Predicting Political Affiliation Based on Vocabulary

Big Data Final Report

Team Members : Vaishali Marar, Harrison Park, Mandeep Singh, Ashazi Shakur

Abstract

In a time in which social media is a medium through which news travels and is spread, whether it be directly from media sources or political figureheads, bias exists from both sides. It is natural for someone that identifies with a particular political party to subscribe to social media accounts, social media news outlets, political figures, that believe and speak on the same ideologies. Being able to identify what political ideology an account on social media is representing or is aligned with can be beneficial when seeking out unbiased news sources.

To approach this problem tweets will be analyzed using a series of data visualization techniques, and then will be used to create machine learning models to predict political affiliation. The language used by Democrats or Republicans are specific to their ideologies, therefore analyzing tweets written by political representatives will be a great asset to our machine learning models. Political representatives use very politically correct and specific jargon that is unique to their political parties lexicon. Using methods such as Count Vectorizer and TF-IDF, a series of models can be created to predict what party a tweet is leaning towards or identifies as. The models will then be compared based on accuracy and their confusion matrix to determine which one is most suitable for this task.

Introduction

Politics has transcended television and passive engagement of users, it has become an interactive entity, with the majority of political figures, news outlets and reporters actively participating on social media platforms. Textual Analysis provides a way to verify the news, and identify subjective sources of information with no ulterior agenda. There have been a number of similar studies over the years that identify the differences in reporting due to political affiliation. Although these studies have shed a light on the problem at hand, which will only get worse in coming years, only as of recent has there been a way to automate this process and quantify the differences. Using predictive models to analyze and quantify the polarizing language used by political figures has numerous applications, past singling out subjective and reliable news sources.

Section 1: Review of Existing Literature

1. *Twitter Language Use Reflects Psychological Differences Between Democrats and Republicans*
   This study was an important reference point during our research as it studies Twitter data as it relates to political affiliation in a unique way. Instead of focusing on obvious differences in terminology this study focuses on the emotion behind certain vocabulary or language. This study analyzed whole accounts and timelines rather than randomized tweets, to determine the category of words that could be associated with a certain political party. For example, this study revealed that liberals or Democrats often use more swear words, words relating to mental health and more emotive language than their Republican counterparts.
1. *"Read My Lips" : Using Automatic Text Analysis to Classify Politicians by Party and Ideology*
   This study used machine learning models on a cleaned and stemmed corpus, using words that appear a 1000 times or more. This study also takes into account senator-sessions, reelection of House Members, to ensure a more "fine-grained" approach. This study uses Decision Tree, Naive Bayes, Support Vector and Lasso-Penalty. This study took into account a number of other factors, to not only identify what party but to what degree a member subscribed to a party.
2. *Automating Political Bias Prediction*
   This study used statistical learning methods to predict political bias from text. Indicative text and features extracted from political speeches, referendums and manifestos were used to create corpuses which were then used to predict political party affiliation. The study used two distinct approaches. The first approach was to use discriminative text features that were extracted that were specific to a particular political party and the second approach was to use sentiment analysis. The sentiment analysis approach was used to link text features with certain measures of political power. This study aimed to define a method in which a model can be used to analyze texts to determine bias and degree of political influence.

These studies were integral to our research prior to taking on this project. Various studies have attempted to study and identify political bias in speech and text through the use of sentiment analysis and through the use of a political corpus. The various studies above have used classifiers such as Naive Bayes and Support Vector Machine as they have proved suitable for sentiment analysis. Doing thorough research on previous studies allowed us to understand how to shape and structure our project. By finding similar size data sets and understanding the methodology used by industry experts we were able to recreate smaller scale but similar models to address a similar problem.

Section 2 : Overview of problem

DATA :

The data set we used was obtained from kaggle a reliable and frequently used source of data. The data was structured in three columns “Party”,”Handle” and “Tweet”, the handles contained notable democratic and republican leaders classified in the Party column, the Tweet column contained tweets under 240 characters. The data was not cleaned and contained many extraneous characters, URL links and other symbols that will later be addressed. The data itself had 86,460 entries, 42,068 of these entries are from democratic figureheads and 44,392 of those entries are from republican figureheads. Each political figure in the dataset has 200 tweets, and there are 211 democratic figures and 222 republican figures.

APPROACH :

Using this dataset we aimed to train a model that could, upon reading a set of tweets, determine which tweets were democratic and which tweets were republican. After thorough research on how to approach this we learned about sentiment analysis, which is essentially the base of the problem we are trying to address. We used common NLP tactics such as stop words, stemming, lemmatization and attempted tokenization. We also learned about the bag of words method, or CountVectorizer and TF-IDF methods which both work together within a pipeline to train the model. We selected three types of models for their specific penchant for sentiment analysis; Naive Bayes, Support Vector Machine and Decision Tree Regressor.

- *Sentiment Analysis :* Sentiment Analysis is a process that aims to identify or classify emotions using textual machine learning. Sentiment analysis extracts opinions from text and determines whether it is positive, negative or neutral. It also extracts derived attributes such as polarity. Sentiment analysis can be applied on the document level, sentence or sub-sentence level. There are varying types of sentiment analysis which aim to identify particular sentiments such as “interested vs non interested”.

- *StopWords, Stemming and Lemmatization:* Stemming and lemmatization are similar in concept in that they both aim to consolidate words based on their root. So words that are in different forms are returned to their root word so that during CountVectorization multiple instances of the same word but in different forms are not counted separately. StopWords are an important aspect of data cleaning for sentiment analysis, it essentially uses a corpus with words that are deemed useless such; “a” , “the”, “at”, etc.

- *CountVectorizer :* CountVectorizer is a method of feature extraction that tokenizes words and then creates an encoded vector with words and the number of instances that word occurs in a particular document.

- *TF-IDF :* TF-IDF is a vectorizer or method of feature extraction similar to CountVectorizer as well but it also takes into account relative frequency of vocabulary. It first calculates term frequency (TF) and then calculates inverse document frequency(IDF) which weighs words against the number of its occurrences in a document. This method is especially useful for sentiment analysis as it disregards words that are not stopwords, but are essentially useless in categorizing a sentence or document.

MODELS USED:

1. Naive Bayes Multinomial Classifier
    - Naive Bayes is a probabilistic algorithm used to predict the tag/label/classification of a document. The Bayes theorem takes into account the probability of a feature with knowledge of its conditions, and then assesses the probability of each given tag for a document. The “naive” aspect of the Bayes model is the aspect that assumes that every individual word in a sentence is independent of other words in the sentence or document.
    - A Multinomial Bayes Model takes into account word count and adjusts probabilities accordingly. Essentially this variation of the Naive Bayes model is more suited for our purposes as word frequency information is taken into account. When dealing with political sentiment analysis such as in our project, the multinomial model is a more suitable variation comparable Naive Bayes Bernoulli Model which also performs well with small vocabulary sizes(McCallum and Nigam).
2. SVM Linear Kernel
    - SVM is an algorithm that uses hyperplanes in n-dimensional spaces, which aim to find the largest margin between data points. Support vectors are the points close to the hyperplane which control and influence the position, and direction of the hyperplane. Manipulating these vector points change the position of the hyperplane and therefore change the margins(Ghandi).
    - The most frequently used SVM kernels are Polynomial, RBF/Gaussian, String and Linear. The Linear kernel has been proved to be suitable for text classification as it works well for instances in which the number of features is larger than the number of observations which would apply in the case of our project.
3. Decision Tree Regressor
    - The Decision Tree Regressor is a structure in which each node indicates a test of an attribute and each branch the outcome of the tested attribute. The DTR works well with categorical data and are able to understand the non-linear relationship between various features and the target variable.
    - As DTR’s do not do well with sparse categorical features, much preprocessing and cleaning of the data was required prior to creating the model.

Section 3 : Methodology

PART 1 : Data Cleaning and Preprocessing

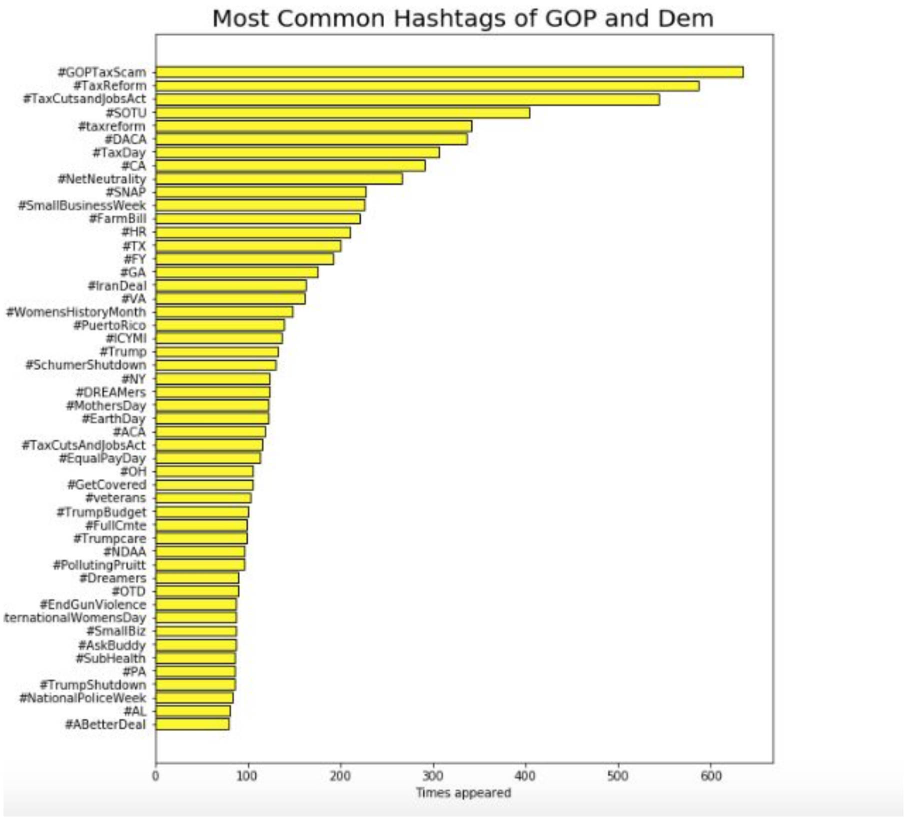
First we used pandas to read the CSV file and place it into a dataframe, we then examined the data to see its columns and attributes. We then used the NLTK library's corpus of stopwords to remove unnecessary data from the tweets, mainly words such as "a","the", etc. We then defined functions to remove usernames, convert all words into lowercase and removed URLs as well. The next step was to then use NLTK's WordLemmatizer function to lemmatize all words to ensure when we CountVectorized the data, that there wouldn't be multiple forms of the same word. We also changed the Party field, by converting all "Democrat" values to 1's and all "Republican" values to 0's.

We also created two new dataframes, one holding democrat data and one holding republican data for our data visualization section

Part 2 : Data Visualization

Hashtag Analysis

First we created a pattern matching function using the Re library to go through all tweets in the data set and find all hashtags and then assign them to a list called matches. A nested for loop then goes through all the values in matches and appends unique hashtags to a dictionary, and appends the value of the unique hashtags key +1 if the hashtag occurs more than once. We then have a dictionary with all hashtags and their number of occurrences. We then sort the dictionary items, separate the keys and values and then generate the top 50 hashtags. Using this information we plot a bar chart with the hashtags and the times it appeared in the data.



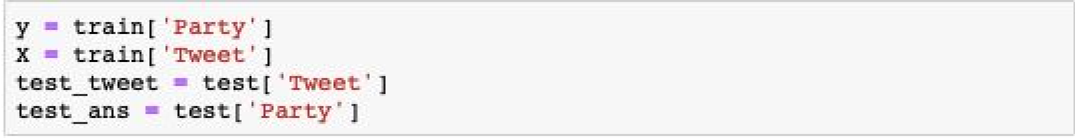
Wordcloud

We used Word Clouds built in functions to create wordclouds, we used the two seperate data sets for the republican and democratic parties as well as the original data set to visualize the differences in vocabulary occurrences

Part 3 : Creating ML Models

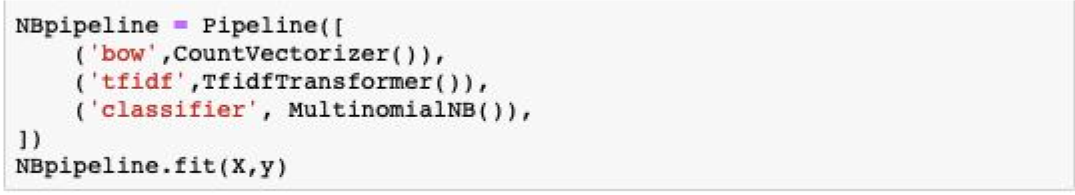
Using SKleanrn's pipelines which help to automate machine learning methods and workflows we were able to create three models. First we created a copy of the original data set to work off of. We then split the data set into training and test data, the test data contained 20% of the data and the train data contained the remaining 80%. We stratified the training and test data so both sets contained an equal amount of democratic and republican values in order to ensure we wouldn't overfit our model.

We defined our target value as the Party (democratic or republican), and the X value or training feature as the Tweets. We also pre-defined the variables we would use when testing our data as well.



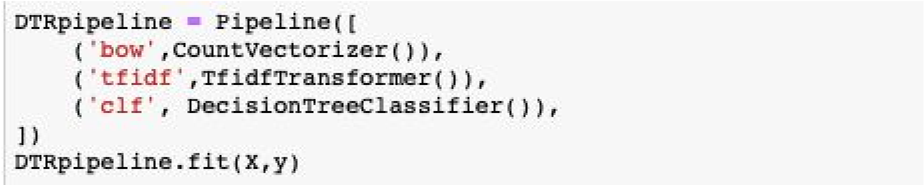
```
y = train['Party']
X = train['Tweet']
test_tweet = test['Tweet']
test_ans = test['Party']
```

We then created our pipelines for the three models : Naive Bayes Multinomial Classifier, SVM Linear Kernel and Decision Tree Regressor.

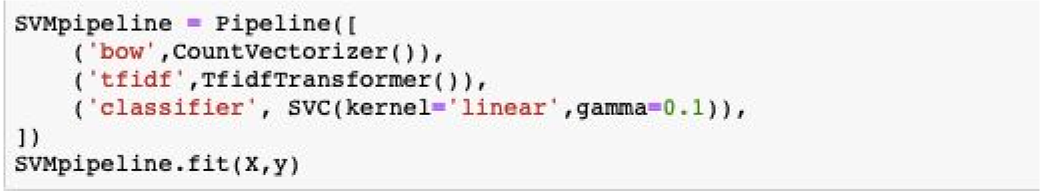


```
NBpipeline = Pipeline([
    ('bow',CountVectorizer()),
    ('tfidf',TfidfTransformer()),
    ('classifier', MultinomialNB()),
])
NBpipeline.fit(X,y)
```

The pipeline takes in the CountVectorizer or the Bag of Words method, TF-IDF and the classifier. We then fit the training data and test data on the new data (testing data). We then repeated this process for SVM and Decision Tree Regressor as well.



```
DTRpipeline = Pipeline([
    ('bow',CountVectorizer()),
    ('tfidf',TfidfTransformer()),
    ('clf', DecisionTreeClassifier()),
])
DTRpipeline.fit(X,y)
```

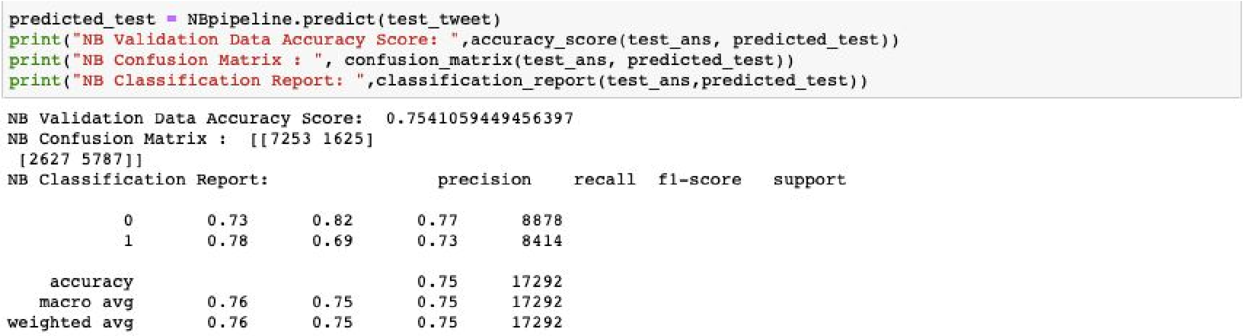


```
SVMpipeline = Pipeline([
    ('bow',CountVectorizer()),
    ('tfidf',TfidfTransformer()),
    ('classifier', SVC(kernel='linear',gamma=0.1)),
])
SVMpipeline.fit(X,y)
```

For the SVM model we originally did not specify a kernel and the model did poorly. So we specified a linear kernel as well as a gamma value. The gamma value essentially specifies how much influence a training model should have.

Section 4 : Results

After creating our models we used Sklearns confusion matrix, classification report and accuracy score to examine how the models performed on the test data.

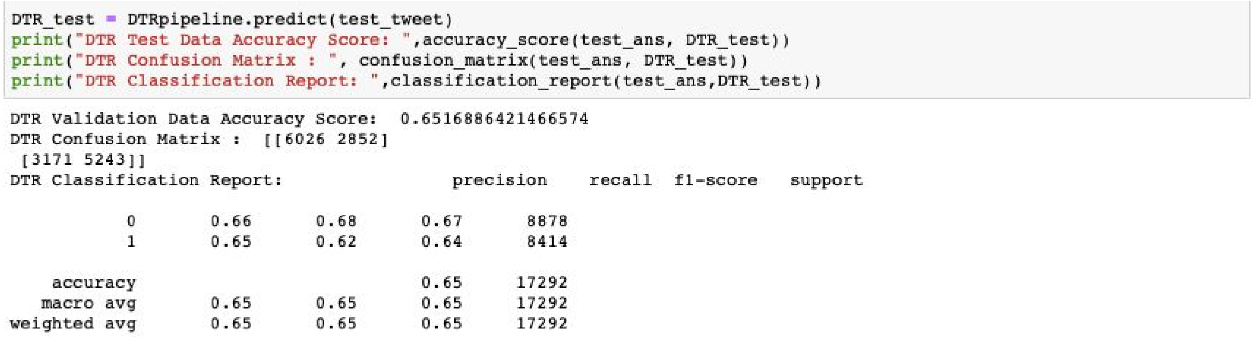


```
predicted_test = NBpipeline.predict(test_tweet)
print("NB Validation Data Accuracy Score: ",accuracy_score(test_ans, predicted_test))
print("NB Confusion Matrix : ", confusion_matrix(test_ans, predicted_test))
print("NB Classification Report: ",classification_report(test_ans,predicted_test))
```

```
NB Validation Data Accuracy Score:  0.7541059449456397
NB Confusion Matrix :  [[7253 1625]
 [2627 5787]]
NB Classification Report:                precision    recall  f1-score   support

           0       0.73      0.82      0.77      8878
           1       0.78      0.69      0.73      8414

    accuracy                           0.75     17292
   macro avg       0.76      0.75      0.75     17292
weighted avg       0.76      0.75      0.75     17292
```

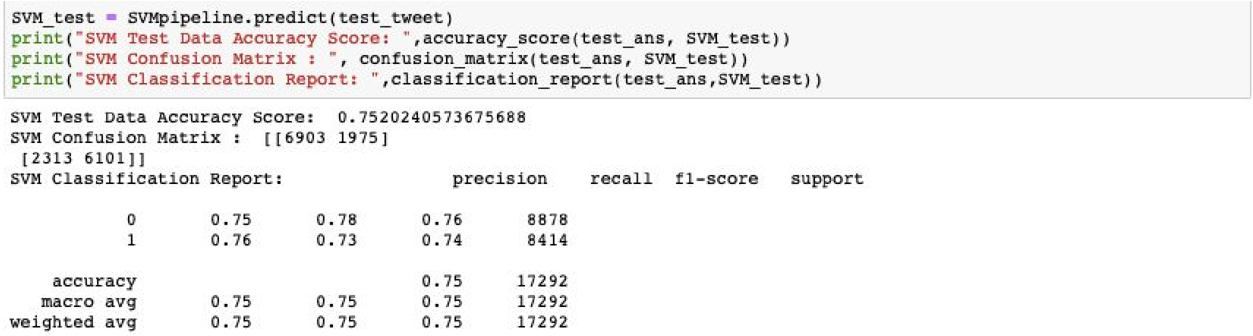


```
DTR_test = DTRpipeline.predict(test_tweet)
print("DTR Test Data Accuracy Score: ",accuracy_score(test_ans, DTR_test))
print("DTR Confusion Matrix : ", confusion_matrix(test_ans, DTR_test))
print("DTR Classification Report: ",classification_report(test_ans,DTR_test))
```

```
DTR Validation Data Accuracy Score:  0.6516886421466574
DTR Confusion Matrix :  [[6026 2852]
 [3171 5243]]
DTR Classification Report:                precision    recall  f1-score   support

           0       0.66      0.68      0.67      8878
           1       0.65      0.62      0.64      8414

    accuracy                           0.65     17292
   macro avg       0.65      0.65      0.65     17292
weighted avg       0.65      0.65      0.65     17292
```



```
SVM_test = SVMpipeline.predict(test_tweet)
print("SVM Test Data Accuracy Score: ",accuracy_score(test_ans, SVM_test))
print("SVM Confusion Matrix : ", confusion_matrix(test_ans, SVM_test))
print("SVM Classification Report: ",classification_report(test_ans,SVM_test))
```

```
SVM Test Data Accuracy Score:  0.7520240573675688
SVM Confusion Matrix :  [[6903 1975]
 [2313 6101]]
SVM Classification Report:                precision    recall  f1-score   support

           0       0.75      0.78      0.76      8878
           1       0.76      0.73      0.74      8414

    accuracy                           0.75     17292
   macro avg       0.75      0.75      0.75     17292
weighted avg       0.75      0.75      0.75     17292
```

Explanation of Results:

Confusion Matrix and Accuracy Score

A confusion matrix is a summary of results, it indicates the number of accurately predicted and wrongly predicted values for each class or value. The classification matrix is then used to calculate the accuracy score.

Classification Report

Precision is a score that references the overall ratio of instances that were classified accurately. Recall references the ratio of values of a certain class that were identified correctly. For our purposes we did not look to F1 score or support for determining accuracy.

Section 5: Conclusion

Overall the models that we used did well, the SVM Linear Kernel and the Naive Bayes Multinomial Classifier were the best choices. According to the accuracy score and the confusion matrix of the above models, the Naive Bayes model would be the best choice but the Linear Kernel did similarly. Our original model for SVM did not do well as we did not specify a kernel, we changed to a linear kernel as SVM Linear is commonly a suitable model for sentiment analysis.

Overall, this project was a great learning experience that required a lot of research and preparation before actual implementation. We had no prior knowledge of machine learning algorithms or commonly used methods and essentially started from scratch.

In the future some changes we would implement would be:

- Choosing a data set with more features that would allow for a better model
- Learning more about how to manipulate models
- Further understand how feature selection can be improved.

Member Contributions :

1. Vaishali Marar and Harrison Park : Model Creation and Data Visualization
2. Mandeep Singh and Ashazi Shakur : Data Preprocessing

REFERENCES

Section 1 References :

Biessmann, Felix. “Automating Political Bias Prediction.” https://arxiv.org/pdf/1608.02195.pdf.

Gheiler, Eitan Sapiro. “‘Read My Lips’: Using Automatic Text Analysis to Classify Politicians by Party and Ideology.”

Sylwester, Karolina, and Matthew Purver. “Twitter Language Use Reflects Psychological Differences between Democrats and Republicans.” *PLOS ONE*, Public Library of Science, https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0137422.

Section 2 References :

McCallum, Andrew, and Kamal Nigam. “A Comparison of Event Models for Naive Bayes Text Classification.” http://www.cs.cmu.edu/~knigam/papers/multinomial-aaaiws98.pdf.

https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca4