Sentiment Analysis of twitter data using Rule based lexicon dictionary.

Submitted by

Aditya Srivastava – 17BCE0613

CSE4020 – Machine Learning (EPJ)

Project Guide

Dr. Manjula V.

SCOPE

Winter Semester 2019-20

B.Tech. C.S.E
Vellore Institute of Technology, Vellore

1. ABSTRACT

We examine sentiment analysis on twitter data using VADER (for Valence (Valence Aware Dictionary and Sentiment Reasoner). In this project, we collect the data from social media platform and the data is assigned a sentiment score. The technique used is VADER (Valence Aware Dictionary and Sentiment Reasoner). The sentiment score is calculate from a lexicon rule based dictionary and the data is plotted for visual representation. A web interface is provided to enter the query and get relevant sentiment data, this data is then plotted using graphs.

Keywords: VADER; Deep Learning; Lexicon based rule dictionary; Sentiment Analysis; Social Media; Validated Sentiment Lexicon; Human-Centred Computing.

2. Table of Content

3. OBJECTIVES AND PROBLEM STATEMENT

Understanding the context and the sentiments conveyed by a paragraph or any textual content is an easy task for humans, but very challenging for a machine as it involves both understanding the content of the text and how to translate this into a natural human language. Many existing methods and algorithms fail at understanding human sentiment as the text itself may not be comprehensible most of the time. During elections, people end up forming their opinions based on information available on social networking sites. Thus, understanding the sentiment of such online content can help formulate an overview of the general perception of political figures beforehand.

Twitter is one of the most popular social network websites and has been growing at a very fast pace. The number of Twitter users reached an estimated 75 million by the end of 2009, up from approximately 5 million in the previous year. Through the twitter platform, users share either information or opinions about personalities, politicians, products, companies, events (Prentice and Huffman, 2008) etc. This has been attracting the attention of different communities interested in analysing its content.

In this project, we collect the data from social media platform and the data is assigned a sentiment score. The technique used is VADER (Valence Aware Dictionary and Sentiment Reasoner). The sentiment score is calculate from a lexicon rule based dictionary and the data is plotted for visual representation.

Twitter Sentiment Analysis can be a useful vehicle to provide deep insight into how citizens feel, thus it enables Governments across the world to track the public's political views. This project analyzes the data from twitter from 2017-18 just before the 2019 elections. This gives an insight into the public's general perception about a party and party members.

4. LITERATURE REVIEW

Literature Survey S.No.	Title of the Paper and Year	Algorithms Used	Data Set Being Used	Performance Measures	Limitations	Scope for future work
1.	Twitter Sentiment Analysis with a Deep Neural Network: An Enhanced Approach using User Behavioral Information 2018	Convolutional Neural Network (CNN)	Twitter Sentiment Analysis of SemEval-2015, SemEval-2013	precision, recall, F1, and accuracy	Lack of sentiment levels	Recurrent Neural Networks (RNN) and gated feedback RNN for sentiment analysis
2.	Twitter sentiment analysis using hybrid cuckoo search method 2017	Hybrid K-means and cuckoo search	Twitter dataset, Testdata.manual.2009.06.14, Twitter-sanders-apple2 , Twitter-sanders-apple3	Accuracy, Computational time, fitness function value	Lack of optimization	Accuracy improvement by optimized feature selection. Dealing with sarcastic tweets.
3.	Twitter Sentiment Analysis of the 2016 U.S. Presidential Election Using an Emoji Training Heuristic	Multinomial Naïve Bayes classifier	Dataset retrieved using twitter API for corpus of tweets containing specified keywords	F-measure, Accuracy	Lack of levels for sentiments, Less accurate for very short tweets, Less accurate for sarcastic	handle sarcasm and irony properly, to bring a higher accuracy of the sentiment model

					emoticon based tweets	
4.	Deep Convolution Neural Networks for Twitter Sentiment Analysis	Deep CNN (GloVe-DCNN)	The Stanford Twitter Sentiment Test (STSTd), The SE2014 dataset was provided in SemEval2014 Task9, The Stanford Twitter Sentiment Gold (STSGd) data set, The Sentiment Evaluation Dataset(SED), The Sentiment Strength Twitter dataset (SSTd)	Precision, Recall, F1-measure	None	Better handling for sarcastic tweets
5.	Contextual semantics for sentiment analysis of Twitter	SentiCircle Algorithm	OMD Diakopoulos and Shamma (2010), HCR Speriosu et al. (2011), STS-Gold Saif et al. (2013)	accuracy and F-measure	None	Optimize, and controlling the size of the term-index

5. PROPOSED ALTERNATIVE

In place of the methods for sentiment analysis listed above 1) CNN, 2) KCS, 3) Naïve Bayes, 4) GloVe-DCNN, the method we are using is VADER (Valence Aware Dictionary for sEntiment Reasoning). This uses sentiment lexicon which are lists of lexical features which are labelled according to their semantic orientation as either positive or negative (Liu, 2010). Manually creating and validating such lists of opinion-bearing features, while being among the most robust methods for generating reliable sentiment lexicons, is also one of the most time-consuming. For this reason, much of the applied research leveraging sentiment analysis relies heavily on pre-existing manually constructed lexicons.

Instead of using a classifier model or a decision tree it uses a predefined dictionary of lexicons to accurately and quickly determine the sentiment of the text. This allows VADER to be used in real time sentiment analysing.

As discussed by (Hutto and Gilbert, 2014), Data is gathered and compared to the lexicon (LIWC, ANEW, GI), then use data driven iterative inductive coding analysis to identify generalizable heuristics for assessing sentiment in text. Then evaluate the impact of text by assigning sentiment score from 4 to -4 inclusive and produce its sentiment analysis in the basis of the score.

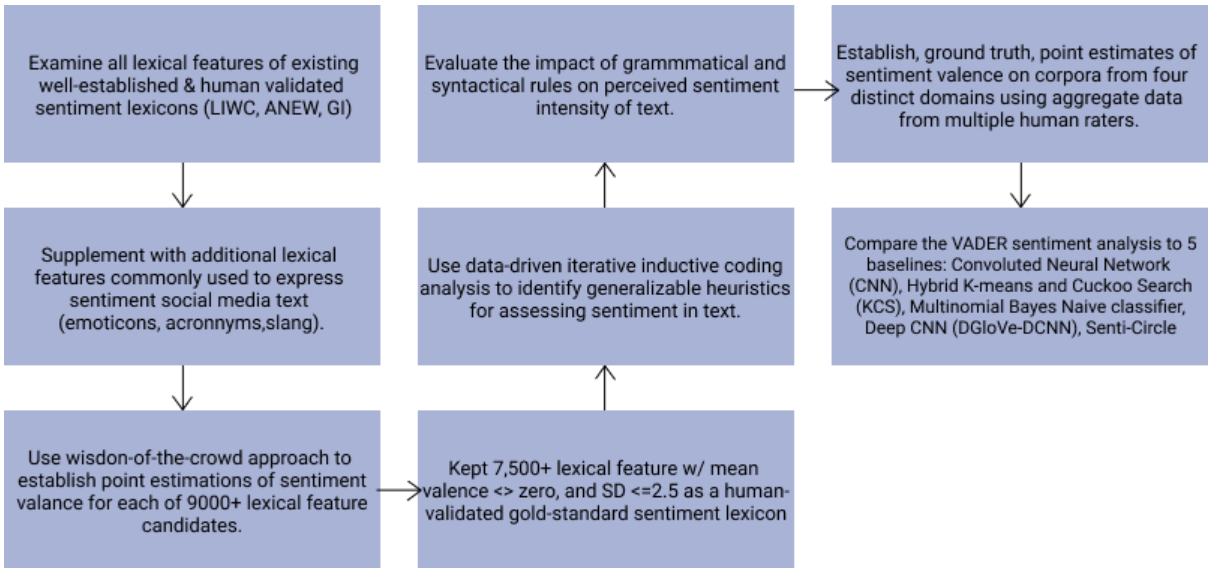

Figure 1 Proposed method flow

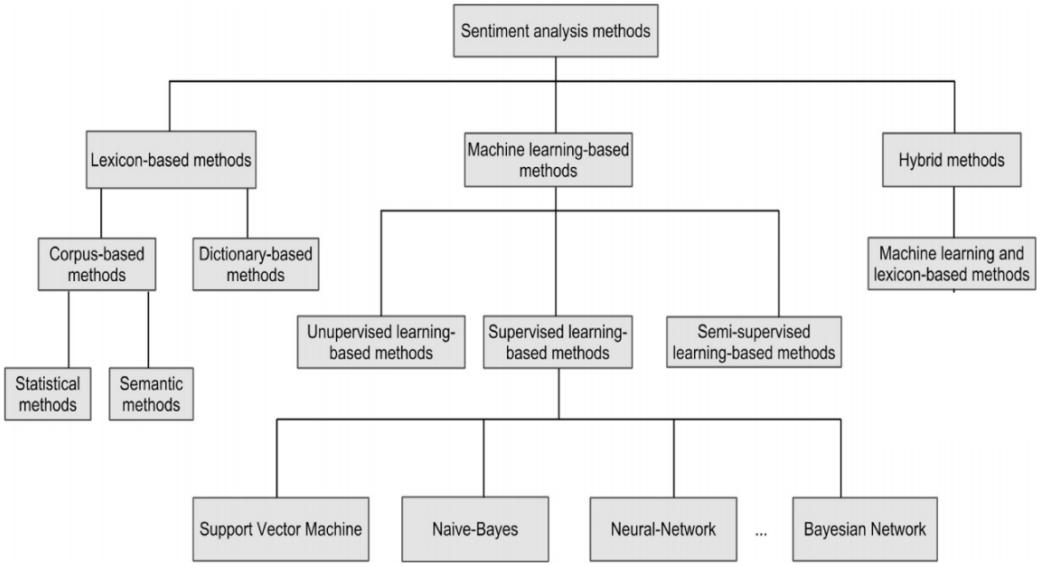

Fig. 1. Sentiment classification methods.

6. IMPLEMENTATION

Extracting data from social media - data mostly from twitter (The data extracted is from end of 2017 to 2018 just before 2019 Indian elections)

The data was filterer using keywords like India, Indian, BJP, Congress, NCP, NDA, INC, Election2019. The data was cleaned - removal of links, stop-words. Each tweet was assigned a sentiment score. The graphs shown is the result of grouping keywords and the sentiment score.

(Hutto and Gilbert, 2014) VADER awards one out of the following sentiment score to the tweet.

1. Slightly Negative [-1]
2. Moderately Negative [-2]
3. Very Negative [-3]
4. Extremely Negative [-4]
5. Neutral [0]
6. Slightly Positive [1]
7. Moderately Positive [2]
8. Very Positive [3]
9. Extremely Positive [4]

This score is then used to understand the sentiment behind a text and render a plot on the basis of this score.

Since we are using a large dataset, we are able to plot the sentiment towards a political party or a political figure over the period of (2017-2019).

Implementation coding is given in the appendix section.

7. PERFORMANCE AND TESTING

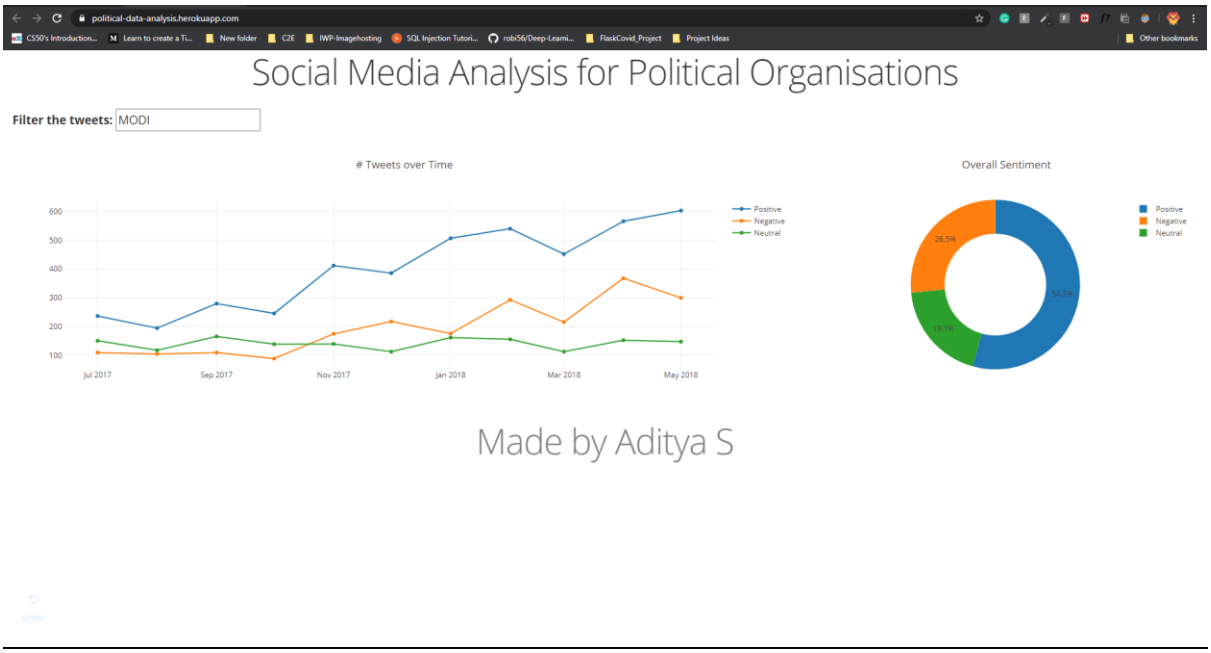

Figure 2: MODI

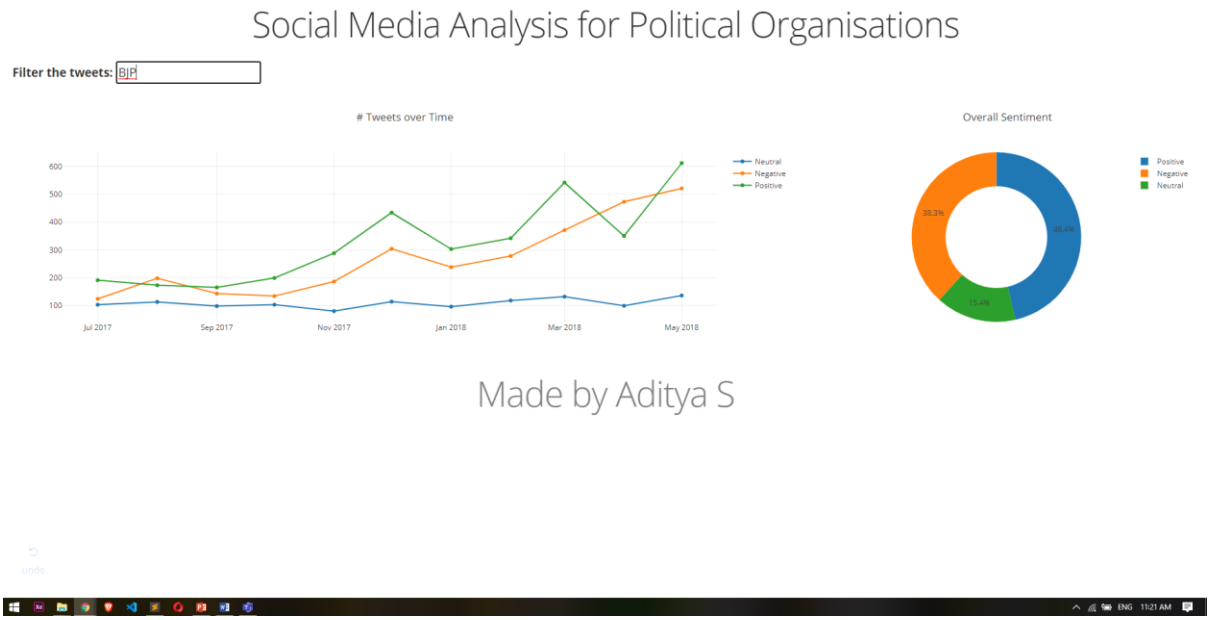

Figure 3: BJP

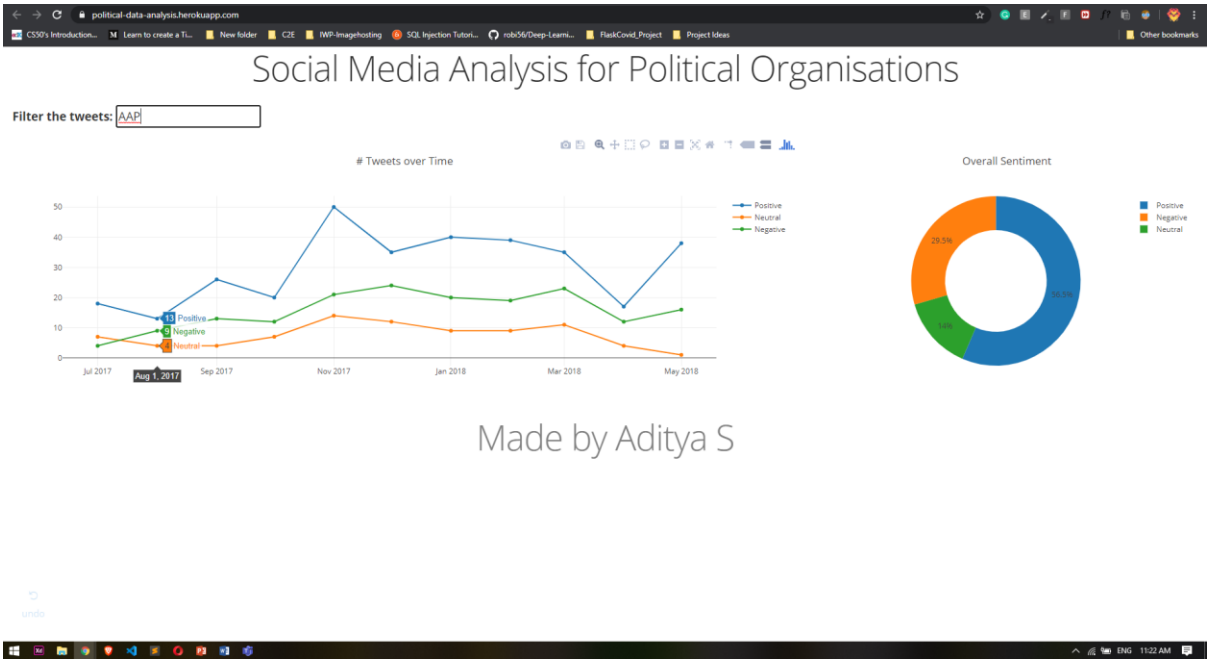

Figure 4: AAP

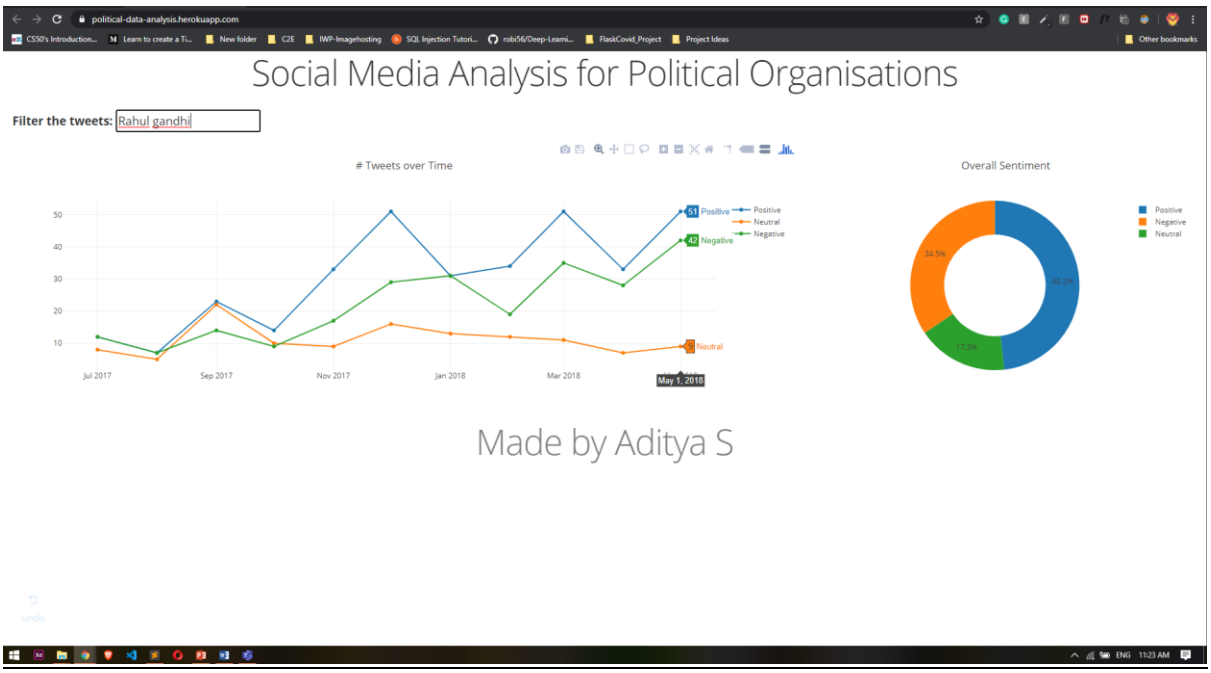

Figure 5: Rahul Gandhi

8. CONCLUSION AND FUTURE IMPROVEMENTS

The systematic development and evaluation of VADER (Valence Aware Dictionary for sEntiment Reasoning). Using a combination of qualitative and quantitative methods, we construct and empirically validate a gold standard list of lexical features (along with their associated sentiment intensity measures) which are specifically attuned to sentiment in microblog-like contexts. We then combine these lexical features with consideration for five general rules that embody grammatical and syntactical conventions for expressing and emphasizing sentiment intensity. The results are not only encouraging – they are indeed quite remarkable; VADER performed as well as (and in most cases, better than) eleven other highly regarded sentiment analysis tools. Our results highlight the gains to be made in computer science when the human is incorporated as a central part of the development process.

Future improvements are directly linked to the human-verified lexicon dictionary, therefore with continuous improvements to the lexicons, the accuracy of the method increases.

9. APPENDIX

A. Render the Website and draw the plots

```
import dash
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output
import datetime as dt
import os
import pandas as pd

# defining the format for timestamp
timestamp_format = "%Y-%m-%d %H:%M"

# reading the data through pandas
politics = pd.read_csv('final_text_sentiment.csv', sep = '|')

# function to extract month-year
def tweet_date(ts):
    result = dt.datetime.strptime(ts, timestamp_format)
    result = dt.datetime(result.year, result.month, 1)
    return result

# converting the text to all lower strings
politics['clean_tweet'] = politics['clean_tweet'].str.lower()

# getting a date column
politics['date'] = politics['date'].apply(tweet_date)

# function to filter based on text
def filter_tweets(filter_text=''):
    df = politics[politics['clean_tweet'].str.contains(str.lower(filter_text))]
    return df

# creating plots

# line chart
def tweets_by_date(in_df):

    unq_class = in_df.sentiment_class.unique()
    list_of_dict = []
    for j in range(0, len(unq_class)):
        df = in_df.loc[in_df['sentiment_class'] == unq_class[j]]
        df = df.reset_index(drop=True)
        group_date = df.groupby('date')['date'].count()
        date_l = group_date.index.tolist()
        val_l = group_date.tolist()
        list_of_dict.append({
                    "type": "scatter",
                    "mode": "lines+markers",
                    "name": unq_class[j],
```

```
                    "x": date_l,
                    "y": val_l
                })

    return {
        "data": list_of_dict,
        "layout": {
            "title": "# Tweets over Time",
            "showlegend": True
        }
    }


# donut chart
def tweets_class(in_df):

    unq_class = in_df.sentiment_class.unique()
    labels = []
    vals = []
    for j in range(0, len(unq_class)):
        df = in_df.loc[in_df['sentiment_class'] == unq_class[j]]
        len_df = len(df.index)
        labels.append(unq_class[j])
        vals.append(len_df)

    return {
        "data": [
            {
                "type": "pie",
                "labels": labels,
                "values": vals,
                "hole": 0.6
            }
        ],
        "layout": {
            "title": "Overall Sentiment"
        }
    }

# initializing the application
app = dash.Dash()

# for heroku deployment
server = app.server

# don't understand this one bit, but apparently it's needed
server.secret_key = os.environ.get("SECRET_KEY", "secret")

# title of the application
app.title = "Social Media Analysis for Political Organisation"

# layout of the application
app.css.append_css({
    "external_url":
"https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
```

```
})

app.css.append_css({
    "external_url": 'https://codepen.io/chriddyp/pen/bWLwgP.css'
})

app.scripts.append_script({
    "external_url": "https://code.jquery.com/jquery-3.2.1.min.js"
})

app.scripts.append_script({
    "external_url":
"https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"
})

# defining the layout now
app.layout = html.Div([
    # row: title
    html.Div([
        html.Div([
            html.H1("Social Media Analysis for Political Organisations",
classNamе="text-center")
        ], classNamе="col-md-12")
    ], classNamе="row"),

    # including some markdown text
    html.Div([
        html.Div([
                dcc.Markdown('''

                ''')
        ], classNamе="col-md-12")
    ], classNamе="row"),

    # box for filtering tweets
    html.Div([
        html.Div([
            html.P([
                html.B("Filter the tweets:  "),
                dcc.Input(
                    placeholder="Try 'Modi'",
                    id="tweet-filter",
                    value="")
            ]),
        ], classNamе="col-md-12"),
        ], classNamе="row"),

    # row: line chart + donut chart
    html.Div([
        html.Div([
            dcc.Graph(id="tweet-by-date")
        ], classNamе="col-md-8"),
        html.Div([
            dcc.Graph(id="tweet-class")
        ], classNamе="col-md-4")
```

```
    ], className="row"),

    # Row: Footer
    html.Div([
        html.Div([
            html.H1("Made by Avnish,Shrayans,Ananya,Bhavya", className="text-
center")
        ], className="col-md-12"),

    ], className="row",
        style={
            "textAlign": "center",
            "color": "Gray"
        })
], className="container-fluid")

# defining the interaction callbacks

@app.callback(
    Output('tweet-by-date', 'figure'),
    [
        Input('tweet-filter', 'value'),
    ]
)
def filter_tweet_year(filter_text):
    return tweets_by_date(filter_tweets(filter_text))


@app.callback(
    Output('tweet-class', 'figure'),
    [
        Input('tweet-filter', 'value')
    ]
)
def filter_tweet_class(filter_text):
    return tweets_class(filter_tweets(filter_text))

# running the app
if __name__ == "__main__":
    app.run_server(debug=True)
```

B. Sentiment_analyser.py

```
import pandas as pd
import re
import numpy as np
import datetime as dt
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
analyser = SentimentIntensityAnalyzer()

# reading the final output file
df = pd.read_csv('../GetOldTweets-python/final_output.csv', sep = '|')
print ('Output from Twitter read...')

# defining functiions to pre-process tweets
def clean_tweet(text):
    # capturing urls
    result = re.sub(r'http\S+', '', text)
    # because many urls have spaces which makes detection harder
    # replacing all full-stops, # and @
    result = re.sub('[.@#]', '', result)
    # broken url has '-' or '/' between words
    result = re.sub(r'\w+(?:/\w+)+',"", result)
    result = re.sub(r'\w+(?:-\w+)+',"", result)
    # deleting digits from the tweet
    result = re.sub(" \d+", " ", result)
    # removing short words with less than 3 character length
    result = " ".join([word for word in result.split() if len(word) > 2])
    return result

# applying the function to clean all tweets
df['clean_tweet'] = df['text'].apply(clean_tweet)
print ('Tweets cleaned...')

# calculating the sentiment scores
df['sentiment'] = df['clean_tweet'].apply(analyser.polarity_scores)
print ('Sentiment scores obtained...')

# function to get the compound value from resultant dictionary
def get_compound_score(dict_row):
    result = dict_row['compound']
    return result

# get the compound value from the sentiment result
df['sentiment_score'] = df['sentiment'].apply(get_compound_score)

# creating sentiment classes based on the composite sentiment scores
def sentiment_class(score):
    if score > 0:
        return 'Positive'
    elif score < 0:
        return 'Negative'
    elif score == 0.0:
        return 'Neutral'
    else:
```

```
        print ('Weird response!')

# obtaining the sentiment classes from the composite scores
df['sentiment_class'] = df['sentiment_score'].apply(sentiment_class)
print ('Saving the final output to disk...')

# saving only relevant columns
df = df[['date', 'clean_tweet', 'sentiment_score', 'sentiment_class']]
df.to_csv('final_text_sentiment.csv', index = False, sep = '|')
```

C. Fetch tweets

```
import os
import pandas as pd

# start date and end date input (format : 2018-01-30)
start_date = 'XXXX-XX-XX'
end_date = 'XXXX-XX-XX'

# the search term
search_terms = ["'india politics'", "'india prime minister'", "'india chief
minister'", "'india pm'", "'india cm'",
                "'india elections'", "'india vote party'", "'india bjp'", "'india
inc'", "'india bsp'", "'india tmc'",
                "'india cpi'", "'india ncp'", "'india aam aadmi party'", "'india
aiadmk'", "'india aifb'",
                "'india ainrc'", "'india aiudf'", "'india ajsu'", "'india asom gana
parishad'", "'india aipfp'",
                "'india bjd'", "'india bpf'", "'india gfp'", "'india hspdp'",
"'india inld'", "'india iuml'",
                "'india jknc'", "'india jknpp'", "'india jkpdp'", "'india janta
dal'", "'india jharkhand vikas morcha'",
                "'india kjp'", "'india kerala congress'", "'india ljp'", "'india
mns'", "'india mgp'", "'india mnf'",
                "'india npf'", "'india congress'", "'india npp'", "'india ndpp'",
"'india dmdk'",
                "'india people democratic alliance'", "'india rjd'", "'india rld'",
"'india rlsp'", "'india samajwadi party'",
                "'india akali dal'", "'india shiv sena'", "'india trs'", "'india
tdp'", "'india united democratic party'",
                "'india ysrc'", "'india aimim'", "'india dmk'", "'india upa'",
"'india nda'","'india pattali makkal'",
                "'india jharkhand mukti morcha'","'india mizoram people
conference'", "'india mizoram people conference'",
                "'india people party arunachal'", "'india revolutionary socialist
party'","'india sikkim democratic front'",
                "'india sikkim krantikari morcha'","'india marxist forward bloc'"]

# issuing queries for each term in the list

for i in range(0, len(search_terms)):

       term = search_terms[i]
       # constructing the query
       query = "python Exporter.py --since " + start_date + " "
       query = query + "--until "
       query = query + end_date + " "
       query = query + "--toptweets "
       query = query + "--querysearch "
       query = query + term

       # firing the query
       os.system(query)

       # moving and renaming the output file
```

```
        oname = 'data_fetch/out' + str(i+1) + '.csv'
        os.rename('output_got.csv', oname)

# merging all the csv files into one
fdata = pd.read_csv('data_fetch/out1.csv', delimiter = '|', error_bad_lines=False)
for i in range(1, len(search_terms)):
        oname = 'data_fetch/out' + str(i+1) + '.csv'
        data_l = pd.read_csv(oname, delimiter = '|', error_bad_lines=False)
        fdata = fdata.append(data_l)

# writing the final csv to disk
fdata = fdata.drop_duplicates()
fdata.to_csv('final_output.csv', sep = '|', index=False)
```

D. Export Parameters

```
import sys,getopt,datetime,codecs
if sys.version_info[0] < 3:
    import got
else:
    import got3 as got

def main(argv):

       if len(argv) == 0:
              print('You must pass some parameters. Use \"-h\" to help.')
              return

       if len(argv) == 1 and argv[0] == '-h':
              f = open('exporter_help_text.txt', 'r')
              print (f.read())
              f.close()

              return

       try:
              opts, args = getopt.getopt(argv, "", ("username=", "near=",
"within=", "since=", "until=", "querysearch=", "toptweets", "maxtweets=",
"output="))

              tweetCriteria = got.manager.TweetCriteria()
              outputFileName = "output_got.csv"

              for opt,arg in opts:
                     if opt == '--username':
                            tweetCriteria.username = arg

                     elif opt == '--since':
                            tweetCriteria.since = arg

                     elif opt == '--until':
                            tweetCriteria.until = arg

                     elif opt == '--querysearch':
                            tweetCriteria.querySearch = arg

                     elif opt == '--toptweets':
                            tweetCriteria.topTweets = True

                     elif opt == '--maxtweets':
                            tweetCriteria.maxTweets = int(arg)

                     elif opt == '--near':
                            tweetCriteria.near = '"' + arg + '"'

                     elif opt == '--within':
                            tweetCriteria.within = '"' + arg + '"'
```

```
                elif opt == '--within':
                        tweetCriteria.within = '"' + arg + '"'

                elif opt == '--output':
                        outputFileName = arg

        outputFile = codecs.open(outputFileName, "w+", "utf-8")


    outputFile.write('username|date|retweets|favorites|text|geo|mentions|hashta
gs|id|permalink')

        print('Searching...\n')

        def receiveBuffer(tweets):
                for t in tweets:
                        outputFile.write(('\n%s|%s|%d|%d|"%s"|%s|%s|%s|"%s"|%s'
% (t.username, t.date.strftime("%Y-%m-%d %H:%M"), t.retweets, t.favorites, t.text,
t.geo, t.mentions, t.hashtags, t.id, t.permalink)))
                outputFile.flush()
                print('More %d saved on file...\n' % len(tweets))

        got.manager.TweetManager.getTweets(tweetCriteria, receiveBuffer)

    except arg:
        print('Arguments parser error, try -h' + arg)
    finally:
        outputFile.close()
        print('Done. Output file generated "%s".' % outputFileName)

if __name__ == '__main__':
    main(sys.argv[1:])
```

E. DRIVER CODE TO QUERY TWEETS

```
import sys
if sys.version_info[0] < 3:
    import got
else:
    import got3 as got

def main():

       def printTweet(descr, t):
              print(descr)
              print("Username: %s" % t.username)
              print("Retweets: %d" % t.retweets)
              print("Text: %s" % t.text)
              print("Mentions: %s" % t.mentions)
              print("Hashtags: %s\n" % t.hashtags)

       # Example 1 - Get tweets by username
       tweetCriteria =
got.manager.TweetCriteria().setUsername('barackobama').setMaxTweets(1)
       tweet = got.manager.TweetManager.getTweets(tweetCriteria)[0]

       printTweet("### Example 1 - Get tweets by username [barackobama]", tweet)

       # Example 2 - Get tweets by query search
       tweetCriteria = got.manager.TweetCriteria().setQuerySearch('europe
refugees').setSince("2015-05-01").setUntil("2015-09-30").setMaxTweets(1)
       tweet = got.manager.TweetManager.getTweets(tweetCriteria)[0]

       printTweet("### Example 2 - Get tweets by query search [europe refugees]",
tweet)

       # Example 3 - Get tweets by username and bound dates
       tweetCriteria =
got.manager.TweetCriteria().setUsername("barackobama").setSince("2015-09-
10").setUntil("2015-09-12").setMaxTweets(1)
       tweet = got.manager.TweetManager.getTweets(tweetCriteria)[0]

       printTweet("### Example 3 - Get tweets by username and bound dates
[barackobama, '2015-09-10', '2015-09-12']", tweet)

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

F. GET TWEETS SUB ROUTINE

```
import urllib.request, urllib.parse,
urllib.error,urllib.request,urllib.error,urllib.parse,json,re,datetime,sys,http.co
okiejar
from .. import models
from pyquery import PyQuery

class TweetManager:

	def __init__(self):
		pass

	@staticmethod
	def getTweets(tweetCriteria, receiveBuffer=None, bufferLength=100,
proxy=None):
		refreshCursor = ''

		results = []
		resultsAux = []
		cookieJar = http.cookiejar.CookieJar()

		active = True

		while active:
			json = TweetManager.getJsonReponse(tweetCriteria,
refreshCursor, cookieJar, proxy)
			if len(json['items_html'].strip()) == 0:
				break

			refreshCursor = json['min_position']
			scrapedTweets = PyQuery(json['items_html'])
			#Remove incomplete tweets withheld by Twitter Guidelines
			scrapedTweets.remove('div.withheld-tweet')
			tweets = scrapedTweets('div.js-stream-tweet')

			if len(tweets) == 0:
				break

			for tweetHTML in tweets:
				tweetPQ = PyQuery(tweetHTML)
				tweet = models.Tweet()

				usernameTweet = tweetPQ("span.username.js-action-
profile-name b").text()
				txt = re.sub(r"\s+", " ", tweetPQ("p.js-tweet-
text").text().replace('# ', '#').replace('@ ', '@'))
				retweets = int(tweetPQ("span.ProfileTweet-action--
retweet span.ProfileTweet-actionCount").attr("data-tweet-stat-count").replace(",",
""))
				favorites = int(tweetPQ("span.ProfileTweet-action--
favorite span.ProfileTweet-actionCount").attr("data-tweet-stat-
count").replace(",", ""))
```

```
                        dateSec = int(tweetPQ("small.time span.js-short-
timestamp").attr("data-time"))
                        id = tweetPQ.attr("data-tweet-id")
                        permalink = tweetPQ.attr("data-permalink-path")
                        user_id = int(tweetPQ("a.js-user-profile-
link").attr("data-user-id"))

                        geo = ''
                        geoSpan = tweetPQ('span.Tweet-geo')
                        if len(geoSpan) > 0:
                              geo = geoSpan.attr('title')
                        urls = []
                        for link in tweetPQ("a"):
                              try:
                                    urls.append((link.attrib["data-expanded-
url"]))
                              except KeyError:
                                    pass
                        tweet.id = id
                        tweet.permalink = 'https://twitter.com' + permalink
                        tweet.username = usernameTweet

                        tweet.text = txt
                        tweet.date = datetime.datetime.fromtimestamp(dateSec)
                        tweet.formatted_date =
datetime.datetime.fromtimestamp(dateSec).strftime("%a %b %d %X +0000 %Y")
                        tweet.retweets = retweets
                        tweet.favorites = favorites
                        tweet.mentions = "
".join(re.compile('(@\\w*)').findall(tweet.text))
                        tweet.hashtags = "
".join(re.compile('(#\\w*)').findall(tweet.text))
                        tweet.geo = geo
                        tweet.urls = ",".join(urls)
                        tweet.author_id = user_id

                        results.append(tweet)
                        resultsAux.append(tweet)

                        if receiveBuffer and len(resultsAux) >= bufferLength:
                              receiveBuffer(resultsAux)
                              resultsAux = []

                        if tweetCriteria.maxTweets > 0 and len(results) >=
tweetCriteria.maxTweets:
                              active = False
                              break


            if receiveBuffer and len(resultsAux) > 0:
                  receiveBuffer(resultsAux)

            return results

      @staticmethod
```

```
	def getJsonReponse(tweetCriteria, refreshCursor, cookieJar, proxy):
		url = "https://twitter.com/i/search/timeline?f=tweets&q=%s&src=typd&%smax_position=%s"

		urlGetData = ''
		if hasattr(tweetCriteria, 'username'):
			urlGetData += ' from:' + tweetCriteria.username

		if hasattr(tweetCriteria, 'since'):
			urlGetData += ' since:' + tweetCriteria.since

		if hasattr(tweetCriteria, 'until'):
			urlGetData += ' until:' + tweetCriteria.until

		if hasattr(tweetCriteria, 'querySearch'):
			urlGetData += ' ' + tweetCriteria.querySearch

		if hasattr(tweetCriteria, 'lang'):
			urlLang = 'lang=' + tweetCriteria.lang + '&'
		else:
			urlLang = ''
		url = url % (urllib.parse.quote(urlGetData), urlLang, refreshCursor)
		#print(url)

		headers = [
			('Host', "twitter.com"),
			('User-Agent', "Mozilla/5.0 (Windows NT 6.1; Win64; x64)"),
			('Accept', "application/json, text/javascript, */*; q=0.01"),
			('Accept-Language', "de,en-US;q=0.7,en;q=0.3"),
			('X-Requested-With', "XMLHttpRequest"),
			('Referer', url),
			('Connection', "keep-alive")
		]

		if proxy:
			opener = urllib.request.build_opener(urllib.request.ProxyHandler({'http': proxy, 'https': proxy}), urllib.request.HTTPCookieProcessor(cookieJar))
		else:
			opener = urllib.request.build_opener(urllib.request.HTTPCookieProcessor(cookieJar))
		opener.addheaders = headers

		try:
			response = opener.open(url)
			jsonResponse = response.read()
		except:
			#print("Twitter weird response. Try to see on browser: ", url)
			print("Twitter weird response. Try to see on browser: https://twitter.com/search?q=%s&src=typd" % urllib.parse.quote(urlGetData))
			print("Unexpected error:", sys.exc_info()[0])
			sys.exit()
			return

		dataJson = json.loads(jsonResponse.decode())
```

```
            return dataJson
```

G. SET TWEET CRITERIA

```
class TweetCriteria:

      def __init__(self):
            self.maxTweets = 0

      def setUsername(self, username):
            self.username = username
            return self

      def setSince(self, since):
            self.since = since
            return self

      def setUntil(self, until):
            self.until = until
            return self

      def setQuerySearch(self, querySearch):
            self.querySearch = querySearch
            return self

      def setMaxTweets(self, maxTweets):
            self.maxTweets = maxTweets
            return self

      def setLang(self, Lang):
            self.lang = Lang
            return self

      def setTopTweets(self, topTweets):
            self.topTweets = topTweets
            return self
```

10. REFERENCES

1. Agarwal, A., Xie, B., Vovsha, I., Rambow, O., & Passonneau, R. (2011). Sentiment analysis of Twitter data. In Proc. WLSM-11s
2. Chandra Pandey, A., Singh Rajpoot, D., & Saraswat, M. (2017). Twitter sentiment analysis using hybrid cuckoo search method. Information Processing & Management, 53(4), 764–779. doi:10.1016/j.ipm.2017.02.004
3. Mengdi Li1, Eugene Ch'ng1 , Alain Chong2, and Simon See3 (2016). Twitter Sentiment Analysis of the 2016 U.S. Presidential Election Using an Emoji Training Heuristic.
4. Saif, H., He, Y., Fernandez, M., & Alani, H. (2016). Contextual semantics for sentiment analysis of Twitter. *Information Processing & Management*, *52*(1), 5-19.
5. ZHAO JIANQIANG1,2 , GUI XIAOLIN1,2, AND ZHANG XUEJUN2,3 (2017). Deep Convolution Neural Networks for Twitter Sentiment Analysis
6. Saif, H., He, Y., Fernandez, M., & Alani, H. (2016). *Contextual semantics for sentiment analysis of Twitter. Information Processing & Management, 52(1), 5–19.*
7. Ding, X., Liu, B., and Yu, P. 2008. A Holistic Lexicon-based Approach to Opinion Mining. WSDM 2008.
8. A. Pak and P. Paroubek, "Twitter as a Corpus for Sentiment Analysis and Opinion Mining.," Lrec, pp. 1320–1326, 2010.
9. U. Pavalanathan and J. Eisenstein, "Emoticons vs. Emojis on Twitter: A Causal Inference Approach," AAAI Spring Symp. Obs. Stud. through Soc. Media Other HumanGenerated Content, 2016.
10. A. Tumasjan, T. Sprenger, P. Sandner, and I. Welpe, "Predicting elections with Twitter: What 140 characters reveal about political sentiment," Proc. Fourth Int. AAAI Conf. Weblogs Soc. Media, pp. 178–185, 2010.
11. Hutto, C. J., & Gilbert, E. (2014, May). Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Eighth international AAAI conference on weblogs and social media*.