Sentiment Classification of Movie Reviews

Daniel Pomerantz

December 16, 2007

Abstract

The goal of sentiment classification is to determine whether or not an author likes what he is writing about . In this paper, we explore various ways to improve upon current techniques involving subjectivity filters and information gain based feature selection. We conclude that the subjectivity filter is useful in certain cases, but needs to be explored further.

1 Introduction

With the massive amount of information and resources available on the internet, the idea of automatically filtering information has become very useful. A lot of work has gone into filtering according to topic. Search engines such as google do this rather effectively. A different, but related problem is that of *sentiment classification*, which is the problem of trying to classify a text document as "in favor of" (i.e. *PRO*) or "against" (i.e. *CON*). An example of a use for this is to be able to search not just for keywords or topics, but to be able to search within these topics for only documents that are PRO or CON.

Nowadays, if one wishes to find only articles in favor of a political candidate, for example, he would have to search for the name of the candidate and then add some keywords that the user thinks would be useful for pruning the search (e.g. "like" or "good"). It would be useful for the search engine to be able to do this automatically.

This paper will seek to compare various ways of classifying sentiment. The domain we will work in is movie reviews. This is a good domain for study as generally in a movie review we are given a "correct" answer (i.e. a score) since the movie review will have a number alongside it. This allows us to test our methods without considering the number, and then compare whether the result is correct or not. We can then generalize these approaches that work on movies to more sophisticated domains that are no labelled (e.g. political blogs, customer feedback, etc.)

One of the major issues with classifying movies (and text in general) is trying to determine whether a particular sentence is actually giving an opinion or a fact. For example, the following sentence gives no information as to whether the reviewer liked a movie or not: "In *The Empire Strikes Back*, James Earl Jones plays a very terrible villain and throughout the movie, he continues to do evil things." (Of course, some *Star Wars* fans will dispute the truth of that.) Despite the fact that the previous sentence contains words that would normally have negative association (terrible, villain, evil), it says absolutely nothing about whether or not the reviewer liked the movie or not. This paper will seek to investigate ways to preprocess the data using a *subjective filter* to remove the objective sentences such as the one above.

Finally, one of the problems with many of the techniques is they have a very large feature space and thus have to retain a large amount of information. This causes the calculations to be computationally more difficult as well as forcing the classifier to remember more information about the training data. For example, it is a waste to consider words that occur only once or twice in the training set, as they tell us little, if anything, about a negative or positive association. However, words that occur a lot in both positive and negative documents (such as "the") also shouldn't be counted. This paper investigates an approach to feature selection using *information gain*, which selects words that occur a lot in one type of document and not very often in the other type.

2 Previous Work

Various machine learning techniques have been used for classifying text. Most of these involve some variation of a "bag of words" approach, which will be described in the next section. However, there are some other interesting approaches to the problem using techniques different than those described below. Many of these techniques involve clustering or expectation maximization. For example in [6], Turney classifies reviews by creating an association measure between all adjectives in the training set. He seeds the good and bad clusters with a couple initial words that he assumes has certain connotations. Then each document is classified by whether it has more words in the positive cluster or negative cluster. These approaches tend to try to cluster adjectives together based on considering when these pairs of adjectives occur together. When they occur together and are separated by the word *and* or *so*, for example, they have a positive correlation, but when they are separated by a word such as *but* or *still*, they have a negative correlation.

Another interesting approach involves learning from unlabeled data. Nigam, [2] describes a process for generating more training data using expectation maximization. The idea is that we can easily cluster unlabeled documents according to their word distributions. Once these clusters are made, we can use the few labels that we have to determine which cluster is positive and which is negative.

There also has been a lot of research done on the linguistic side of things. The idea here is to be able to label individual words based on certain features such as their parts of speech. Then rather than considering the data as a "bag of unrelated words," one could consider the sentence structure. One simple idea is to only consider adjectives. A more complicate approach could involve analyzing the sentence structure to account for relations between words. An obvious example of this is when a negation word is placed in front of an adjective, it's meaning is flipped.

These ideas are all very interesting and have proven useful in practice, but they are beyond the scope of this paper. Our approach mainly focuses on a unigram "bag of words" approach to sentiment classification. It builds on the work by Pang et. al in [4], [5]. Additionally, it considers alternative ways to reduce the dimensionality of the space, focusing on a method involving information gain.

3 Background Information

This paper explores three ideas: sentiment classification, subjectivity filtering, and feature selection.

3.1 Feature Representation and Selection

We need to consider how to represent all of our data. For this paper, we only considered *unigrams*. Although some previous work [4] has demonstrated that including information about relative document position or even the occurrence of certain punctuation marks is useful for determining sentiment (e.g. Question marks might often occur in sentences such as "What was he thinking when he made this movie?" but exclamation marks may be in sentences such as "I loved this movie!"), we did not consider these as our features. Another issue to consider is word stemming. However, this was not done for this paper because word stemming assumes that the various forms of the root word all have the same class distribution (e.g. look, looks, looking, looked, etc all occur with the same proportion in positive and negative documents). This assumption seems reasonable for text classification, where the goal is to determine the topic. However, we did not want to make this assumption at this point for sentiment classification.

We experimented with two types of representations of features or words. One is based on the frequency of the word occurring, and the other is based on the presence of the word in the document. For SVM, the representation is effectively the same, except that for presence, our input vector is in the space $\{0,1\}^N$, but in frequency it is in the space $\mathbb{N}^n$.

For Naive Bayes, we treat our representations a bit differently. In the case of presence, we represent our document as a binary string, where each digit represents whether that particular word occurred or not in the document. (e.g. 10001 would mean the first and fifth word occur in the document, but the second, third, and fourth words don't occur. In the case of frequency, we represent our document as an n-ary string, where n is the number of words we are considering. In this case, each digit is a number between 1 and n representing what word occurred in that position. (e.g. the string 848 would mean the the first word in the document is the 8th word in our vocabulary, the second word in the document is the 4th word in our vocabulary, and the third word in the document is the 8th word in our vocabulary). This representation allows us to calculate the probabilities more easily than if we thought of the document as a binary string of features where features were of the form "word w occurs n times" where we would then have to store n variables for each word.

The next question to consider is how we select which words we will count. We could, naively, consider every single word as a feature and then our feature space would have a dimension equal to the total number of unique words in our training set. However, this has two potential problems: 1)It is potentially susceptible to noise and missing data. If we include words that only occurred a few times, it is more likely that our estimation as to the proportion of times that word occurs in positive documents is inaccurate than if the word occurred more frequently. 2)Computationally, it would be faster and take up less storage to reduce the dimensionality of the data. This would be done by removing words that occur frequently, but in both type of documents (such as the word "the"). Knowing that this type of word occurred in a document does very little to help us classify a document correctly. Thus we would like to not even consider the word at all.

In order to mathematically determine what words are important, there are several accepted approaches. These approaches include simply thresholding a minimum number of occurrences per word, a χ^2 significance test, and approaches based on mutual information. These are investigated further by Yang and Petersen in [7]. We use an approach based on information gain, which according to their study performs at the same level as these other tests.

Information gain relates to the change in our certainty over a document's class after learning whether or not an event (in our case a word) occurs. We wish to select the words that have the most information gain. In essence, we wish to determine what words are worth "asking" whether or not they occurred. To do this, for every word, we consider the uncertainty before asking the question and after asking the question. The words with the biggest difference are the words we wish to choose. The uncertainty before asking the question is irrelevant since when comparing the information gain, it is constant for every different word. Note that our approach considers each word separately, rather than first selecting the word with the most information gain and then selecting the word with the most information gain given that we already knew the answer to the first word. (See [3] for more detail about a sequential selection approach.)

To determine the uncertainty after asking, we use the following formula: Let $P(w)$ be the probability of word w occurring. Let $P(c)$ represent the probability of the class being class c and let $P(w, c)$ be the probability of word w occurring and the class being class c. Finally let $P(\bar{w})$ be the probability of word w *not* occurring and $P(\bar{w}, c)$ be the probability of word w not occurring and the class being class c.

$$IG(w) = \sum_{classC} P(w,c) * log(\frac{P(w,c)}{P(c)P(w)}$$

$$+ \sum_{classC} P(\bar{w},c) * log(\frac{P(\bar{w},c)}{P(c)P(\bar{w})}$$

The first term of the above comes from the expected change in entropy in the case that the word occurs and the second term comes from the expected change in entropy in the case that the word does not occur. We experimented with different threshold levels to consider which words to include in our feature space.

3.2 Sentiment Classification

We considered two classes of machine learning algorithms: Naive Bayes and Support Vector Machines. For each, we considered both keeping track of the frequency of unigrams and simply the occurrence of these unigrams.

3.2.1 Naive Bayes

In the Naive Bayes representation, we use Bayes rule:

$$P(Class = |Data) = \frac{P(Data|Class)P(Class)}{P(Data)}$$

We determine which class has the maximum probability and assign the document to that class. Note that the term in the denominator is simply a constant factor that is present in all classes, so in determining the maximum class, we do not need to consider it. For the probability of a particular class, we simply use the maximum likelihood estimate, which is the proportion of training documents in that particular class with respect to the total number of documents.

For the probability of the data given the class, we make the Naive Bayes assumption that all features of the document are independent once we know the class. Of course, in reality this is not the case as a word always depends in some way on the previous words. However, previous research [4] has shown that results are good despite violating this condition. Thus we can simply consider the probability of each feature occurring independently. This is also done with the maximum likelihood estimate, which is the proportion of times that that feature occurred in our training data with respect to the number of times that class occurred.

In the case of frequency, we think of each word as a random word chosen from our vocabulary (recall the string representation described in the feature representation section above). To determine the probability of this word w occurring given the class, we use the maximum likelihood estimate:

$$P(w\,occurs|class = C) = \frac{|w_c|}{|C|}$$

where $|w_c|$ is the number of times w occurs in class c in training and $|C|$ is the total number of words in class C. Note that by "total number of words," we count duplicate words multiple times. It is important to note that we don't need to explicitly include probabilities of words *not* occurring because this is implicitly included in the denominator of the above terms. Finally, note that we use Laplace smoothing to deal with missing data. In this case it consists of adding 1 to the numerator and adding the total number of *unique* words to the denominator. This is essentially assuming that we are given a document that contains exactly one of every single word in the vocabulary.

So our formula for classification is:

$$\arg\max_c \left(P(c) \prod_{w_i \in document} \left(\frac{(w_i + 1)}{|c| + |V|_c} \right)^{n_i} \right)$$

where $|V|_c$ is the number of unique words in class c and n_i is the number of times that word w_i occurs in the document.

In the case of presence, recall that we represent each document as a binary string, where each digit represents whether or not that word occurs. In this case, we must explicitly consider words that do not occur. If we fail to do this, and one class has documents of longer lengths, then certain unintended biases can occur. For example, suppose that positive documents are normally a lot longer than negative documents. Then in general, a word is more likely to occur in a positive document than a negative document simply because given one word, we would guess that the document is positive because that word occurs in more positive documents than negative documents. However, this only happened because the positive documents have more total words.

The problem with the above is that we are failing to consider what words did not occur. For the first representation we did not have to because we determined the probability of each word occurring in a particular "slot." So we only need to consider the probability of a word occurring in a certain slot, where we assume that the probability of occurring in a slot is independent of which slot we look at. If one class has longer documents, then the total number of words will be higher, and so this will cancel. In the second case, however, we are keeping track of which words occurred and which word didn't occur. We thus have to include the probability of a word not occurring in the entire document. This distinction is further elaborated on in [1].

For each probability, we once again use the maximum likelihood representation with Laplace smoothing. This means that we assume the probability of a word occurring in a class is equal to the number of documents in that particular class which have that word divided by the total number of documents in the class. Thus our formula for classification is now:

$$\arg\max_c \left(P(c) \prod_{w_i \in doc} \left(\frac{|w|_i + 1}{|C| + 2} \right) \prod_{w_i \notin doc} \left(\frac{|\bar{w}_i| + 1}{|C| + 2} \right) \right)$$

where $|w|_i$ and $|\bar{w}|_i$ represent the number of times w_i does and doesn't occur in the class respectively. In this case $|C|$ refers to the number of documents in the class.

One final thing to consider is that we could represent frequency using a similar type string as we use in presence. However, to do this, we would have to be able to calculate the probability of a word occurring exactly n times, for any natural number n. This would require storing, for every word w and every natural number n, the number of documents where word w occurs exactly n times. This would be difficult both computationally and practically (in the sense that it is much more difficult to implement). The frequency representation described above is much better as all we need to remember is the number of times each word occurs in a document and treat all occurrences independently. It assumes that there is no relation between a word occurring once and occurring again.

3.2.2 Support Vector Machines

The goal of SVM-classification in the training phase to determine an optimal decision boundary. By optimal, we mean the decision boundary should be as far from our labeled data as possible. SVM tries to maximize the minimal distance from the decision boundary to the labeled data. Once this decision boundary is decided, we can check for a given query document on which side of the decision boundary it lies.

In order to train the SVM classifier, we must represent each document as a vector. This is done in a very natural way by letting each coordinate of our vector represent one word. In the case of presence SVM, we have the coordinate be 1 if the word occurs and 0 otherwise. In the case of frequency, we have each coordinate storing the number of times that particular word occurs in a document. Note that the difference between the two representations is much smaller than in the case of Naive Bayes classification. For the query phase, we again convert our document to a vector and then check which side of the decision boundary it lies on.

3.3 Subjectivity Filtering

3.3.1 Basic Version

The final issue that we consider is the issue of subjectivity filtering. The idea is to preprocess both our training data and our test data with a subjectivity filter. See figure 1. This would seek to remove sentences that were deemed objective since they should not have any relation to whether the writer liked a movie or not.

There are two methods of subjectivity filtering that we considered. The first method is for every sentence in a document to classify it as "subjective" or "objective" based on a classification technique that is exactly the same as above except with the goal to classify a sentence instead of classifying a document. We would train our classifier by giving it sentences that are labeled as "subjective" or "objective." When given a sentence to classify, our classifier could then output whether or not it believes it is subjective. This could be done with SVM, Naive Bayes, or any other classifier one chooses. In our case, we implemented this with Naive Bayes, but it could be done with any technique.

To get the classification of the sentence, we must use Bayes rule in the same fashion as we do for sentiment classification. The difference is that we are given a training set which consists of *sentences* that are classified as *objective* or *subjective* instead of *documents* which are classified as *good* or *bad.* We compute the MLE for each of the features in the same way as previously.

Using this process, the goal would be to run every training and every test document through a subjectivity filter. We would then keep the sentences that are classified as subjective and discard the objective sentences [1]. Based on these individual classifications, we have a new set of data. We then can run our algorithms from the previous section to classify a document as positive or negative.

[1]Theoretically, it might be useful to use *only* the objective sentences in a topic classification system, but we suspect this is being too idealistic.

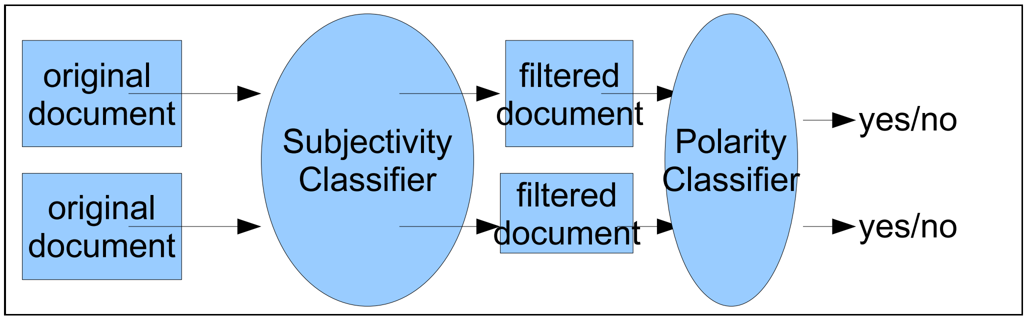


Figure 1: Subjectivity Filtering: First we run every document through a filter that removes all of the objective sentences. The aim is to be left with a set of documents that contains only subjective sentences. These documents are then run through a standard polarity classifier.

3.3.2 Minimum Cut Version

One of the flaws with the basic version of subjectivity filtering is that it does not take into account any relation between sentences. It classifies every sentence of a document independently. This assumption very likely does not hold in reality. For example, a reviewer may write a paragraph of plot summary and then a paragraph of critique. In this case we would like to *associate* sentences with each other based on proximity. Note that this method could be extended to base the association on any other information, but sentence proximity seems the most natural.

The idea now is to partition all of the sentences of a document into two set: one for subjective sentences and one for objective sentences. The idea is to partition the sentences such that we minimize both the probability of being in the other class (based on our "simple" classifier) and the sum of the associations with other classes. Mathematically, this is:

$$\min \sum_{s \in subj} P_o(s) + \sum_{s \in obj} P_s(s) + \sum_{s_1 \in obj} \sum_{s_2 \in subj} assoc(s_1, s_2)$$

where $P_o(s)$ and $P_s(s)$ are the probabilities of sentence s being objective and subjective respectively, based on our original classifier. This seems computationally intractable since there are 2^n possible partitions for a document with n sentences. However, Pang et. al. [5] have shown that this problem can be reduced to a Min-Cut problem, which has a polynomial time solution.

The reduction is the following. For every sentence create a node. Create a source node s which is connected to every sentence. The weight on each one of these edges is given by the probability of the sentence that that node represents being subjective. This is based on our basic classifier. We then create a sink node t which is also connected to every sentence. The weight of an edge to the sink is given by the probability that the sentence is objective (1 minus the probability of it being subjective). Then we connect each sentence to each other via a function (see below) to represent the association. Finally, we find the min-cut/ max s-t flow of the graph and whatever nodes are left on the same side as s are labeled subjective and the nodes on the side of t are labeled objective. We then classify our documents as positive or negative based on this new data set generated.

Note that in order to do this, we now need to compute the probability of being subjective and the probability of being objective rather than just computing which is bigger. We can use the fact that we know the sum of probabilities of the two classes must be one to do this.

In our case, we used a function for association that incorporated the proximity of 2 sentences within a document. In order to make the graph less dense (and increase the efficiency), we assume the association is 0 if the proximity is greater than D where D is a parameter that would normally be determined via cross-validation. The function we used was

$$assoc(x, y) = c\,\frac{1}{(x-y)^2}$$

where c is also a parameter determined by experimentation and x and y represent the count of what number sentence each is. Due to time constraints, we only tested D = 3 and C = .5 and it may be that we could have improved the success of the classifier had we tried different values. Additionally, there are other possible functions, such as an exponential function or simply a constant function (i.e. association = c if the distance is less than D and 0 otherwise)

4 Algorithm

The final algorithm that we employed is the following:

- For every document in our training set:
  - For each sentence generate a Naive Bayes estimate as to whether it is subjective or objective
  - For each pair of sentences compute the association
  - Classify each sentence as subjective or objective based on min-cut algorithm

Table 1: A Comparison of the Algorithms: The p-values are the differences on a t-test between the the min-cut filter and the regular algorithm.

Algorithm	**Frequency?**	**Regular**	**Subjective Filter**	**Min-Cut Filter**	p-value	95% signficant?
Naive Bayes	no	82.15 %	85.65 %	86.5 %	2.68	yes
Naive Bayes	yes	79.90 %	84.90 %	85.05 %	2.751	yes
SVM	no	87.3 %	85.85 %	85.80 %	-.986	no
SVM	yes	82.05 %	83.75 %	85.90 %	2.35	yes

  - Only keep the sentences that are subjective.

- Given a query document, run the same min-cut algorithm on it

- Compute the classification of the document as positive or negative based on Naive Bayes or SVM (or any other technique you prefer)

5 Our implementation

For the sentiment polarity data set, we used the movie review set compiled by Pang et. al [2] . They took 2000 movie reviews from IMDB, 1000 of which were positive (the reviewer gave a score above a threshold), and 1000 of which were negative. The data set only consists of reviews which are sufficiently good or bad and not anything in the middle. This is good from the perspective of training as we can learn what words are in good reviews and bad review separately (i.e. there are no "so-so" reviews). For the purpose of testing, it is still sufficient to use this data set as any review that is in the middle is difficult for even a human to classify. If a review is in the middle, then either classification would be right or wrong depending on who you asked.

Getting subjectivity data is much more difficult. It is a somewhat difficult task for a human to determine whether or not a sentence is subjective. For this reason, subjectivity data set is not as well labeled.Pang et. al. also compiled a data set of subjective sentences[3]. They did this by taking 5000 sentences from "plot summaries" and labeling them as objective and 5000 sentences from an opinion blog and labeled them as subjective. This is not a perfect training set as of course a plot summary may still contain opinions and a blog may contain facts. However, this is the best data set available.

In order to test with SVM, we used the package SVM Light [4]. This package allowed us to train and query with SVM very easily. Note that we used all of the default settings for classification, including using the linear kernel.

We ran 100-fold cross-validation which seems like a large fold, but was necessary in order to include all data in the test case for every algorithm we ran. This way, we only needed to train 20 times for each algorithm, as opposed to running the training, for example, 200 times if we did 10-fold cross-validation. If anything, using this large a fold made our task harder as we removed 5 percent of our data in each fold. Note that when we considered the vocabulary of each, we had to retrain assuming we had only seen the training data since in a real life scenario this would be the case. This means that we have to do the computation of what our vocabulary is on every different fold, which takes a fair amount of time.

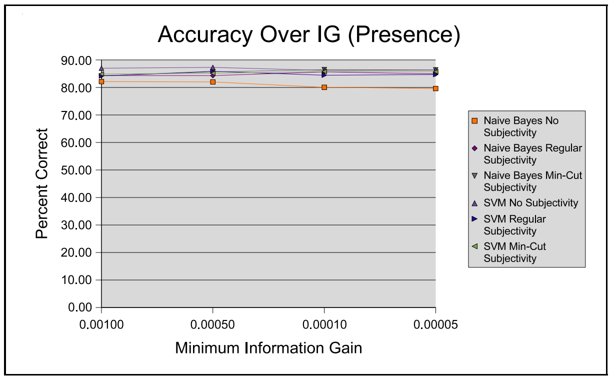
6 Data and Results

We experimented with both SVM and Naive Bayes. For each of these algorithms, we considered keeping track of both frequency and the presence of unigrams and considered a feature space based on four different levels of minimum information gain. Each of these were run on three separate data sets: one on the original movie reviews from IMDB, a second with subjective sentences filtered using just the Naive Bayes filter, and a third with subjective sentences filtered using the minimum cut algorithm proposed by Bo Pang et. al.

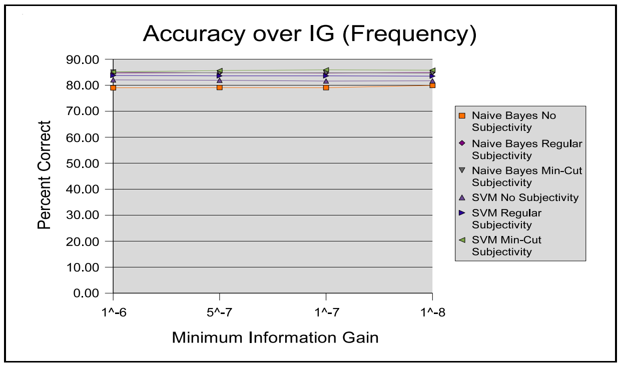
Our results are presented in table 1 . Each algorithm was run with four possible values for information gain. The percentages listed are the highest percent of correct classifications during 100-fold cross-validation. The results indicate that keeping track of simply the presence of a word is more useful than keeping track of the frequency as in almost every case (the only exception being SVM with the Min-Cut filter where the two versions are essentially equal), the presence version out performed the frequency version. Additionally, using the subjectivity filter improved the accuracy significantly in three of the four cases. Using the Minimum Cut filter improved things even further. In the case of SVM presence, however, it actually degrades performance, but this does not pass a 95 % significance t-test.

In figure 2, are two graphs demonstrating the various changes of each algorithm with different threshold values of information gain. Recall that in each case, we con-

[2]http://www.cs.cornell.edu/people/pabo/movie-review-data/review_polarity.tar.gz
[3]http://www.cs.cornell.edu/people/pabo/movie-review-data/rotten_imdb.tar.gz
[4]http://svmlight.joachims.org/



(a) Presence



(b) Frequency

Figure 2: The accuracy of the algorithms compared with the minimum information gain used. The smaller the minimum information gain, the more words we select. One can see that the lines are very horizontal, indicating that accuracy is not lost when we remove words.

structed our feature space by requiring that each word satisfied a minimum information gain. Note that the scales for the frequency and presence is different since there are significantly more pieces of data in the case of frequency. Thus we have put them on separate graphs (since the x-axis refers to information gain as opposed to number of words used). As you can see, including fewer words has no negative effect on the success of the classifier. In fact, if anything it leads to better results, most likely due to the fact that the data is less susceptible to noise. While we do not present the results here, we informally experimented with even smaller information gain, and not surprisingly, eventually discarding data causes problems.

One further thing to note is that while timing was not measured, the training time on filtered data is significantly less than the training time on the unfiltered data for the obvious reason that the amount of data to look through is much less. A total word count reveals that the vanilla subjectivity filter keeps 45.0 % of the positive data and 50.2 % of the negative data (which was shorter to begin with) whereas the min-cut filter keeps 50.5 % of the positive data and 55 % of the negative data. However, despite throwing away about half of our data, our results improved, indicating our hypothesis that this data is not useful and can in fact be misleading is correct. This suggests that even though the SVM implementation did not improve, subjectivity filtering could still be useful as a way to decrease the size of our data. The speed of the query is approximately the same, as once the SVM is trained, classifying the data does not require reexamining the original data–it only requires looking at the decision boundary.

When we decrease the number of features, we improved both the query time and the training time as there were fewer elements to consider. Recall that in the Naive Bayes implementation we had to keep track of how often every word occurred in each class. This means that in order to look up how often a word occurred, we had to look through a list of data that includes every word, so if there are fewer words, we can search more quickly. While the focus of the paper was not on timing optimization issues, it is, needless to say, still a good thing.

7 Conclusions and Future Work

Our results for are very similar to Pang et. al [4], [5], who concluded that generally SVM performs better than Naive Bayes. However, they had better results when using SVM to test using the subjectivity filter. While our drop off does not pass a significance test, and is not nearly as large a difference as in the cases where there is improvement, we still need to consider the possibility that it wasn't due to random chance, especially since they concluded that there was significant *improvement* between the regular algorithm and the Min-Cut formulation.

One potential cause of the difference between the two results is the large size of each fold[5] for cross validation. We have 2000 documents, and each fold represents 5 % of the data. It is possible that this didn't make a major difference in the case of applying no subjectivity filter, as there was still a lot of data to test on. However, it may just be that after applying the subjectivity filter, which removes approximately half of the sentences, and then removing 5 % of the training data, that we just don't have enough information for the classifier to work well. This suggests that we need to consider the possibility of using more training documents in the case of applying a subjectivity filter compared with when we don't. While this would mean we'd need to store more data, it would be negated by the fact that we are storing much less

[5]Pang and Lee used 10-fold cross validation

data per review. The new data set would perhaps be of a comparable size, but it would be more useful data.

Another issue that needs to be explored is changing the association function that is used to compute the weights of the subjectivity graph. The association function we used only relied on the proximity of sentences and did not consider other issues, such as paragraph boundaries. Additionally, we can change the actual distance function that we used and experiment with different constants. With a constant value of .5 in our case, we are creating very large associations between the different sentences. Additionally, it may just be that our original SVM presence algorithm got lucky and had higher results than it normally would as it did much better than in previous studies. (Pang et. al has 82.9 % accuracy.)

The subjectivity filter led to an increase in accuracy, but it is clear that it can be tricked sometimes by keywords, and this problem may be inherent to filtering based on subjectivity. For example, the following sentence was classified as subjective:

> I don't think anyone needs to be briefed on Jack the Ripper

The prior sentence *is* actually a subjective sentence, since the writer is giving his opinion on what he thinks the reader knows. However, it has no relevance to the context of sentiment classification as the statement, while being an opinion, is not an opinion about the movie. The fact that it *is* actually an opinion shows us how difficult the problem of subjectivity filtering is. We have to filter not only all objective sentences but also all subjective sentences that are unrelated, and this seems very difficult. One could design a classifier to classify sentences as "subjective movie sentences" vs. "not subjective movie sentence" (instead of "subjective sentence" vs. "objective sentence"), but it would be very difficult to generate data for this. As it was, the training data was produced in a less than perfect manner.

Fortunately, the subjectivity filter does not have to be perfect in order to work. If it removes a large enough percentage of the sentences that don't involve opinions about the movie, as demonstrated, it can still aid in both timing performance and accuracy. The fact that we are able to remove such a large amount of the data and *improve* accuracy is quite impressive. In [5], they performed a "flip" experiment, where they kept only the *objective* sentences, and this resulted in worse accuracy than the baseline Naive Bayes implementation. We did not run this test but expect we would have very similar results.

In addition to modifying the association function of the subjectivity filter, we could try to improve the subjectivity classifier by analyzing sentence structure. It may be that certain sentence structures tend to be opinions where as others tend to be facts. For example, opinion sentences may often involve more "that clauses" since they often start "I think that..." In our subjectivity classifier, the only features that we currently consider are words. We could add to our feature space features such as sentence structure, sentence length, etc.

There is also some room for improvement with the sentiment classifier. One obvious problem is when the sentence includes words such as "not," the entire meaning of the sentence is changed. However, these issues can be much more subtle. Many of the documents that were misclassified were caused by an intentional deception on the part of the author or a gradual build up which sets up a big *but.* For example, reviewers will sometimes give a list of things that they disliked about a movie and then say they liked it despite all these rational reasons to hate it. This is a bit troubling as it is very easy for a human to interpret this, but our algorithm does not perform well in this setup. This is indeed one of the issues that makes sentiment classification a more difficult task than topic classification.

Another idea would be to first determine the *topic* of the document before running the sentiment analysis, as our algorithm assumes that all topics have the same distribution of words as they related to sentiment. However, this assumption may be inaccurate. For example, a horror movie is more likely to include words that *seem* negative, but are actually good in the context of being a horror film. For example, "This film is scary" has a completely different meaning in a horror movie review than it does in a romantic comedy review. In the former case, it is probably a compliment as the movie was supposed to be scary, but in the latter case, it most likely is a commentary that it is scary how bad the movie is.

One major problem with this is generating training data, as we would need to have a large training set for each of several movie topics. McCallum and Nigam [1] have suggested an approach to text classification using EM that would be very interesting to experiment with.

An issue that appears clear is that reducing the number of words that we kept track of does not hurt our results and if anything, there is an improvement when we reduce the number of words, since as hypothesized, the amount of noise is reduced due to dropping words that don't occur very often.

In summary, there is room for exploration. The information gain approach definitely works well with respect to cutting down on computations and the subjectivity filter has shown it has the potential to be very useful. Areas for future work include determining a better association function, incorporating topic classification into the filter, and learning with smaller amounts of training data.

References

[1] Andrew McCallum and Kamal Nigam. *A Comparison of Event Models for Naive Bayes Text Classification.* 1998

[2] Kamal Nigam, Andrew K. McCallum, Sebastian Thrun, and Tom M. Mitchell. *Text Classification from Labeled and Unlabeled Documents using EM.* 2000

[3] Jana Novovicova, Antonin Malik, and Pavel Pudil. *Feature Selection using Improved Mutual Information for Text Classification.* 2004

[4] Bo Pang, Lillian Lee, and Shivakumar Vaithynathan. *Thumbs up? Sentiment Classification using Machine Learning Techniques.* 2002. Proceedings of EMNLP, pp. 79-86

[5] Bo Pang and Lillian Lee. *A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts* 2004.

[6] Peter Turney. *Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews* 2002.

[7] Yiming Yang and Jan O. Pederson. *A Comparative Study on Feature Selection in Text Categorization.* 1997