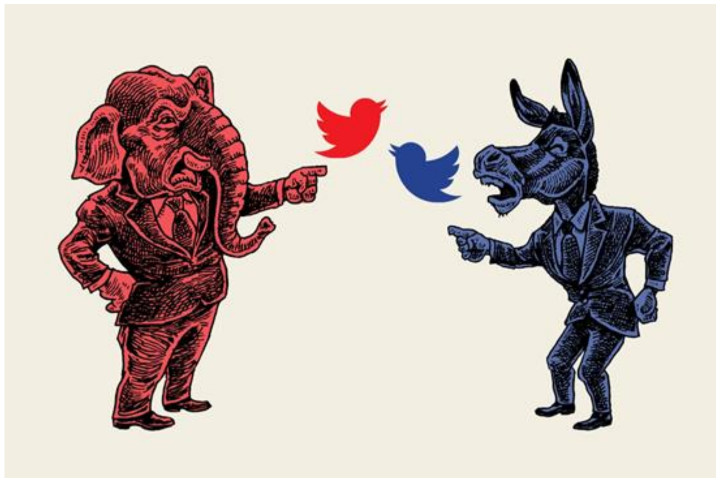
GROUP-323: SENTIMENT ANALYSIS AND CLASSIFICATION OF DEMOCRATS VS REPUBLICANS TWEETS DATA

First Name	Last Name	Email (hawk.iit.edu)	Student ID
Arpitha	Jagadish	ajagadish@hawk.iit.edu	A20453142

Table of Contents

1. Introduction



Sentiment analysis is one of the most interesting NLP topics in which I always wanted to do a project. In the recent trends, the sentiment analysis is extremely useful in social media monitoring as it allows us to gain an overview of the wider public opinion behind certain topic.

As the US election was the main discussion point in the last few months, people are very expressive about their support to Democratic or Republican parties on twitter platform. Just based on some phrases, people say “You sound just like a Democrat!” or “Are you an anti-Republican?” These posts inspired me to search for the extracted tweets dataset to conduct sentiment analysis and use ML algorithms to train a model and determine if the tweet was written by Democrat or a Republican handle/supporter.

2. Data

This dataset is added to kaggle in 2018 by Kyle Pastor. This data set is about political data, which contains extracted tweets handled by representative parties and user information.

- The data set consists of 3 columns –
  - Party : Name of the Party – Democrats or Republicans
  - Handle: Representative from a particular party who post the Tweet or Retweet
  - Tweet: Complete Tweet posted by the party handlers
- It has 86460 tweets by 433 unique handle.
- Party is the label which contains two classes- Democratic and Republican.
- Among the extracted tweets, 51% belongs to Republican Party and 49% belongs to Democrat Party.

Following is the kaggle link to access the dataset:
https://www.kaggle.com/kapastor/democratvsrepublicantweets?select=ExtractedTweets.csv

Python Code for Importing CSV File and verifying the data types:

```
In [153]:  #Load the csv file
           df = pd.read_csv("ExtractedTweets.csv")
           #drop the missing values row wise
           df.dropna(axis = 0, inplace = True)
           df.head()
```

Out[153]:

	Party	Handle	Tweet
0	Democrat	RepDarrenSoto	Today, Senate Dems vote to #SaveTheInternet. P...
1	Democrat	RepDarrenSoto	RT @WinterHavenSun: Winter Haven resident / Al...
2	Democrat	RepDarrenSoto	RT @NBCLatino: .@RepDarrenSoto noted that Hurr...
3	Democrat	RepDarrenSoto	RT @NALCABPolicy: Meeting with @RepDarrenSoto ...
4	Democrat	RepDarrenSoto	RT @Vegalteno: Hurricane season starts on June...

```
In [154]:  # Data Types
           df.dtypes

Out[154]:  Party     object
           Handle    object
           Tweet     object
           dtype: object
```

Figure 1: Structure of Dataset

Counting the number of Tweets extracted from each party:

```
In [159]:  #Count for each label
           df.Party.value_counts()

Out[159]:  Republican    44392
           Democrat      42068
           Name: Party, dtype: int64
```

Analyzing a few Tweets from both the parties to understand the format:

```
In [156]:  #First tweet
           df.Tweet[0]

Out[156]:  'Today, Senate Dems vote to #SaveTheInternet. Proud to support similar #NetNeutrality legislation here in the House… http
           s://t.co/n3tggDLU1L'

In [157]:  df.Tweet[1]

Out[157]:  'RT @WinterHavenSun: Winter Haven resident / Alta Vista teacher is one of several recognized by @RepDarrenSoto for National
           Teacher Apprecia…'

In [158]:  df.Tweet[86459]

Out[158]:  '#Zika fears realized in Florida. House GOP acted to prevent crisis. Dems inaction, inexcusable! Time to put politics aside
           &amp; work together!'
```

3. Problems and Solutions

Below are the research problems we need to solve in this project:

1. What are the Hashtags & words Democrats and Republicans use the most in their tweets?
2. What Sentiments do the tweets have? Are there more negative tweets posted by a particular party?
3. Classification model to determine whether newly posted tweet belongs to Democratic or Republican class
4. Classification of newly entered tweet by the user with the help of Streamlit application

Following are the solutions for the above mentioned problems.

1. **Perform Exploratory Data Analysis to find and visualize the most used hashtags and words used by both the parties.**

    - Each party handles always comes up with a creative hashtags to create the trend. Hashtags in the tweets are like headlines in the newspaper, which plays a key role to differentiate the agenda and principles of two parties.

- The political parties use some of the key words repeatedly to stress their importance to the topic and same words are used by multiple handles while writing the tweet to keep it consistent among party members.
- So, analyzing the most used hashtags and words for a particular party is very important for building the classification models.
- A proper care should be taken during the preprocessing to make sure these keywords will not be eliminated while removing the Stop words.

Top Hashtags Used by Democrats and Republicans:

- The hashtags from each tweet is extracted using the "lambda" function
- The extracted hashtags are classified and assigned to Democrats and Republican party logs
- Plot the Word cloud Image separately for each party to figure out the top hashtags used by Democrats and Republicans

Most used words used by Democrats and Republicans:

- Preprocess the tweets to remove the STOPWORDS
- Perform Lemmatization to get the root word
- Separate the Democrats and Republican tweets and convert It as a list
- Find out the most frequently used words using the python function "FreqDist"
- Create a pie chart using the top 5 words used by each parties and highlight the percentage of usage among the top 5 words.

2. What Sentiments do the tweets have? Are there more negative tweets posted by a particular party?

Sentiment analysis is important to interpret and classify emotions in a subjective data. It utilizes the Natural Language processing (NLP) and Machine learning (ML) techniques to evaluate whether a sentence has positive or negative emotions. Polarity and Subjectivity are the two metrics which helps to analyze the sentiment in a particular sentence.

The **polarity** score is a float within the range **[-1.0, 1.0]**. And it represents emotions expressed in a sentence. **1** means **positive** statement and **-1** means a **negative** statement.

Whereas, **subjectivity** is a float within the range **[0.0, 1.0]** where **0.0** is very **objective** and **1.0** is very **subjective**. It refers to personal opinion, emotion or judgement.

There are certain predefined rules in sentiment analysis which helps to categorize the emotions in better manner

For an Example:

- If there is a prefix as "not" then the standard polarity score will be multiplied by -0.5
- For a prefix "very" the subjectivity gets multiplied by 1.3

- In the Extracted Tweet dataset, the tweets posted by democrats and republicans are separated based on party log
- The overall sentiment of collective tweets for each party is calculated using the "TextBlob" function
- Compare the polarity and subjectivity of Democrats and Republican parties to analyze who had posted more positive tweets and what is the subjectivity score for each party

3. **Classification model to determine whether newly posted tweet belongs to Democratic or Republican class**

   Following steps are followed to build the classification model to predict the political party for a new tweet

   - Preprocess the data by removing unwanted characters like punctuation, hyperlinks, numeric words and STOP words
   - Convert all words to lower case to eliminate the duplicates
   - Also add some of the unwanted and repeated letters to the STOP WORDS dictionary
   - Perform stemming using Lancaster Stemmer to tokenize the sentence into base words
   - Convert the collection of tweets to a matrix of token counts using TF-IDFVectorizer. TF-IDFVectorizer is preferred over Counter vectorizer because,
     - TF helps to weight terms higher if they are frequent in relevant document
     - IDF helps to weight more for rare words and less for common words
     - Provide normalization function

- As the data set is huge, I am using Hold-out evaluation method to build the predictive model using different classification algorithms like Naïve Bayes, Decision Trees.
- Compare the accuracy of different models and use ensemble models like Random Forest and Logistic Regression to improve accuracy.

4. Classification of newly entered tweet by the user with the help of Streamlit application

- Streamlit application is an open source app framework for machine learning and data science applications
- Selection option was assigned to choose between prediction and NLP
- In the Prediction option, the user will enter the tweet in the text box and choose the Machine learning model to figure out to which political party the tweet belongs to
- In the NLP option, the entered tweets will be tokenized and root words are displayed using Lemma function. Also, it has an option to view the Word Cloud image.

4. KDD

4.1. Data Preprocessing

- Importing the packages necessary for Pre-processing:

In [160]:

```
#importing the packages necessary for Data Preprocessing
#Regular expression
import re
#For string operations
import string
#Package for Natural language processing
import nltk
#For lemmatization
from nltk.stem.wordnet import WordNetLemmatizer
#To remove the stop words
from wordcloud import STOPWORDS
```

- Counting the number of unwanted characters present in the extracted tweets:

```
In [161]:  #counting those words which we don't need
           #initializaing each count to 0
           url_count = 0
           punc_count = 0
           number_count = 0
           mention_count = 0
           other_than_character_count = 0


           for i in range(len(df.Tweet)):#run the for loop till the end of tweet

             url_re = re.findall('http\S+',df.Tweet[i])
             url_count += len(url_re)
             punc_re = re.findall('[%s]',df.Tweet[i])
             punc_count += len(punc_re)
             num_re = re.findall('(\d+)', df.Tweet[i])
             number_count += len(num_re)
             mention_re = re.findall('@(\w+)', df.Tweet[i])
             mention_count += len(mention_re)
             alpha_re = re.findall("[^a-zA-Z]", df.Tweet[i])
             other_than_character_count += len(alpha_re)

           print ("Count of url in tweet:", url_count)
           print ("Count of punctuation in tweet:",punc_count)
           print ("Count of numbers in tweet:",number_count)
           print ("Count of mentions in tweet:",mention_count)
           print ("Count of alphanum in tweet:",other_than_character_count)

           Count of url in tweet: 68405
           Count of punctuation in tweet: 537631
           Count of numbers in tweet: 128691
           Count of mentions in tweet: 64786
           Count of alphanum in tweet: 2465479
```

- Converting all words to lower case
- Removing Hyperlinks and Punctuations
- Eliminating Numeric Words
- Replacing all characters other than alpha with a blank space

```
In [162]:  def round1(data_str):
               url_re = re.compile('http\S+')
               punc_re = re.compile('[%s]' % re.escape(string.punctuation))
               num_re = re.compile('(\d+)')
               alpha_re = re.compile("[^a-zA-Z]")
               # convert to lowercase
               data_str = data_str.lower()
               # remove hyperlinks
               data_str = url_re.sub(' ', data_str)
               # remove punctuation
               data_str = punc_re.sub(' ', data_str)
               # remove numeric 'words'
               data_str = num_re.sub(' ', data_str)
               data_str = alpha_re.sub(' ', data_str)
               return data_str
```

- Tokenize the sentence and Performing Lemmatization to get the root word:

In [163]:
```
#Performing lemmatization
def lemmatize(data_str):
    #initial index position for string 0
    list_pos = 0
    #intialize empty string to store the lemma words
    cleaned_str = ''
    #tokenize the sentence
    data_str = nltk.word_tokenize(data_str)
    lemma = nlp.WordNetLemmatizer()
    #for each word in sentence run a for loop and find the lemma word and add it to cleaned_str
    for word in data_str:
        lem = lemma.lemmatize(word)
        if list_pos == 0:
            cleaned_str = lem
        else:
            cleaned_str = cleaned_str + ' ' + lem
        list_pos += 1
    return cleaned_str
```

- Adding some of the common words to the STOPWORDS dictionary and removing STOPWORDS from extracted Tweets

In [164]:
```
#adding some more words to stopwords which was found during EDA process
STOPWORDS.add("rt")
STOPWORDS.add("s")
STOPWORDS.add("u")
STOPWORDS.add("amp")
STOPWORDS.add("th")
STOPWORDS.add("will")
STOPWORDS.add("t")
STOPWORDS.add("m")
STOPWORDS.add("ha")
STOPWORDS.add("wa")
```

In [165]:
```
# Removing stop words
def remove_stops(data_str):
    list_pos = 0
    cleaned_str = ''
    text = data_str.split()
    for word in text:
        #if word is not present in stopword add that to cleaned_str
        if word not in STOPWORDS:
            if list_pos == 0:
                cleaned_str = word
            else:
                cleaned_str = cleaned_str + ' ' + word
            list_pos += 1
    return cleaned_str
```

- Cleaning the Tweets data and assigning it to column “Clean_Tweet” in the data frame

In [166]:
```
#Cleaning the data
#empty list
data_clean = []
for i in range(len(df.Tweet)):
    #removing puntuation, links, numbers
    res = round1(df.Tweet[i])
    #removing the stopwords
    res1 = remove_stops(res)
    #performing lemmatization
    res2 = lemmatize(res1)
    data_clean.append(res2)
```

In [169]:
```
#add the clean tweets to dataframe
df['clean_Tweet'] = data_clean
```

In [177]:
```
df.clean_Tweet
```

Out[177]:
```
0        today senate dems vote savetheinternet proud s...
1        winterhavensun winter haven resident alta vist...
2        nbclatino repdarrensoto noted hurricane maria ...
3        nalcabpolicy meeting repdarrensoto thanks taki...
4        vegalteno hurricane season start june st puert...
                               ...
86455    check op ed need end executive overreach act w...
86456    yesterday betty great time learning forestry i...
86457      forever grateful service sacrifice major barney
86458    happy first day school cobbschools cobbbacktos...
86459    zika fear realized florida house gop acted pre...
Name: clean_Tweet, Length: 86460, dtype: object
```

- Displaying the data frame contains both “Extracted tweet” and “Clean_Tweet” information

In [58]:
```
df
```

Out[58]:

	Party	Handle	Tweet	Party_log	clean_Tweet
0	Democrat	RepDarrenSoto	Today, Senate Dems vote to #SaveTheInternet. P...	1	today senate dems vote savetheinternet proud s...
1	Democrat	RepDarrenSoto	RT @WinterHavenSun: Winter Haven resident / Al...	1	winterhavensun winter haven resident alta vist...
2	Democrat	RepDarrenSoto	RT @NBCLatino: .@RepDarrenSoto noted that Hurr...	1	nbclatino repdarrensoto noted hurricane maria ...
3	Democrat	RepDarrenSoto	RT @NALCABPolicy: Meeting with @RepDarrenSoto ...	1	nalcabpolicy meeting repdarrensoto thanks taki...
4	Democrat	RepDarrenSoto	RT @Vegalteno: Hurricane season starts on June...	1	vegalteno hurricane season start june st puert...
...	...	...	...	...	...
86455	Republican	RepTomPrice	Check out my op-ed on need for End Executive O...	0	check op ed need end executive overreach act w...
86456	Republican	RepTomPrice	Yesterday, Betty & I had a great time lear...	0	yesterday betty great time learning forestry i...
86457	Republican	RepTomPrice	We are forever grateful for the service and sa...	0	forever grateful service sacrifice major barney
86458	Republican	RepTomPrice	Happy first day of school @CobbSchools! #CobbB...	0	happy first day school cobbschools cobbbacktos...
86459	Republican	RepTomPrice	#Zika fears realized in Florida. House GOP act...	0	zika fear realized florida house gop acted pre...

86460 rows × 5 columns

- Serialized object to pickle file, so that the further analysis can easily retrieve the processed information

In [59]:
```
import pickle
```

In [60]:
```
#serialize object to file, which can be used later
df.to_pickle("corpus.pkl")
```

4.2 Data Mining Methods and Processes

4.2.1 Perform Exploratory Data Analysis to find and visualize the most used hashtags and words used by both the parties.

Exploratory Data Analysis:

- Retrieving processed information from Pickle file

```
In [2]:  import pandas as pd

In [35]: #read the preprocessed data from the pickle file
         data=pd.read_pickle("corpus.pkl")
```

- Importing the packages necessary for EDA

```
In [6]: #load the packges required for EDA
        from wordcloud import WordCloud, STOPWORDS
        import matplotlib.pyplot as plt
        from pandas import Series
        import seaborn as sns
        import re
        from nltk.tokenize import word_tokenize
        import matplotlib.ticker as ticker
        import matplotlib.cm as cm
        import matplotlib as mpl
        from matplotlib.gridspec import GridSpec
        import numpy as np
        %matplotlib inline
```

→ Determining the Top Hashtags used by Democrats and Republicans

- Collecting all hashtags from extracted Tweet Data

```
In [6]: #load the packges required for EDA
        from wordcloud import WordCloud, STOPWORDS
        import matplotlib.pyplot as plt
        from pandas import Series
        import seaborn as sns
        import re

In [7]: def hash_tags(tweet):
                #find all the hash tags in the tweets
                hashtag = re.findall('(\#[A-Za-z_]+)', tweet)
                #if there is hash tag return them
                if hashtag:
                    return hashtag
                else:
                     return ""

In [8]: df_h = data
        df_h['top_hashtags'] = df_h['Tweet'].apply(lambda x:hash_tags(x))

In [9]: df_h['top_hashtags']

Out[9]: 0        [#SaveTheInternet, #NetNeutrality]
        1
        2
        3                          [#NALCABPolicy]
        4
                               ...
        86455
        86456
        86457
        86458                  [#CobbBackToSchool]
        86459                             [#Zika]
        Name: top_hashtags, Length: 86460, dtype: object
```

- Assigning Republicans hashtags to “hashtags_rep” and Democrats hashtags to “hashtags_dem”
- Add the count to figure out the top hashtags by each party

```
In [10]:  #initial empty lists for each party
hashtags_rep = []
hashtags_dem = []

for n in range(len(df_h['top_hashtags'])):
    #if party is 0 then republican else democrat hashtag
    if df_h['Party_log'][n] == 0:
        hashtags_rep += df_h['top_hashtags'][n]
    elif df_h['Party_log'][n] == 1:
        hashtags_dem += df_h['top_hashtags'][n]
```

→ Determining the Top Words used by Democrats and Republicans

- Assigning Extracted Tweets to respective parties

```
In [23]:  #assiging tweets to respective parties
democrat_tweets=data[data.Party=="Democrat"]
republican_tweets=data[data.Party=="Republican"]
```

- Converting the preprocessed Democrat Tweets and Republican Tweets to list

```
In [24]:  #Converting the preprocessed democrat_tweets to list
democrat_Tweets=democrat_tweets.clean_tweet.tolist()
#split into words
democrat_tweets=str(democrat_Tweets).split()
#remove the ,[,],' from the data
democrat_tweets=[word.replace("'","") for word in democrat_tweets ]
democrat_tweets=[word.replace("[", "") for word in democrat_tweets ]
democrat_tweets=[word.replace("]","") for word in democrat_tweets ]
democrat_tweets=[word.replace(",", "") for word in democrat_tweets ]
```

```
In [26]:  #similar way do it for republican
republican_Tweets=republican_tweets.clean_tweet.tolist()
republican_tweets=str(republican_Tweets).split()
republican_tweets=[word.replace("'","") for word in republican_tweets ]
republican_tweets=[word.replace("[", "") for word in republican_tweets ]
republican_tweets=[word.replace("]","") for word in republican_tweets ]
republican_tweets=[word.replace(",", "") for word in republican_tweets ]
```

- Determine the Frequency of each word from Democrat tweets. How many times each unique word is used in the text.

```
In [40]:  from nltk.probability import FreqDist
#FreqDist records the number of times each words are used.
fdist_democrat = FreqDist(democrat_tweets)
print("Frequency of each Word in Democrats")
fdist_democrat

Frequency of each Word in Democrats

Out[40]: FreqDist({'today': 3850, 'trump': 2502, 'american': 2053, 'year': 1835, 'thank': 1777, 'family': 1694, 'great': 1676, 'stude
nt': 1660, 'day': 1571, 'congress': 1518, ...})
```

- Determine the Frequency of each word from Republican tweets

```
In [41]: fdist_republican=FreqDist(republican_tweets)
         print("Frequency of each Word in Republicans")
         fdist_republican

         Frequency of each Word in Republicans

Out[41]: FreqDist({'today': 4883, 'tax': 2885, 'great': 2876, 'house': 2382, 'bill': 2379, 'american': 2241, 'thank': 2118, 'year': 1
         961, 'day': 1887, 'act': 1820, ...})
```

The word "Today" is common top word between both the parties. This indicates that the both the party uses update their tweets on daily basis. As this is a common word, this wouldn't help to differentiate the tweet by particular parties. So, this could be added to the STOPWORDS list.

4.2.2 What Sentiments do the tweets have? Are there more negative tweets posted by a particular party?

- Import TextBlob package to perform sentiment analysis. TextBlob finds all the words and phrases that it can assign a polarity and subjectivity to, and averages all of them together

```
In [48]: from textblob import TextBlob
```

```
In [49]: democrat_tweets=data[data.Party=="Democrat"]
         republican=data[data.Party=="Republican"]
```

- Sentiment analysis for Democrats and Republican Tweets

```
In [51]: #Sentiment analysis-Democrat Tweets
         democratblob=TextBlob(str(democrat_tweets))
         democratblob.sentiment

Out[51]: Sentiment(polarity=0.34027777777777785, subjectivity=0.4555555555555555)
```

```
In [52]: #Sentiment analysis-Republican Tweets
         republicanblob=TextBlob(str(republican_tweets))
         republicanblob.sentiment

Out[52]: Sentiment(polarity=0.19837398561739605, subjectivity=0.4590746992419168)
```

From the results,

- The Democrats handlers had posted more positive tweets (Polarity= 0.34) than republican members (Polarity=0.19)
- The Subjectivity score is pretty much the same for both the parties (Subjectivity =0.45)

4.2.3 Classification model to determine whether newly posted tweet belongs to Democratic or Republican class

- Import packages and retrieving processed information from Pickle file
- Divide and assign the data to training and test set (70% - Training, 30% - Test)

```
In [29]: import pandas as pd
         from wordcloud import STOPWORDS
```

```
In [32]: df=pd.read_pickle("corpus.pkl")
```

```
In [33]: from sklearn.model_selection import train_test_split
```

```
In [34]: train, test = train_test_split(df, test_size=0.3, train_size=0.7, random_state=14)
         train.shape, test.shape

Out[34]: ((60521, 5), (25938, 5))
```

- Import packages and perform Lancaster Stemming and Tfidf Vectorization

```
In [35]: #from sklearn.feature_extraction.text import CountVectorizer
         from sklearn.feature_extraction.text import TfidfVectorizer
         from nltk.stem.lancaster import LancasterStemmer
         import nltk
```

- Used LancasterStemmer over PorterStemmer because of the data size and considering processing time

```
In [36]: st = LancasterStemmer()
         def token(text):
             txt = nltk.word_tokenize(text.lower())
             return [st.stem(word) for word in txt]
```

- Used Tfidf vectorizer to build document term matrix

```
tfv = TfidfVectorizer(tokenizer=token,stop_words=STOPWORDS,
                      analyzer=u'word', min_df=4,norm='l2',use_idf=True)
X_train_tfv = tfv.fit_transform(train['clean_tweet']) # fit_transform learns the vocab and one-hot encodes
X_test_tfv = tfv.transform(test['clean_tweet']) # transform uses the same vocab and one-hot encodes
# print the dimensions of the training set (text messages, terms)
print(X_train_tfv.shape)
```

- Assigning Vectorized model to data frame and displaying the feature table

```
In [45]:  X_train_tfv = pd.DataFrame(X_train_tfv.toarray(), columns=tfv.get_feature_names())
          X_test_tfv = pd.DataFrame(X_test_tfv.toarray(), columns=tfv.get_feature_names())
          Y_train = train['Party_log']
          Y_test = test['Party_log']
```

```
In [46]:  X_train_tfv
```

Out[46]:

	aaas	aaf	aajc	aan	aap	aapiequalpay	aapiheritagemon	aaron	aarp	aarpadvoc	...	zink	zion	zip	zoe	zon	zte	zuckerb	zuckerberg	zucke
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
60516	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
60517	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
60518	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
60519	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
60520	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

60521 rows × 8830 columns

- Importing Packages to run classification models and model evaluation

```
In [47]:  from sklearn.ensemble import RandomForestClassifier
          from sklearn.naive_bayes import MultinomialNB
          from sklearn.linear_model import LogisticRegression
          from sklearn.ensemble import AdaBoostClassifier
          from sklearn.tree import DecisionTreeClassifier
```

Multinominal Naïve Bayes Model

```
In [50]:  #Naive Bayes
          nb = MultinomialNB()
          # Train the model
          nb.fit(X_train_tfv, train['Party_log'])
          # Take the model that was trained on the X_train_tfv data and apply it to the X_test_tfv data
          y_pred_tfv_nb = nb.predict(X_test_tfv)
          y_pred_tfv_nb

Out[50]:  array([0, 1, 1, ..., 0, 0, 1], dtype=int64)
```

Ensemble Models:

Logistic Regression

```
In [73]:  #Logistic regression model
          lr = LogisticRegression()
          # Train the model
          lr.fit(X_train_tfv,train['Party_log'])
          # Take the model that was trained on the X_train_tfv data and apply it to the X_test_tfv data
          y_pred_tfv_lr = lr.predict(X_test_tfv)
          y_pred_tfv_lr
```

AdaBoost Classification

```
In [19]:
#AdaBoost Classification
ac = AdaBoostClassifier(n_estimators = 100)
ac.fit(X_train_tfv, train['Party_log'])
y_pred_tfv_ada = ac.predict(X_test_tfv)
```

Random Forest Classification

```
In [14]:
#Random Forest Classification
rfc = RandomForestClassifier(n_estimators=100, n_jobs=-1)
rfc.fit(X_train_tfv,train['Party_log'])
y_pred_tfv_rfc=rfc.predict(X_test_tfv)
```

4.2.4 Classification of newly entered tweet by the user with the help of Streamlit application

- Importing packages to run Streamlit app.

```

# -*- coding: utf-8 -*-
"""
Created on Mon Nov 16 14:56:06 2020

@author: Arpitha Jagadish
"""

import pandas as pd
import re
import nltk
nltk.download("punkt")


import string
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')

import matplotlib.pyplot as plt
from wordcloud import WordCloud
from PIL import Image

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression

#import pickle
#import io
#import joblib
import spacy

nlp = spacy.load("en_core_web_sm")

import streamlit as st
#import neattext as nt
#from neattext.functions import clean_text
from wordcloud import STOPWORDS
from nltk.stem.lancaster import LancasterStemmer


def main():
```

- When the prediction activity is chosen the following code executes

```python
st.title("Sentiment Analysis ")
st.title("Democrats vs Republicans Twitter Data")
# available NlP techniques
activities=["Prediction","NLP"]

#using streamlit sidebar option
choice = st.sidebar.selectbox("Select Activity",activities)

#if prediction is chosen
if choice == "Prediction":
    #read the text from the text_area
    Tweet_text = st.text_area("Enter Text","Type Here")

    #cleaning the tweet entered
    url_re = re.compile('http\S+')
    punc_re = re.compile('[%s]' % re.escape(string.punctuation))
    num_re = re.compile('(\d+)')
    alpha_num_re = re.compile("[^a-zA-Z]")
    # convert to lowercase
    Tweet_text = Tweet_text.lower()
    # remove hyperlinks
    Tweet_text = url_re.sub(' ', Tweet_text)
    # remove puncuation
    Tweet_text = punc_re.sub(' ', Tweet_text)
    # remove numeric 'words'
    Tweet_text = num_re.sub(' ', Tweet_text)
    Tweet_text = alpha_num_re.sub(' ', Tweet_text)

    #just considering the model with highest accuracy, can include other models
    all_ml_models = ["MNB","LRM"]
    #display the models using streamlit selectbox
    model_choice=st.selectbox("Choose ML Model",all_ml_models)
```

```python
    #streamlit button
    if st.button("Classify"):

        #displaying the preprocessed data
        st.text("Pre-Processed Data (stop words will be removed while creating document term matrix(tfidf Vectorizer))::\n{}".format([Tweet_text]))

        #if statement runs depending on the model chosen
        if model_choice == "MNB":

            st.success("You have chosen Multinominal Naive Bayes model")
            #function loaddata returns the predicted party
            prediction = loaddata([Tweet_text],model_choice)


            if prediction == "demo":
                #display the results

                st.success('Party:{}'.format("Democrat"))
                #path for the image
                image='Images/Democrat.jpg'
                img=Image.open(image)
                #display the image
                st.image(img,width=300)

            else:
                st.success('Party:{}'.format("Republican"))
                image='Images/Republican.jpg'
                img=Image.open(image)
                st.image(img,width=300)

        if model_choice == "LRM":

            st.success("You have chosen Logistic Regression model")
            #function loaddata returns the predicted party
            prediction = loaddata([Tweet_text],model_choice)


            if prediction == "demo":
                #display the results
                st.success('Party:{}'.format("Democrat"))
                #path for the image
                image='Images/Democrat.jpg'
                img=Image.open(image)
                #display the image
                st.image(img,width=300)
            else:
                st.success('Party:{}'.format("Republican"))
                image='Images/Republican.jpg'
                img=Image.open(image)
                st.image(img,width=300)
```

- When the prediction activity is chosen the following code executes

```
st.title("Sentiment Analysis ")
st.title("Democrats vs Republicans Twitter Data")
# available NlP techniques
activities=["Prediction","NLP"]

#using streamlit sidebar option
choice = st.sidebar.selectbox("Select Activity",activities)

#if prediction is chosen
if choice == "Prediction":
    #read the text from the text_area
    Tweet_text = st.text_area("Enter Text","Type Here")

    #cleaning the tweet entered
    url_re = re.compile('http\S+')
    punc_re = re.compile('[%s]' % re.escape(string.punctuation))
    num_re = re.compile('(\d+)')
    alpha_num_re = re.compile("[^a-zA-Z]")
    # convert to lowercase
    Tweet_text = Tweet_text.lower()
    # remove hyperlinks
    Tweet_text = url_re.sub(' ', Tweet_text)
    # remove puncuation
    Tweet_text = punc_re.sub(' ', Tweet_text)
    # remove numeric 'words'
    Tweet_text = num_re.sub(' ', Tweet_text)
    Tweet_text = alpha_num_re.sub(' ', Tweet_text)

    #just considering the model with highest accuracy, can include other models
    all_ml_models = ["MNB","LRM"]
    #display the models using streamlit selectbox
    model_choice=st.selectbox("Choose ML Model",all_ml_models)
```


```
st.title("Sentiment Analysis ")
st.title("Democrats vs Republicans Twitter Data")
# available NlP techniques
activities=["Prediction","NLP"]

#using streamlit sidebar option
choice = st.sidebar.selectbox("Select Activity",activities)

#if prediction is chosen
if choice == "Prediction":
    #read the text from the text_area
    Tweet_text = st.text_area("Enter Text","Type Here")

    #cleaning the tweet entered
    url_re = re.compile('http\S+')
    punc_re = re.compile('[%s]' % re.escape(string.punctuation))
    num_re = re.compile('(\d+)')
    alpha_num_re = re.compile("[^a-zA-Z]")
    # convert to lowercase
    Tweet_text = Tweet_text.lower()
    # remove hyperlinks
    Tweet_text = url_re.sub(' ', Tweet_text)
    # remove puncuation
    Tweet_text = punc_re.sub(' ', Tweet_text)
    # remove numeric 'words'
    Tweet_text = num_re.sub(' ', Tweet_text)
    Tweet_text = alpha_num_re.sub(' ', Tweet_text)

    #just considering the model with highest accuracy, can include other models
    all_ml_models = ["MNB","LRM"]
    #display the models using streamlit selectbox
    model_choice=st.selectbox("Choose ML Model",all_ml_models)
```

- When the prediction model is chosen the following code executes and sends the predictive results

```python
def loaddata(Text,mods):
        #read the preprocessed data from pickle file
        df = pd.read_pickle("corpus.pkl")

        STOPWORDS.add("rt")
        STOPWORDS.add("s")
        STOPWORDS.add("u")
        STOPWORDS.add("amp")
        STOPWORDS.add("th")
        STOPWORDS.add("will")
        STOPWORDS.add("t")
        STOPWORDS.add("m")
        STOPWORDS.add("today")


        #split the data into train and test set
        from sklearn.model_selection import train_test_split
        train, test = train_test_split(df, test_size=0.3, train_size=0.7, random_state=14)


        #performing stemming
        lt = LancasterStemmer()
        def token(text):
            txt = nltk.word_tokenize(text.lower())
            return [lt.stem(word) for word in txt]

        #document term matrix using Tfidf vectorizer
        tfv = TfidfVectorizer(tokenizer=token,stop_words=STOPWORDS,analyzer=u'word', min_df=4, norm='l2', use_idf=True)
        X_train_tfv = tfv.fit_transform(train['clean_tweet'])
        X_test_tfv = tfv.transform(test['clean_tweet'])


        X_train_tfv = pd.DataFrame(X_train_tfv.toarray(), columns=tfv.get_feature_names())
        X_test_tfv = pd.DataFrame(X_test_tfv.toarray(), columns=tfv.get_feature_names())

        if(mods=="MNB"):

            st.success("Performing MNB Classification")
            #build the model
            nb = MultinomialNB()
            # Train the model
            nb.fit(X_train_tfv, train['Party_log'])

            #transform the entered text into document term matrix
            vec_text = tfv.transform(Text).toarray()
            #predicting the value for newly entered tweet
            result = nb.predict(vec_text)
            #if result is 1 then democrat else republican
```

```python
        else:
            st.success("Performing Logistic Regression")
            #build the model
            lr = LogisticRegression()
            # Train the model
            lr.fit(X_train_tfv, train['Party_log'])

            #transform the entered text into document term matrix
            vec_text = tfv.transform(Text).toarray()
            #predicting the value for newly entered tweet
            result = lr.predict(vec_text)
            #if result is 1 then democrat else republican

        if result == 1:
             return "demo"
        elif result == 0:
             return "rep"
```

- When NLP activity is chosen the following code executes and display the respective results

```
    #if chosen option is nlp
    if choice == 'NLP':
        st.info("Natural Language Processing")

        #enter the new tweet
        Tweet_text = st.text_area("Enter Here","Type Here")

        #cleaning the tweet entered
        url_re = re.compile('http\S+')
        punc_re = re.compile('[%s]' % re.escape(string.punctuation))
        num_re = re.compile('(\\d+)')
        alpha_num_re = re.compile("[^a-zA-Z]")
        # convert to lowercase
        Tweet_text = Tweet_text.lower()
        # remove hyperlinks
        Tweet_text = url_re.sub(' ', Tweet_text)
        # remove puncuation
        Tweet_text = punc_re.sub(' ', Tweet_text)
        # remove numeric 'words'
        Tweet_text = num_re.sub(' ', Tweet_text)
        Tweet_text = alpha_num_re.sub(' ', Tweet_text)

        STOPWORDS.add("rt")
        STOPWORDS.add("s")
        STOPWORDS.add("u")
        STOPWORDS.add("amp")
        STOPWORDS.add("th")
        STOPWORDS.add("will")
        STOPWORDS.add("t")
        STOPWORDS.add("m")



        list_pos = 0
        cleaned_str = ''
        text = Tweet_text.split()
        for word in text:
            if word not in STOPWORDS:
                if list_pos == 0:
                    cleaned_str = word
                else:
                    cleaned_str = cleaned_str + ' ' + word
                list_pos += 1


        #clean tweet
        Tweet_text = cleaned_str
        #optoin available
        nlp_options=["Tokenization","Lemmatization","POS Tags"]
        #selected option
        nlp_choice=st.selectbox("Choose the NLP option",nlp_options)

```

```python
        if st.button("Start"):

            #displaying the cleaned tweet
            st.info("Original Text::\n{}".format(Tweet_text))

            #nlp coverts text to processed docx object that is understood by spacy
            Sentence= nlp(Tweet_text)
            if nlp_choice=='Tokenization':
                result=[token.text for token in Sentence]
            elif nlp_choice == 'Lemmatization':
                result = ["'Token':{},'Lemma':{}".format(token.text,token.lemma_) for token in Sentence]
            elif nlp_choice == 'POS Tags':
                result = ["'Token':{},'POS':{},'Dependency':{}".format(word.text,word.tag_,word.dep_) for word in Sentence]

            st.json(result)

            #display the results in table form
        if st.button("Tabulize"):
            docx = nlp(Tweet_text)
            c_tokens = [token.text for token in docx ]
            c_lemma = [token.lemma_ for token in docx ]
            c_pos = [token.pos_ for token in docx ]

            #creating dataframe using the results
            new_df = pd.DataFrame(zip(c_tokens,c_lemma,c_pos),columns=['Tokens','Lemma','POS'])
            #display df
            st.dataframe(new_df)

        #display using wordcloud
        if st.checkbox("WordCloud"):
            fig, ax = plt.subplots(figsize=(15,5))
            c_text = Tweet_text
            wordcloud = WordCloud().generate(c_text)
            plt.imshow(wordcloud,interpolation='bilinear')
            plt.axis("off")
            st.pyplot(fig)
            st.set_option('deprecation.showPyplotGlobalUse', False)


if __name__=='__main__':
    main()
```

- Command to Run Streamlit APP

```
Anaconda Prompt (Anaconda3) - streamlit  run Final_Streamlit_App.py

(base) C:\Users\Jagad>cd Data Mining Project

(base) C:\Users\Jagad\Data Mining Project>streamlit run Final_Streamlit_App.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8503
  Network URL: http://192.168.1.7:8503

[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Jagad\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\Jagad\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Jagad\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

5. Evaluation and Results

5.1.1 Determining the Top Hashtags and Words used by Democrats and Republicans

- Plot the Word Cloud image for Democrats hashtags and extract the top hashtags

In [202]:
```
Democrat_Hashtags =''
stopwords = set(STOPWORDS)

# typecaste each val to string
hashtags_list_dem = str(hashtags_dem)

# split the value
tokens = hashtags_list_dem.split()

Democrat_Hashtags += " ".join(tokens)+" "

wordcloud = WordCloud(width = 500, height = 400,
                background_color ='white',
                min_font_size = 5).generate(Democrat_Hashtags)

# plot the WordCloud image
plt.figure(figsize = (5, 4), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```

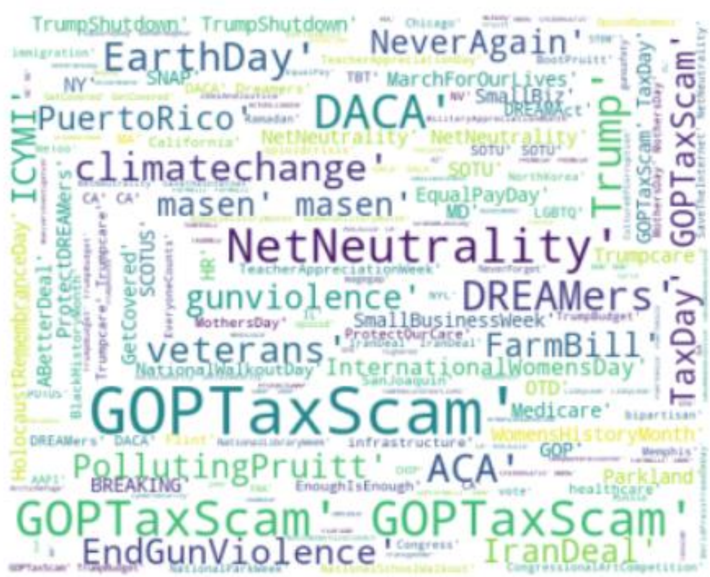


Figure 2 : Word Cloud Image for Democrats Hashtags

From the Word Cloud Image, **#GOPTaxScam, #NetNeutrality, #DACA, #ACA and #DREAMers** are some of the top hashtags Democrats used in their tweets. This seems reasonable as Democrats are fighting for a long time about DACA and NetNeutrality policies.

- Plot the Word Cloud image for Republican hashtags and extract the top hashtags

```
In [203]:  Republican_Hashtags =''

           # typecaste each val to string
           hashtags_list_rep = str(hashtags_list_rep)

           # split the value
           tokens_rep = hashtags_list_rep.split()

           Republican_Hashtags += " ".join(tokens_rep)+" "

           wordcloud = WordCloud(width = 500, height = 400,
                       background_color ='white',
                       min_font_size = 5).generate(Republican_Hashtags)

           # plot the WordCloud image
           plt.figure(figsize = (5, 4), facecolor = None)
           plt.imshow(wordcloud)
           plt.axis("off")
           plt.tight_layout(pad = 0)

           plt.show()
```

Figure 3: Word Cloud Image for Republican Hashtags

From the Word Cloud Image, **#TaxReform, #TaxcutsandJobsAct, #SOTU, #TX and #FarmBill** are some of the top hashtags Republican used in their tweets. This makes sense as refining Tax policies is one of the big items in Republican's agenda.

- Plot the Pie chart for the Top 5 words used by Democrats

```
In [42]: #Sort the count of each words in decending order to get the top 5
         Dem_freqw=Series(fdist_democrat).sort_values(ascending=False).head(5)
         Rep_freqw=Series(fdist_republican).sort_values(ascending=False).head(5)
```

```
In [43]: #labels
         type_labels_Dem = Dem_freqw.sort_values().index
         type_counts_Dem = Dem_freqw.sort_values()
```

```
In [44]: plt.figure(1, figsize=(20,10))
         the_grid = GridSpec(2,2)
         cmap = plt.get_cmap('Spectral')
         colors = [cmap(i) for i in np.linspace(0, 1, 8)]
         plt.subplot(the_grid[0, 1], aspect=1, title='Democrats Top Words')
         type_show_ids = plt.pie(type_counts_Dem, labels=type_labels_Dem, autopct='%1.1f%%', shadow=True, colors=colors)
         plt.show()
```

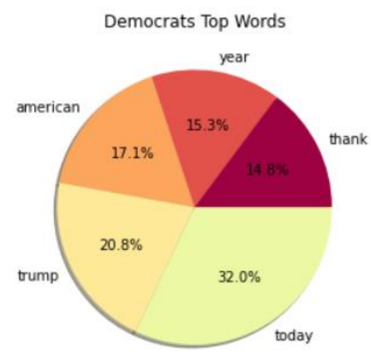


Figure 4: Pie chart for the Top 5 words used by Democrats

From the Pie Chart, the words “Today”, “Trump”, “American”, “Year” and “Thank” are figured out as the most frequently used words by the Democrats in their tweets.

Surprisingly, Democrats used the word “Trump” in a greater number of times than Republicans.

- Plot the Pie chart for the Top 5 words used by Republicans

```
In [45]: type_labels_Rep = Rep_freqw.sort_values().index
         type_counts_Rep = Rep_freqw.sort_values()
         plt.figure(1, figsize=(20,10))
         the_grid = GridSpec(2,2)
         cmap = plt.get_cmap('Spectral')
         colors = [cmap(i) for i in np.linspace(0, 1, 8)]
         plt.subplot(the_grid[0, 1], aspect=1, title='Republican Top Words')
         type_show_ids = plt.pie(type_counts_Rep, labels=type_labels_Rep, autopct='%1.1f%%', shadow=True, colors=colors)
         plt.show()
```

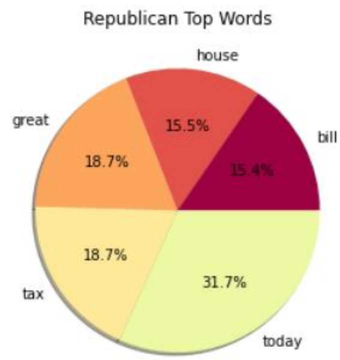


Figure 5: Pie chart for the Top 5 words used by Republicans

From the Pie Chart, the words “Today”, “Tax”, “Great”, “House” and “Bill” are figured out as the most frequently used words by the Democrats in their tweets.

Just like the Hashtags, the Republicans used the word “Tax” in most of their tweets. Also, the word “great” is repeated many times as it’s also mentioned in their campaign slogan “Make America Great Again”.

5.1.2 What Sentiments do the tweets have? Are there more negative tweets posted by a particular party?

- Sentiment analysis for Democrats and Republican Tweets

```
In [51]: #Sentiment analysis-Democrat Tweets
         democratblob=TextBlob(str(democrat_tweets))
         democratblob.sentiment

Out[51]: Sentiment(polarity=0.34027777777777785, subjectivity=0.4555555555555555)
```

```
In [52]: #Sentiment analysis-Republican Tweets
         republicanblob=TextBlob(str(republican_tweets))
         republicanblob.sentiment

Out[52]: Sentiment(polarity=0.19837398561739605, subjectivity=0.4590746992419168)
```

From the results,

- The Democrats handlers had posted more positive tweets (Polarity= 0.34) than republican members (Polarity=0.19)
- The Subjectivity score is pretty much the same for both the parties (Subjectivity =0.45)

5.1.3 Classification model to determine whether newly posted tweet belongs to Democratic or Republican class

<u>Multinominal Naïve Bayes Model</u>

- Confusion matrix

```
In [67]:  cm = confusion_matrix(test['Party_log'], y_pred_tfv_nb)
          sns.heatmap(cm, xticklabels=['predicted_pos', 'predicted_neg'], yticklabels=['actual_pos', 'actual_neg'], annot=True, fmt='d'
          true_neg, false_pos = cm[0]
          false_neg, true_pos = cm[1]
          accuracy = round((true_pos + true_neg) / (true_pos + true_neg + false_pos + false_neg),3)
          precision = round((true_pos) / (true_pos + false_pos),3)
          recall = round((true_pos) / (true_pos + false_neg),3)
          f1 = round(2 * (precision * recall) / (precision + recall),3)
          print('Accuracy: {}'.format(accuracy))
          print('Precision: {}'.format(precision))
          print('Recall: {}'.format(recall))
          print('F1 Score: {}'.format(f1))
```

```
Accuracy: 0.785
Precision: 0.799
Recall: 0.744
F1 Score: 0.771
```

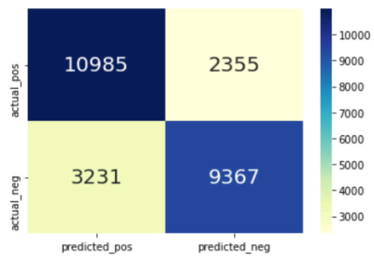


Figure 6: Confusion matrix for Naïve Bayes Model

- Using classification report, calculating the accuracy, precision and recall for each class in the label

```
In [66]:  print(classification_report(y_pred_tfv_nb,test['Party_log']))
```

	precision	recall	f1-score	support
0	0.82	0.77	0.80	14216
1	0.74	0.80	0.77	11722
accuracy			0.78	25938
macro avg	0.78	0.79	0.78	25938
weighted avg	0.79	0.78	0.79	25938

As the data are quite well balanced, there is lesser chances of overfitting problem

Logistic Regression

- Confusion matrix

In [74]:
```
cm = confusion_matrix(test['Party_log'], y_pred_tfv_lr)
sns.heatmap(cm, xticklabels=['predicted_pos', 'predicted_neg'], yticklabels=['actual_pos', 'actual_neg'], annot=True, fmt='d'
true_neg, false_pos = cm[0]
false_neg, true_pos = cm[1]
accuracy = round((true_pos + true_neg) / (true_pos + true_neg + false_pos + false_neg),3)
precision = round((true_pos) / (true_pos + false_pos),3)
recall = round((true_pos) / (true_pos + false_neg),3)
f1 = round(2 * (precision * recall) / (precision + recall),3)
print('Accuracy: {}'.format(accuracy))
print('Precision: {}'.format(precision))
print('Recall: {}'.format(recall))
print('F1 Score: {}'.format(f1))
```

```
Accuracy: 0.781
Precision: 0.781
Recall: 0.764
F1 Score: 0.772
```

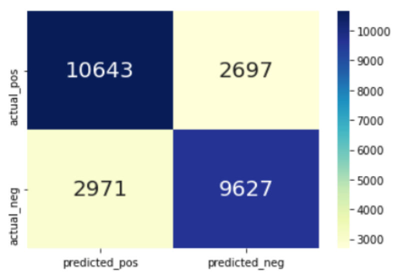


Figure 7: Confusion matrix for Logistic Regression Model

- Using classification report, calculating the accuracy, precision and recall for each class

In [75]:
```
print(classification_report(y_pred_tfv_lr,test['Party_log']))
```

```
              precision    recall  f1-score   support

           0       0.80      0.78      0.79     13614
           1       0.76      0.78      0.77     12324

    accuracy                           0.78     25938
   macro avg       0.78      0.78      0.78     25938
weighted avg       0.78      0.78      0.78     25938
```

AdaBoost Classification

- Confusion matrix

In [20]:
```
cm = confusion_matrix(test['Party_log'], y_pred_tfv_ada)
sns.heatmap(cm, xticklabels=['predicted_pos', 'predicted_neg'], yticklabels=['actual_pos', 'actual_neg'], annot=True, fmt='d'
true_neg, false_pos = cm[0]
false_neg, true_pos = cm[1]
accuracy = round((true_pos + true_neg) / (true_pos + true_neg + false_pos + false_neg),3)
precision = round((true_pos) / (true_pos + false_pos),3)
recall = round((true_pos) / (true_pos + false_neg),3)
f1 = round(2 * (precision * recall) / (precision + recall),3)
print('Accuracy: {}'.format(accuracy))
print('Precision: {}'.format(precision))
print('Recall: {}'.format(recall))
print('F1 Score: {}'.format(f1))
```

```
Accuracy: 0.637
Precision: 0.588
Recall: 0.845
F1 Score: 0.693
```

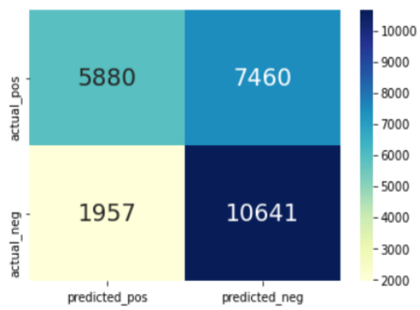


Figure 8: Confusion matrix for AdaBoost Classification Model

- Using classification report, calculating the accuracy, precision and recall for each class

In [21]:
```
print(classification_report(y_pred_tfv_ada,test['Party_log']))
```

```
              precision    recall  f1-score   support

           0       0.44      0.75      0.56      7837
           1       0.84      0.59      0.69     18101

    accuracy                           0.64     25938
   macro avg       0.64      0.67      0.62     25938
weighted avg       0.72      0.64      0.65     25938
```

Random Forest Classification

- Confusion matrix

```
In [17]:  cm = confusion_matrix(test['Party_log'], y_pred_tfv_rfc)
          sns.heatmap(cm, xticklabels=['predicted_pos', 'predicted_neg'], yticklabels=['actual_pos', 'actual_neg'], annot=True, fmt='d'
          true_neg, false_pos = cm[0]
          false_neg, true_pos = cm[1]
          accuracy = round((true_pos + true_neg) / (true_pos + true_neg + false_pos + false_neg),3)
          precision = round((true_pos) / (true_pos + false_pos),3)
          recall = round((true_pos) / (true_pos + false_neg),3)
          f1 = round(2 * (precision * recall) / (precision + recall),3)
          print('Accuracy: {}'.format(accuracy))
          print('Precision: {}'.format(precision))
          print('Recall: {}'.format(recall))
          print('F1 Score: {}'.format(f1))
```

```
Accuracy: 0.742
Precision: 0.756
Recall: 0.693
F1 Score: 0.723
```

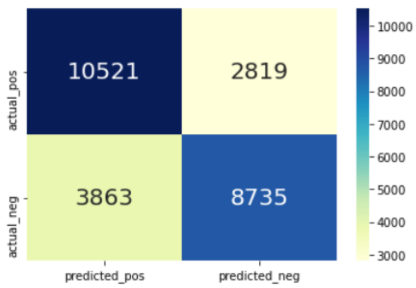


Figure 9: Confusion matrix for Random Forest Classification Model

- Using classification report, calculating the accuracy, precision and recall for each class

```
In [18]:  print(classification_report(y_pred_tfv_rfc,test['Party_log']))
```

```
              precision    recall  f1-score   support

           0       0.79      0.73      0.76     14384
           1       0.69      0.76      0.72     11554

    accuracy                           0.74     25938
   macro avg       0.74      0.74      0.74     25938
weighted avg       0.75      0.74      0.74     25938
```

Decision Tree Classification

- Confusion matrix

In [22]:
```
#Decision Tree Classification
dtc = DecisionTreeClassifier(random_state = 42)
dtc.fit(X_train_tfv, train['Party_log'])
y_pred__tfv_dtc = dtc.predict(X_test_tfv)
```

In [24]:
```
cm = confusion_matrix(test['Party_log'], y_pred__tfv_dtc)
sns.heatmap(cm, xticklabels=['predicted_pos', 'predicted_neg'], yticklabels=['actual_pos', 'actual_neg'], annot=True, fmt='d'
true_neg, false_pos = cm[0]
false_neg, true_pos = cm[1]
accuracy = round((true_pos + true_neg) / (true_pos + true_neg + false_pos + false_neg),3)
precision = round((true_pos) / (true_pos + false_pos),3)
recall = round((true_pos) / (true_pos + false_neg),3)
f1 = round(2 * (precision * recall) / (precision + recall),3)
print('Accuracy: {}'.format(accuracy))
print('Precision: {}'.format(precision))
print('Recall: {}'.format(recall))
print('F1 Score: {}'.format(f1))
```

```
Accuracy: 0.678
Precision: 0.673
Recall: 0.656
F1 Score: 0.664
```

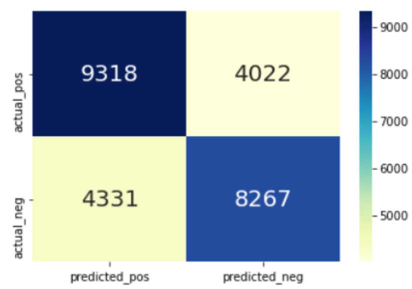


Figure 10: Confusion matrix for Decision Tree Classification Model

- Using classification report, calculating the accuracy, precision and recall for each class

In [25]:
```
print(classification_report(y_pred__tfv_dtc,test['Party_log']))
```

```
              precision    recall  f1-score   support

           0       0.70      0.68      0.69     13649
           1       0.66      0.67      0.66     12289

    accuracy                           0.68     25938
   macro avg       0.68      0.68      0.68     25938
weighted avg       0.68      0.68      0.68     25938
```

5.1.4 Classification of newly entered tweet by the user with the help of Streamlit application

- **Streamlit APP - User Interface**

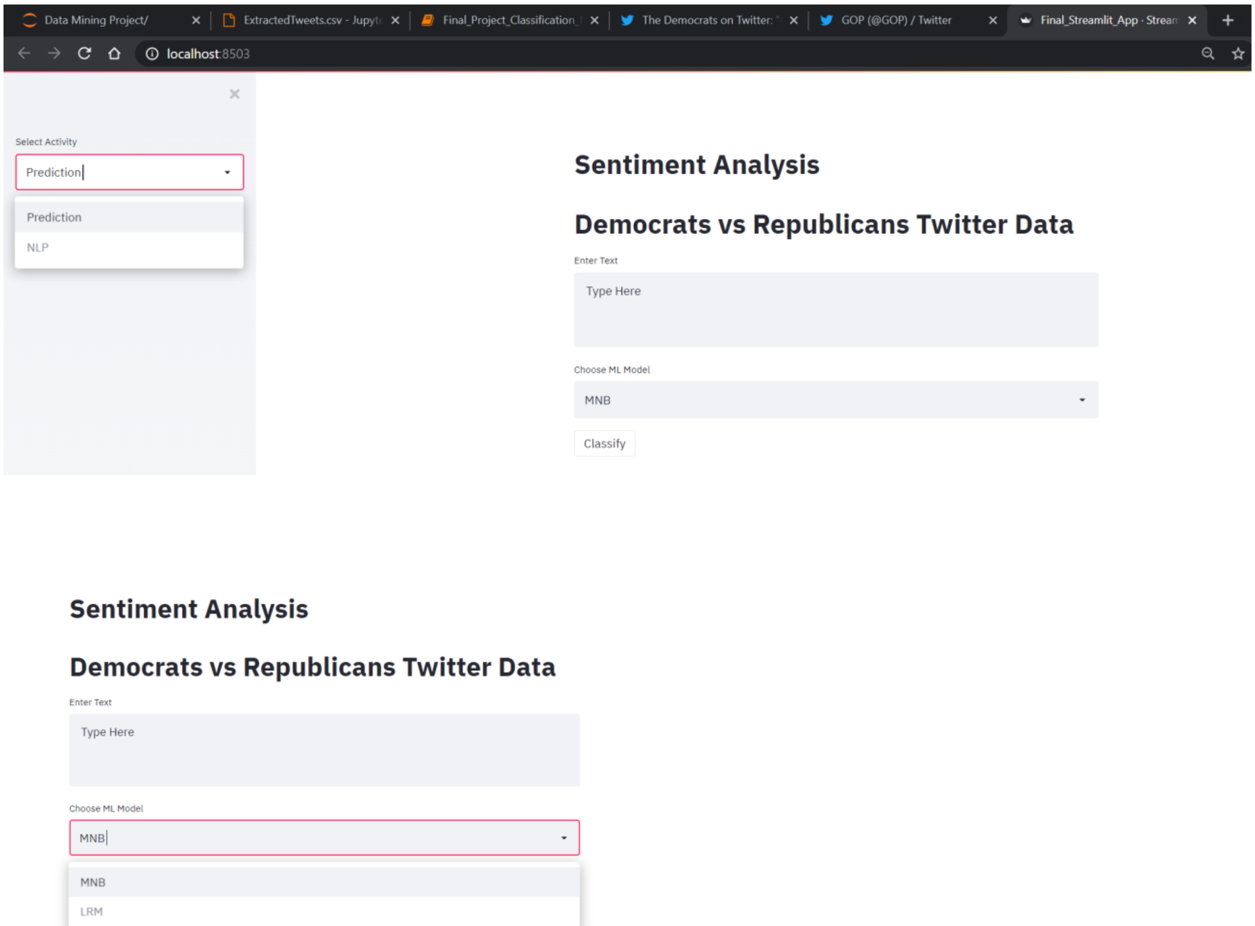


Figure 11: Streamlit APP - User Interface

- **Streamlit APP – Classification Model Results**

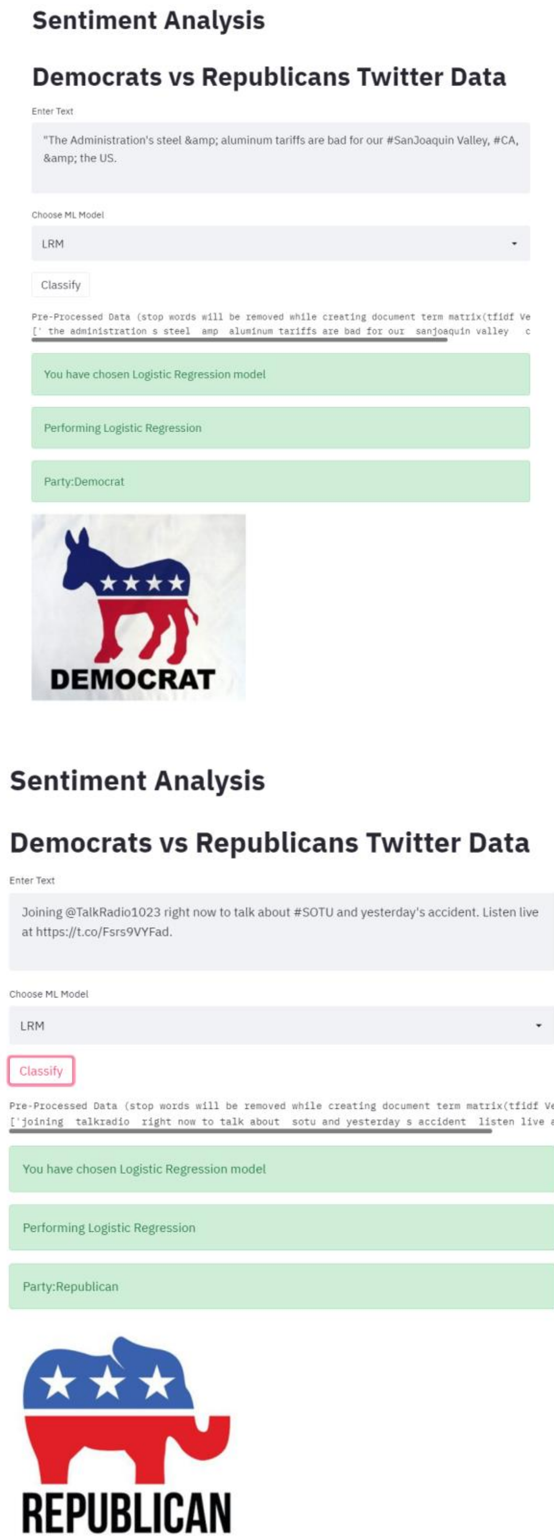


Figure 12: Streamlit APP – Classification model results

- **Streamlit APP – Recent Tweet by The Democrats**

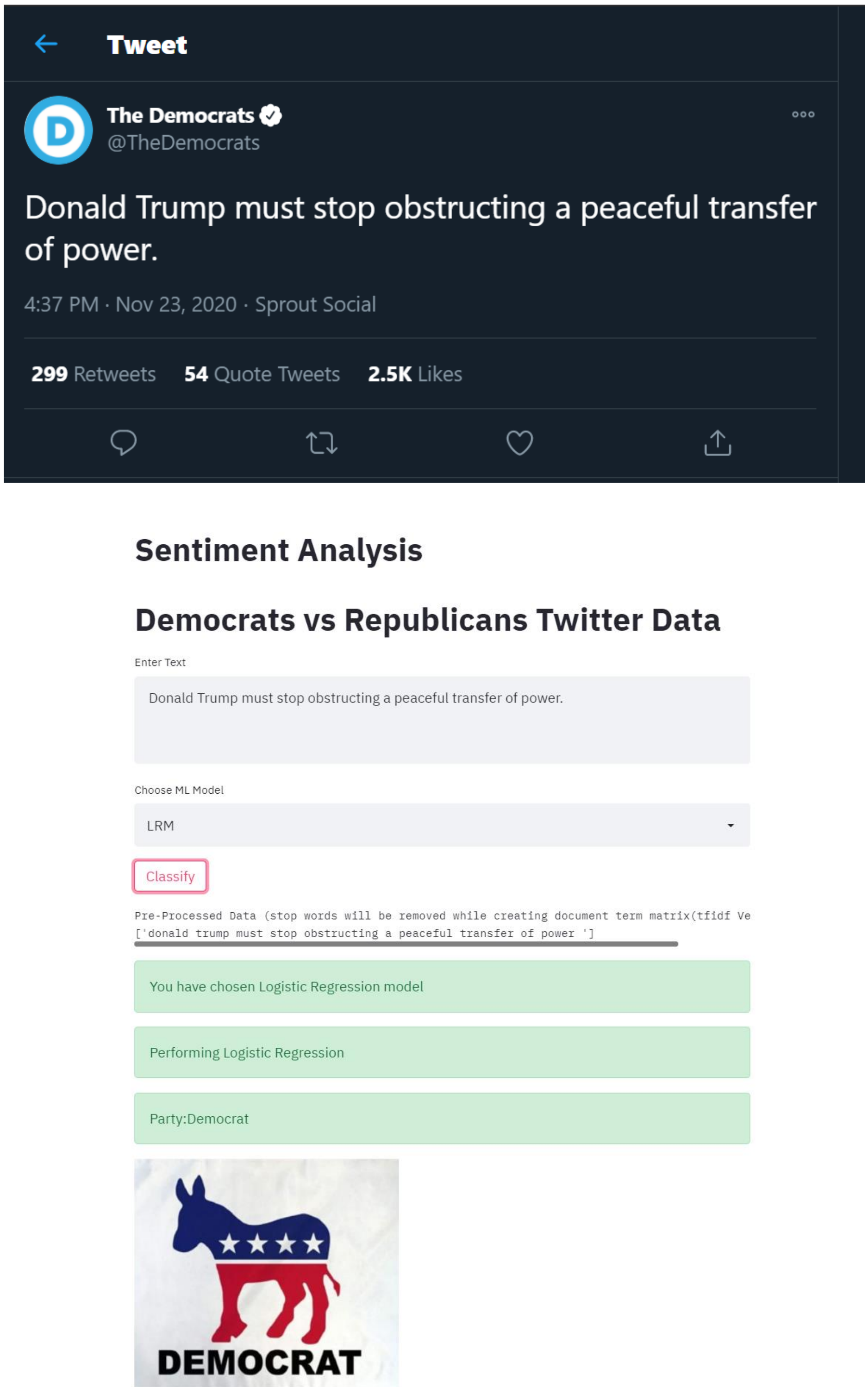


Figure 13: Streamlit APP - Recent Tweet by The Democrats

- **Streamlit APP – Recent Tweet by Republican (@GOP)**

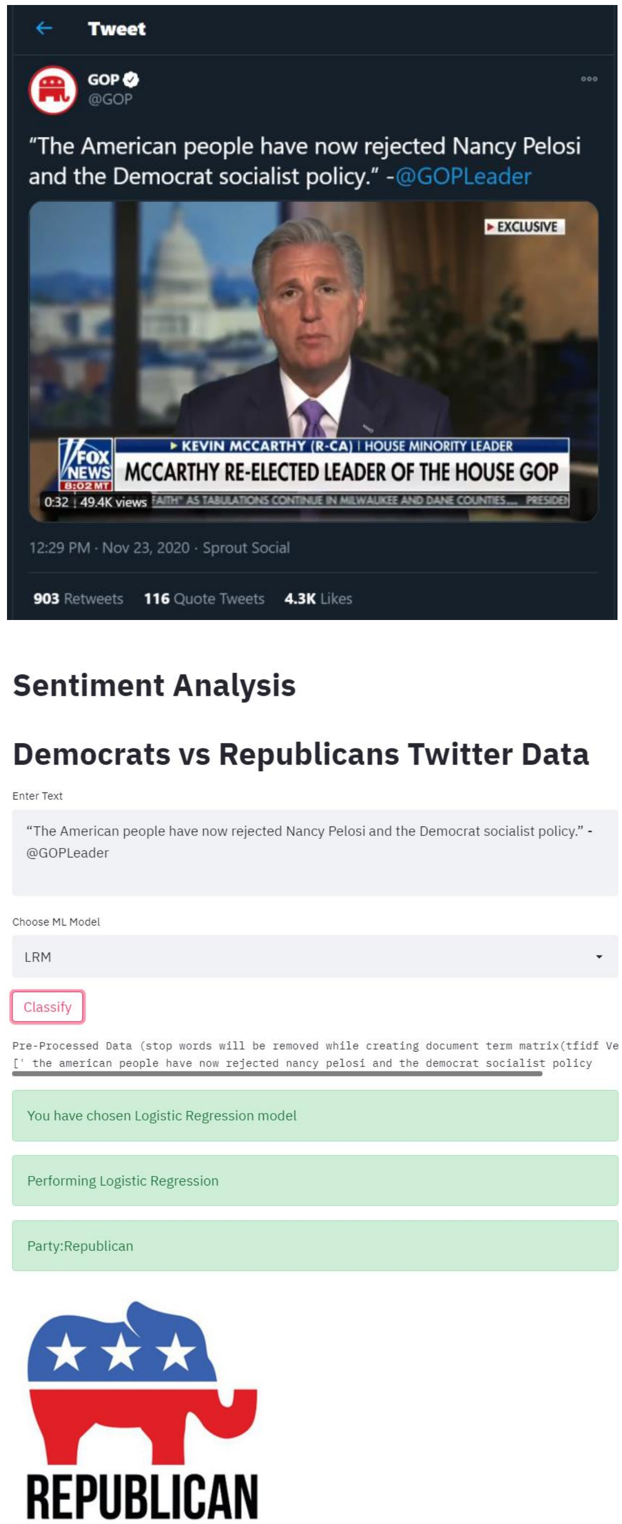
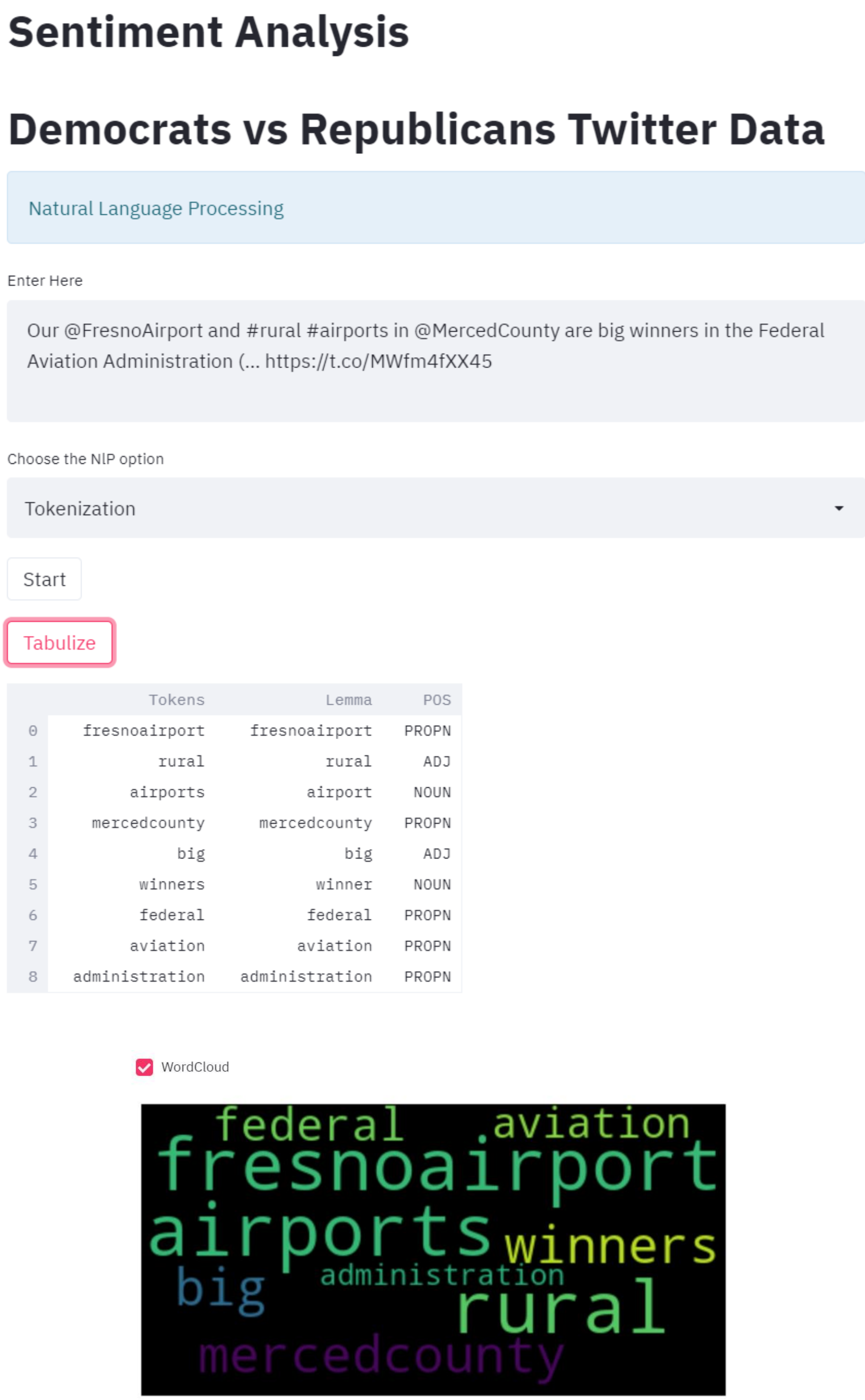


Figure 14: Streamlit APP - Recent Tweet by The Republicans

- **Streamlit APP – NLP**



	Tokens	Lemma	POS
0	fresnoairport	fresnoairport	PROPN
1	rural	rural	ADJ
2	airports	airport	NOUN
3	mercedcounty	mercedcounty	PROPN
4	big	big	ADJ
5	winners	winner	NOUN
6	federal	federal	PROPN
7	aviation	aviation	PROPN
8	administration	administration	PROPN

Figure 15: Streamlit APP - NLP

6. Conclusions and Future Work

6.1. Conclusion

"Logistic Regression" is considered as the best model by observing the F1 score of each class and overall accuracy

```
In [75]:  print(classification_report(y_pred_tfv_lr,test['Party_log']))

              precision    recall  f1-score   support

           0       0.80      0.78      0.79     13614
           1       0.76      0.78      0.77     12324

    accuracy                           0.78     25938
   macro avg       0.78      0.78      0.78     25938
weighted avg       0.78      0.78      0.78     25938
```

6.2. Limitations

The limitations of my project are:

- Tweets are collected between January 22, 2018 and January 3, 2019
- The elected congressmen and twitter handle might be changed in recent times
- Also, the recent top issues like Covid-19 are not covered in prior tweets

6.3. Potential Improvements or Future Work

- Perform Web scraping using Python Beautiful Soup to extract recent tweets to update the Train set.
- Improving the accuracy of Decision Tree Model by Post-Pruning Technique.

Current Progress:

```
In [36]:
#Comparing accuracy for gini vs entropy
dtree = DecisionTreeClassifier(criterion='gini')
dtree.fit(X_train_cv, train['Party_log'])
pred = dtree.predict(X_test_cv)
print('Criterion=gini', accuracy_score(pred, test['Party_log']))


dtree = DecisionTreeClassifier(criterion='entropy')
dtree.fit(X_train_cv, train['Party_log'])
pred = dtree.predict(X_test_cv)
print('Criterion=entropy', accuracy_score(pred, test['Party_log']))

Criterion=gini 0.6915336571825121
Criterion=entropy 0.6962371809700054
```

Visualizing Decision Tree:

```
In [20]: #visualizing decision tree
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
import pydotplus
dot_data = StringIO()
export_graphviz(dtc, out_file=dot_data)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('/Users/arpitha/Documents/SEMESTER 3/Data Mining/Final Project/project/tree.png')
Image(graph.create_png())
```

```
dot: graph is too large for cairo-renderer bitmaps. Scaling by 0.137453 to fit
```

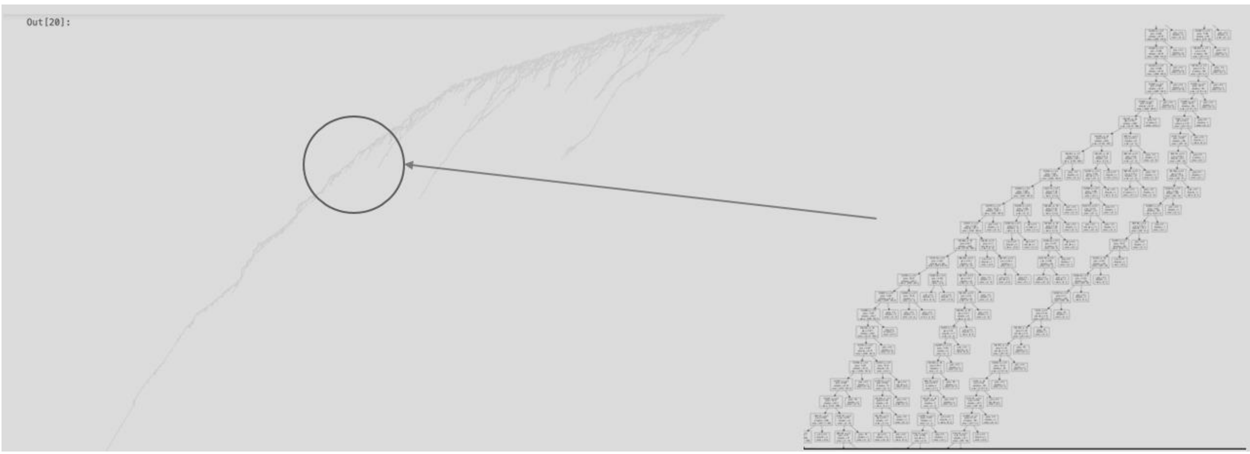


Figure 16: Decision Tree Visualization

Optimizing the maximum tree depth using post-pruning technique

```
In [*]: #optimizing decision tree
max_depth = []
acc_gini = []
acc_entropy = []
for i in range(10,40):
    dtree = DecisionTreeClassifier(criterion='gini', max_depth=i)
    dtree.fit(X_train_tfv,Y_train)
    pred = dtree.predict(X_test_tfv)
    acc_gini.append(accuracy_score(pred, Y_test)

    dtree = DecisionTreeClassifier(criterion='entropy', max_depth=i)
    dtree.fit(X_train_cv, train['Party_log'])
    pred = dtree.predict(X_test_cv)
    acc_entropy.append(accuracy_score(pred, test['Party_log']))

    max_depth.append(i)
```

```
In [ ]: d= pd.DataFrame({'acc_gini':pd.Series(acc_gini),
 'acc_entorpy':pd.Series(acc_gini),
 'max_depth':pd.Series(max_depth)})
```

```
In [ ]: # visualizing changes in parameters
plt.scatter('max_depth', 'acc_gini', data=d, label= 'gini')
plt.plot('max_depth', 'acc_entropy', data=d, label='entropy')
plt.xlabel('max_depth')
plt.ylabel('accuracy')
plt.legend()
```

7. References

- Kyle Pastor - Democrat Vs. Republican Tweets
  https://www.kaggle.com/kapastor/democratvsrepublicantweets?select=ExtractedTweets.csv
- Democrats Twitter Account- @TheDemocrats
  https://twitter.com/TheDemocrats?ref_src=twsrc%5Egoogle%7Ctwcamp%5Eserp%7Ctwgr%5Eauthor
- Republican Twitter Account - @GOP
  https://twitter.com/GOP?ref_src=twsrc%5Egoogle%7Ctwcamp%5Eserp%7Ctwgr%5Eauthor