# *Detailed Report on Twitter Text Mining for Sentiment Analysis and Like Prediction*

By Kartik Kokane

## Introduction

Every year, companies and organisations spend millions of dollars into market analysis so they can understand how a product or service would do in the market. Since the dot com bubble bursted in 2000, internet has become a whole new world in its own. Every organisation has a virtual identity where they promote their brand and conduct e-commerce. Similarly, social media has gained popularity in recent years, but never had a considerable impact on presidential elections until 2016 when Trump used Twitter to tip the election into his favour.

Twitter is one of the most widely used social networking sites, which has over 353 million active users. It is a microblogging platform that enables users to communicate in 280 characters or fewer about their thoughts, feelings, and ideas. Tweets are a perfect data source for sentiment analysis because of their condensed nature. Finding and extracting subjective information from text data is the process of sentiment analysis, commonly referred to as opinion mining. It entails examining the language, tone, and feelings conveyed in the text to ascertain the writer's feelings on a given subject, good, or service. Businesses have a chance to learn more about client preferences, opinions, and comments thanks to the enormous volume of data provided on Twitter. Text mining methods can be applied to Twitter data to extract useful information and gauge user sentiment towards a certain subject or company. Twitter sentiment analysis may assist businesses in making data-driven decisions, such as finding areas for improvement, comprehending client demands and preferences, and tracking the success of their marketing activities.

In this report, we perform text mining on the tweets of Donald Trump and try to find a relation between the 2016 presidential elections and his twitter campaign. We perform sentiment analysis and understand how it works in depth. We also attempt to build machine learning models that can accurately predict the amount of likes a tweet would get.

## Methodology

The dataset being used is a raw data file with 40,000 rows and needs to be processed in order to make it suitable for all machine learning tasks. The dataset has five variables, text, retweets, likes, date and isFlagged. In order to make the tweets understandable to the algorithm, they have to be cleaned and processed. Like the term, ‘Garbage in, garbage out’, data cleaning is of high priority as a bad or incorrect dataset might yield poor results.

### Preprocessing

***Focusing on data from presidency periods:***
Since the tweets in the dataset are from 2011 to 2021, we reduce the complexity of the data by choosing only tweets from 2014 to 2020 which signifies the period just before, during and just after the Trump presidency.

***Removing short length tweets:***
Since tweets with shorter lengths do not really form a complete sentence, we eliminate them to avoid any unfair results in the analysis. We discard all tweets that have less than 5 words.

***Tokenization of text:***
The first column, 'text', which contains all the tweets is loaded into a variable. This text is then converted into a dictionary of words by using the tokenize() function.
Removing frequent unique words:
Some words like names of people, slang words, texting jargons are not included in the NLTK libraries. To tackle this issue, we create our own dictionary of stopwords which contains Trump's social handle, internet jargons like 'http', twitter jargons like 'rt'. Since these words are used very frequently, they can yield incorrect results.

***Removing symbols and special characters:***
In general, twitter has a character limit of 140 for tweets so users tend to use special characters to tag people and get their point across like '@' for account handles, 'https' for a sharing image or web page links. The NLTK library provides sub-sets or dictionaries of words that can be used to deal with these characters like stopwords(), which is used to remove all punctuation marks and special characters used in the tweets.

***Remove filler words:***
The .isAlpha() function is used to remove any words that are not important, these are usually filler words like 'the', 'in', which adds unnecessary load on the algorithm.

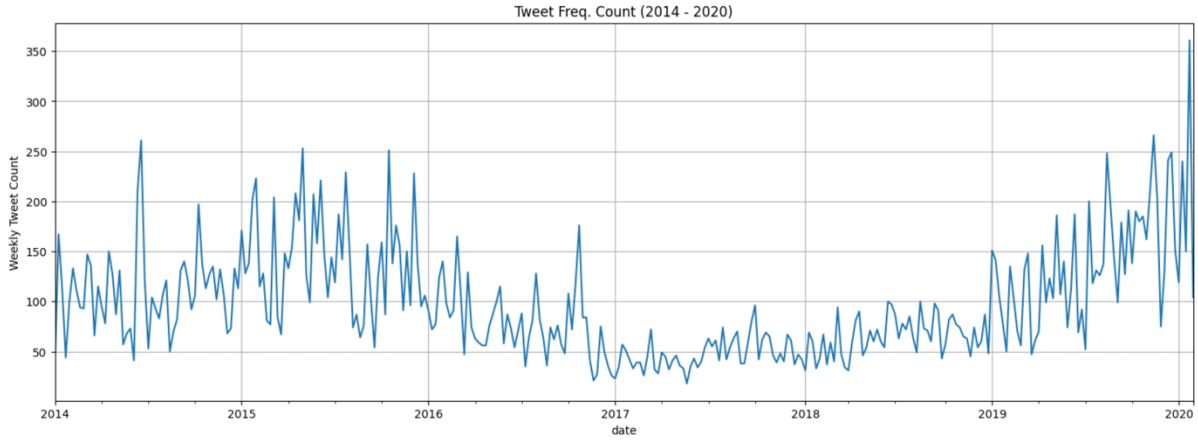
***Converting to lowercase:***
Since the NLTK libraries are case sensitive, all text needs to be converted to lower case in order to avoid duplicates in our results.

After processing the data, we are left with a dataframe of 31077 rows.

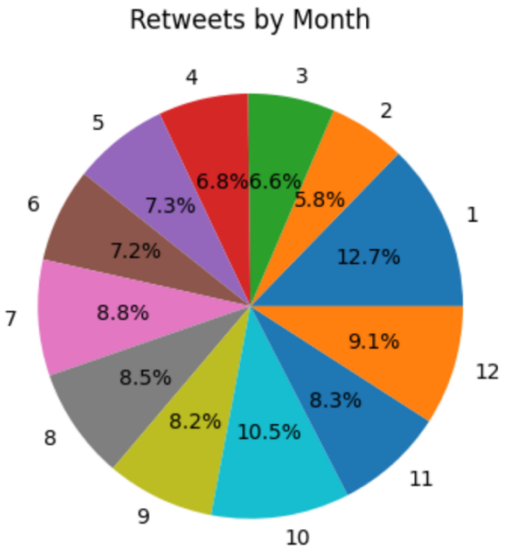
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 31077 entries, 2 to 56565
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   text            31077 non-null  object
 1   likes           31077 non-null  int64
 2   retweets        31077 non-null  int64
 3   date            31077 non-null  datetime64[ns]
 4   isFlagged       31077 non-null  object
 5   Number of Words 31077 non-null  int64
```

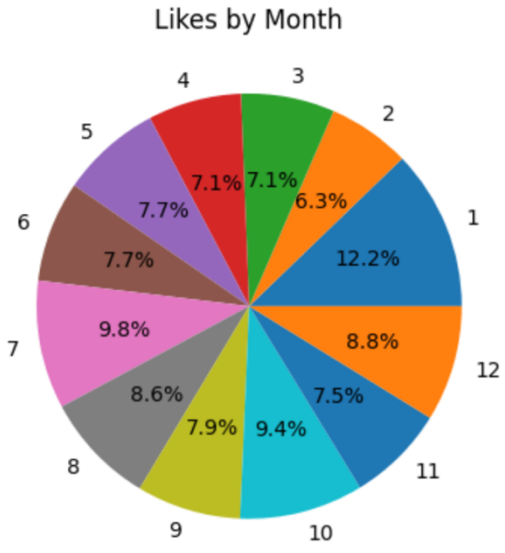
**Visualization**


As seen above, we create a word cloud that represents the frequency of words by the size of text.




In order to get the frequency distribution of tweets, the values are grouped into bins of size 1 week and plotted over time. We observe that the frequency of tweets drops down from the year 2016 to 2020, indicating that President Trump was more active on twitter before and after his presidency.




***Finding Collocations:***

NLTK provides powerful features like getting collocations with simple function calls. Collocations are series or combination of words that are frequently used in the text being analysed. We use Trigrams and Quadgrams to find strings that are frequently used and their frequency in an attempt to understand what words have more impact on the likeability of tweets.

**Trigrams:**

| | | | | |
|---|---|---|---|---|
| ('the', 'United', 'States') 426 | ('I', 'will', 'be') 331 | ('AMERICA', 'GREAT', 'AGAIN') 270 | ('all', 'of', 'the') 237 | ('MAKE', 'AMERICA', 'GREAT') 234 |
| ('the', 'White', 'House') 228 | ('one', 'of', 'the') 224 | ('the', 'Fake', 'News') 202 | ('Fake', 'News', 'Media') 178 | ('run', 'for', 'president') 177 |

**Quadgrams:**

| | | | |
|---|---|---|---|
| ('MAKE', 'AMERICA', 'GREAT', 'AGAIN') 227 | ('Make', 'America', 'Great', 'Again') 147 | ('of', 'the', 'United', 'States') 139 | |
| ('in', 'the', 'history', 'of') 116 | ('the', 'Great', 'State', 'of') 107 | ('the', 'Fake', 'News', 'Media') 105 | |
| ('my', 'great', 'honor', 'to') 86 | ('was', 'my', 'great', 'honor') 84 | ('I', 'will', 'be', 'interviewed') 74 | ('will', 'be', 'interviewed', 'on') 67 |

## Vader Sentiment analysis

Vader is a pre-trained NLTK sentiment analyser that obtains results far quicker than other analyses. It is suitable for analysing text from social media, slangs or short texts but does not do well with larger texts. We choose the top 10 tweets and perform sentiment analysis on them.

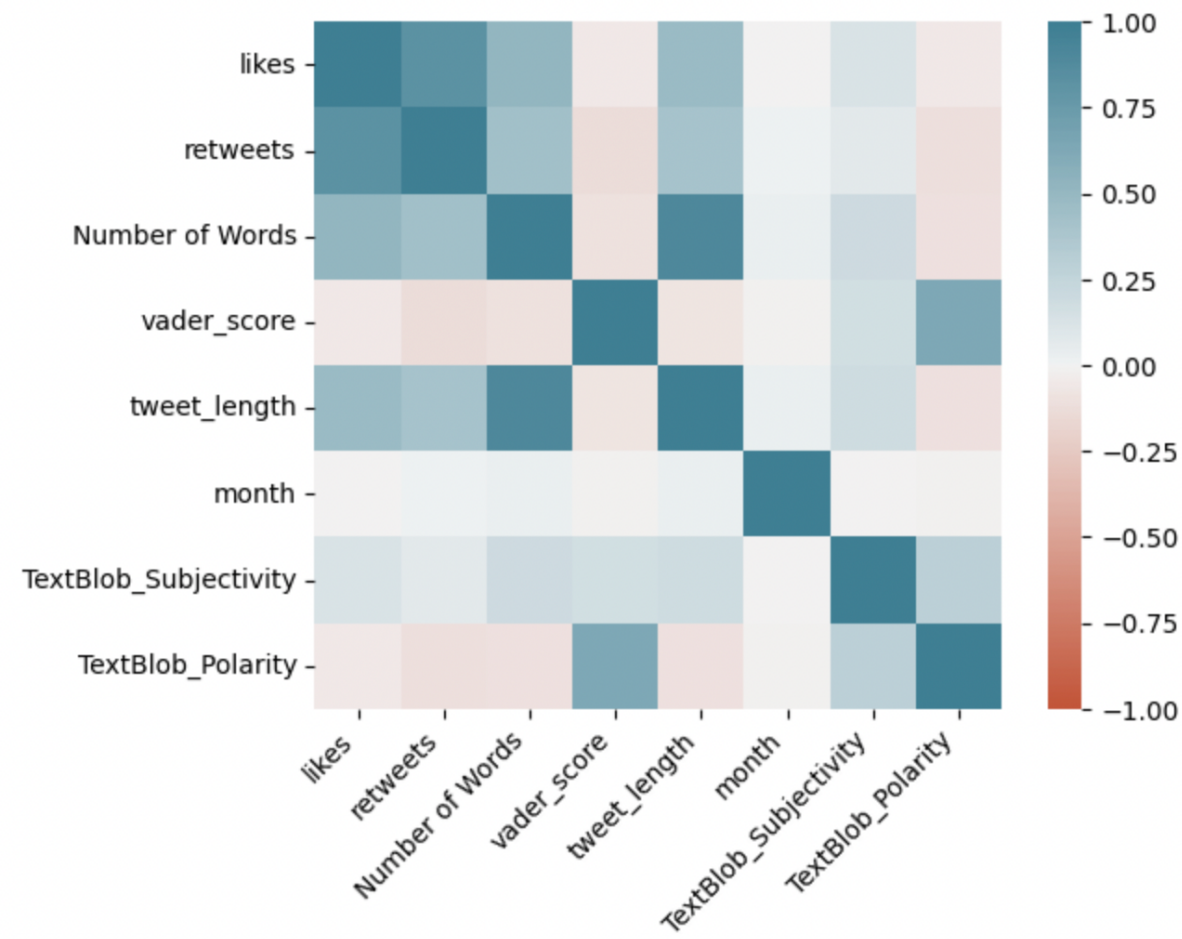
| text | vader_neg | vader_neu | vader_pos | vader_score |
|---|---|---|---|---|
| A$AP Rocky released from prison and on his way... | 0.173 | 0.681 | 0.147 | -0.1280 |
| Kobe Bryant, despite being one of the truly gr... | 0.277 | 0.405 | 0.318 | 0.4215 |
| Just spoke to @KanyeWest about his friend A$AP... | 0.000 | 0.569 | 0.431 | 0.8979 |
| Such a beautiful and important evening! The fo... | 0.230 | 0.424 | 0.345 | 0.4404 |
| "Why would Kim Jong-un insult me by calling me... | 0.196 | 0.583 | 0.221 | 0.1531 |
| We have declassified a picture of the wonderfu... | 0.190 | 0.474 | 0.336 | 0.5267 |
| TODAY WE MAKE AMERICA GREAT AGAIN! | 0.000 | 0.423 | 0.577 | 0.6249 |
| North Korean Leader Kim Jong Un just stated th... | 0.117 | 0.717 | 0.166 | 0.1280 |
| Good Morning, Have A Great Day! | 0.000 | 0.222 | 0.778 | 0.7906 |
| LOOK AT THIS PHOTOGRAPH! https://t.co/QQYTqG4KTt | 0.000 | 1.000 | 0.000 | 0.0000 |

As seen above, Vader sentiment analysis provides the scores for negative, neutral and positive sounding words in each tweet. These values range from 0 to 1 and their sum is always equal to 1. Vader score is a compound score of the tweet and gives the overall sentiment value. Tweets where majority of the text is a link or symbols, the sentiment analyser gives it a neutral score since it cannot determine the emotion behind it. Whereas, words like ‘make America great again’ gives a high positive sentiment and words like ‘despite’, ‘insult’ get a higher negative sentiment score.

## Machine Learning

In order to estimate the amount of likes a tweet might get, we can use regression techniques to accurately fit the algorithms to the data. Since the original independent variables are not enough, we create new features from the tweets to help the ML models to yield better results. Since Vader sentiment results are not enough, we also use TextBlob analysis which provides two values, polarity and subjectivity (whereas vader only gives polarity scores of the tweets. We create four new features namely, ‘tweet_length’, ‘textBlob_subjectivity’, ‘textBlob_polarity’, ‘vader_score’.

Now we find and plot the correlation matrix for all numerical variables to determine the relation between the dependent variable and the independent variables.




The correlation matrix displays pearson's correlation coefficients where values can range from -1 to 1. The closer the value is to 1, higher the chances of a positive correlation whereas the closer the value is to -1, higher the chances of a negative correlation between the two variables. As observed in the above plot, retweets and tweet_length has a positive correlation.

Linear Regression, Random Forrest Regression, Gaussian Naïve Bayes machine learning models are used to predict the amount of likes a tweet gets. Linear Regression was chosen as it is one of the most versatile regression techniques. Random Forrest was chosen since it fits multiple classifying decision trees and uses averaging methods to increase predictive accuracy and prevent over-fitting of data points.

**Results**

| Model | R2 Score | Mean Squared Error |
|---|---|---|
| Random Forest Regression | 0.726182 | 4.950311e+08 |
| Linear Regression | 0.715486 | 5.143680e+08 |
| GNB | 0.109684 | 1.609587e+09 |

As seen above, Random Forest and Linear Regression performed really well, yielding an accuracy of 72.61% and 71.54% respectively. Whereas, Gaussian Naïve Bayes yielded terrible results and hence should not be used for this kind of data as the model relies on normal distributions and high variance may cause over-fitting.

Our analysis reveals that the usage of words like ‘Make America Great again’, ‘the united states’, ‘great state’, ‘the white house’ induces a sense of patriotism in the public whereas words like ‘fake news’, ‘fake media’ causes controversies.

These controversial statements cause the tweets to go viral resulting in more people sharing it and thus getting more likes in comparison to any other tweet. Even though the Trump had a negative effect on twitter users, the viral controversies helped him to imprint his words among the large masses thus winning him the election.

## Conclusion

In conclusion, Businesses can use Twitter text mining to their advantage to learn more about the tastes and opinions of their customers. The data collection, pre-processing, sentiment analysis methods, assessment metrics, and case study are all covered in this report's thorough approach to sentiment analysis. These strategies can be used by businesses to enhance their goods, services, and advertising efforts.

High amounts of noise and spam, difficulty identifying sarcasm and irony, a lack of context, and ethical implications are all problems with sentiment analysis of Twitter data. Future research could concentrate on creating more reliable pre-processing methods to eliminate noise and filter out unimportant data. Additionally, more advanced sentiment analysis methods could recognise irony and sarcasm and change the polarity accordingly. Sentiment analysis methods may be more accurate when using contextual data like user profiles and hashtags. To ensure user privacy and data protection are honoured, ethical standards for the gathering and use of Twitter data are also required.

## Appendix

```
#Import all important libraries
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from wordcloud import WordCloud
from nltk.sentiment import SentimentIntensityAnalyzer
from textblob import TextBlob

from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_squared_error
```

```
import seaborn as sns
import nltk
nltk.download('all')

#Read the dataset
df = pd.read_csv('/content/trump.csv', parse_dates=['date'])
df.columns

#only choose data from year 2014 to 2020
df2 = df.loc[df["date"].between("01-01-2014", "30-01-2020")]
df2["Number of Words"] = df2["text"].apply(lambda n: len(n.split()))
new_df = df2.loc[(df2["Number of Words"] >= 5)]

#store stopwords into variables
stopwords = nltk.corpus.stopwords.words("english")
freq_words = ['https', 'realdonaldtrump', 'trump', 'rt', 'amp', 'http']

tweets = new_df.text
tweets = tweets.astype("string")
tweet_corpus = "".join(tweets)
new_df.info()

#Define a function to carry out all pre-processing tasks
def preprocess_text(text):
  words = nltk.word_tokenize(text)
  words = [w for w in words if w.isalpha()]
  words = [w for w in words if w.lower() not in stopwords]
  words = [x for x in words if x.lower() not in freq_words]
  words = [x.lower() for x in words]
  text = ' '.join(words)
return text


#Defining Vader and TextBlob Sentiment Analysers
sia = SentimentIntensityAnalyzer()

def get_subjectivity(text):
  return TextBlob(text).sentiment.subjectivity
def get_polarity(text):
  return TextBlob(text).sentiment.polarity

#Create new features to be used in ML models
new_df['cleaned_tweets'] = new_df['text'].apply(preprocess_text)
new_df['vader_score'] = new_df['cleaned_tweets'].map(lambda text:
sia.polarity_scores(text)['compound'])
new_df['tweet_length'] = new_df['cleaned_tweets'].apply(lambda x: len(x))
new_df['month'] = new_df['date'].dt.month
```

```
# Add new columns for subjectivity and polarity
new_df['textBlob_Subjectivity'] = new_df['cleaned_tweets'].apply(get_subjectivity)
new_df['textBlob_Polarity'] = new_df['cleaned_tweets'].apply(get_polarity)

#compute vader scores for top 10 tweets with most likes and retweets
vader_sentiment_df = new_df.nlargest(10,['likes','retweets'], keep = 'all')
new_df['vader_score'] = new_df['cleaned_tweets'].map(lambda text:
sia.polarity_scores(text)['compound'])
vader_sentiment_df['vader_neg'] = vader_sentiment_df['cleaned_tweets'].map(lambda
text: sia.polarity_scores(text)['neg'])
vader_sentiment_df['vader_neu'] = vader_sentiment_df['cleaned_tweets'].map(lambda
text: sia.polarity_scores(text)['neu'])
vader_sentiment_df['vader_pos'] = vader_sentiment_df['cleaned_tweets'].map(lambda
text: sia.polarity_scores(text)['pos'])
print(vader_sentiment_df[['text', 'vader_neg', 'vader_neu', 'vader_pos', 'vader_score']])

#tokenize text
words = nltk.word_tokenize(tweet_corpus)
words = [w for w in words if w.isalpha()]
collocation_text = words
words = [w for w in words if w.lower() not in stopwords]
words = [x for x in words if x.lower() not in freq_words]
words = [x.lower() for x in words]

#Create a frquency distribution of words
fd = nltk.FreqDist(words)
#Displaying the top 50 most frequent words
common_words = fd.most_common(50)

#create a word cloud
cloud_words_50 = [i[0] for i in common_words]
cloud_words = ' '.join(cloud_words_50)
word_cloud = WordCloud(collocations = False, background_color =
'white').generate(cloud_words)
# Display the generated Word Cloud
plt.imshow(word_cloud, interpolation='bilinear')
plt.axis("off")
plt.show()

#graph for tweet frequency over time
new_df_tweetFreq = new_df.groupby(pd.Grouper(key='date', freq='1W',
convention='start')).size()
new_df_tweetFreq.plot(figsize=(18,6))
plt.ylabel('Weekly Tweet Count')
plt.title('Tweet Freq. Count (2014 - 2020)')
plt.grid(True)
```

```
new_df_tweetFreq.info()



#graph for tweet frequency over time
likes_by_month = new_df.groupby('month')['likes'].sum()


retweets_by_month = new_df.groupby('month')['retweets'].sum()

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))

# Pie chart for tweets
ax1.pie(retweets_by_month, labels=retweets_by_month.index, autopct='%1.1f%%')
ax1.set_title('Retweets by Month')

# Pie chart for likes
ax2.pie(likes_by_month, labels=likes_by_month.index, autopct='%1.1f%%')
ax2.set_title('Likes by Month')

#Computing Trigrams
finder = nltk.collocations.TrigramCollocationFinder.from_words(collocation_text)
finder.ngram_fd.tabulate(10)

#Computing Quadgrams
finder1 = nltk.collocations.QuadgramCollocationFinder.from_words(collocation_text)
finder1.ngram_fd.tabulate(10)

# Display the correlation matrix
corr = new_df.corr()
ax = sns.heatmap(
  corr,
  vmin=-1, vmax=1, center=0,
  cmap=sns.diverging_palette(20, 220, n=200),
  square=True
)
ax.set_xticklabels(
  ax.get_xticklabels(),
  rotation=45,
  horizontalalignment='right'
);

# Define the independent variables and dependent variable
X = new_df[['retweets', 'tweet_length', 'textBlob_Polarity', 'textBlob_Subjectivity',
'vader_score']]
y = new_df['likes']
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

#Linear regression
lr = LinearRegression()
lr.fit(X_train, y_train)
lr_pred = lr.predict(X_test)

# random forest regression
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
rf_pred = rf.predict(X_test)

#gaussian Naive-Bayes
gnb = GaussianNB()
gnb.fit(X_train, y_train)
gnb_pred = gnb.predict(X_test)



#Display the accuracy results of all models
models = pd.DataFrame({
  'Model': ['Linear Regression', 'Random Forest Regression', 'Gaussian Naive Bayes'],
  'R2 Score': [r2_score(y_test, lr_pred), r2_score(y_test, rf_pred), r2_score(y_test,
gnb_pred)],
  'Mean Squared Error': [mean_squared_error(y_test, lr_pred), mean_squared_error(y_test,
rf_pred), mean_squared_error(y_test, gnb_pred)]
})

models = models.sort_values('R2 Score', ascending=False).set_index('Model')
print(models)
```