Twitter Analysis For COVID-19

Project Report on Task 3

Indian Sentiment analysis after 3 lockdowns using Tweets

BY

Komal Mehta
Dilip Kumar
Stuti Chauhan
Eruvuri Somasekhar

Abstract

These days, individuals from all around the globe utilize web-based social networking locales to share data. Twitter for instance is a stage where clients send, read posts known as 'tweets' and associate with various networks. Clients share their day by day lives, post their sentiments on everything, for example, brands and places. Organizations can profit by this huge stage by gathering information identified with sentiments on them. The point of this paper is to introduce a model that can perform opinion examination of genuine information gathered from Twitter. Information in Twitter is exceptionally unstructured which makes it hard to dissect.

The process of performing sentiment analysis as follows: Tweet extracted directly from Twitter API, then cleaning and discovery of data performed. After that the data were fed into several models for the purpose of training. Each tweet extracted classified based on its sentiment whether it is a positive, negative or neutral. Data was collected on the subject: - Covid 19. We have used supervised learning for training and modeling data. Moreover, our model demonstrates strong performance on mining texts extracted directly from Twitter.
Each tweet extracted was classified based on its sentiment whether it is a positive, negative or neutral. Tweets were collected on following subjects: LockDownIndia, Covid19India , CoronaIndia, quarantine, lockdown, lockdown1, lockdown2, lockdown3, IndiaFightsCorona, india_against_covid19. The Classifier used here is logistic regression. The result from this approach was shown using data visualization methods.

1. Introduction

1.1 Introducing Sentiment Analysis

Otherwise called "Assessment Mining", Sentiment Analysis alludes to the utilization of Natural Language Pre-processing to decide the mentality, conclusions and feelings of a speaker, author, or other subject inside an online notice. Basically, it is the way toward deciding if a bit of composing is certain or negative. This is additionally called the Polarity of the substance. As people, we can arrange text into constructive/antagonistic subliminally. For instance, the sentence "The child had a flawless grin all over", will in all likelihood give us a positive conclusion. In layman's terms, we sort of show up to such end by inspecting the words and averaging out the positives and the negatives. For example, the words "lovely" and "grin" are bound to be certain, while words like "the", "child" and "face" are extremely impartial. Along these lines, the general assessment of the sentence is probably going to be certain. A typical use for this innovation originates from its organization in the web-based social networking space to find how individuals feel about specific themes, especially through clients' promise of-mouth in printed posts, or with regards to Twitter, their tweets. To accept contribution as text based just as voice based.
We have followed the following steps as follows:

1. Data Gathering
2. Data Cleaning and preprocessing
3. Creating labeled Data
4. Training Logistic model using the labelled data
5. Analysis and Visualization

1.1.1 Prerequisites

Basic programming knowledge

Although Python is highly involved in this mini-project, it is not required to have a deep knowledge in the language, as long as you have basic programming knowledge.

Installed Tools

For this program, we will need Python to be installed on the computer. We will be using the libraries twitter, nltk, re, csv, time, and json. You are likely to have to install the first two libraries. The rest already come with the Python interpreter. It doesn’t hurt to check that they’re up-to-date though.

1.1.2 Data set splitting concept

This is critical to fully understand the process pipeline. You only need to know the difference between Training and Test data sets, and in what context each one is used.

1.1.3 Basic Restful API knowledge

This is not crucial, but it could help. We will be using the Twitter API here and there in the code, making normal calls to the API and dealing with the JSON objects it returns. In case you need it, you can find the official Twitter API documentation here.

2. Pre-processing and Cleaning of Data

The pre-processing of the text data is an essential step as it makes the raw text ready for mining, i.e., it becomes easier to extract information from the text and apply machine learning algorithms to it. If we skip this step then there is a higher chance that you are working with noisy and inconsistent data. The objective of this step is to clean noise those are less relevant to find the sentiment of tweets such as punctuation, special characters, numbers, and terms which don't carry much weightage in context to the text.

It includes the following steps as follows:

- Loading required libraries
- Loading Dataset
- Analyses the Data
- Dropping Non-Essential Columns for analysis
- Cleaning the Data
  It includes the following:
  - Dropping Na Records
  - Drop Older records from the specified date
  - Drop Duplicate Records
  - Convert text to small case
  - Remove Numbers, Single Character, punctuation, Special Character, Extra spaces and Removing Stop Words
  - Removing Non-English Words
  - Dropping records with words count less than specified
  - Remove Hyper Links from the tweets
- Tokenization
- Stemming
- Vectorization

1. Loading Libraries

```
import pandas as pd
import numpy as np
import csv
import re #regular expression
from textblob import TextBlob
import string
import preprocessor as p
import nltk
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import warnings
warnings.filterwarnings('ignore')
```

2. Loading Data and Analyse Data

```
df= pd.read_csv('hashtag_data.csv',parse_dates= ['date'],encoding='utf-8-sig')
totallen=len(df)
print("Initial total length of the dataset : ",totallen)
df.head()
```

```
Initial total length of the dataset :  124384
```

	id	conversation_id	created_at	date	time	timezone	user_id	username	name	place	...	geo
0	1262787913311387649	1262787913311387649	1589907074000	2020-05-19	16:51:14	UTC	1250079805980045318	dramaflick	The Drama Flick	NaN	...	NaN
1	1262787786152620040	1262787786152620040	1589907044000	2020-05-19	16:50:44	UTC	807843238648299520	knowpuneet	TravelTrainee	NaN	...	NaN
2	1262787219498000384	1262787219498000384	1589906909000	2020-05-19	16:48:29	UTC	1085426639570235392	narasinhpurohit	Narasinh Purohit	NaN	...	NaN
3	1262786998592434176	1262786998592434176	1589906856000	2020-05-19	16:47:36	UTC	1104213868467806208	ka_trolls	Humans Of Hindutva	NaN	...	NaN
4	1262786970163441669	1262786970163441669	1589906849000	2020-05-19	16:47:29	UTC	392180204	rajendrabohora	rajendrabohora	NaN	...	NaN

5 rows x 34 columns

3. Drop unnecessary Columns from the Data

```
#columns to drops
to_drop=['conversation_id','timezone','name','place','mentions','urls','photos',
         'cashtags','link','retweet','quote_url','video','near','geo','source',
         'user_rt_id','user_rt','retweet_id','reply_to','retweet_date','translate','trans_src',
         'trans_dest']

print("Columns To drop : ",to_drop)
#drop columns
df.drop(to_drop, inplace=True, axis=1)
print("No. of columns Dropped",len(to_drop))
```

```
Columns To drop :  ['conversation_id', 'timezone', 'name', 'place', 'mentions', 'urls', 'photos', 'cashtags', 'link', 'retweet',
'quote_url', 'video', 'near', 'geo', 'source', 'user_rt_id', 'user_rt', 'retweet_id', 'reply_to', 'retweet_date', 'translate',
'trans_src', 'trans_dest']
No. of columns Dropped 23
```

4. Cleaning Data

We clean the data so we get the right sentiments from the sentences.

As discussed, punctuations, numbers and special characters do not help much. It is better to remove them from the text just as we removed the twitter handles. Here we will replace everything except characters and hashtags with spaces.

So, for this we perform the following operations

I. Drop Empty (NA) Records
We dropped all the records with NA and empty values to prevent inconsistency of the dataset so that it doesn't affect the sentiment analysis.

```
#droping na records
df = df.dropna()
df = df.reset_index(drop=True)
print("No. of NA records removed: ",totallen-len(df) )
```

```
No. of NA records removed:  0
```

II. Drop Records older than specified Date (22-3-2020)
We dropped older records because we are doing the analysis of the three months data of lockdown starting from 22nd March till May 30 2020.

```
#drop old tweets older than 3-22-2020
df.drop(df[df['date'] < pd.to_datetime("22-3-2020")].index , inplace=True)
df = df.reset_index(drop=True)
print("No. of Older records removed: ",totallen-len(df) )
```

```
No. of Older records removed:  7445
```

III. Drop Duplicate Record's

Here we dropped the duplicate records to prevent extra processing power or affect our analysis.

```
#droping dulplicates records
df = df.drop_duplicates()
#reset index after dropping
df = df.reset_index(drop=True)
print("duplicates records removed: ",totallen-len(df) )
```

```
duplicates records removed:  8374
```

IV. Convert all text data into Small Case

We converted all the data into same case so the data will not make any inconsistency in our analysis.

```
#convert text to small case
df['tweet'] = df['tweet'].str.lower()
print("Converted to small case")
```

```
Converted to small case
```

V. Remove Numbers

As number doesn't add any specific sentiment meaning to our analysis so we removed all the occurrence of the numbers in all tweets.

```
#removes numbers from text
df['tweet'] = df['tweet'].str.replace('\d+', '')
print("Numbers are removed from text")
```

```
Numbers are removed from text
```

VI. Remove Single Character

Single character doesn't has any sentiment meaning so it will not add up any sentiment meaning to our sentiment analysis so we just removed it from our tweets.

```
#remove single character chracter
df['tweet'] = df['tweet'].replace(re.compile(r"(^| ).( |$)"), "")
print("Single character words are Removed")
```

```
Single character words are Removed
```

VII. Remove Hyperlink's

Hyperlinks are just URL's and doesn't has any sentiment meaning as it is a string of some combination of the specifies format so we removed it using the below regex expression

```
#removes links and urls
df['tweet'] = df['tweet'].replace(re.compile(r'((www\.[\S]+)|(https?://[\S]+))'),"")
print("HyperLinks are removed")
```

```
HyperLinks are removed
```

VIII. Remove Punctuation & Special Character's

Punctuations doesn't have any special meaning in the text or special character also don't have any meaning so we removed the same as using the below regex expression.

```
#removes puntuation
df['tweet'] = df['tweet'].str.replace('[^\w\s]',' ').str.replace('\s\s+', '')
print("Puntuations & Special characters are removed...")
```

```
Puntuations & Special characters are removed...
```

IX. Remove Stop Words

In computing, stop words are words which are filtered out before or after processing of natural language data text. Though "stop words" usually refers to the most common words in a language, there is no single universal list of stop words used by all-natural language processing tools, and indeed not all tools even use such a list.

So we removed all the stops words from all tweets. Using the stop words list.

```
#stops words
stop = stopwords.words('english')

#remove stop words
df['tweet'] = df['tweet'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop)]))
print("Stop words are removed...")
```

```
Stop words are removed...
```

X. Remove Non-English Words

We here are doing sentiment analysis of the English language as the English is the most preferred language on social media so other language words doesn't add any specific sentiment to our analysis so we removed all non-English words

```
#english words
english_word = set(nltk.corpus.words.words())

#remove non english words
df['tweet'] = df['tweet'].apply(lambda x: ' '.join([word for word in x.split() if word in (english_word)]))
print("Non english words are removed")
```

```
Non english words are removed
```

XI. Remove records having word length less than 3

As we found that there are many tweets are of very short length like 1 or 2 words so it will not give any strong sentiment so we removed all such records from our dataset

```
#drops tweets less than 3 words
df.drop(df[df['tweet'].str.count(" ") < 3].index , inplace=True)
#reset index after dropping
df = df.reset_index(drop=True)
print("tweets having words less than 3 words are removed...")
print("word count less than 3 records removed: ",totallen-len(df) )
```

```
tweets having words less than 3 words are removed...
word count less than 3 records removed:  48353
```

5. Tokenize Text

Now we will tokenize all the cleaned tweets in our dataset. Tokens are individual terms or words, and tokenization is the process of splitting a string of text into tokens.

```
tokenized_tweet = df.tweet.apply(lambda x: x.split())
tokenized_tweet.head()
```

```
0    [drama, flick, comes, keep, high, keep, hit, s...
1                              [naam, hi, sab, unlocked]
2    [corona, virus, overcome, stress, make, pic, t...
3                  [fight, best, doctor, degree, good]
4    [bring, attention, stop, audit, happening, man...
Name: tweet, dtype: object
```

6. Stemming

Stemming is a rule-based process of stripping the suffixes (“ing”, “ly”, “es”, “s” etc) from a word. For example, For example – “play”, “player”, “played”, “plays” and “playing” are the different variations of the word – “play”.

```
from nltk.stem.porter import *
stemmer = PorterStemmer()
tokenized_tweet = tokenized_tweet.apply(lambda x: [stemmer.stem(i) for i in x]) # stemming
tokenized_tweet.head()
```

```
0    [drama, flick, come, keep, high, keep, hit, su...
1                              [naam, hi, sab, unlock]
2    [corona, viru, overcom, stress, make, pic, twi...
3                  [fight, best, doctor, degre, good]
4    [bring, attent, stop, audit, happen, mani, rem...
Name: tweet, dtype: object
```

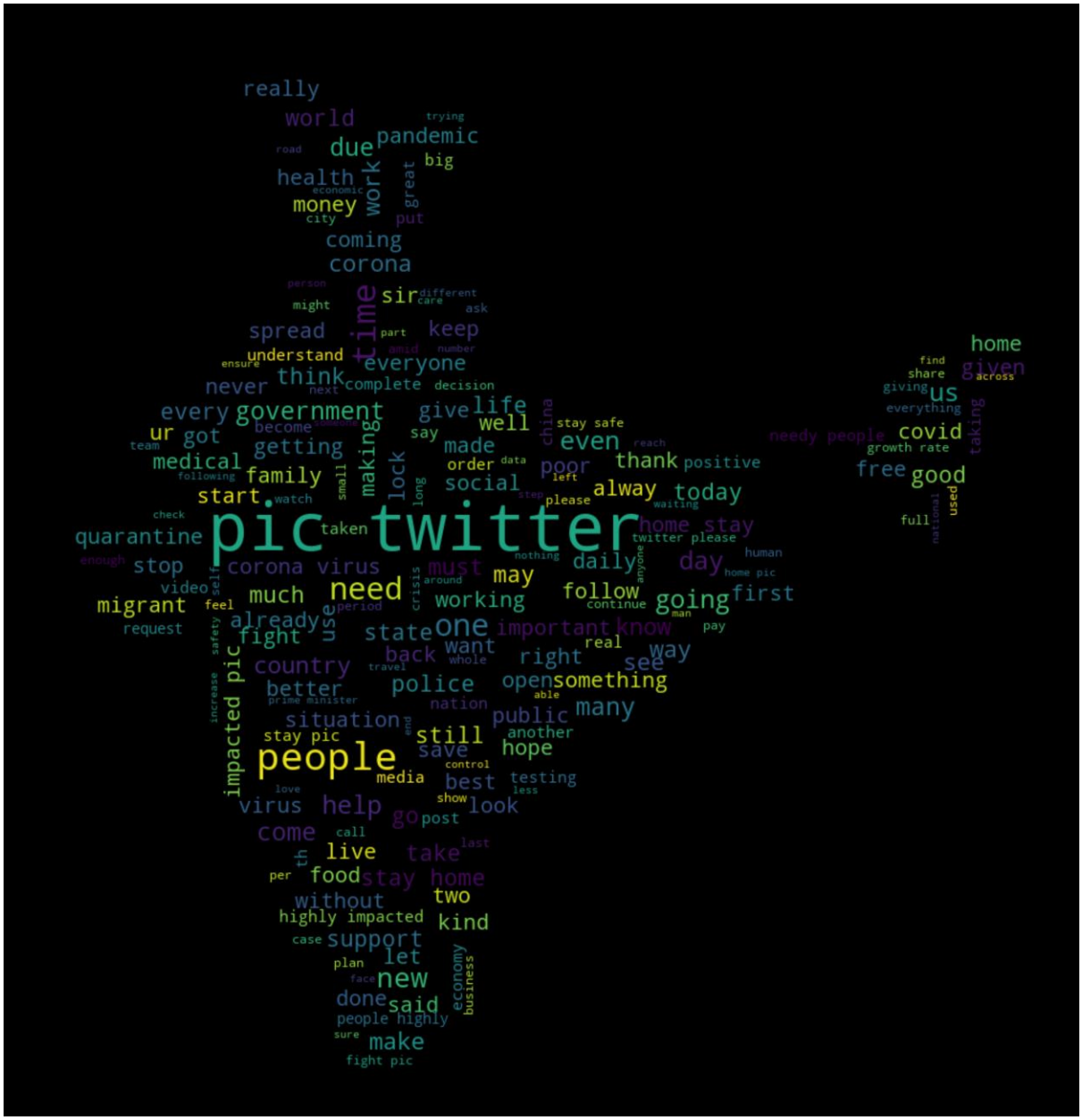
7. Vectorisation

In order for this data to make sense to our machine learning algorithm we’ll need to convert each review to a numeric representation, which we call vectorization.

```
X_train_vectorized = vect.transform(X_train)
```

In order for this data to make sense to our machine learning algorithm we’ll need to convert each review to a numeric representation, which we call vectorization.

8. Visualise the most frequent fords in dataset



9. Save Clean Data

Here we save the clean data into the new csv file which can be used as the input for the sentiment predictions and modelling and training the model.

```
#write clean data to new file
df.to_csv('preprocessed_data.csv', index=False, encoding="utf-8")
print("clean data is saved on preprocessed_data.csv")
```

```
clean data is saved on preprocessed_data.csv
```

3. Training and Prediction of Sentiments of Tweets using

Logistic Regression

Logistic regression is a statistical method for predicting binary classes. The outcome or target variable is dichotomous in nature. Dichotomous means there are only two possible classes. For example, it can be used for cancer detection problems. It computes the probability of an event occurrence.

It is a special case of linear regression where the target variable is categorical in nature. It uses a log of odds as the dependent variable. Logistic Regression predicts the probability of occurrence of a binary event utilizing a logit function.

Logistic Regression is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable. In logistic regression, the dependent variable is a binary variable that contains data coded as 1 (yes, success, etc.) or 0 (no, failure, etc.). In other words, the logistic regression model predicts P(Y=1) as a function of X. Logistic regression is fast and relatively uncomplicated, and it's convenient for you to interpret the results. Although it's essentially a method for binary classification, it can also be applied to multiclass problems.

Linear Regression Equation:

$$y = \beta 0 + \beta 1X1 + \beta 2X2 + \ldots\ldots + \beta nXn$$

Where, y is dependent variable and x1, x2 ... and Xn are explanatory variables.

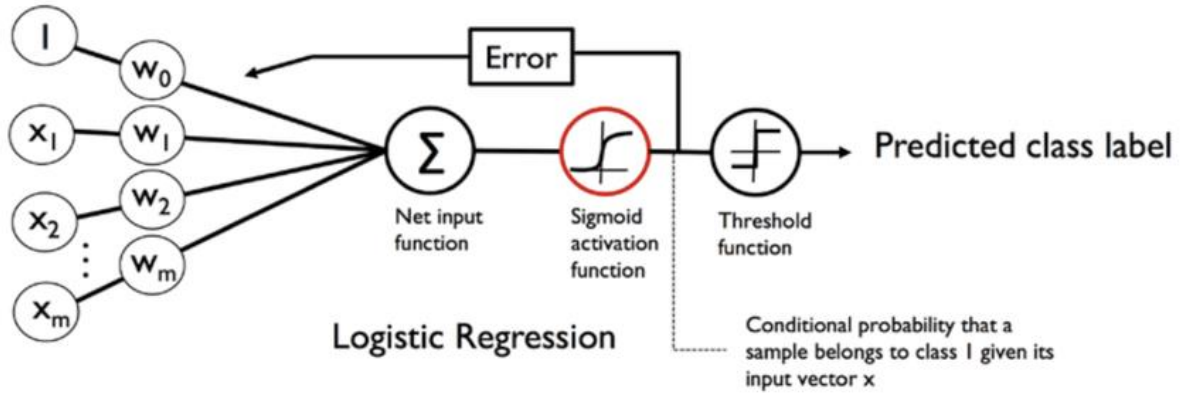
Sigmoid Function:

$$p = 1/1 + e^{-y}$$

Apply Sigmoid function on linear regression:

$$p = 1/1 + e^{-(\beta 0 + \beta 1X1 + \beta 2X2 \ldots\ldots \beta nXn)}$$

Simple Architecture of logistics model



Properties of Logistic Regression:

- The dependent variable in logistic regression follows Bernoulli Distribution.
- Estimation is done through maximum likelihood.
- No R Square, Model fitness is calculated through Concordance, KS-Statistics.

We have done the following steps for the Training the model and for predicting the sentiment through the logistic regression model.
The steps are as follows:

Load Data:

```
#Loading data
df = pd.read_csv('sentiment.csv')
print("Columns in the original dataset:\n")
df.columns
```

```
Columns in the original dataset:


Index(['id', 'created_at', 'date', 'time', 'user_id', 'username', 'tweet',
       'replies_count', 'retweets_count', 'likes_count', 'hashtags',
       'p_score'],
      dtype='object')
```

Step 1. Import the model

In Sklearn, all machine learning models are implemented as Python classes

```
from sklearn.linear_model import LogisticRegression

X = df['tweet']
y = df['p_score']
one_hot_encoded_label = pd.get_dummies(y)
print(one_hot_encoded_label.head())
```

```
   -1   0   1
0   1   0   0
1   0   1   0
2   1   0   0
3   0   0   1
4   1   0   0
```

Step 2. Make an instance of the Model

Here we created an instance of the model loaded from the sklearn library.
And created the model instance as follows.

model = LogisticRegression(C=c,solver='saga')

Step 3: Splitting the Training data and the testing data

Here we split the data into training data and the testing data. We split data as 75% for training and 25% for testing.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state=42)

print("percentage of the X_train data:{:0.2f}%".format(len(X_train)/len(df)*100))
print("percentage of the X_test data:{:0.2f}%".format(len(X_test)/len(df)*100))
print("percentage of the y_train data:{:0.2f}%".format(len(y_train)/len(df)*100))
print("percentage of the y_test data:{:0.2f}%".format(len(y_test)/len(df)*100))
```

```
percentage of the X_train data:75.00%
percentage of the X_test data:25.00%
percentage of the y_train data:75.00%
percentage of the y_test data:25.00%
```

Step 4. Extracting Features

In this step we extract the all features from the tweets. And convert the data into vector form.

```
from sklearn.feature_extraction.text import TfidfVectorizer

vect = TfidfVectorizer(min_df=5).fit(X_train)
print("Features Length ", len(vect.get_feature_names()))
X_train_vectorized = vect.transform(X_train)
feature_names = np.array(vect.get_feature_names())
sorted_tfidf_index = X_train_vectorized.max(0).toarray()[0].argsort()

print('Smallest tfidf:\n{}\n'.format(feature_names[sorted_tfidf_index[:10]]))
print('Largest tfidf: \n{}'.format(feature_names[sorted_tfidf_index[:-11:-1]]))

X_train_vectorized = vect.transform(X_train)
```

```
Features Length  5204
Smallest tfidf:
['plentiful' 'toxin' 'tortured' 'governmental' 'signature' 'bias'
 'volatility' 'infant' 'criminally' 'sequential']

Largest tfidf:
['effective' 'quarantine' 'fighting' 'set' 'environment' 'dumb' 'breath'
 'locked' 'train' 'national']
```

Step 4. Train the Model

Training the model on the data, storing the information learned from the data
Model is learning the relationship between (x_train) and labels (y_train)

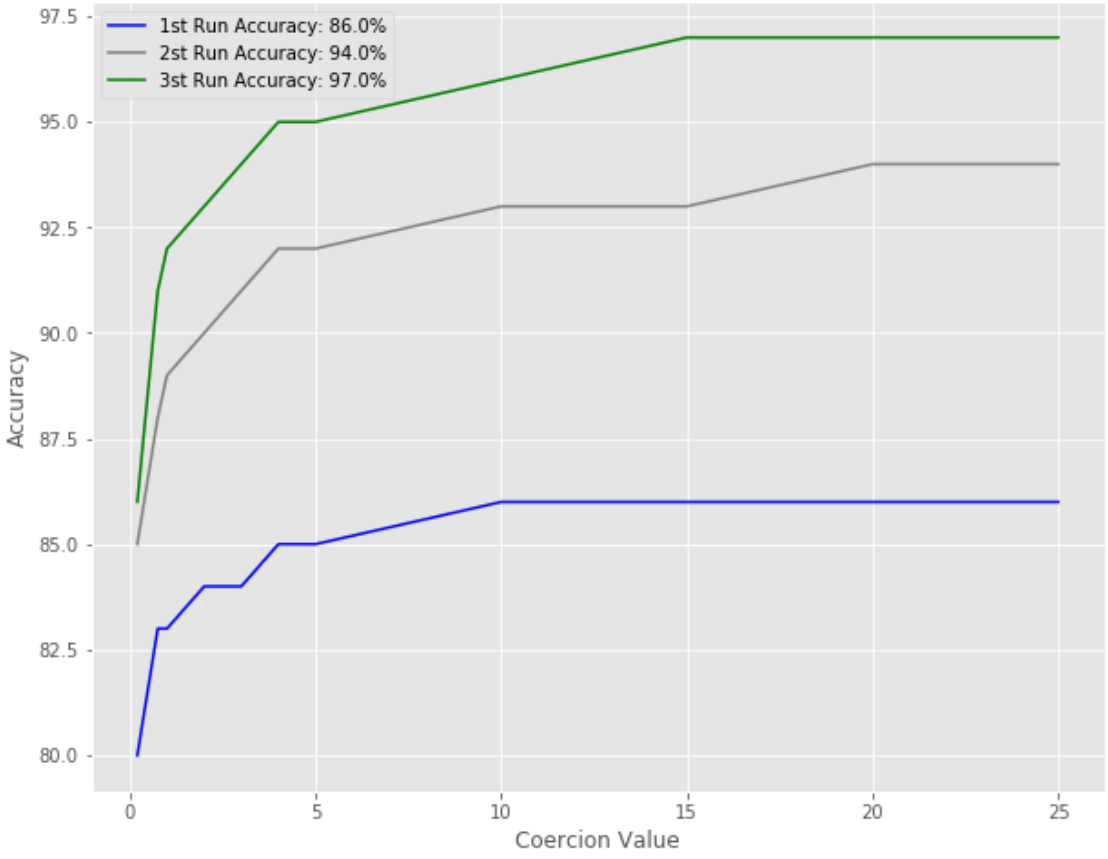
```
c_val =[0.2,0.75, 1, 2, 3, 4, 5, 10,15,20,25]
model = LogisticRegression(C=c,solver='saga')
model.fit(X_train_vectorized, y_train)
```

we have run this model three time and achieved good accuracy from thus model.

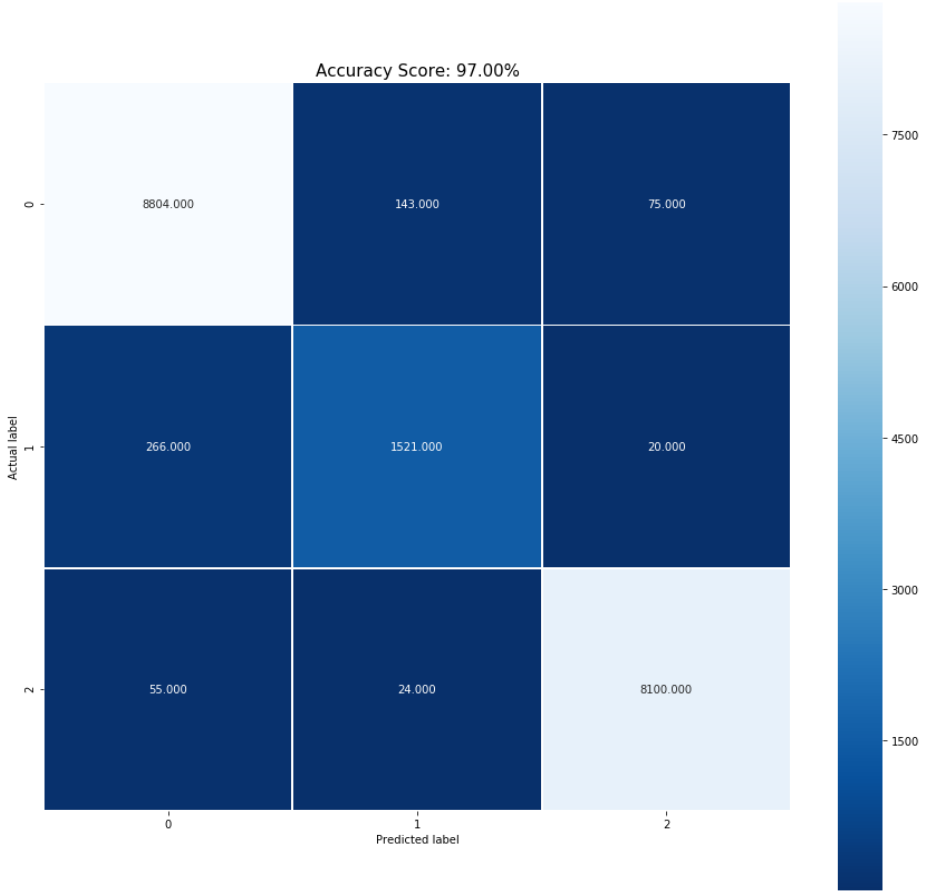
Step 4. Model Accuracy vs coercion plot and confusion matrix

Here we plot the line curve between the accuracy vs coercion value and also generate the confusion matrix.

1. Accuracy vs coercion plot



2. Confusion matrix



3. Precision matrix

	precision	recall	f1-score	support
-1	0.96	0.98	0.97	9022
0	0.90	0.84	0.87	1807
1	0.99	0.99	0.99	8179
accuracy			0.97	19008
macro avg	0.95	0.94	0.94	19008
weighted avg	0.97	0.97	0.97	19008

4. Analysis & Visualization

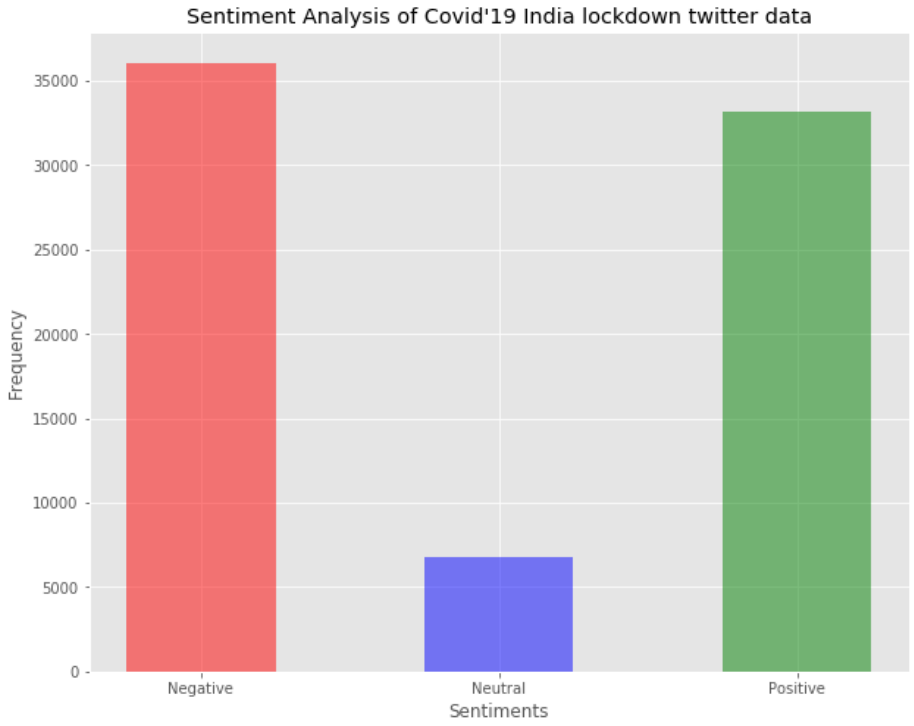
4.1 Sentiment Statistics

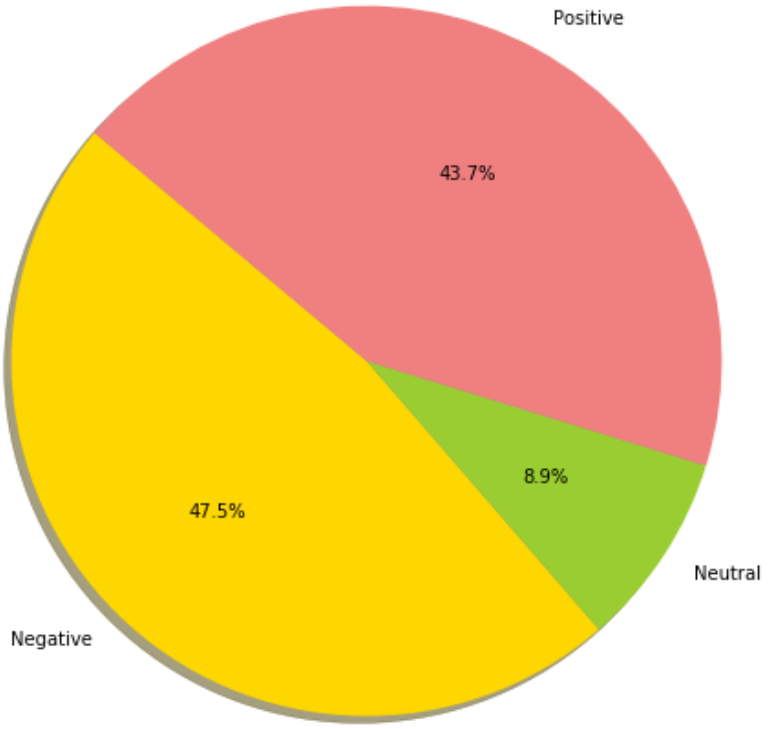
After prediction through regression model we found the following statistics of the tweet sentiments

Sentiments Class	Percentage
Positive	43.65%
Negative	47.45%
Neutral	8.88%

4.2 Sentiments Bar Plots and Pie chart

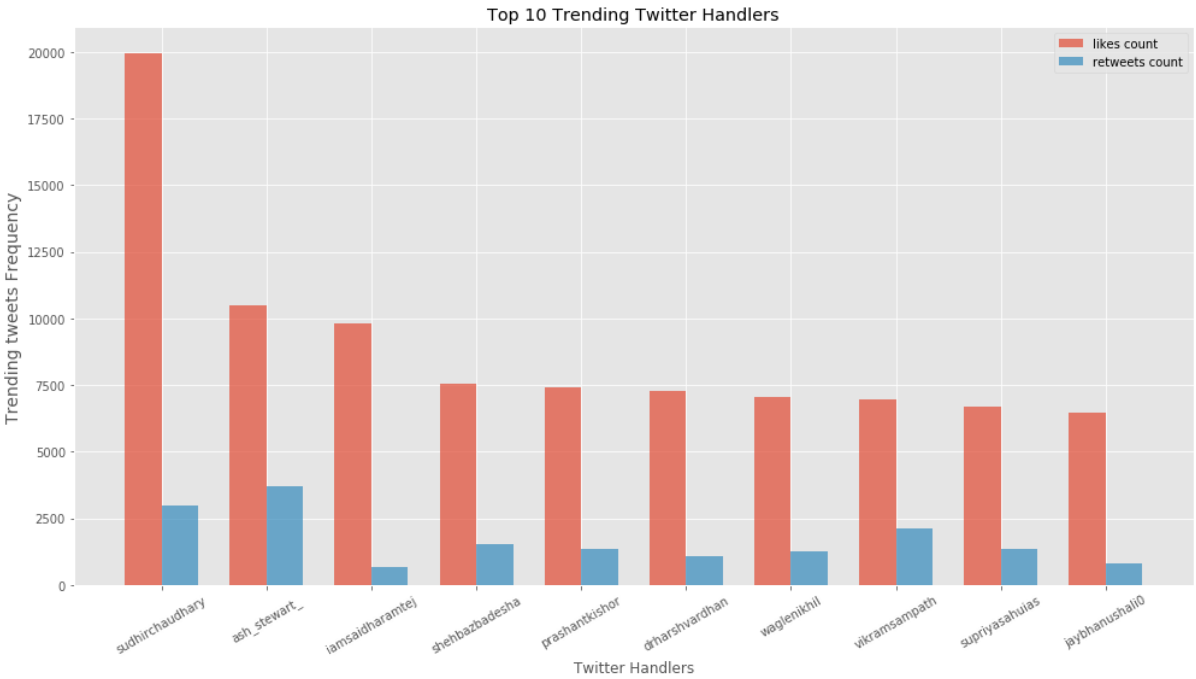
We plotted the bar plot and pie chart for better visualization of the above statistics





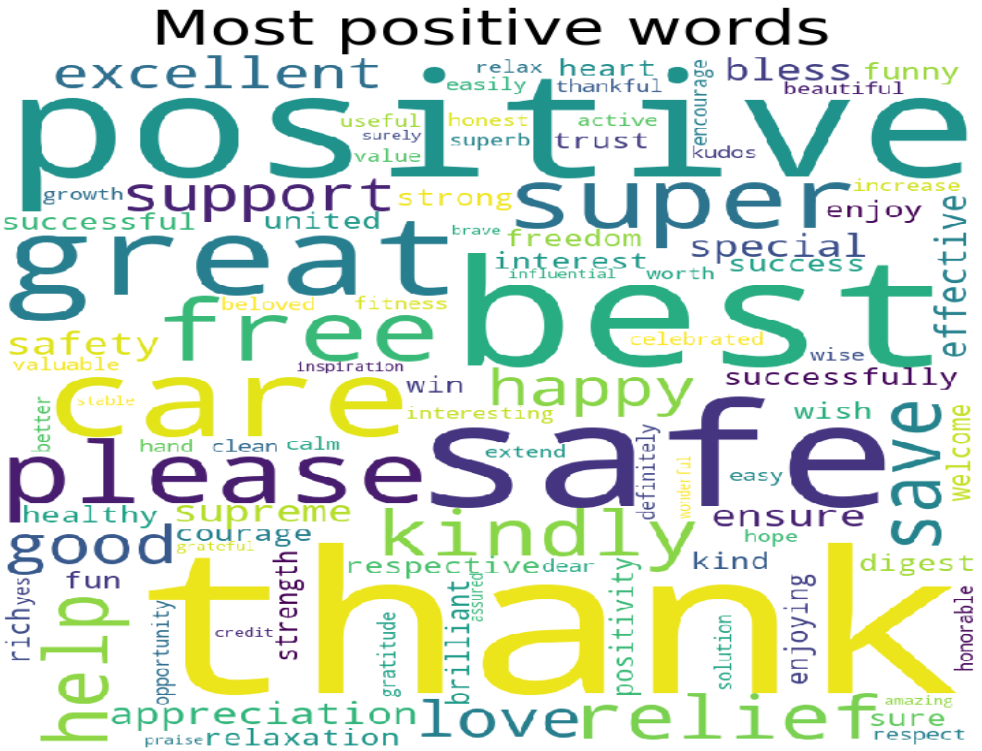
4.3 Top 10 trending twitter handlers

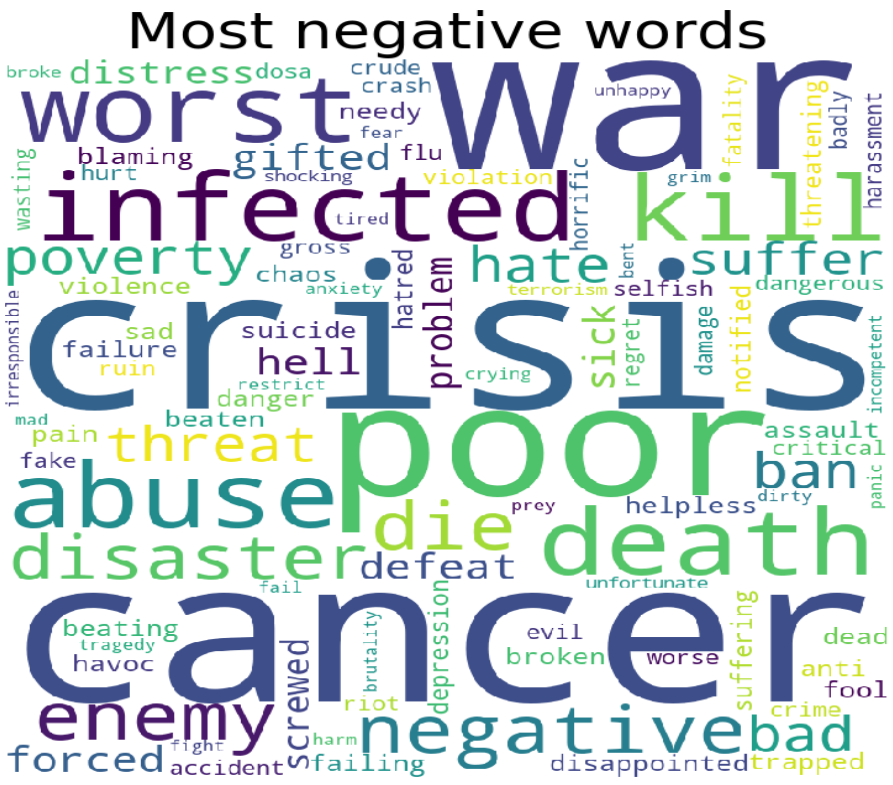
We sorted the top 10 twitter handlers based on likes counts and retweet counts from the dataset and we plot the top 10 twitter handlers on the basis of above parameters.



4.4 Word Cloud for Positive and Negative tweets words

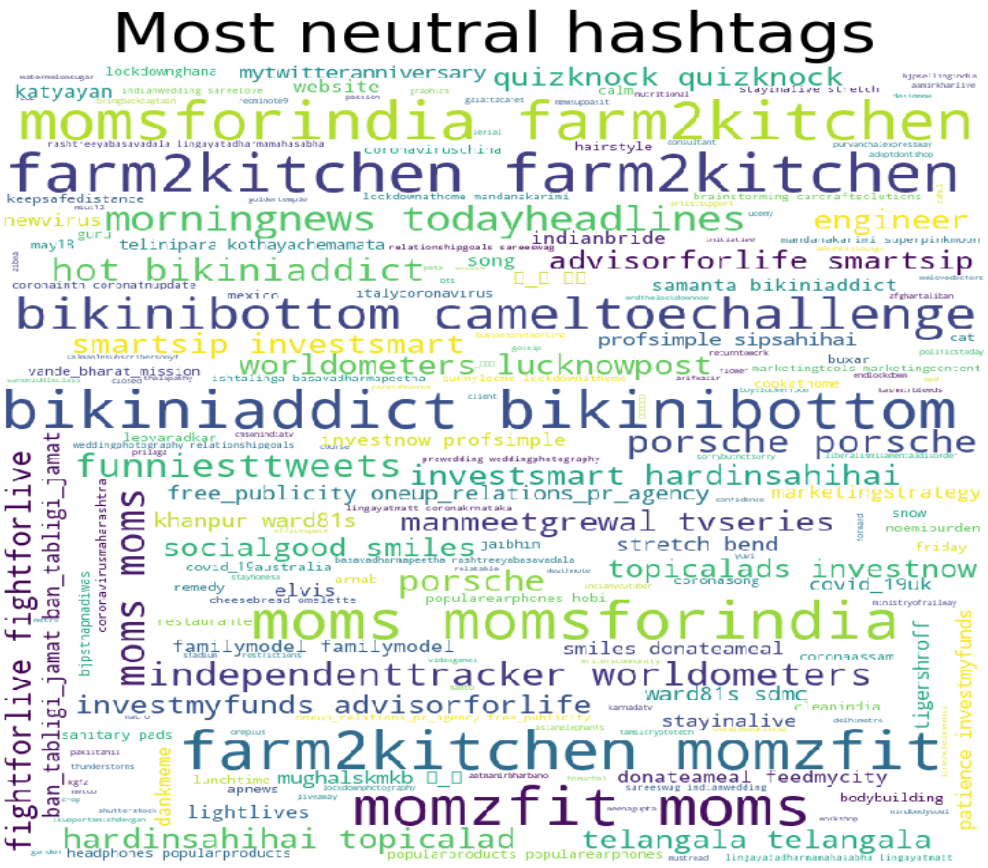
We found the positive and negative words through the extraction of the features through the vectorization and then plot the word cloud for the same.





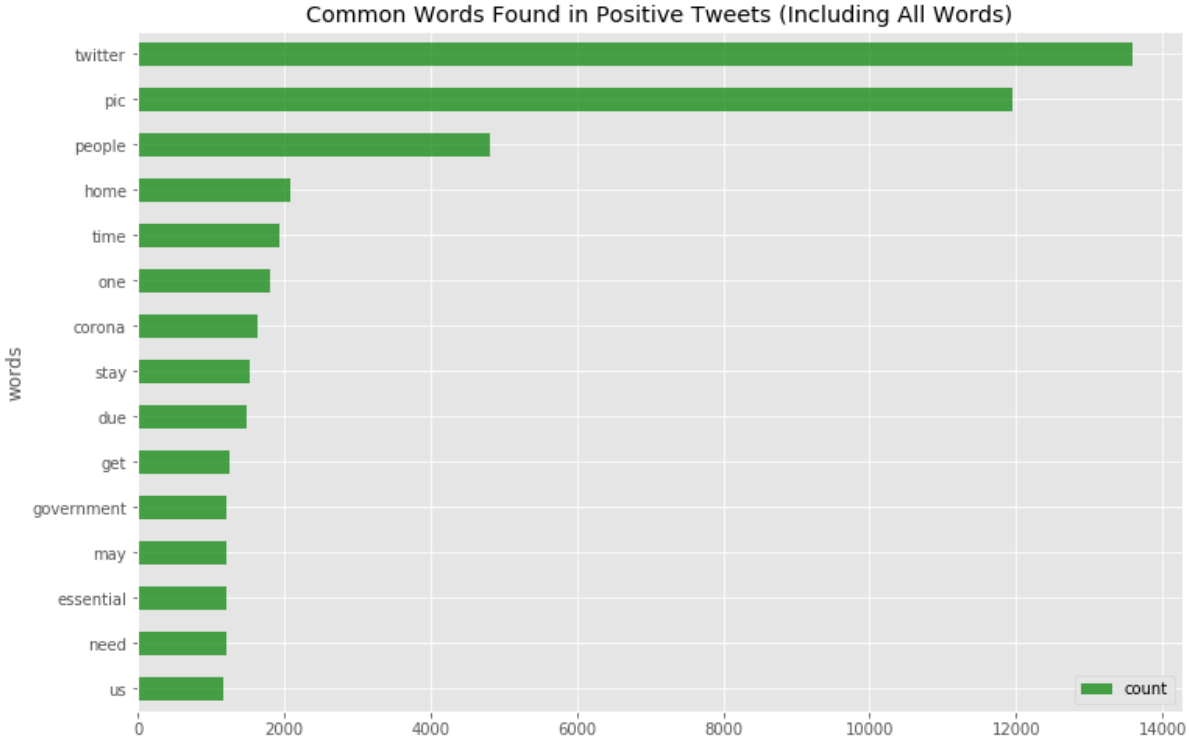
4.5 Word Cloud for positive, negative and neutral hashtags according to tweets sentiments

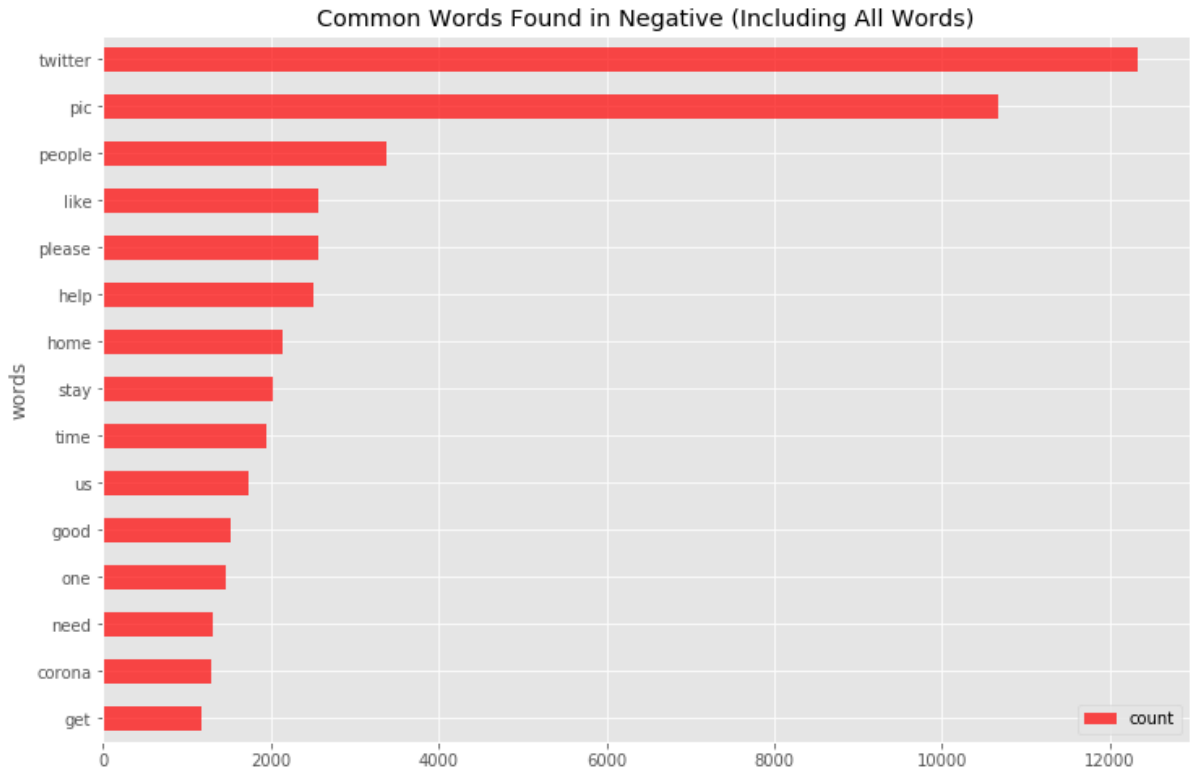
On the basis of twitter sentiments we extracted the different hashtags of different sentiment classes that is positive, negative and neutral and we plotted word cloud for the same.

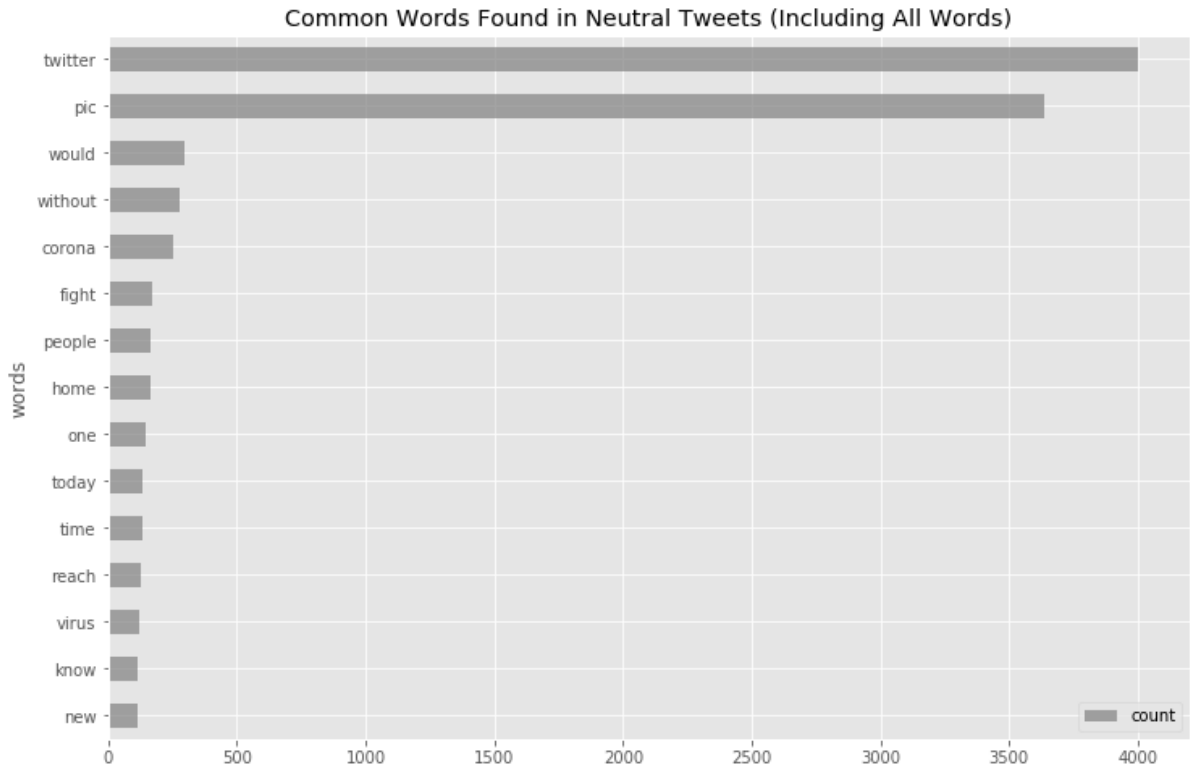


4.6 Most common positive, negative and neutral words in tweets

On the basis of twitter sentiments, we extracted the different words of different sentiment classes that is positive, negative and neutral and we plotted horizontal bar plot for the same.

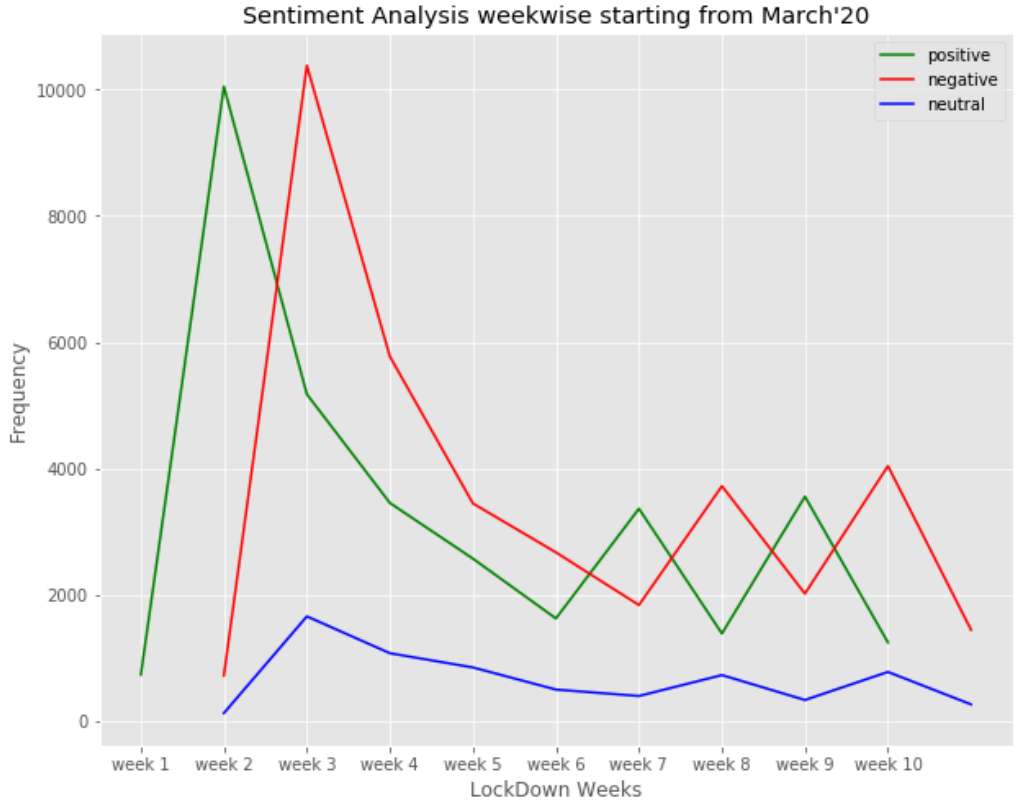


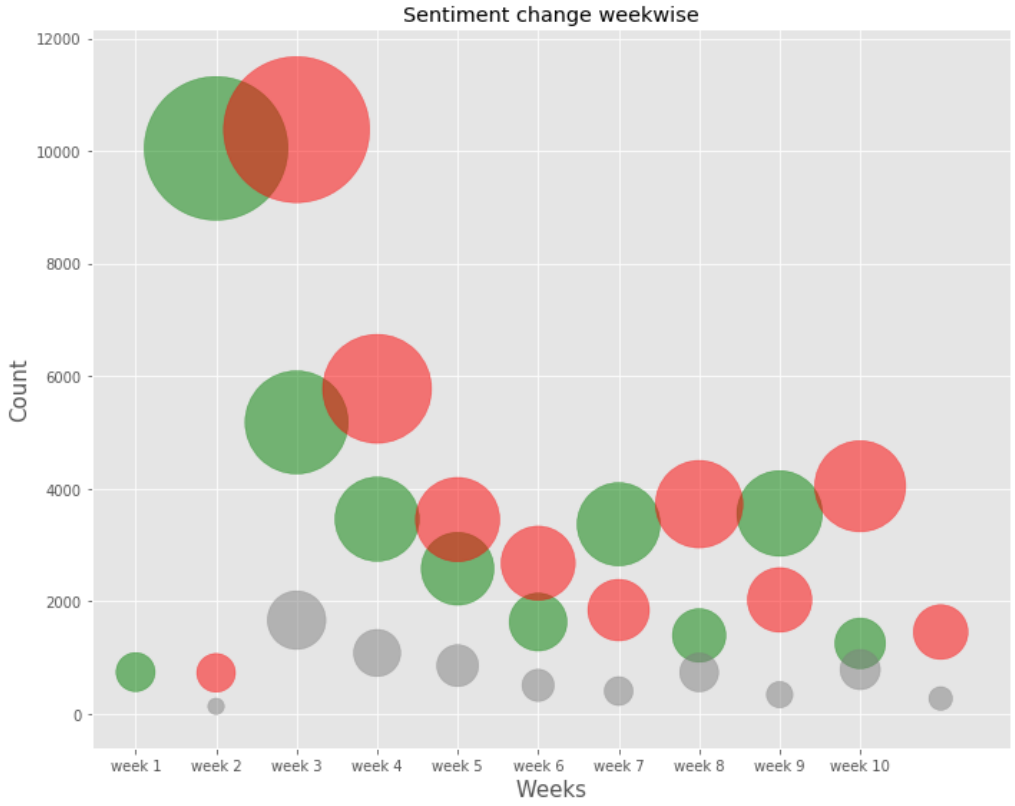

Common Words Found in Negative (Including All Words)
twitter
pic
people
like
please
help
home
stay
time
us
good
one
need
corona
get
words
count
0
2000
4000
6000
8000
10000
12000


Common Words Found in Neutral Tweets (Including All Words)
twitter
pic
would
without
corona
fight
people
home
one
today
time
reach
virus
know
new
words
count
0
500
1000
1500
2000
2500
3000
3500
4000

4.7 Week wise sentiment analysis

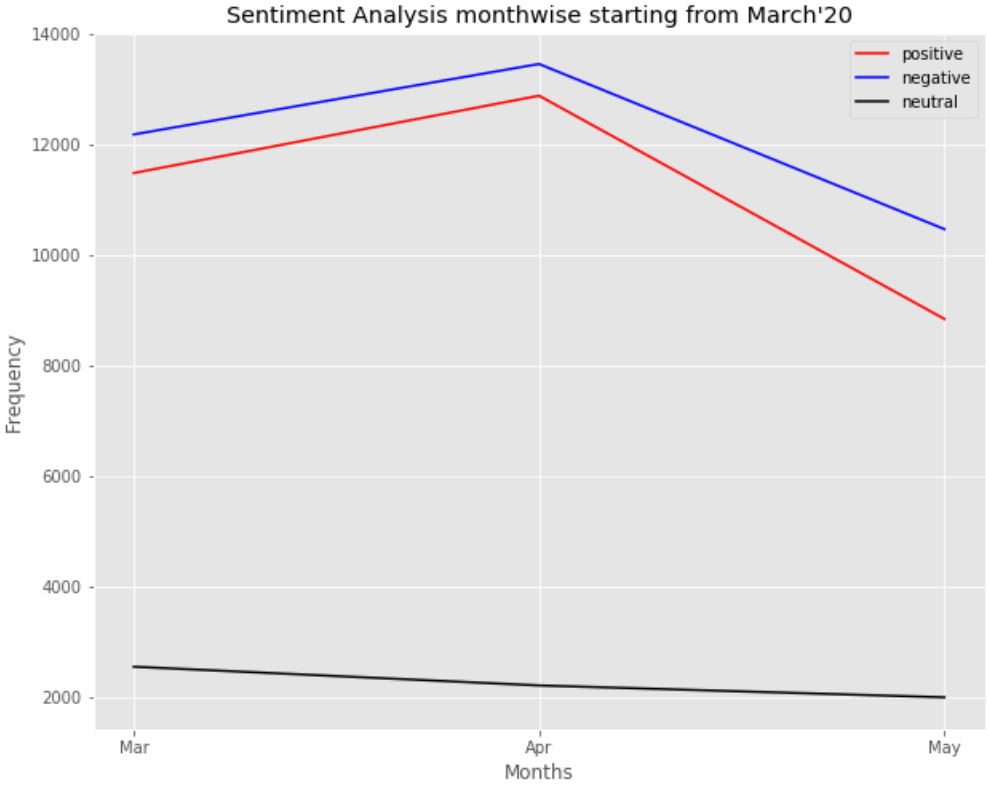
We grouped the data of different sentiment classes week wise and plotted the line chart for the same.

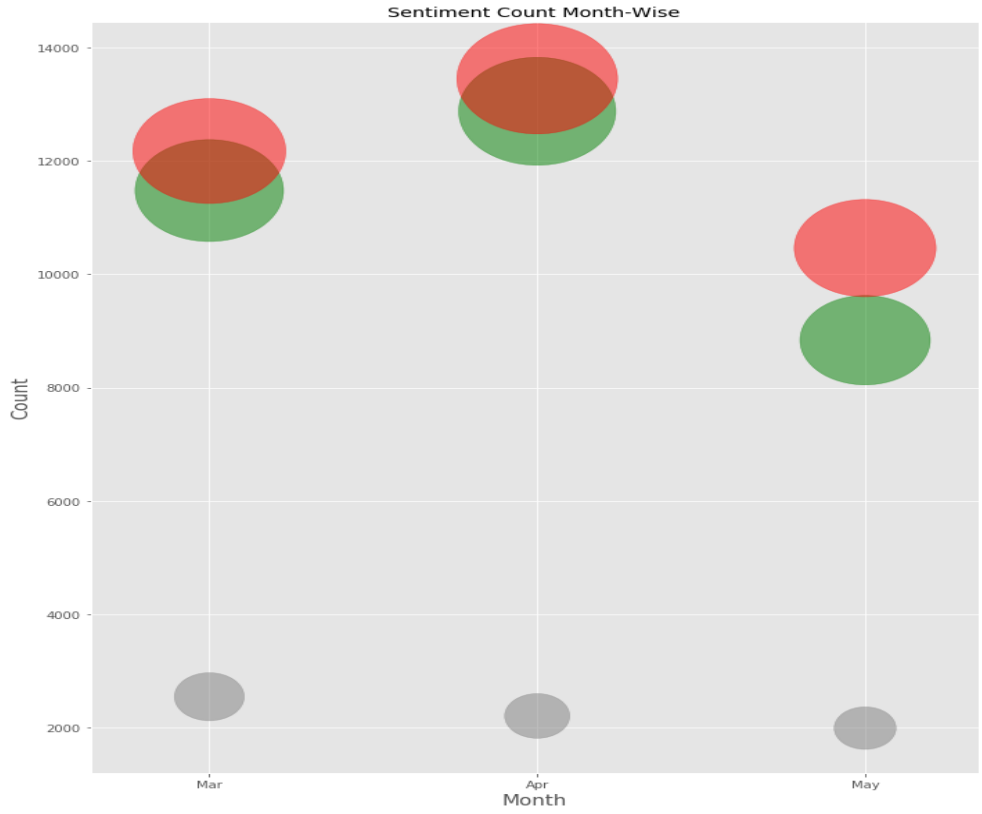




4.8 Month-wise sentiment analysis

We grouped the data of different sentiment classes month wise and plotted the line chart for the same.





5. Conclusion

Sentiment analysis of the tweets determine the polarity and inclination of vast population towards specific topic, item or entity. Sentiment Analysis or Opinion Mining is the computational treatment of opinions, sentiments and subjectivity of text. Sentiment analysis, a branch of digital analytics aims to determine the attitude of a speaker or a writer with respect to some topic or the overall contextual polarity of a document.

We have used the Logistics regression for the sentimental analysis and for prediction of the sentiments of the tweets. We have achieved the accuracy of 86% on first run of the model and at second run it is 94% and the accuracy achieved is 97% after third run. After doing analysis we have found that approx. 47.45% tweets sentiments are negative and approx. 43.65 % are positive and approx. 8.88% are neutral. So, we can thus conclude that the people are doing negative tweets due to lockdown in India.