**Project Contributors**

**Christian DeMarco**

Student ID: 001220184

Email: cdemarco@albany.edu

Department: College of Engineering and Applied Sciences

**Christopher Snyder**

Student ID: 001178681

Email: cjsnyder@albany.edu

Department: College of Engineering and Applied Sciences

**Jonathan Negron**

Student ID: 001169822

Email: jnegron414@gmail.com

Department: College of Engineering and Applied Sciences

***Abstract -* We use a Twitter rest API to gather a large sum of tweets related to a query of car manufacturers. The tweets are then filtered, extracting only the tweets that actually relate to the car manufacturer from our query. In doing this we can analyze the data and mine it for notable patterns that may be helpful to marketing experts.**

1. **Introduction**

The commercial viewpoint of data mining has always been a significant part of data mining. There will always be businesses trying to sell a product and those businesses will always want to know more about the consumers of their products. One of the industries that always want to know more about their consumers is the car industry. The more that car companies know about their consumers the more they know about how to sell their product to those consumers and make more money. Our team is driven to build a model that collects a vast amount of tweets based on a query of car manufacturer names using the Rest API in the Tweepy Python client and discover patterns from this data that car manufacturers may find useful using data mining techniques that we learned from class.

The problem is how can we effectively gather thousands of tweets related to our research and discover patterns and trends with accuracy. The query we used to gather our tweets are the names of the car manufacturers we are conducting research on. These manufacturers include, Volkswagen, Honda, Toyota, Nissan, Audi, BMW, Hyundai, Chevrolet, GMC, Mitsubishi, Jaguar, Lincoln, Subaru, Mercedes, Jeep, Acura, Ford, Dodge, Land Rover, Cadillac, and Chrysler. However, using these names as a query, many tweets we collect will not actually be related to cars. For example, when collecting tweets with the query as “audi”, our API would collect tweets that contain words such as “audit” or “audition” which is not relative to what we want in our data. Our first problem is to filter these tweets and classify them as positive or negative. Positive being that they relate to the vehicle manufacturers that we are using as a query, and negative being that they are of no relation to these vehicle manufacturers. This is significant because if our data set contains an abundance of negative tweets then anything we conclude about this data set will be invalid. Our next problem is discovering patterns and useful information about these tweets. This is significant because this is the focal point of our research. Without this problem we have no reason to be mining the data that we have.

1. **Related Work**

Indian Pythonista is a similar work to us where they collect tweets and parse through them to mine sentiment score on tweets. Like us, they use the Tweepy client to collect their tweets. The point of their project is related to the commercial viewpoint of data mining. Companies use sentiment value to develop strategies and gain an understanding as to why consumers may or may not like something and what consumers want in their products. This is similar to our work because we are also trying to mine information out of tweets to discover useful information on the tweets we gather to help companies better understand their consumers. Our work is closely related to text classification which is a very much studied field in data mining.

1. **Proposed Approaches**

We spent a lot of time trying to figure out how we were going to collect accurate and relevant positive tweets on each of the car manufacturers used in our query. Our first step was to gather tweets. We decided to approach this problem by using the Tweepy rest API. By using this python client we can easily collect as many tweets as we want using any query we want. Once we have done that, our next approach was to label the tweets we have collected from the API as positive or negative. It is essential for us to classify these tweets as positive or negative because it is in our best interest to only use positive tweets when analyzing our data or else we cannot draw any conclusions. We knew that if we wanted to classify these tweets as efficiently as possible then we would have to find a way to have the computer classify the tweets for us. We decided the best way to approach this was to use classification data mining by building a support vector machine to figure out a way we can have our code dynamically give a positive or negative label to a data set of unlabeled tweets of any size. It is extremely time consuming to manually label thousands of tweets as positive or negative, this approach we thought was best for our project. In doing this approach, we will have created a very useful prediction method to classify tweets. This will also give us useful information on the patterns of keywords related to the car manufacturers that match our query. And we can extract relative statistics on those keywords and on our data. Once we have our support vector machine working, we can calculate how effective it is by calculating the three metrics, API recall, quality precision, and quality recall. This will help us further understand how well our model works. Once we have done that we needed to decide what kind of analysis we were going to conduct with our data. We had many approaches we could have done. We could conduct tweet level analysis, where we can turn each tweet into a vector of keyword frequencies, we could conduct user level analysis where we turn each user into an object and convert their tweets into vectors of keyword frequencies, or we can do a city level analysis, which is what we decided to do, where we turned each city into an object and merge all the tweets collected from a city into one document and turned the text into a vector of keywords. This will provide us with useful information based on the patterns of our data. Another data mining technique that we wanted to utilize was clustering. We thought that it would provide us with a lot of insight into the patterns in our data. We decided clustered our tweets at a city level as well. This means that similarly to the support vector machine, we wanted to convert each tweet in each city into a vector, and then cluster them based on their cities and we can then examine each cluster of each city to find some useful information. After executing this approach, we should have files relating to the frequencies of keywords of positive tweets in each city, and clusters of tweets related to each city. Once we have this information we should be able to examine it to draw a conclusion on some patterns related to the consumers of these car manufacturers.

1. **System Design and Implementation**

Our system and design can be classified into multiple modules, Car Query, Score Tweets, Support Vector Machine Model, Calculate Metrics, Get Frequency, Clusters. Each of these modules fulfills a certain duty in efficiently executing our proposed approaches. While these are not the only modules used, they are the main modules. Some of these modules leverage smaller helper functions that we created to make our code more readable and easier to maintain and debug.

**Car Query**

This module fulfills the duty of collecting a vast amount of tweets. The parameter of this function is an integer. This integer is passed into our API search and tells the API how many tweets to collect. This module uses a global array of cars that we want to use for our query and a global dictionary of cities with their corresponding latitude and longitude to collect tweets from a specified city of our choice. We decided to collect tweets from four different cities in America. These cities include, New York City, Miami, San Francisco, and Albany. The output of this module are multiples files with each file name being the name of the city and each file containing tweets from that city. We decided to limit our API to only collect tweets that do not contain the string “RT @”. We decided on this design because without it, we were collecting a lot of repeated tweets from different users who were all retweeting one tweet. We also decided to convert every tweet to lower case to keep our data consistent for analysis.

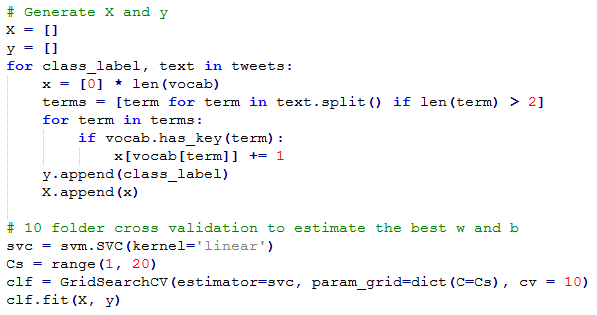
**Score Tweets**

This module is used to take the file created from Car Query, which is a large list of tweets, and ask the user if each tweet in that file is positive or negative. This module was created in order to speed up the process of manually labeling each tweet as positive or negative. Upon each tweet being read from the file, the tweet will then be printed to the console and wait for a user input of 1 (positive) or 0 (negative). This module was helpful in building our training set in a short amount of time.

**SVM Model**

This module is the guts of our system and design. This module will take in four parameters, a file of the tweets that we labeled from the previous module (this is the training set), a file of new unlabeled tweets (this is the testing set), and an output file to print all the positive tweets that were predicted from the testing set. This SVM creates an array and fills this array with each tweet from the training set. The next step is where we convert out tweets into vectors of keyword frequencies. A dictionary is created and the dictionary is filled based on the data inside the training set. The key for each element of the dictionary represents a word in the file and the value of that key is how many times that word appears in the text. However, our system is designed to only consider words which are *not* inside of our “stopwords” array which is an array of neutral words which have little to no significance to our research. Words such as “the”, “is”, “and”, etc… Once this is done we generate our X and Y vectors based on these keyword frequencies and perform a 10 folder cross validation to estimate the best w and b for the SVM. The following code in *figure 1* will demonstrate exactly how this is done.

*Figure 1*

**

Once this is done we can now predict the class label of any future tweets. So we collected a vast amount of unlabeled tweets from each city, Albany, New York City, Miami and San Francisco. These unlabeled tweets were stored in a text file named by their city names. The names of these text files are, alb.txt, nyc.txt, mia.txt and sf.txt. Now that our SVM is working we can run these unlabeled tweets into the SVM and extract only the positive tweets. We stored these positive tweets in a text file based on their city names, albpos.txt, nycpos.txt, miapos.txt, fspos.txt. These files are essential for our project because these will be the file containing only the positive tweets that we want to analyze and search for patterns. We take the tweets from this file and paste them into a word cloud to analyze the most frequent words as a visual representation of data. The following word cloud in *figure 2* represents the most frequent words in the positive tweets from Miami.

*Figure 2*

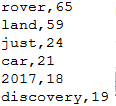


It is clear by this word cloud that “land rover”, “discovery” and “car” are some of the most frequent words in our positive tweets for Miami. We can evaluate the validity of this word cloud by executing a function from our next module, get frequency.

**Get Frequency**

This part of our code is similar to the first part of our support vector machine. We use the same method as we did in SVM model by reading in each tweet from the input file. In this case our input file will be the files associated with all the positive tweets from each city. The tweets from these files are then converted to a vector of keyword frequencies using a dictionary. Once the dictionary is complete, we just print out the most frequent tweets from the dictionary. In *figure 3* you will see a list of the most frequent words in miapos.txt. This list is stored in a text file for each corresponding city named, “miafreq.txt”, “nycfreq.txt”, “albfreq.txt” and “sffreq.txt”.

*Figure 3*

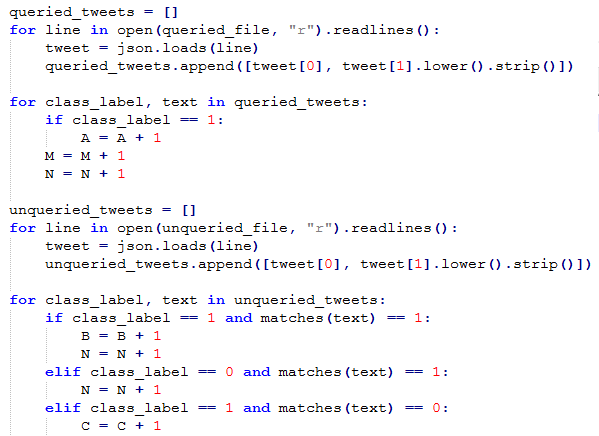


As you can see, this confirms the validity of our word cloud. The most common words from out positive tweets form Miami are “rover”, “land”, “just”, and “car” amongst others.

**Calculate Metrics**

This part of our code is where we test how well our support vector machine works. A support vector machine is considered to work well if it accurately classifies each item as they should be. We can measure how well the machine achieves this by calculating three key statistics, API recall, quality precision and quality recall. These three metrics are all related to M, retrieved tweets from the API, A, positive tweets inside of M, N, tweets that match our query from a random sample, B, positive tweets in N, and C, positive tweets that did not match our query. The calculate metrics function works by taking in one labeled file of tweets from our query, and one labeled file of random tweets as a parameter. The labeled tweets from our query are read line by line and if the tweet is positive then we increment |A|, then regardless of the label we increment |N| and |M| as these tweets will be subsets of |N| and |M|. Then we read each tweet from our file of random tweets line by line. We leveraged a helper function “matches” that reads in a tweet and returns 1 if the tweet matches our query and returns 0 otherwise. If the tweet is positive and matches our query we increment |B| and |N|. If the tweet is negative and matches our query then we only increment |N|. If the tweet is positive and doesn’t match our query then we increment |C|. In this case we do not care about calculating |D| because it is not used to calculate metrics. *Figure 4* will demonstrate exactly how this is done.

*Figure 4*



After this runs we can easily calculate the metrics as follows. These are our actual results.

API recall = |M|/|N| = 17%

Quality precision = |A|/|M| = 27.4%

Quality recall = |A|/(|A|+|B|+|C|) = 5.4%

Our API recall was reasonable. This means that we did a good job at collecting and abundance of tweets that match our query. Our quality precision was about 25%. This means that of all the tweets collected form our API only 25% of those tweets are positive. Considering how fast we can gather a large amount of tweets, we think that this quality precision is good and we can easily collect a vast amount of positive tweets. The most important metric, quality recall, was about 5.4% for us. This is a reasonable quality recall and we were pleased to see it.

**Clusters**

Clustering works similarly to the SVM model as it converts tweets into vectors. The difference is that the SVM model will predict the label of each tweet and classify them as positive or negative. While the clustering method will organize the tweets into groups or clusters based on the similarity of their attributes. We leveraged K-means clusters into 5 clusters each and tried 100 different initial centroids. Each cluster was written to a file which allowed us to analyze each cluster for patterns.

1. **Conclusion**

We are all in agreement that this project has made us better programmers and was a very effective first step into the world of data mining. We learned a tremendous amount about gathering large sum of data and how to analyze and manipulate it while using advanced techniques to discover interesting trends and patterns. These trends and patterns are something that a human being could not easily recognize by eye. I felt that the techniques and skills learned from this project are relevant to real world problems.

Ultimately, we felt that the trends and patterns that we discovered from doing this project were not conclusive enough to be useful to a car manufacturer. We noticed that in all of the cities we gathered tweets from, “land” and “rover” seemed to be the most positively gathered tweet. Therefore in all of the city frequency files, “land” and “rover” are the most frequent word. This may be because the Land Rover vehicle is just the most tweeted about car on twitter, which in that case, would be useful information for the Land Rover company to know because then they should consider doing more marketing and advertising on Twitter. However, we do not think our results are conclusive enough to assume this.

Some ways we could improve our design is we could have gathered more tweets for our training set. Our training set was about 1,000 tweets which many of them were repeats. If we collected more tweets and modified our design to do a better job at limiting the tweets to only unique tweets, then it would have more accurately predicted the label of our queried tweets.

We found that some tweets that related to different car manufacturers were more abundant than others. For example, when collecting tweets based on the query, “ford”, most of the tweets were about Harrison Ford. So there were not a lot of positive tweets related to the Ford vehicle company. While some queries such as, “Land Rover”, were abundant in positive tweets. We could have done a better job at making sure an equal amount of positive tweets per car manufacturer were collected to keep our data set balanced.

One thing we would do differently is try to focus more on sentiment score of the tweets related to the car. If we focused more on the sentiment of the tweet, then we could derive the opinion the consumer has about each car which is something that would be useful for car companies to know.

There was room for improvement but we still feel that the data we collected and the techniques we used were relevant as an exercise for data mining. The techniques and skills we learned form this experience will be very useful to us in the computer programming industry.

**References**

[1] Python Tweepy Documentation

<http://docs.tweepy.org/en/v3.5.0/>

[2] IndianPythonista “Twitter Sentiment Analysis with Python”

<https://indianpythonista.wordpress.com/2017/01/24/twitter-sentiment-analysis-with-python/>

[3] Twitter Sentiment Classification using Distant Supervision

Alec Go, Richa Bhayani, Lei Huang