Twitter Sentiment Analysis of Electoral College in U.S.

Subject: Springboard Capstone Project-2 Final Report
~ Sneha Krishnamurthy

---

1. Introduction:

Twitter is a treasure trove of sentiments. In this report we assess the feasibility of using Python based supervised learning to classify tweets.The goal of this project is to predict Twitter users' political sentiment towards electoral college in U.S. for a given Twitter post. Many Americans are critical of the Electoral College, an attitude that seems to have intensified since Donald Trump defeated Hillary Clinton in the 2016 presidential election despite losing the popular vote. The data used for this model is obtained by using Tweepy to scrape Twitter API for data collection using the keyword "electoral college".

Sentiment Analysis is simply gauging the feelings behind a piece of content.It maps and measures the relationships and flows of information between people, communities and organizations In this work, we will build a machine learning classifier that can detect sentiment to classify tweets (pieces of text) as positive, negative or neutral by analyzing the feeling that Twitter users have towards "electoral college".In this report, we discuss the nuisances met in handling the data , cleaning the data, and extracting features from the data. Finally, using the extracted features, classification algorithms are trained, and their performance is assessed. Python code used for data preprocessing, analysing, modeling in this Github link.

2. Problem:

This project explores on predicting how positive, neutral or negative each tweet is about electoral college. What is being said about Electoral College on social media? How has the sentiment changed over the years? We also want to understand the overall feelings towards Democrats and Republicans when it comes to the topic of electoral college.

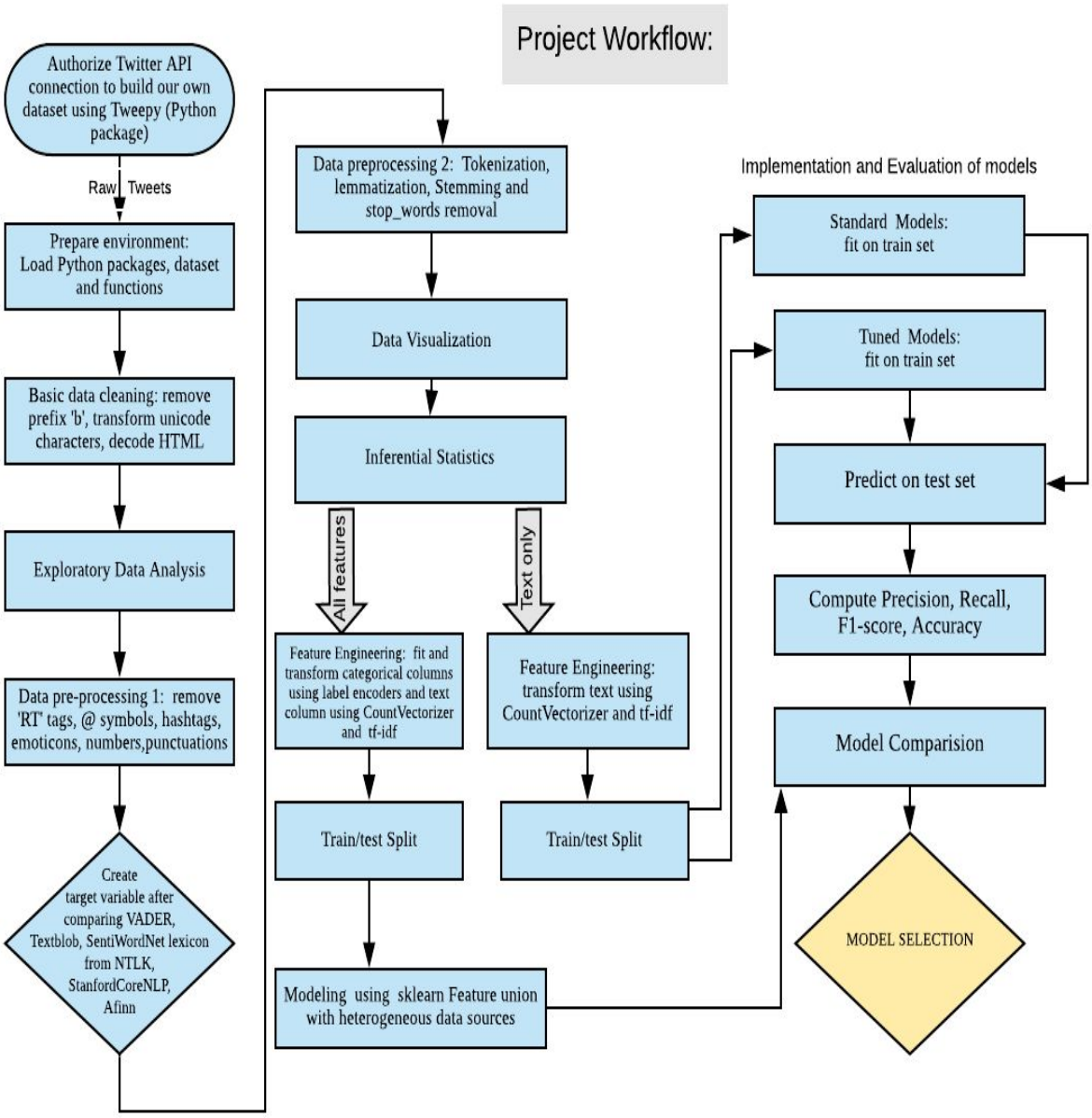
3. Clients:

Sentiment Analysis models can be used by election campaign officials, election committees and also the general public to learn about the public sentiment towards electoral college. This

study seeks to display helpful information about electoral college that even the candidates may consider in future while on the campaign trail.

4. Approach:

A project flowchart below shows the order of methods involved in the completion of this project.



5. Data:

This project uses Tweepy to access Twitter API for data collection using the keyword “electoral college”.

After dropping columns such as twitter handle and usernames, to comply with Twitter API policy, we were left with 47182 rows and 13 columns in the dataframe.

4.1 Data Dictionary:

The columns that are included are:

Attribute	Description
User_Id	The integer representation of the unique identifier for the Tweet.
Total Tweets	total number of Tweets the account has posted
Favourites_Count	Indicates approximately how many times this Tweet has been liked by Twitter users.
Followers	number of followers
User_Verified	Boolean value True if the blue verified badge is present on Twitter which lets people know that an account of public interest is authentic.
User Location	Place associated with tweet
Date of Tweet	UTC time when the Tweet was created.
Tweet Id	The numerical ID of the desired Tweet.
Tweet Text	The actual UTF-8 text of the status update
Language	When present, indicates a BCP 47 language identifier corresponding to the machine-detected language of the Tweet text
Tweet Source	Utility used to post the Tweet
Tweet Retweet	Number of times this Tweet has been retweeted
Tweet Reply To Id	If the represented Tweet is a reply, this field will contain the integer representation of the original Tweet’s author ID.

After basic cleaning of data extracted from the Twitter API, sentiment score was generated for each tweet, which would be our target variable. To create target variable, we have compared several sentiment analyzer tools which are widely available for classifying the data such as VADER, Textblob, SentiWordNet lexicon from NTLK, StanfordCoreNLP, Afinn as provided in this link

StanfordCoreNLP was used to label our tweets dataset since it is designed to help evaluate a model's ability to understand representations of sentence structure, rather than just looking at individual words in isolation.

6. Exploratory Data Analysis:

In this project, we'll be retaining only English language tweets. After filtering the data-frame with only english language tweets, we were left with 46831 rows of data. The first step in building the model is cleaning the data obtained and analyzing the data by applying EDA techniques using Python and other libraries such as the natural language toolkit (nltk), gensim, scikit-learn and spaCy.

6.1 Character length:

The average character count was approximately 128, which is well within the limit of overall average tweet length of around 140 characters. As far as Twitter is concerned, every single character in a Tweet counts as one for the purposes of the character count which includes letters, numbers, spaces,letters with accent marks, mentions,hashtags and other symbols.There is only one exception to this rule which is that Twitter uses its own URL shortening service, any website address you post in a tweet will count as 22 characters, regardless of whether it originally was longer or shorter than that. Even extremely basic features such as character counts can prove to be very useful to find outliers. Looks like we do not need to worry about that for now since the character length is well within the limit.

Most of the tweets in our dataset have some elements that we do not want in our character counts. Below is a conversations at glance extracted from "Tweet Text" column:

"b'We\\xe2\\x80\\x99re still skewed by population but not quite near as much. TGI Electoral College... \\n#ElectionResults2019\\xe2\\x80\\xa6 #ElectionResults2019 #ElectoralCollege #Election2020 #YangGang #FeelTheBern #TrumpTrain https://t.co/5ilhukuqnr'"

As we can see, we have to remove 'b' prefix which is a decoded string in Python to stop it from interfering in future coding. The hexadecimal representation \xe2\x80\x99 of the Unicode

character U+2019 is actually the right single quotation mark. We also see strange patterns of characters "\xe2\x80\xa6" which are UTF-8 BOM (Byte Order Mark)."The UTF-8 BOM is a sequence of bytes that allows the reader to identify a file as being encoded in UTF-8."

It looks like HTML encoding has not been converted to text as can be seen at row #2, and ended up in text field as '&amp','&quot',etc. Decoding HTML to general text was my next step of data preparation. I used BeautifulSoup for this.

After basic text cleaning, the tweet conversations look as below:

"'We're still skewed by population but not quite near as much. TGI Electoral College... #ElectionResults2019\\xe2\\x80\\xa6 #ElectionResults2019 #ElectoralCollege #Election2020 #YangGang #FeelTheBern #TrumpTrain https://t.co/5ilhukuqnr'"

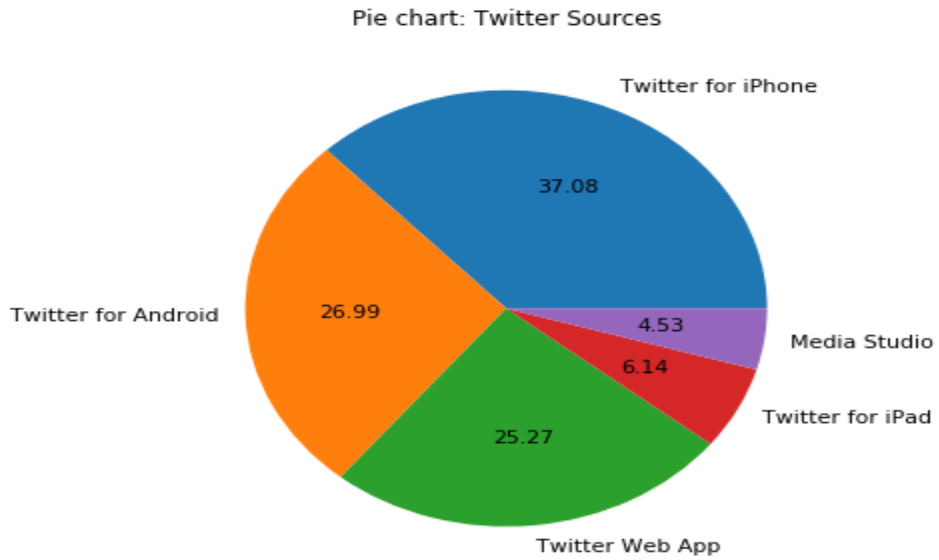
URL links, @ symbols and hashtags need to be included in character count and was preprocessed later.

6.2 Emoticons:

We also found out that there were no emojis left when we ran a function to check if any emojis are left in text column.

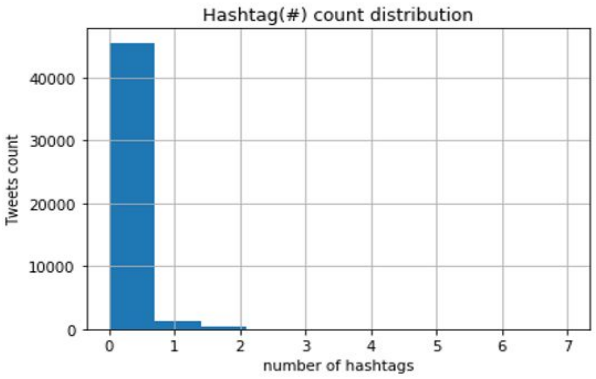
6.3 Twitter Sources:

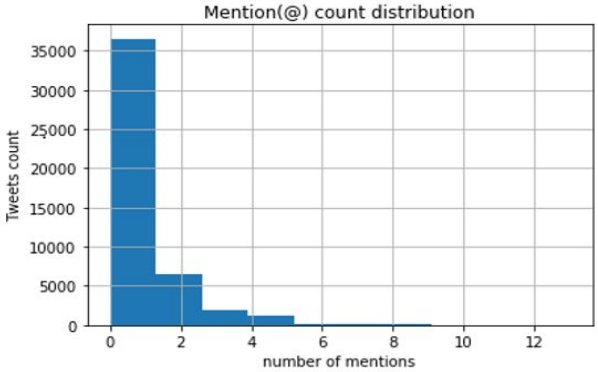
When we analyzed twitter sources, we realized that basically this twitter dataset has four major sources such as “Twitter for iPhone”, “Twitter for Android”, “Twitter Web App” ,”Twitter for iPad” and the rest such as “AshburnTimes”, “Hootsuite Inc.”, etc. were consolidated as ” Media Studio” as shown in the pie chart below:



6.4 Hashtags and Mentions:

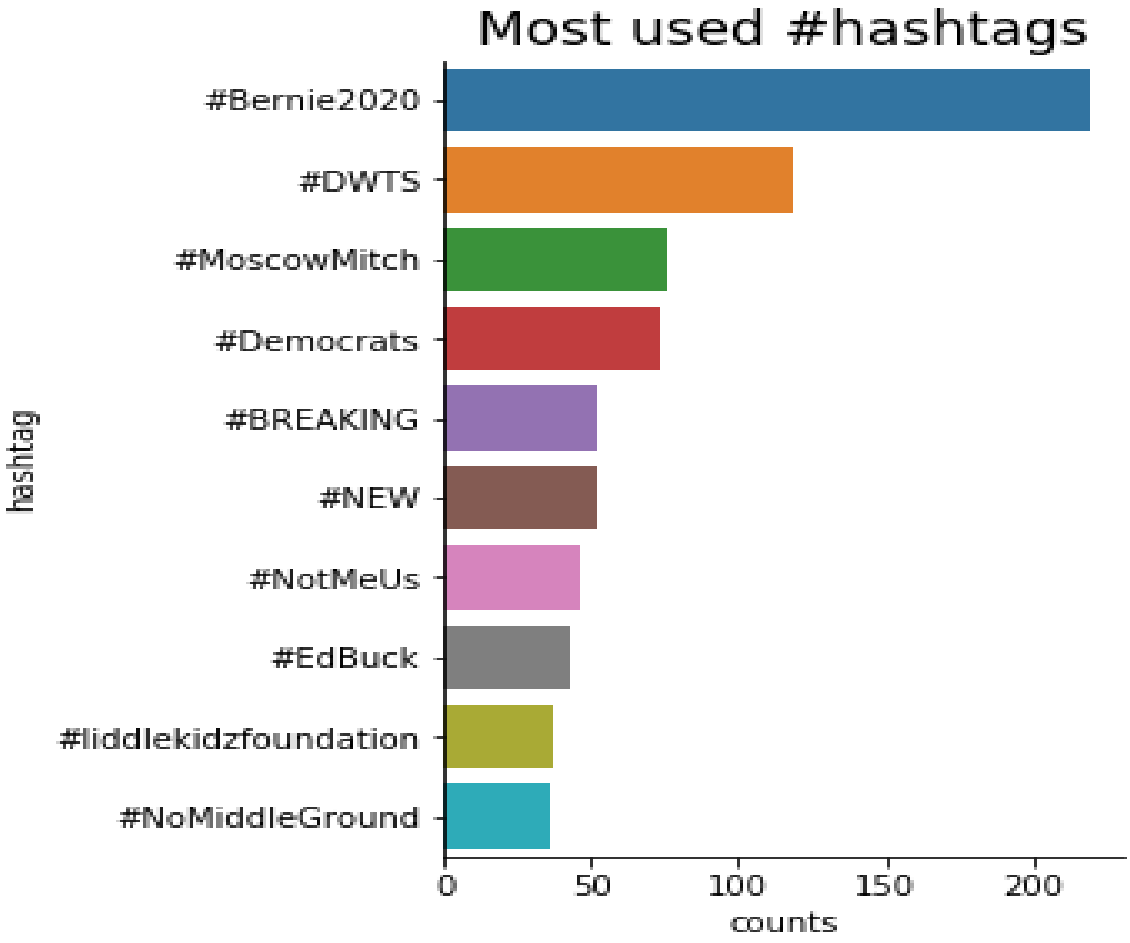
Comparison of the distribution of hashtags and mentions in tweets from this dataset:



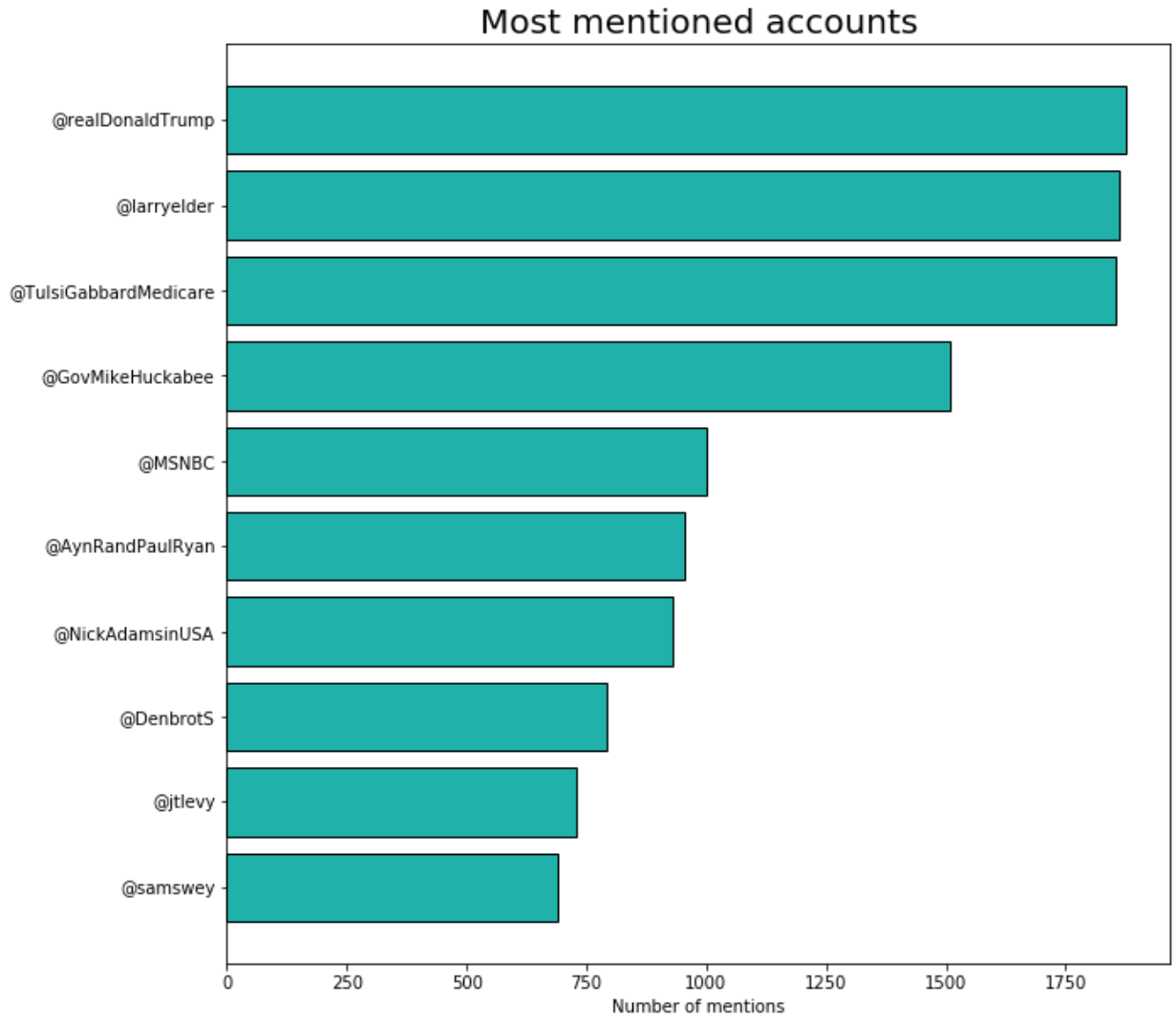


It looks like the number of mentions is more than the number of hashtags in our dataset.

Below is a graph created using Seaborn to visualize most used hashtags:



From this figure it can be clearly seen that #Bernie2020 is the most used hashtag. #NotMeUs also is also a political organization looking to further the progressive cause by following Bernie's lead. We can guess that many of Bernie's supporters are actively opposing electoral college.



It is no surprise that @realDonaldTrump leads the list.

7. Data Preprocessing:

It is important to check if we have **duplicates** which is common in tweets because of the RT tag (Retweets), After **dropping duplicates**, we were left with 18367 rows of data.

Regular expressions were used to get rid of

1) special characters, numbers, punctuation, etc,.
2) replace username-tags with blank space
3) replace hashtags with blank space
4) Retain words with only 3 characters are more
5) Retain alphabetic words

Using a function, the text was further cleaned and the following tasks were performed.

1) Reduce all letters to **lower- case**
2) Stop words are removed using ntlk library and additional stop words
3) Tokenize text using word_tokenize
4) Stem the words using ntlk Porterstemmer
5) Lemmatize all tokens using WordNetLemmatizer.
6) Detokenize lemmatized tokens for vectorizing

Below is an example to show the difference in the raw text and processed tweet:

Tweet Text #21 as is:
'@RepKevinBrady @GOPLeader @KPRC2Khambrel We\xe2\x80\x99ll find out with public hearings. Let\xe2\x80\x99s be clear. Trump lost by 3 milli\xe2\x80\xa6 https://t.co/VXdYuJdjQK'

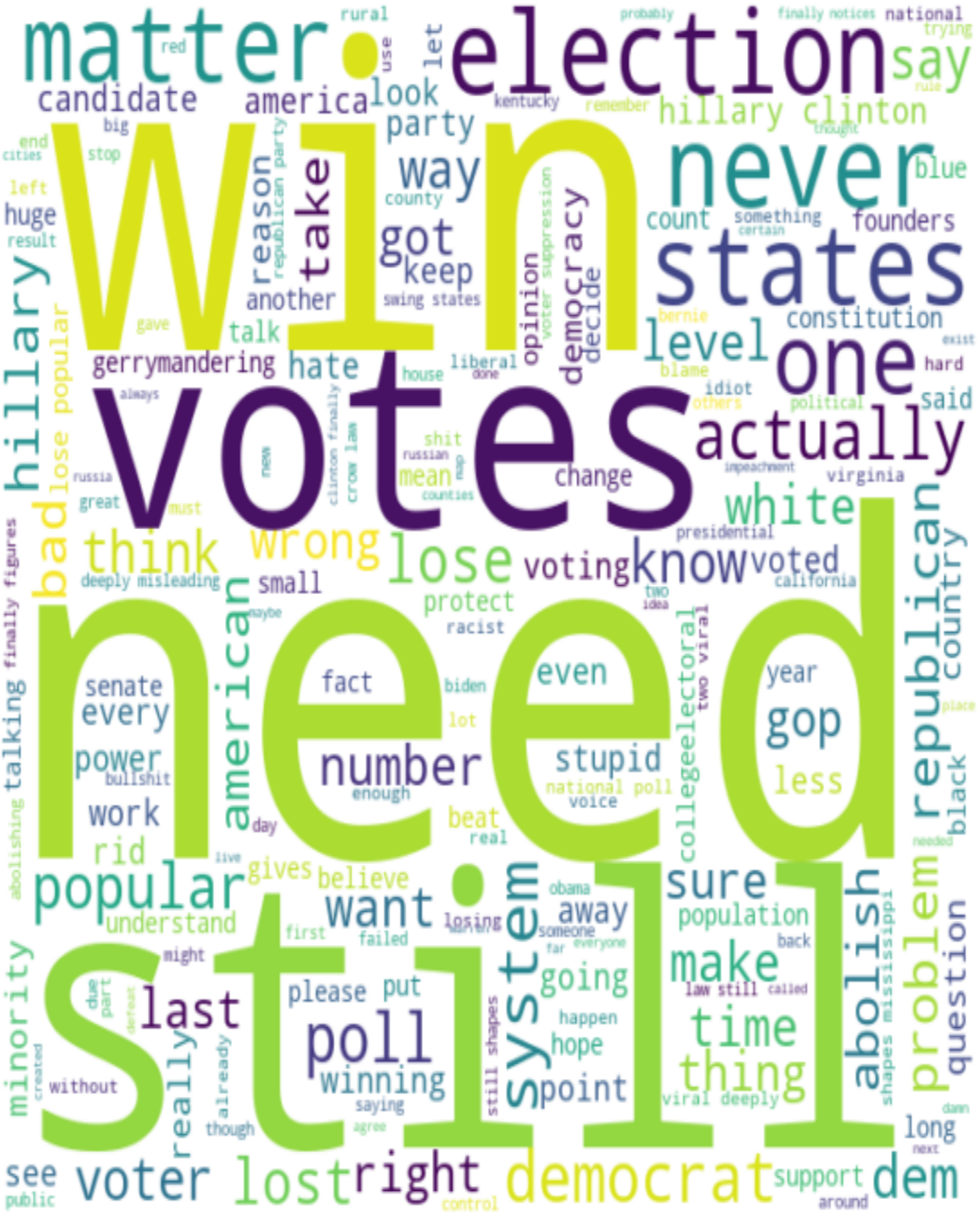
processed text - stemmed, Lemmatized and de-tokenized:
'find public hear let clear trump lost milli'

8. Data Visualization:

A word cloud represents word usage in a document by resizing individual words proportionally to its frequency and then presenting them in a random arrangement. The textual analysis of our tweets provides a general idea of what kind of words are frequent in our corpus.

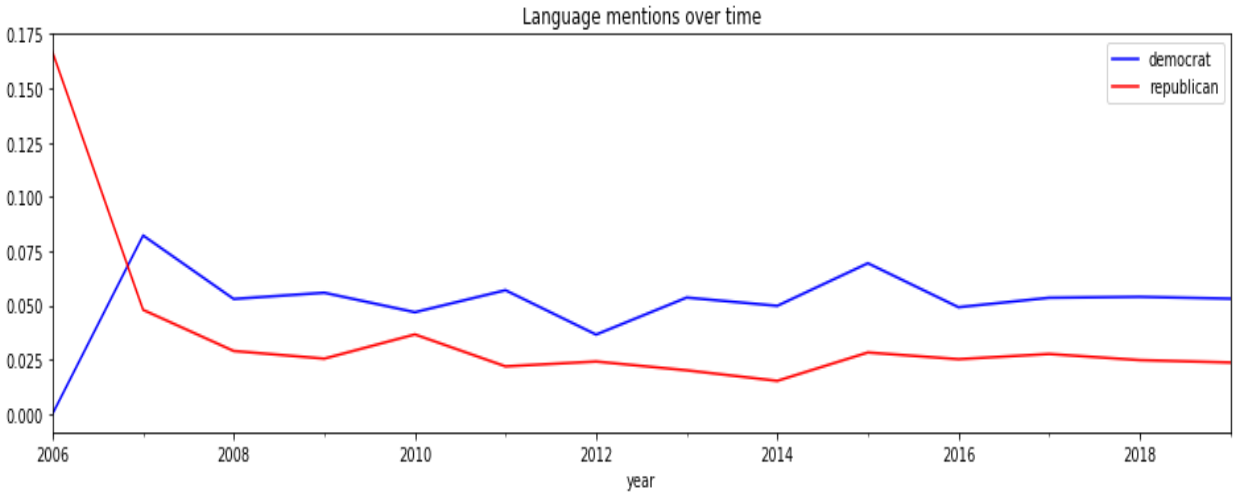
The word cloud visualization of the tweets having positive sentiment, neutral sentiment and negative sentiment is provided below:

Word cloud of negative text

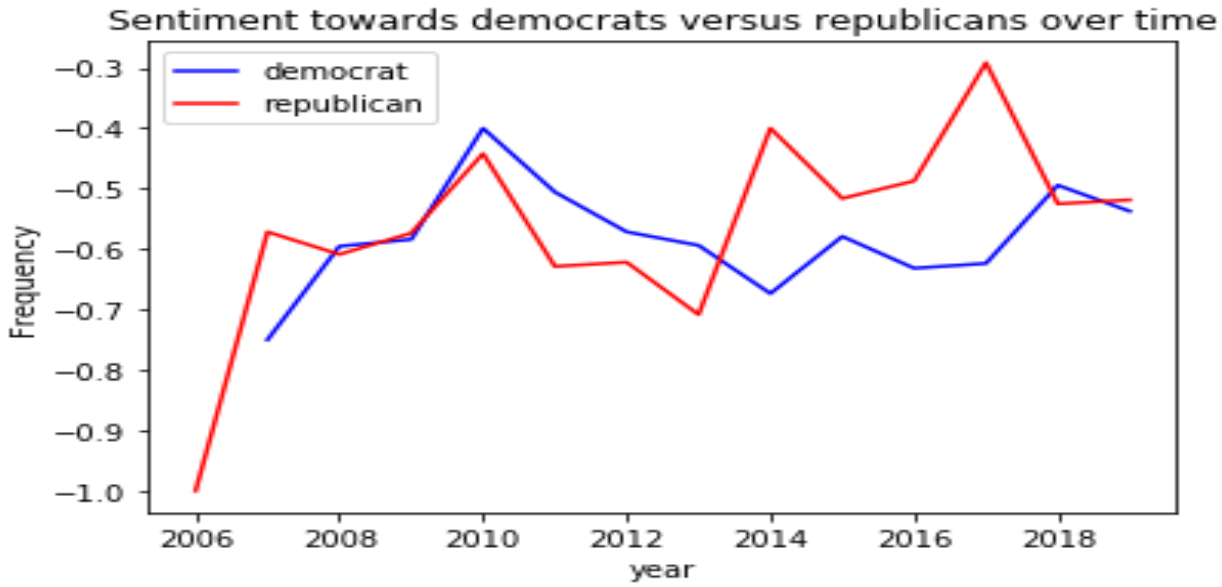


Interesting Trends:

Twitter has a reputation as the most political of all the social media platforms. According to Pew survey, 36% of U.S.-based Twitter users are Democrats and only 21% of Twitter users are Republicans. Here, we are interested to capture the number of times the language mentions of "democrat" and "republicans" occured in our dataset containing tweets associated with electoral college and compare it to sentiment (opinion trend) associated with language mentions in time series.

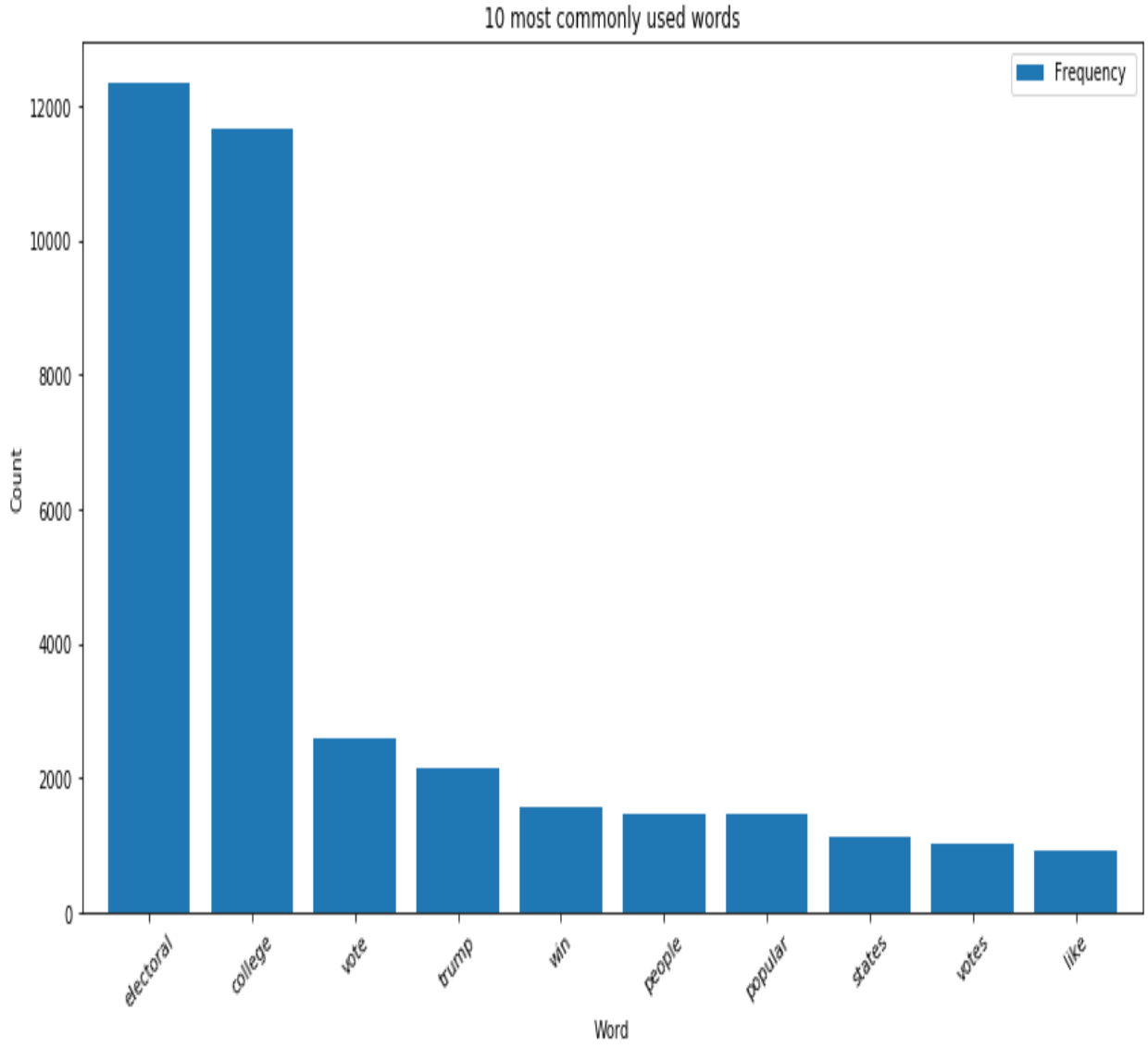


As we can see in the plot above, the term "republican" has been mentioned mostly fewer number of times compared to the term "democrat" except in years 2006 and 2007.

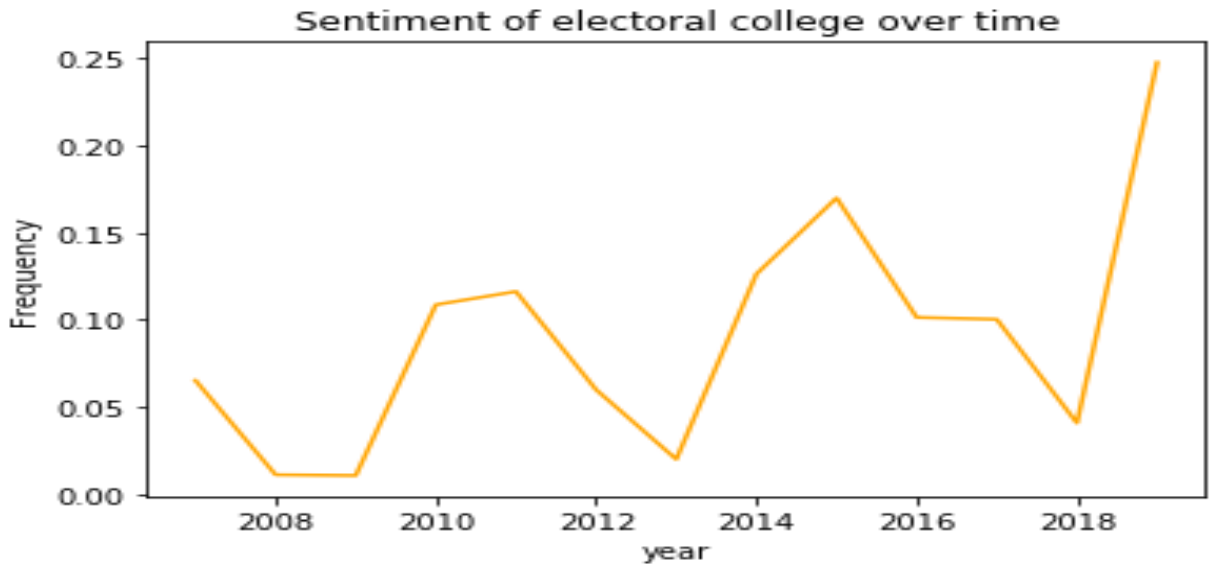


When we compare the opinion trend as seen in the plot above, surprisingly, sentiment toward republicans looks slightly better than democrats just after 2013.

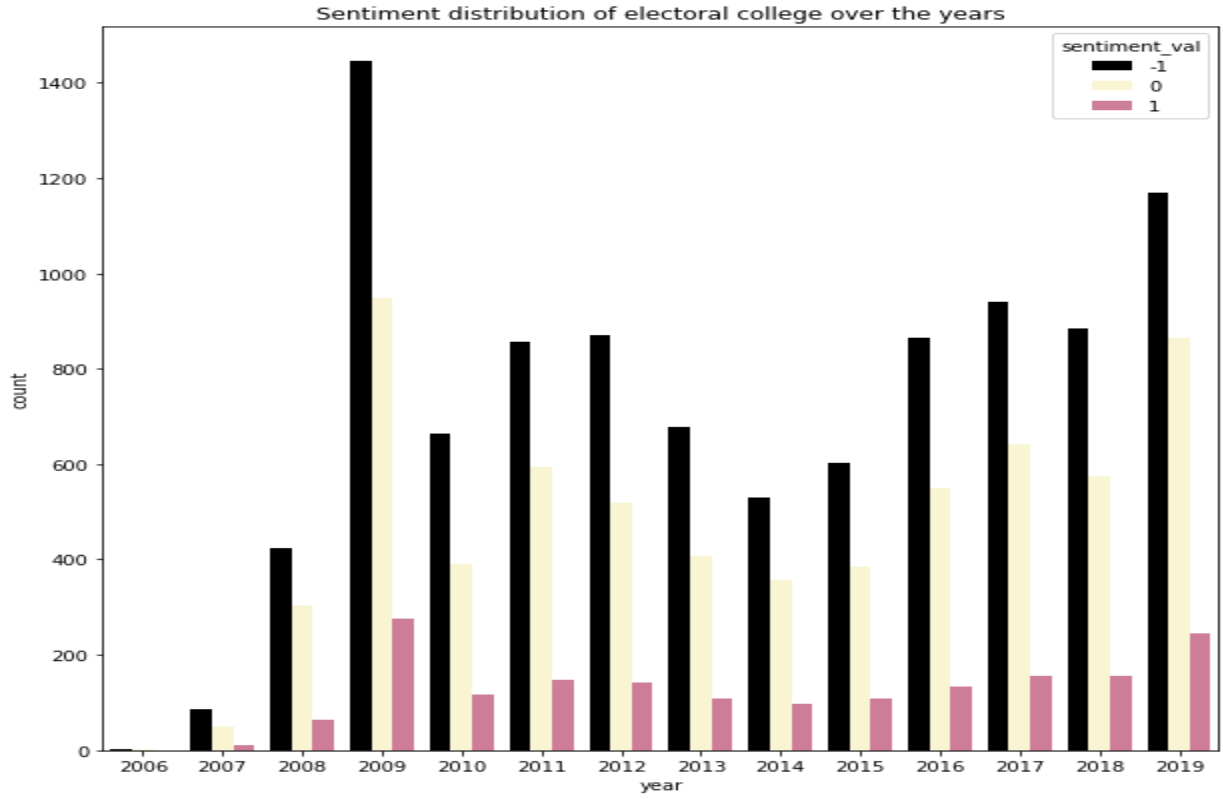
Now, let's explore the 10 most popular words in our data-frame:



Next, let's analyse how sentiment about electoral college change over time in the plot below:



We can see from the plot above that the average sentiment towards electoral college has been negative for the large part comparatively, it peaked in 2015 suggesting it could have been due to 2016 election results. We will now segregate tweets based on sentiment counts and compare it over the years.



9. Inferential Statistics:

Based on the plot above, we can see that there was a noticeable variations in polarity of average sentiment through the years. I would like to to test whether this hypothesis is commensurate with the data, especially from 2014 to 2016. The dataset has twitter data from from 2006 to 2018. The samples derived based on years are independent of each other and are random. I would be performing a two sample Bootstrap hypothesis test with null hypothesis being that there is no difference in means with regards to the average sentiment polarity.

Null Hypothesis:

$Ho \Rightarrow \mu 1 = \mu 2$

Alternate Hypothesis:

$Ha \Rightarrow \mu 1 \neq \mu 2$

The average sentiment polarity for the year 2014 was -0.44, and that of year 2016 was 0.47 with an empirical difference of 0.042. It is possible this observed difference in mean was by chance. We will compute the probability of getting at least a 0.042 difference in average sentiment polarity under the hypothesis that the average sentiment polarity in both years are identical. We want to test the hypothesis that the year 2014 and 2016 have the same average sentiment polarity with respect to electoral college using the two-sample bootstrap test. Here, we shift both arrays to have the same mean, since we are simulating the hypothesis that their means are, in fact, equal. We then draw bootstrap samples out of the shifted arrays and compute the difference in means. This constitutes a bootstrap replicate, and we generate many of them. The p-value will be the fraction of replicates with a difference in means greater than or equal to what was observed. Two-sample bootstrap hypothesis test for difference of means was performed using the following code:

```
def bootstrap_replicate_1d(data, func):
    return func(np.random.choice(data, size=len(data)))
def draw_bs_reps(data, func, size=1):
    """Draw bootstrap replicates."""
    # Initialize array of replicates: bs_replicates
    bs_replicates = np.empty(size)
    # Generate replicates
    for i in range(size):
        bs_replicates[i] = bootstrap_replicate_1d(data, func)
    return bs_replicates                                                    .

# Concatenate sentiments: sent_concat
sent_concat = np.concatenate((sentiment_2014,sentiment_2016))
# Compute mean of all trips: mean_trips
mean_sentiment = np.mean(sent_concat)

# Generate shifted arrays
sentiment_2014_shifted = sentiment_2014 - np.mean(sentiment_2014) + mean_sentiment
sentiment_2016_shifted = sentiment_2016 - np.mean(sentiment_2016) + mean_sentiment

# Compute 10,000 bootstrap replicates from shifted arrays
bs_replicates_2014 = draw_bs_reps(sentiment_2014_shifted , np.mean,size= 10000)
bs_replicates_2016 = draw_bs_reps(sentiment_2016_shifted , np.mean, size=10000)

# Get replicates of difference of means: bs_replicates
bs_replicates = bs_replicates_2014 - bs_replicates_2016

# Compute and print p-value: p
p = np.sum(bs_replicates == empirical_diff_means) / len(bs_replicates)
```

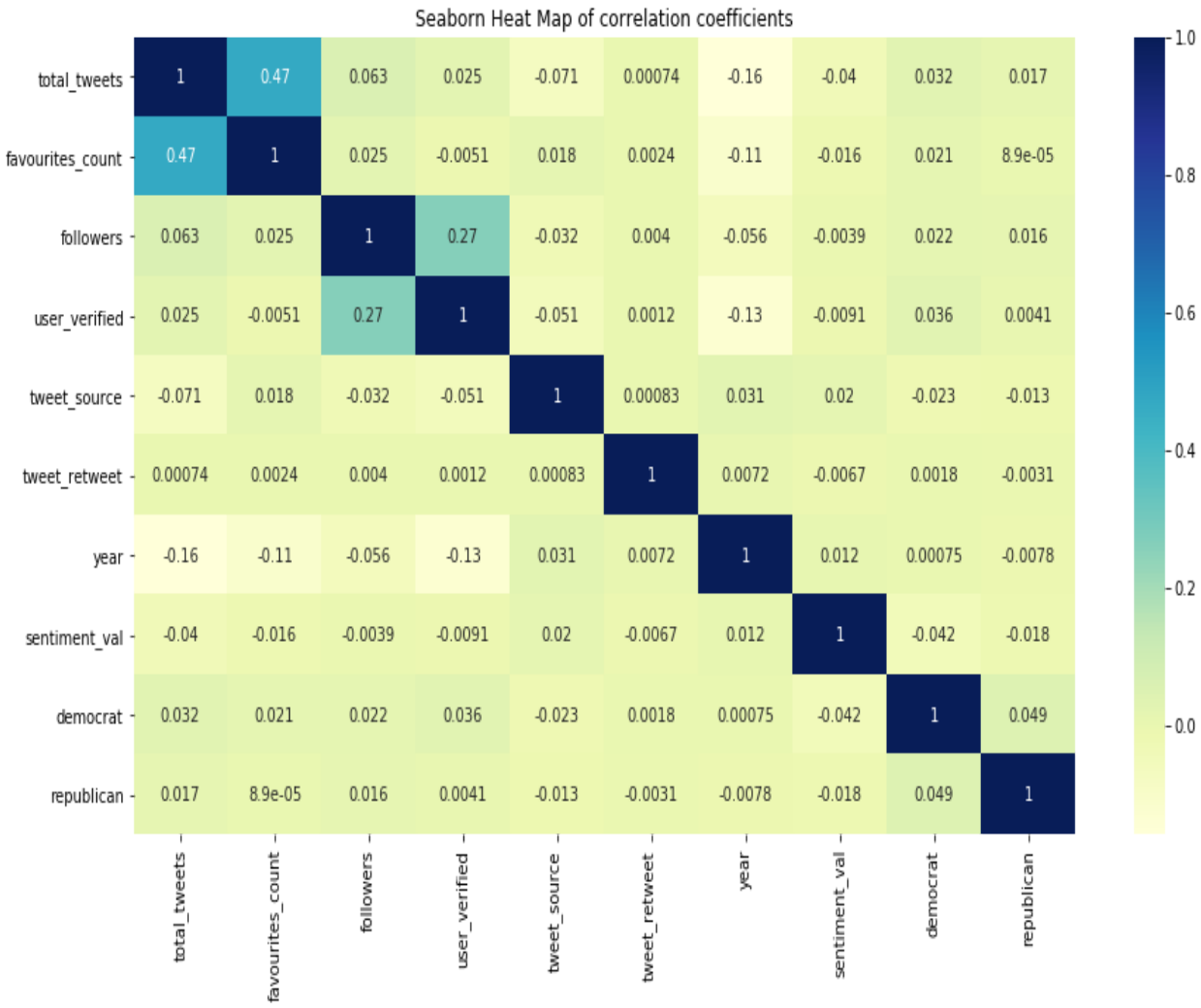
.We got a p-value of 0, which suggests that there is a statistically significant difference in the average sentiment polarity from 2014 to 2016. But it is very important to know how different they are! We got a difference of 0.042 between the means. The difference doesn't seem to be that substantial and there might be few situations where sentiment polarity varies.

Since the p-value is < 0.05, the difference is statistically significant rendering evidence to reject the hypothesis with 95% confidence.

10. Modeling:

In our dataset, we have both text features and numerical values so we need to transform both the text and numerical values differently. First, I tried modeling with Heterogeneous Data Sources (text and categorical) using all features and then modeling with text only classification.

Below is the Seaborn Heatmap of correlation coefficients after encoding categorical columns as numeric columns using sklearn's LabelEncoder:



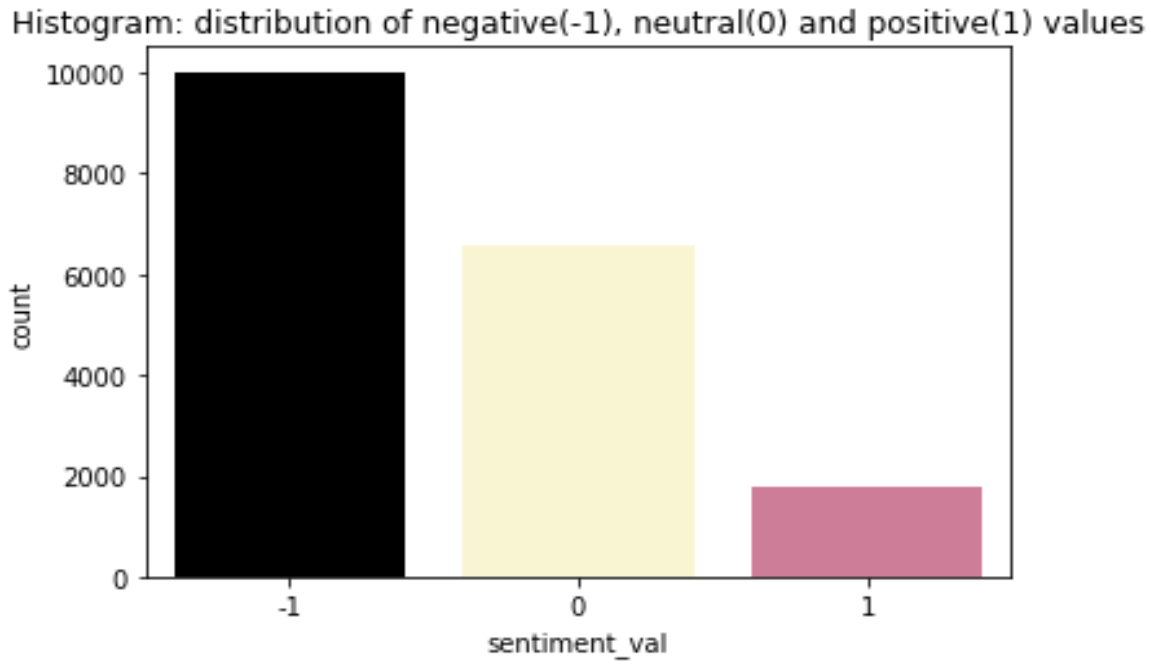
It's good to note that with the color gradient map created, high correlations is displayed in increasing blue shades, and lower correlations trending into the yellow hues. The diagonal plane mirrors itself on either side, with self to self correlations.Our target variable "sentiment_val" does not seem to have either strong positive or strong negative correlation with any of the other numerical features.

10.1 Intensity breakdown:

Overall, there are three sentiment categories - very positive & positive combined as positive category; very negative & negative combined as negative category, and neutral category. The percentage breakdown is given below:

- ➢ Percentage of positive tweets: 9.60%
- ➢ Percentage of neutral tweets: 35.84%
- ➢ Percentage of negative tweets: 54.56%

Label distribution of sentiments can be seen in the plot below:



We have more tweets that are labelled negative compared to neutral and negative tweets.Some labels don't occur very often, but we want to make sure that they appear in both the training and the test sets while modeling. We provide a parameter stratify which means that the train_test_split method returns training and test subsets that have the same proportions of class labels as the input dataset

10.2 Train/Test Split:

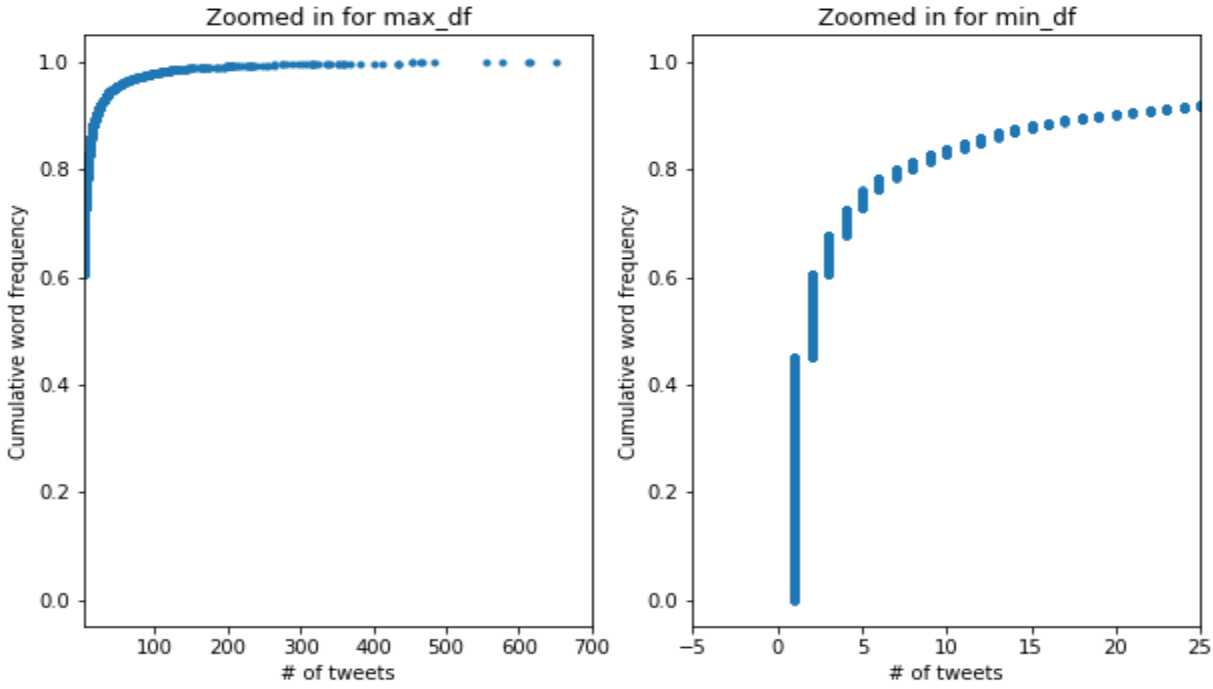
Using train_test_split helper function from scikit-learn library, the dataset was split into random Training and testing sets with 80:20 ratio.

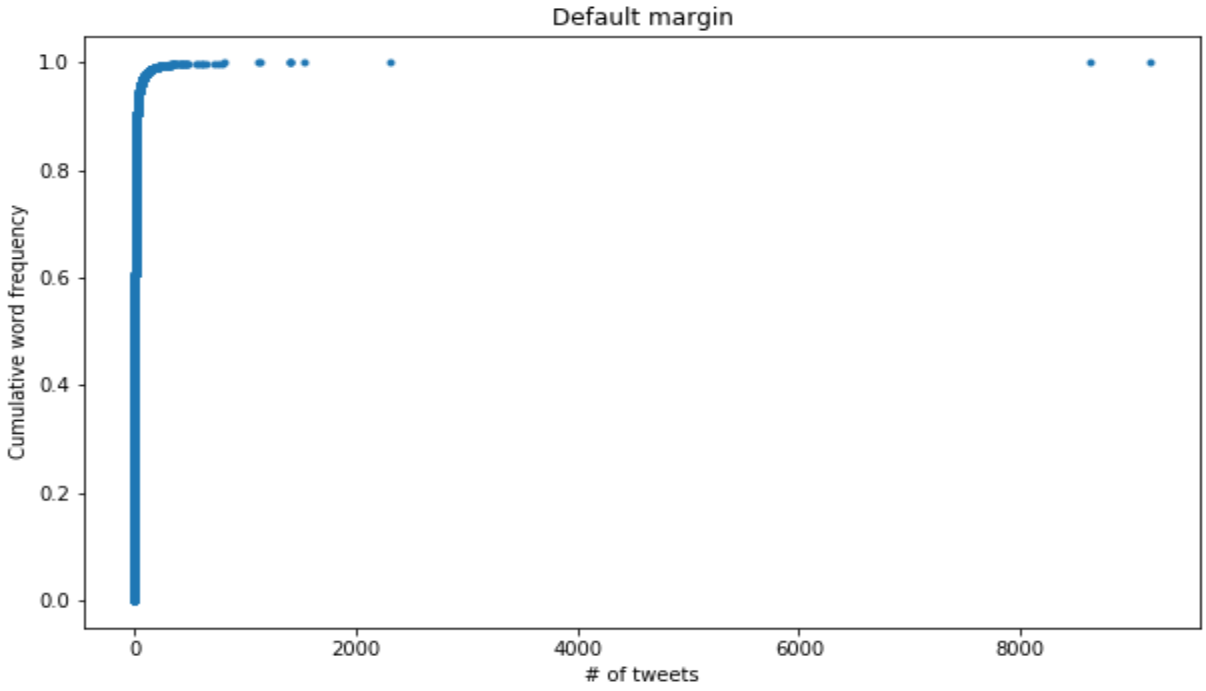
10.3 Feature Engineering:

After splitting the data set, the next steps includes feature engineering. We will convert our text documents to a matrix of token counts (CountVectorizer with n_gram range(1,2)), then transform a count matrix to a normalized tf-idf representation (tf-idf transformer). After that, we train several classifiers from Scikit-Learn library.

Next, we construct the cumulative distribution of document frequency (df). The x-axis is a document count xi and the y-axis is the percentage of words that appear less than xi times. We have to look for the point at which the curve begins climbing steeply. This may be a good value for `min_df` (used for removing terms that appear too infrequently) and for `max_df` (used for removing terms that appear too frequently), we would likely pick the value where the curve starts to plateau. Looking closely at the plot below, the curve rises steeply just around 0. To set the min_df and max_df, we need to see the zoomed in version of the plots by setting axis limits. 15 tweets or less had 85% of words in vocabulary. Min_df is used for removing terms that appear too infrequently., and max_df is

used for removing terms that appear too frequently. The curve starts to plateau from 100 till it reaches 400. So, I think if we set min_df = 15 and max_df = 250, that should cover maximum% of the vocabulary.

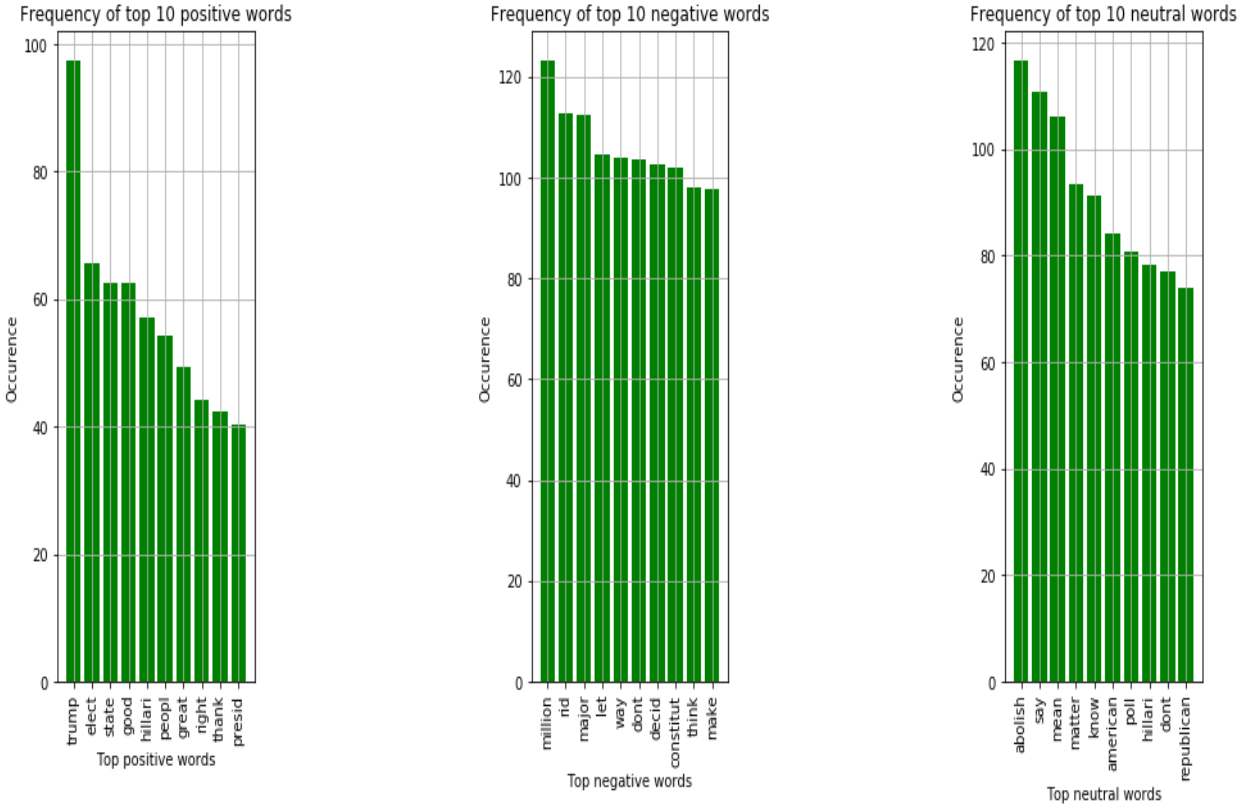




Looking closely at the plot above, the curve rises steeply just around 0. To set the min_df and max_df, we need to see the zoomed in version of the plots by setting axis limits. 15 tweets or less

had 85% of words in vocabulary. Min_df is used for removing terms that appear too infrequently., and max_df is used for removing terms that appear too frequently. The curve starts to plateau from 100 till it reaches 400. So, I think if we set min_df = 15 and max_df = 250, that should cover maximum% of the vocabulary.

Next, we plot the frequencies of top 10 features belonging to positive, negative and neutral categories as below:



10.4 Feature Union with Heterogeneous Data Sources:

Using sklearn.feature_extraction.FeatureUnion and RandomForestClassifier on our dataset, we combine numerical (10 columns) and text features (1 column) for multi-class classification. First,we split the data into Train/Test sets. We then convert text to word count vectors with CountVectorizer using standard bag-of-words features for the "processed_text" column after dropping null entries. To numerical features, we impute missing values.Scikit learn provides a really nice feature for building the model using Pipeline.We create a FeatureUnion with nested pipeline for text and numeric data using FunctionTransformers. We could achieve the accuracy of 0.62 on the test data. We focus on multiclass text classification henceforth.

10.5 Multi-Class Text Classification:

In this section, we train various classification models and compare their performance. After we have our features, we can train classifiers to see how different classifiers predict the tag of a post. We will

start with baseline classifiers first with default parameters and then tune the models with hyperparameters.Since this is multi-class classification, some arguments will have to be changed within each algorithm.Then we initiate a function to log the performance a score table to store the results of all classifiers which actually aids in comparing the results. In this report we try:

1. Logistic regression
2. Support Vector Machine
3. Random Forest Classifier
4. Gradient Boosting Classifier
5. Decision Tree Classifier

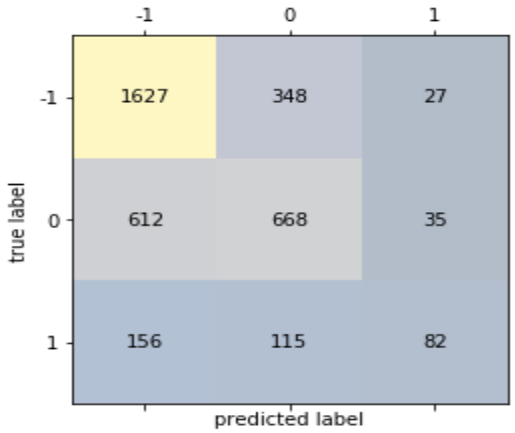
Hyper parameter tuning is undertaken which yields the optimal hyper parameters. We set one of the folds as a test set, and the others as a training set. We take a subset of each of the 4 folds, we gridsearch or RandomizedSearch over the parameters and cross-validate over the reduced folds to pick the optimal hyper-parameters. Using the obtained optimal hyper-parameters, we train the model on the 4 full folds and test on the test fold. Then we plot a confusion matrix and print classification report to get precision, recall, f1-score and support. We also compute the predictive accuracy score across the models below in section 11.

10.5.1 Logistic Regression:

Logistic Regression is a type of Generalized Linear Model (GLM) that uses a logistic function to model a binary variable based on any kind of independent variables. Since this is multi-class classification, some arguments will have to be changed within each algorithm.To fit a multi-class logistic regression with sklearn, we use the LogisticRegression module with multi_class set to "auto" and solver to “liblinear”. The parameters chosen for this algorithm are:

1. C :C- value represents the regularization which controls the trade-off between smooth decision boundary and classifying the training points correctly.
2. Max_iter: Maximum number of iterations taken to converge.

We also use sklearn’s MinMaxScaler to scale all training features between -1, and 1. The performance of the tuned model is also further described in the confusion matrix shown below:



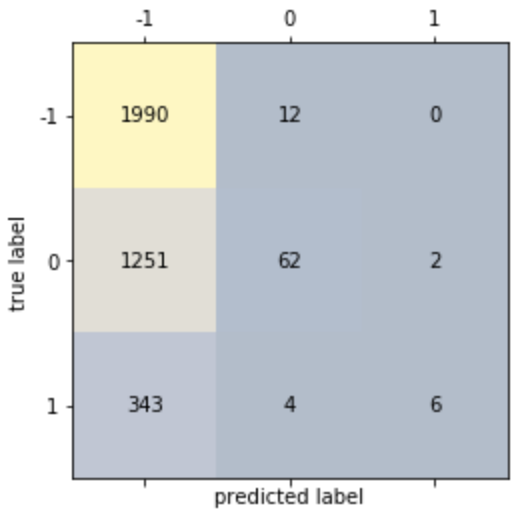
Accuracy score: 0.647

10.5.2: Support Vector Classifier:

Support Vector Classifier was computationally expensive. A pipeline constructor was set up using two steps: the scaling step using MinMax scaler, followed by the instantiation of the classifier which we fit on the training data and predict on the test set. The parameters chosen for this algorithm are:

- C :C- value represents the regularization which controls the trade-off between smooth decision boundary and classifying the training points correctly.
- Gamma: gamma is a parameter of the kernel and can be thought of as the 'spread' of the kernel and therefore the decision region.

The performance of the tuned model is also further described in the confusion matrix shown below:



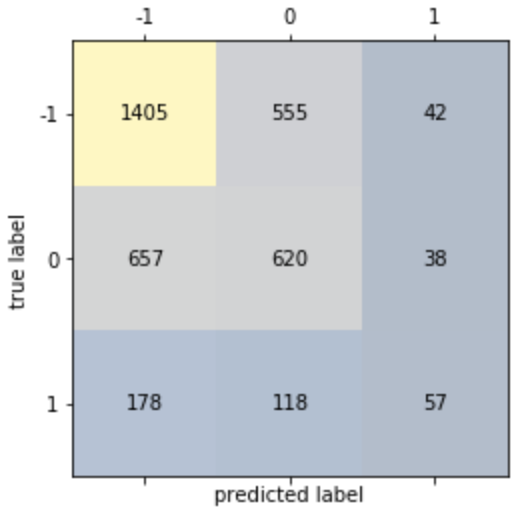
Accuracy score: 0.5608

10.5.3: Random Forest Classifier:

We implement Random Forest classification using the sklearn's RandomForestClassifier class.The parameters chosen for this algorithm are:

- n_estimators : The number of trees in the forest.
- max_features:The number of features to consider when looking for the best split
- max_depth:the maximum depth of the tree.
- criterion:The function to measure the quality of a split

The performance of the tuned model is also further described in the confusion matrix shown below:



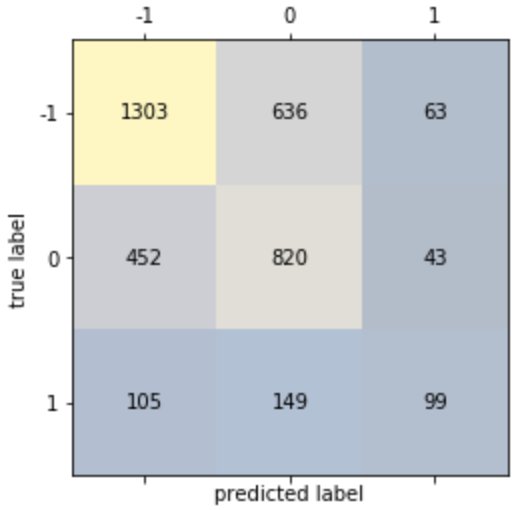
Accuracy score:0.567

10.5.3: Gradient Boosting Classifier:

The parameters chosen for this algorithm are:

- gbc__n_estimators : The number of boosting stages to perform.
- Max_features:The number of features to consider when looking for the best split

The performance of the tuned model is also further described in the confusion matrix shown below:



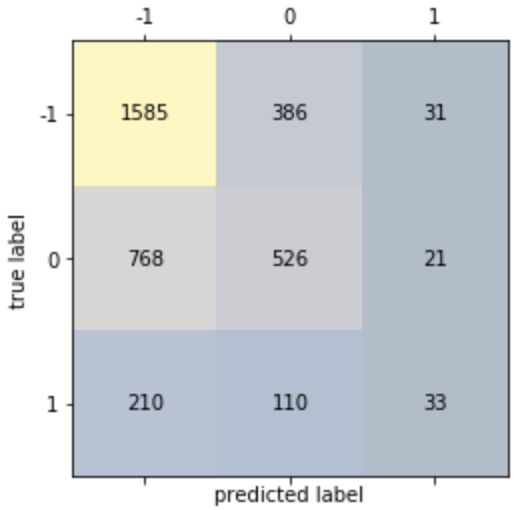
Accuracy score: 0.6054

10.5.4: Decision tree Classifier:

Decision trees have many parameters that can be tuned, such as max_features, max_depth, and. We used the RandomizedSearchCV to find the optimal hyperparameters and then trained on the train set. Predict on the test set as before. The parameters chosen for this algorithm are:

- min_samples_leaf :
- max_features:The number of features to consider when looking for the best split
- max_depth:the maximum depth of the tree.
- criterion:The function to measure the quality of a split

The performance of the tuned model is also further described in the confusion matrix shown below:



Accuracy score:0.547

11. Model Comparison and Selection:

The performance of all models can be compared in the score table below:

	precision	recall	f1-score	support	accuracy
LogisticRegression	0.597777	0.631019	0.613856	1658.5	0.602997
Tuned LogisticRegression	0.63498	0.660336	0.643124	1658.5	0.647684
SVC	0.557968	0.62115	0.579028	1658.5	0.554768
Tuned SVC	0.675059	0.520577	0.400756	1658.5	0.560763
RandomForestClassifier	0.607895	0.658654	0.621494	1658.5	0.602725
Tuned RandomForestClassifier	0.553369	0.586641	0.568942	1658.5	0.567302
GradientBoostingClassifier	0.568056	0.608507	0.587423	1658.5	0.584469
Tuned GradientBoostingClassifier	0.605721	0.637212	0.618212	1658.5	0.60545
Decision Tree	0.552667	0.566342	0.551339	1658.5	0.532425
Tuned DecisionTree	0.483791	0.503195	0.363319	1658.5	0.547139

As we can see from the score table, Precision seems to be better in hypertuned Logistic Regression when compared to all other classifiers. Precision shows how precise/accurate your model is out of those predicted positive, how many of them are actual positive. Recall actually calculates how many of the Actual Positives our model capture through labeling it as Positive. Again, tuned Logistic Regression and RandomForest Classifier seems to do a better job than other models. F1 Score might be a better measure to use if we need to seek a balance between Precision and Recall and there is an uneven class distribution (large number of Actual Negatives). Even here, we can see that tuned Logistic Regression outperformed the rest of the classifiers. If we consider accuracy as the metric, we can see that tuned Logistic Regression outperformed all the other models. Therefore, I think it's best if we choose tuned Logistic regression with hyperparameter tuning as a model of our choice.

12. Conclusion and Limitations:

For each of the multi-class classifiers we present two models and a report on overall accuracy, precision, recall, f1-score for 3-way classification tasks: positive versus negative versus neutral:

- Baseline Model with default parameters
- Tuned Model with hyper parameters

We also present a multi-class classification model by combining heterogeneous data sources using Scikitlearn's Feature Union.

Our model of choice Logistic Regression after hyperparameter tuning gave the accuracy score of 0.65. Sentiment analysis research topic has evolved during the last decade with models reaching the efficiency greater than 85%. I look forward to working with a bigger dataset to improve accuracy by also considering emoticons based on their sentiment polarity for classifying the tweets. There is a lot of work to be done to detect the exact sentiment of texts because of the complexity in the English language and its associated ‘cyberspeak’ language forms.

13. Future work:

In this project, 'LogisticRegression', 'SVC', 'RandomForestClassifier', GradientBoostingClassifier', 'Decision Tree' were used. I would also like to explore other machine learning algorithms like Naive Bayes, MaxEnt and Artificial Neural networks.

14. Acknowledgements:

I would like to express my gratitude towards Springboard for their support in completing the project. This work would not have been possible without the guidance and supervision of my mentor Dipanjan Sarkar.