3.2 Training and Learning

After we had whittled the data down to its essentials, the next step was to extract useful features for the learning algorithm. Our primary features came from unigram and bigram dictionaries that we constructed on the training data. We used several techniques to narrow the dictionaries down to their most essential entries.

3.2.1 Linguistic Assumptions

Our choice of features implicity illustrates our linguistic assumptions. We assumed unigram, bigram, and capitalization counts to be sufficient to classify

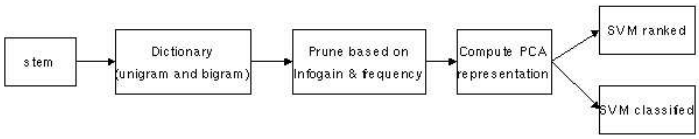

Figure 2: Learning Stage

sentences as summaries. We completely disregarded any other linguistic information, such as parts of speech and syntactic or semantic structure. These features could very well have improved the performance of the algorithm, and are an area for future research.

3.2.2 Stemming and Information Gain

Whenever we dealt with unigrams or bigrams, we first converted words to their stems using off-the-shelf stemming code by Porter [4]. This is a heuristic-based stemmer and cuts the number of detected distinct words considerably. We then implemented a method for selecting information-rich bigrams, described by Tan et al. [1]. This method attempts to use a few different heuristics to find information-rich bigrams for use as features.

3.2.3 Unigram Dictionary Generation

The first step in this process was to assemble the unigrams (`createdict.pl`). We began by reading in all the words. We then threw away words that did not appear at least 3 times in the corpus. This corrected for spurious character combinations that escaped detection in previous steps, and pruned away words that would not be of much use at all. This set became our unigram dictionary (62426 words). We then took this set and eliminated all words that did not appear in at least 1% of the sentences in either the positive or the negative class. This became the pruned dictionary (2960 words) we would use when detecting bigrams. We computed the information gain of all of these words, and saved it for later use in bigram generation.

3.2.4 Bigram Dictionary Generation

For bigram generation (`createbidict.pl`), we considered all the bigrams in which at least one of the words in the bigram appeared in the pruned dictionary. We only used bigrams that appeared in at least .1% of the positive and negative training sentences, and at least 3 times in the entire data set. Finally, we threw away the bigrams that did not have an information gain worse than the top 99% of the unigrams. We found we had to set the threshold this low because, since the size of the non-target category was so large, and the target category

profan	6.518	is rate	3.474
de	6.228	rate r	3.334
violenc	5.250	com	3.253
vulgar	4.619	nuditi	3.247
rate	4.243	que	3.053
com movi	4.207	brief	2.798
movi review	4.118	y	2.792
web	3.911	en	2.776
review _nmbr	3.785	gore	2.522
morn	3.784		

Table 1: top 20 word stems in one principal component and their relative weights

so small, the infogain for most words was very large. At the end of this process, we ended up with our bigram dictionary (1312 bigrams).

3.2.5 Principle Components Analysis

Another approach that we explored was to run principal components analysis on the training data. We encoded each sentence as a vector with one row per word in our dictionary, so entry i encodes the frequency that word$_i$ in our dictionary appears in each sentence. In addition, the vector for each article is the sum of the vectors for its component sentences. We took 1000 documents at random from our training set (about 30,000 sentences) and extracted the 100 principal components with the largest eigenvalues. The components that we found do seem to fit together reasonably well to encode 'types' of reviews. If we examine the twenty most highly weighted unigram/bigram stems in a vector, we get the words in Table 1.

Once the principal components of the movie review space were found, we could then reduce the dimensionality of our original review vectors by projecting them into the space of the 100 most significant principal components. In addition, since a document vector is just a sum of its component sentence vectors, any sentence vector could also be projected into the same review space. Since the principal components are derived from entire reviews and not individual sentences, it was unclear how the sentences would project into review space. We then combined this review space representation of each sentence with the representation for the review that it came from. This combined representation was then treated as a single labeled training example for the SVM. The results of this process were not great. The final precision of the SVM on a test set represented with the same principal components was about 16.5% (slightly lower than the word features alone).

3.2.6 Support Vector Machine

All of the preceding sections have discussed how the various feature representations that we tried were implemented. Each of them was eventually passed into a support vector machine for training and then evaluation. The SVM that we used was Thorsten Joachim's svm_light. This SVM package was selected because it provides the ability both to classify examples using traditional SVMs and to rank examples using pairwise rank constraints [2]. This is important in our case because it provided a way to require that at least one sentence per review be output. It also allowed us to return a sentence that might not have been considered "good" in a global sense, as long as it was considered better than the other sentences in its review. In contrast, an SVM that classified would only be able to classify sentences into the two bins of good summary/not good summary and may have decided that some review had no good summary sentence. In fact, with a training set of 1000 movie reviews consisting of 32,147 sentences, a traditional SVM earns a 96% training accuracy by classifying every sentence as a non-summary, yielding 0% precision.

Using the ranking feature in svm_light, we still labeled each summary sentence as a +1 and each non-summary as a -1. That is, each non-summary sentence was treated as equally bad and each summary-sentence was treated as equally good. The most important information was that any summary sentence was known to be better than any non-summary sentence from the same review. A rank constraint (a, b) indicates that $a > b$, so we needed one rank constraint for every summary sentence with each of the non-summary sentences in its review. If D documents had S sentences, and exactly one sentence in each document were a summary sentence, there would be $S - D$ rank constraints. In the case of the 1000 training movie reviews above, we would expect 31,147 rank constraints to be generated. Actually, one of the 1000 reviews had two sentences that together formed the summary, so 31,181 constraints were generated.

The output from the SVM on a test set no longer corresponded to a class label, but rather a ranking. The ranking numbers themselves were somewhat arbitrary in scale, but did imply an ordering of best to worst of the sentences that come from a single movie review. In order to obtain multiple measures for precision by varying our recall, we could change how many of the top rated sentences we output. To generate our precision/recall charts, we defined the ranking space of a particular review as the range between the lowest ranked sentence and the highest ranked sentence in the review. We used this to translate from a ranking back to a summary/non-summary class label that we originally wanted. Any point in the range of the ranking space could be used as a cut-off between which reviews we labeled as summaries and which were labeled as non-summaries. By varying the cut-off parameter we could generate many points along the precision-recall graph.

4 Testing and Results

4.1 Human Performance

The task of selecting the best summary sentence out of a review of twenty to one hundred sentences (average = 34) is inherently somewhat ill-posed. We refined the task by insisting that the "best" sentence was exactly the one that RottenTomatoes.com chose. However, there may be several very good summary sentences, and RT's choice may be somewhat arbitrary. We performed an experiment with human subjects (the authors) in an attempt to determine a baseline for how difficult it was to guess the same sentence that RottenTomatoes.com did. Each of us received ten untagged reviews, read them, and picked out a sentence that we felt was an appropriate expression of summary sentiment for each. The results were surprising. Out of 30 documents, we collectively labelled 12 right, giving us a success rate of 40%. In going over and examing our mistakes, we noticed that there were usually a few sentences in which the author made a blanket statement of sentiment, and any of these would probably make acceptable sentences. Our rough estimation for human performance of this task was therefore around 40% for locating the sentence that RottenTomatoes picked out, which exceeded our expectations.

4.2 Random Performance

A summary sentence detector that selects one sentence from the document at random as the summary sentance should expect to be correct once in every N documents, where N is the number of sentences in each document. If we assume and average document length of 34, this translates to an success rate of 3%. These two measures give us a wide range in which the performance of our system could be expected to fall.

4.3 Our System's Performance

After all of the work that we did to prune our dictionaries based on information gain and to do principal components analysis, our best precision came from using the ranking SVM with plain old unigram features. Our best performance was about 20% precision at guessing the same sentence that RottenTomatoes.com chose, which is about half way between random and human performance. Using the pruned dictionary resulted in about an 18% precision and using PCA with the pruned dictionary shaved off another percent.

One more encouraging result is that the "wrong" sentences that we chose subjectively didn't seem to be that bad as summaries. We chose two documents at random from our test set where we mislabeled the summary sentence. We present them below. The first sentence in the pair is the sentence that we liked the best in the review, and below it is the "correct sentence":

1. **We Liked**: "Above all else, he captured the children's inexhaustible fascination at being in front of the camera, and it's their grinning curiosity,

rather than outward signs of deprivation, that makes for the most lasting images."

Correct Answer: "It has the pointless verisimilitude of an expensively rendered computer simulation, and this makes it almost the opposite of what Scott wants it to be."

2. **We Liked**: "He makes such a dispassionate, even boring, lead that you may find yourself rooting for the film's villain to win."

   Correct Answer: "And in an apparent effort to make up for the lame script, Hyams overloads the film with action sequences that aren't particularly thrilling, horrific scenes that aren't very scary and dramatic moments (including one with Schwarzenegger crying!) that aren't even slightly moving."

In addition, we examined those occurrences when our ranking scheme was far off from correct. That is, when do we rate the correct sentence as one of the worst summary sentences in the entire review? Again, the results are quite surprising. In two randomly chosen examples, our subjective opinion is that at least one is a better summary than the one that RottenTomatoes chose:

1. **We Hated (Correct Answer)**: "What goes on in between involves TV-trained performers yelling at each other, second-rate computer-generated graphics failing to conjure up an enchanted world and Marlon Wayans doing what can only be called a heartfelt tribute to Butterfly McQueen, the screechy-voiced actress best known for her role as Prissy in "Gone With the Wind" ("I don't know nothin' 'bout birthin' no babies, Miss Scarlett!")."

   And Preferred Instead: "Don't let the star billing of Jeremy Irons and Thora Birch ("American Beauty") deceive you into thinking this is a classy production: Were it not for the brand-name recognition of the title, it's likely this slapped together mess would be debuting on cable rather than on the big screen."

2. **We Hated (Correct Answer)**: "Rupert Everett knows it: He mugs away desperately as the feckless Algernon Moncrieff and overplays Mutt to the Jeff of Colin Firth, who renders earnest Jack Worthing in his usual key of hunky worrywart as he struggles opposite Frances O'Connor's wan Gwendolen."

   And Preferred Instead: "Mercifully, the supporting cast saves the day by grasping clearly that in a comedy of manners you have to act mannered, though not to the point of situation comedy."

5 Things that Didn't Work

Our best performance came from using unigram counts, capitalized word counts, and document position as features. We put a lot of work into constructing a well-

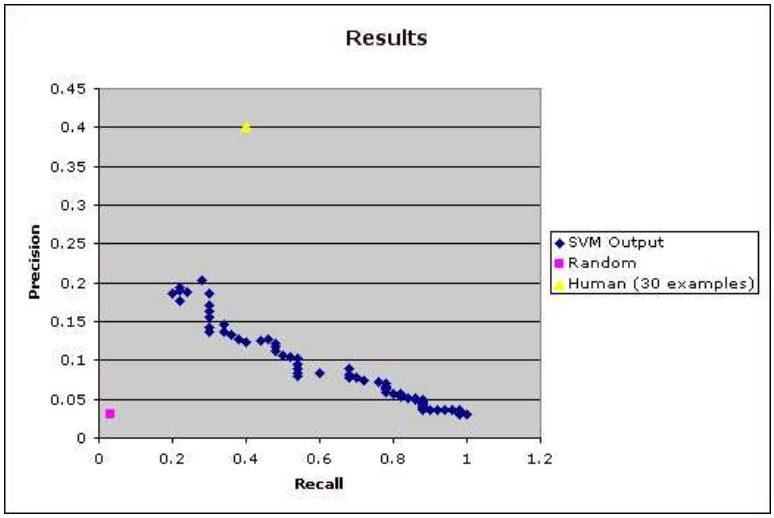

Figure 3: Precision/Recall for varying rank cutoffs

pruned bigram dictionary, but adding bigram counts reduced performance from 20% to 18%. The effect on training set performance was less severe: precision was reduced from 70.1% to 69.7%, and recall from 73.5% to 73.0%.

We tried adding a new set of features that would give information about a sentence relative to the review it was in. The features were, for each entry in the n-gram dictionary, the ratio of the number of times the n-gram appeared in the sentence to how many times it appeared in the document as a whole. We thought this might be informative by setting the sentence within the context of its document and discounting features that were common for all sentences in the document. With these new features, however, our performance actually decreased from 18% to about 16%, indicating that the information misled the SVM.

We ran principal components analysis to reduce the feature vectors from 63,740 dimensions down to 100. This reduced our precision from 20% down to 16%. It may be that reducing to 100 dimensions discarded too much information, and that better results could be obtained with more dimensions.

6 Further Study & Late Breaking News

As a first pass at this difficult problem, our system performs reasonably well at this difficult task. However, there are many different avenues of further study that could be pursued. One possible extension would be to attempt to classify the review as positive or negative first. Considerably more research has been done on this topic and results are typically above 80%. Using this positive or

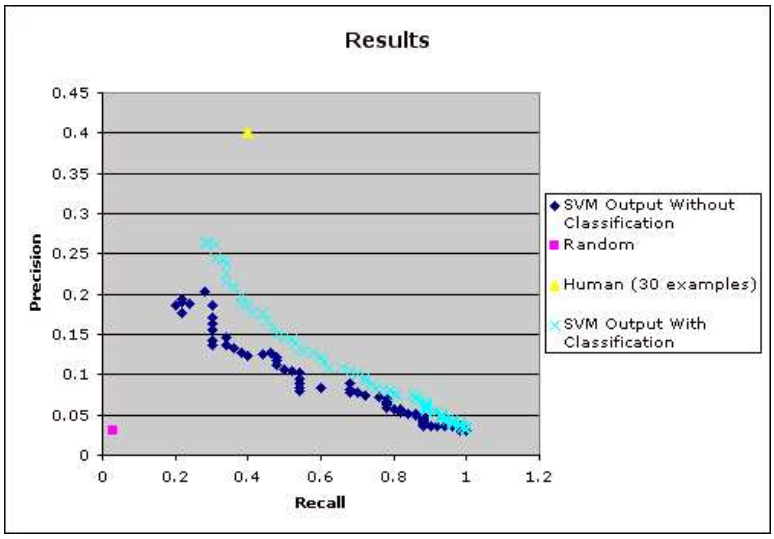

Figure 4: Results with positive/negative classification

negative class label as a starting point, two different ranking SVMs could be trained, one for each case.

With less than 24 hours left before our project was due, we decided to test the above strategy. We quickly coded a new SVM classifier to guess whether the review was classified on RottenTomatoes as either "fresh" or "rotten". Our classifier used unigram features indicating the presence of individual words. The classifier was able to correctly classify 80% of test cases. We then used the positive/negative opinion classifier to partition our training set. We took 5000 reviews in the training set and had the SVM label them as either positive or negative. 1530 were classified as negative, and 3470 were positive. We then trained two ranking SVMs, one for the positive examples and one for the negative. Test set documents were first classified as positive or negative and then sent to the correct ranking SVM for final summary/not summary labeling. The summary sentences for positive examples were predicted with 27% accuracy while the negative examples were predicted with only 24% accuracy. This resulted in a combined precision of 26.4% with 29% recall. At this time, we are still not sure why negative summaries are more difficult to predict. Additional study will be required to see if this method can be improved.

One could also think of other ways to use a pre-processing categorization phase to boost performance. For example, the movie could be categorized by genre. One could imagine that a summary sentence for a horror film would be a lot different than one for a romantic comedy.

In addition, thus far, we have not utilized any deeper linguistic information such as parts of speech or fuller parses. There may be ways to utilize syntactic

information to predict the level of importance of a sentence in a paragraph.

Finally, our system only attempts to find the complete sentence in the review that eventually became the RottenTomatoes summary. Often times the two are equivalent, but RottenTomatoes also frequently edits the sentences, most often to shorten them. If our system is to eventually become useful in a commercial context, it will also have to paraphrase longer sentences by removing superfluous words and phrases.

7 Team Member Responsibilities

Most of the work of the project was done with all three team members working side by side, and most of the programs were a joint effort. Dan wrote the bulk of detectquote, htmlstrip, and the unigram and bigram dictionary code. Jeff did all the PCA, SVM, and result processing code, as well as a lot of the downloading and text regularization code. Jamie wrote dodiffs and code for downloading data and feature extraction (extractFeatures).

References

[1] Yuan-Fang Wang Chade-Meng Tan and Chan-Do Lee. The use of bigrams to enhance text categorization. In *Information Processing & Management 38(4)*, pages 529–546, 2002.

[2] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining*, 2002.

[3] Julian Kupiec, Jan O. Pedersen, and Francine Chen. A trainable document summarizer. In *Research and Development in Information Retrieval*, pages 68–73, 1995.

[4] Martin F. Porter. An algorithm for suffix stripping. In *Program 14(3)*, pages 130–137, 1980.