Sentiment Analysis of Twitter Data

Dean D'souza, Student, Harrisburg University of Science and Technology, dpdsouza@my.harrisburgu.edu

Lecturer: Majid Shaalan, PhD., Associate Professor of Computer Science, Harrisburg University of Science and Technology, MShaalan@Harrisburgu.edu

Assistant Lecturer: Raja Shanmugavelan, Lecturer of Computer Information Systems, Harrisburg University of Science and Technology, RShanmugavelan@Harrisburgu.edu

Abstract:

Sentiment analysis is a process which makes use of various methods such as Natural Language Processing, Text Analysis, etc. to find and extract information from textual data. [1] It can be applied to various kinds of textual data, out of which a popular source is from twitter, a social networking website. The project aims at replicating and possibly improving on certain steps in analyzing twitter data through the python package of Natural Language Tool Kit (NLTK), while testing a few ideas on how to improve the features for the Sentiment Classifier built.

Keywords: Sentiment Analysis, Natural Language Processing, Text Analysis, Twitter, NLTK

Introduction:

Sentiment Analysis (also known as opinion mining) is an important field in Computer Science which focuses on obtaining information from text data, usually in the form of attitude of the author with regards to a topic or the overall polarity of the document. [1] It involves a variety techniques to achieve this goal, which usually involves Natural Language Processing, Text Analysis, and Computational Linguistics.

Natural Language Processing (or NLP) is another important field in Computer Science (related to Human Computer Interaction) which focuses on how computers and humans interact through human language. NLP generally utilizes Machine Learning Techniques and systems built in such a fashion have the advantages of having automatic learning procedures which focuses on the most common cases, makes use of statistical inference algorithms, and having the ability of improving accuracy through the addition of more input data. [2]

NLP itself consists of several tasks, one of which is sentiment analysis of text data, and involves some of the following steps:

1. **Tokenization**: which is the process of breaking down a collection of characters or a string into smaller parts known as tokens. In terms of breaking down textual data, this would involve breaking down the data into individual words, punctuations, and possible numerical data.
2. **Chunking**: which involves breaking down a collection of strings into individual strings. In more general terms this can be thought of as breaking down paragraphs into individual sentences, or even breaking down long sentences into individual units.
3. **Stemming**: which involves converting word tokens into their base form. An easier way to understand this would be with the example of the word "looking", which is one of the many forms of the word "look". Obtaining this base form of "look" is known as stemming.

4. **Part-of-Speech tagging**: which is an important task on its own and involves correctly identifying what part of speech each word token belongs to. An example of this would be the word token "book" which could be a noun ("The book on the shelf") or a verb ("I booked a flight to Las Vegas").

Many of the above-mentioned tasks can be easily performed in python using the Natural Language Tool Kit (NLTK), which provides many pre-packaged functions and data structures. NLTK in one of the most popular and easy to use toolkits for performing Sentiment Analysis through python. Some of the important tools that are used in this project include the Naïve Bayes Classifier and the Decision Tree Classifier algorithms.

The project attempts to make subtle improvements on the tasks of tokenization and part-of-speech tagging (over the default tools provided in NLTK) to obtain required features for building a model which can automatically tag tweets (twitter text or posts) as positive or negative.

Data:

The original dataset was obtained from the Sentiment140 website [3] and consists of around 1,600,000 records (tweets) collected over a period of a few days. The original authors utilized the Twitter Search API to collect tweets by using a keyword search and then processed the data by matching emoticons to determine the polarity of the tweet, i.e., tweets with emoticons like '☺' were labelled as positive while those with emoticons like '☹' were labelled as negative. The data file which is available for download is in the form of a '.csv' file and has the following fields [3]:

1. The polarity of the tweet (0 = negative, 2 = neutral, 4 = positive)
2. The id of the tweet (2087)
3. The date of the tweet (Sat May 16 23:58:44 UTC 2009)
4. The Query. If there is no query, then this value is NO_QUERY.
5. The user that tweeted (eg. robotickilldozr)
6. The text of the tweet (eg. Lyx is cool)

Due to limited memory, which prevented loading of the entire data, a sample of the original was taken for the project. This sample consisted of a total of 7200 records, out of which 3600 were negative (or 0) and 3600 were positive (or 4). As the entire data set could not be loaded, records having a sentiment of '2' or neutral could not be found and were left out from this analysis. However, some previous work done estimates that at least 600 records of each type should be present for training the models and hence the current sample is sufficient. [5]

To process the data efficiently, a few fields had to be dropped, namely:

1. **Tweet ID**: this field was dropped as it is a unique number which identified each tweet and did not contribute much towards estimating the tweet polarity.
2. **Date of the Tweet**: while this field may have contributed towards the polarity of the tweet, the sample extracted was within a timespan of less than a day (between late Monday and early Tuesday) and hence had no significant effect on tweet polarity.
3. **The Query**: several records showed the value NO_QUERY, especially in the sample. Hence, this field was removed as it would not contribute much to the classification task under the circumstances.

4. **The User / Author**: the sample contained a variety of authors, and for the most part, each record displayed unique authors. Hence, it was removed.

The variables or fields left were the **Tweet Polarity** and the actual **Contents** of the tweet, which under most circumstances is more than enough for classification of twitter data.

The characteristics of the sample tweets can be best seen through the following statistics:

```
Number of Words : 100966
Number of Unique Words : 20472
Lexical Diversity : 4.93190699492
```

Figure 1 Lexical Diversity with stopwords included

```
Number of Words : 68416
Number of Unique Words : 18123
Lexical Diversity : 3.77509242399
```

Figure 2 Lexical Diversity without stopwords

As we can see from the above screen shots, the lexical diversity, which is a measure of the range of the vocabulary, is 4.93 with stopwords included and 3.77 when we exclude stopwords, which is on the lower end of what would be considered good for Sentiment Analysis. **Stopwords** are words considered to be not important to the task of analysis and can be removed to improve performance.

We now try to see which tokens or word types are used the most often.

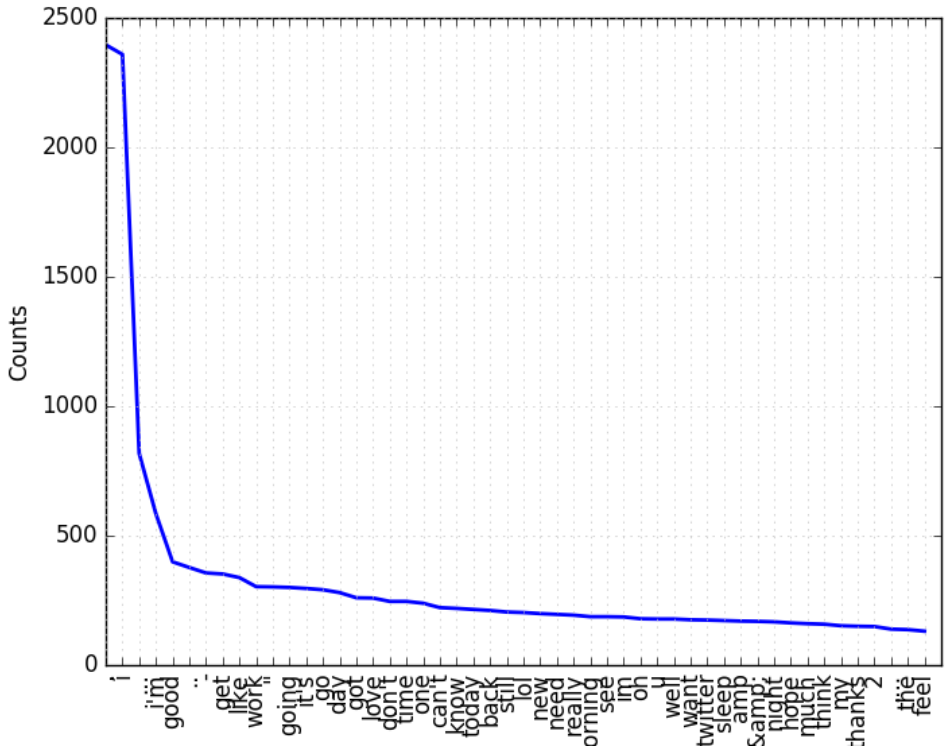

Figure 3 Frequency Distribution of Top 50 Tokens

As we can see above, after tokenizing the text with a custom tokenizer script, ‘,’ has the most frequency followed by ‘I’ and so on. However, caution should be taken in interpreting these results as there are a few issues with the tokenization process. These include issues such as failure to separate texts of the form ‘i……u……☺’, which may occur due to precedence of tokenizing ‘&’ and ‘"’. Further, these tokens themselves are not always tokenized correctly due to the variation in characters used along with them.

We now look at some of the interesting collocations that are contained in the data as follows:

```
Building collocations list
amp &amp;; can't wait; looking forward; don't know; last night; Good
morning; feel like; don't want; Can't wait; fall asleep; ice cream;
can't sleep; can't find; good luck; GUYS EVERYONE; looks like; HEY
GUYS; ADD @MattWayneCeleb; EVERYONE ADD; Gossip Girl
```

Figure 4 Collocations

Collocations, which are the most frequently occurring pairs of words that are too frequent to be by chance [4], can improve our understanding of the data and help in building a more accurate model. However, as they do not occur quite as frequently in our training data, they were not taken into consideration.

We now look at the important parts-of-speech that are contained in the data:

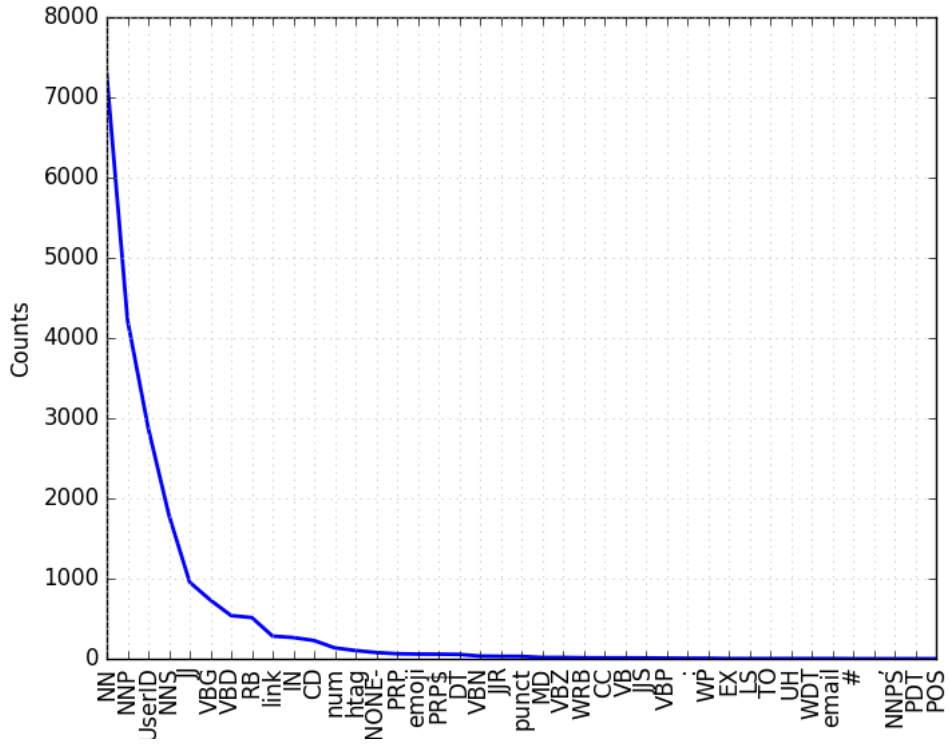

Figure 5 Frequncy Distribution plot of Parts-of-Speech

As we can see from the above frequency distribution plot, several of the words contained in the data belong to the 'NN' or Noun type, followed by 'NNP' and so on. We can also see that there is a flaw in the POS tagger which fails to tag certain punctuations and fails to tag a few tokens completely as seen by the '-NONE-' tag.

As this issue, could not be rectified even with the use of the default tagger which comes pre-packaged with NLTK, we leave them as is.

The Model:

While the original authors built a Maximum Entropy Classifier model for classifying the data [3], the Maximum Entropy Classifier supplied with NLTK proves to be quite memory intensive which would often lead to memory errors on machines with less RAM or with older processors. Hence, the project utilizes two classifiers in an attempt to achieve similar results.

The first classifier is the Naïve Bayes Classifier which comes pre-packaged with NLTK. The Naïve Bayes Classifier is based on Bayes theorem which is based on the assumption that all features are independent of each other. This also means that every feature gets a say in determining the classification of the given data. In NLTK, the Naïve Bayes Classifier begins by calculating the prior probability of each label (determined by checking the frequency of each label in the training set) which is then combined with the contribution from each feature to perform the classification. [4] This classifier has also been one of the most widely used classifiers for performing sentiment analysis on python through NLTK. [6][7]

The second classifier is the Decision Tree Classifier which also comes pre-packaged with NLTK. The algorithm for building the classifier works on the idea of selecting the best decision stump based on all the available features and then making incremental improvements by removing unnecessary features. [4]

The basic steps involved in building the classifier and utilizing it can be best described through the following flowchart [5]:

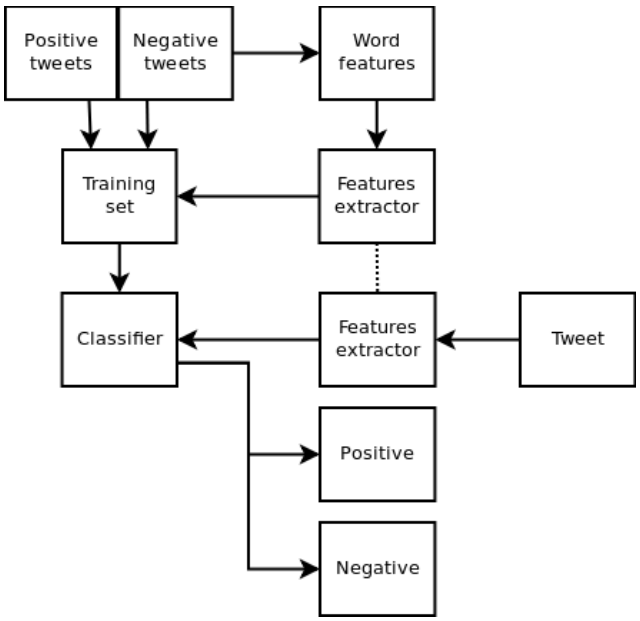

Figure 6 Flowchart of Classifier construction and utilization [5]

As seen above, the more general steps involve extracting word features from the positive and negative tweets and applying them to the training data. This set of applied training data features is then used to train the classifier. Once the classifier is constructed, the feature extractor function can extract the features of the new data (the test data in our case) and classify the data as positive or negative.

The pre-processing steps included in the feature extractor for the classifiers include:

1. **Tokenization**: in this step, the custom tokenizer script was used to break the contents of each tweet into the appropriate words.
2. **POS Tagging**: this was mainly done to filter out additional words to narrow down the pool of best words (or features). Words with the tags of 'UserID', 'email' and 'link' were removed as these were considered to have minimal contribution towards the analysis.
3. **Feature Selection**: the features used in the building the classifiers were the word themselves, if they had a length of at least 3. Additionally, the occurrence of at least two '!' and '?', as well as the occurrence of at least three periods, were also taken into consideration as these were thought to influence the intensity of the sentiments contained.

However, to improve accuracy of the classifiers, the pool of words had to be further reduced. This required the construction of a dictionary to be used as a reference within the feature extractor, which contains the most important words obtained from the training set, and a score to indicate how much more negative or positive information that word represented. The dictionary was constructed using the chi-square test (provided in NLTK through 'nltk.metrics' package) to score each word and was highly influenced by the work of Andy Bromberg for the basic metrics of scoring. [6] This process is shown in the following code snippet:

```
def getWordFeatures(tWordList, num):
    # Creating empty frequency and conditional frequency distributions
    tfd = nltk.FreqDist()
    tcfd = nltk.ConditionalFreqDist()
    # Building the distribution
    for (word,sentiment) in tWordList:
        tfd.inc(word)
        tcfd[sentiment].inc(word)
    # Obtaining negative, positive and total word counts
    negCount = tcfd['0'].N()
    posCount = tcfd['4'].N()
    totalCount = negCount + posCount
    # Defining empty holder for word scores
    wScores={}
    # Calculating word scores
    for (word,freq) in tfd.iteritems():
        negScore = nltk.metrics.BigramAssocMeasures.chi_sq(tcfd['0'][word], (freq,negCount), totalCount)
        poScore = nltk.metrics.BigramAssocMeasures.chi_sq(tcfd['4'][word], (freq,posCount), totalCount)
        wScores[word] = negScore + poScore
    # Sorting based on word scores and then creating set of 'n' best word features
    bestVals = sorted(wScores.iteritems(), key=lambda (w,s): s, reverse=True)[:num]
    bestFeats = set([w for (w,s) in bestVals])
    return bestFeats
```

Figure 7 Code snippet of dictionary building function

The following code snippet shows the feature extractor as described before:

```
# Using the getWordsInTweets() and getWordFeatures() to get the best features to use in the FeatExt()
tweetsFeats = getWordFeatures(getWordsInTweets(trainTweets),num)
# Function to extract features from Tweets
def FeatExt(ttweet):
    tagdWords = pTagger([w.lower() for w in pToken(ttweet) if len(w)>2 and w not in stopwords.words('english')])
    tWords = []
    for w in tagdWords:
        if (w[0][1] not in ['UserID','email', 'link']):
            tWords.append(w[0][0])
    feat = {}
    for e in tweetsFeats:
        feat['contains(%s)' % e] = (e in tWords)
    for e in tWords:
        if re.search('\!{2,}',e):
            feat['!']=True
    for e in tWords:
        if re.search('\?{2,}',e):
            feat['?']=True
    for e in tWords:
        if re.search('\.{3,}',e):
            feat['.']=True
    return feat
```

Figure 8 Code Snippet of the Feature Extractor

Once the dictionary has been properly constructed and the feature extractor function has been appropriately defined, construction of the classifiers is quite simple and involves applying the feature extractor to the training set and then passing it to the default classifier train functions. For the Naïve Bayes Classifier, an example of the code involved is as follows:

```
# Applying features to training and testing sets
trainTweetsApplied = nltk.classify.apply_features(FeatExt, trainTweets)
testTweetsApplied = nltk.classify.apply_features(FeatExt, testTweets)
# Building the Naive Bayes Classifier
classifierNB = nltk.NaiveBayesClassifier.train(trainTweetsApplied)
```

Figure 9 Code Snippet for Naive Bayes Classifier construction

Evaluation:

Evaluation of the model was done using two ratios of training and testing data, which was refined by evaluating the individual classifiers themselves. The classifiers were evaluated using 10, 100, 1000, 1500, 2000, 2500 and 3000 of the most important features obtained through the 'getWordFeatures()' function seen in the previous section.

The evaluation metrics used were:

1. **Accuracy**: which is a less-specific measure which indicates the proportion of the testing set which was correctly tagged. [6]
2. **Precision**: which measures the exactness of the classifier. In other words, it measures the proportion of false positives, hence, higher precision would mean less false positive and lower precision means more false positives. [7]
3. **Recall**: which measures the completeness or sensitivity of the classifier. In other words, it is a measure of the false negatives such that higher recall means less false negatives and lower recall means more false negatives. [7]

With regards to the Naïve Bayes Classifier, we can see the evaluation performed as follows:

```
For 5400 training data and 1800 testing data we have:
Evaluating best 10 features:
0.633888888889
Precision for negative Tweets:  0.623843782117
Recall for negative Tweets:  0.674444444444
Precision for positive Tweets:  0.645707376058
Recall for positive Tweets:  0.593333333333
Most Informative Features
           contains(why) = True                0 : 4      =     22.3 : 1.0
           contains(sad) = True                0 : 4      =     17.2 : 1.0
         contains(sick) = True                 0 : 4      =     13.3 : 1.0
       contains(thanks) = True                 4 : 0      =      6.6 : 1.0
         contains(hate) = True                 0 : 4      =      5.7 : 1.0
         contains(sorry) = True                0 : 4      =      3.7 : 1.0
                       ? = True                0 : 4      =      3.6 : 1.0
         contains(miss) = True                 0 : 4      =      3.3 : 1.0
         contains(good) = True                 4 : 0      =      2.6 : 1.0
         contains(love) = True                 4 : 0      =      2.6 : 1.0
None
Evaluating best 100 features:
0.679444444444
Precision for negative Tweets:  0.669464847849
Recall for negative Tweets:  0.708888888889
Precision for positive Tweets:  0.6906729634
Recall for positive Tweets:  0.65
Most Informative Features
           contains(why) = True                0 : 4      =     22.3 : 1.0
           contains(sad) = True                0 : 4      =     17.2 : 1.0
         contains(sucks) = True                0 : 4      =     13.7 : 1.0
          contains(sick) = True                0 : 4      =     13.3 : 1.0
       contains(missing) = True                0 : 4      =     10.6 : 1.0
         contains(heard) = True                0 : 4      =      9.7 : 1.0
          contains(ugh.) = True                0 : 4      =      9.0 : 1.0
     contains(tomorrow!) = True                4 : 0      =      9.0 : 1.0
        contains(stupid) = True                0 : 4      =      8.6 : 1.0
          contains(poor) = True                0 : 4      =      8.4 : 1.0
None
```

```
Evaluating best 1000 features:
0.701666666667
Precision for negative Tweets:  0.691253951528
Recall for negative Tweets:  0.728888888889
Precision for positive Tweets:  0.713278495887
Recall for positive Tweets:  0.674444444444
Most Informative Features
           contains(why) = True                0 : 4      =     22.3 : 1.0
           contains(sad) = True                0 : 4      =     17.2 : 1.0
         contains(sucks) = True                0 : 4      =     13.7 : 1.0
          contains(sick) = True                0 : 4      =     13.3 : 1.0
       contains(missing) = True                0 : 4      =     10.6 : 1.0
         contains(heard) = True                0 : 4      =      9.7 : 1.0
     contains(tomorrow!) = True                4 : 0      =      9.0 : 1.0
          contains(ugh.) = True                0 : 4      =      9.0 : 1.0
        contains(stupid) = True                0 : 4      =      8.6 : 1.0
          contains(poor) = True                0 : 4      =      8.4 : 1.0
None
Evaluating best 1500 features:
0.706111111111
Precision for negative Tweets:  0.697972251868
Recall for negative Tweets:  0.726666666667
Precision for positive Tweets:  0.714947856315
Recall for positive Tweets:  0.685555555556
Most Informative Features
           contains(why) = True                0 : 4      =     22.3 : 1.0
           contains(sad) = True                0 : 4      =     17.2 : 1.0
         contains(sucks) = True                0 : 4      =     13.7 : 1.0
          contains(sick) = True                0 : 4      =     13.3 : 1.0
       contains(missing) = True                0 : 4      =     10.6 : 1.0
         contains(heard) = True                0 : 4      =      9.7 : 1.0
     contains(tomorrow!) = True                4 : 0      =      9.0 : 1.0
          contains(ugh.) = True                0 : 4      =      9.0 : 1.0
        contains(stupid) = True                0 : 4      =      8.6 : 1.0
          contains(poor) = True                0 : 4      =      8.4 : 1.0
None
```

```
Evaluating best 2000 features:
0.711111111111
Precision for negative Tweets:  0.706073752711
Recall for negative Tweets:  0.723333333333
Precision for positive Tweets:  0.716400911162
Recall for positive Tweets:  0.698888888889
Most Informative Features
           contains(why) = True                0 : 4      =     22.3 : 1.0
           contains(sad) = True                0 : 4      =     17.2 : 1.0
         contains(sucks) = True                0 : 4      =     13.7 : 1.0
          contains(sick) = True                0 : 4      =     13.3 : 1.0
       contains(missing) = True                0 : 4      =     10.6 : 1.0
         contains(heard) = True                0 : 4      =      9.7 : 1.0
     contains(tomorrow!) = True                4 : 0      =      9.0 : 1.0
          contains(ugh.) = True                0 : 4      =      9.0 : 1.0
        contains(stupid) = True                0 : 4      =      8.6 : 1.0
          contains(poor) = True                0 : 4      =      8.4 : 1.0
None
Evaluating best 2500 features:
0.708888888889
Precision for negative Tweets:  0.69624217119
Recall for negative Tweets:  0.741111111111
Precision for positive Tweets:  0.723277909739
Recall for positive Tweets:  0.676666666667
Most Informative Features
           contains(why) = True                0 : 4      =     22.3 : 1.0
           contains(sad) = True                0 : 4      =     17.2 : 1.0
         contains(sucks) = True                0 : 4      =     13.7 : 1.0
          contains(sick) = True                0 : 4      =     13.3 : 1.0
       contains(missing) = True                0 : 4      =     10.6 : 1.0
         contains(heard) = True                0 : 4      =      9.7 : 1.0
          contains(ugh.) = True                0 : 4      =      9.0 : 1.0
     contains(tomorrow!) = True                4 : 0      =      9.0 : 1.0
        contains(stupid) = True                0 : 4      =      8.6 : 1.0
          contains(poor) = True                0 : 4      =      8.4 : 1.0
None
```

```
Evaluating best 3000 features:
0.708333333333
Precision for negative Tweets:  0.688064192578
Recall for negative Tweets:  0.762222222222
Precision for positive Tweets:  0.733499377335
Recall for positive Tweets:  0.654444444444
Most Informative Features
           contains(why) = True                0 : 4      =     22.3 : 1.0
           contains(sad) = True                0 : 4      =     17.2 : 1.0
         contains(sucks) = True                0 : 4      =     13.7 : 1.0
          contains(sick) = True                0 : 4      =     13.3 : 1.0
       contains(missing) = True                0 : 4      =     10.6 : 1.0
         contains(heard) = True                0 : 4      =      9.7 : 1.0
     contains(tomorrow!) = True                4 : 0      =      9.0 : 1.0
          contains(ugh.) = True                0 : 4      =      9.0 : 1.0
        contains(stupid) = True                0 : 4      =      8.6 : 1.0
          contains(poor) = True                0 : 4      =      8.4 : 1.0
None
```

Figure 10 Naïve Bayes Classifier 75:25 evaluation

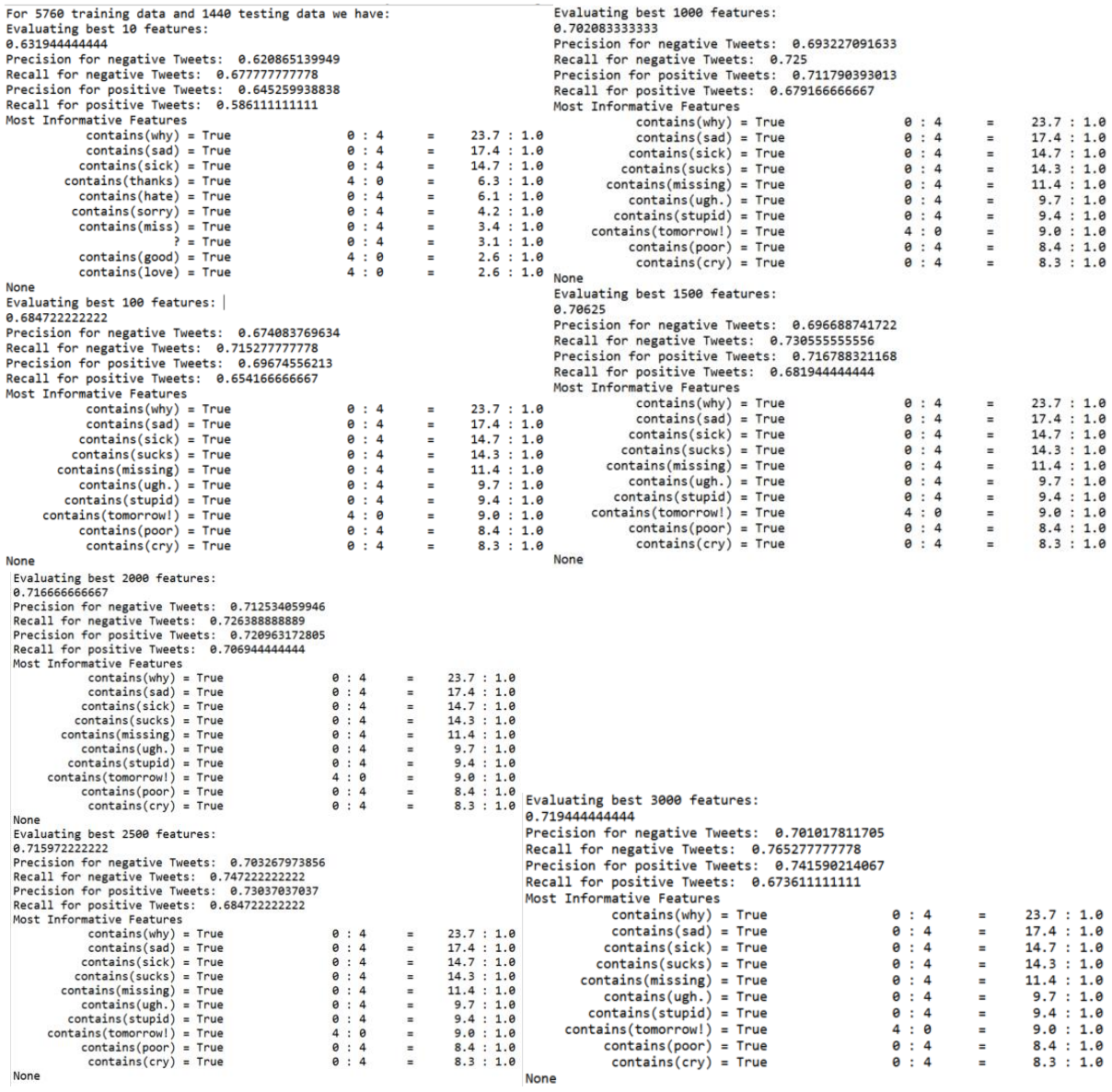

Figure 11 Naïve Bayes Classifier 80:20 evaluation

Based on the evaluation metrics, it was found that using a ratio of 80% training data to 20% testing data, and selecting the 2000 best word features, gives an overall good result for all metrics considered.

The Decision Tree Classifier on the other hand failed in terms of being built successfully and attempts to construct the classifier with the produced feature sets resulted in runtimes which far exceeded 2 hours. As the runtime far outweighed the runtime of the Naïve Bayes Classifier (which took a maximum of 15 minutes), the idea to combine the results had to be dismissed.

Conclusion:

While the model achieves decent accuracy of roughly 72%, there is a lot of room for improvement of the classifier. However, as the metrics of precision and recall are quite close to each other in what was found to be the optimal set of features, we can safely conclude that the classifier can classify the data

evenly. As we can see in the following screenshot, the feature set constructed also has appropriate weightage for the different word features:

```
Most Informative Features
           contains(why) = True              0 : 4      =     23.7 : 1.0
           contains(sad) = True              0 : 4      =     17.4 : 1.0
          contains(sick) = True              0 : 4      =     14.7 : 1.0
         contains(sucks) = True              0 : 4      =     14.3 : 1.0
       contains(missing) = True              0 : 4      =     11.4 : 1.0
          contains(ugh.) = True              0 : 4      =      9.7 : 1.0
        contains(stupid) = True              0 : 4      =      9.4 : 1.0
    contains(tomorrow!) = True              4 : 0      =      9.0 : 1.0
         contains(poor) = True              0 : 4      =      8.4 : 1.0
          contains(cry) = True              0 : 4      =      8.3 : 1.0
        contains(stuck) = True              0 : 4      =      7.8 : 1.0
     contains(anymore) = True               0 : 4      =      7.7 : 1.0
        contains(agree) = True              4 : 0      =      7.7 : 1.0
        contains(hurts) = True              0 : 4      =      7.4 : 1.0
       contains(missed) = True              0 : 4      =      7.4 : 1.0
       contains(crying) = True              0 : 4      =      7.0 : 1.0
     contains(congrats) = True              4 : 0      =      7.0 : 1.0
         contains(hehe) = True              4 : 0      =      7.0 : 1.0
         contains(lost) = True              0 : 4      =      6.8 : 1.0
       contains(thanks) = True              4 : 0      =      6.3 : 1.0
     contains(hospital) = True              0 : 4      =      6.3 : 1.0
       contains(loving) = True              4 : 0      =      6.3 : 1.0
          contains(yep) = True              4 : 0      =      6.3 : 1.0
       contains(behind) = True              0 : 4      =      6.3 : 1.0
        contains(sadly) = True              0 : 4      =      6.3 : 1.0
            contains(3) = True              0 : 4      =      6.3 : 1.0
        contains(hate) = True               0 : 4      =      6.1 : 1.0
        contains(blog) = True               4 : 0      =      5.9 : 1.0
      contains(heard) = True                0 : 4      =      5.8 : 1.0
        contains(fell) = True               0 : 4      =      5.7 : 1.0
```

Figure 12 30 most important features of the Naive Bayes Classifier

We have also seen that the effects of the number of '!', '?' and periods are not important for analyzing the sentiment of most tweets. However, additional data on the usage of such punctuations may lead to better formulations for usage in the feature set.

Future work:

The model shows much need for improvement and some areas of improvement are as follows:

1. The process of tokenization has a few issues associated with it. For example, handling the occurrence of '"', a result of improper reading of the csv files content field, could be improved by adding additional possible variations of its occurrence. Also, the portion which handles breaking up of tokens which are of the form 'hello…u…:/' is not always successful.
2. Further work can also be done to obtain the stem of words which can improve the feature set and the feature extractor function.
3. The part-of-speech tagger is not always accurate especially when it comes to classifying a few link types.
4. An idea discussed in [8], could be applied in this case where the Decision Tree Classifier could instead be used to form a better feature set for use in the Naïve Bayes Classifier.
5. Rather than just classifying the text to be only positive or negative, additional classifications could be made to give more value to each tweet.

References:

[1] "Sentiment Analysis", Wikipedia, (https://en.wikipedia.org/wiki/Sentiment_analysis)

[2] “Natural Language Processing”, Wikipedia, (https://en.wikipedia.org/wiki/Natural_language_processing)

[3] “Sentiment140 – For Academics”, Sentiment140, (http://help.sentiment140.com/for-students)

[4] “Natural Language Processing with Python”, Steven Bird, Ewan Klein, and Edward Loper, O'Reilly Media, 2009, (http://www.nltk.org/book_1ed/)

[5] “Twitter Sentiment Analysis using Python and NLTK”, Laurent Luce, January 2nd 2012, (http://www.laurentluce.com/posts/twitter-sentiment-analysis-using-python-and-nltk/)

[6] “Second Try: Sentiment Analysis in Python”, Andy Bromberg, February 14,2013, (http://andybromberg.com/sentiment-analysis-python/)

[7] “Text Classification for Sentiment Analysis – Precision and Recall”, Jacob(streamhacker.com UserID), May 17,2010, (http://streamhacker.com/2010/05/17/text-classification-sentiment-analysis-precision-recall/)

[8] “OPINION MINING USING DECISION TREE BASED FEATURE SELECTION THROUGH MANHATTAN HIERARCHICAL CLUSTER MEASURE”, JEEVANANDAM JOTHEESWARAN, DR. Y. S. KUMARASWAMY, Journal of Theoretical and Applied Information Technology, December 10,2013, (http://www.jatit.org/volumes/Vol58No1/8Vol58No1.pdf)