Chris Johnston
Sarah A. Cantu
https://github.com/NumbersAndStuff/STAT689-Project/

Abstract

A Kaggle dataset of alleged Russian troll tweets was explored with several different classification techniques and models. A bag of words model was used to initially explore the data. A naive Bayes model was used to classify tweets with high interest terms and this sentiment classification was used to explore any linear regression and autocorrelation in the data. Next, a one-hot model was created with another Kaggle tweet corpus and a decision tree classifier was used to determine sentiment of the troll tweets. Finally, the decision tree was expanded into a random forest to directly compare it with the decision tree algorithm.

It was found that the bag of words model, while not versatile, is easy to implement and provides immediate information about the dataset being explored. The naive Bayes method was found to have an apparent bias toward neutrality, while the decision tree and random forests have an apparent bias toward positive sentiment. Both of these biases are likely due to systematic error introduced by the students.

Introduction

The datasets used in this project were found in Kaggle, a platform for competitions in machine learning and statistical analysis. It is a rich resource of large datasets and challenges. Recently, there was a challenge based on alleged Russian troll tweets. The dataset consists of about 200,000 individual tweets from about 3,000 twitter accounts that were traced back to a propaganda organization linked with the Kremlin. The tweets were first deleted by the company, Twitter, and then recovered by NBC News investigators. According to an NBC News article on the subject, the organization responsible for the twitter accounts has been "assessed by the U.S. Intelligence Community to be part of … [an] effort to influence … the 2016 U.S. presidential race" (NBC News). Regardless of the original intent, the dataset is now publicly available. This type of troll activity is what help propagate the rampant "fake-news" on social media now and during the election. Primarily, the alleged purpose of the tweets was to push a racist, conspiracy theory filled agenda that targeted entities such as the democratic party and the Black Lives Matter movement.

A variety of natural learning process (NLP) analyses could be done with a dataset with this much information, however, the focus in this project was on a general text classification and sentiment analysis. NLP is an area of computer science that has been around since the 1950's. NLP algorithms are often used to translate languages and filter content by classifying the input document. Originally, classification was based off of handwritten notes. With the advent of

machine learning algorithms, NLP advanced in complexity and speed. Machine learning combined statistical methods and computational power and revolutionized NLP. The NLP and machine learning algorithms explored in this project are Naive Bayes, Bag of Words, Decision Trees, and Random Forests.

Pre-Processing

Before any text data can be classified it has to be preprocessed. Tweets in particular are full of urls, acronyms, and purposeful misspellings. One pertinent point that was not addressed in this study, is to account for emoticons which are prevalent social media/twitter and carry a heavy sentiment implication. Two different methods were used in this project, both of them using python packages; NLTK and SpaCy were the primary methods used to clean and tokenize data. There are benefits and drawbacks to both methods.

When filtering for running TextBlob and preliminary analysis, we used a combination of NLTK and regular expressions to clean and preprocess the data. The regular expressions used filter out URLs, links to other user accounts, and hashtags. Once the tweets have been cleaned, they are passed into the NLTK tokenizer in lower case, to prevent capitalizations from affecting the results. The NLTK tokenizer removes inconsequential words from the tweets that serve only to join clauses together and helps to catch anything the regular expressions missed. The tokenized data was then used for the exploratory analysis and for the sentiment analysis using TextBlob.

SpaCy has the benefit of being relatively easy to implement and the output works well with many models and algorithms. It carries with it the option to use a variety of different language models to lemmatize the data. For this study, and based on documentation, a smaller english model was chosen to find appropriate lemma values for the tokens. This was to avoid overfitting, as can be a problem in many analytic projects. The tweets were first read in, with pandas, as raw data that included stop words, misspellings, urls, special characters, etc. Using a multi-threaded for-loop, we were able to utilize SpaCy to tokenize the tweets, lemmatize them, pull out parts of speech tagging, and dependency parsing. The only information kept for the purposes of this project were the lemmatized tokens. The lemmatization, implemented and discussed later in the Decision Tree and Random Forest algorithms, allowed for uniformity among words with misspellings, plurality, and conjugation.

Bag of Words

At the foundation of this entire project is the Bag of Words (BoW) model. BoW provides a simple way to model text both for simple analysis purposes and for feature extraction to input into machine learning algorithms. Both purposes are used in this project. Fundamentally, BoW counts a corpus vocabulary of known words and measures the presence of those words in each document.

The initial data exploration was done in this manner. The tweet corpus was treated as one document and the total vocabulary explored. Ignoring words that had less than two characters and repeated less than 100 times over the corpus, a list of the top words was visually inspected to select keywords to focus on for the succeeding analysis. The visual inspection cut words that were tokenized as partial contractions or common, e.g., man or one, and were chosen to be relevant to the political nature of the dataset. These data cuts resulted in a list of 23 words (see Figure 1) that are used in the subsequent analysis. Some examples will be provided in this document, however for the full set of discussed plots, please see the original notebook located in the project github, ProjectNotebook.ipynb.

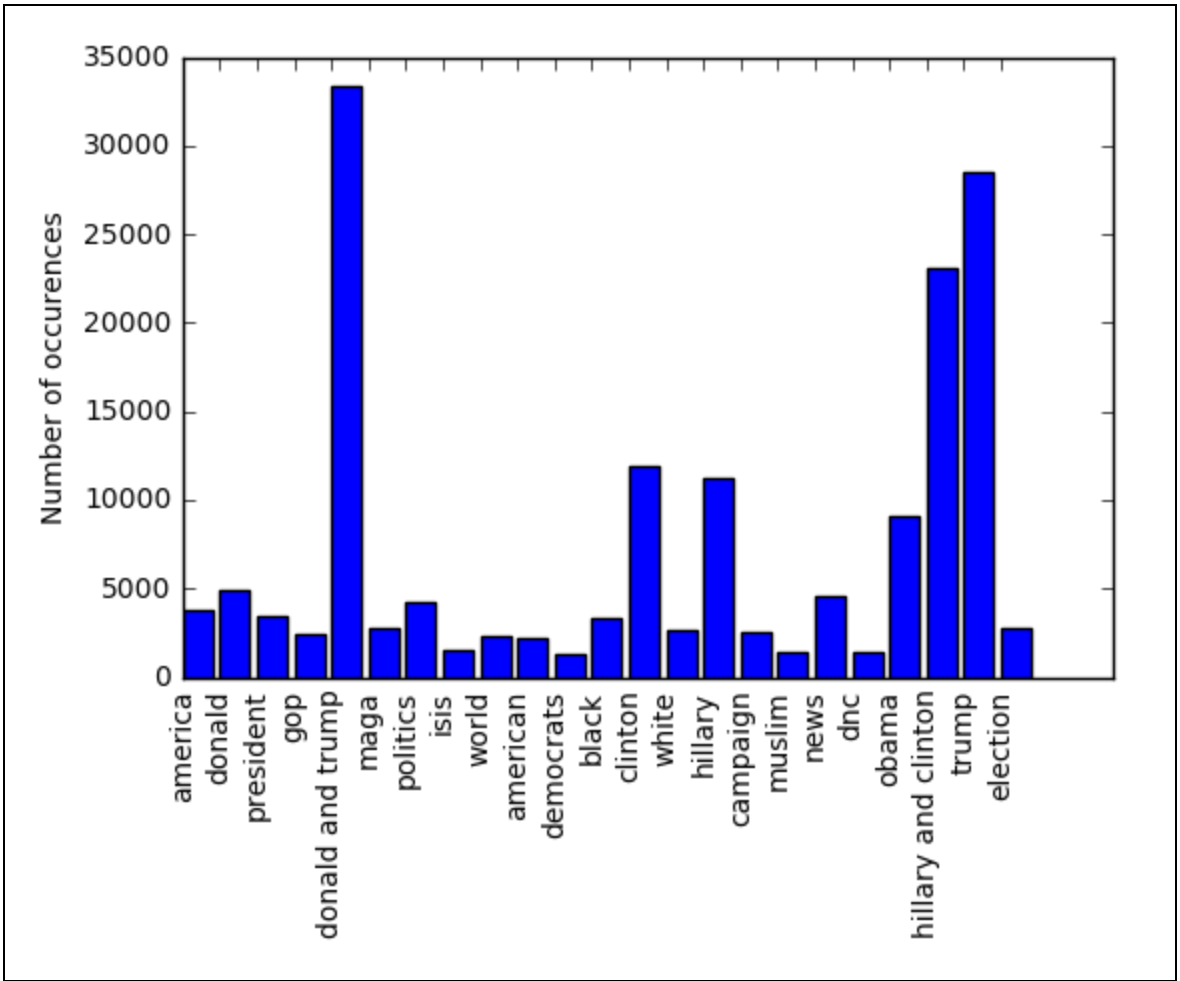

Figure 1. Histogram of number of tweets containing keywords.

Once the words of interested had been selected, we produced plots showing the number of tweets containing each word on a given day. Many of these plots, such as the plots below, show spikes during primary season, approximately in March 2016, as well as spikes during the GOP

and DNC national conventions. Below, are plots showing when tweets containing the keywords “gop” and “black” were posted. More plots can be found in ProjectNotebook.ipynb.

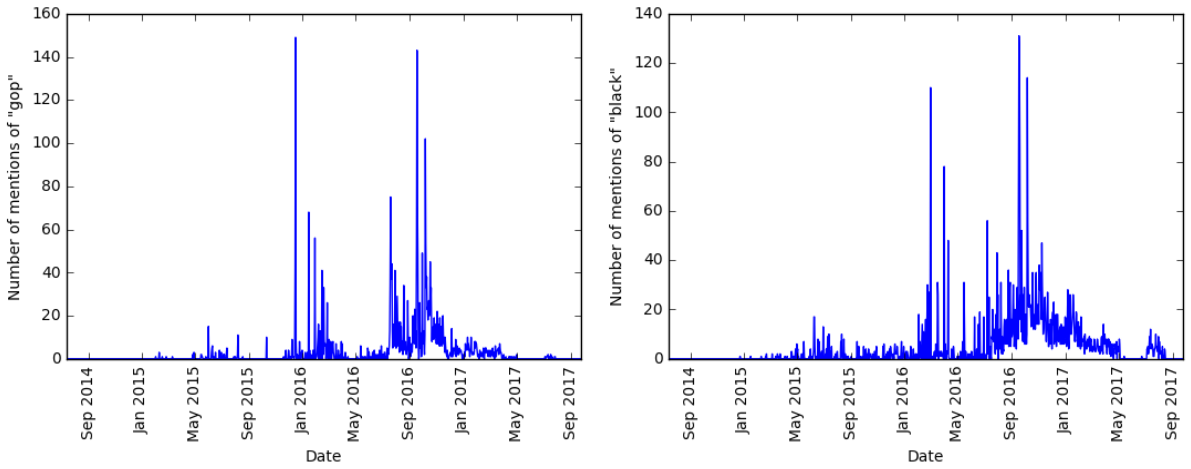

Figure 2. Plots of tweets containing “gop” vs date and “black” vs date.

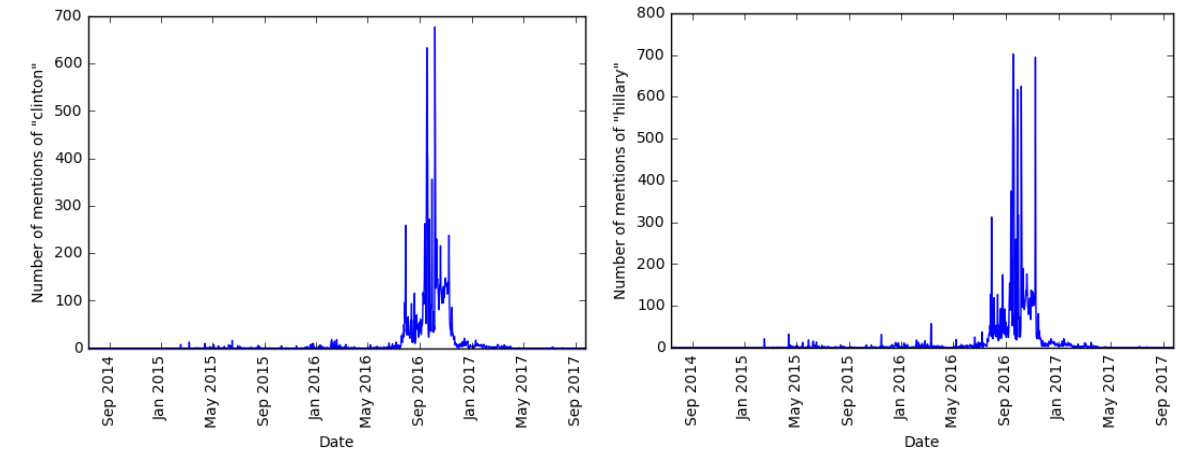

Figure 3. Plot of tweets containing “clinton” vs date and “hillary” vs date..

With these plots, we noticed that there was increased activity across the board in the months from July 2016 through January 2017, so we narrowed our plots down to these regions. A particularly notable plot, is the plot of tweets containing “clinton”, shown above. It shows very few mentions of “clinton” or “hillary” prior to July 2016, and a steep drop-off prior to January 2017 in mentions.

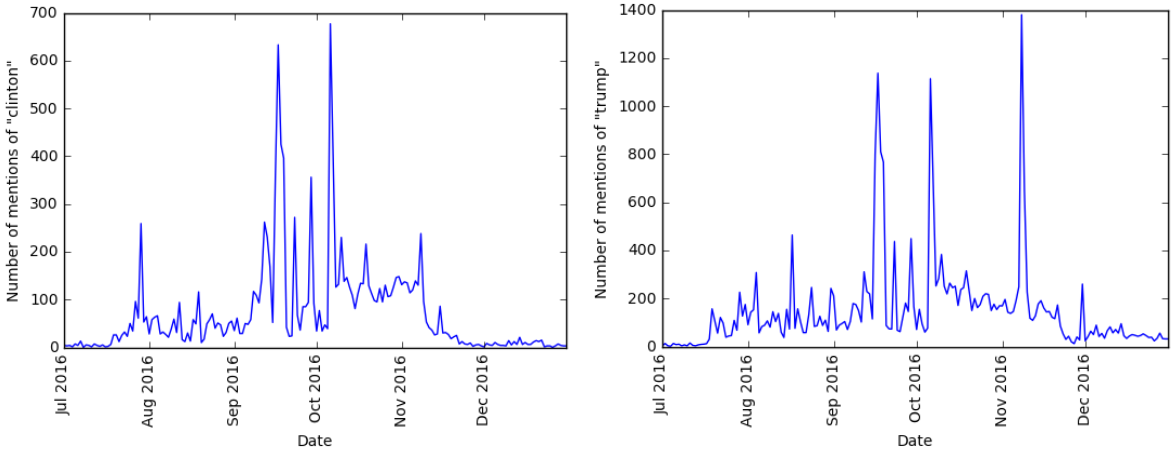

Figure 4. Plot of tweets containing "clinton vs date and "trump" vs date, narrowed down to July 1, 2016 through December 31, 2016.

In the above plots, there are spikes toward the end of July, mid-to-late September, early October, and November. The spikes in late July correspond with the DNC and GOP national conventions, the spikes in September to the first presidential debate, the spike in October to the second presidential debate, and the spike in November to Election Day.

Naive Bayes

The Naive Bayes model is also known as the *idiot Bayes model*. This is due to the simplistic nature of the model. It applies Bayes' theorem with the "naive" assumption of independence between model features. Often, but not necessarily every time, the naive Bayes classification model is used in conjunction with maximum likelihood algorithms. Literature values place standard naive Bayes algorithms at about 0.5-0.74 accuracy depending on the level of preprocessing and n-gram that is used. In this project, we used a python package called TextBlob to explore the naive bayes algorithm. TextBlob implements two different classifiers, if specified, however the second classifier is an SVM algorithm and for the purposes of this project was deemed to be too computationally expensive and over-kill for tweets. TextBlob is trained with a movie review corpus on positive and negative sentiment. We used the incorporated sentiment analysis function to classify the polarity of individual features in each tweet document. This was then used to compute a sentiment for each tweet, ranging through negative, neutral, and positive sentiment.

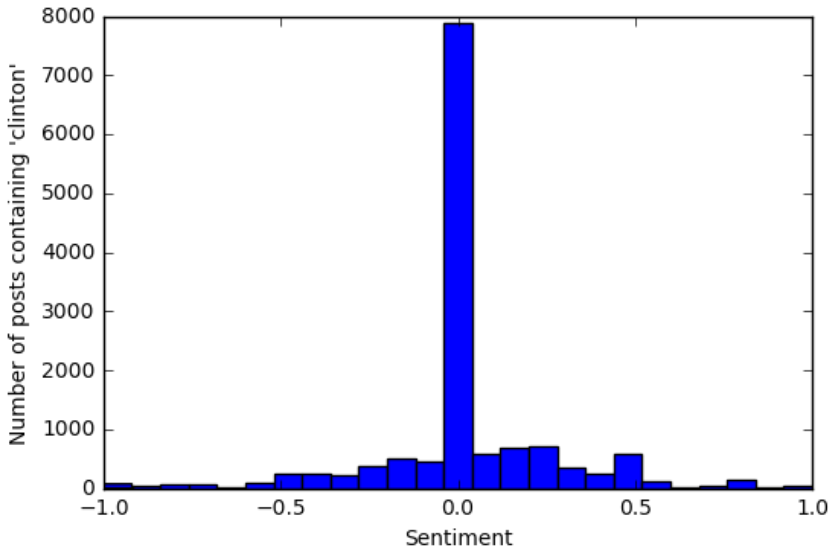

Figure 5. Histogram of tweets containing “clinton”.

Figure 5. above is a histogram of sentiment of tweets containing “clinton” found using the Naive Baye’s classifier from TextBlob. Most tweets are identified as neutral, with 26.5% of tweets being positive, 18.3% negative, and 55.1% neutral. Below, Figure 6. is a histogram of sentiments of tweets containing “clinton” but not “trump”. Of these, 26.3% are positive, 18.8% are negative, and 55.4% are neutral.

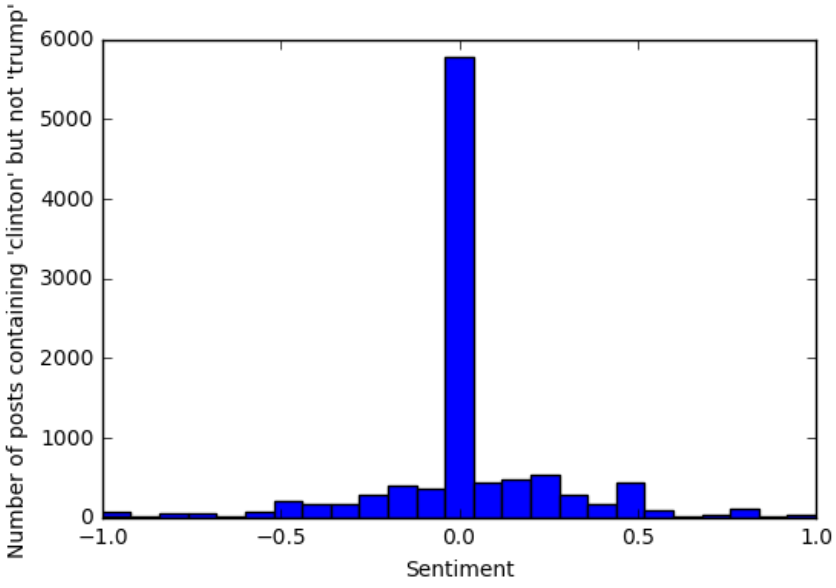

Figure 6. Histogram of tweets containing “clinton” but not “trump” or “donald”

As a point of comparison, Figure 7 is a histogram of tweets containing “trump” but not “clinton” or “hillary”. This criteria give a positive percentage of 31.6%, a negative percentage of 18.8%, and a neutral percentage of 49.6%.

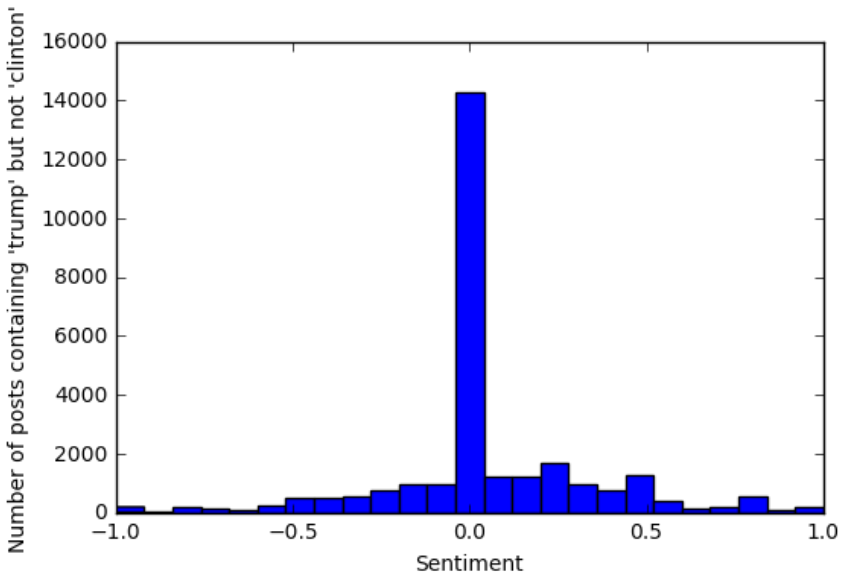

Figure 7. Histogram of tweets containing “trump” but not “clinton” or “hillary”

We attempted to find a linear trend in sentiments of tweets containing our keywords by producing plots of the average sentiment for a given day, using the day as a predictor. When we did this, we found no significant results: a hypothesis test always failed to reject the hypothesis that there is no trend. When this failed, we introduced an indicator variable which takes the value 1 if the date falls between July 1st, 2016 and December 31, 2016, so that the full model became $y = \beta_0 + \beta_1 x + \beta_2 i_{ElectionSeason}$. When regressing against this model, the results were still negative. Then, we can conclude that there is not a linear, non-constant trend to the selected sentiments. The scatter plots can be viewed in ProjectNotebook.ipynb.

Decision Trees

Decision trees are so called due to their flowchart like illustrative graph. Decision trees are commonly utilized for regression analysis and classification. The beauty of decision trees is their “white box” aspect. White box refers to the transparency of the inner workings of the algorithm. For our project, the output of the SpaCy preprocessing and python package, scikit learn, was used. Decision trees rely on user-defined features that act decision nodes in which the algorithm determines whether some criterion is met. Our decision tree uses the Gini Impurity as this criterion, which the decision tree seeks to minimize. It quantifies how often a random

element might be labeled wrong if it were labeled using that branch's distribution. The other criterion typically used is known as Information Gain, which seeks to minimize entropy.

As the classifiers in this package do not come pre-trained with some dataset, another kaggle dataset with pre-classified sentiment was found and split for the purposes of training and testing. This new dataset contains about 100,000 tweets that were split into a training set of 80,000 and a test set of about 20,000. These dataframes, along with the troll dataframe, were prepared for the classification algorithm by vectorizing and normalizing the tweets. The vectorization is essentially an implementation of the BoW model discussed earlier, in addition to assigning an integer key to each unique term in the corpus. These vectors are then transformed with tf-idf (term-frequency inverse document-frequency) weighting scheme and normalized by a Euclidean norm. This is done to account for the low importance, high frequency terms and normalize the test and troll datasets to the training set.

For our purposes, we used the output of the BoW as is, meaning we populated our feature space with all the unique terms found in the corpus. This gives a vocabulary of about ~80,000 features and two targets; 0 for negative sentiment and 1 for positive. In terms of computational expense, the tree took about 1.5 minutes to run, a few seconds to test, and another minute to classify the troll tweets. The resulting accuracy, using statistics provided by the package and evaluated on the pre-classified test set, was ~0.68 with a bias toward positive.

Random Forests

Random Forests are an extension of the Decision Tree algorithm. One of the major drawbacks to decision trees is that it is prone to overfitting. This can be handled with pruning, but pre-pruning requires some effort due to avoid pre-pruning too soon and post pruning requires extensive cross-validation that comes with its own issues. Another way to deal with this is to use a random forest algorithm. Random forests use a bootstrapping technique to create a random subset of the features of interest during the decision making process. The set up for them is almost identical to decision trees and for the same reasons. Here, we took the same prepared features, targets, and prepared datasets and ran them through a forest of 100 trees. In comparison with the decision tree, on the same computer, the random forest took just over six minutes to train, about 15 seconds to test, and another couple of minutes to classify the troll tweets. The resulting accuracy was 0.75, again with a slight bias toward positive. A better visualization on the accuracy of the random forest is a confusion matrix as shown in Figure 8. This clearly shows the high number of true positives and comparatively high number of positives in general.

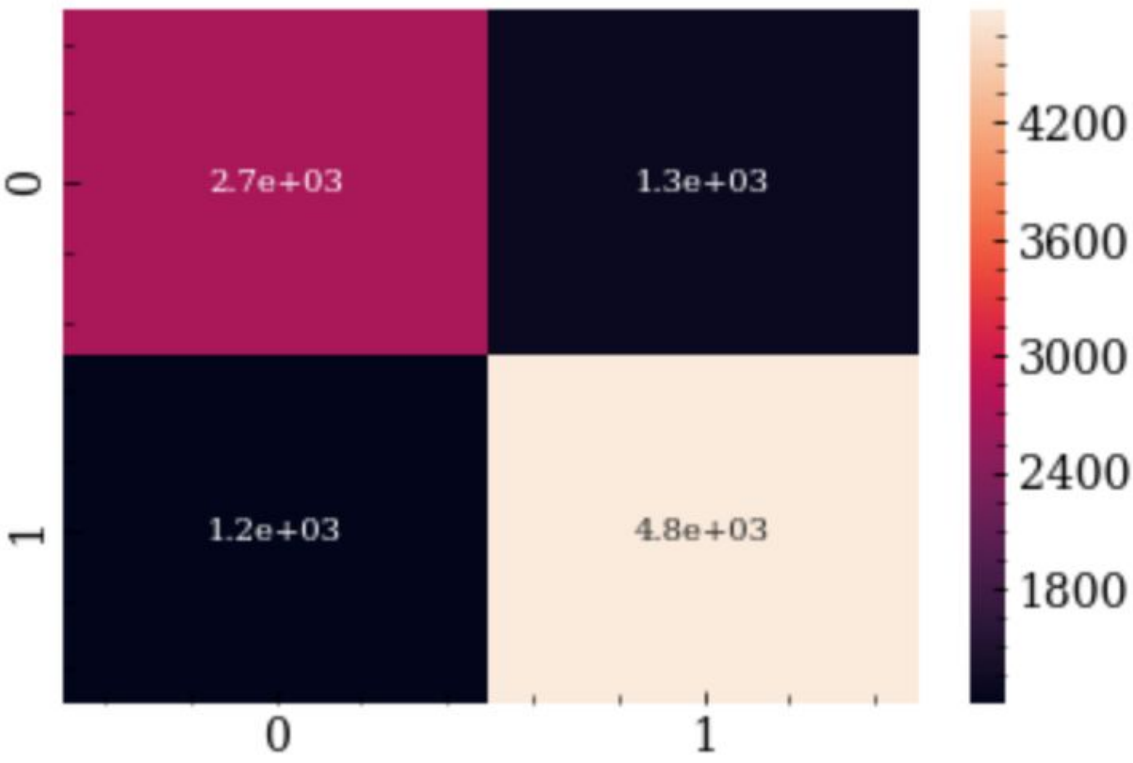

Figure 8. Confusion matrix from the random forest showing rate of true vs false results. On diagonal are true pos/neg and off diagonal of false. At a minimum, one would hope for similar values in the diagonal and very low values in the off.

Discussion

Some time was spent exploring and analyzing a corpus of tweets from a publicly available dataset of alleged Russian troll tweets. Out of the various techniques that were implemented, the pre-processing of the tweets was the most computationally expensive part of the project. The output of both NLTK and SpaCy had slightly different emphasis and both took about ~10 minutes on the troll corpus, depending on how stringent the lemmatization. We believe that our results were sufficient for the exploratory nature of our project, however considerable more time could have been spent cleaning the corpus to ensure uniformity and effectiveness. It would be interesting to see how much the preprocessing affects the final recall and accuracy of the machine learning techniques used in this project. That would be a project in and of itself though.

The bag of words model was relatively simple to write and also available in various forms with each python package that was used in this project. Transforming text data to vector and integer form is essential to cross natural language processing and machine learning algorithms. It was generally straightforward and successfully implemented both in data exploration and to prepare the corpia for the machine learning techniques.

TextBlob and the naive Bayes model provided thorough and immediate sentiment analysis, which was our ultimate goal. The documentation and source code of TextBlob do make it hard

to determine what classifying technique it uses to apply the model. The comments in the code simply state that it is "lazily" classifying. Given the way the total tweet sentiment is computed, it is computationally prohibitive to classify the entire corpus rather than focus on high interest terms. It is interesting to note that the high frequency of neutral tweets, with this algorithm, could possibly be due to the polarizing nature of the political content. It would be interesting to see how this classifier works on more mundane content.

The output of the decision tree algorithm and random forest was roughly similar. The sentiment polarity was preserved, as opposed to the inferred neutrality with TextBlob, and this makes it hard to compare the techniques in a quantitative way. In addition, since different training sets were used, a meaningful statistical analysis of the difference in the efficacy of the techniques would be hard to quantify. It should also be noted that a bias toward positive tweets was detected in the trees' training data set. One of the other major drawbacks, not mentioned previously, in a weakness toward bias. This could very well explain the significantly better accuracy found in True positives and the distribution of the confusion matrix.

In conclusion, all of these supervised machine learning algorithms can be effectively implemented to classify text data with a relative minimum of computational expense. However, the applicability of individual methods is highly dependent on the desired results and content of the documents. For example, the naive Bayes was sufficient to thoroughly analyze the term specific tweets for sentimentality. However, one can see where the easy implementation of trees and computationally inexpensive feature space could be useful. The Russian troll tweets corpus seem to have no trend. It is of interest to further pre-process the data to see how robust these algorithms are against contamination. Also, it would be interesting to extend the analysis of each method in such a way as to be directly comparable and to attempt to get rid of the apparent bias in each method.

References

1. "TextBlob: Simplified Text Processing¶." *TextBlob: Simplified Text Processing - TextBlob 0.15.1 Documentation*, textblob.readthedocs.io/en/dev/.
2. *Accessing Text Corpora and Lexical Resources,* http://www.nltk.org/book/ch02.html
3. "Russian Troll Tweets", https://www.kaggle.com/vikasg/russian-troll-tweets
4. "Twitter sentiment analysis", https://www.kaggle.com/c/twitter-sentiment-analysis2
5. *Sentiment Analysis of Twitter Data,* http://www.cs.columbia.edu/~julia/papers/Agarwaletal11.pdf
6. *A Primer on Neural Network Models for Natural Language Processing,* http://u.cs.biu.ac.il/~yogo/nnlp.pdf
7. "Scikit-Learn." *Scikit-Learn: Machine Learning in Python - Scikit-Learn 0.19.1 Documentation*, scikit-learn.org/stable/.
8. *Sentiment Classification using Decision Tree Based Feature Selection,* https://www.researchgate.net/profile/Annamalai_Suresh/publication/312068124_Sentiment_Classifi_cation_using_Decision_Tree_Based_Feature_Selection/links/586dd8dc08aebf17d3a7327e/Sentiment-Classifi-cation-using-Decision-Tree-Based-Feature-Selection.pdf

9. *Sentiment Analysis of Twitter Data: A Survey of Techniques,* https://arxiv.org/pdf/1601.06971.pdf
10. Brostoff, Ben. “A Brief Look at Sklearn.tree.DecisionTreeClassifier.” *Hacker Noon*, Hacker Noon, 27 Jan. 2018, hackernoon.com/a-brief-look-at-sklearn-tree-decisiontreeclassifier-c2ee262eab9a.
11. “Twitter Deleted Russian Troll Tweets. So We Published More than 200,000 of Them.” *NBCNews.com*, NBCUniversal News Group, www.nbcnews.com/tech/social-media/now-available-more-200-000-deleted-russian-troll-tweets-n844731.
12. “SpaCy · Industrial-Strength Natural Language Processing in Python.” *SpaCy · Industrial-Strength Natural Language Processing in Python*, spacy.io/.
13. “Twitter Deleted Russian Troll Tweets. So We Published More than 200,000 of Them.” *NBCNews.com*, NBCUniversal News Group, www.nbcnews.com/tech/social-media/now-available-more-200-000-deleted-russian-troll-tweets-n844731.