Twitter as a Model for Sentiment Analysis and Text Classification

John Saja, Ian Stanton, Pawandeep Singh

May 12, 2016

Introduction

An important subtopic in the field of Artificial Intelligence is data science. Data science is the study of the generalizable extraction of knowledge from data [5], which is used to garner new insights and help develop solutions that make intelligent use of trends and models from data. One subfield that data science encompasses is predictive analytics, which is a broad term used to describe a variety of statistical and analytical techniques that evaluate current and historical facts to forecast future events or behaviors [17]. One current and popular topic in the field of predictive analytics is Social Media. Social Media is the largest marketing platform on the web today, and is one of the most frequently used web utilities throughout all people who use the internet. It logically follows that there is a vast wealth of untapped data in these networks, since the volume of data that flows through Social Media networks each day are enormous - which is easily many millions of messages and posts, much of which can be used to analyze otherwise unnoticed trends, and help to shape new meaningful technologies for the future. One interesting aspect of Social Media is the Twitter platform. Twitter is a popular microblogging service which enables people to easily share aspects of their lives with others, and talk about what they find interesting on a day-to-day basis by "tweeting".

One way that this platform is beneficial in the field of predictive analytics is that one can look at the collection of a user's tweets, and then may be able to draw relatively accurate inferences about the user's personality, preferences, and habits. This is interesting and has real value because it could prove useful in the field of psychology to possibly to identify personality types. Additionally, it also currently has applications in marketing and business, because a better understanding of a potential client can help direct targeted advertising.

One interesting problem in the field of opinion mining is sentiment analysis. Sentiment analysis is the process of analyzing text in order to categorize and identify subjective material from a particular source. A wide area of application for sentiment analysis is social media. Another interesting open problem in natural language processing and machine learning is the task of categorizing certain texts as belonging to certain topics or categories. This is known as text classification or document classification. We attempt to approach both of these problems in our project

Literature Review / Related Work

This portion of the paper will discuss different topics as they relate to predictive analytics. The intent is to gain a better understanding of the type of solutions that can be made from pre-existing

data, what techniques can be used to implement these solutions, and how previous work in the field relates to the problem we are endeavoring.

Topic 1: Predictive analytics in social media and their use in modern applications

Many current applications and inventions are driven by the use of certain inference and modeling techniques in Bayesian statistics. Three recent studies and experiments that make use of these techniques are ones that attempt to predict disruptive events/ natural disasters, crime, and personality. These all differ greatly in their approaches, and the respective Bayesian techniques they most heavily use are the following: naïve bayes classifier statistic, kernel density estimation function, and a lesser known m5-rules algorithm.

Predicting disruptive events

The first article is about a group of researchers who analyze their invention of an event detection system which uses Twitter as their source of media/information. Their framework is a 6 stage process: data collection, preprocessing, classification, clustering, feature selection, and summarization. One of the most important steps to success in this experiment was having what is known in the AI and machine learning field as a "training set" of data, so that observations gathered from raw data have a basis to learn from. For this experiment, the live Twitter stream region chosen was Abu Dabhi and the training set event was the Formula 1 Grand Prix 2013 [1]. Tweets were collected both prior to and during the event as to check the validity of their system. The heart of the problem solving in this framework is the classification step, which uses a popular machine learning statistic called the Naive Bayes Classifier. This classifier provides a solution to sentiment analysis. Informally, this is taking a snippet of text and classifying the mood or sentiment which the text conveys. The researchers use this to,"... Reduce the amount of noise from the incoming tweets and filter out as many non-event tweets as possible." [1]

Predicting personality

The second article focused on another group of researchers approaching the problem of predicting personality based on previous social media activity on Facebook. The first step of this process was having a modest sample size of people and then administering , "...The Big Five Personality Inventory to 279 subjects through a Facebook application" while also collecting all their public information on their profiles such as gender, race, religion, education history, etc [7]. Hundreds of features that they personally gathered were given scores computed on the following algorithms they used - one of these algorithms is called the M5Rules [19]. This algorithm is a modification of the logistic model tree classification model used in machine learning, which makes use of linear regression techniques to classify various types of data [13]. One of the faults of this model it is only accurate when the dimensionality of data is low (the data might be overfit otherwise); Since the researchers are working with hundreds of different features, the M5Rules algorithm corrects this exact problem. The scores computed were then compared and given a correlation to the original results from the personality test, which gave scores in five categories: openness, conscientiousness, agreeableness, extraversion, and neuroticism. Overall, they were able to achieve results within 10 percent error for each of the five traits [7].

Predicting crime

Another interesting application of predictive analytics is predicting crime. The methodology of one researcher's experiment is the following - the researcher focused on the city of Chicago, and gathered all crime listings/reports in a 2 month period. They then categorized the crimes into 25 different types. At the same time, they kept track of tweets that fell within the boundaries of the city, using the Twitter Streaming API. For a one-month period, crime data was gathered in order to form the training set, and a kernel density estimation function was used to form of a density map (figure 2) [6]. The same was done in the period of tweet collection, but instead used a form of syntactic analysis to identify the topic of the tweet, and grouped topics based on the neighborhood they came from (additionally,figure 5 provides a nice visual comparison between the training set and predicted set). Out of the 25 topics that they had categorized, it was shown that 19 showed improvements when the twitter data was introduced [6].

Topic 2: Real world applications

Though predictive analytics and data mining through social media and micro blogs is a new way that researchers are gathering data, predictive analytics has generally been around for some time. The business world is very familiar with the use of predictive analytics in marketing. Many companies hold enormous databases full of consumer information. These databases are sometimes as large as 100+ terabytes in capacity. For instance, "General Motors has 12 million GM credit card holders in a database containing detailed data on customer buying habits " [10]. Companies like GM, WalMart, Vodafone, etc. use this consumer information to develop new strategies that better serve their customers. Many companies are even using predictive analytics for each phase of their customer life cycles, which includes the acquisition of new customers, revenue increase from existing customers, and keeping good customers. While this method of gathering data is great for marketing in business, it has many other useful applications in the world we live in.

Health

One of the more useful applications of social network data mining is using the data to predict and monitor global health information. For example, applications such as 'Who Is Sick', 'HealthMap' and 'Google Flu Trends' help to monitor where and when people are getting sick [4]. These applications harvest their data with a data crawler that automatically searches the web every 10 minutes. The information is gathered from a variety of social media websites. 'WeFeelFine' is another application that assesses the effects of events around the world on the feelings and emotions of individuals. Once 'WeFeelFine' has gathered its information, the data is scanned against a list of defined 'feelings'. If a piece of information matches one of those feelings, it then represents one person in that particular mood.

Aside from predicting moods and outbreaks of influenza, predictive analytics is also used in epidemic intelligence. Organizations collect data about "potential disease outbreaks from both formal and increasingly informal sources" [18]. For example, a study was conducted where 135,000 tweets were collected during the Swine Flu pandemic. These tweets were collected by using Twitter's REST API and Search API. The group gathered tweets that contained the keyword 'flu', and then wrote PHP code to parse and save the resulting tweets to a MySQL database. Following the study, the group stated that "these results highlight the potential for Twitter to be used in conjunction with pre-existing epidemic intelligence tools"[18].

Movies

Another study used information mined from Twitter to "forecast box-office revenues for movies"[2]. The group's initial hypothesis was that movies that people are talking about in a positive manner will receive more viewers, more positive ratings, and therefore more revenue. During this study, text classifiers were used to separate positive tweets from negative tweets. Keywords present in the movie titles were used as search arguments. In order to ensure that the information included the time-stamp, author and tweet text, tweets were extracted using the Twitter REST API and Search API. Over the course of three months, the study collected 2.89 million tweets that mentioned 24 different films released during that time frame. The results from this study showed that "social media feeds can be effective indicators of real-world performance"[2] and that "the rate at which movie tweets are generated can be used to build a powerful model for predicting movie box-office revenue"[2]. The results were also better than the industry standard, Hollywood Stock Exchange.

Stock Market

Some even believe that predictive analytics combined with Twitter data can be correlated to the value of the Dow Jones Industrial Average (DJIA). In this study, two different tools (OpinionFinder and GPOMS) were used to analyze the moods of individuals based on their tweets. OpinionFinder is a tool that helps analyze textual content from tweets on a particular day. This data is used to give a positive vs negative daily time series for individuals around the world. GPOMS performs the same general job, but gives more detailed results. Tweets that contained statements about someone's current mood such as 'I feel...', 'I don't feel...', and 'I am feeling' were used during this study. The results of this study showed that "the prediction accuracy of standard stock market prediction models is significantly improved when certain mood dimensions are included, but not others. Variations along the public mood dimensions of Calm and Hapiness as measured by GPOMS seem to have a predictive effect, but no general happiness as measured by the OpinionFinder tool"[3].

Topic 3: Marketing

Marketing is one of places where Predictive Analysis is used. In Marketing, customer data patterns are used to answer questions about customer performance. There are several applications and models that utilize Predictive Analysis to find data patterns.

CRM

Analytical Customer Relationship Management (CRM) is a frequent commercial application of Predictive Analysis. According to industry sources,1 worldwide CRM-related investments reached $3.3 billion in 1999 and are expected to reach 10.2$ billion by 2003 [11]. Companies use this application to learn much more about their customers' buying habits and desires. Many organizations are using the techniques to help manage all phases of the customer life cycle, including acquiring new customers, increasing revenue from existing customers, and retaining good customers [**?**]. The application provides the capability of studying the data and analyzing it to predict behavior of consumer, organization, competitors and upcoming business trends, which helps the management set up their goals and objectives. Predictive models often perform calculations during live transactions, for example, to evaluate the risk or opportunity of a given customer or transaction, in order to guide a

decision. Analytical Customer Relationship Management can be applied throughout the customers' lifecycle.

Customer Retention

With the number of competing services available, businesses need to focus efforts on maintaining continuous consumer satisfaction, rewarding consumer loyalty and minimizing customer attrition. When businesses tend to respond to customer attrition on a reactive basis, acting only after the customer has initiated the process to terminate service, the chance of changing the customer's decision is almost zero. Predictive analytics can help them come up with a more proactive retention strategy by frequently examining customer's past service usage, service performance, spending and other behavior patterns, predictive models can determine the likelihood of a customer terminating service sometime soon. Predictive analytics can also help predict customer behavior when they slowly but steadily reduce the usage of the product, so company can take proper actions to increase customer activity. During the study conducted on 100,000 customers, it was clear that some 13 percent of the customers bought a new product during the follow-up period, whereas fewer customers (6.8 percent of the customers) decided to cancel a product with a non-ending status [14]. This study proves to show that predictive analysis help companies retain customers.

Cross-Selling

Often corporate organizations collect and maintain customer records and sale transactions as exploiting hidden relationships in the data can provide a competitive advantage. For an organization such as banks that offers multiple products, predictive analytics can help analyze customers' spending, usage and other behavior, leading to efficient cross sales, or selling additional products to current customers. Today, marketing is the biggest user of predictive analytics with cross-selling, campaign management, customer acquisition, and budgeting and forecasting models top of the list, followed by attrition and loyalty applications [15]. This directly leads to higher profitability per customer and stronger customer relationships.

Relation to Our Project

Most of the approaches associated with these topics lie more on the statistical side, but still fully represent AI and machine learning techniques that have a wide range of practical real life applications - From marketing strategies to predicting crime. The core differences between all of these related topics in the field are for the most part based on the algorithms and statistics they use for computation and analysis, which range between commonly used machine learning statistics such as the Naive Bayes Classifier, to using tools like OpinionFinder.

As it relates to our current project - We are working on making a slightly modified version of sentiment analysis using Twitter, gathering user data, and analyzing and interpreting the results. Understanding the approaches, techniques, and tools that researchers have used in the past should give us beneficial and useful insights on how to approach our project.

Approach

How?

We had found that both of these problems can be approached by using a naive bayes classifier. For text categorization, we will only use this classifier.

For sentiment analysis on the other hand, there are many other algorithms that can be used to approach the problem. As you will see in the experiment section, we will use a combination of multiple algorithms in order to improve accuracy. These algorithms include (but are not limited to) a few variations of the naive bayes classifier.

One important thing to note is that the naive bayes classifier uses what is known as a supervised learning technique. A supervised learning algorithm one where it uses examples, or training data, to devise some algorithm that maps instances of examples to their labels, given any example. In the case of text classification as it relates to this project, this is finding a mapping between tweets and their proposed category. In order to get a general feel for how the naive bayes classifier words, one must first understand Bayes Theorem.

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

Figure 1: Bayes Theorem

This theorem assumes two independent events, A and B, and a conditional probability between them. This simply calculates the probability of an event A given an observance of an event B. On the other hand, the naive bayes classifier is a much more advanced topic better left to be explained by statisticians, but the general principle of finding the probability of an event given that another has occurred is still heavily used for the classifier.

Our Goal

In relation to this project, we would like to use sentiment analysis on a user's tweets to get a better idea of what kind of person they are (negative or positive). By classifying tweets as negative or positive, we will be able to graph the user's sentiment for observation.

The goal of text categorization as it relates to this project is the following: We want to see how many times our trained classifier guesses the correct label for any given manually correctly labeled tweet. A higher percentage of correct guesses would lead us to conclude that our model is sound.

The Methods

Our model for sentiment analysis is using positive and negative movie reviews as training data for our classifiers. We will only train with adjectives, adverbs and verbs from these reviews. We then use these trained classifiers to classify tweets as positive or negative. This process will be applied to 5 collections of tweets, each from a University of Minnesota Student.

Our model for topic categorization is using random tweets containing hashtags that match our topic as training data. We also have 3 files of manually labeled music, food, and sports tweets. We

essentially throw the random tweets into training data list to train a naive bayes classifier, where we then use that trained classifier to guess our manually assigned labels.

These methods will be covered in much more depth in the experiments section.

Experiment Design and Results

Tools and Software

When researching our topic, we found numerous tutorials describing the many interesting things you can do with Twitter data. We noticed that the majority of these tutorials were using Python, along with various Python modules that needed to be installed separately. Our initial decision was to use Python for our tweet gathering and analysis as in the tutorials we had found. This, however, posed one issue for us. Because we would have to install new software packages, we wouldn't be able to use the CSELab machines at all. Once we confirmed that we all had Python installed on our machines, we proceeded by installing the long laundry list of packages and dependencies.

We started by installing 'pip' on our machines. 'Pip' is a package management program used for installing python packages. Next on the list was IPython. While this was not completely necessary for our project, we installed IPython because it is a much more powerful interactive python shell compared to the standard Python interpreter. IPython's features made it much easier for us to work with our data. Once we had the basics installed, we moved on to installing Twitter and Tweepy. The Twitter API was necessary for accessing and gathering tweets from the Twitter database. Tweepy is an easy to use module that allows you to communicate with the Twitter platform and use its Twitter API through Python. Lastly, we used the textBlob python library in the text classification portion for its straightforward simple approach in training a naive bayes classifier. It only requires a list of tuples as training and test data, where the tuples are informally of the form: (tweet, 'corresponding label'). Another advantage is that it does not require that you have a feature extractor. All modified source code used in the following sentiment analysis steps has been borrowed from Harrison Kinsley[12].

Data Collection - Sentiment Analysis

After installing the essentials, we were able to find and modify code that would allow us to access and download tweets from any public (or mutually followed) account. An important modification we made to this code was adding the ability to save tweets to a plain text file. This would allow us to read one tweet per line later in our data analysis. We also modified the code so that it would filter out re-tweets. A re-tweet is a when someone shares someone else's tweet. We viewed re-tweets as another person's tweet entirely, and decided to keep them out of our data in order to obtain the most accurate results.

One of the most important tools we used in processing our data was the Natural Language Toolkit (NLTK)[20]. As the name suggests, NLTK is a module used by Python that assists with natural language processing. NLTK allows you to perform tasks such as text parsing and extraction, recognizing the part of speech of a particular word, and can even help in training your machine to understand the meaning of text. Bundled with NLTK is the NLTK Corpus, which is a very large collection of text from just about anywhere you can imagine. Some examples include multiple Shakespeare plays, overheard conversations, and even posts made by singles on online dating sites.

The NLTK corpus includes a particular collection of text that would prove to be useful in our analysis; a collection of both positive and negative movie reviews. We eventually wanted to be able to classify tweets as negative or positive, and we thought that this collection of reviews could be used to train our machine. With the help of NLTK, we were able to read in data from these reviews one by one. The data would then be stored as a tuple containing the review itself separated by word (represented as a list), and its category ('pos' or 'neg'). In order to observe trends in specific types of words in positive and negative reviews, we put together a list of the top 3000 most used words. These words would be our 'features'. We then used a function that would identify all words in a particular movie review (or any document) that were also members of the top 3000 most used words (our feature set).

Data Collection - Text Classification

In order to eventually run our intended analysis on the tweets, we need two sets of data to use in conjunction with our naive bayes classifier.

Training Data

We acquired an equal number of randomly selected tweets pertaining to the following hashtags: music, food, and sports. This was done by using the Tweepy library's search function, with the following code:

```
musicTweet = tweepy.Cursor(api.search, q='music', lang="en").items(750)
foodTweet = tweepy.Cursor(api.search, q='food', lang="en").items(750)
sportsTweet = tweepy.Cursor(api.search, q='sports',lang="en").items(750)
```

Figure 2: Tweepy Code

When running experiments against our test set of data, we started by collecting 75 tweets (25 per topic), then 150, and finally 2,250. The reasoning behind this was to see how much the performance of our classifier varied between larger data sets and smaller ones, while still comparing the performances between two smaller ones.

Test Data

We also acquired a set of tweets for testing against our training set. We manually collected 15 tweets per topic (picking a variation of what we considered to be easy and hard for the classifier to guess) which we thought were a good representation of the possible topics as a whole. We put these into a text file to later be picked up by used by our classifier.

Pre-processing

In general, when processing documents or even tweets in our case, we have a lot of unnecessary data. For example, words like 'a', 'and' , 'this', etc.don't really provide much use because they are of low semantic value, and can generally be taken out of a document before running any kind of analysis. Thus, these are what are known in natural language processing as stop-words. In our

project, we don't filter out stop-words because the word tokenizer in the NLTK is fairly inaccurate when attempting to tokenize tweets, sometimes tokenizing an entire tweet as a single token.

Processing - Sentiment Analysis

Our next step would be to start categorizing words as having positive or negative sentiment. This would be accomplished by using the Naive Bayes classifier provided by NLTK. In order to avoid bias in our data, we needed to create a 'training set' and a 'testing set'. Our training set consisted of the first 1900 words in the feature set, and the testing set contained the remaining words in the set.

After splitting our features into two sets, we 'trained' the machine by running NLTK's Naive Bayes Classifier on our training set and assigning the result to a 'classifier' variable. The classifier looks through all of the documents in the training set and takes note of the location and frequency of each feature. It then classifies each feature word as either 'neg' or 'pos'. For example, if a word were to occur more often in positive reviews, the classifier will assume it is a positive word. The same can be said for words that occur more often in negative reviews. Once trained, we use NLTK's classifier to test accuracy. The testing set is fed to the NLTK classifier, which makes a guess as to whether the features are positive or negative. Accuracy is then calculated by comparing our testing results with the known categorization of each feature.

While NLTK's Naive Bayes algorithm suited our needs for sentiment analysis, we eventually wanted to run multiple algorithms to improve classification accuracy. We also wanted to test these algorithms on much larger data sets. This, however, was an issue because the algorithms would take a very long time to run. The answer to this problem was 'pickling'. 'Pickle' is a Python module that allows you to save Python objects that can later be loaded into a Python program. This would not only prove to be useful for our classifiers, but also for the document data being imported from the NLTK corpus.

With our freshly pickled Naive Bayes classifier saved, it was time to start testing additional classifiers based on new algorithms. The new algorithms we would use for classifying came from a popular Python module called 'Scikit-learn'. While NLTK comes packaged with its own classifier out of the box, Scikit-learn offers a plethora of classifiers. For example, Scikit-learn offers a few variations of the Niave Bayes algorithm. These variations include the Multinomial Naive Bayes algorithm and the Bernoulli Naive Bayes algorithm. In addition to these new algorithms, we also introduced Logistic Regression, Linear Support Vector Classification, and Stochastic Gradient Descent Classification.

By running these classifiers one after another, we would be able to generate multiple opinions regarding the classification of a word. In other words, for each feature word, all of the classifiers get one vote. Whichever classification (positive or negative) received the most votes would be the classification we choose. This allowed us to increase accuracy and reliability in our data. This also allowed us to calculate the classifier confidence. For example, if all classifiers vote 'positive', our confidence would be 100%. Up to this point, we generated our feature set from our movie reviews and then ran our classifiers on those movie reviews. With the ability to run any sort of document or data through our classifiers, we were almost ready to start classifying our tweets.

While the current training data was great for classifying the movie reviews, it would prove to be almost useless when classifying Tweets. The classifiers tended to lean heavily towards negative classification. This could have been because the reviews from the current training set were about a paragraph in length. We wanted to train with data similar to tweets, so we used a new data set

consisting of much shorter, one line movie reviews. Like before, these reviews were classified as negative or positive. The new data set was also much larger than the last, with more than 10,600 reviews in total (5,300+ negative, 5,300+ positive). Using a data set of this size would help to improve classification accuracy.

The next step was to create our own sentiment analysis module. This would allow us to create a sentiment function that would return the classification and confidence level for each tweet. Pickling all of our classification algorithms was essential here. Without pickling, each tweet analysis would take roughly 30 minutes. The following figure is the result of training and testing each classifier:

```
10664
Original Naive Bayes Algo accuracy percent: 67.16867469879519
Most Informative Features
              engrossing = True              pos : neg    =     20.9 : 1.0
                 boring = True               neg : pos    =     18.3 : 1.0
               touching = True               pos : neg    =     16.1 : 1.0
                generic = True               neg : pos    =     15.8 : 1.0
                routine = True               neg : pos    =     15.8 : 1.0
              supposed = True                neg : pos    =     13.7 : 1.0
             inventive = True                pos : neg    =     13.6 : 1.0
             wonderful = True                pos : neg    =     12.1 : 1.0
                  dull = True                neg : pos    =     11.1 : 1.0
                stupid = True                neg : pos    =     10.7 : 1.0
                  save = True                neg : pos    =      9.7 : 1.0
            apparently = True                neg : pos    =      9.7 : 1.0
                nicely = True                pos : neg    =      9.6 : 1.0
                  loud = True                neg : pos    =      9.5 : 1.0
              annoying = True                neg : pos    =      9.1 : 1.0
MNB_classifier accuracy percent: 66.86746987951807
BernoulliNB_classifier accuracy percent: 66.86746987951807
LogisticRegression_classifier accuracy percent: 69.57831325301204
LinearSVC_classifier accuracy percent: 68.37349397590361
SGDClassifier accuracy percent: 67.16867469879519
voted_classifier accuracy percent: 67.01807228915662
```

Figure 3: Classifier Accuracy

After pickling each algorithm and creating our sentiment module, it was time to test it out. To do so, we called our sentiment function on hard-coded sentences (one positive and one negative) resembling tweets. The following figures are the test itself, followed by the results:

```
import sentiment_mod as s


print(s.sentiment("I am so confident with my performance on the CSCI 4511W final. Expecting a great grade."))
print(s.sentiment("That final couldn't have gone any worse. Hardest final I have ever taken"))
```

Figure 4: Test Input

```
('pos', 1.0)
('neg', 1.0)
```

Figure 5: Test Classification

Now that we had confirmed our sentiment analysis module was working, all we needed to do was import the module into our original tweet gathering program. The following is a piece of the

code in our tweet gathering program.

```
        #write the txt
        with open("FilteredTweets.txt","a+") as f:
                for i in filtered_tweets:
                        f.write(i.decode('utf-8'))
                        f.write('\n')
                        print(i)
                        print(s.sentiment(i.decode('utf-8')))
                        print('\n')
                        sentiment_value, confidence = s.sentiment(i.decode('utf-8'))
                        output = open("results/[twitterHandle].txt","a")
                        output.write(sentiment_value)
                        output.write('\n')
                        output.close()


if __name__ == '__main__':
        #pass in the username of the account you want to download
        get_all_tweets("[twitterHandle]")
```

Figure 6: Sentiment Module Implementation

From here, we specified the Twitter handle we were interested in and our output file name. Running this would write the sentiment classification ('pos' or 'neg') for each tweet to a new line in our output text file.

The final step in our Twitter sentiment analysis was plotting users' tweet trends on a graph. Observing this graph would give us a good idea of the user's personality based on their tweets. We were able to do this with help from Python's 'matplotlib' module (a library used for 2D plotting in Python). Initially, we had our classifiers trained using only adjectives. We found that there was a large bias toward negative classification when using this approach. Here is an example graph using this configuration:

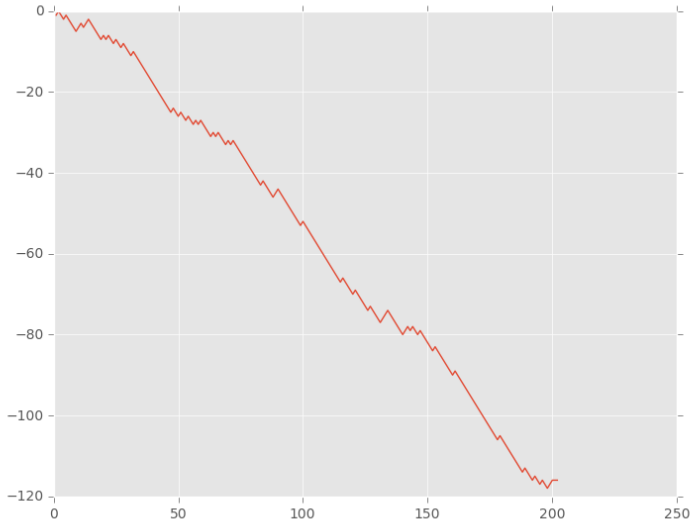

Figure 7: Results Before Bug Fix

After modifying the training to additionally include verbs and adverbs, we saw much more accurate results. We also found a bug in the code that was gathering duplicate tweets. Because our users' tweet counts varied, we set the limit to their last 100 tweets. All of these fixes were implemented in the figure below:

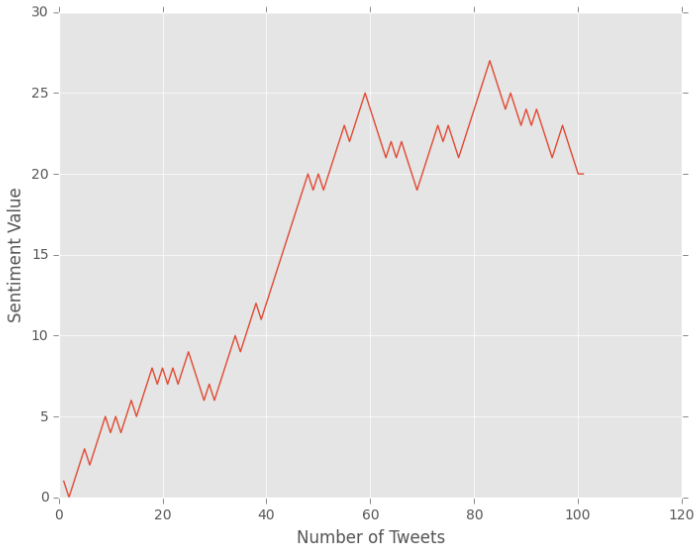

Figure 8: Results After Bug Fix

Processing - Text Classification

After collecting the tweets to be used as training and test data respectively, the next step involved preparing each tweet to be labeled as training data. What we did was append each tweet and its corresponding category label into the training data list (as shown below).

```
31  trainingData = []
32  musicTweet = tweepy.Cursor(api.search, q='music', lang="en").items(750)
33
34  for tweet in musicTweet:
35          tweetText = tweet.text.encode("utf-8")
36          tweetText = unicode(tweetText, errors='ignore')
37          x = (tweetText, 'music')
38          trainingData.append(x)
```

Figure 9: Training Data

An interesting unforeseen problem that we ran into was the encodings of the tweet were usually a mix of utf-8 and other encodings such as ASCII. One possible cause of this is the use of emoji's. This prompted us to encode the text as utf-8 and then ignore any encoding errors that may have been incurred later on in the classification process. After these lines of code are executed, we have what can be roughly described as the following: The training data list is a list of tuples of the following form: (tweet, 'label'). Now that the training data is all set, we need to train the classifier with the training data list. This can be done in a single line using textBlob's naive bayes classifier.

```
54  cl = NaiveBayesClassifier(trainingData)
```

Figure 10: Naive Bayes Classifier

After this line of code is executed, the classifier took some time to train. On inputs of 75 and 150 tweet training sets, this took roughly around 15 to 30 seconds. However, on an input of a 2,250 tweet training set, the classifier took 34 minutes to train.

Once the training was finished, we opened the three files where we had stored our test set of tweets, singled out a specific topic, and ran our analysis on that topic. Our results we were aiming to obtain were the number of correct classifier guesses out of the total number of tweets per category (each tweet in the test set was put on a single line of a text file).

```
#Contain test set of tweets
musicTest = open('musicData.txt', 'r')
foodTest = open('foodData.txt', 'r')
sportTest = open('sportData.txt', 'r')

t = 0 # Number of lines in musicTest
u = 0 # Number of correct guesses for musicTest
for line in musicTest:
        a = line
        t += 1
        if cl.classify(line) == "music":
                u += 1
```

Figure 11: Topic Analysis Code

This concludes the processing portion of the project.

Results - Sentiment Analysis

The following figures are the results from running our tweet classifier on 5 University of Minnesota students:

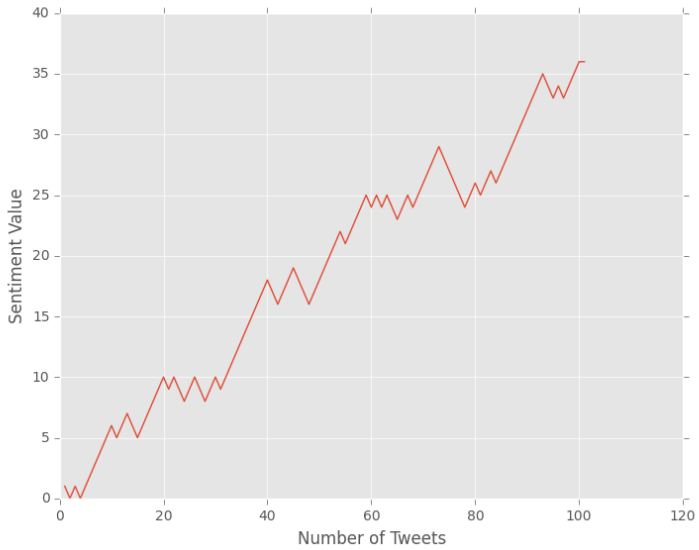

Figure 12: University of Minnesota student 1

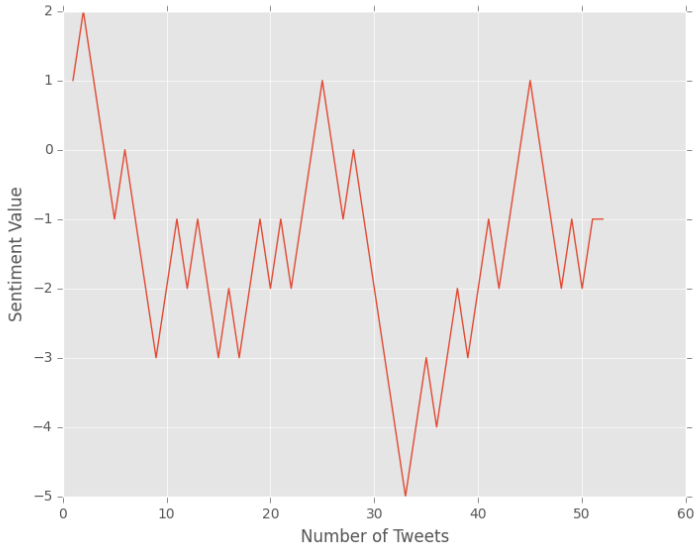

Figure 13: University of Minnesota student 2

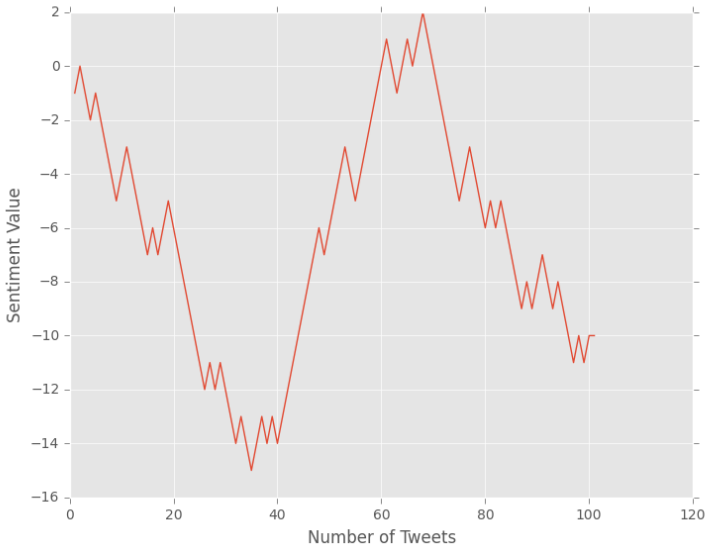

Figure 14: University of Minnesota student 3

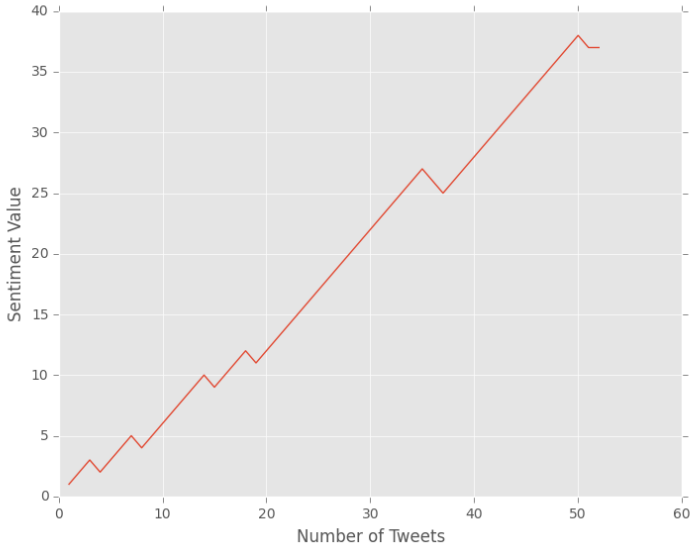

Figure 15: University of Minnesota student 4

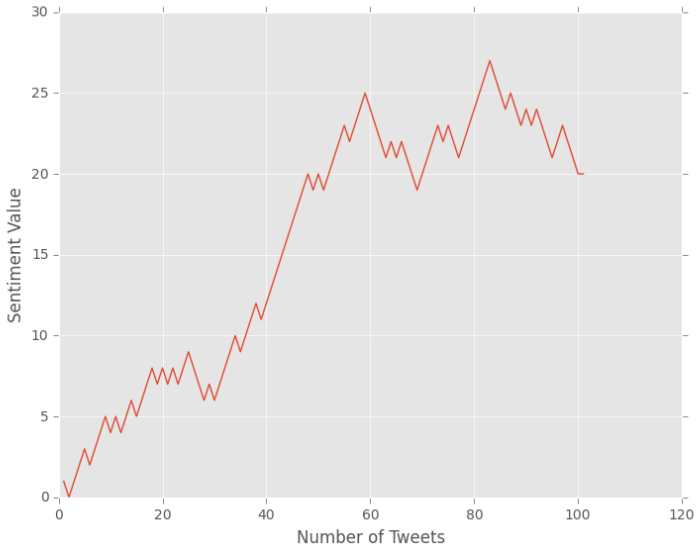

Figure 16: University of Minnesota student 5

Results - Text Classification

75 tweets to train (25 per topic), average over 5 distinct classifiers:
Music: 5 + 10 + 6 + 6 + 7 = 33/74 = 44%
Food: 3 + 4 + 3 + 3 + 9 = 19/75 = 25.333%
Sport: 5 + 2 + 11 + 11 + 3 = 32/75 = 42.666%

150 tweets to train (50 per topic), average over 5 distinct classifiers:
Music: 5 + 3 + 2 + 3 + 9 = 22/75 = 29.333%
Food: 6 + 9 + 4 + 7 + 6 = 32/75 = 42.666%
Sport: 8 + 6 + 6 + 3 + 6 = 29/75 = 38.666%

2,250 tweets to train (750 per topic), only over 1 distinct classifier:
Music: 12/15 = 80%
Food: 6/15 = 40%
Sport: 6/15 = 40%

Analysis

Analysis - Sentiment Analysis

After analyzing our results graphed above, we are able to see each user's positivity score:

University of Minnesota student 1: 36
University of Minnesota student 2: -1
University of Minnesota student 3: -10
University of Minnesota student 4: 37
University of Minnesota student 5: 20

While we are pleased with our results, there are a number of things that could have been done differently in order to improve accuracy. One example would be the user's tweet count. We set a maximum number of tweets for better results, but we did not set a minimum. University of Minnesota students 2 and 4 both had less than 60 tweets to account for, while the other three students reached the 100 tweet maximum. It is possible that student 2 and 4 may have had different results if they had more tweets to analyze. If we had the time and resources, I think analyzing 500 tweets per user would produce more accurate results.

Analysis - Text Classification

The difference between the 75 and 150 tweets to train seems negligible, because the performance in detecting music and sports tweets both declined by a fair percentage. This is most likely due to there being a larger gap between the information presented in the training data, and the information presented in the test data. However, we see a slight general increase across the board for the 2,250 tweets to train, likely because there was a more robust lexicon present throughout all the training data.

The Bottom Line

With just this data, it is difficult to come to any conclusions about the success rate of our model/ approach (using random recent tweets with certain hashtags for a given topic as training data). The 2,250 tweets to train seems somewhat promising in showing us that a larger training set could prove more useful in topic categorization, but because smaller training sets on average over more distinct classifiers were able to outperform this in some cases, we cannot conclusively say that this training set would be able to train a classifier to outperform others.

With a larger test set of manually classified tweets, we may have seen more significant and meaningful improvements in topic categorization. Additionally, another point of possible improvement is if we had been able to successfully filter out stop-words. If we had chosen a different approach/model (i.e chosen a different way of creating our training set) we may have also seen improvements.

Conclusion

Limitations

Throughout the process of completing this project, we came to realize there are many roadblocks and limitations in performing sentiment analysis and topic categorization. Here are a few of the following, in descending order from the biggest problems to the smallest.

Working with small data sets for both the training data and test data

Many machine learning algorithms require a lot of input data to be accurate. Because of the rate limits of the API, we were limited to only being able use 15 API calls in a 15 minute window [8]. It is unclear to what the limit is on calls to the Tweepy library's search function, as we were able to gather 750 tweets per category topic in one trial, and hit the rate limit in a different instance when trying to gather 1,000 tweets per category topic. The official Search API documentation states that you can have up to "...450 queries/requests per 15 minutes on its own behalf without a user context."[9]. With a clever solution, you could theoretically keep waiting to download more tweets to a text file in order to feed a classifier more input data, but this is still a limitation nonetheless. The results may have been more interesting and accurate if we could have trained our naive bayes classifier on a corpus of 10,000 or more tweets. Additionally, our test set of data of manually classified tweets was extremely small at only 15 tweets, which explains why we did not have more than three categories (it would have been painfully time consuming to label more test data). To manually classify more in a reasonable amount of time, we would need paid volunteers to participate in a research study.

One word tokens from NLTK

During tokenization of tweets in an attempt to remove stop-words, we came across multiple tweets where the only token created was the tweet itself. We used NLTK's word_tokenize function for this, but it did not prove itself to be a very stable or reliable function. Not wanting to exclude entire tweets from our analysis, we decided not to use the NLTK function. Again, our results may have been more accurate if we had been able to filter out stop-words.

Word-Sense Disambiguation

Word-sense disambiguation deals with the problem of identifying a semantic meaning for a particular span of text [16]. Simply put, this deals with the issue of how we should interpret any given word in its context. For example, the word 'spring', given its context, could mean "the season where flowers grow". In another context, it could mean "the object that is in mattresses and mechanical structures". Thus, its semantic value is ambiguous without understanding the context, and the task of disambiguating it would require that understanding. Creating an algorithm to make this distinction is objectively very difficult, because it has been an open problem in natural language processing for almost as long as the field has existed [16]. There weren't any ways that this directly affected our work, but it is a fairly complex and immediately relevant problem.

Processing power

One of the most frustrating points of the project was waiting for the naive bayes classifier to train over the input data. Over 2,250 tweets, the classifier took 34:06 to complete, using nearly 100% of the cpu on an i7 processor throughout training. If we would have had more input data, it may have easily taken over a day to compute. If we could have more processing power (access to a server or better computer), it may have taken less time to train, although the majority of the training time could have been from the run-time complexity of the algorithm itself.

Closing Words

Overall, we were not able to be conclusive about our results from text-classification. If we were to conduct our experiments again, we would first start by seeking a more robust tokenizer that would be able to handle data from tweets. After conducting a new experiment on that, we would try to choose a different model for how we gathered tweets for the training set, possibly by using a feature extractor function. On the other hand, we were able to correctly see and analyze the results from Sentiment Analysis, and we would have done nothing different. To cover this topic more thoroughly, this would be a project best suited for an internship.

Team Member Contributions

John Saja

Picked topic for project
Solely responsible for the text-classification approach
Wrote introduction, conclusion, a portion of the approach section (How?) text classification portion of the experiment design, results, and analysis sections.

Ian Santon

Collecting tweets using Tweepy
Head member for the Sentiment Analysis approach
Wrote all parts of the Sentiment Analysis for the experiment, design, results, and analysis section
Wrote portions of the approach section (Methods)

Pawandeep Singh

Created Latex
Synthesized everyone's sources
Assisted Ian in the entire process of Sentiment Analysis
Wrote portion of the approach section (Goals)

References

[1] N. Alsaedi, P. Burnap, and O. F. Rana. A combined classification-clustering framework for identifying disruptive events. 2014.

[2] S. Asur and B. A. Huberman. Predicting the future with social media. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on*, volume 1, pages 492–499, Aug 2010.

[3] J. Bollen, H. Mao, and X. Zeng. Twitter mood predicts the stock market. *Journal of Computational Science*, 2(1):1 – 8, 2011.

[4] M. N. K. Boulos, A. P. Sanfilippo, C. D. Corley, and S. Wheeler. Social web mining and exploitation for serious applications: Technosocial predictive analytics and related technologies for public health, environmental and national security surveillance. *Computer Methods and Programs in Biomedicine*, 100(1):16 – 23, 2010.

[5] V. Dhar. Data science and prediction. *Communications of the ACM*, 56(12):64–73, 2013.

[6] M. S. Gerber. Predicting crime using twitter and kernel density estimation. *Decision Support Systems*, 61:115 – 125, 2014.

[7] J. Golbeck, C. Robles, and K. Turner. Predicting personality with social media. In *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '11, pages 253–262, New York, NY, USA, 2011. ACM.

[8] T. Inc. Api rate limits, 2016.

[9] T. Inc. The search api, 2016.

[10] J. F. H. Jr. Knowledge creation in marketing: the role of predictive analytics. *European Business Review*, 19(4):303–315, 2007.

[11] W. A. Kamakura, M. Wedel, F. de Rosa, and J. A. Mazzon. Cross-selling through database marketing: a mixed data factor analyzer for data augmentation and prediction. *International Journal of Research in Marketing*, 20(1):45 – 65, 2003.

[12] H. Kinsley. Pythonprogramming/nltk-3—-natural-language-processing-with-python-series.

[13] N. Landwehr, M. Hall, and E. Frank. Logistic model trees. *Machine Learning*, 59(1):161–205.

[14] B. Larivière and D. Van den Poel. Predicting customer retention and profitability by using random forests and regression forests techniques. *Expert Systems with Applications*, 29(2):472–484, 2005.

[15] S. Liu and J. Yap. Method and apparatus for identifying cross-selling opportunities based on profitability analysis, Nov. 7 2001. US Patent App. 10/008,731.

[16] R. Mitkov. *The Oxford handbook of computational linguistics*. Oxford University Press, 2005.

[17] C. Nyce and A. CPCU. Predictive analytics white paper. *American Institute for CPCU. Insurance Institute of America*, pages 9–10, 2007.

[18] E. Quincey and P. Kostkova. *Electronic Healthcare: Second International ICST Conference, eHealth 2009, Istanbul, Turkey, September 23-15, 2009, Revised Selected Papers*, chapter Early Warning and Outbreak Detection Using Social Networking Websites: The Potential of Twitter, pages 21–24. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[19] J. R. Quinlan et al. Learning with continuous classes. In *5th Australian joint conference on artificial intelligence*, volume 92, pages 343–348. Singapore, 1992.

[20] E. K. Steven Bird and E. Loper. Natural language toolkit.