OPINION MINING ON TWITTER'S TRENDING TOPICS

VISHNU KARTHIK RAVINDRAN – 985286667

PRATIK PATIL – 262088929

VISHL APTHARE – 375758029

Abstract

This document contains information about the Opinion Mining on Twitter's trending topics. This project was created mainly for the Principles of Social Media and Data Mining course. All code, presentation and slides were created by Vishnu Karthik Ravindran, Pratik Patil and Vishal Pathare. The content in this document includes the following.

- Application Overview
- Runtime Instructions
- User Interface
- Technical Components

We have also included the following in the appendix section

- Data for sentiment analysis
- Data for Category analysis
- List of helper functions
- Source code for Opinion mining on twitter's trending topics

Contents

Chapter 1 - Application Overview

1.1 Overview:

In recent years, online social media platforms are generating timely and rich information related to all kinds of real-world events around the world. One of such social media platform is Twitter. It plays an important role in providing people, the trending topics and it also provides a platform for people to share their valuable opinion and express sentiments on trending topics related to events, news, products, sports etc. This project is based on delivering the sentiment analysis for real time trending topics wherein we will categorize each trending topic into prespecified categories using Naïve Bayes model, then perform supervised learning for analyzing sentiments of the tweets gathered using twitter APIs, and finally visualizing the outputs using an interactive GUI made using Bokeh Web Server like fig. 1.

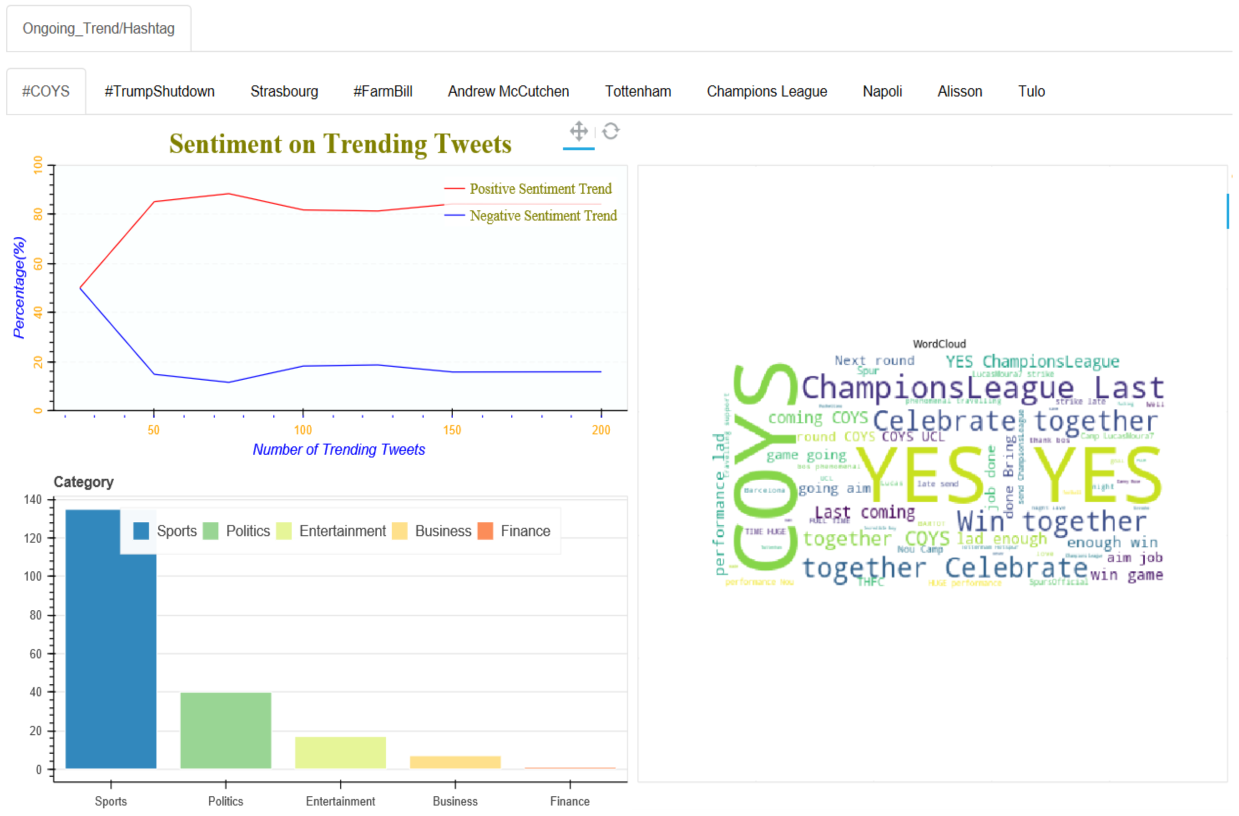


Fig. 1- Visualization of the Application – Opinion mining on trending topics

1.2 Limitations

1.2.1 Twitter API

Twitter API has rate limit according to its official website. “Rate limiting of the standard API is primarily on a per-user basis — or more accurately described, per user access token. If a method allows for 15 requests per rate limit window, then it allows 15 requests per window per access token.” – from official site. Hence, we are using only 200 tweets per trending topics.

Chapter 2 – Runtime Instructions

2.1 Configuration

2.1.1 API accounts

To use this application, the user should have a developer account for Twitter API. Registering with these services will give users the ability to access Twitter APIs'.

2.1.2 Authentication File

Once accounts are setup with the two API services, the keys and tokens will need to be stored in JSON format in a file called twitter_credentials.py. This file is used for retrieving the credentials later.

2.1.3 Supporting Data

Lastly, the data subdirectory must be included with the python code. This folder, included in the submission, contains data for the all the analysis we will be doing for this project.

2.2 Running Code

Open screen_generator.py file in Spyder from Project folder. Set Spyder path to the project location.
Run screen_generator.py file in Spyder. After successful execution, static dashboard will be generated showing all plots, in same folder with output.html file. Open WEB_OUTPUT.html from the project folder to see the results.

Chapter 3 - Visualizations

In this project, our output will be sentiment, category and frequent word related to each category. For visualization, we have created static-interactive dashboard using bokeh library in python. The sentiment plot is multiple line plot, categorical data has been plotted using histogram, and for top words we used word-cloud. We will discuss more detail about each plot further in this section.

3.1.1 Dashboard Workflow

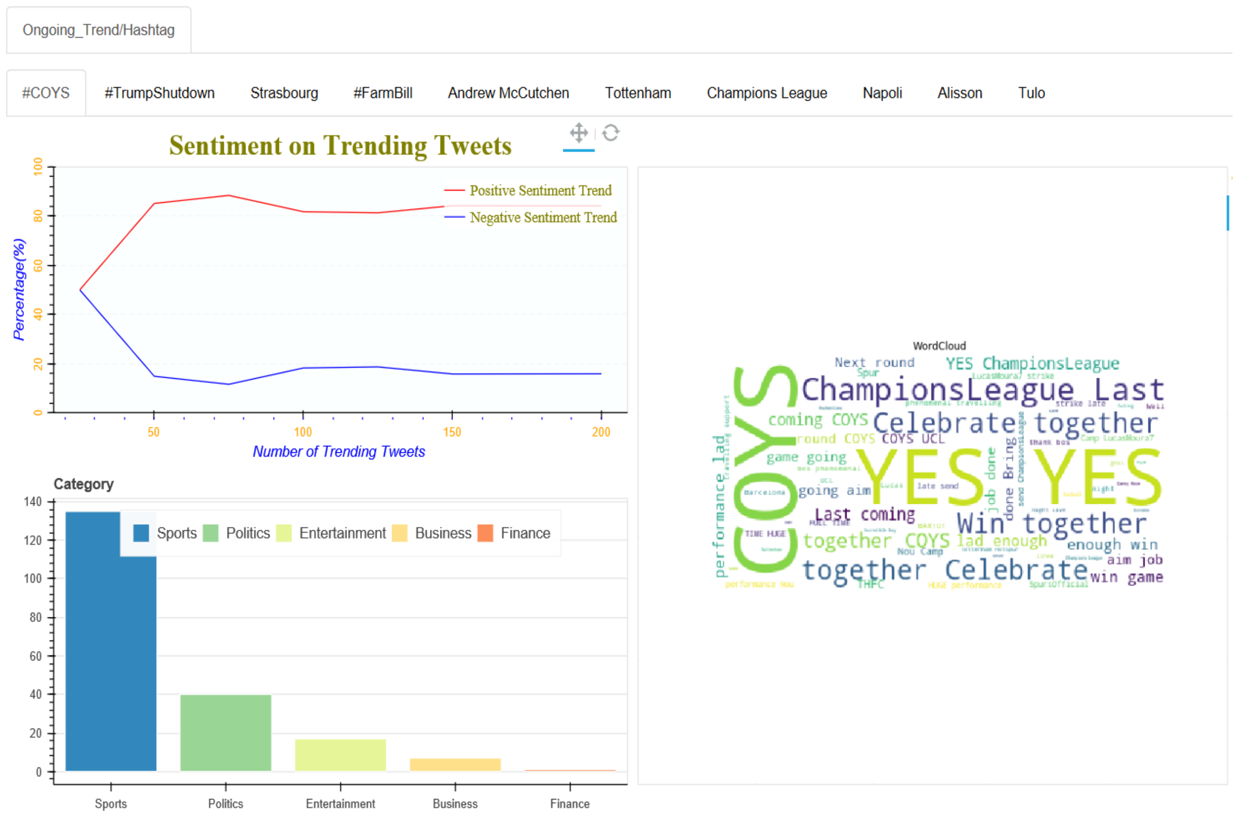


Fig. 2- Visualization of the Application

In above figure 1.1 top tabs will be the current trending topics of twitter (Fig 1.1 based on data dated 11-dec). and each tab will show plots related to it. As you move along the tabs you can see plot related to that trending topic. To create static Dashboard, we used object of following libraries: bokeh.layouts, bokeh.models, bokeh.plotting, bokeh.palettes, matplotlib. Initially we created sentiment plot (Multiline Plot), Bar Plot for categories, and word-cloud for Top words. Initially, we created Tab named Ongoing Trend/Hashtag, Inside Ongoing Trend/Hashtag we wrote dynamic code which will create a new tab for each ongoing trend and will append all the tabs into one panel. To arrange sentiment plot, Categorical Plot and word-cloud into proper layout we used row and column instances of bokeh.Layout. First, we arranged sentiment plot and category plot into row format and Wordcloud we arranged in separate column format and then combined both row and column. There is no direct way to plot word-cloud into bokeh, so to solve this problem we combined bokeh and matplotlib. We plot word-cloud using matplotlib. pyplot, saved that plot into image and shown that image in bokeh simple glyph. X-axis co-ordinate and Y-axis co-ordinate for that image has been manually adjusted based on dashboard scale.

Further in this section of visualization we will discuss each plot in detail.

3.1.2 Sentiment Plot

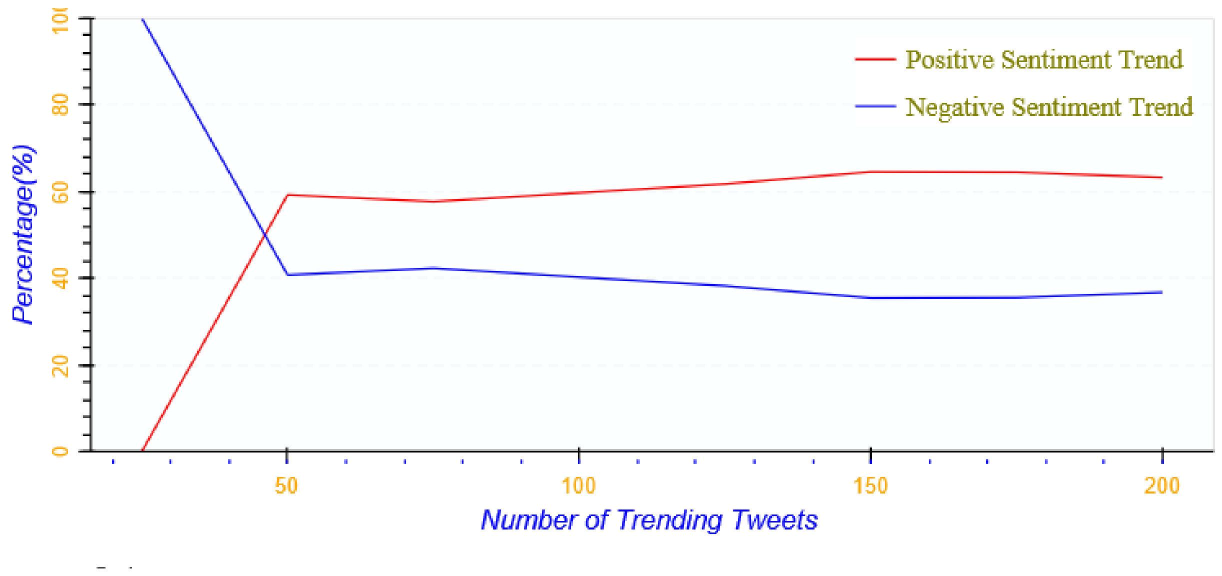


Fig. 3- Sentiment Analysis

The given sentiment plot has been plotted using multiline glyph and it is based on last 200 tweets. The reason we opt for multiline plot is it's easy to figure out how trend is changing over the period. So, x axis will range from 0-200 i.e. tweet count and y axis show percentage of positive and negative tweets, so y-axis will range from 0-100. To create this plot, initially we sorted the tweets based on time and date and select only top 200 i.e. latest tweets. The red line in a plot will say the positive sentiment behavior over the period (as number of tweets increases) and negative line in a plot will say negative sentiment trend over the period.

3.1.3 Category Plot

Here, our aim is to show that in which category a given trend belongs. Moreover, it can be possible that, a2 person/Trending topics may belongs to multiple category. So, to show that in what categories a trend belongs, we plotted bar plot. In the given bar plot x-axis will represent categories, in total we have six default categories {Entertainment, Business, Politics, Sports, Technology and Finance}. The y axis will show number of tweets related to each category. Moreover, we are plotting category plot based on latest 200 tweets

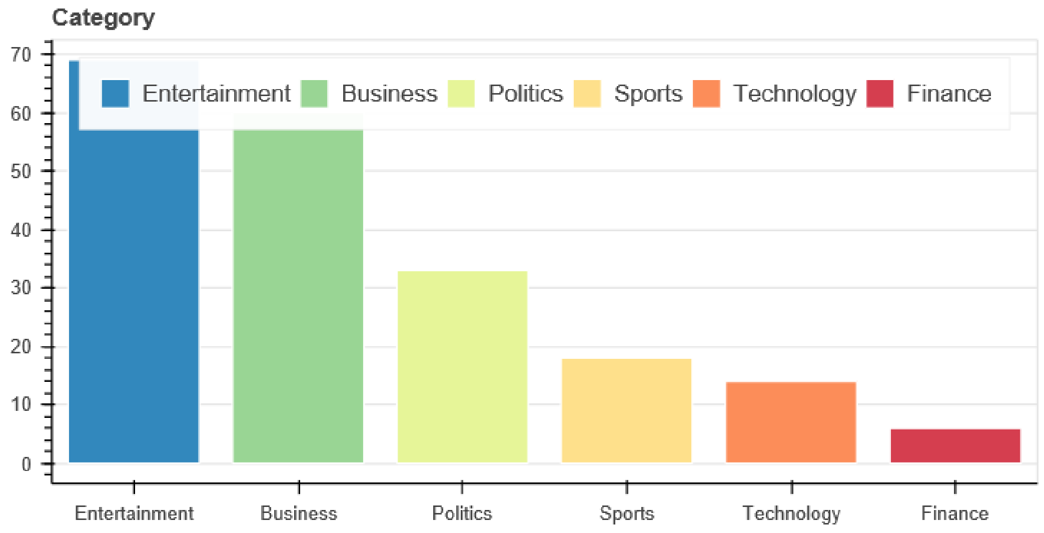


Fig. 4- Category Analysis

3.1.4 Word-cloud

Here, Top-words in Trends will give more detailed explanation about that topic. So, for now we have sentiment plot which says, what's people opinion on it, category will say in what categories it belongs and Word-Cloud will give more detailed explanation.

Initially we removed stop-words from the tweets and then used matplotlib. pyplot library to generate Word-Cloud. Further, we saved plot (Wordcloud) into an image and then plotted that image using bokeh. Here is snippet of code and Wordcloud:

```
plt.savefig('a'+col+'.png')
p = figure(x_range=(0,1), y_range=(0,1))
p.image_url(url=['a'+col+'.png'], x=0, y=1, h=1, w=1)
```

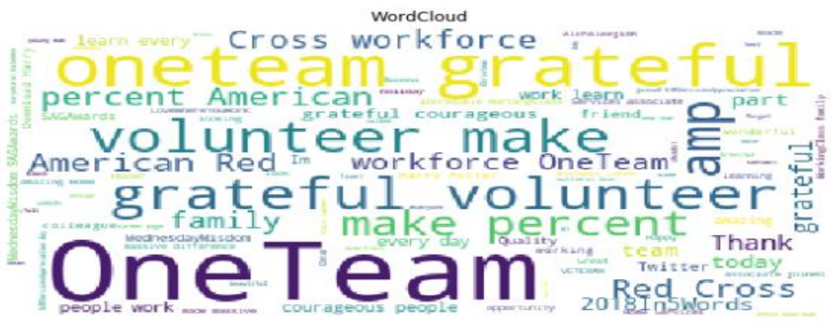


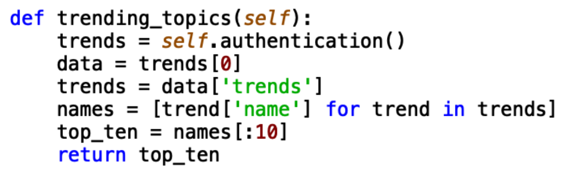
Fig. 5- Word cloud for Trending Topic One Team

Chapter 4 Technical components

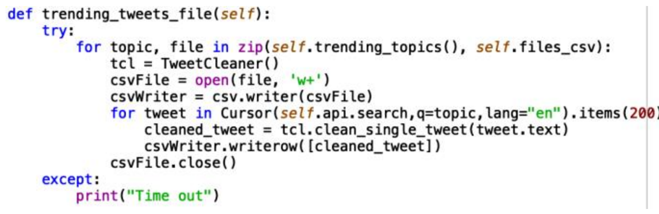
4.1 Category Analysis

Data Collection and Pre-processing

The Twitter has API access limits for a common developer account. Using Tweepy API, we first retrieved trending topics from the Twitter. Out of total 40 current trending topics, we chose only 10 for categorizing and sentiment analysis. Along with these trending topics, we also retrieved tweets related to each of the 10 trending topics. Due to API access limits on trending hashtags, we could retrieve around 200 tweets of each topic. These tweets were first cleaned and then stored into a separate .csv file for each trending topic. These 10 .csv files were then sent to next module for sentiment analysis.



```
def trending_topics(self):
    trends = self.authentication()
    data = trends[0]
    trends = data['trends']
    names = [trend['name'] for trend in trends]
    top_ten = names[:10]
    return top_ten
```



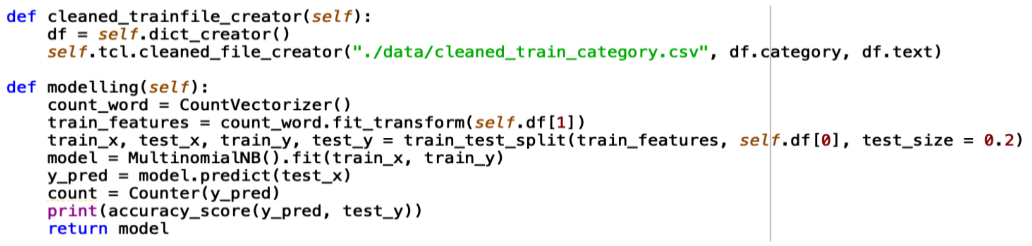
```
def trending_tweets_file(self):
    try:
        for topic, file in zip(self.trending_topics(), self.files_csv):
            tcl = TweetCleaner()
            csvFile = open(file, 'w+')
            csvWriter = csv.writer(csvFile)
            for tweet in Cursor(self.api.search,q=topic,lang="en").items(200):
                cleaned_tweet = tcl.clean_single_tweet(tweet.text)
                csvWriter.writerow([cleaned_tweet])
            csvFile.close()
    except:
        print("Time out")
```

Fig. 6: Coding snippets for retrieving trending topics and tweets

As in visualizing module, we are showing the categories of each trending topic, therefore had to categorize each trending topic into prespecified categorize such as technology, sports, finance, business, politics, entertainment. We used a machine learning model to categorize each trending topic correctly. We first, trained the Naïve Bayes model on a training data collected using Tweepy streaming API. To insure the correctness of the dataset collected for each category, we used several hashtags related to each category and then fetched around 80,000 tweets for each category. Tweets, in general, contain some words which are most common among all the tweets, such words are meaningless for the prediction task. For such tweets, we need a separate cleaning task which is generally called Text Pre-processing.

In this project we have used following tools to remove such meaningless text:

- Regex: To remove hyperlinks, non-ASCII characters, extra spaces, and punctuation.
- NLTK library: We have used this text-processing library to remove stop words and lemmatizing the tweets and description.



```
def cleaned_trainfile_creator(self):
    df = self.dict_creator()
    self.tcl.cleaned_file_creator("./data/cleaned_train_category.csv", df.category, df.text)

def modelling(self):
    count_word = CountVectorizer()
    train_features = count_word.fit_transform(self.df[1])
    train_x, test_x, train_y, test_y = train_test_split(train_features, self.df[0], test_size = 0.2)
    model = MultinomialNB().fit(train_x, train_y)
    y_pred = model.predict(test_x)
    count = Counter(y_pred)
    print(accuracy_score(y_pred, test_y))
    return model
```

Fig. 7: Code Snippet for Naïve Bayes modeling for category analyzer

The last step of the pre-processing would be text feature extraction. The text file can be considered as a chain of ordered words. We need to convert the text file into feature vectors containing numbers so that the machine learning model can work on that data. For this purpose, we used Sklearn' s text feature extraction tool, Count Vectorizer, which could split the text file by space between words and then create a 'tweets to terms' matrix containing numeric values. After this, initially we trained the Naïve Bayes on the dataset of each category file and then took new trending topics and tested on them.

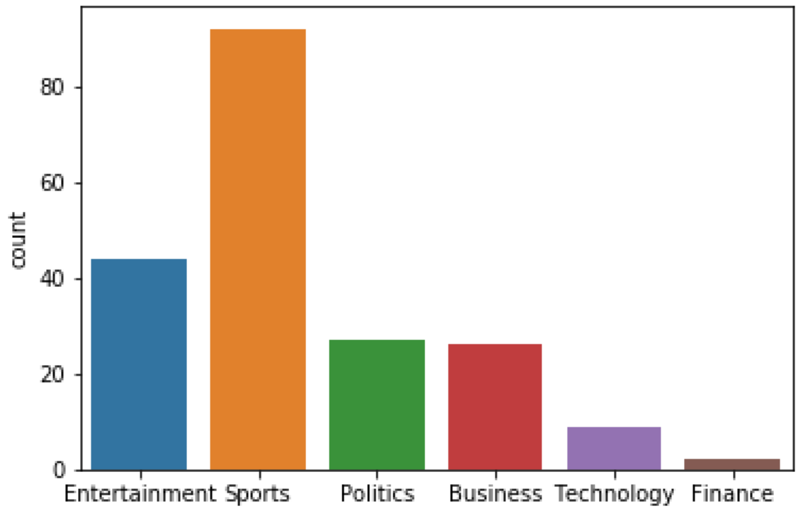


Fig. 8: **Results of NB**

The above figure shows that, when we applied NB on the testing data, we got results that shows that the testing data i.e. a new trending topic contained the tweets related mostly about sports hence that topic got categorized as Sports category.

4.2 Sentiment Analysis

Twitter has become one of the largest platforms with people posting real-time messages about products, movies, politics and many more categories. The flow of information in this platform has been tremendous since its inception. As many people have started using this platform, many companies have started to be active by replying its customers' feedbacks. Even so, it requires lots of unbelievable manual effort to go through all the tweets to find out sentiment of a topic. Hence, there was a need for an automated system which finds out the sentiment. There are three classes related to sentiment analysis: positive, negative and neutral. In our project, we have used only positive and negative class.

Mostly, sentiment analysis is done using Natural Language processing. In python, there is a separate library named NLTK to support NLP tasks. We started this module by fetching the tweets related to top 10 twitter trending topic. The next stage is the data preprocessing part, and this is one of the most important part. People consider this as one of the important stages as it will decrease the accuracy if the cleaning is not done properly. In this stage, we removed URLs, usernames, special characters, non ascii characters.

The next stage is the modelling part. In this stage, we tried to fit the training data into different classifiers. We had used AdaBoost, Linear SVM, Naïve Bayes and Random Forest. Random Forest is one of the ensembling method with various random trees with certain set of depths generated till it finds a suitable model for its data. We had split the dataset in the ratio 80:20, where 80 is to the training data and 20 is to the test data.

The parameters used for Logistic Regression is solver = liblinear, max_iter = 200. For decision tree, we have used default parameters. For Random Forest, we have used n_estimators = 30. For AdaBoost and Multinomial Naïve Bayes, we have used default parameters. We have used TfidfVectorizer for feature extraction. This class objective is to give weightage to rare words used in the document.

TfidfVectorizer works good in most of the cases as we know that the words which are more frequent like stop words is less important than most of the rare words in the document. After all these steps, we ran for the best model and received an accuracy of 74.4 percentage.

APPENDIX A - SOURCE CODE

```
********************************tweet_cleaner.py****************************
import re
import string
import ast
import nltk
import csv
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
lemma = nltk.WordNetLemmatizer()

class TweetCleaner:
   def __init__(self):
      self.stop_words = set(stopwords.words('english'))
      self.punc_table = str.maketrans("", "", string.punctuation) # to remove punctuation from each word
in tokenize

   def compound_word_split(self, compound_word):
      matches = re.finditer('.+?(?:(?<=[a-z])(?=[A-Z])|(?<=[A-Z])(?=[A-Z][a-z])|$)', compound_word)
      return [m.group(0) for m in matches]

   def remove_non_ascii_chars(self, text):
      return ''.join([w if ord(w) < 128 else ' ' for w in text])

   def remove_hyperlinks(self,text):
      return ' '.join([w for w in text.split(' ')  if not 'http' in w])

   def get_cleaned_text(self, text):
      cleaned_text = text.replace('\"','').replace('\'','').replace('-',' ')
      cleaned_text =  self.remove_non_ascii_chars(cleaned_text)
      if re.match(r'RT @[_A-Za-z0-9]+:',cleaned_text):
         cleaned_text = cleaned_text[cleaned_text.index(':')+2:]
      cleaned_text = self.remove_hyperlinks(cleaned_text)
      cleaned_text = cleaned_text.replace('#','HASHTAGSYMBOL').replace('@','ATSYMBOL')
      tokens = [w.translate(self.punc_table) for w in word_tokenize(cleaned_text)]
      tokens = [lemma.lemmatize(w) for w in tokens if not w.lower() in self.stop_words and len(w)>1]
      cleaned_text = ' '.join(tokens)
      cleaned_text = cleaned_text.replace('HASHTAGSYMBOL','#').replace('ATSYMBOL','@')
      cleaned_text = cleaned_text
      return cleaned_text

   def clean_tweets(self, tweets, is_bytes):
      test_tweet_list = []
      for tweet in tweets:
         if is_bytes:
            test_tweet_list.append(self.get_cleaned_text(ast.literal_eval(tweet)))
            else:
                test_tweet_list.append(self.get_cleaned_text(tweet))
        return test_tweet_list

    def clean_single_tweet(self, tweet, is_bytes = False):
        if is_bytes:
             return self.get_cleaned_text(ast.literal_eval(tweet).decode("UTF-8"))
        return self.get_cleaned_text(tweet)

    def cleaned_file_creator(self, op_file_name, value1, value2):
        csvFile = open(op_file_name, 'w+')
        csvWriter = csv.writer(csvFile)
        for tweet in range(len(value1)):
            csvWriter.writerow([value1[tweet], value2[tweet]])
        csvFile.close()
```

*******************************category_analyser.py**************************

```
import pandas as pd
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from collections import Counter
from sklearn.feature_extraction.text import CountVectorizer
from tweet_cleaner import TweetCleaner

class CategoryAnalyser:
    def __init__(self, cleaned_file = "./data/cleaned_train_category.csv", cleaner = False):
        self.tweets = []
        self.filelist = ['./data/tech_raw.txt' , './data/sports_raw.txt','./data/fnl_raw.txt',
                './data/business_raw.txt','./data/politics_raw.txt','./data/ent_raw.txt']
        self.labels = ['Technology','Sports','Finance','Business','Politics','Entertainment']
        self.tcl = TweetCleaner()
        if(cleaner):
            self.cleaned_trainfile_creator()
        self.df = pd.read_csv(cleaned_file, encoding = 'latin-1', header = None)
        self.feature_names = None

    def dict_creator(self):
        for k in range(0, len(self.filelist)):
            my_dict = {}
            self.my_list = self.my_list = open(self.filelist[k], 'r').read().split('\n')
            for i in range(0, len(self.my_list)):
                my_dict["category"] = self.labels[k]
                my_dict["text"] = self.tcl.clean_single_tweet(self.my_list[i], True)
                self.tweets.append(my_dict.copy())
        df = pd.DataFrame(self.tweets)
        return df

    def test_feature_count(self, test_feature):
        count_word = CountVectorizer(vocabulary = self.feature_names)
        return count_word.fit_transform(test_feature)

    def counter_value(self, pred):
        return Counter(pred).most_common()

    def cleaned_trainfile_creator(self):
        df = self.dict_creator()
        self.tcl.cleaned_file_creator("./data/cleaned_train_category.csv", df.category, df.text)

    def modelling(self):
        count_word = CountVectorizer()
        train_features = count_word.fit_transform(self.df[1])
        train_x, test_x, train_y, test_y = train_test_split(train_features, self.df[0], test_size = 0.2)
        model = MultinomialNB().fit(train_x, train_y)
        self.feature_names = count_word.get_feature_names()
        y_pred = model.predict(test_x)
        print(accuracy_score(y_pred, test_y))
        return model
```

***************************sentiment_analyser.py*******************************

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.model_selection import train_test_split
import nltk
from tweet_cleaner import TweetCleaner
nltk.download("stopwords")

class SentimentAnalysis:
    def __init__(self, file_name = "./data/cleaned_train_sentiment.csv", clean_required = False):
        self.tcl = TweetCleaner()
        if(clean_required):
            self.cleaned_train_file("./data/train.csv")
        self.cleaned_train_tweets = pd.read_csv(file_name, encoding = 'latin-1', header = None)
        self.max_features = 2000
        self.feature_names = None
        self.Classifiers = [
            LogisticRegression(C=0.000000001,solver='liblinear',max_iter=200),
            DecisionTreeClassifier(),
            RandomForestClassifier(n_estimators=30),
            AdaBoostClassifier(),
            MultinomialNB()]

    def set_max_features(self, value):
        self.max_features = value

    def analyser(self):
        count_word = TfidfVectorizer(max_features = self.max_features, analyzer = "word")
        train_features = count_word.fit_transform(self.cleaned_train_tweets[1].values.tolist())
        self.feature_names = count_word.get_feature_names()
        return self.modelling(train_features)

    def count_test_feature(self, testfeature):
        count_word2 = TfidfVectorizer(vocabulary = self.feature_names, analyzer = "word")
        test_feature = count_word2.fit_transform(testfeature)
        return test_feature

    def best_model(self, model_list):
        return sorted(model_list)[0][1]

    def modelling(self, train_f, run_all_classifiers = False):
        model_list = []
        train_x, test_x, train_y, test_y = train_test_split(train_f, train_tweets[0], test_size = 0.2)

        if(not run_all_classifiers):
            self.Classifiers = [MultinomialNB()]

        for classifier in self.Classifiers:
            model = classifier.fit(train_x, train_y)
            pred = model.predict(test_x)

            #Evaluation
            accur = accuracy_score(pred, test_y)
            print('Accuracy of '+ classifier.__class__.__name__+' is '+str(accur))
            model_list.append((accur, model))

        return self.best_model(model_list)

    def cleaned_train_file(self, file_name):
        train_tweets = pd.read_csv(file_name, encoding = 'latin-1')
        train_tweet_list = self.tcl.clean_tweets(train_tweets.SentimentText, False)
        self.tcl.cleaned_file_creator("./data/cleaned_train_sentiment.csv", train_tweets.Sentiment,
train_tweet_list)
```

*******************************twitter_bokeh_interactor.py**************************

```
from twitter_retr import TweetRetriever
from category_analyser import CategoryAnalyser
from sentiment_analyser import SentimentAnalysis
import pandas as pd

class TwitterBokehInteractor:
    def __init__(self):
        self.sentiment_analysis = SentimentAnalysis()
        self.sentiment_analysis_model = self.sentiment_analysis.analyser()
        self.category_analysis = CategoryAnalyser()
        self.category_analysis_model = self.category_analysis.modelling()
        self.twitter_api_obj = TweetRetriever()
        self.twitter_api_obj.trending_tweets_file()

    def sentiment_array(self, trend_number):
        data = pd.read_csv("./data/T" + str(trend_number) + ".csv", header = None)
        test_feature = self.sentiment_analysis.count_test_feature(data[0])
        value = []
        for i in range(2, test_feature.shape[0] - 1, 25):
            pred = self.sentiment_analysis_model.predict(test_feature[:i, :])
            value.append(sum(pred)/ len(pred))
        return value

    def trending_topics(self):
        return self.twitter_api_obj.trending_topics()

    def countplot(self, trend_number):
        data = pd.read_csv("./data/T" + str(trend_number) + ".csv", header = None)
        test_feature = self.category_analysis.test_feature_count(data[0])
        pred = self.category_analysis_model.predict(test_feature)
        value = self.category_analysis.counter_value(pred)
        return value

    def word_cloud_words(self, trend_number):
        data = pd.read_csv("./data/T" + str(trend_number) + ".csv", header = None)
        return " ".join(data[0])
```

**********************************screen_generator.py**********************

```
from twitter_bokeh_interactor import TwitterBokehInteractor
from bokeh.models.widgets import Panel, Tabs
from bokeh.plotting import figure
from bokeh.layouts import layout
from bokeh.layouts import row,column
from bokeh.io import save, output_file
from bokeh.models import Range1d, PanTool, ResetTool
from wordcloud import WordCloud, STOPWORDS
import nltk
from bokeh.palettes import Spectral6
from bokeh.models import ColumnDataSource
from bokeh.transform import factor_cmap
import matplotlib.pyplot as plt

def wordcloud(tweets,col):
    nltk.download("stopwords")
    stopwords = set(STOPWORDS)
    wordcloud =WordCloud(background_color="white",stopwords=stopwords).generate(tweets)
    plt.figure(figsize=(10,10), facecolor='k')
    plt.imshow(wordcloud)
    plt.axis("off")
    plt.title("WordCloud")
    plt.savefig('./images/a'+col+'.png')
    p = figure(x_range=(0,1), y_range=(0,1))
    p.image_url(url=['./images/a'+col+'.png'], x=0, y=1, h=1, w=1)
    p.xaxis.visible= False
    p.yaxis.visible= False
    return p

def call_Sentimentplot(s):
    a=[25,50,75,100,125,150,175,200]   #Convert dataframe into list range list
    pos= s
    pos = [j*100 for j in pos]
    neg=[]
    for i in range(len(a)):
        neg.append(100-pos[i])  #
    fig = figure()
    fig.width=600
    fig.height=300
    fig.background_fill_color='azure'
    fig.background_fill_alpha=0.2

    #Style the title
    fig.title.text="Sentiment on Trending Tweets"
    fig.title.text_color="olive"
    fig.title.text_font="times"
    fig.title.text_font_size="25px"
    fig.title.align="center"


    #Style the axes
    fig.xaxis.minor_tick_line_color="blue"
    fig.yaxis.major_label_orientation="vertical"
    fig.xaxis.visible=True
    fig.xaxis.minor_tick_in=-6
    fig.xaxis.axis_label="Number of Trending Tweets "
    fig.yaxis.axis_label="Percentage(%)"
    fig.axis.axis_label_text_color="blue"
    fig.axis.major_label_text_color="orange"

    #Axes geometry
    fig.y_range = Range1d(start=0, end=100)

    #Style the grid
    fig.xgrid.grid_line_color = None
    fig.ygrid.grid_line_alpha = 0.3
    fig.grid.grid_line_dash = [5,3]

    #adding glyph:
    fig.line(x=a,y=pos,color='red',legend='Positive Sentiment Trend')
    fig.line(x=a,y=neg,color='blue',legend='Negative Sentiment Trend')

    #Style the legend
    fig.legend.location = 'top_right'
    fig.legend.background_fill_alpha = 0.8
    fig.legend.border_line_color = None
    fig.legend.padding = 18
    fig.legend.label_text_color = 'olive'
    fig.legend.label_text_font = 'times'

    #Style the tools
    fig.tools = [PanTool(),ResetTool()]
    fig.toolbar_location = 'above'
    fig.toolbar.logo = None
    return(fig)

def Categorical_plot(c):
    c=dict(c)
    category = list(c.keys())
    counts = list(c.values())
    source = ColumnDataSource(data=dict(category=category, counts=counts))
    p = figure(x_range=category,plot_height=300, toolbar_location=None, title="Category")
    p.vbar(x='category', top='counts', width=0.8,source=source, legend="category",
      line_color='white', fill_color=factor_cmap('category', palette=Spectral6, factors=category))

    p.xgrid.grid_line_color = None
    p.legend.orientation = "horizontal"
    p.legend.location = "top_center"
    return p

def callDashboard():
    l=["1","2","3","4","5","6","7","8","9","10"]
    TwitterBokehInteractor_Object=TwitterBokehInteractor()
    Trendingtopic=TwitterBokehInteractor_Object.trending_topics()
    objectList=[]
    for i in range(len(l)):
        objectList.append(Panel(child=row(
                column(call_Sentimentplot(TwitterBokehInteractor_Object.sentiment_array(l[i])),
                Categorical_plot(TwitterBokehInteractor_Object.countplot(l[i])))),
                wordcloud(Twitter.word_cloud_words(l[i]),l[i])) , title=Trendingtopic[i] , ))
    tabs = Tabs(tabs=[t for t in objectList ])
    BigPanel = Panel(child=tabs, title="Ongoing_Trend/Hashtag " )
    tabs1 = Tabs(tabs=[BigPanel])
    lay_out1=layout([[tabs1]],sizing_mode='stretch_both')
    print("Return")
    return lay_out1

layout_web=callDashboard()
output_file("WEB_OUTPUT.html")
save(layout_web)
```