Twitter and Tweepy for Election 2020 Classification and Prediction

By Rushiv Arora, Younghoon Jeong, Sahil Malhotra

Abstract

National scale elections, like the recent 2020 Presidential elections, can be incredibly hard to predict and even harder to manage. To help aid candidates throughout their campaign, we set out with the goal of creating an efficient and effective social media classifier that could be used to predict the political affiliation of user accounts. In this project we specifically focused on Twitter as our social media platform of choice and trained several classification models on individual tweets. These classification models included Naive Bayes, Logistic Regression, and Neural Networks. Overall the Logistic Regression model outperformed the rest.

1. Introduction

As is with any political course of action, elections can be incredibly hard to win and manage. Candidates typically work for hours on end throughout the campaign in order to maximize voter turnout in their favor. As such, these campaigns involve a lot of planning ahead and political strategizing. Using various natural language processing methods, we were able to create several Twitter classifiers that predicted the political affiliation of a given tweet. Such a tool is incredibly useful for candidates and their campaign staff as it allows for gaining insights into voter demographics and the level of support a party has in a given area. These insights can then be used to better strategize the plan of attack leading up to the closing of the polls on election day.

Our approach for creating the classifiers started with a collection phase where tweets were collected from Twitter using the publicly available Twitter Developer API. These tweets were all collected before election day such that election day results did not influence the trends in the data. After collection, the data is then pre-processed and tokenized before being passed into one of our three constructed models: Naive Bayes, Logistic Regression, and Neural Networks. It is important to note that the preprocessing and tokenization steps for all 3 models were exactly the same in order to ensure a fair comparison of the results for each model. In the end, Logistic Regression had the best performance with an average accuracy of 92%.

2. Related Work

Election Prediction/Analysis is a crucial part of any candidate's campaign for elections all over the world, and such analysis and predictions have been done multiple times before. Some important works from the past that we thought are worth mentioning are:

1. *A Method for Predicting the Winner of the USA Presidential Elections using Data extracted from Twitter* by Oikonomou, Lazaros & Tjortjis, Christos. [1]

   This paper extracted twitter data in order to predict the outcome of the 2016 presidential

election in the states of Florida, Ohio, and North Carolina. The authors of this paper gathered data by directly building their own application to interact and communicate with Twitter's developer API, querying on tweets from the three states that had the most controversial and ambiguous polls and using political hashtags to find relevant tweets and to classify them as favoring, opposing, or being neutral towards a particular candidate. With this data they built a Naive Bayes classifier and also utilized a python library tool known as TextBlob which aided in the training of the classifier by providing a sentiment and subjectivity analysis for each tweet, or in simpler terms scoring each tweet based on how subjective and how many powerful sentimental words were used.

This paper has related work and provides a good comparison to our work. Our application uses tweets from all over the country as compared to this paper that focuses specifically on three states, thereby shaping ideas for future work on our project. On the other hand, our project supplements this work by including three separate models, which gives more scope to compare the performance of different NLP models for a given application.

2. *New York Times Decision Tree*, US Presidential Election 2008: [2]

   This decision tree measured the Obama-Clinton divide in the Democratic Representations for the 2008 US presidential election. The data includes exit polls and results from various counties in 48 US states (missing Florida and Michigan), and the information gathered includes race, location, demographic, level of education, and wealth.

   This method was interesting since it uses non-NLP techniques to make predictions on the general public's opinions. It is worthwhile and interesting to investigate techniques other than NLP to get a good comparison of the techniques we are applying in this class.

3. *Learning Political Polarization on Social Media using Neural Networks* by Belcastro, L. & Cantini, Riccardo & Marozzo, Fabrizio & Talia, Domenico & Trunfio, Paolo [3]

   This paper utilized a method known as Iterative Opinion Mining using Neural Networks, or IOM-NM for short, on twitter data in order to discover the polarization of social media users during an election campaign. The way this method works is that you start off with a limited set of classification rules, which are created from a small set of political hashtags, and then iteratively generating new classification rules where collection of posts is a continuous process. These classification rules can then be used to determine the polarization of users towards a political faction, especially when combined with data on post statistics, such as likes, shares, etc., and user demographic data. This methodology was then tested twice, once during the 2016 US Presidential Election and once during the 2018 Italian general election, achieving incredibly accurate and efficient results. The

paper then goes into a discussion of the benefits seen with this approach over other standard natural language processing and sentiment analysis techniques.

3. Data

To train and test classifiers to categorize tweets supporting each candidate, we collected a dataset of 337,306 tweets using Twitter Developer API[5] and *tweepy*[4], python module to use the API with python, by using some of APIs for searching tweets with specific keywords. We annotated each tweet by the search keyword by which the tweet was searched. For example, tweets searched with “#trump2020” is annotated as supporting Donald Trump. We removed tweets written in non-english to get tweets of people who are related to the election, and kept emojis in tweets. *Figure 1* is a table about search keywords representing each candidate, and how many tweets are annotated as supporting each candidate.

Label	**Trump**	**Biden**
Search Keywords (Hashtags)	"#TrumpPence", "#trump2020", "#TrumpPence2020","#4MoreYears", "#TRUMP2020ToSaveAmerica", "#TrumpTrain", "#Trump2020LandslideVictory"	"#BidenHarris", "#biden2020", "#BidenHarrisToSaveAmerica", "#BidenHarris2020", "#BlueWave2020", "#VoteThemOut"
# of tweets	149,880	187,426
	337,306	

[Figure 1] A table for search keywords representing each candidate.

Since the Twitter Developer API only searches tweets posted in the last 7 days as from the time of search, we gathered tweets for three weeks right before the election started. By this method, we can get the larger range of tweets in terms of dates which do not include the result of the election. Also, we removed duplicated tweets and tweets with errors such as tweets with texts saying that inappropriate contents are included in them, or tweets that do not include keywords they are searched by.
The format of each sample of a tweet is JSON, and each sample contains a lot of data of a tweet from its text to the place where it is posted. For this project, we only used field *full_text*, or field *full_text* in field *retweeted_status* if it exists since it has the true full text of a tweet while field *full_text* gives texts limited with 200 characters. This was not a big problem as most of the tweets have field *retweeted_status*.

4. Method

a. Preprocessing / Tokenizer

For preprocessing, we applied the following process: (1) make all the text lowercase, (2) remove hashtags and taggings, and (3) remove all the links. The texts were converted into lowercase letters since we want classifiers to analyze words in terms of their meanings. Hashtags, taggings, and links were removed since they could be too strong hints for classification. Then, we apply *TweetTokenizer* of *NLTK* as a tokenizer due to its ability to split emojis and to keep symbols such as arrows like "-->". For Naive Bayes classifier and logistic regression classifier, stopwords and punctuation tokens are removed and the remaining tokens are lemmatized by *WordNetLemmatizer* of *NLTK* to emphasize the meaning of words in the texts, but not for neural network classifier since we believe it has the ability to regard punctuations and stop words as tokens with low importance, and to deal with the morphology of words.

b. Naive Bayesian Classifier

First of all, we decided to develop and test a Naive Bayesian classifier as a baseline for the other classifiers. The classifier is based on the number of each word in vocabulary with the bag-of-words method. This means the classifier does not use any high dimensional criterions for classification, and this property leads us to determine the purpose of the classifier as a baseline of the project and a basic analysis of the dataset. To train the classifier, we use 80% of the dataset for training. And, to determine the value of hyperparameter α, a value for the additive smoothing, we divide training set in 5 folds and apply cross validation to get the average accuracy of each value of α among [1, 10, 50, 100, 500, 1000, 5000, 10000]. After we choose the value of α which returns the largest average accuracy, we trained the classifier with the determined value of α with the whole training set and tested with the remaining 20% of dataset.

c. Logistic Regression

The Logistic Regression model functions by mapping training data to its classifications, in our case tweets for Trump or Biden. The classifier then uses the mappings to generate a logistic function as the basis of the model. Similar to the Naive Bayesian model, the Logistic Regression model took a bag of words representation of each tweet as input with the classification of Biden or Trump being the output.

To create the Logistic Regression model we took advantage of the "CountVectorizer" and the "LogisticRegression" classes in the "scikit-learn" package. CountVectorizer provided us with a tool to tokenize and obtain a bag of words representation of each tweet as well as providing an overarching vocabulary with word counts for the entire dataset. We were then able to utilize the LogisticRegression class in order to create the model once the data was split into test and train data sets. In order to split the data up

into test and train data sets, and to really test the validity of Logistic Regression models, we took advantage of a process called cross validation with 5 different split groups, iterating over these split groups so that each one was utilized once as testing data (with all the others used as training data) and then averaging out the metrics for each iterations.

d. Neural Network

The neural network is a simple feedforward neural network with 2 linear layers and a ReLU activation layer. It is programmed in PyTorch and TorchText using FastText embeddings of dimensions 300. The data is preprocessed by removing hashtags, keywords, names (@name), and replacing links with “[link]”. Since word embeddings are used, Lemmatization and Stemming were not used for preprocessing. Further, as opposed to the other models, the ‘TokTok’ tokenizer was used since TorchText currently doesn’t provide functionality for the Tweet tokenizer. The TokTok tokenizer is the next best alternative to the Tweet tokenizer since it tokenizes a sentence in a similar manner, except for asterisks and contractions. A train/test split of 80/20 was used to evaluate the model that was trained for 5 epochs with a learning rate of 0.01 on a Cross Entropy Loss function with the Adam optimiser. These values were selected after the model was trained multiple times on different values of alpha ([0.01, 0.01, 0.1, 1] and different epochs ([5, 10, 50, 100, 1000]) to select the best parameters.
The Neural Network can be summarised in the table below:

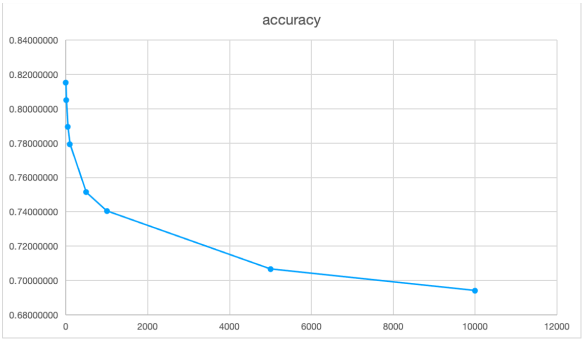
Feature	Description
Main Libraries	PyTorch and Torchtext
Layers	2 Linear and 1 ReLU
Word Embeddings	FastText Simple, 300D
Preprocessing	Links, Hashtags, Keywords, Names
Stemming/Lemmatization	No
Tokenizer	TokTok
Epochs	5
Learning Rate	0.01
Loss Function/Optimiser	Cross Entropy Loss/Adam
Evaluation	Train/Test: 80/20
GPU	Yes

[Figure 2] Settings of the neural network classifier

5. Results

a. Naive Bayesian Classifier

The result of the Naive Bayesian classifier is in *Figure 8*. According to the result of the classifier, it seems that the classifier satisfies the purpose of it, a baseline of the other classifiers as the value of the results are high enough.
To determine the value of α, we tested each candidate of the value. The graph of the accuracy for each value of α appears at *Figure 3*. The graph shows decrease of the accuracy as the value of α increases. This means that some words have a more obvious trend, than the other words, that they appear more in documents supporting an only either candidate of the election. To find those words, we calculated the ratio of the number of appearances of each word between the candidates, and the words are shown in *Figure 4*. Words with smaller number of appearances in documents of a candidate of the numerator than 100 are ignored and the number of appearances of words in documents of a candidate of the denominator is added by 1 to prevent dividing by 0. As we can see in *Figure 4*, the gap between the values of the division of top-20 words of each candidate is large. This confirms what we figure out by the graph of the value of α, which is the existence of words that appear much more on one candidate than the other.



[Figure 3] The graph of the accuracy of the value of α

Top-20 Words (Trump / Biden)	Trump / Biden
🌐	3776.0
12:35	2285.0
opa-locka	2282.0
ame	2272.0
88022	1566.0
pre-covid	785.2
motorcycle	550.0
jaffa	543.0
33.1	507.4
50k	399.5
contaminate	275.0
💰	233.0
prev	227.0
carpedonktum	220.0
modi	207.3
🇻	200.7
10k	200.0
northwest	188.7
graceful	184.0
kill-shot	181.0

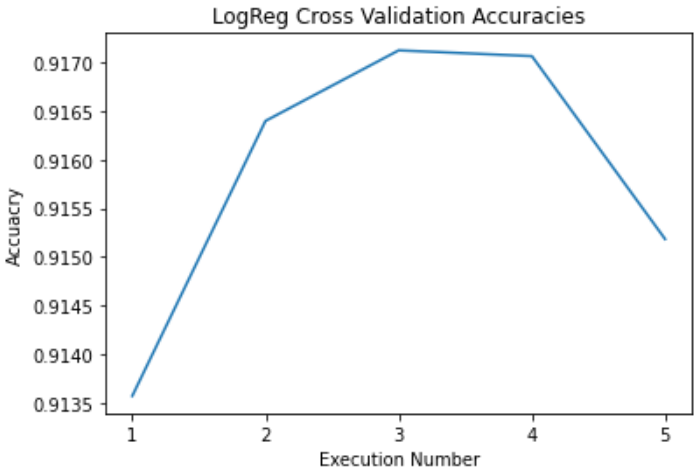
Top-20 Words (Biden / Trump)	Biden / Trump
watergate	3106.0
<)🐕>	2268.0
🐕	1512.0
bandaid	812.0
autocratic	788.0
idealization	778.0
tяump	766.0
stagger	391.4
jon	314.1
resister	310.0
amplify	308.6
io	306.0
shitler	306.0
soho	305.0
swore	286.0
600k	283.0
reassurance	277.5
beyoncé	276.7
encourages	257.9
tikka	233.0

[Figure 4] Top-20 words of (Trump / Biden) and (Biden / Trump)

b. Logistic Regression

The Logistic Regression accuracy results can be seen in *Figure 5*, which shows the individual accuracy of each iteration of the 5 group cross validation fold. The average between each of these iterations ends up being 92%, which is the highest accuracy of all three explored models. We also used precision and recall measurements for each candidate as additional measures of validity in the model. Averaged across all 5 cross

validation iterations, Biden's mean recall was 93% with a mean precision of 91%. Trump on the other hand had a mean recall of 89% and a mean precision of 92%.



[Figure 5] Logistic Regression Accuracy Values

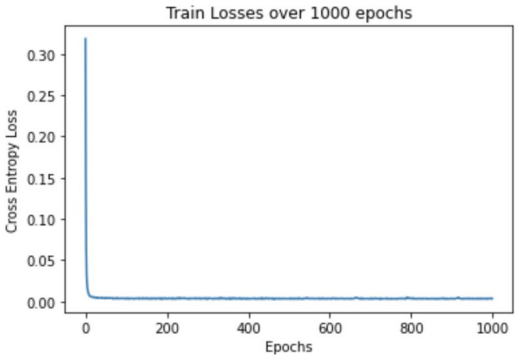
c. Neural Network

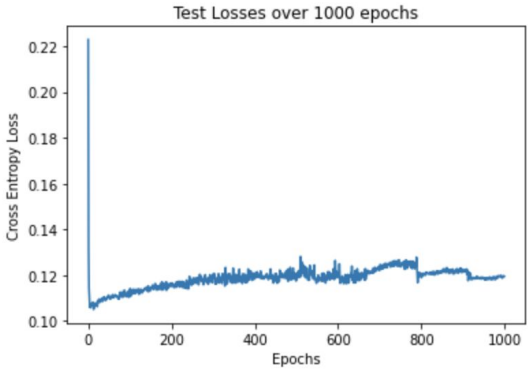
Overall, the neural network provided the best performance thereby demonstrating the presence of non-linear trends in the data and the importance of capturing those trends.

The neural network was evaluated using a train/test split of 80/20 with a Cross Entropy Loss Function and the Adam Optimiser for 5 epochs. After 5 epochs, the training accuracy was 93.1% and the training loss was 0.146, while the testing accuracy was 85.1% and the testing loss 0.358.

It is interesting to note that the first guess of the neural network produced accuracy scores of around 0.45 - 0.55, which is very intuitive since it means we are fifty percent accurate with initial random weight values, equal to the odds of making a random guess.

The plots below ran the example for 1000 epochs, and it can be noticed that after the first few (5-10) epochs the training loss decreases drastically and then stays around 0 with very minor changes. After the initial few epochs, the testing loss stops decreasing and then starts increasing because we are now overfitting the data. This graph comparison helped me decide on the optimal number of epochs to train for.





[Figure 6] The loss graph for training dataset

[Figure 7] The loss graph for test dataset

d. Summary

	Naive Bayes	Logistic Regression	Neural Network
Accuracy	0.82	0.92	0.85
Precision	0.82	0.91	0.80
Recall	0.76	0.91	0.86
F1-score	0.79	0.91	0.83

[Figure 8] A table for the result of each classifier.

6. Discussion and Future Work

Overall, we have made significant progress towards creating our social media political affiliation classifier. All of the models that we experimented on performed incredibly well especially with the reported accuracy of 92% from the Logistic Regression Model. These promising results indicate that the model performs well enough to be used in a production system. Had there been more time in the semester, we would have created such an application that would be able to take advantage of our classifier and dynamically provide political insights to the user based on social media demographics and the predicted political affiliation of users in live time. We would have also been more robust in our data selection to also include other social media platforms besides for Twitter so that our overall classifier would be more general and robust.

References

[1] L Oikonomou, and Christos Tjortjis,"*A Method for Predicting the Winner of the USA Presidential Elections using Data extracted from Twitter",* South-Eastern European Design Automation, Computer Engineering, Computer Networks and Society Media Conference, September 2018 pp 1-8.

[2] New York times, *Decision Tree: The Obama-Clinton Divide,* The New York Times Archive April 16 2008.

[3] L. Belcastro, R. Cantini, F. Marozzo, D. Talia and P. Trunfio, *Learning Political Polarization on Social Media Using Neural Networks*, in *IEEE Access*, vol. 8, pp. 47177-47187, 2020, doi: 10.1109/ACCESS.2020.2978950.

[4] Joshua Roesslein*, Tweepy Documentation*, http://docs.tweepy.org/en/latest (5 Dec, 2020)

[5] Twitter, *Twitter API v1.1*, https://developer.twitter.com/en/docs/twitter-api/v1 (5 Dec, 2020)