Data Science Project

Congressional Tweets Dataset -- 2008-2017 data
Client: Lobbyists4America

Alice Stephens
July-September 2020

Project Summary: Lobbyists4America is a company that seeks to provide insights to their customers (who aim to affect legislation within the US). They want to analyse the 2008-2017 congressional tweets in order to understand key topics, members, and relationships within Congress. These insights will help them focus and strengthen their lobbying efforts.

Table of Contents

1. Project Proposal

I chose this particular dataset and client as I have always had an interest in politics, and it is one of those areas that has recently benefited from a huge increase in available data with the proliferation of social media. This means it is now possible to gain insights into political sentiment and opinion which would previously have been inaccessible. It is also a perfect example of being able to use data for a higher purpose, providing insights that will shape our world.

1.1 Questions to answer:

- What topics are most frequently discussed?
    - This will help inform what issues are most important to members of congress, what subjects form most of the focus
- Who are the most influential figures?
    - Are there certain individuals who command a larger audience?
    - Are there certain individuals who tend to get retweeted more?
    - Do these individuals tend to have certain traits (such as having a more positive or negative tone)?
- Are there certain tweets which have resonated more?
    - What do these tend to be about?
    - Do these tweets tend to have a particular tone?

1.2 Hypotheses:

1.2.1 What topics are most frequently discussed?

As the presidential election occurred in 2016, I expect to find a significant number of tweets referencing this, including references to Obama and Trump. There may also be references to major policy initiatives, such as Obamacare, or Trump’s wall.

1.2.2 Who are the most influential figures?

I believe that Donald Trump will have the largest audience, as his liking for Twitter is well known. I also think that the more recent eminent politicians will have more followers as Twitter has gained in popularity more during the latter half of my dataset period. It may also be the case that the more followers someone has, the more retweets they are likely to have.

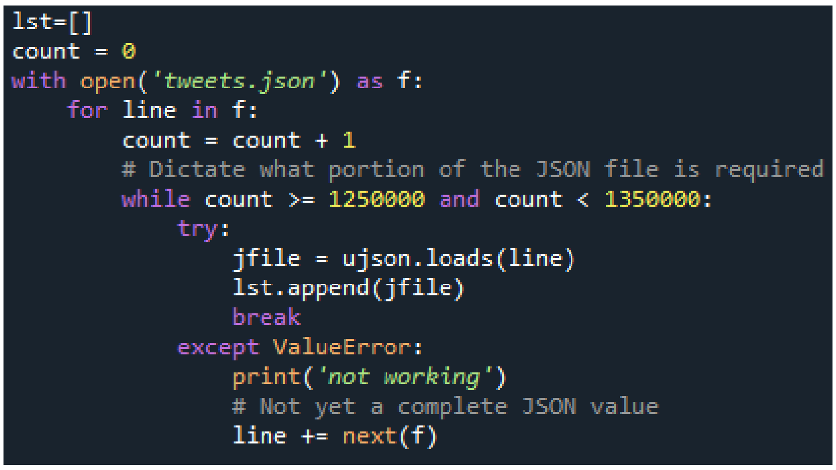
1.2.3 What are the most popular tweets?

It is my opinion that the most popular tweets (popularity being measured by the number of retweets a tweet has) will have a more positive tone – based on the idea that perhaps a more inspirational text might move someone to retweet it.

2. Data Cleaning

Data was provided as a tar.gz file, containing two json files; one containing tweet information and one containing user information. The file was unzipped using TarTool.exe due to the child file sizes.

Python in Spyder was used for initial cleaning/analysis of data. Due to the size of the tweets file, the whole thing could not be processed in one go by the software, so the ujson library in Python was used, and looped through a limited number of entries before stopping (see figure 1). When the time comes for deeper analysis, relevant information can be brought in from the database so all available information can be used (where appropriate).

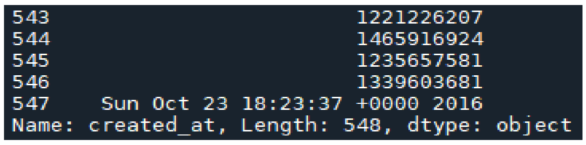


```
lst=[]
count = 0
with open('tweets.json') as f:
    for line in f:
        count = count + 1
        # Dictate what portion of the JSON file is required
        while count >= 1250000 and count < 1350000:
            try:
                jfile = ujson.loads(line)
                lst.append(jfile)
                break
            except ValueError:
                print('not working')
                # Not yet a complete JSON value
                line += next(f)
```

figure 1 – code to import data from json file

2.1 Dates

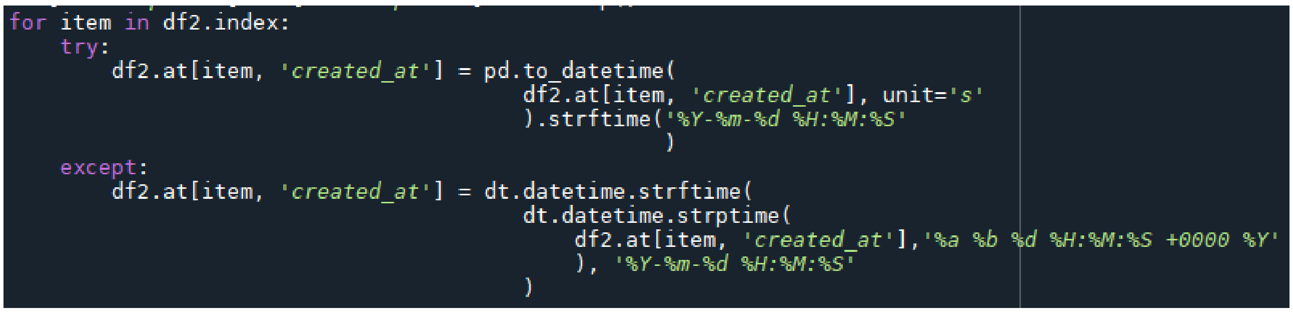
Dates provided were a mixture of unix timestamp and date strings (see figure 2)



```
543                          1221226207
544                          1465916924
545                          1235657581
546                          1339603681
547    Sun Oct 23 18:23:37 +0000 2016
Name: created_at, Length: 548, dtype: object
```

Figure 2 – sample of dates provided

I used a try/except function in order to apply to appropriate method based on date format (see figure 3)

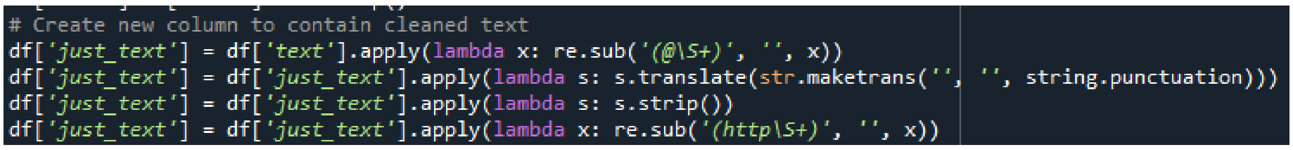


```
for item in df2.index:
    try:
        df2.at[item, 'created_at'] = pd.to_datetime(
                                        df2.at[item, 'created_at'], unit='s'
                                        ).strftime('%Y-%m-%d %H:%M:%S'
                                                   )
    except:
        df2.at[item, 'created_at'] = dt.datetime.strftime(
                                        dt.datetime.strptime(
                                            df2.at[item, 'created_at'],'%a %b %d %H:%M:%S +0000 %Y'
                                            ), '%Y-%m-%d %H:%M:%S'
                                        )
```

figure 3 – converting date information to a usable string format

2.2 Text

A "just_text" column was created within the dataframe in order to more easily pull out textual information during analysis. For this all mentions were deleted (@something), punctuation was deleted, trailing whitespace was stripped and hyperlinks were deleted (see figure 4). The process was repeated for the "description" column for user details ("just_desc" column created).

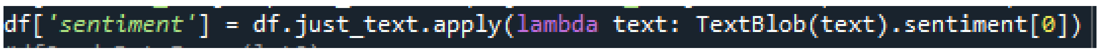


```python
# Create new column to contain cleaned text
df['just_text'] = df['text'].apply(lambda x: re.sub('(@\S+)', '', x))
df['just_text'] = df['just_text'].apply(lambda s: s.translate(str.maketrans('', '', string.punctuation)))
df['just_text'] = df['just_text'].apply(lambda s: s.strip())
df['just_text'] = df['just_text'].apply(lambda x: re.sub('(http\S+)', '', x))
```

figure 4 – text cleaning

2.3 Sentiment

A "sentiment" column was also created, based on the "just_text" column in order to perform sentiment analysis later on, looking at correlation with other features (see figure 5).



```python
df['sentiment'] = df.just_text.apply(lambda text: TextBlob(text).sentiment[0])
```

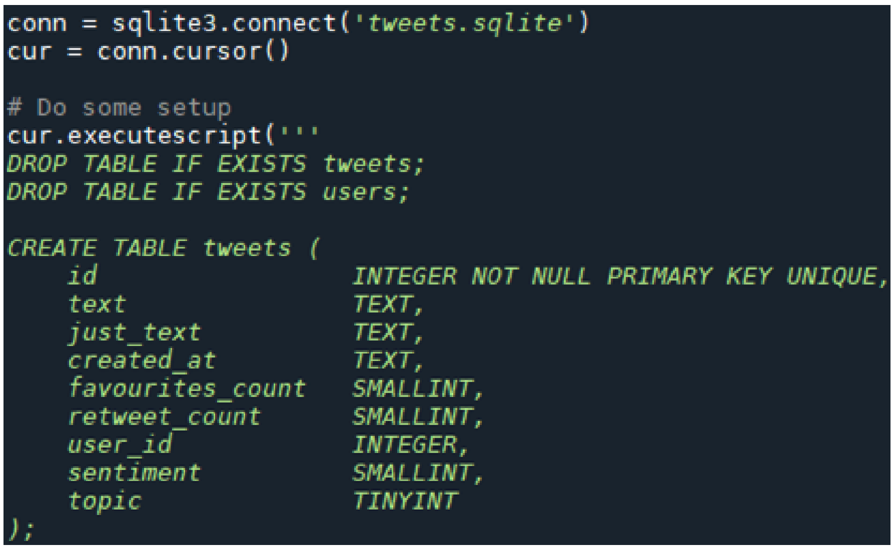
figure 5 – calculating tweet sentiment

2.4 Other

In addition to the above, tweets were filtered out which did not have language set to "English", and only relevant column information was added to the dataframe. A search was performed for NaN/ missing values, but there were none.

3. Database

Due to the large amount of data, and the cleaning involved, it became necessary to create a database from which only the relevant information could be pulled for analysis. I used DB Browser for SQLite as the database, and data was uploaded through Spyder (see figures 6 and 7).

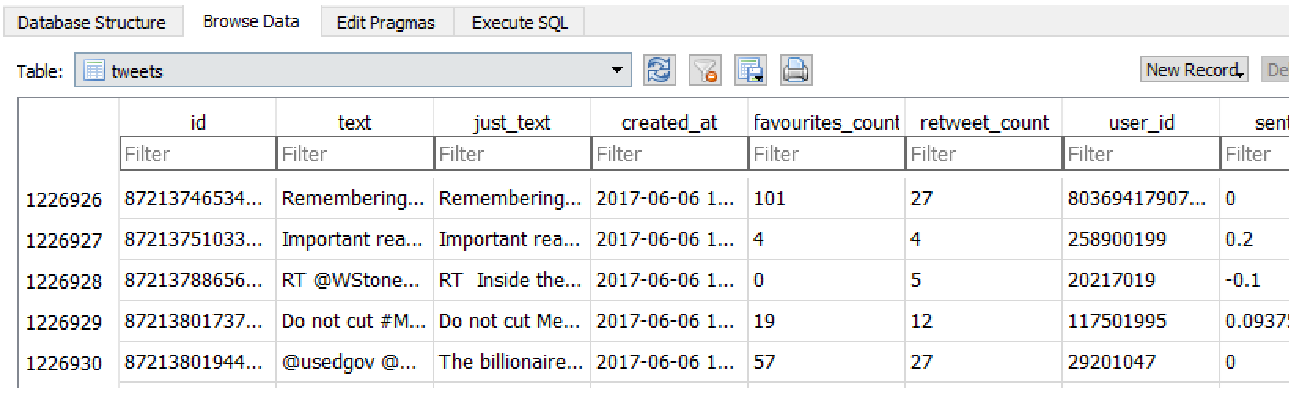


```
conn = sqlite3.connect('tweets.sqlite')
cur = conn.cursor()

# Do some setup
cur.executescript('''
DROP TABLE IF EXISTS tweets;
DROP TABLE IF EXISTS users;

CREATE TABLE tweets (
    id                  INTEGER NOT NULL PRIMARY KEY UNIQUE,
    text                TEXT,
    just_text           TEXT,
    created_at          TEXT,
    favourites_count    SMALLINT,
    retweet_count       SMALLINT,
    user_id             INTEGER,
    sentiment           SMALLINT,
    topic               TINYINT
);
```

figure 6 – creating a database



	id	text	just_text	created_at	favourites_count	retweet_count	user_id	sent
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1226926	87213746534...	Remembering...	Remembering...	2017-06-06 1...	101	27	80369417907...	0
1226927	87213751033...	Important rea...	Important rea...	2017-06-06 1...	4	4	258900199	0.2
1226928	87213788656...	RT @WStone...	RT Inside the...	2017-06-06 1...	0	5	20217019	-0.1
1226929	87213801737...	Do not cut #M...	Do not cut Me...	2017-06-06 1...	19	12	117501995	0.0937
1226930	87213801944...	@usedgov @...	The billionaire...	2017-06-06 1...	57	27	29201047	0

figure 7 – SQLite database – tweets table

For resulting ER diagram see figure 8. On starting out, I had initially planned that more tables would be created, linked by foreign keys, but on beginning to input the data and start to use it, it became clear that this more simple layout was the most optimal.

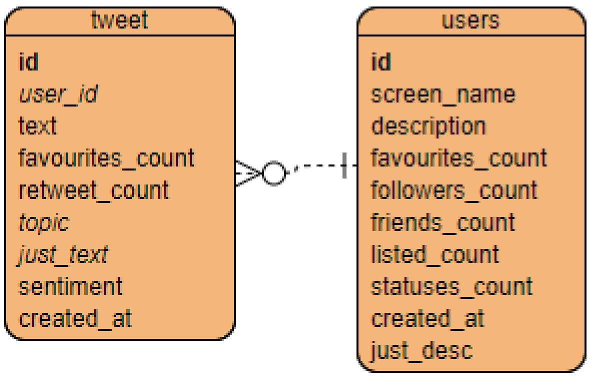
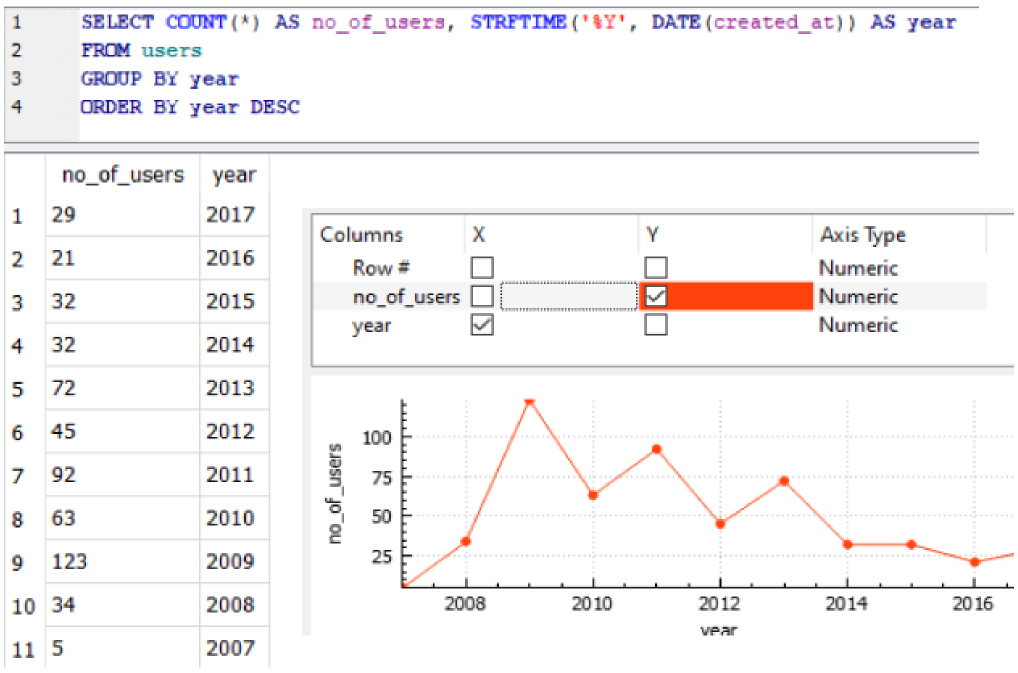


figure 8 – ER diagram

4. Initial Analysis

Now that the data is loaded to the database, it can be seen there are 548 users, and 1,226,949 tweets. Another benefit of the database is that it is possible to quickly look at simple statistics. This can be demonstrated in figure 9 (number of users joining Twitter per year) and figure 10 (number of tweets tweeted per year – this increases every year as politician's use of social media increases, and the reason a dip is seen in 2017 is that the data cuts off in August 2017).

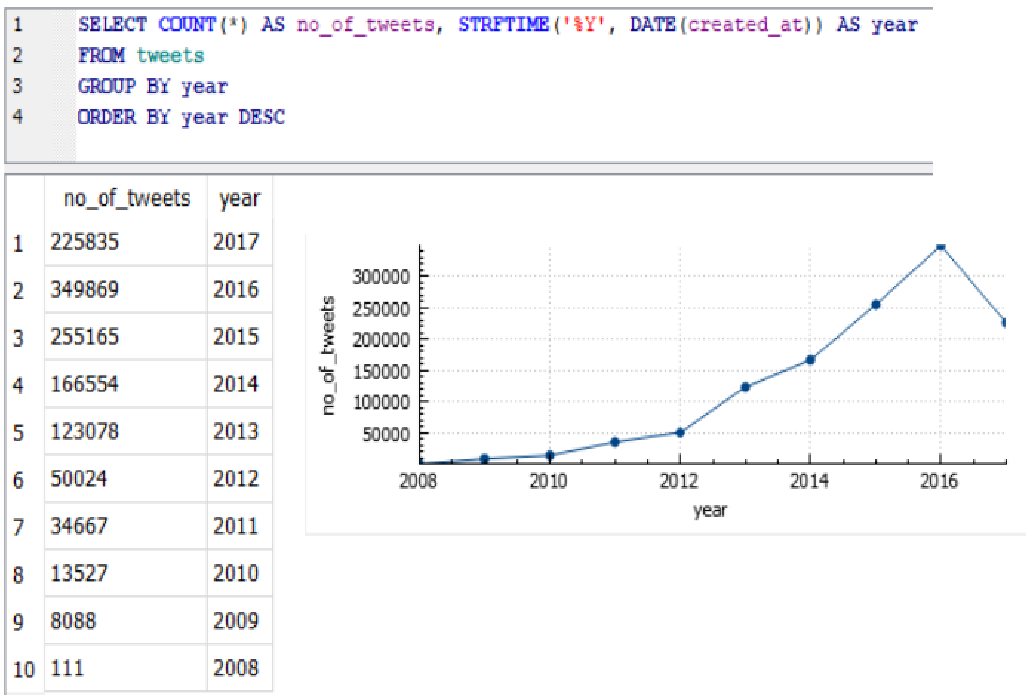


```
SELECT COUNT(*) AS no_of_users, STRFTIME('%Y', DATE(created_at)) AS year
FROM users
GROUP BY year
ORDER BY year DESC
```

	no_of_users	year
1	29	2017
2	21	2016
3	32	2015
4	32	2014
5	72	2013
6	45	2012
7	92	2011
8	63	2010
9	123	2009
10	34	2008
11	5	2007

Columns	X	Y	Axis Type
Row #	☐	☐	Numeric
no_of_users	☐	✓	Numeric
year	✓	☐	Numeric

figure 9 – number of congressional members joining Twitter per year



```
SELECT COUNT(*) AS no_of_tweets, STRFTIME('%Y', DATE(created_at)) AS year
FROM tweets
GROUP BY year
ORDER BY year DESC
```

	no_of_tweets	year
1	225835	2017
2	349869	2016
3	255165	2015
4	166554	2014
5	123078	2013
6	50024	2012
7	34667	2011
8	13527	2010
9	8088	2009
10	111	2008

figure 10 – number of tweets tweeted per year

4.1 Most followed Twitter users

It can see here that, as predicted in my hypothesis, Trump does have the most followers (see figure 11). Following close behind is Bernie Sanders, another prominent politician this time in the Democratic party.

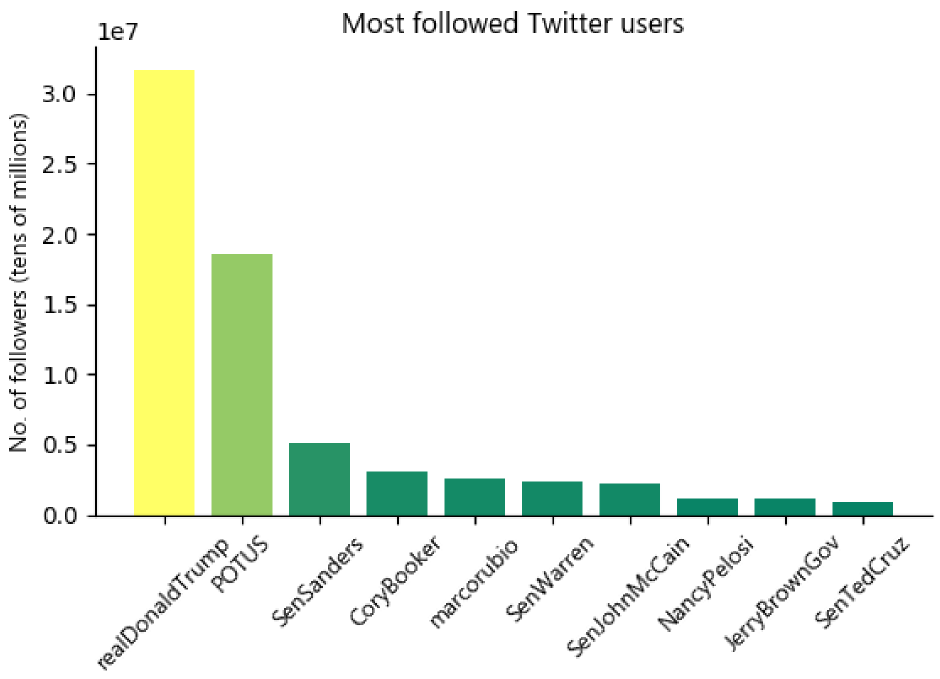


figure 11 – most followed Twitter users

4.2 Most retweeted Twitter users

As illustrated in figure 12, it can be seen that many of the figures with lots of followers also get the most retweets. It would be worth investigating the strength of this correlation later.

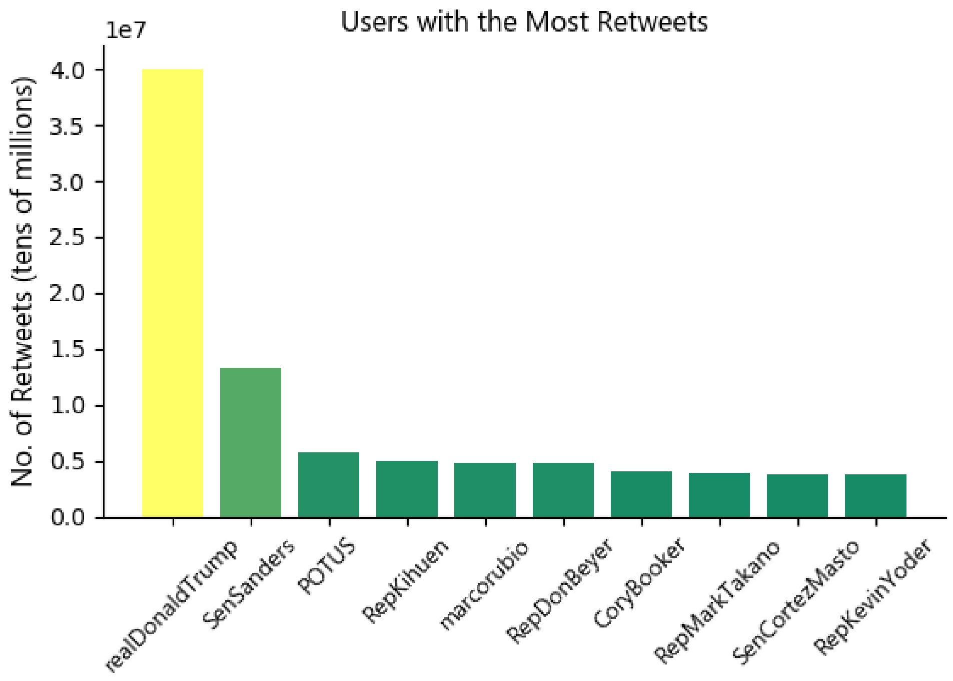


figure 12 – users with the most retweets

5. Correlation Measures

5.1 Number of followers versus number of retweets

From the graphs in section 4.1 and 4.2 it could be seen that the same individuals seemed to have lots of followers who also had lots of retweets – which supports my earlier hypothesis. To prove this, different measures of correlation were considered.

Initially it is difficult to really see a true correlation when I plot these two variables together (see figure 13) due to a few individuals with very large numbers of retweets and followers (such as Trump)

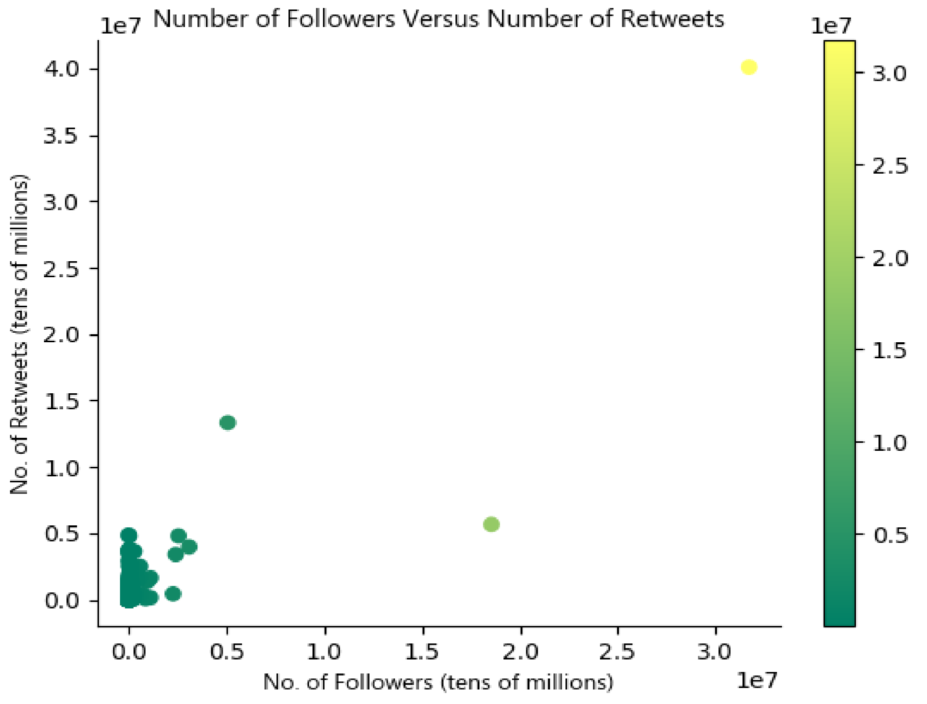


Figure 13 – number of followers versus number of retweets

Correcting for this by removing those data points with more than 10,000,000 retweets or more than 10,000,000 followers, I can see a much clearer picture in figure 14. This shows there may be some correlation here although it is not very obvious to see.

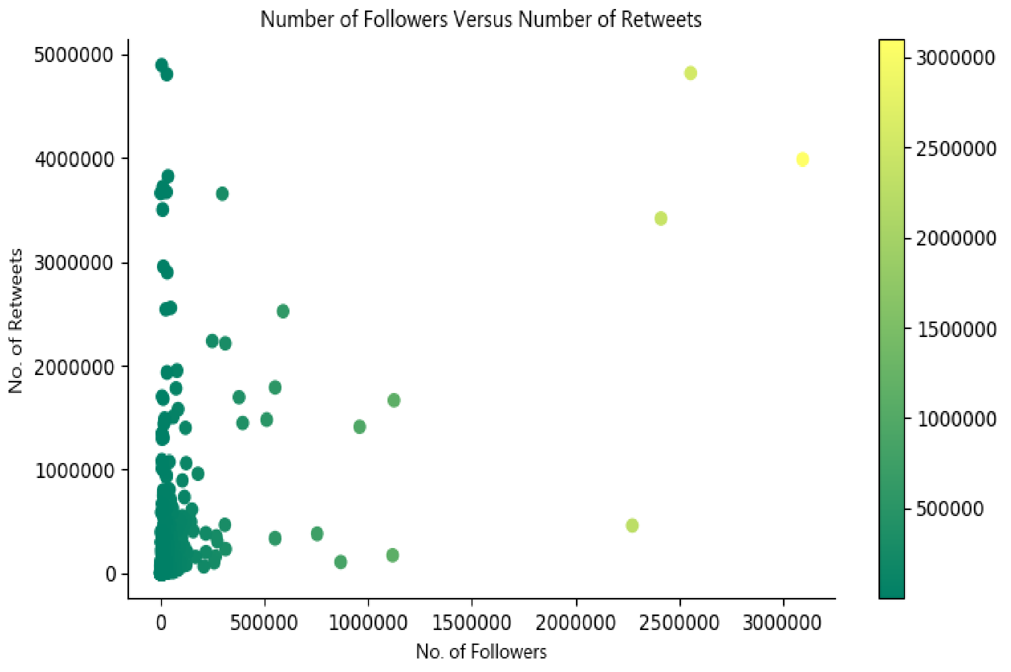


figure 14 – number of followers versus number of retweets (no outliers)

To determine which correlation measure to use, at this point it is necessary to do a test for normality on both variables. Two tests were carried out: the D'Agostino's K2 test, and the Anderson-Darling test. The results are as follows:

Number of followers (D'Agostino's K2 test):
statistic = 858.2561, p value = 9.441xe-183

Number of followers (Anderson-Darling test):
statistic = 148.1367, critical values = 0.572, 0.651, 0.781, 0.911, 1.084

Since in both cases the statistic is very high, and in the case of the Anderson-Darling test, higher than the critical values, this clearly shows that the data is not normally distributed, as one might expect.

Number of retweets (D'Agostino's K2 test):
statistic = 466.7973, p value = 4.3277xe-102

Number of retweets (Anderson-Darling test):
statistic = 98.304, critical values = 0.572, 0.651, 0.781, 0.911, 1.084

Again, like with the followers distribution, this is clearly not normally distributed. It is worth also noting that it was attempted to find the underlying normal distribution of this data using the box-cox function in Python, however this did not result in normally distributed data. Therefore, I will use the Spearman's and Kendall's Tau tests for correlation. The results are as follows:

Spearman's correlation: statistic = 0.506, p value = 9.08xe-37
Kendall's correlation: statistic = 0.358, p value = 8.13xe-36

Both of these measures clearly show some positive correlation, which confirms my theory.

5.2 Number of retweets versus tone/ sentiment

It can be seen from the chart in figure 15 that there is no obvious correlation between the sentiment of a tweet and how many retweets it gets. As before, normality tests were performed on the data before applying correlation measures. The results are below.

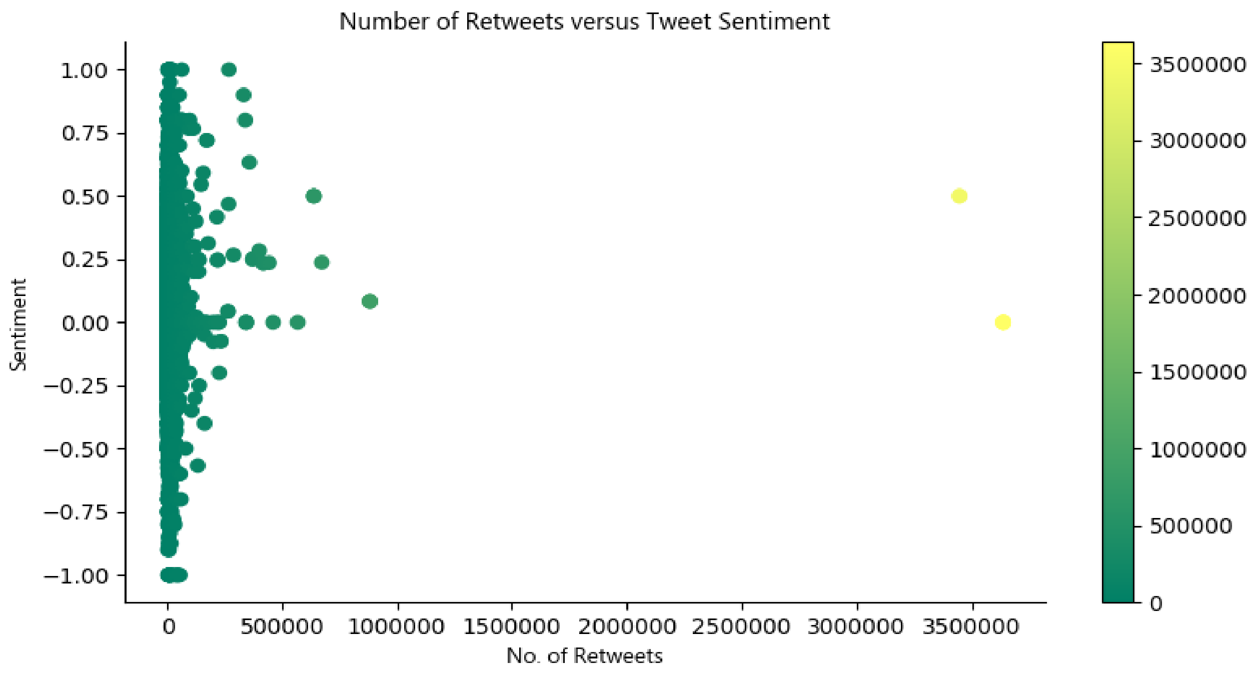


figure 15 – number of retweets versus tweet sentiment

Number of retweets (different from my retweets data from 5.1 - D'Agostino's K2 test):
statistic = 24490.305, p value = 0.0

Number of retweets (Anderson-Darling test):
statistic = 2836.525, critical values = 0.576, 0.656, 0.787, 0.918, 1.092

Sentiment (D'Agostino's K2 test):
statistic = 660.022, p value = 4.765xe-144

Number of retweets (Anderson-Darling test):
statistic = 266.428, critical values = 0.576, 0.656, 0.787, 0.918, 1.092

As can be seen from these results, neither data is normally distributed. This is slightly of a surprise regarding sentiment, as it might have been presumed that most tweets would be fairly neutral in tone, with fewer and fewer becoming more extreme. However this does not seem to be the case. I can now apply the non-parametric correlation measures as below:

Spearman's correlation: statistic = -0.073, p value = 1.837xe12
Kendall's correlation: statistic = -0.049, p value = 6.335xe-12

This shows that I have an extremely weak negative correlation, if any at all, between the number of times a tweet is retweeted and it's tone.

5.3 Number of followers versus tone

After removing data points with a number of followers greater than 10,000,000 (for the purposes of the graph only), as in section 5.1, as before it is still difficult to tell if there is any correlation (see figure 16). It looks as though there may be a slight negative correlation, which I can test.

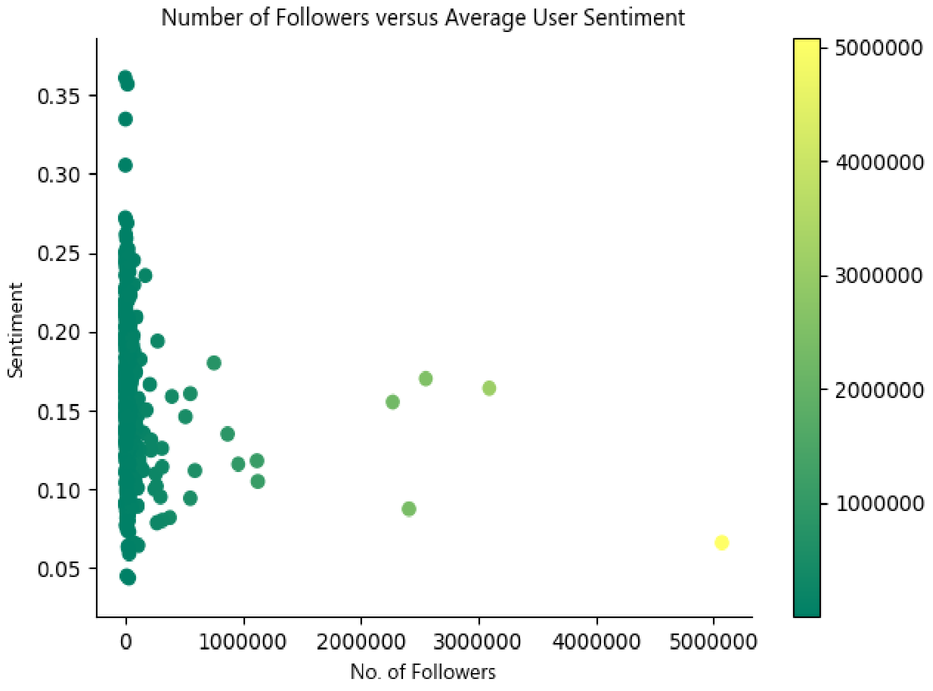


figure 16 – number of followers versus average user sentiment

I already know that the number of followers is not normally distributed, so I can apply my non-parametric correlation measures directly.

Spearman's correlation: statistic = -0.277, p value = 4.989xe11
Kendall's correlation: statistic = -0.190, p value = 3.054xe-11

This does seem to show a slight negative correlation between a user's tone and the number of followers they have. It is interesting to note that Donald Trump supports this, with an average sentiment of 0.113 (average sentiment of all users is 0.1541).

6 Topic Analysis

6.1 Word frequency

For a simple way to see what sorts of issues are frequently mentioned or talk about, I looked at word counts. In order to get more meaningful results, counts were also taken after first trying porter stemming, then lemmatizing. Both stemming and lemmatizing are essentially methods to combine words with similar meanings (such as great, greatly, greatest etc.) The main difference being that stemming may not result in an actual word (and tends to be faster), whereas lemmatizing does.

Below are the word cloud results of all the tweets from 2016 (figure 17), all the tweets from 2017 (figure 18) and all user descriptions (figure 19). In all cases this was subsequent to lemmatizing.

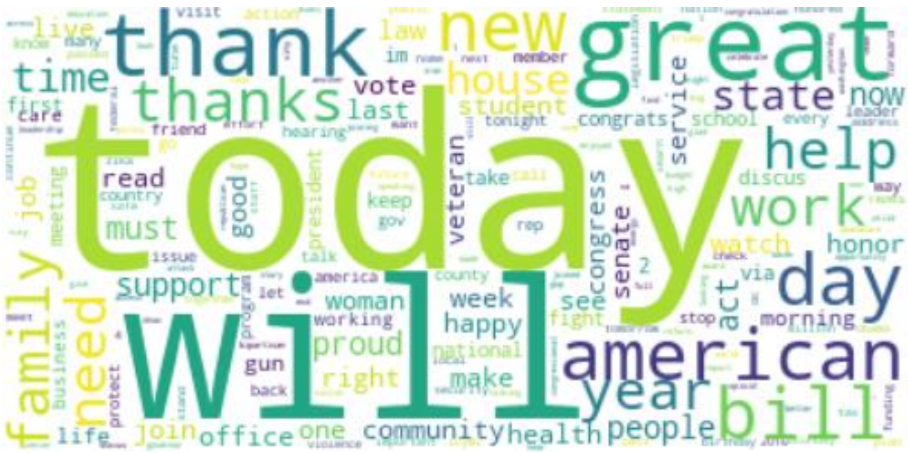


figure 17 – word cloud of all tweets from 2016

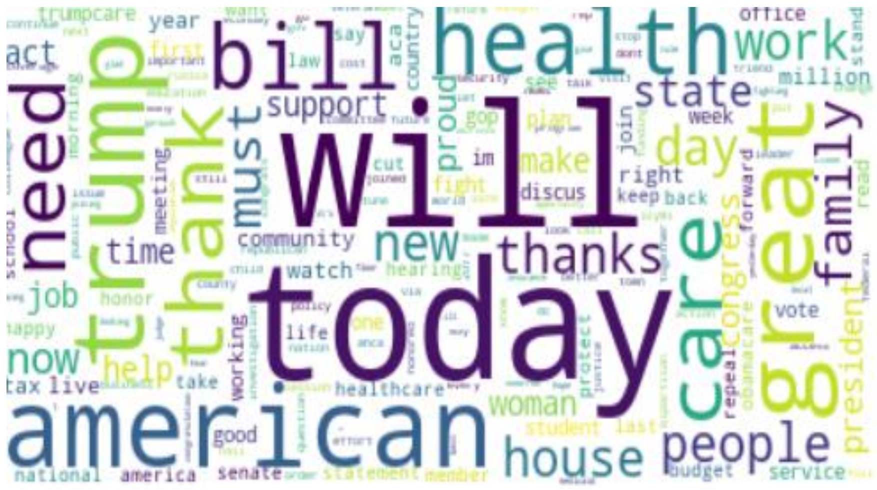


figure 18 – word cloud of all tweets from 2017

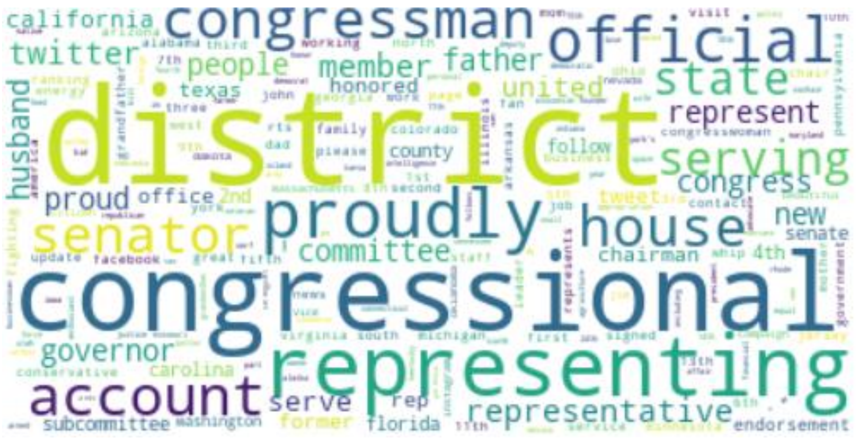
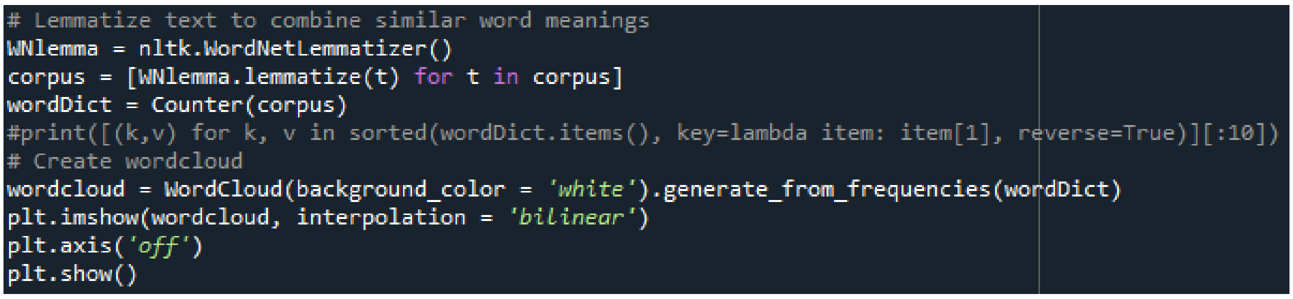


figure 19 – word cloud of all user descriptions

It is perhaps not surprising that "Trump" is mentioned a lot in 2017, immediately after his inauguration. It can also be seen that in 2017 word like "health" and "care" start to appear more often, suggesting that this issue of health has become higher on the agenda – which again is not surprising considering the conflict between Trump and the obamacare system. The user descriptions seem to reflect a very formal tone, with a simple description of their role, which perhaps befits a congressional member. Figure 20 demonstrates the code used to generate the above word clouds

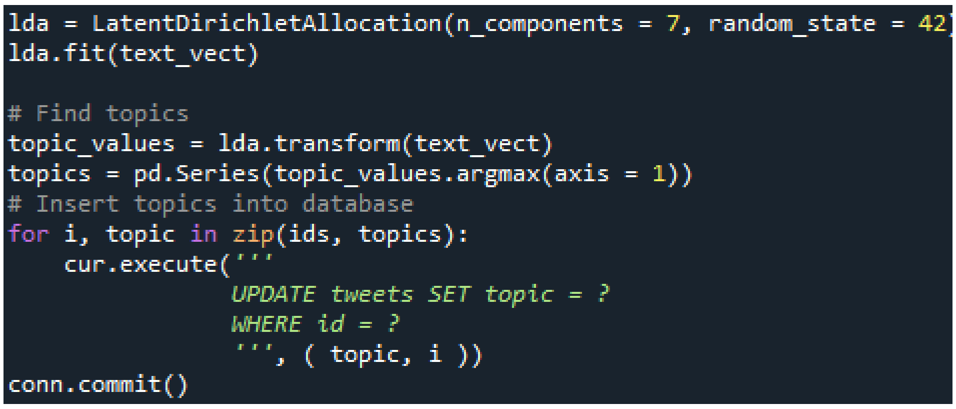


```
# Lemmatize text to combine similar word meanings
WNlemma = nltk.WordNetLemmatizer()
corpus = [WNlemma.lemmatize(t) for t in corpus]
wordDict = Counter(corpus)
#print([(k,v) for k, v in sorted(wordDict.items(), key=lambda item: item[1], reverse=True)][:10])
# Create wordcloud
wordcloud = WordCloud(background_color = 'white').generate_from_frequencies(wordDict)
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis('off')
plt.show()
```

figure 20 – word cloud code

6.2 NMF & LDA Topic analysis

Both NMF (non-negative matrix factorization) and LDA (Latent Direchlet Allocation) were used to find topics in the tweet text. Initially, this was applied only for 2017 tweets due to memory limitations. Once topics had been allocated to each tweet, they could then be inserted into the database (see figure 21)

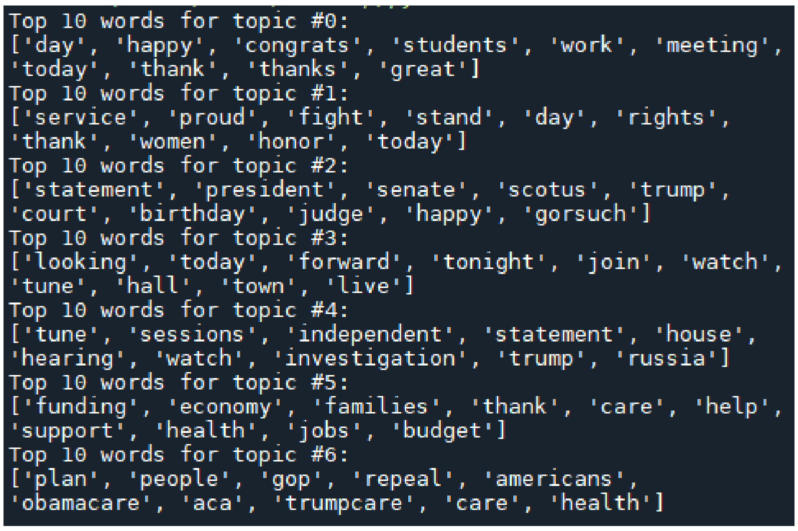


```
lda = LatentDirichletAllocation(n_components = 7, random_state = 42
lda.fit(text_vect)

# Find topics
topic_values = lda.transform(text_vect)
topics = pd.Series(topic_values.argmax(axis = 1))
# Insert topics into database
for i, topic in zip(ids, topics):
    cur.execute('''
                UPDATE tweets SET topic = ?
                WHERE id = ?
                ''', ( topic, i ))
conn.commit()
```

figure 21 – code for transferring LDA topics to database

It was found that LDA resulted in the most meaningful topics (see figure 22) along with a human interpretation of each topic:


```
Top 10 words for topic #0:
['day', 'happy', 'congrats', 'students', 'work', 'meeting',
'today', 'thank', 'thanks', 'great']
Top 10 words for topic #1:
['service', 'proud', 'fight', 'stand', 'day', 'rights',
'thank', 'women', 'honor', 'today']
Top 10 words for topic #2:
['statement', 'president', 'senate', 'scotus', 'trump',
'court', 'birthday', 'judge', 'happy', 'gorsuch']
Top 10 words for topic #3:
['looking', 'today', 'forward', 'tonight', 'join', 'watch',
'tune', 'hall', 'town', 'live']
Top 10 words for topic #4:
['tune', 'sessions', 'independent', 'statement', 'house',
'hearing', 'watch', 'investigation', 'trump', 'russia']
Top 10 words for topic #5:
['funding', 'economy', 'families', 'thank', 'care', 'help',
'support', 'health', 'jobs', 'budget']
Top 10 words for topic #6:
['plan', 'people', 'gop', 'repeal', 'americans',
'obamacare', 'aca', 'trumpcare', 'care', 'health']
```
figure 22 – LDA topics most significant words

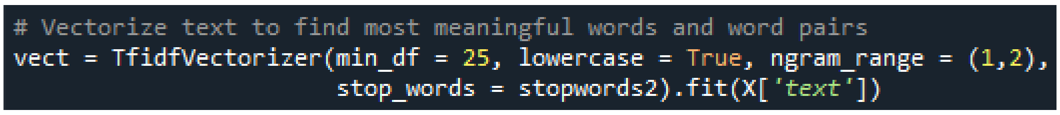
Topic 0: Everyday work
Topic 1: Justice
Topic 2: Political institutions
Topic 3: Events
Topic 4: Russian interference investigation
Topic 5: Growing the economy and creating jobs
Topic 6: Healthcare

6.3 TF-IDF

A TF-IDF (term frequency – inverse document frequency) vectorizer was also applied to the "just_text" column, to find the most "significant" words, which will also be the first step in any potential machine learning. I also attempted to use an ngram_range of (1,2) to find significant pairs of words, however this can only be done to limited chunks of the data, due to the memory limitations of the software. For purposes of demonstration, I applied this only to Trump's tweets; some of the most significant pairs of words were:

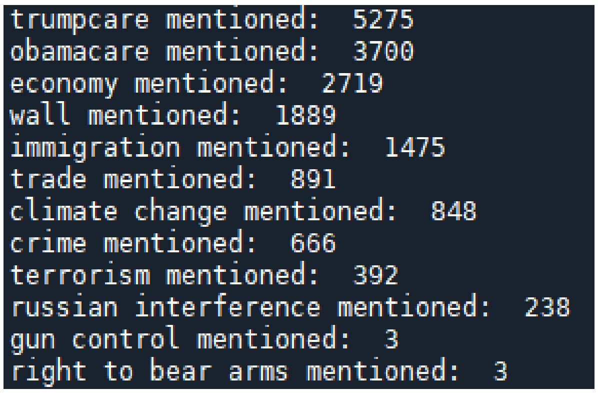
"wife melania", "totally rigged", "treated badly", "today signed", "total disaster", "obamacare dead"

Which, while not being especially enlightening is at least amusing. See figure 23 for the code.


```
# Vectorize text to find most meaningful words and word pairs
vect = TfidfVectorizer(min_df = 25, lowercase = True, ngram_range = (1,2),
                       stop_words = stopwords2).fit(X['text'])
```
figure 23 – code for applying TF-IDF vectorizer

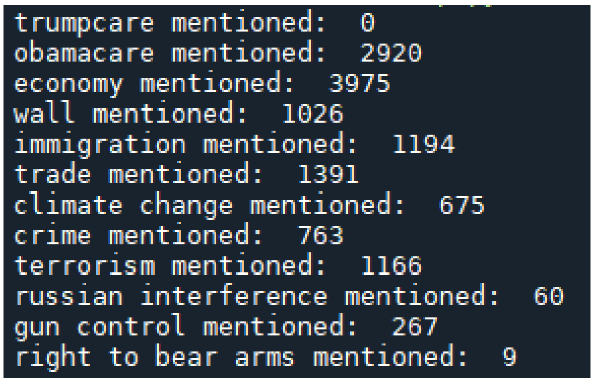
6.4 Issues count

I also looked at the frequency of certain issues that matter, and how this changed over time. Some key policy issues were picked out, and their frequency looked at in 2017 (figure 24) and in 2016 (figure 25) as major and eventful years in US politics.



```
trumpcare mentioned:  5275
obamacare mentioned:  3700
economy mentioned:  2719
wall mentioned:  1889
immigration mentioned:  1475
trade mentioned:  891
climate change mentioned:  848
crime mentioned:  666
terrorism mentioned:  392
russian interference mentioned:  238
gun control mentioned:  3
right to bear arms mentioned:  3
```

figure 24 – issue counts 2017



```
trumpcare mentioned:  0
obamacare mentioned:  2920
economy mentioned:  3975
wall mentioned:  1026
immigration mentioned:  1194
trade mentioned:  1391
climate change mentioned:  675
crime mentioned:  763
terrorism mentioned:  1166
russian interference mentioned:  60
gun control mentioned:  267
right to bear arms mentioned:  9
```

figure 25 – issue counts 2016

It is clear that trumpcare became a big focus in 2017 with the inauguration of Trump, so this is a major recent focus, and confirms the recent increased focus on healthcare I've already discovered.
The economy is mentioned much more in 2016 as this would have been an important factor in determining the outcome of the election.

I can also compare this to issues that voters actually care about. See figure 26 for statistics from the Pew Research Center, showing what issues are the most important to voters in the 2016 election (with a comparison in the form of my 2016 counts in figure 27).

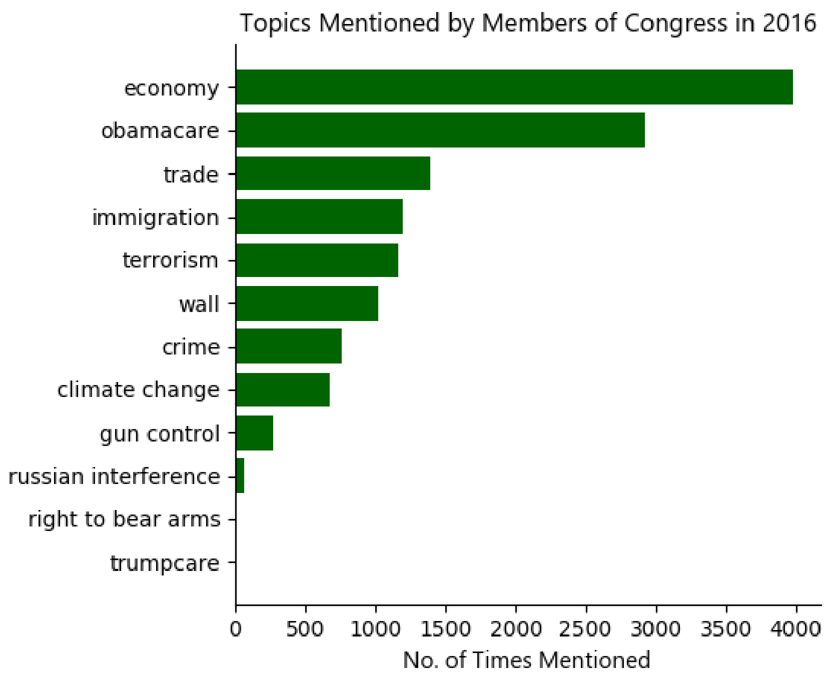


figure 27 – topics mentioned by members of congress 2016

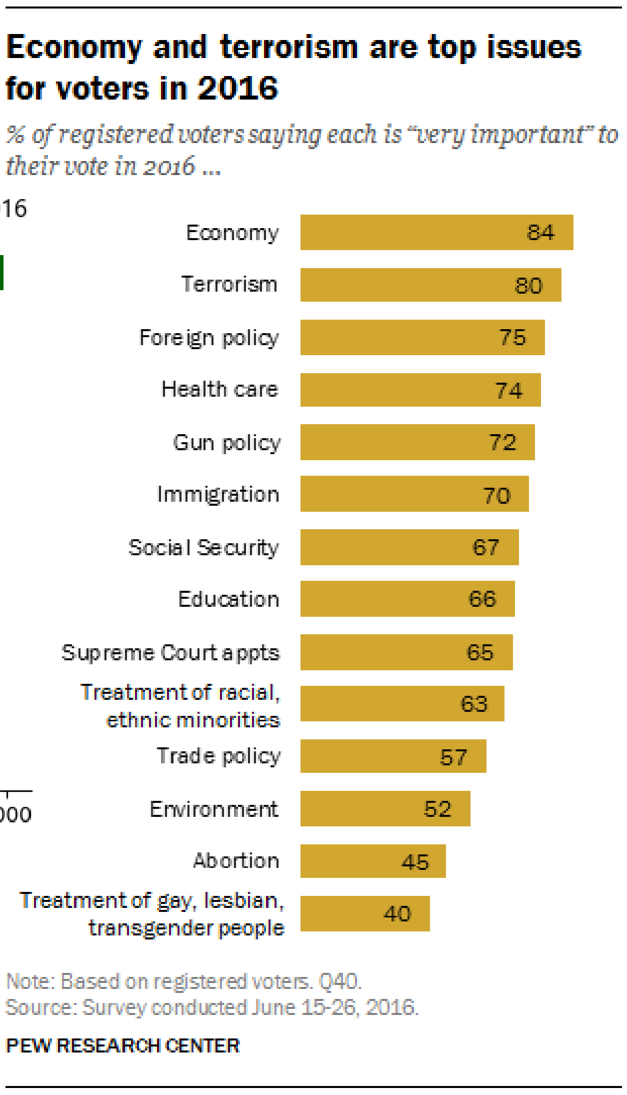


figure 26 – voter top issues 2016 (Pew Research Center)

It does seem to agree with official statistics, that the economy and terrorism were top issues in the run up to the election, and also healthcare and immigration. This seems to show that for the most part. politicians were talking about the issues that mattered most to voters.

The main differences are that politicians seem to talk more about trade, and less about gun control, as issues that matter to voters.

Of course, this method of finding topic importance has its limitations; such as the fact it is perfectly possible to talk about, say, terrorism, without actually mentioning the word "terrorism". Therefore this is to be thought of as an indication only, as I may not have found all actual instances of a topic being discussed in this way. To take this further I could look at searching for more words/phrases related to a topic, maybe even look at lemmatizing to reduce the number different words with similar meanings.

7. Summary of Findings

Referring back to my original hypotheses, I am now able to summarize my findings as follows:

- What topics are most frequently discussed?
    - Healthcare seems to be an important topic in 2017, even more so than in 2016, with particular focus on Trumpcare.
    - In 2016, congress members seemed to talk about topics that, in the main, coincided with what mattered to voters in the lead up to the 2016 election. Thus the economy, terrorism, healthcare and immigration were among the most mentioned /important topics.
    - Regarding the above points, my hypotheses that major policy initiatives (such as Obamacare and Trump's wall) would be mentioned often, turned out to be correct.
- Who are the most influential figures?
    - Trump had the most followers and therefore the largest audience, with Bernie Sanders and other eminent congress members also having large numbers of followers. This confirms my hypothesis.
    - I have shown a significant positive correlation between number of followers and number of retweets, confirming my hypothesis.
- What are the most popular tweets?
    - My results showed very little, if any, correlation between tweet popularity (measured by number of retweets) and sentiment. If anything, correlation was slightly negative, disproving my theory.

In addition to findings referring to my hypothesis, I have also discovered the following:

- Total number of tweets tweeted increases year by year as Twitter gains in popularity
- There is a significant negative correlation between average user sentiment and number of followers.

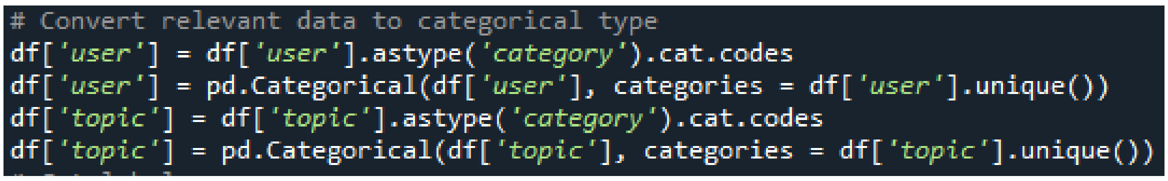
8. Machine Learning

Lastly, a machine learning model will be created with the purpose of predicting the number of retweets a certain tweet will get. Date created will be omitted, as tweets are likely to get less retweets in the past, and as the model will be used on new tweets, this could muddy the data. I will also take tweets from the most recent, backwards, and none prior to 2016. This will ensure that there are no tweets that are essentially "incorrectly" labelled as having a low number of retweets, simply because it is old. Favourites_count will also be omitted, as this information is only known after a tweet is tweeted, and so will bias the results. It is, after all, essentially another popularity label, and including it in the training data would thus constitute data leakage.

8.1 Data preparation

After much experimentation, I found that taking 50,000 tweets from the dataset was the optimum amount to enable as high an accuracy as possible without the model taking hours to run (law of diminishing returns). Therefore the last 50,000 tweets were chosen.

Next, the topic and user_id columns had to be changed to categorical (see figure 28)



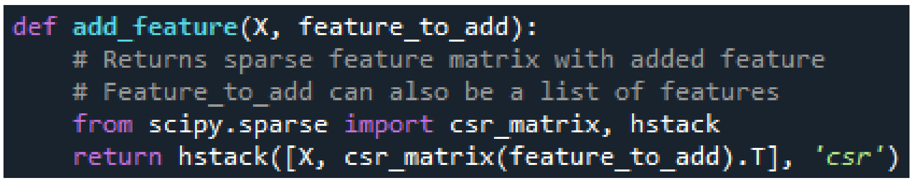
```
# Convert relevant data to categorical type
df['user'] = df['user'].astype('category').cat.codes
df['user'] = pd.Categorical(df['user'], categories = df['user'].unique())
df['topic'] = df['topic'].astype('category').cat.codes
df['topic'] = pd.Categorical(df['topic'], categories = df['topic'].unique())
```

figure 28 – converting data to categorical type

Furthermore, porter stemming was found to have better results over lemmatizing the text data. After that, a simple train/test split was used, as being more efficient on a larger dataset than k-fold cross validation.

TruncatedSVD was also used on the sparse matrix returned after text vectorization, in order to reduce the number of features into a range that the model could handle without overfitting.

Additional features (topic, sentiment, and user id) were added to the sparse matrix using the code shown in figure 29.

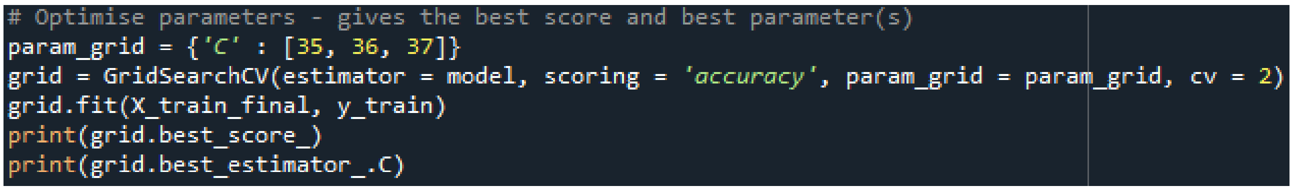


```
def add_feature(X, feature_to_add):
    # Returns sparse feature matrix with added feature
    # Feature_to_add can also be a list of features
    from scipy.sparse import csr_matrix, hstack
    return hstack([X, csr_matrix(feature_to_add).T], 'csr')
```

figure 29 – adding features to data for machine learning

8.2 Parameter tuning

Parameter tuning was performed via a grid search (see code in figure 30 for example)



```
# Optimise parameters - gives the best score and best parameter(s)
param_grid = {'C' : [35, 36, 37]}
grid = GridSearchCV(estimator = model, scoring = 'accuracy', param_grid = param_grid, cv = 2)
grid.fit(X_train_final, y_train)
print(grid.best_score_)
print(grid.best_estimator_.C)
```

figure 30 – grid search for parameter tuning

cv (cross validation) in this case had to be set to 2, due to the large amount of data involved – even with this the grid search would take a long time to run.

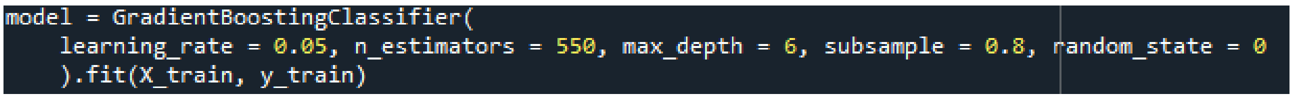
8.3 Multi-class classification

For multi-class classification, I used 50,000 tweets, but for each class ensured that each had the same number of samples, to avoid the any problems associated with having imbalanced classes (such as biasing the classifier to the class with the most examples). The classes were defined as:

- number of retweets = 0
- number of retweets = between 1 and 100
- number of retweets = greater than 100

These classes seemed to represent a good intuitive distinction of "not popular", "slightly popular" and "very popular". They also seemed to yield reasonable results in the resulting model.

After applying the data preparation outlined in section 8.1, several different models were trialled, starting simply, then progressing through random forest classifier and support vector classifier, which produced reasonable results. The best results were found with the gradient boosting classifier (see figure 31)



```
model = GradientBoostingClassifier(
    learning_rate = 0.05, n_estimators = 550, max_depth = 6, subsample = 0.8, random_state = 0
    ).fit(X_train, y_train)
```

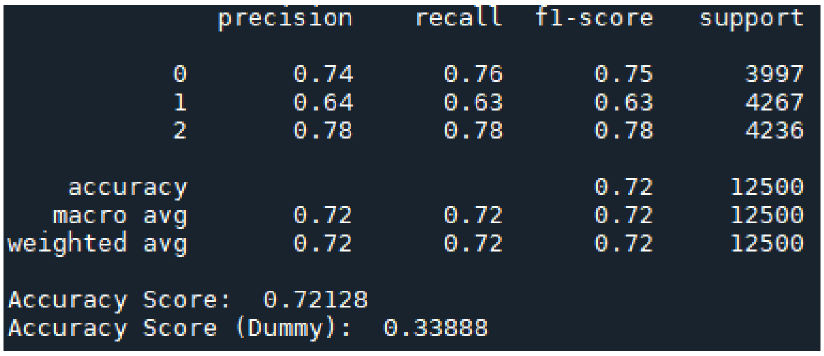
figure 31 – model fitting gradient boosting classifier

Essentially, boosting algorithms such as the above create a sequence of models that attempt to correct the mistakes of the models before them in the sequence. Once created, the models make predictions which can be weighted by their calculated accuracy and the results are combined to create the final result.

The distance between prediction and truth of each individual model represents the error rate of the model. These errors are used to calculate the gradient (basically the partial derivative of the loss function – so it describes the steepness of the error function) The gradient is then used to minimize the loss function (essentially the rate of error).

When tuning, there is always a trade-off between the learning rate and the number of models, or "trees" (n_estimators). To slow down learning (and prevent over-fitting), I can lower the learning rate (default is 0.1). This is also sometimes referred to as the shrinkage factor, and acts like a weighting for the corrections by new trees when added to the model. Setting this value lower means that fewer corrections are made for each tree added to the model. This in turn results in more trees needing to be added, hence the high number of n_estimators in the final model above.

Furthermore, a dummy classifier was trained on the data to create a baseline for comparison. The results are shown (along with dummy for comparison) in figure 32.



```
              precision    recall  f1-score   support

           0       0.74      0.76      0.75      3997
           1       0.64      0.63      0.63      4267
           2       0.78      0.78      0.78      4236

    accuracy                           0.72     12500
   macro avg       0.72      0.72      0.72     12500
weighted avg       0.72      0.72      0.72     12500

Accuracy Score:  0.72128
Accuracy Score (Dummy):  0.33888
```

figure 32 – results multi class classifier

Also see figure 33 for the confusion matrix.

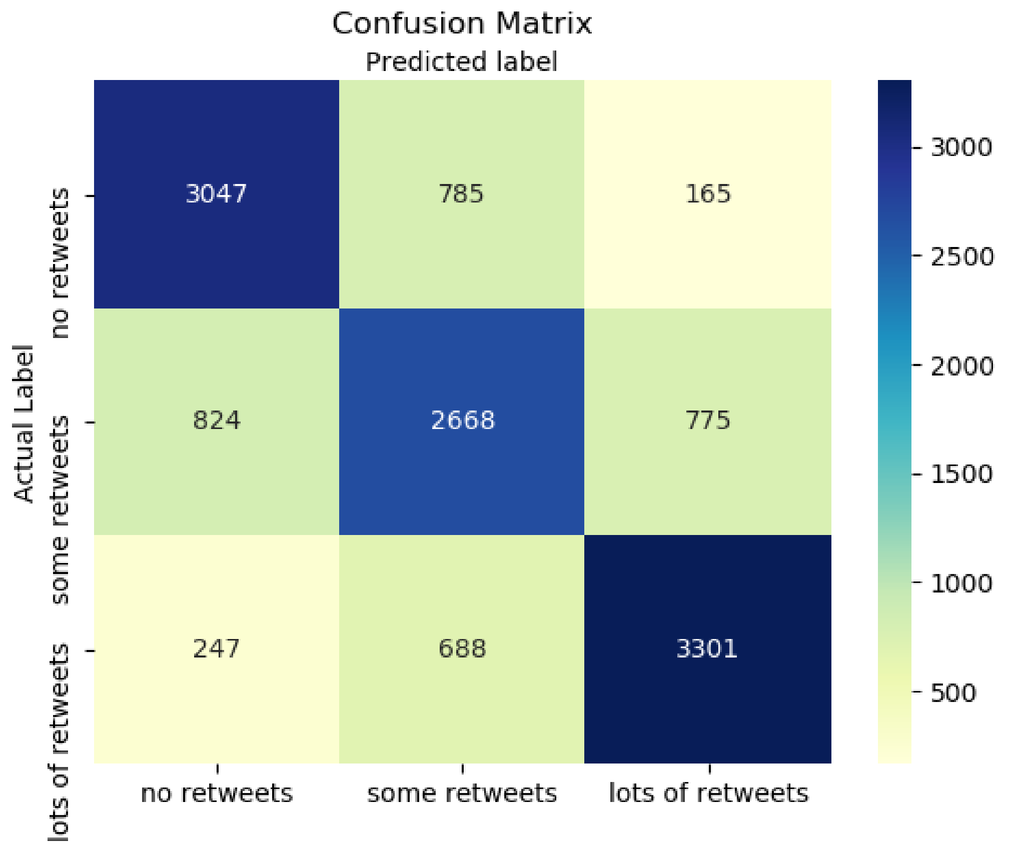
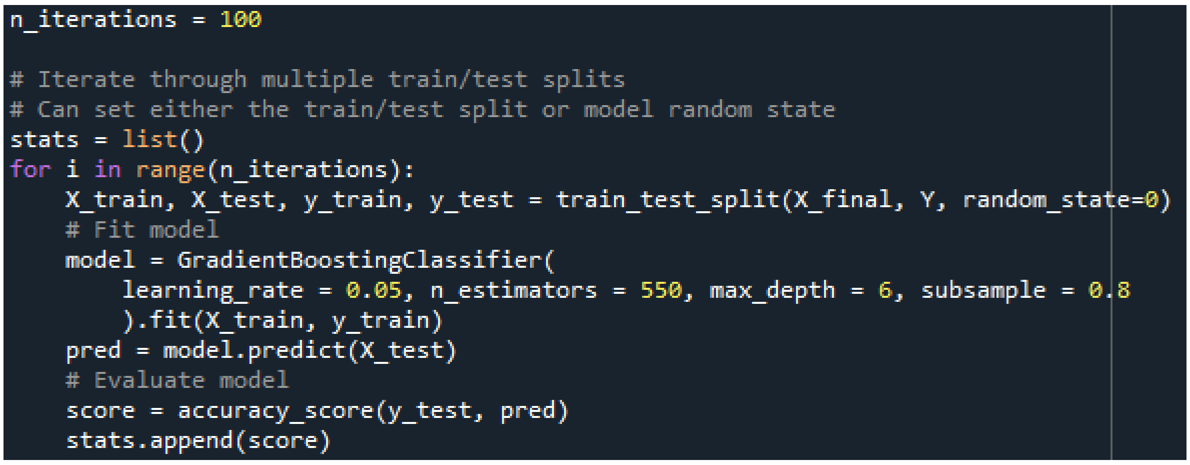


figure 33 – confusion matrix multi class classifier

This is a reasonable result, especially as the classes chosen are fairly arbitrary as far as the machine learning model is concerned. It can be seen from the confusion matrix that labels next to each other (ie. no retweets and some retweets) are more likely to be misclassified as the other, which makes sense as a tweet with 99 retweets and one with 100 retweets, say, are likely to have very similar features. It is also reassuring that my classifier is working as expected – also with no particular bias towards precision or recall.

8.3.1 Classification confidence interval

To find out the confidence interval of my classifier accuracy, I removed the fixed random state on my model, and created the model over a number of iterations (see figure 35). 100 iterations was chosen as a sufficient number to produce a normal distribution of accuracy scores, while not taking too long to run (it still took around 24 hours).

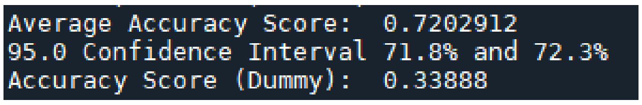


```
n_iterations = 100

# Iterate through multiple train/test splits
# Can set either the train/test split or model random state
stats = list()
for i in range(n_iterations):
    X_train, X_test, y_train, y_test = train_test_split(X_final, Y, random_state=0)
    # Fit model
    model = GradientBoostingClassifier(
        learning_rate = 0.05, n_estimators = 550, max_depth = 6, subsample = 0.8
        ).fit(X_train, y_train)
    pred = model.predict(X_test)
    # Evaluate model
    score = accuracy_score(y_test, pred)
    stats.append(score)
```

figure 35 – code to find accuracy score distribution of model

The results are shown in figure 36 with a histogram of the accuracy scores shown in figure 37.



```
Average Accuracy Score:  0.7202912
95.0 Confidence Interval 71.8% and 72.3%
Accuracy Score (Dummy):  0.33888
```

figure 36 – accuracy confidence interval of model

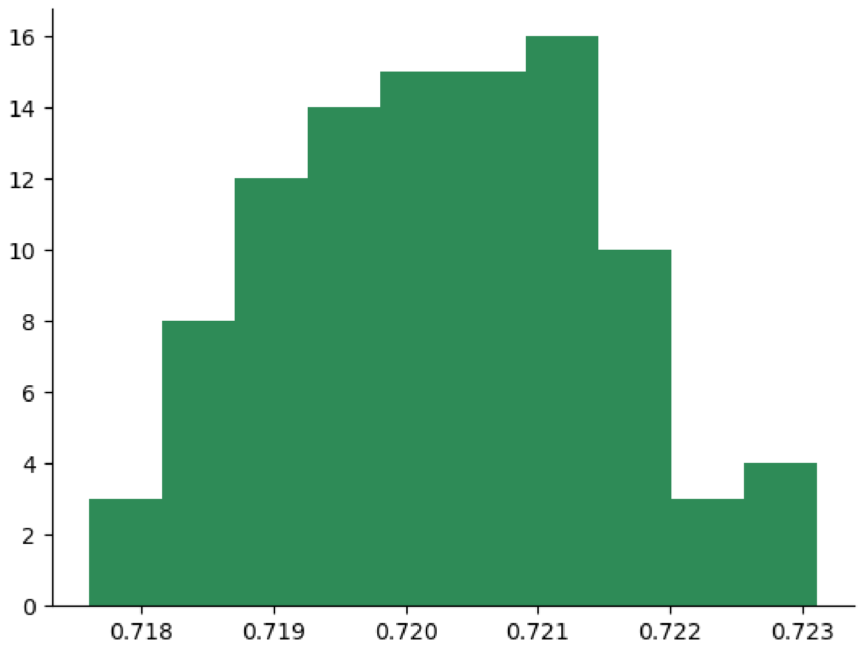


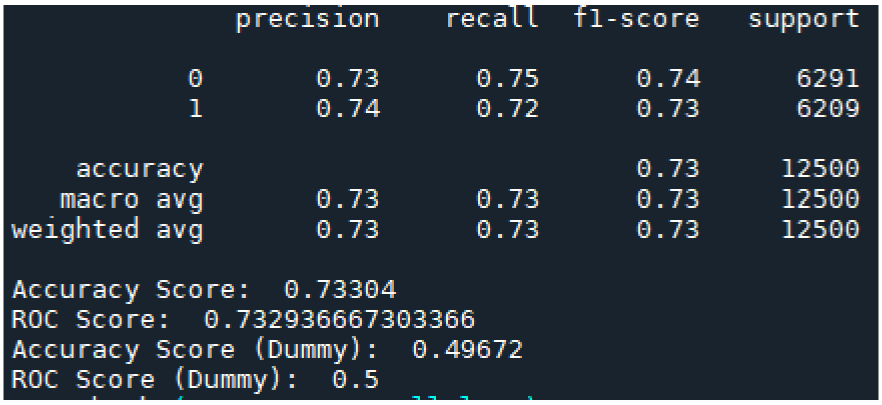
figure 37 – accuracy score distribution of model

8.4 Binary class classification

As a comparison, and as an attempt to improve the accuracy of my model, I can also redefine the problem into a binary classification problem; where class 0 is no retweets, and class 1 means 1 or more retweets. So essentially – will this tweet be taken notice of?

As before, it was ensured that there were an equal number of instances in each class, with a total number of 50,000 instances.

It was then found that the same model with the same parameters also achieved the most optimal results for this problem too. See figure 38 for results (along with dummy for comparison) and figure 39 for confusion matrix.



```
              precision    recall  f1-score   support

           0       0.73      0.75      0.74      6291
           1       0.74      0.72      0.73      6209

    accuracy                           0.73     12500
   macro avg       0.73      0.73      0.73     12500
weighted avg       0.73      0.73      0.73     12500

Accuracy Score:  0.73304
ROC Score:  0.732936667303366
Accuracy Score (Dummy):  0.49672
ROC Score (Dummy):  0.5
```

figure 38 – results binary classifier

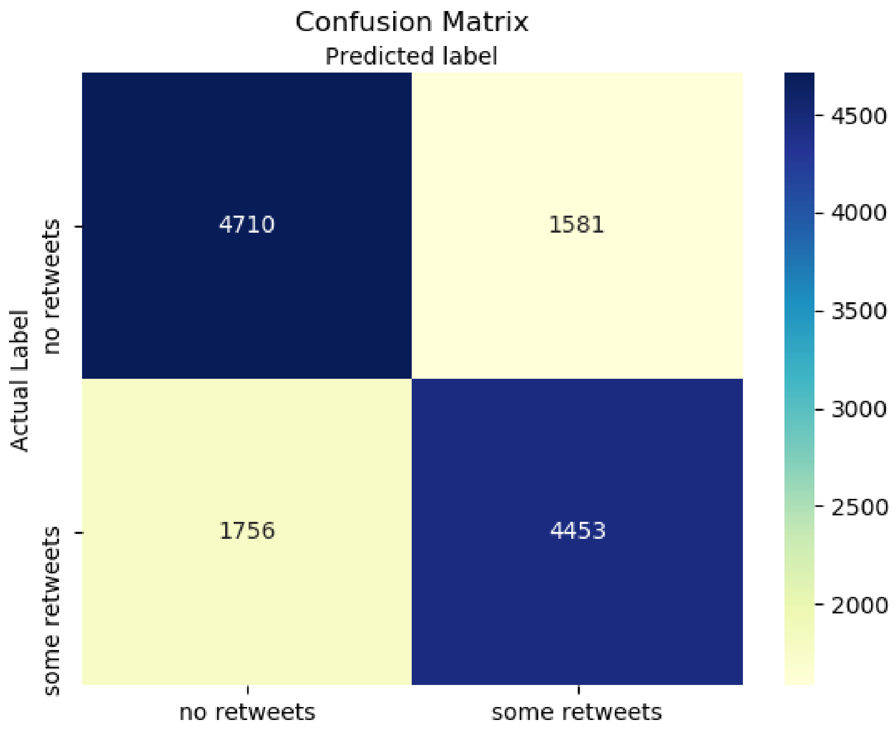


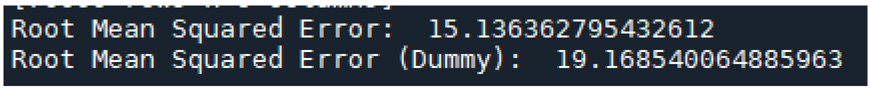
figure 39 – confusion matrix binary classifier

It can be seen here that I have not achieved the increase in accuracy that I might have hoped for; however this is perhaps to be expected, as there is in reality very little difference between a tweet that gets no retweets, and a tweet that gets 1, say. And so setting our boundary to 0 or more than 0 is slightly arbitrary as far as our model is concerned. That being said, there is still a significant improvement on the score from the dummy.

8.5 Regression

As a final challenge, I will change this into a regression problem. It will be a very difficult task to predict the exact amount of retweets a tweet may get, so it will be interesting to see how accurate I can make my model. For this problem the root mean squared error was used as the main scoring metric (along with a dummy score for a baseline, with strategy='mean'). Also the number of retweets of any tweet in the dataset was capped at 100; otherwise tweets with large numbers of retweets ended up massively increasing the error of the model due to the very high values the model could predict up to. In this case, I would treat any value of above 100 as simply having "lots" of retweets and outside of my scope.

The final results can be seen in figure 40.



```
Root Mean Squared Error:  15.136362795432612
Root Mean Squared Error (Dummy):  19.168540064885963
```

figure 40 – results regression

Here it was found that slightly more data (70,000 data instances) was the most optimal for achieving the best RMSE. However, despite that the performance of the model is not hugely better than the dummy. This was to be anticipated to to the nature of the problem, but was an interesting exercise nonetheless.

8.6 Taking it further

Going forward, it would be really interesting to see how an XGBoost model might perform on this data, as this has a reputation for working well for text classification problems, and I have already demonstrated that gradient boosting works well on this data. It should also improve execution speed of the model.

XGBoost stands for extreme gradient boosting, and differs from ordinary gradient boosting (the one used in sections 8.3, 8.4 and 8.5) in that, as well as the loss function, it computes the second-order gradient (I.e. second partial derivative) of the loss function, which provides more information about the direction of gradients and helps reduce the amount of error. It also uses regularization (L1 and L2) to improve the generalisation of the model.

Unfortunately, it was not possible to install XGBoost on the software version available, so it was not possible to try it out here.