Alec Go (alecmgo@stanford.edu)
Lei Huang (leirocky@stanford.edu)
Richa Bhayani (richab86@stanford.edu)
CS224N - Final Project Report
June 6, 2009, 5:00PM (3 Late Days)

Twitter Sentiment Analysis

Introduction

Twitter is a popular microblogging service where users create status messages (called "tweets"). These tweets sometimes express opinions about different topics.

The purpose of this project is to build an algorithm that can accurately classify Twitter messages as positive or negative, with respect to a query term. Our hypothesis is that we can obtain high accuracy on classifying sentiment in Twitter messages using machine learning techniques.

Generally, this type of sentiment analysis is useful for consumers who are trying to research a product or service, or marketers researching public opinion of their company.

Defining Sentiment

For the purposes of our research, we define sentiment to be "a personal positive or negative feeling." Here are some examples:

Sentiment	Query	Tweet
Positive	jquery	dcostalis: Jquery is my new best friend.
Neutral	San Francisco	schuyler: just landed at San Francisco
Negative	exam	jvici0us: History exam studying ugh.

For tweets that were not clearcut, we use the following litmus test: If the tweet could ever appear as a newspaper headline or as a sentence in Wikipedia, then it belongs in the neutral class. For example, the following tweet would be marked as neutral because it is fact from a newspaper headline, even though it projects an overall negative feeling about GM:

ThomasQuinlin: RT @Finance_Info Bankruptcy filing could put GM on road to profits (AP) http://cli.gs/9ua6Sb #Finance

Related Work

There have been many papers written on sentiment analysis for the domain of blogs and product reviews. (Pang and Lee 2008) gives a survey of sentiment analysis. Researchers have also analyzed the brand impact of microblogging (Jansen). We could not find any papers that analyzes machine learning techniques in the specific domain of microblogs, probably because the popularity of Twitter is very recent.

Overall, text classification using machine learning is a well studied field (Manning and Schuetze 1999). (Pang and Lee 2002) researched the effects of various machine learning techniques (Naive Bayes (NB), Maximum Entropy (ME), and Support Vector Machines (SVM) in the specific domain of movie reviews. They were able to achieve an accuracy of 82.9% using SVM and a unigram model.

Researchers have also worked on detecting sentiment in text. (Turney 2002) presents a simple algorithm, called semantic orientation, for detecting sentiment. (Pang and Lee 2004) present a hierarchical scheme in which text is first classified as containing sentiment, and then classified as positive or negative.

Work (Read, 2005) has been done in using emoticons as labels for positive and sentiment. This is very relevant to Twitter because many users have emoticons in their tweets.

Twitter messages have many unique attributes, which differentiates our research from previous research:
1. Length. The maximum length of a Twitter message is 140 characters. From our training set, we calculated that the average length of a tweet is 14 words, and the average length of a sentence is 78 characters. This is very different from the domains of previous research, which was mostly focused on reviews which consisted of multiple sentences.
2. Available data. Another difference is the sheer magnitude of data. In (Pang and Lee 2002), the corpus size 2053. With the Twitter API, it is much easier to collect millions of tweets for training.
3. Language model. Twitter users post messages from many different mediums, including their cell phones. The frequency of misspellings and slang in tweets is much higher than other domains.

Procedure

Data Collection

There are not any existing data sets of Twitter sentiment messages. We collected our own set of data. For the training data, we collected messages that contained the emoticons :) and :(via the Twitter API.

The test data was manually. A set of 75 negative tweets and 108 positive tweets were manually marked. A web interface tool was built to aid in the manual classification task.

See Appendix A for more details about the data.

Classifiers

Several different classifiers were used. A Naive Bayes classifier was built from scratch. Third-party libraries were used for Maximum Entropy and Support Vector Machines. The following table summarizes the results.

Accuracy	Keyword	Multinomial Naive Bayes Unigram	Multinomial Naive Bayes Unigram using MI	MaxEnt OpenNlp	MaxEnt Stanford Classifier (Sigma= 10)
Unigram	0.42622950819672	0.80874316939891	0.84153005464481	79.23	0.61
Unigram Limited with feature threshold = 100	-	0.80874316939891	-	-	0.63
Unigram + POS	-	0.74863387978142	0.78142076502732	-	0.683
Bigram	-	0.75956284153005	0.73770491803279	0.7322	0.667

Table 1. Accuracy results from various classifiers

Training size also has an effect on performance. Figure 1 shows the effect of training size on accuracy.

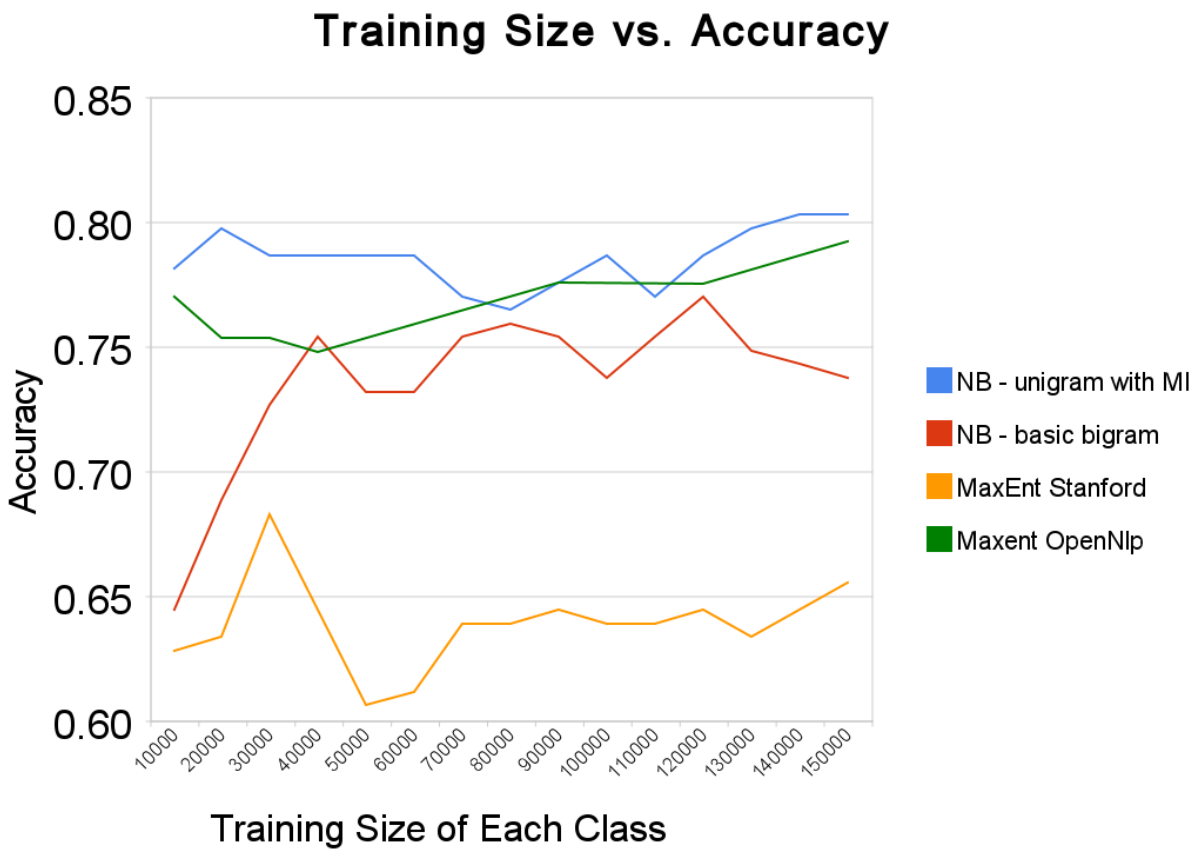

Figure 1. Effect of training size on different classifiers.

Naive Bayes

Naive Bayes is a simple model for classification. It is simple and works well on text categoration. We adopt multinomial Naive Bayes in our project. It assumes each feature is conditional independent to other features given the class. That is,

$$P(c \mid t) = \frac{P(c)P(t \mid c)}{p(t)}$$

where c is a specific class and t is text we want to classify. P(c) and P(t) is the prior probabilities of this class and this text. And P(t | c) is the probability the text appears given

this class. In our case, the value of class c might be POSITIVE or NEGATIVE, and t is just a sentence.

The goal is choosing value of c to maximize P(c | t):

Where $P(w_i \mid c)$ is the probability of the ith feature in text t appears given class c. We need to train parameters of P(c) and $P(w_i \mid c)$. It is simple for getting these parameters in Naive Bayes model. They are just maximum likelihood estimation (MLE) of each one. When making prediction to a new sentence t, we calculate the log likelihood $\log P(c) + \sum_i \log P(w_i|c)$ of different classes, and take the class with highest log likelihood as prediction.

In practice, it needs smoothing to avoid zero probabilities. Otherwise, the likelihood will be 0 if there is an unseen word when it making prediction. We simply use add-1 smoothing in our project and it works well.

Feature selection

For unigram feature, there are usually 260,000 different features. This is a very large number. It makes model higher variance. (Since more complicated model has higher variance). So it will need much more training data to avoid overfitting. Our training set contains hundreds of thousands sentences. But it is still a large number of features for our training set. It is helpful if we discard some useless features. We try 3 different feature selection algorithms.

Frequency-based feature selection

This is the simplest way to do feature selection. We just pick features (unigram words in our case) for each class with high frequency occurrence in this class. In practice, if the number of occurrences of a feature is larger than some threshold (3 or 100 in our experiments), this feature is a good one for that class. As we seen in the result table, this simply algorithm increases about 0.03 of accuracy.

Mutual Information

The idea of mutual information is, for each class C and each feature F, there is a score to measure how much F could contribute to making correct decision on class C. The formula of MI score is,

$$\mathrm{MI}(\mathrm{C};\ \mathrm{F}) = \sum_{\mathrm{ef} \in \{1,0\}} \sum_{\mathrm{ec} \in \{1,0\}} \mathrm{P}(\mathrm{C} = \mathrm{ec}, \mathrm{F} = \mathrm{ef}) \log \frac{\mathrm{P}(\mathrm{C} = \mathrm{ec}, \mathrm{F} = \mathrm{ef})}{\mathrm{P}(\mathrm{C} = \mathrm{ec})\mathrm{P}(\mathrm{F} = \mathrm{ef})}$$

In practice, we also use add-1 smoothing for each Count(C = ec, F = ef) to avoid divided by zero. The code is below.

```
double n = polarityAndFeatureCount.totalCount() + 4;
for(String feature : featureCount.keySet()) {
  for(int polarity : polarityCount.keySet()) {
    double n11 = polarityAndFeatureCount.getCount(polarity, feature) + 1;
    double n01 = polarityCount.getCount(polarity) -
```

```
                polarityAndFeatureCount.getCount(polarity, feature) + 1;
    double n10 = featureCount.getCount(feature) -
                polarityAndFeatureCount.getCount(polarity, feature) + 1;
    double n00 = n - (n11 + n01 + n10);

    double n1dot = n11 + n10;
    double n0dot = n - n1dot;
    double ndot1 = n11 + n01;
    double ndot0 = n - ndot1;

    double miScore = (n11 / n) * Math.log((n * n11) / (n1dot * ndot1))
                   + (n01 / n) * Math.log((n * n01) / (n0dot * ndot1))
                   + (n10 / n) * Math.log((n * n10) / (n1dot * ndot0))
                   + (n00 / n) * Math.log((n * n00) / (n0dot * ndot0));

    mi.setCount(polarity, feature, miScore);
  }
}
```

After calculating MI score, only top k features with highest scores will be picked for feature set to test. We can see that if k is small, the model is too simple that data is underfitting. But if k is large, the model is too complicated that data is overfitting. The best number of features in our unigram case is about 40,000. As k grow up to 20,000, the accuracy and F score are also grow up quickly. This is because in this area, the model is high bias. So it is helpful to add features to avoid underfitting data. When the number is larger than 100,000, the accuracy and F score decrease gradually. Since the large number of features makes model so complicated that there are not enough training sentence to avoid overfitting.

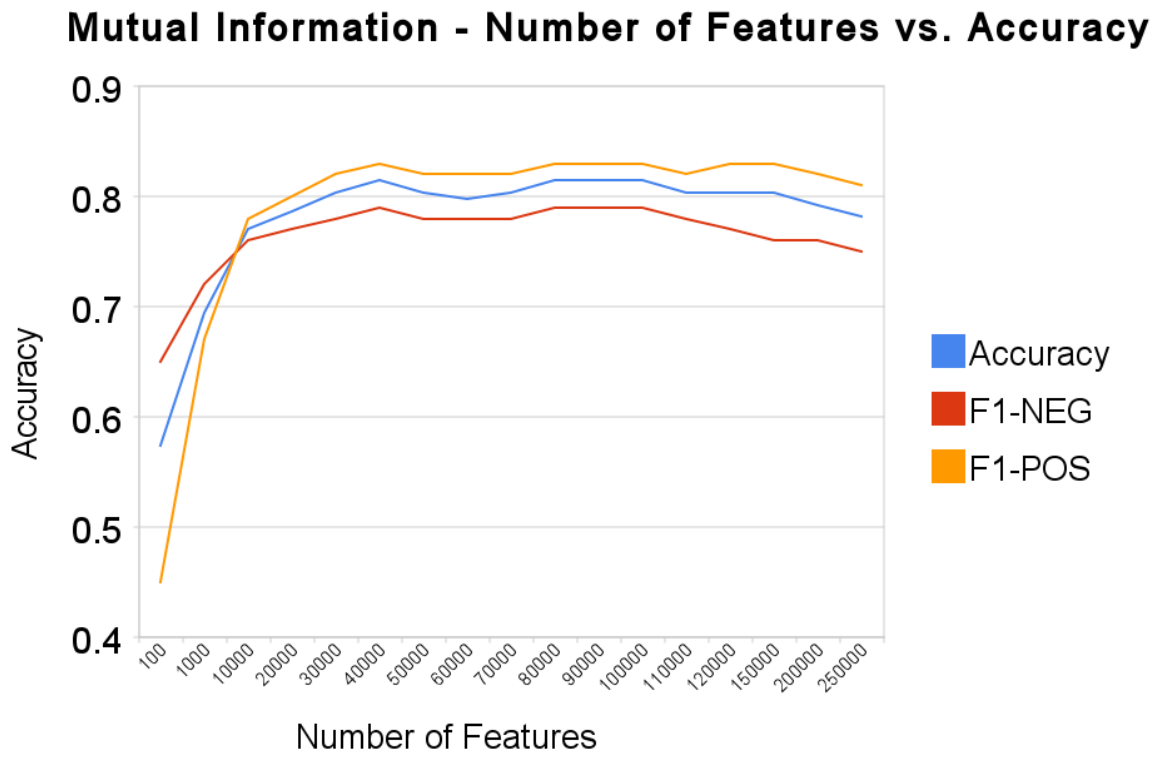

Figure 2 - Mutual Information - Number of Features vs. Accuracy

X^2 Feature selection

The idea of X^2 Feature selection is similar as mutual information. For each feature and class, there is also a score to measure if the feature and the class are independent to each other. It uses X^2 test, which is a statistic method to check if two events are independent. It assumes the feature and class are independent and calculates X^2 value. The large score implies they are not independent. For example, the critical value of 0.001 is 10.83. This means, if they are independent to each other, then the probability this score larger than 10.83 is only 0.001. Alternatively, if the score is larger than 10.83, then it is unlikely the feature and the class independent. The larger the score is, the higher dependency they have. So we want keep features for each classes with highest X^2 scores. The formula of X^2 score is,

$$\chi^2(F,C) = \frac{N(N_{11}N_{00} - N_{10}N_{01})^2}{(N_{11} + N_{01})(N_{11} + N_{10})(N_{10} + N_{00})(N_{01} + N_{00})}$$

where N is the total number of training sentences. N_{11} is the number the co-occurrence of the feature F and the class C. N_{10} is the number of sentences contains the feature F but is not in class C. N_{01} is the number of sentences in class C but doesn't contain feature F. N_{00} is the number of sentences not in C and doesn't contain feature F. We implement it similar as mutual information, except using this different formula.

The performance of X^2 Feature selection is very similar as mutual information in our project. Both these two method could increase both accuracy and F score by 0.05.

Basic unigram feature selection for NB	Presence	Frequency-based(3)	Frequency-base(Presence, 100)	MI (Presence, 40000)	Chi2 (Presence, 40000)
Accuracy	0.7650273224043	0.78142076502732	0.79781420765027	0.81420765027322	0.80327868852459
F1-NEG	0.73	0.75	0.79	0.79	0.78
F1-POS	0.79	0.81	0.81	0.83	0.83

Table 2. Results from various feature selection tests

Maximum Entropy

The idea behind MaxEnt classifiers is that we should prefer the most uniform models that satisfy any given constraint. MaxEnt models are feature based models. We use these features to find a distribution over the different classes using logistic regression. The probability of a particular data point belonging to a particular class is calculated as follows:

$$p(c \mid d, \vec{\lambda}) = \frac{\exp\left[\sum_i \lambda_i f_i(c, d)\right]}{\sum_{c'} \exp\left[\sum_i \lambda_i f_i(c', d)\right]}$$

Where, c is the class, d is the data point we are looking at, and λ is a weight vector.

MaxEnt makes no independence assumptions for its features, unlike Naïve Bayes. This means we can add features like bigrams and phrases to MaxEnt without worrying about feature overlapping.

We tried using two packages for the MaxEnt implementation: the Stanford Classifier and the OpenNLP package.

Performance

The Stanford Classifier package gave bad results for the default parameter settings. Over different training sizes (Figure 1) it did improve a bit, but was a lot worse than the other classifiers. We changed the smoothing constants, but it never got very close to the NB classifier in terms of accuracy. As shown in Figure 3, different sigma (smoothing) values did not contribute much to higher accuracy.

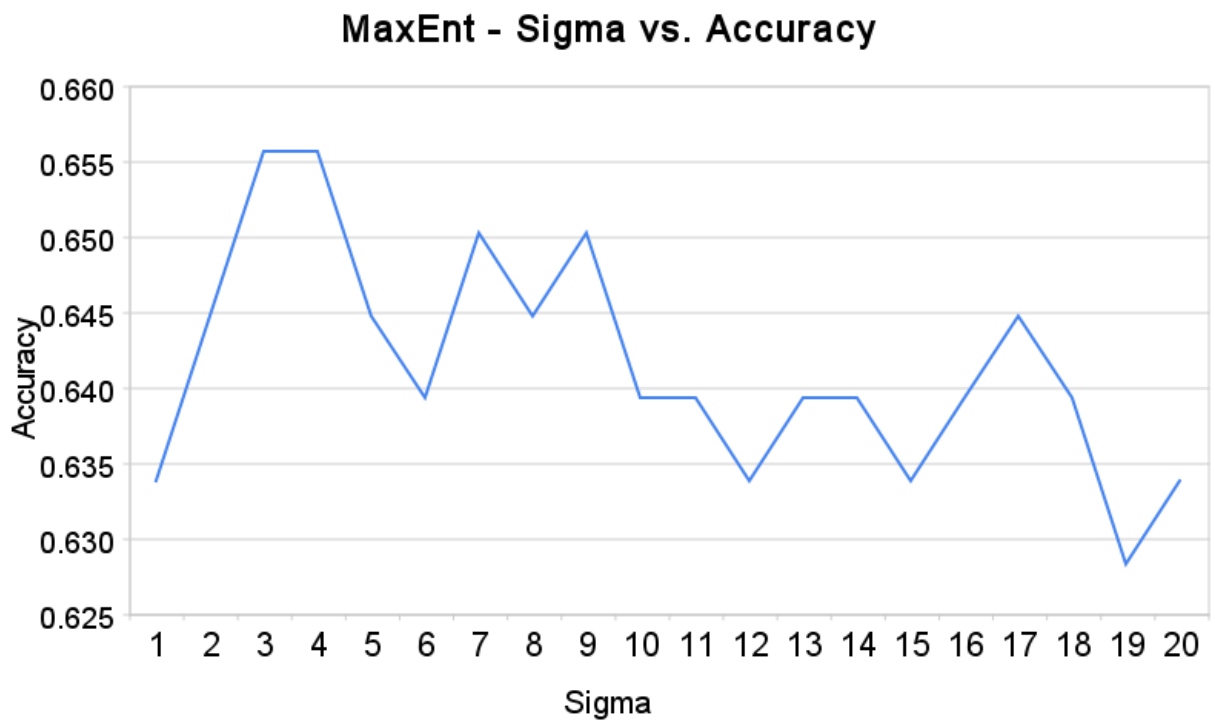

Figure 3. Sigma (the smoothing parameter) vs accuracy

After testing for different smoothing values and trying different functions in place of ConjugateDescent, we decided to try OpenNLP's MaxEnt classifier since time was running short.

MaxEnt from OpenNLP did perform considerably better. As one can see from Figure 1, MaxEnt performs similar to how the NB performs. Since it doesn't significantly improve performance and takes very long to train and test, we decided to pursue NB for some other experiments.

Support Vector Machines

Support Vector Machines were also explored using Weka software. We tested SVM with a unigram feature extractor, and achieved only 73.913% accuracy. We used a linear kernel. SVMs have many parameters. We believe that performance can be improved here by trying different parameters and kernels.

Feature Extractors

1. Unigram

Building the unigram model took special care because the Twitter language model is very different from other domains from past research. The unigram feature extractor addressed the following issues:

a. Tweets contain very casual language. For example, you can search "hungry" with a random number of u's in the middle of the word on http://search.twitter.com to understand this. Here is an example sampling:
huuuungry: 17 results in the last day
huuuuuuungry: 4 results in the last day
huuuuuuuuuungry: 1 result in the last day

Besides showing that people are hungry, this emphasizes the casual nature of Twitter and the disregard for correct spelling.

b. Usage of links. Users very often include links in their tweets. An equivalence class was created for all URLs. That is, a URL like "http://tinyurl.com/cvvg9a" was converted to the symbol "URL."

c. Usernames. Users often include usernames in their tweets, in order to address messages to particular users. A de facto standard is to include the @ symbol before the username (e.g. @alecmgo). An equivalence class was made for all words that started with the @ symbol.

d. Removing the query term. Query terms were stripped out from Tweets, to avoid having the query term affect the classification.

2. Bigrams

The reason we experimented with bigrams was we wanted to smooth out instances like 'not good' or 'not bad'. When negation as an explicit feature didn't help, we thought of experimenting with bigrams.

However, they happened to be too sparse in the data and the overall accuracy dropped in the case of both NB and MaxEnt. Even collapsing the individual words to equivalence classes did not help.

Bigrams however happened to be a very sparse feature which can be seen in the outputs with a lot of probabilities reported as 0.5:0.5.

For context: @stellargirl I loooooooovvvvvveee my Kindle2. Not that the DX is cool, but the 2 is fantastic in its own right.
Positive[0.5000] Negative[0.5000]

3. Negate as a features

Using the Stanford Classifier and the base SVM classifiers we observed that identifying NEG class seemed to be tougher than the POS class, merely by looking at the precision, recall and F1 measures for these classes. This is why we decided to add NEGATE as a specific feature which is added when “not” or ‘n’t” are observed in the dataset. However we only observed a increase in overall accuracy in the order of 2% in the Stanford Classifier and when used in conjunction with some of the other features, it brought the overall accuracy down and so we removed it.

Overlapping features could get the NB accuracy down, so we were not very concerned about the drop with NB. However it didn't provide any drastic change with OpenNLP either.

4. Part of Speech (POS) features

We felt like POS tags would be a useful feature since how you made use of a particular word. For example, 'over' as a verb has a negative connotation whereas 'over' as the noun, would refer to the cricket over which by itself doesn't carry any negative or positive connotation. On the Stanford Classifier it did bring our accuracy up by almost 6%. The training required a few hours however and we observed that it only got the accuracy down in case of NB.

Handling the Neutral Class

In the previous sections, neutral sentiment was disregarded. The training and test data only had text with positive and negative sentiments.

In this section, we explore what happens when neutral sentiment is introduced.

Naive Bayes with Three Classes

We extended the Naive Bayes Classifier to handle 3 classes: positive, neutral, and negative. Collecting a large amount of neutral tweets is very challenging. For the training data, we simply considered any tweet without an emoticon to be part of the neutral class. This is obviously a very flawed assumption, but we wanted to see what the test results would be. For the test data, we manually classified 33 tweets as neutral.

The results were terrible. The classifier only obtained 40% accuracy. This is probably due to the noisy training data for the neutral class.

Subjective vs. Objective Classifier

Another way to handle the neutral class is to have a two phased approach:
1. Given a sentence, classify the sentence as objective or subjective.
2. If the sentence is subjective, classify it as positive or negative.

We modified our Naive Bayes classifier to handle a subjective class and a objective class. Unfortunately, the results were terrible again, with an accuracy of only 44.9%. Again, this is probably due to the noisy training data of the neutral class.

Error Analysis

Naive Bayes Error Analysis

Example 1

Naive Bayes's independence assumption sometimes causes havoc in classification. This is most notable for negative words like "not" that precede adjectives. Here is an example:

As u may have noticed, not too happy about the GM situation, nor AIG, Lehman, et al

The actual sentiment is negative, but Naive Bayes predicted positive. The unigram model has a probability on the word "happy" for the positive class, which doesn't take into account the negative word "not" before it.

Example 2

In some cases, our language model was simply not rich enough. For example, the Naive Bayes classifier failed on the following example:

Cheney and Bush are the real culprits - http://fwix.com/article/939496

The actual sentiment is negative, but the Naive Bayes classifier predicted positive. The reason is that the word "culprits" only occurred once in our training data, as a positive sentiment.

We thought stemming words may help because the word "culprit" appears in the training corpus: 1 time in the positive class and 4 times in the negative class. We tried the Porter Stemmer in the unigram feature extractor to help with this situation, but it ended up bringing down overall accuracy by 3%.

Example 3

The Naive Bayes classifier doesn't take the query term into account. For example:

Only one exam left, and i am so happy for it :D

With respect to the query term "exam", this sentence should be classified as negative because this implies that the user doesn't like exams. In the current Naive Bayes model, it's impossible to detect this.

Example 4

In the Naive Bayes classifier, there was a URL equivalence class. In our training data, URLs occur much more often in positive tweets than in negative tweets. This would very often throw off short sentences. For example:

obviously not siding with Cheney here: http://bit.ly/19j2d

In this sentence, the word "not" biased the sentence towards the negative class. But, URL occurs so often in the positive class that this tweet was classified incorrectly as positive.

MaxEnt Error Analysis

In this section, we use the encoding "0" to denote the negative class and "4" to denote the positive class.

Example 1

Maxent instance: Collapsing Query term vs not

For context: arg . QUERY_TERM is making me crazy .
4[0.3784] 0[0.6216]

Predicted: 0
Actual: 0

For context: Arg. Twitter API is making me crazy.
4[0.5046] 0[0.4954]

Predicted: 4
Actual: 0

We reasoned that the query term should not be taken into account while classifying a tweet and experimented with collapsing to an equivalent class QUERY_TERM. This offsets the negativity/ positivity associated with the query term while classifying the tweet.

Example 2
Maxent: Collapsing person vs not

For context: my exam went good. @HelloLeonie: your prayers worked (:
4[0.4690] 0[0.5310]

Predicted: 0
Actual: 4

For context: my QUERY_TERM went good . PERSON your prayers worked SMILE_10
4[0.6105] 0[0.3895]

Predicted: 4
Actual: 4

Exams have an inherent negative quality , and a sparse feature like person name may not compensate for the strong negativity attached with the word 'exam', in this case collapsing it to a generic term PERSON helped a lot more users to collapse to the same class, thus improving accuracy.

Example 3
For context: omg so bored & my tattoos are so QUERY_TERM ! ! help ! aha SMILE_7
4[0.9643] 0[0.0357]

Predicted: 4
Actual: 0.

The original tweet read:

omg so bored & my tattoooos are so itchy!! help! aha =)

There were two unknowns in this sentence :tattooos and &amp which affected the probabilities to a large extent.

Conclusion and Future Improvements

Machine learning techniques perform reasonably well for classifying sentiment in tweets. We had many more ideas for improving our accuracy, however. Below are list of improvements that can be made.

Semantics

Our algorithms classify the overall sentiment of a tweet. Depending on whose perspective you're seeing the tweet from the polarity may change.

Example: *Federer beats Nadal :)*

This tweet is positive for Federer and negative for Nadal. While this classification didn't pose a problem for us since our aim was only to classify the tweet overall, the polarity would depend on your query term. For this, we feel using more semantics would be needed. Using a sementic role labeler could tell you which noun is mainly associated with the verb and the classification would take place accordingly, so *Nadal beats Federer :)* should be classified differently from *Federer beats Nadal :)*.

Part of Speech (POS) tagger

The POS tagger took about 3 hours to train and hence we could not run too many tests on it. It did improve the accuracy in case of Maxent and could have been helpful to NB with some more variations but we didn't have enough time to conduct these tests.

Domain-specific tweets

Our classifiers produce around 85% accuracy for tweets across all domains. This means an extremely large vocabulary size. If limited to particular domains (such as movies) we feel our classifiers would perform even better.

Support Vector Machines

(Pang and Lee 2002) shows that SVM performed the best when classifying movie reviews as positive or negative. An important next step would be to further explore SVM parameters for classifying tweets.

Handling neutral tweets

In real world applications, neutral tweets cannot simply be ignored. Proper attention needs to be paid to neutral sentiment. There are some approaches that use a POS tagger to look at adjectives to determine if a tweet contains an sentiment.

Dealing with words like "not" appropriately

Negative words like "not" have the magical affect of reversing polarity. Our current classifier doesn't handle this very well.

Ensemble methods

A single classifier may not be the best approach. It would be interesting to see what the results are for combining different classifiers. For example, we thought about using a mixture model between unigrams and bigrams. More sophisticated ensemble methods, like boosting, could be employed.

Using cleaner training data.

Our training data does not have the cleanest labels. The emoticons serve as a noisy label. There are some cases in which the emoticon label would normally not make sense to a human evaluator. For example user ayakyl tweeted, *"agghhhh :) loooosing my mind!!!!"* If we remove the emoticon from this phrase, it becomes *"agghhhh loooosing my mind!!!!"* in which a human evaluator would normally assess as negative.

Contributions

This project was created from scratch for CS224N. No prior code existed before this class started (besides the third party libraries).

- Alec Go wrote tweet scraper, the framework for the classifier tester, the unigram feature extractor, the first version of Naive Bayes classifier, the SVM component, and the web application.
- Richa Bhayani wrote MaxEnt and POS
- Lei Huang wrote a better Naive Bayes classifier, with Mutual Information and X^2 feature selection.

References

B. Jansen, M. Zhang, K. Sobel, A. Chowdury. The Commerical Impact of Social Mediating Technologies: Micro-blogging as Online Word-of-Mouth Branding, 2009.

C. Manning and H. Schuetze. Foundations of Statistical Natural Language Processing. 1999.

B. Pang, L. Lee, S. Vaithyanathan. Thumbs up? Sentiment Classification using Machine Learning Techniques, 2002.

B. Pang and L. Lee. "Opinion Mining and Sentiment Analysis" in Foundations and Trends in Information Retrieval, 2008.

B. Pang and L. Lee. "A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts" in Proceedings of ACL, 2004.

J. Read. Using Emotions to Reduce Dependency in Machine Learning Techniques for Sentiment Classification, 2005.

P. Turney. "Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews" in Proceedings of the 40th Annual Meeting of the Association for Computatoinal Linguistics (ACL), 2002.

Appendix

A. Code Details

You can run the classifier test by calling ClassifierTester. It takes the following arguments:

```
com.twittersentiment.classifiers.ClassifierTester /
  -classifier com.twittersentiment.classifiers.NaiveBayesClassifier /
  -featureextractor com.twittersentiment.features.UnigramFeatureExtractor /
  -train1 smiley.txt /
  -train2 frowny.txt /
```

-test testdata.manual

The "classifier" argument specifies the type of classifier you want to use. The available classifiers are:
- KeywordClassifier: a simple classifier based on hand-picked keywords
- NaiveBayesClassifier: a simple Naive Bayes classifier
- NaiveBayesClassifierLimitedFeatures - Naive Bayes that only uses terms that appear more than 3 times
- NaiveBayesClassifierLimitedFeaturesChi2 - Naive Bayes that uses chi-squared for feature selection
- NaiveBayesClassifierLimitedFeaturesMI - Naive Bayes that uses Mutual Information for feature selection
- MaxentClasssifier: runs the Stanford Maxent classifier
- MEOpenNlp: runs the OpenNLP MaxEnt package

The "featureextractor" argument specifies the type of feature extractor you want to use. The available feature extractors are:
- UnigramFeatureExtractor - a simple unigram extractor
- BigramFeatureExtractor - a simple bigram extractor
- POSFeatureExtractor - runs the Stanford POStagger to extract features

The code has the following dependencies:
1. Stanford Classifier library:
http://nlp.stanford.edu/software/classifier.shtml
2. OpenNLP MaxEnt library:
http://maxent.sourceforge.net/index.html
3. Twitter4J is an external library for parsing tweets:
http://yusuke.homeip.net/twitter4j/en/index.html
4. Weka is a data mining library: http://www.cs.waikato.ac.nz/ml/weka/. Tip: Our data sets require a lot of memory. Set the JVM memory size in RunWeka.ini.

Test and training data can be downloaded from here:
http://www.stanford.edu/~alecmgo/cs224n/twitterdata.2009.05.25.c.zip

This dataset has the following:
1. Training Files:
smiley.txt.processed.date - tweets that have ":)"
frowny.txt.processed.date - tweets that have ":("

2. Test files:
testdata.manual.date - tweets manually classified
testdata.auto - crawled tweets that have :(or :) that are not part of training set
testdata.auto.noemoticon - the same data as testdata.auto, except emoticons stripped off

3. Weka files:
train.40000.date.arff - training ARFF file for Weka with 40000 tweets
testdata.manual.date.arff - test ARFF file for Weka
train.40000.date - the file that train.40000.date.arff was generated from

Data file format has 6 fields, separated by a double semicolon (;;). Here is an example:

4;;2087;;Sat May 16 23:58:44 UTC 2009;;lyx;;robotickilldozr;;Lyx is cool.

The fields are the following:
0 - the polarity of the tweet
1 - the id of the tweet
2 - the date of the tweet
3 - the query (if there is no query, then this value is NO_QUERY)
4 - the user that tweeted
5 - the text of the tweet

B. Web Application

We launched a prototype of our sentiment analyzer at http://twittersentiment.appspot.com on April 9, 2009, initially with the keyword classifier. The web application was picked up by a few websites:

1. April 29, 2009 - Programmable Web (http://www.programmableweb.com/) picked it as "Mashup of the Day."

2. May 16, 2009 - LiveMint (part of the WSJ) used Twitter Sentiment to track the 2009 Indian elections. Source: http://blogs.livemint.com/blogs/last_24_hours/archive/2009/05/16/twitter-sentiment-update-everyone-being-nice-to-the-nda-now.aspx

3. June 4, 2009 - *The Measurement Standard* reviewed various sentiment tools, including our web application. Unfortunately, they gave it a "Very limited usefulness" rating. Source: http://www.themeasurementstandard.com/issues/5-1-09/neartwittersentiment5-1-09.asp

Overall, we received 828 unique visitors between April 9, 2009 and June 5, 2009. Figure 4 shows traffic for the following 9 weeks.

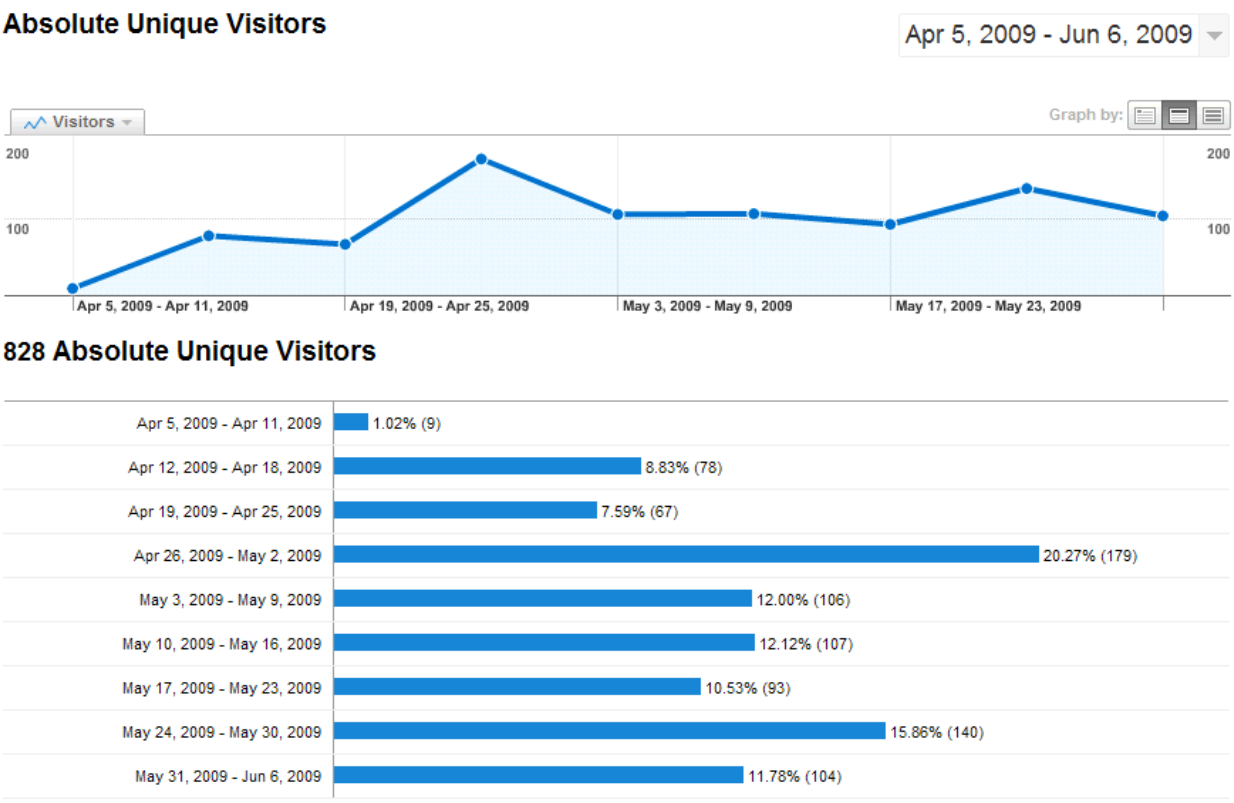

Figure 4. Traffic to http://twittersentiment.appspot.com from April 9 to June 5, 2009.