Sentiment Analysis of User-Generated Twitter Updates using Various Classification Techniques

Ravi Parikh and Matin Movassate

June 4, 2009

1 Introduction

Twitter is a "micro-blogging" social networking website that has a large and rapidly growing user base. Thus, the website provides a rich bank of data in the form of "tweets," which are short status updates and musings from Twitter's users that must be written in 140 characters or less. As an increasingly-popular platform for conveying opinions and thoughts, it seems natural to mine Twitter for potentially interesting trends regarding prominent topics in the news or popular culture.

A successful sentiment classification model based on the expansive Twitter data could provide unprecedented utility for businesses, political groups and curious Internet users alike. For example, a business could gauge the effectiveness of a recent marketing campaign by aggregating user opinion on Twitter regarding their product. A user saying "I just used [Product A] today and it JUST BLOWS HARD!" would detract from the overall sentiment, whereas a user claiming "[Product A] is my FAVORITE product ever!" would add to the overall sentiment. Similarly, a political lobbyist can gauge the popular opinion of a politician by calculating the sentiment of all tweets containing the politician's name.

Obviously, this hypothetical application would be exceedingly useful. But to construct it, we would first need to build an accurate sentiment analyzer for tweets, which is what this project aims to achieve. Therefore, the problem we chose to tackle within natural language processing is to determine the sentiment of a given tweet. That is, given a user-generated status update (which can not exceed 140 characters), our classification model would determine whether the given tweet reflects positive opinion or negative opinion on

the user's behalf. For instance, the tweet "I'm in florida with Jesse! i love vacations!" would be positive, whereas the tweet "Setting up an apartment is lame." would be negative.

Sentiment analysis in Twitter is a significantly different paradigm than past attempts at sentiment analysis through machine learning, providing a dramatically different data set that proposes a multitude of interesting challenges. Notable past projects include sentiment classification of movie reviews. Twitter is different in that sentiment is conveyed in one or two sentence blurbs rather than paragraphs, leading to fewer ambiguities in the form of "This movie has [list of good characteristics over many sentences]. However, it is still not worth seeing." There are instead numerous other difficulties. Twitter is much more informal and less consistent in terms of language, and users cover a much wider array of topics touching on many facets of their life than the limited rhetoric of movie reviews (e.g. a movie they just watched, a test they're studying for, a person they're hanging out with). Also, sentiment is not always as obvious when discussing human-generated status updates; many tweets are ambiguous even to a human reader as to their sentiment. Finally, a considerably large fraction of tweets convey no sentiment whatsoever, such as linking to a news article, which provide some difficulties in data gathering, training and testing.

In this paper, we apply several common machine learning techniques to this problem, including various forms of a Naive Bayes and a Maximum Entropy Model. We attempt various optimizations as well based on error analysis and intuitions that are specific to the unique rhetoric and language of Twitter.

2 Data

We took advantage of a JAR archive called "jtwitter.jar", which leveraged a version of the Twitter API specifically for Java. We wrote a small app that pulled queries from Twitter's public timeline in real-time, and then these queries were evaluated and hand-tagged for sentiment by us. We had a total of 370 positive tweets and 370 negative tweets that were used for training and testing. Of these, we randomly chose 100 of each for testing, and the rest of the tweets were used for training. We considered acquiring more data, but both the Naive Bayes models as well as the MaxEnt classifier were able to achieve high metrics with this small training set.

This data set does indeed seem small, or at least small enough so that we would be unable to obtain preferable metrics with it. Data-gathering is perhaps the biggest issue that Twitter-based sentiment analysis poses com-

pared to more traditional problems of sentiment analysis such as movie or product reviews. Thus, we figured it would not be worth the significant extra time simply for a few more data points. However, as will be evident in the Results section, we were able to achieve excellent accuracy (particularly with the Naive Bayes classifier).

Each tweet is between 1 and 140 characters. We did not bother tagging tweets in foreign languages or tweets with excessive amounts of colloquialisms/mispellings that made it difficult to decipher for even humans. Additionally, many tweets do not convey sentiment; for example, "just tried pesto for the first time" is a tweet that is neither positive nor negative. Even tweets like "got promoted at work today" don't actually convey sentiment; the author here is not rendering judgment on whether this is a positive development, despite our cultural notions about the nature of promotions. Instead, a tweet like "got promoted at work today...gonna celebrate tonight!" clearly demonstrates positive sentiment. These tweets where sentiment was not conveyed were not used in training or testing.

Even with a robust Twitter API and a quick method of obtaining tweet classifications for training data purposes, a two-person effort was not enough to generate a truly sizable collection of positive and negative tweets (on the order of 1000 tweets). This was because a large majority of tweets either contain only a link, are in a foreign language, or convey no sentiment whatsoever, meaning tweets that conveyed some level of opinion or emotion were surprisingly difficult to encounter. On average, processing 200 tweets resulted in roughly 5-10 positive tweets and 5-10 negative tweets alike, with the rest expressing no demonstrable sentiment. This is an interesting reflection on how Twitter is used as a service, demonstrating that users turn to the site more to discuss events objectively rather than rendering judgment.

3 Code Structure

Our source code for testing and implementation was developed in Java. We took advantage of starter code from CS124 HW 3 and CS224n HW 3. The main files utilized are cs224n.assignments MaximumEntropyClassifierTester.java (a MaxEnt classification model), cs224n.classifier.UnigramBernoulliNaiveBayes.java, cs224n.classifier.UnigramMultinomialNaive Bayes (our Naive Bayes models), and cs224n.classifier.Preprocessor (a class that preprocesses tweets as described below).

In addition to these classes we created the class cs224n.classifier.TwitterRetriever, which leveraged the Java jtwitter API to pull tweets in real time from Twitter's public timeline and allowed easy hand-tagging. This file was

not used beyond this initial hand-tagging. We chose not to include this file in the final submission, since it takes advantage of a large library and is not useful in running our models.

4 Preprocessing

Through our error analysis and intuition, we noticed several characteristics of tweets that could be standardized through preprocessing. Twitter users are much more likely to have grammatical/spelling errors, colloquialisms, and slang incorporated into their output, due to the 140 character limit that is imposed on users. As a result, regular expression matching of common errors and substituting with standard language helped our models' performance by providing more consistent lexical behavior across tweets. For example, substituting any matches of the regular expression "lu?v?" with "love" allows all similar tokens to be classified as one. ("Luv" or "luuuuvvvv" are slang expressions for "love" within internet parlance). Another example is substituting the expression

```
wh?(a|u)t'?(s|z) up
```

with the phrase "what's up." The above expression enumerates 16 possible common ways that Twitter users might write this phrase.

We also eliminated some punctuation, such as periods, commas, and end marks. Twitter user names mentioned within a tweet (as signified by an @ sign appearing before the token) were also removed from our training and test data sets. While exclamation marks and other punctuation can be strong indicators of emotion and sentiment, we noticed that our performance increased without these. This is likely because exclamation marks can indicate strength of negative and positive emotion, but aren't particular to one. However, we chose not to eliminate colons and parentheses; often, Twitter users indicate sentiment with emoticons such as :) (a smiley face). We instead decided to standardize these, with a single regular expression encompassing several different forms of smiley faces. These punctuation-based optimizations also helped performance.

In addition, inspired by the practice of Pang and Lee in their sentiment analysis research, we stripped out the word "not" from tweets and appended the characters "NOT_" before the following token in a tweet. So for instance, the tweet "not feeling well today" would be altered to become "NOT_feeling well today." This way, negatory words could be matched together in the training process, lowering the probability of a tweet like "Family Guy is

NOT GREAT OR FUNNY jeez" to be incorrectly classified as positive due to the presence of the words "great" and "funny."

5 Unigram Naive Bayes

The ultimate task here for our sentiment analyzer is to calculate the probability that tweet d is in class c, where $c = 0$ or 1. We implemented two unigram Naive Bayes models. In both of these, we use the Naive Bayes simplifying independence assumption:

$$P(c|d) = P(c) \prod_{1 \leq k \leq n_d} P(t_k|c)$$

Where t_k denotes the kth token sequentially in a tweet, and n_d is the size of a tweet. The Naive Bayes assumption is that these probabilities for each token are independent, and thus the joint probability is merely the product. In order to compute the most probable class, we compute the arg max as follows:

$$\begin{aligned}
\arg\max_c \hat{P}(c|d) &= \arg\max_c \hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(t_k|c) \\
&= \arg\max_c \log(\hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(t_k|c)) \\
&= \arg\max_c \log(\hat{P}(c)) + \sum_{1 \leq k \leq n_d} \log(\hat{P}(t_k|c))
\end{aligned}$$

In the above equation, notice that $\hat{P}(c) = 0.5$, since we make the assumption that both negative and positive tweets are equiprobable (indeed, we have equal amounts of training and testing data for both). We have two different methods of computing $\hat{P}(t_k|c)$, however, yielding two unigram models. We implemented a multinomial model, in which the counts of the number of occurrences are used as follows:

$$\hat{P}_{multi}(t_k|c) = \frac{T_{ct_k}}{|V_c|}$$

Where T_{ct_k} denotes the number of times token t_k appears in class c, and $|V_c|$ is the number of total words appearing in class c. A slightly different model is the Bernoulli model, which instead records $\hat{P}_{ber}(t_k|c)$ as the fraction of tweets in which the term t_k occurs. In both of these situations, we used add-one (or Laplace) smoothing, so that zero counts would not adversely affect the performance of our Naive Bayes model.

5.1 Results

We achieved the following accuracy on the test set for each model. We trained on a total of 270 positive and negative tweets each, and 100 tweets each for testing. These metrics were achieved after preprocessing (note that the F1 score equal to $\frac{2np}{n+p}$):

- Multinomial Unigram: 0.81 negative, 0.91 positive, 0.857 F1
- Bernoulli Unigram: 0.85 negative, 0.85 positive, 0.850 F1

These metrics were achieved without word replacement preprocessing (regular expression matching), but keeping intact other preprocessing:

- Multinomial Unigram: 0.77 negative, 0.91 positive, 0.834 F1
- Bernoulli Unigram: 0.84 negative, 0.81 positive, 0.825 F1

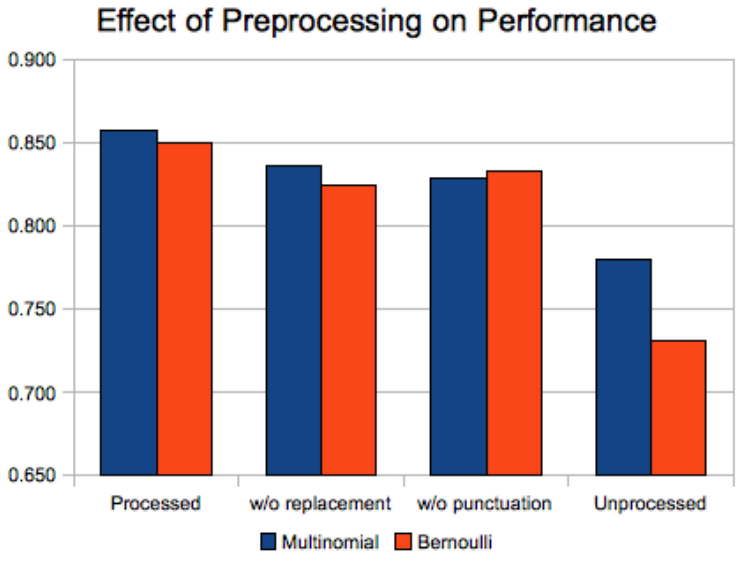
These metrics were achieved without punctuation preprocessing, but keeping intact other preprocessing:

- Multinomial Unigram: 0.80 negative, 0.86 positive, 0.829 F1
- Bernoulli Unigram: 0.88 negative, 0.79 positive, 0.833 F1

These metrics were achieved without any preprocessing:

- Multinomial Unigram: 0.78 negative, 0.78 positive, 0.780 F1
- Bernoulli Unigram: 0.87 negative, 0.63 positive, 0.731 F1

This shows a chart of performance under varying preprocessing steps:



In an effort to boost the size of our training data set, we also tried testing our classifiers using non-Twitter training data. Namely, we used pre-classified movie reviews from LingPipe to train on. Here were our achieved metrics:

- Multinomial Unigram: 0.92 negative, 0.26 positive, 0.405 F1
- Bernoulli Unigram: 0.80 negative, 0.46 positive, 0.584 F1

It was unsurprising that both models performed poorly on this data. It was interesting to note that both models were heavily skewed towards negative tweets, and classified a very high ratio of the total number of tweets as negative. The increased propensity for our Naive Bayes classifier to classify a tweet as negative indicates that our model is much stronger at correctly identifying negative sentiment, and substantially weaker at correctly identifying positive sentiment. Reasons for this anomaly are described in the following error analysis section.

5.2 Error Analysis

The majority of the errors made by both the Bernoulli and Multinomial Naive Bayes models above were identical, with a few exceptions. This was to be expected; due to the brevity of tweets, words tend to appear at most once in a tweet. Even common words such as "the" tend not to appear more than once in a 100-character tweet. As a result, the Bernoulli and Multinomial probabilities for several words will have the same effect on the computed probabilities. Thus our error analysis is for the most part common for both models. We do briefly address some sentences on which the two models differed, however, below.

5.2.1 Without Preprocessing

Without preprocessing, there were several basic errors that we later eliminated. (We address the errors that still persisted after preprocessing in the next section.) These errors were due to a multitude of reasons. The following are examples of misclassified tweets:

- "@epiphanygirl we gon' be like 15 deep coming to see u on Saturday in atlantic city. We luuuuvvvvv you!!! Lol!" - There were several instances of misclassified tweets where our classifier was unable to use strong indicator words to identify sentiment due to nonstandard though common misspellings. Clearly "love" is a strong indicator of positive

emotion, and indeed there were several tweets that we noticed in training and testing data that used variations of the word. However, the nonstandard spelling in this tweet confounded our classifier.

- Another example of nonstandard slang is in the tweet "Frack! The new Bones isnt up on hulu!!!!!!!!" Here, the expletive at the beginning is an indicator of negative emotion but is not a standard expletive, though its similarity to another commonly-used expletive makes it meaning obvious to humans.

- "It's soooo funny... Guys, U should watch it! (=" - This is another misclassified positive tweet. A large indicator of sentiment in many tweets are emoticons, or smiley faces and sad faces simulated with punctuation. This tweet contains "(=" which is a rarer emoticon for smiley faces (the conventional one is "(:"). This is another situation where tweets have a wide array of ways of expressing the same token. This was caught through preprocessing, with a regular expression identifying various forms of emoticons and standardizing them.

We tried to avoid overfitting our training and testing data; as a result, many regular expression word replacements that could have better standardized the language and potentially improved our results were not implemented.

We also noticed that several tweets had tokens such as "@Adree6603", where the @ sign indicates that the tweet is directed at another Twitter user with that monicker. These tokens are clearly not useful for determining sentiment, and every time these were observed, they were classified as unknown tokens. As a result another preprocessing step that we included was eliminating these, as they have virtually zero effect on the net sentiment of a tweet.

5.2.2 With Preprocessing

Some phrases still eluded our Naive Bayes classifiers even with preprocessing. The following are examples of misclassified tweets:

- "your realy meen" - a misclassified negative tweet. This was due to misspellings of indicator words such as "mean." This was not corrected during preprocessing; this is not a common typographical error, and thus it was not caught. There were several tweets which were likely misclassified by Naive Bayes since certain tokens were misspelled in an unconventional way. The rapid pace of Twittering and the 140-character limit of tweets encourages users to post their thoughts quickly and

frequently, through as many channels as possible, encouraging hasty spelling and unconventional grammatical quirks. The nature of the quirks are hard to normalize, even with adequate levels of preprocessing, so any dramatic misspellings similar to the preceding tweet are difficult to deal with properly.

- "i thought i saw a preview for that on mtv movie awards which was a joke" - Referring to something as "a joke" is clearly a negative sentiment indicating its uselessness or futility, and is the main indicator word for sentiment in this tweet. However, a joke is often associated with lighthearted, humorous and thus positive tweets, and therefore this tweet was misclassified.

- "got home from a long trip into the mountains happy to be back down battery charged" - The main indicator word in this positive tweet is "happy"; however, there are several other words such as "long" which might have a propensity for negative tweets that would lead this to be misclassified. Such nuances are difficult for a generative classifier like Naive Bayes to deal with, and the best way for approaching these complexities would be to add more complicated, additional features besides just the words involved in a tweet. Potential ideas for additional features are described in the Further Improvements section below.

We tried to avoid overfitting our training and testing data; as a result, many regular expression word replacements that could have better standardized the training data and potentially improved our results were not implemented. Instead, we only implemented regular expression substitutions for words that occur very frequently. Had we decided to sift through the training data and find commonly-occurring words, we would have skewed our results to more accurately fit the positive/negative words occurring specifically in the training data, causing a level of overfitting that would fall apart on the test data. SImilarly, if we had decided to sift through the test data and find commonly-occurring words, we would be destroying the spirit of the classifier assessment, and would cause an unfair bias towards the test data that would artificially inflate our results.

5.2.3 Differences Between Models

All tweets misclassified by the Bernoulli model that were classified correctly by the multinomial model were 6 positive tweets, and all tweets misclassified by the multinomial that were correctly classified by the Bernoulli were 4 negative tweets. This was just a coincidence, however. Most of the incorrectly

classified tweets in these cases were ones where they were borderline cases with a lack of clear indicator words, and the probabilities happened to go one way in one classifier and the other way in the other classifier. Thus, it seems that the class of our Naive Bayes model - Bernoulli or multinomial - made negligible differences on the classifications and overall accuracy of our Naive Bayes classifier.

5.3 Further Improvements

Several potential improvements could be implemented in the preprocessing phase, but would require careful consideration. We considered implementing a spell-check feature that would further standardize the observed lexicon, lessening the probability of encountering UNKNOWN tokens during testing and providing a more reliable and expansive classifier model. Essentially, it would work by reading in a token, determining whether it's present in a Lexicon, and then performing a maximum of 4 edits to see if any local set of changes would be a word within the Lexicon. But, as is evident from the Twitter user data we collected, there are several acronyms and slang terms that arise in Twitter that can be strong indicators of sentiment that would be too brazenly corrected by an automatic spell-checker. A useful spell-checking implementation would therefore need a large lexicon of these types of tokens, and creating this would be tedious.

Additionally, we could implement more nuanced features for the Naive Bayes classification instead of simply each word within the tweet. For instance, we could've taken advantage of a Part-of-Speech tagger to determine not only which words contribute to positive/negative sentiment, but also what POS those words have when they're influencing the sentiment of a tweet. For example, the two tweets "just fooling around before my econ final" and "my last test made me feel like a giant fool" convey completely different levels of sentiment even though they both involve use of the word "fool." Of course, a Part-of-Speech tagger requires relatively sanitized input, and Twitter user data has widely varying levels of grammatical and spelling deficiencies, so it did not seem feasible to include it at this level of development. This would increase the number of token types we'd see, and would thus increase their sparsity, so it would be necessary to attain more training data to offset this sparsity.

Also, from analyzing a number of tweets, it becomes evident that certain words that appear near trigger words such as "me" or "I", essentially pronouns which refer directly to the speaker him/herself, should be considered more important and have more weight towards describing the sentiment of the tweet. The intuition behind this is that nearby words to a first-person

pronoun directly describe the speaker's state, which is the best indication of the speaker's attitude and thus the overall sentiment of the tweet. For example, the tweets "I just finished watching Barca DESTROY MANCHESTER" and "that final completely destroyed me" both convey very different levels of sentiment, even though they both include the word "destroy," which would probably be a bigger indicator of negativity. But in the former example, "destroy" isn't being used in reference to the speaker's state, so it doesn't actually imply that the user is expressing sentiment like the latter example would.

For our Naive Bayes classifier, we made no effort to distinguish between low-content words such as "the" or "some". While actively filtering out low-content words and retaining high-content words could've provided a much more authentic distribution of how words and tokens affect overall Twitter sentiment (since "happy" is much more likely to affect the sentiment of a tweet than "the" is). Ultimately, however, we decided against implementing this level of filtering during training and testing, primarily because a Naive Bayes classifier is a generative model - specifically, in our circumstance, one that deals with binary outputs - and so it needs to compare both the positive and negative possibilities to determine the highest classification likelihood. Most of the time, low-content words would affect positive and negative possibilities equally; there's no reason to assume they would skew the distribution one way or another, though this may very well not have been the actual case.

6 Multinomial Bigram Naive Bayes

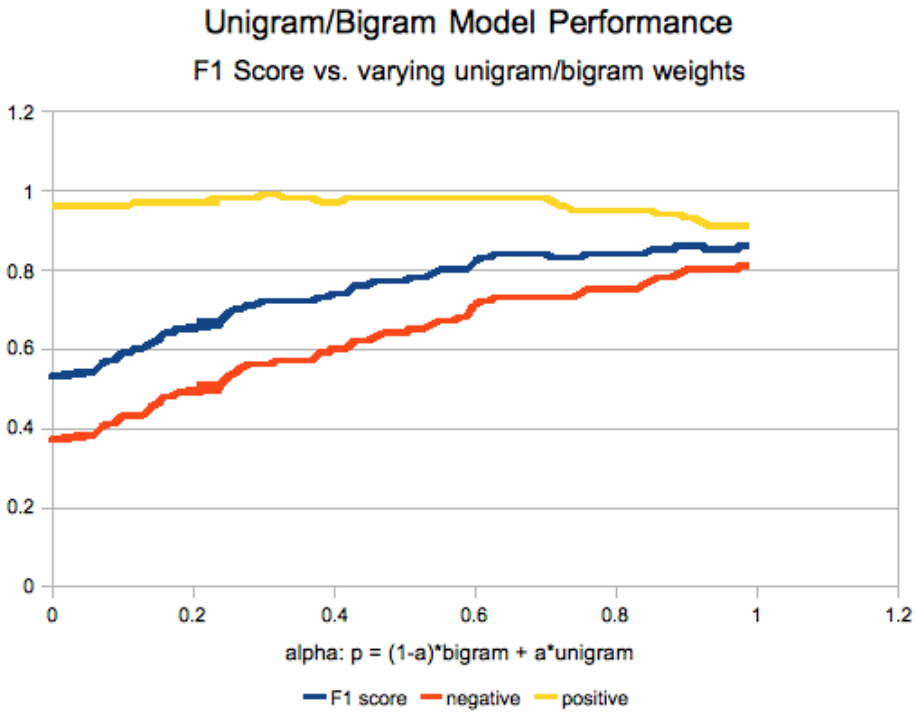
We also implemented a multinomial bigram Naive Bayes model, which calculated the log probability in a method similar to the multinomial unigram model, but using bigrams instead of single tokens. Due to the sparsity of the data, the majority of bigrams would appear only once in training or testing, and therefore it did not make sense to have a pure bigram model. Instead, we also calculated the log probability as given by the unigram model as well, and added these together using a weight α in a linear interpolation fashion:

$$P(c|d) = \alpha P_{unigram}(c|d) + (1 - \alpha) P_{bigram}(c|d)$$

6.1 Results

We tested different values for α, running the model incrementing alpha by 0.01 from 0.00 to 1.00. The optimal values of alpha were $0.90 \leq \alpha \leq 1.00$, due largely to the sparsity of bigrams in the data. The following chart illustrates

the F1 scores, as well as positive and negative classification accuracy, of our model on different values of α:



As shown above, the bigram model was unable to improve on the unigram model in any significant way. This might be partly due to the sparsity of the data. In earlier studies involving classification of movie reviews, there were many more tokens per review than the number of tokens per tweet, and the language was more standardized (i.e. followed rules of standard English grammar). As a result bigram data was more meaningful there, but in a 140-character limit status update, there's going to be few meaningful sets of bigrams to merit the use of a bigram model for Naive Bayes classification.

6.2 Future improvements

Increasing the amount of training data could possibly help, as it would provide a far larger set of bigrams. In fact, a larger training data set would much more greatly benefit a bigram Naive Bayes classifier than a unigram Naive Bayes classifier, for this very reason. There are other ways to address the sparsity of training data, however, and various Backoff methods or smoothing implementations could help us deal with the sparsity of bigram in our Naive Bayes classifier.

In addition, better preprocessing would allow the language to become more standardized and perhaps lead to a more effective bigram model. For instance, the bigrams "I'm bored" and "im bored" and "ima bored" all represent the same level of sentiment, but the preceding tokens would be bucketed into different portions of the Naive Bayes generative probability distribution, thereby lowering the probability mass of common constructs that are spelled in a variety of ways. This level of sparsity affects bigram distributions for the Naive Bayes model much more dramatically than it does a unigram-based Naive Bayes model.

7 Maximum Entropy Classification

The intuition of the MaxEnt model is to use a set of user-specified features and learn appropriate weights. We built an appropriately smoothed Maximum Entropy Classifier that aimed to select feature parameter values to maximize the log-likelihood of the tweet test data we generated. In addition, High weights given to features mean that these are strongly indicative of a certain class. The estimate of $P(c|d)$ for class c and tweet d is given by:

$$P(c|d) = \frac{1}{Z(d)} \exp(\sum_i \lambda_{i,c} F_{i,c}(d, c))$$

Where $Z(d)$ is a normalization function that ensures a proper probability distribution, $F_{i,c}$ are binary feature functions that give a value for the presence of feature f_i in class c in the tweet d, and λ are feature-weight parameters. These parameters are learned to maximize the entropy of the distribution.

7.1 Results

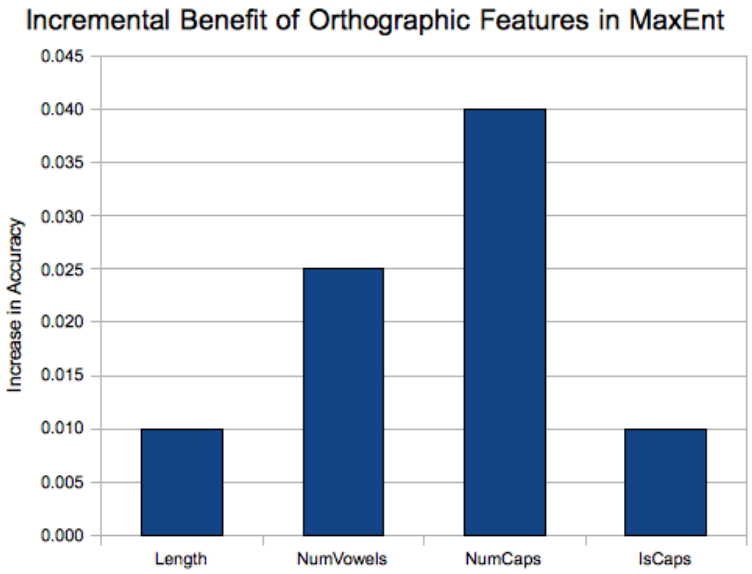
We achieved the following accuracy on the test set for each model. We trained on a total of 229 positive and negative tweets each, and 100 tweets each for testing. The highest metrics achieved were as follows:

- Negative Test Points: 100, Number Correctly Classified: 70
- Positive Test Points: 100, Number Correctly Classified: 48
- Overall Accuracy: 0.64

The features we took advantage of in our MaxEnt model are the following, with the description of each feature alongside the feature name:

Name	Description
WORD	the current word
LENGTH	the length of the current word
NUMVOWELS	number of vowels in current word
STARTSCAPS?	current word starts with a caps letter
NUMCAPS	# of capital letters in the current word
ISCAPS?	current word is in all capital letters?
SUF	suffix, last 3 chars in current word
PRE	prefix, first 3 chars in current word
ISCAPS_AND_PREVWORD	PREVWORD and ISCAPS?
PREVWORD	the previous word
PREV2WORD	word appearing 2 before current word
NEXTWORD	the next word
NEXT2WORD	word appearing 2 after current word
STARTSCAPNW	next word starts with a capital letter

This shows a chart of performance under various feature combinations. Namely, it indicates how certain orthographic features contributed to the overall accuracy of the MaxEnt classifier. The larger the bar, the more important the feature was in determining how a word described the overall sentiment of a tweet:



7.2 Error Analysis

Without preprocessing, our models tend to make several easily fixable errors, as shown above in our error analysis for our Naive Bayes models. Therefore

we chose not to detail our error analysis of our MaxEnt model without preprocessing, instead focusing on how it performed after preprocessing.

Comparing the dramatically different results between the Naive Bayes and MaxEnt classifier, it becomes clear that the MaxEnt method of taking advantage of sequenceable features simply doesn't apply well to the unique nature of tweets. Each different form of the Naive Bayes classifier worked overwhelmingly better than than the MaxEnt classifier, prompting us to stick with Naive Bayes as the primary mode of classification in future development of our Twitter sentiment analyzer.

However, we could explore Support Vector Machines (SVM) in future experiments, as Pang and Lee demonstrated comparable results to Naive Bayes and MaxEnt within their movie review analyzer. And while Pang and Lee did in fact demonstrate excellent results with a MaxEnt classifier on movie reviews, this is primarily because movie reviews are much more well-suited towards the sequence-based model that MaxEnt facilitates. Without leveraging CFG parsing-based or Part-of-Speech features, it becomes difficult to extract context-based features from a single given tweet that indicate levels of sentiment. The local context of a sentence within a movie review document greatly aids in determining the sentiment of that sentence, that is, determining whether the surrounding sentences have overlapping features helps in determining whether the current sentence will contribute to the overall review's sentiment. Comparing this to the sequence-based model applied to tweets, it becomes clear that the surrounding feature context of words within a tweet do very little in assessing the overall level of sentiment on a tweet. Thus, we couldn't take advantage of features based on locality, which are the truth strengths of the MaxEnt classification technique.

Most of the features we were able to leverage involved loose orthographic features, based solely on the characters and structure of the word being looked at (as MaxEnt calculates log probabilities on each word within a given tweet). Some examples involve the number of capital letters in a word and the length of a word. As indicated in our results above, the feature that most aided in our classification model was the number of capital letters in a word, as this increases when a user feels more passionately about a certain subject. However, though there are plenty of orthographic features, very few of them shed any significant insight on how the word itself will contribute to the overall sentiment of a tweet. So the fundamental issue with the MaxEnt classifier is as follows: the short nature of tweets limit the amount that MaxEnt can leverage locality and sequence, which requires us to resort to ineffective features based almost primarily on orthographic details (though we did include some combination features such as ISCAPS_AND_PREVWORD).

As evidenced by our results, there was a much greater propensity for

the MaxEnt classifier to predict a negative sentiment than it was to predict a positive sentiment, causing our percent correct for negative tweets to be nearly 1.5 times greater than the percent correct rate for positive tweets. One possible reason for this anomaly is that certain words are very flexible in the level of sentiment they convey, and that our training data happened to disproportionately present these words in a negative light. For example, the tweets "I am so freaking tired of proving myself to people" and "freaking 24 season finale TONIGHT!" indicate that the word "freaking" can occur in dramatically different levels of sentiment. In addition, a tweet of the form "today was not at all as bad as I thought it would be" could get classified as a negative tweet, only because of the token "bad" and the fact that the negating "not" occurs many slots before it.

For further improvement on our models, we will attempt methods of discovering beneficial feature templates that are less rooted in trial-and-error and ad hoc methods. Ultimately, all we could do to determine fruitful features for classification was to simply look over our Twitter data and determine In order to determine which features are successful and which ones arent, we were limited to simply making incremental changes to our feature set and seeing how that affected the existing classifier performance. But some features work better in the presence of other features, as some features can overlap with a given feature, thus decreasing their utility, or neatly complement a given feature, thus increasing their utility. Again, since no set of features were very compelling for determining the overall sentiment of a tweet, we simply resorted to these trial-and-error means of developing features.

8 Acknoledgments

We leveraged the starter code from CS124 assignment 3, which was a Naive Bayes model used to classify movie reviews by sentiment, as well as the starter code for CS224N assignment 3, in which we implemented a MaxEnt model. We also used the jtwitter Java API, available at http://www.winterwell.com/software/jtwitter.php, in order to more easily pull tweets off Twitter for hand-tagging. We also used sentiment-tagged movie review data from LingPipe (http://alias-i.com/lingpipe/demos/tutorial/sentiment/read-me.html), but only briefly to compare different training data sets, and not in any other substantive way. We also would like to thank Professor Manning and the entirety of the CS224N course staff for expansive feedback regarding our proposal and advice on how to proceed with our project.

9 Conclusion

Sentiment classification on tweets is a significantly different problem than performing similar analysis on movie or product reviews. In the latter cases, there are several problems that arise in a long-format review in determining sentiment. However, Twitter offers an entirely different challenge, which is largely created by the nonstandard and diverse language employed by Twitter users.

We implemented two Naive Bayes unigram models, a Naive Bayes bigram model and a Maximum Entropy model to classify tweets. We found that our Naive Bayes classifiers worked much better than our Maximum Entropy model could. Our results were quite good, and both models performed better than known performances on movie review classification, for example. However, our best accuracy still leaves room for improvement, which could come in the form of better preprocessing and more clever feature selection. In addition, we could take the time to collect much more data (a factor of 10 times more, perhaps), making sure to implement the proper preprocessing techniques to ensure that overfitting doesn't occur in the widely varying class of language that can occur on Twitter.

Even with a robust Twitter API and a quick method of obtaining tweet classifications for training data purposes, a two-person effort was not enough to generate a truly sizable collection of positive and negative tweets (on the order of 1000 tweets). This was because a large majority of tweets either contain only a link, are in a foreign language, or convey no sentiment whatsoever, meaning tweets that conveyed some level of opinion or emotion were surprisingly difficult to encounter. On average, processing 200 tweets resulted in roughly 5-10 positive tweets and 5-10 negative tweets alike, with the rest expressing no demonstrable sentiment.

Sentiment classification on tweets is a significantly different problem than performing similar analysis on movie or product reviews. In the latter cases, there are several problems that arise in a long-format review in determining sentiment. However, Twitter offers an entirely different challenge, which is largely created by the nonstandard and diverse language employed by Twitter users.

We implemented two Naive Bayes unigram models, a Naive Bayes bigram model and a Maximum Entropy model to classify tweets. We found that our Naive Bayes classifiers worked much better than our Maximum Entropy model could. Our results were quite good, and both models performed better than known performances on movie review classification, for example. However, our best accuracy still leaves room for improvement, which could come in the form of better preprocessing and more clever feature selection.

In addition, we could take the time to collect much more data (a factor of 10 times more, perhaps), making sure to implement the proper preprocessing techniques to ensure that overfitting doesn't occur in the widely varying class of language that can occur on Twitter.

The explosive growth of Twitter has attracted considerable media and consumer attention to this service. In terms of long-term development, the ultimate utility of our sentiment analyzer is to build a classifier adept enough to mine the Twitter database given a certain keyword, and return the overall site-wide sentiment associated with the provided keyword. As Twitter attracts more and more users, the accuracy and utility of such an application will only increase, since an ever-increasing volume of opinions and ideas will spread across the site. The usefulness of such a sentiment analyzer would provide an unprecedented level of analytics for companies and politicians alike, as well as provide an interesting source of experimentation for regular users.

10 Sources

Bo Pang and Lillian Lee. "A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts." 2004. *Proceedings of ACL*, pp. 271–278.

Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. "Thumbs up? Sentiment Classification using Machine Learning Techniques." 2002. *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 79–86.