Sentiment Analysis on Limited Twitter Data

Essay prepared under the direction of

Kata Gabor, PostDoc, LIPN Université Paris 13

Submitted by

Alexander Fabbri

in partial fulfillment of

the requirements toward the Directed Research Option

Columbia University Undergraduate Programs in Paris

May 18, 2016

Contents

Introduction

The past decade has seen a substantial growth in the use of microblogging services such as Twitter and similar platforms that allow a user to publish short-length messages. With this development has come the desire to automatically extract sentiment from these messages. In Chapter 7 of the book *Speech and Language Processing*, Daniel Jurafsky and James Martin define sentiment as “the positive or negative orientation that a writer expresses toward some object." The implications of being able to automatically classify a message's sentiment can be felt in many sectors; consumer opinions alter product marketing while public sentiment plays a role in daily politics and stock markets. SemEval is an ongoing series of semantic evaluation exercises that addresses such current developments in semantic analysis. Specifically, SemEval-2016 Task 4 addresses the problem of sentiment analysis in Twitter. Given a tweet, the goal is to classify the message according to a category such a positive, negative or neutral. The task includes various subtasks that differ in their ordinal classification of sentiment. Provided in the scope of the competition is a corpus of tweets human-labeled as positive, negative or neutral. These short informal messages such as tweets, however, pose new challenging problems to the task of text classification and sentiment analysis. They are short-length and thus a single word, negation or punctuation often holds the key to the message's sentiment. Additionally, these messages contain many intentional misspellings, abbreviations and special symbols such as hashtags. With the SemEval exercise as a base, we will first examine state of the art techniques and features for sentiment classification. We will concern ourselves with building a classifier to distinguish between the two categories of positive and negative tweets. To do so we will implement a naive Bayes algorithm that makes use of pertinent features to learn from and classify tweets. Having achieved a running implementation, we will then look to examine the errors of our classification system and find relevant features and possibilities for improvement.

We will finally turn towards recent developments in using word vectors and neural networks to classify text in hopes of achieving higher classification rates and new insights into classification.

SemEval 2016 Task 4 [1] provides an annotated corpus of Twitter messages and proposes five subtasks varying in their ordinal classification and classification goal. Some tasks concentrate on a binary classification into either a positive or negative class, while others propose a five-point scale for sentiment analysis. Additionally, four of the five tasks relate to tweets known to be about a given topic. For our research we concentrate on binary classification into positive and negative tweets. Provided is a script to download the tweets used. [2] After removing any duplicates or tweets labeled as neutral, we were left with 3,892 tweets. Understanding the task at hand and having data for future tests, we begin a formal analysis of text classification and sentiment analysis in relation to a chosen algorithm.

Naive Bayes Algorithm

Jurafsky and Martin analyze a general algorithm for classification demonstrated on the class of text classification problems. This task aims to classify an entire text according to a label drawn from a given set of labels. To deal with this task Jurafsky and Martin introduce the multinomial naive Bayes classifier. Naive Bayes is a probabilistic classifier that returns the class or label, c, which has the maximum posterior probability given the document, d. For the purposes of this project there will only be two classes, positive and negative, over which the maximum probability is taken. The most probable class is found as follows:

$$\hat{c} = \arg\max_{c} P(c|d)$$

making use of Bayes' Rule:

[1] `http://alt.qcri.org/semeval2016/task4/`

[2] `https://github.com/aritter/twitter_download`

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

becomes

$$\hat{c} = \arg\max_{c} \frac{P(d|c)P(c)}{P(d)}$$

Utilmately, the dominator may be dropped since this term will be identical for each class.

This type of classification makes use of the bag-of-words model which portrays a document as an unordered set of words with their positions ignored. Thus, a given word has the same effect on classification regardless of where it occurs in the document. All that matters is the frequency of the word in the document. The final representation of the document is as a set of features f_1,f_2, ... , f_n. These features are the individual words from the bag-of-words models but also may include other characteristics of the document such as the presence of all caps words. Dropping the denominator from the previous equation and taking into account this new representation, the equation for Naive Bayes becomes:

$$\hat{c} = \arg\max_{c} P(f_1, f_2, \ldots\ f_n|c)P(c)$$

Additionally, the Naive Bayes assumption states that the conditional probabilities above are independent. Thus the conditional probability of the features can be calculated as the product of the individual feature conditional probabilities. As a result, the class with the maximum posterior probability is:

$$\hat{c} = \arg\max_{c} P(f_1|c) \cdot P(f_2|c) \cdot \ldots\ \cdot P(f_n|c) \cdot P(c)$$

This formula can be seen as the likelihood of the features given the class multiplied by the prior probability that a document belongs to that class.

Given this formula, one must find a way to calculate the individual probabilities. One can use a maximum likelihood estimate to estimate the probabilities P(c) and P(f|c). The prior probability, P(c), equals the number of documents in the training data which are labeled class c divided by the the total number of documents. The probability of a feature such as the presence of a word given a certain class, P(w_i | c), equals the number of occurrences of the word in documents of class c divided by the number of occurrences of all words in documents of class c.

$$\hat{P}(w_i|c) = \frac{count(w_i, c)}{\sum_{w \in Vocabulary} count(w, c)}$$

However, a problem arises for words that are not seen when training the model but that arise in testing. If we are trying to estiamte the likelihood for a word but have not seen that word in the training, the count is zero and we will be multiplying by a zero conditional probability when calculating the max class probability. One solution is to use Laplace smoothing to avoid zero probabilities, although we do not truly get a good estimation of the actually conditional probabilites. One is added to the numerator while for the denominator one is added for each word in the vocabulary, so that the probability becomes:

$$\hat{P}(w_i|c) = \frac{count(w_i, c) + 1}{\sum_{w \in Vocabulary} (count(w, c) + 1)}$$

Jurafsky and Martin note various ways to optimize naive Bayes. Expanding on this analysis of word frequencies, they present an optimized version of naive Bayes called binary multinomial naive Bayes. This algorithm follows from standard naive Bayes except that for each document, in our research each tweet, we remove all duplicate words before calculating the frequencies overall towards a given class. The occurrence of the word matters more than its frequency in a document. From an intuitive standpoint, this means that seeing a word once likely signals its future occurrences in the document, so knowing its first occurrence tells us about future cases. Additionally, for very rare words, we are only likely to get one occurrence so counting the frequency does not make a difference. We have chosen this version of the algorithm to create our sentiment classifier. Improvements to the base of this algorithm will be discussed further with the implementation of features.

Evaluation Method

Having chosen a model to classify data, we must choose a method to test the quality of the classifier. Like many other supervised learning problems, we separate our data into training and testing sets. The training set is used to train the model. Additionally, one can use a development test set (dev-test set) to tune the parameters of the model. Finally, the test set is used to see the accuracy of the model on new data. We test our classification output against gold labels, human defined labels for the document we are trying to classify. The test data, however, may not be an accurate representation of the true random data if we have too few tweets in the data. Another option is to use cross-validation. Specifically, we make use of 10-fold cross-validation. We randomly choose a training and test set division of the data, train the data and then compute the accuracy of our predictions on the test set. This process is repeated 10 times and each of the 10 subsamples is used exactly once as the test set. The overall accuracy is then taken as the mean accuracy of the 10 rounds. This method is also not without its limitations. Since all of the data used in cross validation is at one point used in the test set, we cannot examine the errors in the test set to suggest improvements to the model. Doing so would fit the model too close to the data given and risk overfitting, the inability of a model to generalize well to new data. To aide this problem, we do cross validation on part of the corpus and use that for analysis while keeping a separate set as the final test set. We will now discuss the implementation of the algorithm on specific data.

Data

Upon examining the data of the 3892 tweets more closely, we found that the majority of these tweets were labeled positive. There were 2832 positive tweets to 1060 negative tweets.

Table 1: Counts of Full and Equalized Data

	Total Words	Positive to Negative Words
Total Data	81,456	58,801 to 22,655
Equalized Data	44,563	21,908 to 22,655

The prior probability for a given tweet being positive was thus 0.727576. Since the data consists mostly of positive tweets, it is useful to test the inherent bias in this dataset: the extent to which the large positive prior affects the decisions of our system. If the bias is not too large in the full dataset, we want to use the full dataset as it contains the largest number of tweets available. To test these ideas, we decided to create a new dataset where the percentage of positive and negative tweets were equal. The equalized dataset, however, contained only 2120 tweets in total. The number of tweets is already quite small in the full dataset, despite the fact that naive Bayes tends to works well comparatively on small datasets. Table 1 above shows the difference in the number of words as defined by text separated by whitespace. We see the obvious size difference between the total data and the equalized data, which contains almost half the number of words. The number of unique words lowers from 12,203 to 8,459. This shows that there are many words which must repeat in the large dataset as the ratio is not as extreme as when comparing total word counts.

Initial tests and Tokenization

We determined to run the algorithm on both the full dataset and the equalized dataset to compare results. We ran an initial test using 10 fold cross validation. We used the bag-of-words unigram model that takes the probability of each word as a feature after having separated tweets into words based on whitespace. This is a very basic delimiter, but we only wanted to get a quick sense of the data. The accuracy in classification using the total dataset was 72.02% while 61.22% for the equalized dataset. Since the prior probability for a tweet to be positive is 72

Table 2: Improvements due to Tokenization

	Accuracy%	FP to FN
Total Data	.7871	38:20
Equalized Data	.7912	17:14

percent in the full dataset, we see that the classifier has just learned to predict positive. The equalized dataset, however, has surpassed its baseline of 50% by 10 percentage points. The equalized dataset has, as expected, a more balanced average false positive, tweets incorrectly labeled positive, to false negative, tweets incorrectly labeled negative, ratio (26.8 FP to 30.6 FN) than the total dataset (63.1 FP to 13.0 FN). In order to better separate a tweet into its word components in preparation for Naive Bayes, we use Christopher Potts's sentiment tokenizer. [3]

Tokenization is the process of breaking up a text into words, phrases, symbols or other meaningful parts called tokens. Using simple whitespace as a delimiter can miss important punctuation marks. Additionally, separating every punctuation mark fails to capture elements such as emoticons which are essential for sentiment analysis. Potts's tokenizer captures 96% of emoticons used on Twitter. The tokenizer also captures phone numbers, html tags, Twitter usernames and hastags as well as other punctuation types. URL's and tweets directed at another user using the '@' symbol make up a signification portion of the training set but do not hold any distinctive quality in sentiment classification other than their present and count. As a result, in preprocessing we change any URL's to "html" and any @ followed by a username to "user."

The results from running Naive Bayes 10 fold cross validation on the two datasets, as seen in Table 2, are fairly similar. We see a large improvement due to the addition of tokenization in both cases. The false positive to false negative ratio is, as expected, smaller for the equalized data. We attempted to use equal priors in calculating conditional probabilities with the larger dataset to see the extent to which the prior was influencing the classification and whether we

[3] `http://sentiment.christopherpotts.net/code-data/happyfuntokenizing.py`

obtained improved results with the equalized probabilities. The classification accuracy in this test lowered slightly to 78.53% while the ratio of false positives to false negatives was much more balanced (28.1 FP to 30.3 FN). We chose to perform the following experiments on the full data with unequal priors but return to the idea of equalizing the priors later on.

Feature Implementation

In hopes of improving classification accuracy, we implemented state of the art features for sentiment analysis on Twitter as used in Mohammad et al., 2013. We create a binary array according to the presence or absence of our features and then multiply the probability of that feature vector by our conditional probabilities from above to obtain our final conditional probability.

An important type of feature includes punctuation. Exclamations, interrogations and other punctuation marks expressing emotion often summarize the sentiment of a tweet. In light of this, we used the presence of exclamation points and question marks as features. While exclamation points often point towards a positive emotion (within the data as a whole they are twice as likely to appear in positive tweets) they may also appear as a simple emphasis of a negative emotion or part of a mixed feeling as expressed in punctuation such as '?!.' With regards to the question mark, the sentiments are not entirely clear as the user can simply be asking a question or expressing disbelief in a positive manner as opposed to necessarily expressing anger or negative questioning. Question marks only appear slightly more often in negative tweets in the data (1.19 times more often). Understanding the complexities in these punctuation marks, we also added a separate feature for each case where these marks end the tweet. This is done under the belief that the end of the tweet holds some summarizing information about the whole tweet. The question mark and exclamation features added only slightly to the accuracy of our system, however. Additionally, we grouped emoticons to add in the classification. Emoticons often directly express the sentiment of a tweet. However, there is a wide range of emoticons that express similar sentiments. To aid in our classification we replaced all emoticons identified through tokenization with the word "PosEmoticon," "NegEmoticon" and "NeutEmoticon" to express basic sentiments as well as "ExPosEmoticon" and ExNegEmoticon to express stronger polar sentiments. We obtained a list of emoticons classed into the above 5 polar categories from Agarwal et al 2011.

Besides using punctuation we tested to see the characteristics of the words our system was dealing with. We first ensure that each word is lowercased before examining its frequency as words such as 'He' and 'he' should be considered identical. We account for all-caps words at a later time. As a reminder, we use the bag of words model which takes the testing tweets and treats them as a set of words.

Mohammad et al., 2013 suggest using elongated words as a feature. An elongated word is any word that contains a character repeated more than two times, such as "coool." This often adds emphasis to an emotion. Whereas that paper made the number of elongated units a feature we use the presence of two or more of these elongated words as a feature. As a note, contiguous dots are kept as a single unit by our tokenizer and are only broken up into single units by the tokenizer if they include other punctuation marks in between. In the data there were 853 occurrences of elongated words. The elongated feature goes in hand with the idea of internet slang. Often users misspell words in order to either abbreviate a word, to emphasize an idea or just by mistake. As a result, words like cool and coool are treated as two different words in the vocabulary. To account for this, we normalized the vocabulary using a dictionary of 41,181 common pairs of misspellings/slang words. [4] There were 470 occurrences of these misspelled words in the data. Related to these misspellings is the use of hastags on Twitter. We added the presence of a hashtag as a feature and this resulted in a slight increase in classification accuracy.

[4]`https://github.com/coastalcph/cs_sst/blob/master/data/res/emnlp_dict.txt`

In the total data there were 832 hashtags found in positive hashtags as opposed to 271 found in negative tweets. Worried about the bias that the lack of data and the overabundance of positive tweets might have on the results, we ran the model just using these engineered features, and not words, in calculating conditional probabilities. The accuracy lowered to 70.63%. This shows the importance of the vocabulary in determining sentiment and that features, although important for refining a system and obtaining better accuracy, cannot work by themselves.

Negation

We finally examine the importance of negation in determining sentiment. Sentiment words behave quite differently when placed within a negative context. We have built a bag-of-words model in which the presence of words takes precedence. Thus many positive words will weight the conditional probability towards being positive even though the words may be expressed following a negation. Jurafsky and Martin describe a common baseline to deal with negation. It consists of adding a negation marker to every word following a negative token such as can't until the next punctuation mark. Christopher Potts expands upon the intuition and implementation of this idea [5]. Potts explains two cases of negation on words. When a mildly sentimental word such as good is negated, it behaves as its opposite, not good. However, when intense words such as superb and terrible are negated, the meanings of the resulting phrase are more general and harder to pinpoint. Not superb can have ranges of meaning. Thus one cannot define a general rule such as reversing the sentiment score of each negative rule, but rather must deal with the specific words being treated. Potts proposes adding the suffix _NEG to every word in a negated context and defines a negated context as the words following a negated token and before the next clause-level punctuation defined as: ".", "::, ";", "!" and "?." We make use of this implementation and the regular expression used to identify negative tokens. Implementing this

[5] `http://sentiment.christopherpotts.net/lingstruc.html`

negation technique resulted in the second largest gain in accuracy, behind tokenization. Using just tokenization without other features and adding negation results in an increase from 78.71% accuracy in ten fold cross validation to 78.92%. There were no major changes in the ratio of false positives to false negatives; there is still about 1.9 false positives to every false negatives, although the total average number of errors during cross validation is less due to the increased accuracy. With regards to the discussion of negation on mildly and strongly sentimental words, we can see the pattern even with the small dataset we use. The word good is 8.73 times more likely to be in a positive tweet as opposed to a negative tweet in our dataset. Its negation, however, is 2.21 times more likely to be found in a negative tweet than a positive tweet. This is not a direct opposite sentiment, but we see a clear reversal of the more likely class. On the other hand, the word terrible is 7.74 times more likely to be found in a negative tweet. It's negation is 1.93 times more likely to be found in a negative tweet. As we see, it is not the opposite sentiment, but rather a more general and perhaps less intense version of the word, reinforcing our intuition.

Filtering Techniques

Jurafsky and Martin explain the problem of overfitting, when the weights of the features attempt to perfectly fit the data and thus do not generalize well to new data. In another case, working with large datasets makes it too computationally expensive to deal with all features generated. They discuss regularization in which a regularization term is added to the objective function for optimization and will help give weights to the most important features. As they mention, however, many types of classifiers such as Naive Bayes do not have regularization, so feature selection must be used instead. Feature selection assigns some metric to each feature, ranks the features and then keeps the most pertinent ones. In our research, we have very little data, so the probability of words appearing in certain classes is not always accurate. As a result, non-sentimental words may bias the conditional probability more than the truly sentimental words. We want to see which words are the most important for determining sentiment and potentially discard the remaining words to improve classification accuracy. Feature selection methods help to obtain the most pertinent words for classification.

Seeing how word count plays a large role in Naive Bayes, we first tested filtering based on the total frequencies of words in all the tweets. We initially removed words that have a count of one. It is possible that these include strongly sentimental words that simply do not occurr frequently in the corpus, though it is more likely that these are just neutral words such as proper nouns and named entities as well as unusual constructions. Keeping these words can worsen the system in that future classification will be biased towards a given class if this word appears again, despite its lack of sentiment. This filtering removes on average 6451 of approximately total 10,500 words, leaving 4043 words left in the vocabulary. Our vocabulary thus contains on average mostly words that occur only once in the training data. This reflects the problem of lack of data. We record the results for our filtering methods in Table 3. The "%" removed refers to the preset percentage of the vocabulary from training that we wish to filter. This is not applicable to cases where we remove according to count, which does not conform to a percentage a priori. The "# in vocab" refers to the number of words on average per tweet during testing cross validation which are not in the vocabulary creating during the training step. "# not in vocab" refers to the average number of words per tweet in testing which are not seen in the training vocabulary. As seen in the last column of Table 3, there was a fairly large decrease in accuracy to 76.51% using this filtering by count-of-one system. As a followup to this test, we attempted removing all words with a frequency of two or less. 7662 words were filtered on average, leaving 2696 words. We saw an even larger decrease in accuracy. From an intuitive view we have reached a point where we have started removing words significant for determining the sentiment of a tweet. Interestingly, the number of words in new tweets that

Table 3: Affects of Filtering

Method	% removed	# in vocab	# not in vocab	accuracy
count = 1	N/A	17.4445	3.5132	0.7651
count <= 2	N/A	16.6742	4.2835	0.7044
IG	5	18.4007	2.5570	0.7993
	10	18.3191	2.6386	.7978
	25	18.0118	2.9460	0.7783
	75	16.5949	4.3629	0.6676
PMI	1	14.6077	6.3500	0.7430
	25	4.7680	16.1897	0.5621
	50	1.9684	18.9893	0.2900

are in the vocabulary from training does not decrease much as we move from removing words with one occurrence to those with two. Thus, the decrease in accuracy shows us that words with two occurrences, while infrequent, play an important role in determining sentiment for our task. Since removing infrequent words did not improve classification accuracy, we then tried removing stopwords from the vocabulary during training. Stopwords are words often filtered due to their highly frequent nature. They include words such as "a", "and" and "the" which are thought to not aid significantly in such tasks since they do not inherently express a sentiment. There is no definite list of stopwords but we used a given list found online, [6] In our case, removing the stopwords reduced the accuracy of our system by 8 percent. These stopwords are likely the most frequent words in our vocabulary due to the lack of a large dataset. As a result, we do not have enough sentiment words to perform the task accurately without these words. Before proceeding to other standard feature selection techniques, it is worth noting that most of the following techniques are used after already removing stopwords. However, these techniques and trends can still be applied to our task with stopwords present.

Information gain and pointwise mutual information (PMI) are two metrics for measuring the importance of features in classification tasks. Rather than removing words based on frequency,

[6]`http://www.ranks.nl/stopwords/`

we will calculate these metrics to determine to what extent certain words affect our classification. Jurafsky and Martin introduce the metric of information gain. For a given word w, the information gain of that word is:

$$IG(w) = \sum_{i=1}^{C} P(c_i) log P(c_i) + P(w) \sum_{i=1}^{C} P(c_i|w) log P(c_i|w) + P(\bar{w}) \sum_{i=1}^{C} P(c_i|\bar{w}) log P(c_i|\bar{w})$$

while pointwise mutual information between a word w and a class c is:

$$PMI(w, c) = \log \frac{P(w, c)}{P(w)P(c)}$$

$$= \log \frac{P(w|c)}{P(w)}$$

Using the above formula for information gain, we calculate the information gain for each word in the training vocabulary and then remove a given percentage of that vocabulary based on the information gain score. To avoid cases of log 0 we added a smoothing to the probability calculations similar to the laplace smoothing performed on counts in training. We began our filtering with information gain removing 75% and 90% of words, which in fairly drastic reductions in accuracy (75% filtering results in 66.76% accuracy). We test filtering various percentages of words, as shown in Table 3. Seeing that the number of words in new tweets that were not part of the training vocabulary was about 75% of words, we tested filtering lower pourcentages of the vocabulary. The best system turned out to be information gain removing 10% of the vocabulary. Most of the words (18.32) in new tweets are seen in the training vocabulary while only certain (2.64) are not present there. An interesting note is that only 6.6 stop words were removed on average during filtering. This plays an important role in comparison with PMI later on.

We next tried to filter by pointwise mutual information. Since PMI calculates the correlation between a word and a given class, a value had to be calculated individually for the negative and for the positive class for a given word. As a result there were many cases of 0 counts so we added the same smoothing to the counts as in information gain. We initially tried to combine the two classes for a global score by using an average score as well as a max score but these tests failed to yield any promising results. Since we are creating scores for both negative and positive classes, we do not take an absolute score and remove words below a thresthold. The two classes are on a different scale. Rather, we score the positive and negative vocabularies (the vocabulary of words that occur in positive tweets and that of those in negative tweets) and then remove the same amount from each of the vocabularies. As a result we are not removing an exact percentage of the overall vocabulary since there is often overlapp between the two class vocabularies. PMI performed poorly in practically all cases. What is notable is that after only removing about 25% of the overall vocabulary using PMI the number of words not in the vocabulary is already muuch greater than those in the vocabulary (16.19 to 4.77). Additionally, compared to information gain, pointwise mutual information removes many more stopwords. Removing 1 percent of the vocabulary with PMI already removes 111.4 stopwords (number of stopwords not unique stopword count) on average during training. The system at that point is only removing 117 words thus the 1 percent with the lowest score is practically all stopwords. Information gain at 50% removes only 11.6 stopwords on average versus 286.5 for PMI). We will examine these results in the context of previous work in the area of feature selection.

Bigram Model

In hopes of capturing more complex relations between words, we decided at this point to add bigrams, sequences of two adjecent tokens, as features.

Table 4: Bigram Model

Bigram Model	Accuracy%	False Positives	False Negatives
Basic	.7765	17.9	42.9
Info 10%	.7191	9.2	67.2
1 count removed	.4779	1.5	140.5
Equal Dataset	.7939	20.5	10.0

Surprisingly, adding bigrams did not improve the model accuracy. Also notable is the large ratio of false negatives to false positives. The bigram model performs better than the model with or without information gain (.7824 and .7892 %), however, when tested on the dataset which has an equal number of positive and negative tweets.

Related Works on Feature Selection

Yang and Pederson 1997 evaluate five methods of feature selection for text categorization and find information gain to be the most effective. Meanwhile pointwise mutual information, what they refer to as mutual information, is the least effective. On the otherhand, Mladenic and Grobelnik in *Feature selection for unbalanced class distribution and Naive Bayes* find information gain to be a poor feature seleciton method and discuss the reasons for differences between their study and that of Yang and Pederson. As Mladenic and Grobelnik note, Yang and Pederson agree with them on the general idea of feature selection in using a small feature subset since it gives as good or better results than using a large subset. A reduction in the number of features of up to 90% resulted in as good or better results in the two studies. Additionally, they agree that using a simple term or document feature ranking to create a subset gives good results as well. Mladenic and Grobelnik, however, emphasize the differences in results as due to the difference in domain and classification algorithms used in the two studies.

Yang and Pederson deal with two m-ary classifiers, a k-nearest neighbors classifier and a regression method named the Linear Least Squares Fit mapping. The details of the algorithms are not revelevant. We just note that the domain for the text classification includes one class value for each category as opposed to Mladenic and Grobelnik who split the problem into subproblems that correspond to a category and have a binary-valued class. They want to categorize documents relevant for a selected category where usually only between 1 and 10 percent of the examples belong to this category. This case of highly unbalanced class distribution pertains to the idea of asymmetric misclassification costs; it is the target class value for which we want to get our predictions and thus it is preferable to err by false positives over false negatives. It is more important to misclassify than to not classify a tweet that belongs to a category. The unbalanced class distribution is relevant to our research as we have a large porcentage of positive tweets. We do not have any reason to prefer one misclassification type over the other at this point except in terms of equalizing the false positive and false negative rates. Additionally, they also use a Naive Bayes classifier as in our study. .

Mladenic and Grobelnik break down the formula of information gain and its relationship to word occurrence in a document. Intuitively the prior probability that a word occurs, P(W), is fairly small. As a result, most of the top features according information gain are features with a relatively high probability that the word does not occur, $P(\bar{w})$. Thus the most informative features according to information game come from the third part of the equation above, where $P(\bar{w})\sum_{i=1}^{C} P(c_i|\bar{w})logP(c_i|\bar{w})$ is large. They emphasize this fact that knowing that a word W does not occur in a document holds information about the class value while, intuitively, a good classifier should be based on words that do occur in a document. They state that classification based mostly on the absence of words requires a larger feature subset than classification based on word occurrences. This could partially explain the fact that information gain on our data sets works best with only removing 5 or 10 percent of the words. This leaves open the question of whether other feature selection methods could remove a larger percentage of features while maintaining or improving results. As seen in Table 5, removing words with count equal to only

Table 5: Information Gain: False Positives to False Negatives

Information Gain	5%	10%	25%	75%	90%
false positives	31.6	26.5	16.8	6.3	1.6
false negatives	23.2	27.9	43.5	84.1	129.7

one removes about 62% but maintains a better accuracy than PMI. This will be discussed further in a later section on future areas for research.

Mladenic and Grobelnik point that that since they have a very unbalanced class distribution of 90% of the documents being of the negative class, most of the features are characteristic for the negative class. This idea can be extended to our model where a majority of the tweets are positive. Analyzing the false positives and false negatives for removing varying percentages of the vocabulary according to information gain supports this idea.

The table above presents the average number of false positives and false negatives in 10-fold cross validation for removing various percentages of the vocabulary with information gain. We see a decrease in the number of false positives and an increase in false negatives as we filter more words from the vocabulary using information gain. This coincides with the greater prior positive probability and thus the most informative features coincide with positive probability. We just lose many negative words and tend to mislabel tweets negative since there are more words missing from them. As we remove words and keep positive words, we are more likely to encounter words we have not seen before. At this point, words we have not seen are considered more likely to be negative thatn positive, which contributes to the increase in false negatives.

In terms of pointwise mutual information, the arguments of both articles support our results that pmi is not very effective. PMI is strongly affected by rare words. Yang and Pederson mention how the pmi score is strongly affected by the marginal probabilities of terms. The above formula for pmi can be rewritten.

$$PMI(w,c) = logP(w|c) - logP(w)$$

As a result, if two terms have the same conditional probability then rare terms with a lower P(w) will be favored. In this way PMI does not work well when terms differ greatly in terms of their frequency. Our results validate the fact that pmi favors rare terms. Common words such as stop words are often deleted, but as we saw above these often play a role in determining sentiment on our data. While the bias of pmi towards low frequency terms is known, Yang and Pederson note the problem of probability estimation errors. This refers to that fact that pmi is sensitive to estimation errors when the P(w) is near zero in the formula above.

Final Naive Bayes Classifier

Before analyzing the errors on test data we needed to chose our final system for testing. We ran cross validation on 70 percent of the full dataset (2724 tweets). Two systems gave the highest, and similar, accuracies during cross validation. Both of these systems implemented the following features discussed earlier: preprocessing changing URL's to 'html' and @'s to 'user' as well as grouping emoticons and correcting misspellings; tokeinzation; negation; presence of '!'; presence of '?'; presence of '...'; presence of a hashtag; presence of more than two upper-cased words; presence of more than 2 elongated words; tweet ends in '?'; tweet ends in '!'. One classifier did not use any filtering techniques but used an equal prior distribution in calculating probabilities on the full dataset. The other system used information gain filtering 10% and an unequal prior distribution. Since both systems had a similar cross validation accuracy of about 80% we needed a new metric to pick the better system. An F statistic is a measure of a system's accuracy that considers both precision and recall. Precision is defined as the number of true positives over the combined number of true positives and false positives. Recall is defined as the number of true positives over the number of true positives plus false negatives. An F1 statistic can be seen as a weighted average of precision and recall. An F1 score is best at 1 and worst at 0. It is defined by the following equation:

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

The F1 statistic for system using equal priors was .862 as opposed to .8597 for the system with information gain filtering and unequal priors. As a result, our final system used for analyzing errors used the above features with no filtering methods and an equal prior distribution. We ran this classifier on a test set of 1168 tweets after learning based on 2724 tweets. We obtained 81.59% accuracy on the new data of the test set and had 215 errorrs. We proceeded to examine each of the misclassified tweets to try to group the errors in order to find ways for improvement. We will explain the types of errors encountered and then open the discussion to possibilities for improving our classifier based on these errors.

Analysis of Errors

Before analyzing errors we found it important to recall how the tweets were classified in the first place. In the article *SemEval-2015 Task 10: Sentiment Analysis in Twitter*, Rosenthal et al. describe the procedures for producing this Twitter dataset. They used Amazon's Mechanical Turk service to annotate the tweets according to various tasks. Each tweet was annotated by five Mechanical Turk workers who had an approval rate of greater than 95% and had previously completed at least 50 approved Human Intelligence Tasks (HIT's) such as this annotation. The so called Turkers had to mark all subjective words/phrases in the tweet and indicate their start and end positions as well as their sentiment (positive, negative, neutral). They also indicated the overall sentiment of the message. Message annotations were discarded if they contained conflicts such as overlapping subjective phrases or if every word was marked as subjectve, among others. A word had to be considered subjective by 3 of the five Turkers to be marked as subjective. Afterwards the word had to be labeled a particular sentiment by 3 of the five

Turkers to be finally labeled that sentiment. At first we had mistakenly used the labels of the sentimental phrase or word in the tweet as the overall sentiment of the tweet. However, on analyzing the errors from the system there were over 25% that we felt were misclassified to begin with, so we realized there was a mistake with the data and found the correct labels. This mistake, however, points out a key difficulty of this type of sentiment analysis. There are often words or phrases that express one sentiment while the overall sentiment or feeling of the tweet may be the opposite. As an example, we analyzed the following tweet which has a positive sentimental phrase: "Tell me that Kevin Rudd's cat dying did not just make national news. C'mon Australia, we can do better...." However, it is clearly expressing a negative emotion towards the story actually making the news.

The largest category of mistakes, consisting of about 42% of the total errors, were tweets in which we felt one or a couple sentimental words were key to the tweet but the remaining majority of truly non-sentimental words were skewing the probability. This is largely due to the scarcity of data and examples; our classifier is unable obtain a good idea of the probability that a word usually expresses a certain sentiment or it does not express the sentiment as drastically as it should. For example, the word "fighting" is 1.52 times more likely to be positive than to be negative. While the word can vary in connotation, it seems odd that overall it would have this positive sentiment. On the other hand, a proper name like "Manning" is 1.97 times more likely to be negative than positve. This is simple a neutral named entity that on the whole should not express a strong sentiment. Meanwhil,e a word like "explode" which expresses at least serious action is 1.01 times more likely to be positive, which is very neutral. The point of these examples is to show that our lack of data means that we do not have enough examples to automatically understand the sentiment of every word. While the probability of some words is aptly captured (for example the word 'best' is 5.72 more likely to positive than negative), these words are often obscured by many neutral words that are given a highly sentimental value.

In addition to errors containing single words that carry a sentiment, we found about 18% of errors where the sentiment was carried within a more complex phrase or expression rather than a single word. For example, "Danica Patrick let her temper get the best of her Sunday and knocked herself out of NASCAR race http://t.co/ziT7wf95" expresses a negative sentiment despite the positive word best making up the phrase "get the best of." This mirrors what we have mentioned previously about the presence of a sentiment word or phrase within the context of a tweet expressing the opposite sentiment. Our bag of words system using unigrams does not capture these more complex relations between phrases.

About 13% of tweets misclassified were positive tweets with complex negation patterns that our system thought of as truly negative. Expressions like "can’t wait" or "couldn’t help but" which express a strong desire were taken as negative and we added the suffix _NEG to words following it even though these words are truly in a positive sentimental context. Also, there are cases where the negation is used more in a adjective sense. In the example "So pumped for no school tomorrow and sleeping in and even more excited for the hooch Tennessee here I come!," no is simply negating school and describing just that word and not all of those that follow it. However, our system adds the negated suffix to all words following it and classifies the tweet as negative.

Connected with this idea of negation and capturing more complex relationships, we found about 7% of tweets which contained the conjunction "but" which separated the phrase into two parts expressing varying sentiments. In the following difficult example "Right then..diet starts Monday, probably not the wisest choice with christmas coming up..but im gonna smash it!!," the first part of the sentence expresses some regret while the second part expresses a strong positive sentiment that trumps the beginning of the phrase.

Additionally, in about 10% of tweets we felt that changes in tokenization could have helped our classifier’s accuracy. For example, the word "cannot" was not detected as a negative token so we did not add negative suffixes to words following it. Also, a double dot ".." was not marked as a clause level punctuation to stop the addition of negative suffixes to words. On another note, our preprocessing of replacing commonly misspelled words did not replace abbreviations and many slangs such as "idk" for "I don't know" or "idc" for "I don't care." We thus miss some negations. Additionally, replacing words like "im" with "I am" could improve performance since we would recognize more words as part of the vocabulary.

Finally, there were some tweets which were misclassified or confusing to classify. We found the following tweet interesting: "This President talking about growing the economy is like Billy Cundiff explaining to someone how to make a field goal in the 4th quarter." The tweet is labeled positive but its sentiment is not obvious without a prior knowledge of who Billy Cundiff is and his relation to making field goals in the 4th quarter. Billy Cundiff is an American football player at the kicker position who has a strong record of making field goals in the final quarter, which explains the positive sentiment of the tweet.

Conclusions on Naive Bayes and Improvements

Seeing that a large portion of the errors miss the essential sentiment words, much improvement could be made in finding a way to capture these essential words. Sentiment lexicons are often used to this effect. Sentiment lexicons, as used in and described in Mohammad et al., 2013., are lists of words with associations to positive and negative sentiments. Mohammad et al. create two lexicons by collecting hundreds of thousands of tweets and labelling their sentiment according to the presence of a hashtag word or emoticon. For example, tweets with "#joy" were labeled positive while tweets with "#sadness" were classified as negative. A score was then calculated for each word and then used alongside other features. The score used in that article is positive pmi minus negative pmi. The score calculated for each feature and polarity is then used in features based on the count of tokens with scores greater than zero, the total score as well as other calculations.

Besides using a sentiment lexicon, another way to emphasize important sentiment words is through filtering and feature selection. While we have tried pointwise mutual information and information gain, we have by no means exhausted the list of possibilities. Information gain performs much better than pmi, which can be expected from pmi's preference for rare words. However, we do not know how information gain compares to other techniques for our task. As stated earlier, Mladenic and Grobelnik find information gain to perform poorly with respect to other techniques. They find odds ratio and term frequency to be the best feature selection methods. Term frequency refers to the simple count of terms in the documents after stopwrods have been removed. The odds ratio for a term w is given by the following formula:

$$OddsRatio(w) = log\frac{P(w|pos)(1 - P(w|neg)}{(1 - P(w|pos))P(w|neg)}$$

As opposed to information gain where the absence of features is informative and where a negative class is favored (assuming we are hoping to classify into one group), odds ratio takes into account the fact that there is a target class. We are not truly using a target class since we do not prefer false positives to false negatives or vice versa. However, the difference between a classifier based more on the presence versus the absence of features is an interesting path to explore. The paper proposes variants on odds ratios as well as other feature selection methods which can be examined.

There are some tractable changes to our classifier's preprocessing and tokenization which can make improvements. As stated earlier, we can examine our negation tokenizer and cleaning the vocabulary to exchange slang words and abbreviations. Additionally, we can search for ways to deal better with hashtags and punctuation. We make use of the presence of a hashtag as a feature, but often the hashtag words themselves contain important information, especially at the end of a tweet. We could try splitting the hastag and its words and adding those to the vocabulary. Additionally, some exclamation points and question marks which occur at the end of the tweet or words that occur at the end of a tweet hold more importance. They often summarize the tweet, and it would be interesting to create a weighting system based on the part of speech and placement of the word within the tweet. Also, creating features according to the presence or count greater than a certain amount doesn't capture the same information as using the strict count of a feature such as an exclamation, that which is mentioned in implementations such as Mohammad et al., 2013.

While we were able to notice interesting trends, these tractable improvement as well as the methods mentioned above which were not implemented prevent us from drawing stronger conclusions. Improvements gained through tokenization and negation are well-founded trends. We also noted the weakness of PMI compared to Information Gain, although there are many other filtering techniques that must be tested before drawing conclusions about the effectiveness of such methods on this data. This importance of frequent words is evident in our model, but we must also consider how the addition of a sentiment lexicon, as well as other methods, aid the lack of data.

In spite of possible improvements to our system, some of the more complex errors with complicated structure do in fact point to the limitations of the bag of words model. For example, we cannot capture syntatctic relations between words, which can contain the key to complex sentiment. For this reason we choose to close the chapter on Naive Bayes and make a short study of state of the art word vectors and words embeddings.

Vector Space Models

In order to better understand text, there has been a search for new technologies for semantic processing of text. Turney and Pantel (2010) describe vector space models (VSM's) for semantic processing of text, an essential model for natural language processing. As they describe, the point of the VSM is to represent each text document in a collection as a point in space. The document may be represented by a vector in which the elements are taken from the occurrences in a document using various contexts. By keeping using these co-occurrences of terms, this models helps keep tract of the context in which the words appear. Points that are semantically similar are then close together in space while those less similar points are farther apart. Much of the work with the VSM is based on the distribution hypothesis that words that occur in similar contexts tend to have similar meanings. Thus by organizing co-occurences of words in similar contexts we can hope to obtain information about the similarities of words. With a large collection of documents and vectors, the vectors are organized into a matrix based on the context and relationship between the co-occurrence of terms. The article describes three categories of VSM's based on the matrix involved: term-document, word-context, and pair-pattern. In term-document matrices the row vectors of the matrix correspond to terms while the column vectors correspond to documents. However, full document length may not be the best way to test for for similary. As a result one may use word-context matrices in which the word in rows are matched to column vextors of phrases, sentences or other breakdowns of a document. Finally, pair-pattern matrices represent row vectors that correspond to pairs of words and column vectors that correspond to the patterns in which the pairs occur. The example given is "X cuts Y" and "X words with Y." Turney and Pantel mention the relationship of the VSM to document classification. In classifying a document we are implicited saying that the documents in one class are similar to each other. We can thus use the idea of document similary present in the VSM (often through a term-document matrix) to calculate document similarity and classify according to the most similar document. We implement this idea later on.

Baroni et al. 2014 compare these models based on the count of co-occurrences (count models) with a newer distributional semantic models called context-predicting models (also known as embedding or neural language models). They stress how raw occurrences for previous models have been shown to not work well, so various transformations are performed for reweighting and smoothing the vectors. This is generally an unsupervised task. Baroni et al. note the recent trend that instead of finding the vector and then reweighing it, the vector weights are directly chosen to optimally predict the context that the words appear in. Details aside, Baroni et al. conclude that these predictive models greatly outperform the count models. The toolkit they used to train the predictive models is word2vec from Mikolov et al. 2010. Mikolov et al. propose two model architectures for neural networks in learning the representations of words that minimize computational complexity. The continuous bag-of-words model predicts a word based on its surrounding context while the continuous skip-gram model predits words within a range of words before and after the current word, thus the mirror of the bag-of-words architecture. Le and Mikolov 2014 extend these ideas to the concept of a paragraph vector, an algorithm that learns fixed length feature representations from texts of varying lengths. The model is very similar to that used to determine word representations with the paragraph thought of as another word in the learning architecture used to predict the next word in the context. The paragraph acts as a memory of what is missing from the current context and for this reason the system is called the Distributed Memory Model of Paragraph Vectors as well as doc2vec.

Although we have abstracted away many of the details in this previous description, what we have found is a way to model word representations using a predictive model which is state of the art and has given better performance across many tasks. Out aim is to implement these word vector models to our Twitter classification task.

Implementation of Word Vectors

We make use of the gensim implementations of word2vec and doc2vec readily available online and well documented.[7] We also make use of the following tutorial which helps in preparing

[7]`https://radimrehurek.com/gensim/`

the data to be served to gensim and for testing.[8] We preprocessed the files slightly differently than before. We ran our tokenizer and then lowered-cased every word and removed punctuation to make the data suitable for doc2vec. We used a linear support vector machine (SVM) to classify the tweets. SVM's are more often used in these settings than Naive Bayes and aim to achieve the widest margin between the classes we aim to distinguish between. We achieved an accuracy of 75.94%, which was surprising. A possible cause for this accuracy is the lack of data. Neural networks are often trained on and work best with large datasets. On examining the word vectors and model created through training we saw that the model does not find similar words very accurately. Gensim provides an method to find words in space closest to a given word. We test this function on the word "terrible." The most similar words are "manager", "people", "tressa", "chill", "bad", "madison", "waaa", "understand", "never" and "is." While the words "bad" and "never" make sense, the remaining words are not relevant. Additionally, the measure of similarity for the most similar word is only .55 on a scale where 1 is the most similar. I assume that with a larger dataset we would get more reasonable representations. We downloaded the pretrained word vectors from Mikolov et al.[9] to get an idea of their results and to try to make use of them for our classification. The vectors are trained on the Google News dataset of about 100 billion words. The system finds similar words that are much more relevant: "horrible", "horrendous", "dreadful", "awful", "horrid", "atrocious", "horrific", "bad", "appalling", "horrible_horrible." Also, the similarity of horrible is .92 which is rightly very high.

Seeing how well the pretrained vectors find similar words, we attempted to make use of these vectors for our classification task. We used an idea similar to k-means clustering, where we calculated each word average vector and then an average vector for each tweet in the training set. We then took the average of all the positive tweet vectors and the average of all the negative tweet vectors. To classify a new tweet, we created a vector for the words and then

[8]`https://github.com/linanqiu/word2vec-sentiments`

[9]`https://code.google.com/archive/p/word2vec/`

overall tweet and classified it according to which mean it was most similar to in terms of cosine similarity. This classification, however, resulted in only 74.23% accoracy. Mitchell and Lapata in *Vector-based Models of Semantic Composition* speak about the ways to combine individual word vectors for larger phrases state how vector averaging is insensitive to word order, as in the bag-of-words model. The model then can't capture differences in syntatical structure. They ultimately find multiplicative compositional models to be most effective. We leave this possibility open for future work.

Conclusion

Our work aimed towards a better understanding of sentiment analysis on Twitter through an examination of the SemEval competition. We were able to examine common trends in sentiment analysis and text classification and their effectiveness when working with small amounts of data that are unbalanced in terms of class frequency. We note the ineffectiveness of the method of pointwise mutual information as well as some surprising trends such as the decrease in accuracy with bigrams. We found information gain to help somewhat in filtering. However, it is necessary to check other filtering methods as well as using sentiment lexicons before drawing further conclusions. While having access to more data would be an obvious addition here, we find a reasonable amount of room for improvement in terms of feature analysis, filtering and lexicons. We switched our focus following Naive Bayes to a more state of the art use of vector representations for words. It is noteworthy that word embeddings and doc2vec perform worse than Naive Bayes. Naive Bayes is known to perform well on low amounts of data and it would be interesting to do a comparison of the two models performance on data size. Moving forward with the research, we would like to be able to compete in SemEval and test our model on the same data and with the same metrics in order to better understand how our model compares. We look forward to SemEval 2017 to make this a reality.

References

[1] Apoorv Agarwal et al. “Sentiment Analysis of Twitter Data”. In: *Proceedings of the Workshop on Languages in Social Media*. LSM ’11. Portland, Oregon: Association for Computational Linguistics, 2011, pp. 30–38. ISBN: 978-1-932432-96-1. URL: `http://dl.acm.org/citation.cfm?id=2021109.2021114`.

[2] Marco Baroni, Georgiana Dinu, and Germán Kruszewski. “Don’t count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors”. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Baltimore, Maryland: Association for Computational Linguistics, 2014, pp. 238–247. URL: `http://www.aclweb.org/anthology/P/P14/P14-1023`.

[3] Marco Baroni and Roberto Zamparelli. “Nouns Are Vectors, Adjectives Are Matrices: Representing Adjective-noun Constructions in Semantic Space”. In: *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. EMNLP ’10. Cambridge, Massachusetts: Association for Computational Linguistics, 2010, pp. 1183–1193. URL: `http://dl.acm.org/citation.cfm?id=1870658.1870773`.

[4] Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2Nd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2009. ISBN: 0131873210.

[5] Svetlana Kiritchenko, Xiaodan Zhu, and Saif M. Mohammad. “Sentiment Analysis of Short Informal Texts”. In: *J. Artif. Int. Res.* 50.1 (May 2014), pp. 723–762. ISSN: 1076-9757. URL: `http://dl.acm.org/citation.cfm?id=2693068.2693087`.

[6] Quoc Le and Tomas Mikolov. “Distributed Representations of Sentences and Documents”. In: *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*. Ed. by Tony Jebara and Eric P. Xing. JMLR Workshop and Conference Proceedings, 2014, pp. 1188–1196. URL: `http://jmlr.org/proceedings/papers/v32/le14.pdf`.

[7] Tomas Mikolov et al. “Efficient Estimation of Word Representations in Vector Space”. In: *CoRR* abs/1301.3781 (2013). URL: `http://arxiv.org/abs/1301.3781`.

[8] Jeff Mitchell and Mirella Lapata. “Vector-based models of semantic composition”. In: *In Proceedings of ACL-08: HLT*. 2008, pp. 236–244.

[9] Dunja Mladenic and Marko Grobelnik. “Feature Selection for Unbalanced Class Distribution and Naive Bayes”. In: *Proceedings of the Sixteenth International Conference on Machine Learning*. ICML ’99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 258–267. ISBN: 1-55860-612-2. URL: `http://dl.acm.org/citation.cfm?id=645528.657649`.

[10] Saif M. Mohammad, Svetlana Kiritchenko, and Xiaodan Zhu. “NRCCanada: Building the State-of-the-Art in Sentiment Analysis of Tweets”. In: *In Proceedings of the International Workshop on Semantic Evaluation, SemEval ’13*. 2013.

[11] Sara Rosenthal et al. “Semeval-2015 task 10: Sentiment analysis in twitter”. In: *Proceedings of the 9th International Workshop on Semantic Evaluation, SemEval* (2015).

[12] Peter D. Turney and Patrick Pantel. “From Frequency to Meaning: Vector Space Models of Semantics”. In: *J. Artif. Int. Res.* 37.1 (Jan. 2010), pp. 141–188. ISSN: 1076-9757. URL: `http://dl.acm.org/citation.cfm?id=1861751.1861756`.

[13] Yiming Yang and Jan O. Pedersen. “A Comparative Study on Feature Selection in Text Categorization”. In: *Proceedings of the Fourteenth International Conference on Machine Learning*. ICML ’97. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, pp. 412–420. ISBN: 1-55860-486-3. URL: `http://dl.acm.org/citation.cfm?id=645526.657137`.