**Twitter Sentiment Analysis**

**Nishad Trivedi**

{triv0025@umn.edu}

LING 5801, Spring 2014

University of Minnesota, Twin Cities

**1 Introduction**

Social media has led to an explosion of varied kinds of information. This is due to the nature of microblogs on which people are able to post real time messages about their opinions and feelings on a variety of topics, discuss current issues, complain, and express positive sentiments on their daily life. It is no wonder then that many companies have started polling these microblogs as part of their research to get a general sentiment for a product they may be developing or marketing. As for most operations in the modern age, it is the task of many scientists and engineers to replace manual labor with machine labor. This is done to improve speed and efficiency. One challenge then, in the field of polling microblogs, is to develop technology that can detect and summarize an overall sentiment. This process is called *sentiment analysis*.

In this paper, I will look at one such popular microblogging website called Twitter and build a model for classifying “tweets” into positive, negative, and neutral sentiment. I will build a model that can classify sentiment into positive and negative classes and then create a task for classifying a tweet into positive, negative, or neutral sentiment. The model will consist of a naïve Bayes classifier as well as a few other probability theorems that will be discussed further later. Additionally, I will use the Natural Language Toolkit 3.0 (NLTK), a library for Python (the language of choice) to aid in the classification process.

**2 Importance**

Before I begin to explain technical details that went into this project, let’s discuss why sentiment analysis is important. As discussed before, many companies use social media as a medium to communicate with consumers. These entities include corporate businesses, celebrities (musicians, actors, writers, etc.), and politicians among others. Politics is one of the biggest arenas that can benefit from sentiment analysis. Such analysis is especially useful when new bills are passed or presidential debates are held in order to see what the popular opinion is. With this valuable data, theoretically, politicians could review their platform and adapt in a way that could get them a better popular opinion. Sentiment analysis is also useful for movie producers. It lets them plan for the future; for example, if a movie is successful and many people are talking about it, executives can increase sales by more marketing and investing in close fields (books, video games, toys, etc.), or take opposite action if the movie reaction is unfavorable.

Finally, I will explain why Twitter was specifically used in contrast to other social media websites like Facebook, Instagram, Pinterest, etc. Twitter is unique in that the data available is open. Although there is a social graph and friend/follower connections, much of the data is open to sift through by anyone, which is why Twitter is often used for polling. It’s easy to query tweets by certain keywords. In contrast, Facebook is a closed-network where most of the data can only be valuable and extractable among a group of friends. Ultimately, since we are doing sentiment analysis, we need to parse text. This is why Instagram and Pinterest is not preferred because data from those sites are in the form of pictures which with very little words—making it difficult to analyze.

**3 Challenges**

Although Twitter is preferred over Facebook and Pinterest, it still has its own challenges. In a survey conducted in May 2013, the Pew Internet & American Life Project found 30% of Twitter users to fall in the 18-29 age group, a figure that has risen since November 2010[[1]](#footnote-1). We can expect the rhetoric to be used by younger populations to be vastly different than that by older populations. We can assume that a lot of slang will be used. Additionally, a large challenge of Twitter is its length constraint of 140 characters. Therefore, we can also assume that many words or sentences will be too long and will have to be shortened. It is due to the length constraint that we may suffer from a large data loss as we cannot successfully predict every single abbreviated or shortened word correctly.

**4 Classification Model**

The main algorithm used to detect sentiment is known as the naïve Bayes classification model. This is a simple probabilistic classifier based on applying Bayes’ theorem with strong (naïve) independence assumptions. In simple terms, a naïve Bayes classifier assumes that the value of a particular feature is unrelated to the presence (or absence) or any other feature. For example, a cat may be considered a feline if it has a tail, claws, fur, and omits a certain cry. A naïve Bayes classifier considers each of these features to contribute independently to the probability that a cat is a feline, regardless of the presence or absence of any other features.

Abstractly, the probability model for a classifier is the conditional model

(C \vert F_1,\dots,F_n)\,

Using Bayes’ theorem, this can be written as

(C \vert F_1,\dots,F_n) = \frac{p(C) \ p(F_1,\dots,F_n\vert C)}{p(F_1,\dots,F_n)}. \,

Using Bayesian Probability terminology, the above is the equivalent of

mbox{posterior} = \frac{\mbox{prior} \times \mbox{likelihood}}{\mbox{evidence}}. \,

There are a few other types of prediction models. They include decision trees, support vector machines, clusters, etc. Out of all of them, it seems that the naïve Bayes classifier is the simplest. However, in spite of its apparent simplicity, naïve Bayes classifiers have performed well in many situations—particularly Twitter, our domain of interest. In a research paper published from the University of Minnesota, it was found that the naïve Bayes classifier was performed the best in detecting and classifying tweets in a data set compared to various other algorithms[1]. They are easy to implement because they require a small amount of training data to estimate the necessary parameters. NLTK has an already-implemented version of the classifier. However, I quickly found out that its training algorithm is very slow. This was not feasible for the project, as the classifier would need to be trained everytime the program was run (although I did find a slick solution with pickling the binary data, thus preventing constant retraining). Therefore, it was beneficial to implement my own naïve Bayes algorithm. As said before, since the algorithm is not very complex, writing the algorithm was not very time-consuming and was a good idea because I could make adjustments to the code to obtain better results—a liberty I would not have had, ha[[2]](#footnote-2)d I used NLTK’s or any other library’s own classifier. Through my research in the domain of naïve Bayes models, I found that there were multiple types of classifiers. The only difference between them, however, was the assumptions made in regarding the distribution of the probability models. The specific classifier I chose implemented the naïve Bayes training and classification algorithms for data that was distributed according to multivariate Bernoulli distributions. In the data that was collected, there could be multiple features, but each one is assumed to be binary-valued Bernoulli Boolean variable. Therefore, in the case of text classification, a Bernoulli model uses word occurrence vectors

**5 Training**

The naïve Bayes classifier uses a supervised learning-based approach to train itself. This is a inference-based approach where it identifies and labels words based on already labeled training data. Therefore, in order to train the classifier, I had to extract tweets using the Tweepy API and manually classify its sentiment. I gathered approximately a two thousand tweets and classified each manually as either positive or negative. I wrote various scripts in order to make this process a bit faster. Overall, this process took a few hours spread over a few days. Additionally, I compiled a list of feature words—words that clearly displayed a positive or negative sentiment. Every time the program is run, the classifier is trained by going through the classified tweets and modifying the feature list vector by giving a probability score to each word. If the word is not found in the feature list, then it is simply ignored. The feature words are also added to both the positive and negative sentiment vectors. Their score depends on the sentiment and occurrences in the already classified training set. For example, the word “happy” would get a lower score in the negative sentiment vector compared to the positive sentiment vector. A sentiment vector essentially consists of the sentient as the key and the value as a dictionary of word-frequency pairs.

Since the goal was to classify tweets, there needed to be some kind of filtering mechanism to remove words that were not helpful in classification. Some of these words were specific to Twitter such as “RT” (retweets), usernames, and hashtags. Other words to filter out are what are called stopwords—pronouns, articles, prepositions, etc. These words are very common in the English language, but don’t convey any sentiment. Therefore, it is safer to disregard these words as they may corrupt our data. This sanitization method was used on the training data as well as any tweet we wanted to classify. Additionally, since Twitter is a form of social media, I expected there to be a lot of online abbreviations to be used. Therefore, I created a small dictionary mapping common abbreviations to their actual meaning, i.e. “lol” -> “laughing”, “omg” -> “surprised”, etc.

**6 Classification**

The classification method was simple. Any text that is read in goes through the sanitization described previously. The remaining set of tokens is then compared to the words in the positive sentiment vector as well as the negative sentiment vectors. The probability function used is the posterior equation described above where each word is given a certain score. Ultimately, the text is given a score from the positive vector and another from the negative vector. If negation words such as “don’t”, “despite”, “not”, etc. were encountered, the score least likely score is turned positive in order to guarantee that it is picked.

For example, the tweet, “I’m not having a good day today” would be sanitized to {“not“, “good”, “day”, “today”}. The only word in that set that conveys a sentiment is “good”, while all others would be classified as neutral making this sentence positive. However, the “not” negates the “good” in this sentence and so the negative sentiment score is turned positive, so it is higher than its opposing score. The program will then classify this tweet as negative as it should.

The following threshold function was used to decide whether the sentiment conveyed by the text was positive, negative, or neutral.

Positive if pos\_score > log(1.3) + neg\_score

Negative if neg\_score > log(0.9) + pos\_score

Neutral otherwise

Neutrality was used as a last resort because the classifier was never trained on neutral tweets. Therefore, it has a much lower accuracy in detecting neutrality and appropriately classifying it. Due to the low accuracy of successfully being able to identify a tweet as neutral, it is more appropriate to think of neutrality as though there was not enough information to classify the tweet, and thus, it is inconclusive. The numbers inputted to the logarithm functions were based on trial-and-error. It was found that these numbers gave the highest recall, precision, and F-score. The reason positive classification has a higher threshold is because positive sentiment is initially given a higher prior score to begin with. Therefore, there is a slight bias. This was done because it was harder accuracy with negative sentiment tweets was slightly lower with the data I gathered.

**7 Results**

Based on a Twitter dataset I acquired (separate from the training set), I was able to analyze how accurate my program was. The classifier was able to correctly identify 85% of the positive tweets and 79% of the negative tweets.

To use statistical terms, *precision* is defined as the fraction of retrieved documents that are relevant to the findings and *recall* is defined ass the fraction of all relevant documents that is returned by the search. The classifier had a precision of .90 and recall of .89. Using the recall and precision, we found the F-measure to be .90. An F-measure is essentially the measure of a test’s accuracy by considering both the precision and recall. These numbers show that the classifier was well trained and performed quite well. However, it would’ve been better had I used more data to evaluate my classifier. Additionally, there was some ambiguity as to ho the words were being defined. For example, querying “finals” on Twitter gave me tweets regarding exams that were occurring at the moment as well as playoff games in various sports. Although the sentiment analyzer may still predict the correct sentiment, it may be extracting tweets that are not relevant to the query. However, this is more of a problem on the Twitter client side rather than the classifier.

**8 Flaws**

While the results of the classifier show that project was successful to some degree, there are a few flaws with this program. In it current form, the classifier is not able to detect sarcasm. Having read a few research papers on the topic of sarcasm detection, there does not seem to be great success in detection although one French company claims to have built a program that is able to detect sarcasm with 80% accuracy[[3]](#footnote-3). Research shows that sarcasm detection in sites like Twitter where there is a length constraint is difficult mainly because there is little to no context. Only a human would be able to tell by vocal inflections and prevailing cultural attitudes that a sentence is sarcastic while a computer program only has text to go by[[4]](#footnote-4). Additionally, using a database like WordNet would have helped in word sense disambiguity, so we get precise words. Finally, besides getting better data (diverse and more quantity), I should have filtered less stopwords. Some of the words that were filtered out may have actually helped in detecting the sentiment.

**9 Conclusion**

With the rise of microblogging websites such as Twitter, it is valuable to poll data on attitudes of various things. Sentiment analysis hopes to mine this data and predict what opinions and emotions are being conveyed. The naïve Bayes classifier had a good prediction model, but lacked in the sense of being deprived of certain data due to Twitter’s length constraint, unable to detect sarcasm, and word ambiguity/slang. In future work, I will explore richer linguistic analysis such as semantic analysis and perhaps make the classifier adapt to the data it accumulates to improve its accuracy. Currently it is open sourced and maintained on GitHub in case others would like to improve upon the code I’ve built[[5]](#footnote-5).

1. eMarketer, “Twitter Use Rises Across US Age Groups”, 9 August 2013 [↑](#footnote-ref-1)
2. A. A. Taheri, M. Tepper, A. Banerjee, and G. Sapiro, “If You are Happy and Know It ... Tweet,” Technical Report

   TR-12-017, Department of Computer Science & Engineering, University of Minnesota, Twin Cities, 2012. [↑](#footnote-ref-2)
3. Zoe Kleinman, “Authorities 'use analytics tool that recognises sarcasm”, BBC News Technology, 2 July 2013. [↑](#footnote-ref-3)
4. Roberto González-Ibáñez, Smaranda Muresan , and Nina Wacholder, “Identifying Sarcasm in Twitter: A Closer Look “, Technical Report

   School of Communication & Information, Rutgers, The State University of New Jersey, 2011. [↑](#footnote-ref-4)
5. Sentapy – Sentiment Analysis in Python, http://www.github.com/trivedi/sentapy [↑](#footnote-ref-5)