**CHAPTER 1**

**INTRODUCTION**

Opinion Mining is a process of automatic extraction of knowledge from the opinion of others about some particular topic or problem. The idea of Opinion mining and Sentiment Analysis tool is to “process a set of search results for a given item, generating a list of product attributes (quality, features etc.) and aggregating opinion”. But with the passage of time more interesting applications and developments came into existence in this area and now its main goal is to make computer able to recognize and generate emotions like human.

* 1. **Background**

Microblogging today has become a very popular communication tool among Internet users. Millions of users share opinions on different aspects of life everyday. Therefore microblogging web-sites are rich sources of data for opinion mining and sentiment analysis. Because microblogging has appeared relatively recently, there are a few research works that were devoted to this topic. In our paper, we focus on using Twitter, the most popular microblogging platform, for the task of sentiment analysis. We show how to automatically collect a corpus for sentiment analysis and opinion mining purposes. We perform linguistic analysis of the collected corpus and explain discovered phenomena. Using the corpus, we build a sentiment classifier, that is able to determine positive, negative and neutral sentiments for a document. Experimental evaluations show that our proposed techniques are efficient and performs better than previously proposed methods. In our research, we worked with English, however, the proposed technique can be used with any other language.

* 1. **Problem Statement**

To develop software to automate all the manual work of fetching tweets from twitter and to analyze the tweets. The purpose of this project is to make the entire manual work automatic with a rich and enhanced platform. It would be a one-stop solution for every new associate who joins the research. It also increases productivity of every engineer.

* 1. **Importance** 
     1. **Need for automation:**

The primary purpose of this research was to make the fetching of the tweets from the twitter automatic which was previously manual.

* + 1. **Need for general software:**

The secondary purpose of this research was to make the software be used for any keyword for any location and for any duration. That is, basically make the software generalized where the input is done during run-time.

* + 1. **Need for fetching any number of Tweets:**

The other purpose of this research was to fetch any number of tweets from twitter without user intervention. Basically, wanted to automatize the fetching of tweets from twitter and can be used by any person for analysis.

* + 1. **Need for sentimental analysis and graphical representation:**

Another key purpose of this research was to do the analysis of the tweets fetched from twitter i.e. positive, negative and neutral according to the user sentiments. And also based on those sentiments generate graph to make the analysis more easy and fruitful.

* 1. **Organization of Report**

The main body of the report is preceded by detailed contents including lists of figures, tables, annexes and an executive summary giving briefly the scope and objectives of the study, importance of the topic, methodology, limitations & roadblocks.

**CHAPTER 2**

**OVERVIEW AND PLANNING**

* 1. **Proposed System Overview**

The software to be made for automation of all processes must encompass all manual work carried out by an associate and to make software user friendly and efficient. In order to do this it has been divided into varies modules:

* User Interface: The user interface contains various options to fetch tweets from twitter and it is designed using QT designer.
* Module using Keyword: This includes Twitter Streaming function to fetch tweets from twitter based on the keywords. **Note: Twitter Streaming function cannot use both keyword and location together.**
* Module using Location: This includes Twitter Streaming function to fetch tweets from twitter based on location coordinates. **Note: Twitter Streaming function cannot use both keyword and location together.**
* Module using both location and keyword together: This includes Twitter Search Api to fetch tweets from twitter in this we can use both location and keyword together.
* Module using multithreading: This includes Twitter Search Api to fetch tweets from twitter but in this case added Multithreading feature to fetch tweets for more and one keywords using threads. And can be used for any number of keywords which depends on the number of threads used.
* Sentimental Analysis: This includes the analysis of the tweets fetched from twitter using the words used in the tweets to decide whether it is positive, negative or neutral.
* Graphical Representation of the Sentimental Analysis: This includes the graphical representation of the analyzed tweets as positive, negative or neutral in order to make analysis more precise and easy to understand. And also generate an automatic table stating day wise analysis as positive, negative or neutral.

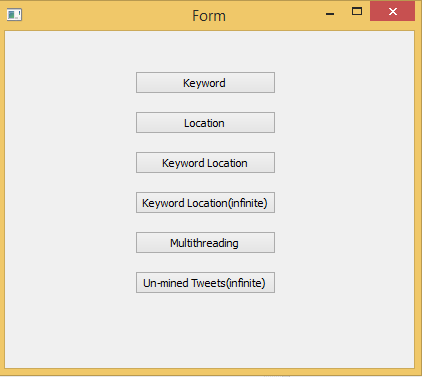


Fig.2.1.1 Form with all Modules

* 1. **Challenges**
     1. **Automating the functionality of previous system:**

The initial challenge was that the system designed using previous research was not efficient to be used for any number keyword or any location or any specified date, i.e. user has to define the keywords, location, start date, end date and it required that user must have programming skills. In order to make this thing more useful for normal people and to make surveys and analysis, it was required to design software which takes all the information at the run-time and no user modification to code is required. This made it easy for using the software and also made it efficient.

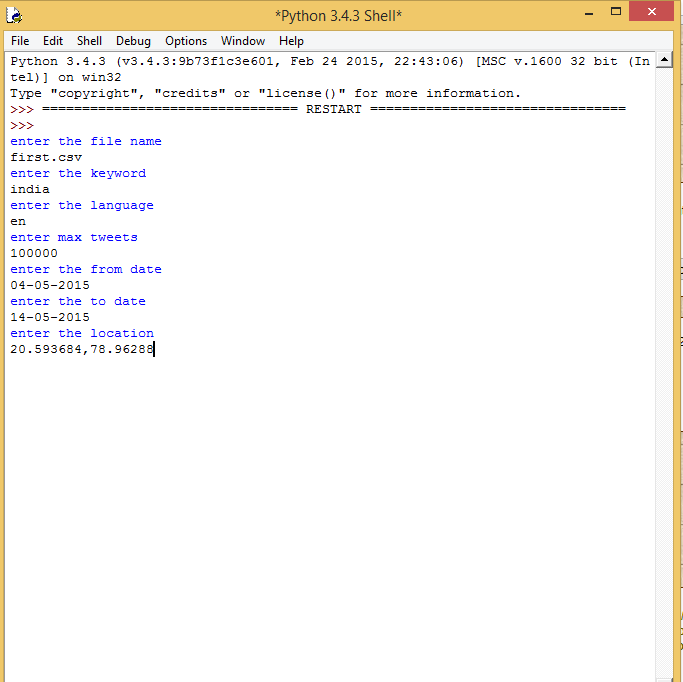


Fig. 2.2.1.1 Runtime User Input

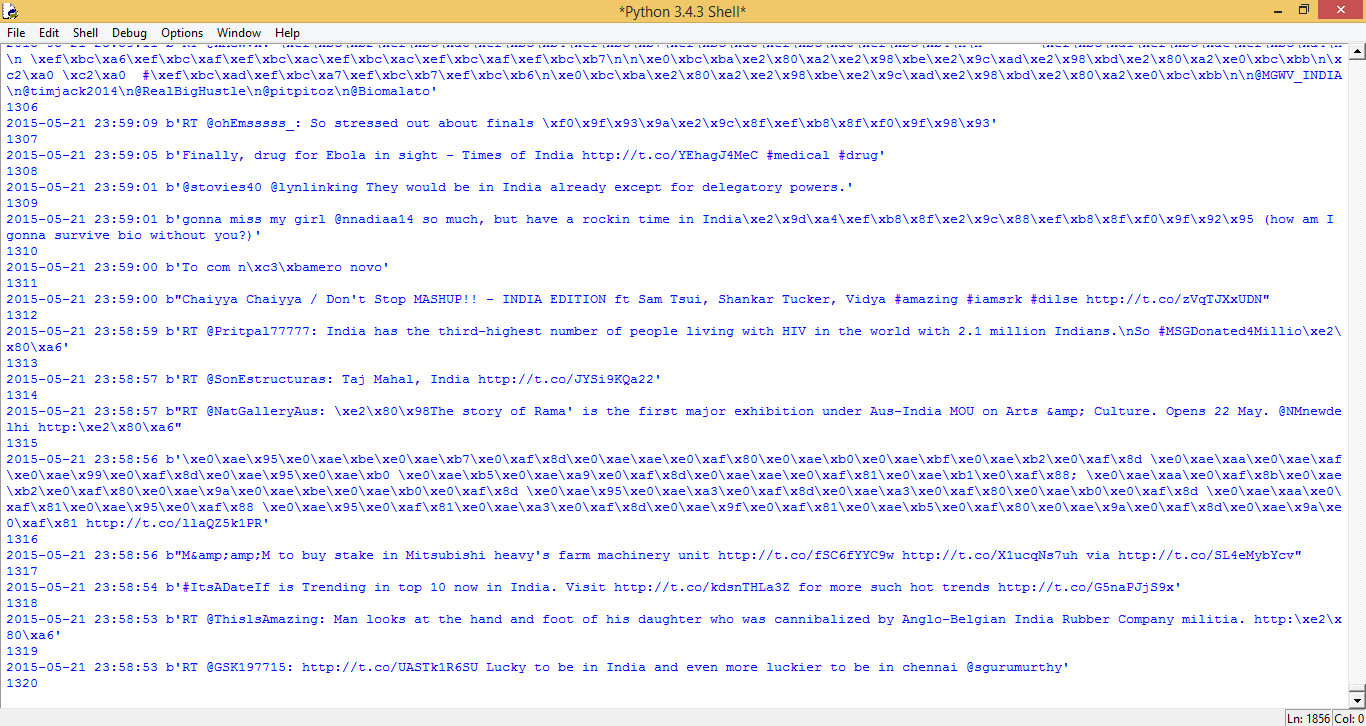


Fig. 2.2.1.2 Tweets fetched based on the input

* + 1. **Fetching of any number of tweets:**

Another challenge faced during the research was to fetch any number of tweets. In this the fetching was supposed to run for days without user intervention required. So the key problem faced was that the

* Twitter Streaming function:

**twitterStream.filter(track=[keyword],locations=location, languages=[language])**

This function can be used but the problem was that it cant operate on keyword and location simultaneously, so it can either fetch tweets based on keyword or based on location. So there was no option to make them use together.

* Twitter Search Api Function:

**api.search(q=keyword,language=language,since\_id=sinc,until=to,location=location)**

This function can be used to fetch tweets in which we can add start date, end date keyword, language and location to work together. But the problem was that it can fetch only 100 tweets at a time after which user has to restart the process. So, in-order to overcome this problem it was required to use the max-id of the fetched tweets and run it in a loop in such a way that the max-id of the last tweet fetched becomes the since id and the tweet fetching start from the last tweet fetched in previous looping and make it the reference tweet and fetch the further tweets. So this keeps on running until the tweets available gets over.

* + 1. **Graphical Representation:**

Another challenge faced during the research was to represent the sentiments of tweets graphically. This involved the representation of tweets day wise on the bar graph and also to represent the day wise sentiments of the tweets in tabular format.

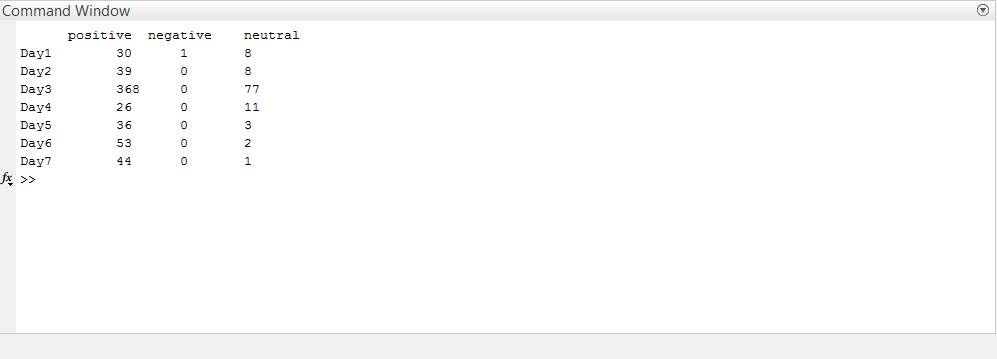


Fig. 2.2.3.1 Day wise Sentiment Analysis in tabular format

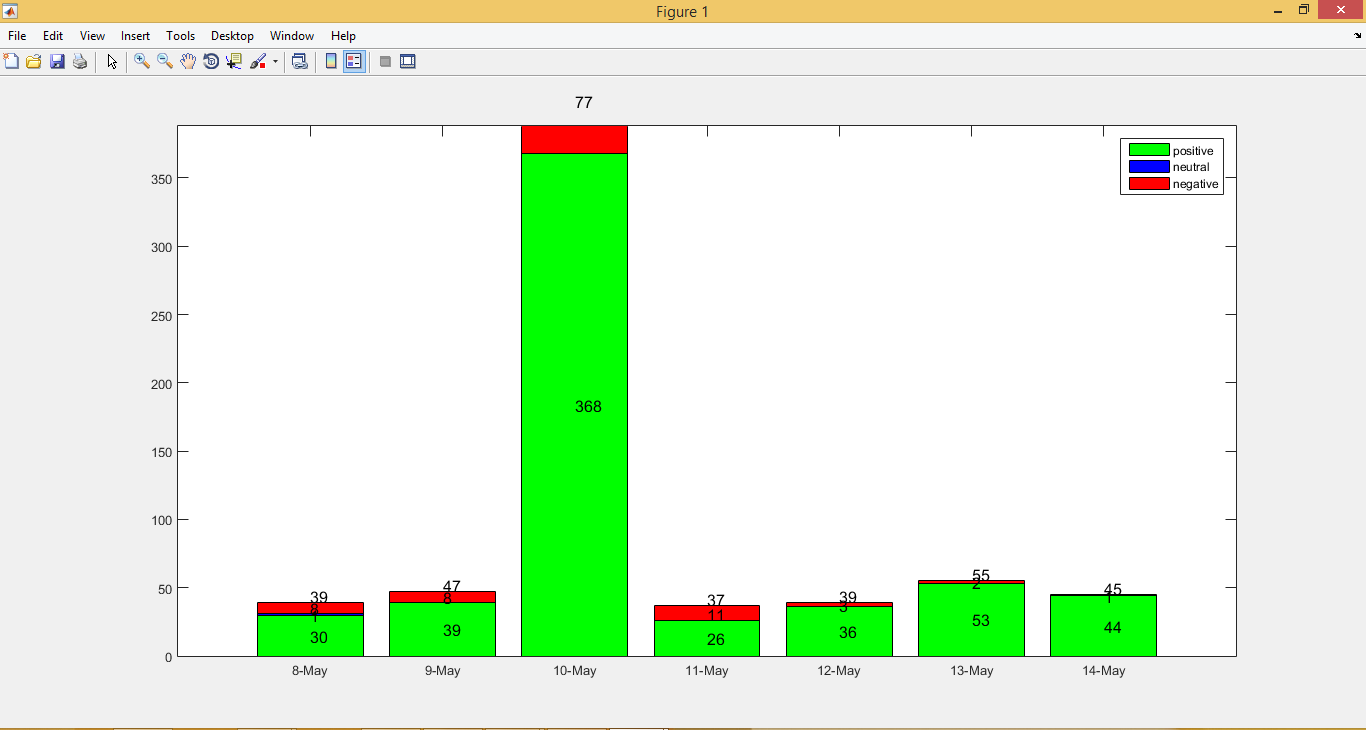


Fig. 2.2.3.2 Graphical Representation of tweets Day wise

* + 1. **Introduce Multi-Threading feature:**

Another challenge faced during the research was to introduce multi-threading feature in the software designed such that, it can fetch can be used for more than one keywords simultaneously and store them in different files. And it can be used for any number of keywords depending on number of threads defined in the program.

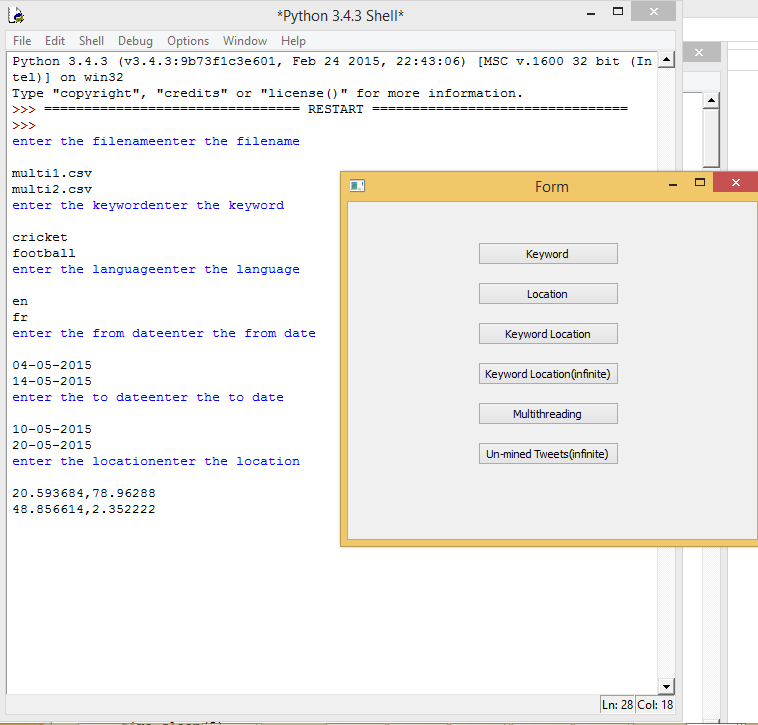


Fig. 2.2.4.1 Multithreading feature fetching tweets for two keywords as number of threads defined is two.

* + 1. **Storing the tweets fetched in mined format**

Another challenge was to mine the fetched tweets into meaning full data so that it can be used for the sentimental analysis.

Initially, the tweets fetched were bulk of data which included hashtags, texts, username, location, user-id, screen name, retweet information and many more. So in order to make it into meaningful information which can be easily used for sentiment analysis mining was required.

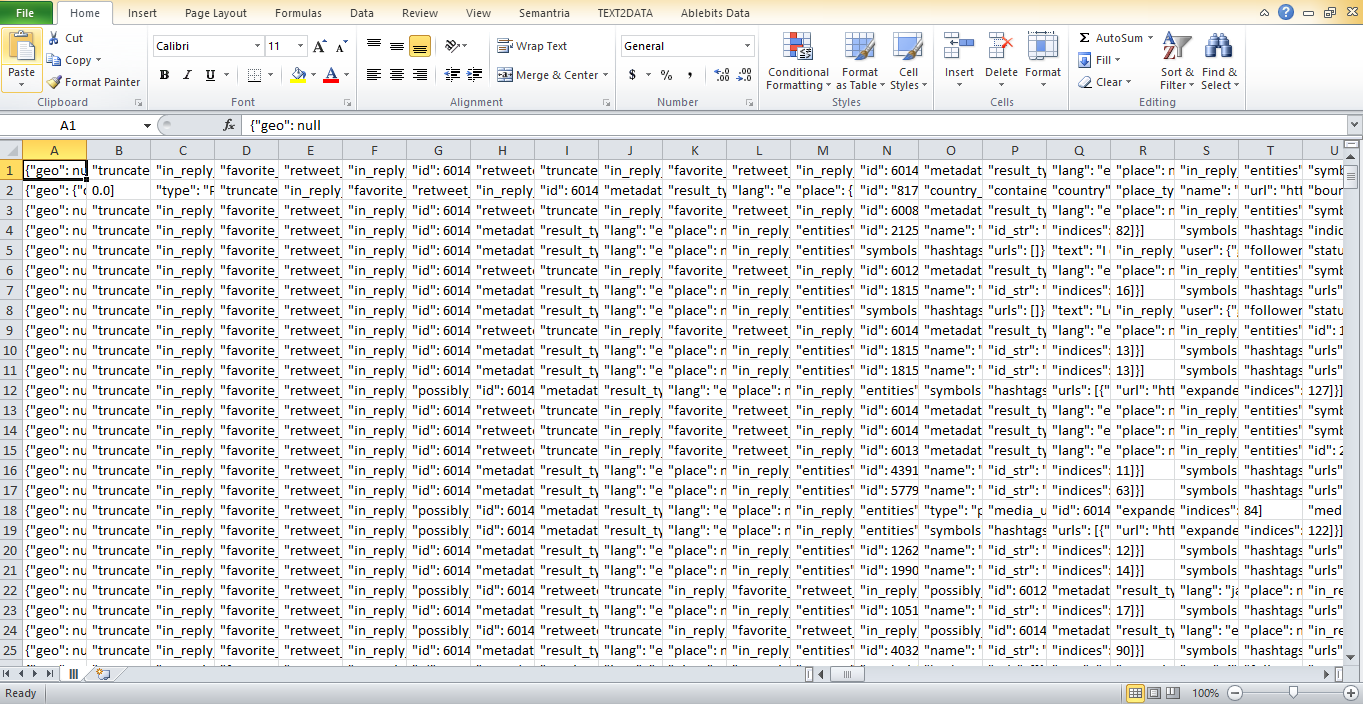


Fig.2.2.5.1 Un-mined tweets containing all information

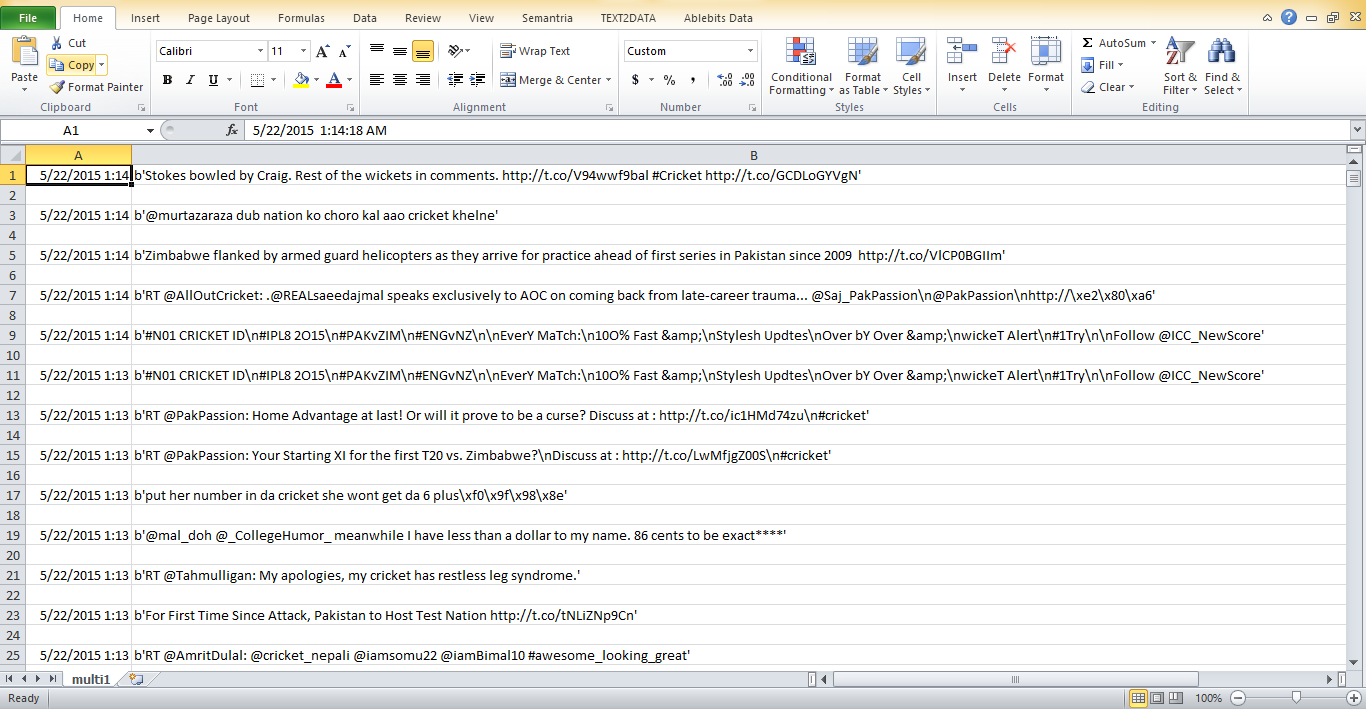


Fig. 2.2.5.2 Mined tweets containing meaningful information

* 1. **Assumptions**

The following assumptions were considered while developing the tool:

* The software could run on OS containing Python 3.4.
* Assumed that designer requires knowledge of Python 3.4.
* The graphical representation can be shown only with help of MATLAB and requires user knowledge in MATLAB.
* The software can be used to fetch data for one keyword or any number of keywords simultaneously.
* It was assumed the availability of large database to store the tweets fetched.
  1. **Architectural Specification**

Opinion Mining also called sentiment analysis is a process of finding user’s opinion towards a topic or a product. Opinion mining concludes whether user’s view is positive, negative, or neutral about product, topic, event etc. Opinion mining and summarization process involve three main steps, first is Opinion Retrieval, Opinion Classification and Opinion Summarization. Review Text is retrieved from review websites. Opinion text in blog, reviews, comments etc. contains subjective information about topic. Reviews classified as positive or negative review. Opinion summary is generated based on features opinion sentences by considering frequent features about a topic.

* + 1. **Opinion Retrieval**

It is the process of collecting review text from review websites. Different review websites contain reviews for products, movies, hotels and news. Information retrieval techniques such as web crawler can be applied to collect the review text data from many sources and store them in database. This step involves retrieval of reviews, micro-blogs, and comments of user.

* + 1. **Opinion Classification**

Primary step in sentiment analysis is classification of review text. Given a review document D = {d1.....dn} and a predefined categories set C = {positive, negative}, sentiment classification is to classify each di in D, with a label expressed in C. The approach involves classifying review text into two forms namely positive and negative. Machine learning and lexicon based approach is more popular Opinion Summarization. Summarization of opinion is a major part in opinion mining process. Summary of reviews provided should be based on features or subtopics that are mentioned in reviews. Many work have been done on summarization of product reviews. The opinion summarization process mainly involves the following two approaches. Feature based summarization a type summarization involves finding of frequent terms (features) that are appearing in many reviews. The summary is presented by selecting sentences that contain particular feature information. Opinion Mining which says how the input is being classified on various steps to summarize the reviews.

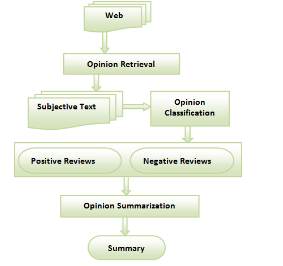
****

Fig. 2.4.1 Architectural Analysis

* 1. **Hardware Requirements**
* Any desktop computer with Windows 7 or above(32-bit/64-bit).
* Browser: Internet Explorer 8 or above, Chrome, Firefox, Safari, Opera.
* Desktop/Laptop - Intel based with Pentium IV processor or above.
  1. **Software Requirements**
* **Front-end:** QT designer.
* **Back-end:** Python 3.4, MATLAB.
* **Other software’s:** CSV file reader, Text2Data software.
  1. **Project Schedule**

The development ecosystem in this project follows the agile method of development. There are repeated iterations with deliverables at the end of each week. Following are two Gantt charts representing the activity flow during the research project:

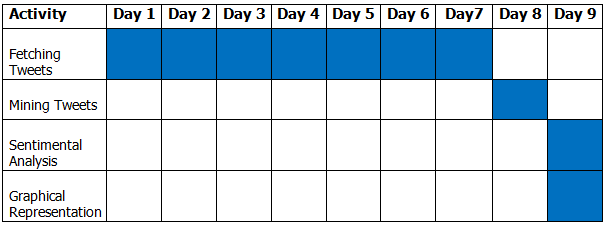


Table. 2.7.1 Gantt Chart representing the activity flow for a week.

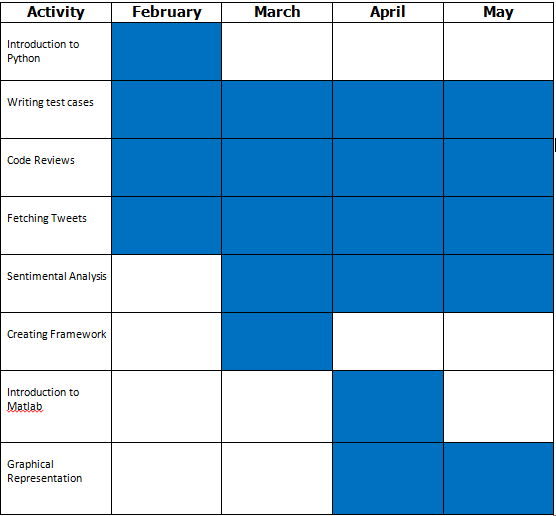


Table. 2.7.2 Gantt Chart for months of February to May, tracing progress of the project.

* 1. **Work Breakdown Structure**

This is a hierarchical and incremental decomposition of the project into phases, deliverables and packages. The representation of the WBS as a tree to deliver objectives is shown below.

Fig. 2.8.1 Work Breakdown Structure

**Un-Mined**

**Multi-threading**

**Keyword & Location**

**(infinite)**

**Keyword & Location**

**(Max Tweets)**

**Location**

**Keyword**

**User Interface**

**QT designer**

Number of threads required

Max number of tweets:

Max\_Tweets

Database

(After analysis)

Keyword

Language

Location

Since Date

Until Date

**Attributes**

DataBase

(CSV file)

Mining Tweets

Sentimental Analysis

**SAVE**

Graphical Representation and Analysis Representation

**CHAPTER 3**

**LITERATURE SURVEY AND REVIEW**

**3.1 Literature Survey**

The project involved the study of various python features and researching the most effective ways to implement them. The technologies in use are – Python, QT Designer, MATLAB. The documentation for each has been studied and constantly referenced throughout the project development.

* **Python:** Python is a widely used general-purpose, high-level programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java. The language provides constructs intended to enable clear programs on both a small and large scale. Python supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library. There are various python modules used during this project which are as follows:
* **Tweepy Module:** Tweepy is open-sourced and enables Python to communicate with Twitter platform and use its API.
* **Streaming Module:** Streaming module is open-sourced and enables Python to stream through the twitter in order to fetch tweets.
* **Threading Module:** Threading module is open-sourced and enables Python to use multi-threading and define threads in program.
* **CSV Module:** CSV module is open-sourced and enables Python to open, read, write and append data in .csv file.
* **OS Module:** The OS module in Python provides a way of using operating system dependent functionality. The functions that the OS module provides allows you to interface with the underlying operating system that Python is running on – be that Windows, Mac or Linux.
* **PyQt4 Module:** It is for exposing much of the functionality of QT to python. We use QtCore and QtGui libraries of PyQt4. QtCore- Core non-GUI classes used by other modules. QtGui- Graphical User Interface implementation.
* **JSONPICKLE Module:** jsonpickle is a Python library for serialization and deserialization of complex Python objects to and from JSON.
* **Time Module:** Time module is used to get current time from the system.
* **SYS Module:** The sys module provides information about constants, functions and methods of the Python interpreter.
* **QT Designer:**Qt Designer is Qt's tool for designing and building graphical user interfaces (GUIs) from Qt components. You can compose and customize your widgets or dialogs in a what-you-see-is-what-you-get (WYSIWYG) manner, and test them using different styles and resolutions.Widgets and forms created with Qt Designer integrated seamlessly with programmed code, using Qt's signals and slots mechanism, that lets you easily assign behavior to graphical elements. All properties set in Qt Designer can be changed dynamically within the code. Furthermore, features like widget promotion and custom plugins allow you to use your own components with Qt Designer.
* **MATLAB:** MATLAB (matrix laboratory) is a multi-paradigm numerical computing environment and fourth-generation programming language. Developed by MathWorks, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, Fortran and Python.
  1. **Literature Review**

The Theory and Documentation for the above mentioned topics were studied thoroughly and referenced continuously throughout the development of this project. The documentation gave us the knowledge and enabled us to complete this project successfully on time. These helped us in understanding the concepts and methodologies and implement them for the development of this project.

**CHAPTER 4**

**SYSTEM DESIGN**

* 1. **High Level Design**

Below is the representation for each module which was made prior to the start of development to get an idea of how the tool should look and also to streamline the process of development.

* + 1. **Master Page**

This page consists of all modules inherited in it. By clicking on the options can run the specific module and use its functionality.

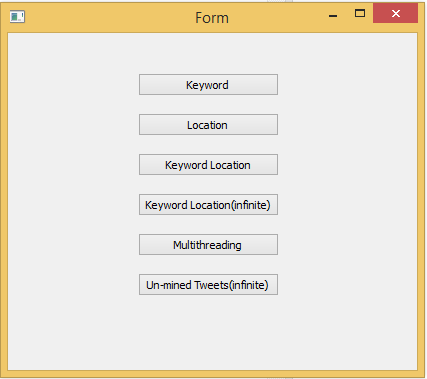


Fig.4.1.1.1 Master Page.

* + - 1. **Keyword Module**

On clicking the keyword button, input options are generated on the **python shell** where user inputs the keyword and the language in which user wants to fetch the data.

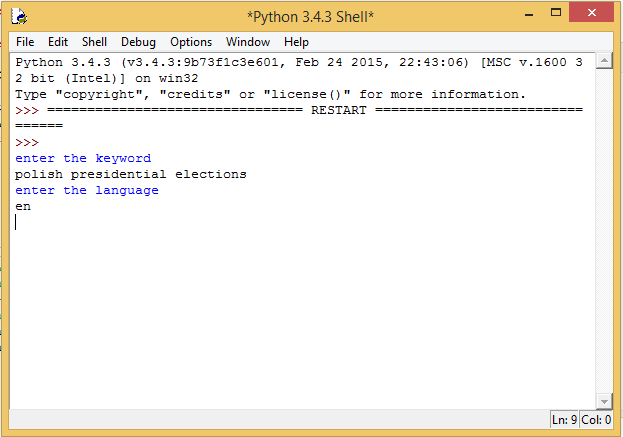


Fig. 4.1.1.1.1 Keyword Module Options

* + - 1. **Location Module**

On clicking the location button, input options are generated on the python shell where user inputs the location and the language in which user wants to fetch the data.

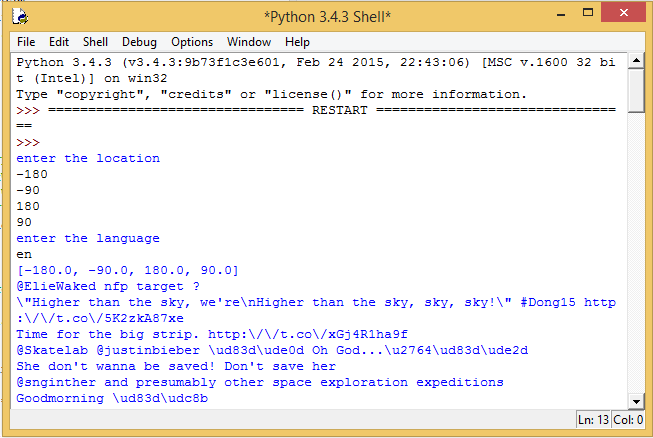


Fig. 4.1.1.2.1 Location Module Options

* + - 1. **Keyword Location Module**

On clicking the Keyword Location button, input options are generated on the python shell where user inputs the keyword, the filename, number of tweets required, start date for fetching tweets, end date for fetching tweets, the location and the language in which user wants to fetch the data.

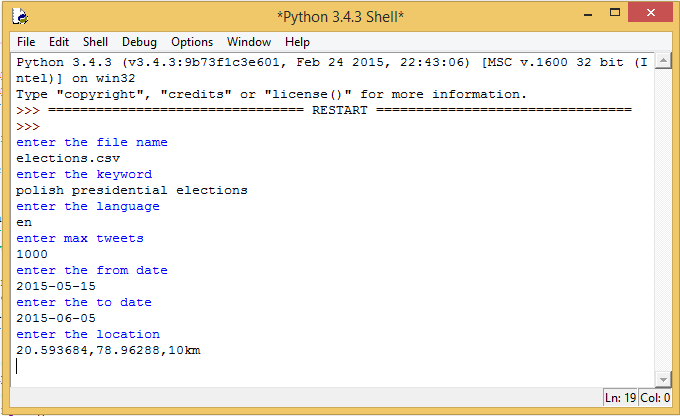


Fig.4.1.1.3.1 Keyword Location Module Options

* + - 1. **Keyword Location(infinite) Module**

On clicking the Keyword Location button, input options are generated on the python shell where user inputs the keyword, the filename, start date for fetching tweets, end date for fetching tweets, the location and the language in which user wants to fetch the data.

**Note:** This is similar to **Keyword Location Module** and the only difference between the two is that in **Keyword Location (infinite) Module** “number of tweets required” need not to be mentioned because it can fetch any number of tweets available from start date to end date.

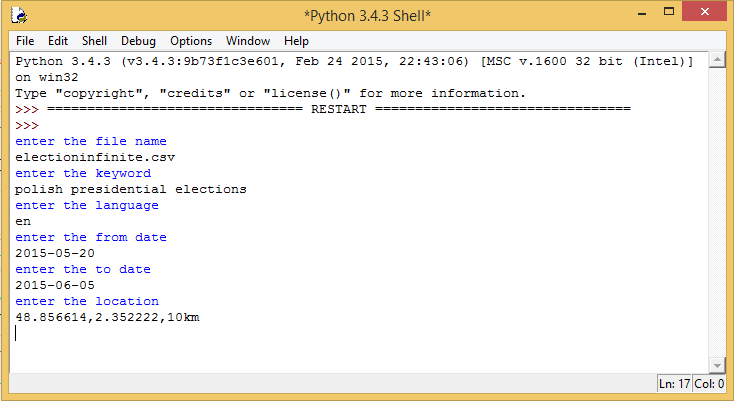


Fig. 4.1.1.4.1 Keyword Location (infinite) Module Options.

* + - 1. **Multithreading Module**

On clicking Multithreading button, input options are generated on the python shell where user inputs the keyword, the filename, start date for fetching tweets, end date for fetching tweets, the location and the language in which user wants to fetch the data. The only advantage of this is that we can do fetching simultaneously for any number of keywords depending upon the number of threads included in the module.

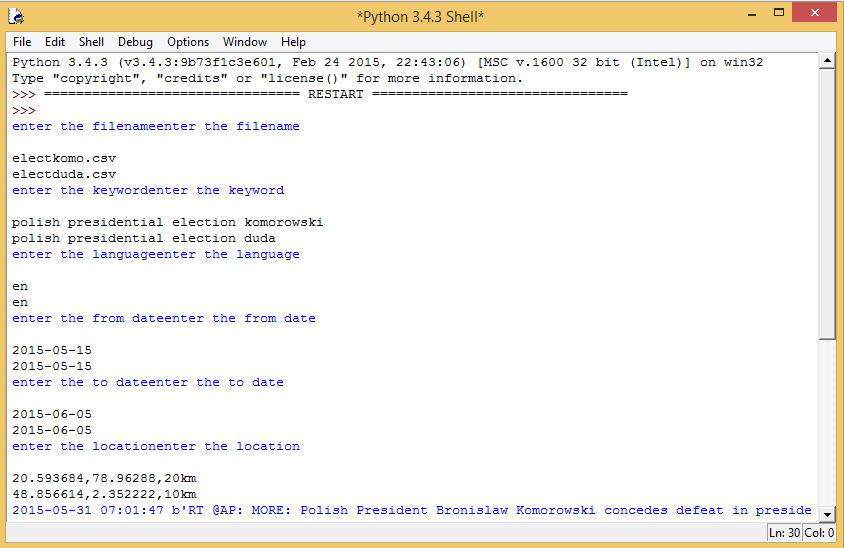


Fig.4.1.1.5.1 Multithreading Module Options

* + - 1. **Un-mined Tweets(infinite) Module**

On clicking on Un-mined Tweets (infinite) button, input options are generated on the python shell where user inputs the keyword, the filename, start date of tweets, end date of tweets, the location and the language in which user wants to fetch the data.

**Note:** The only advantage of this module is that the tweets fetched are un-mined, that is, tweets contain all the information like user id, hash tags, date, time, location, number of re-tweets, etc.

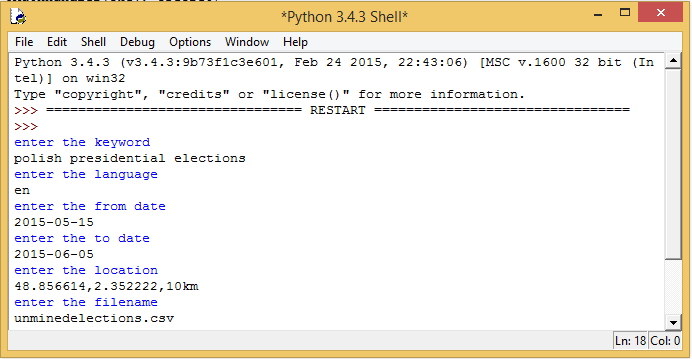


Fig.4.1.1.6.1 Un-mined Tweets (infinite) Module Options

* 1. **Low-Level Design**

The UI of the project was developed based on the modules listed above. Below are the screenshots showing the UI developed for the project.

* + 1. **Mined CSV Files**

These files are of **CSV** format and they contained all the data of the tweets fetched which include one the date and time column and other column for tweets which contains the data.

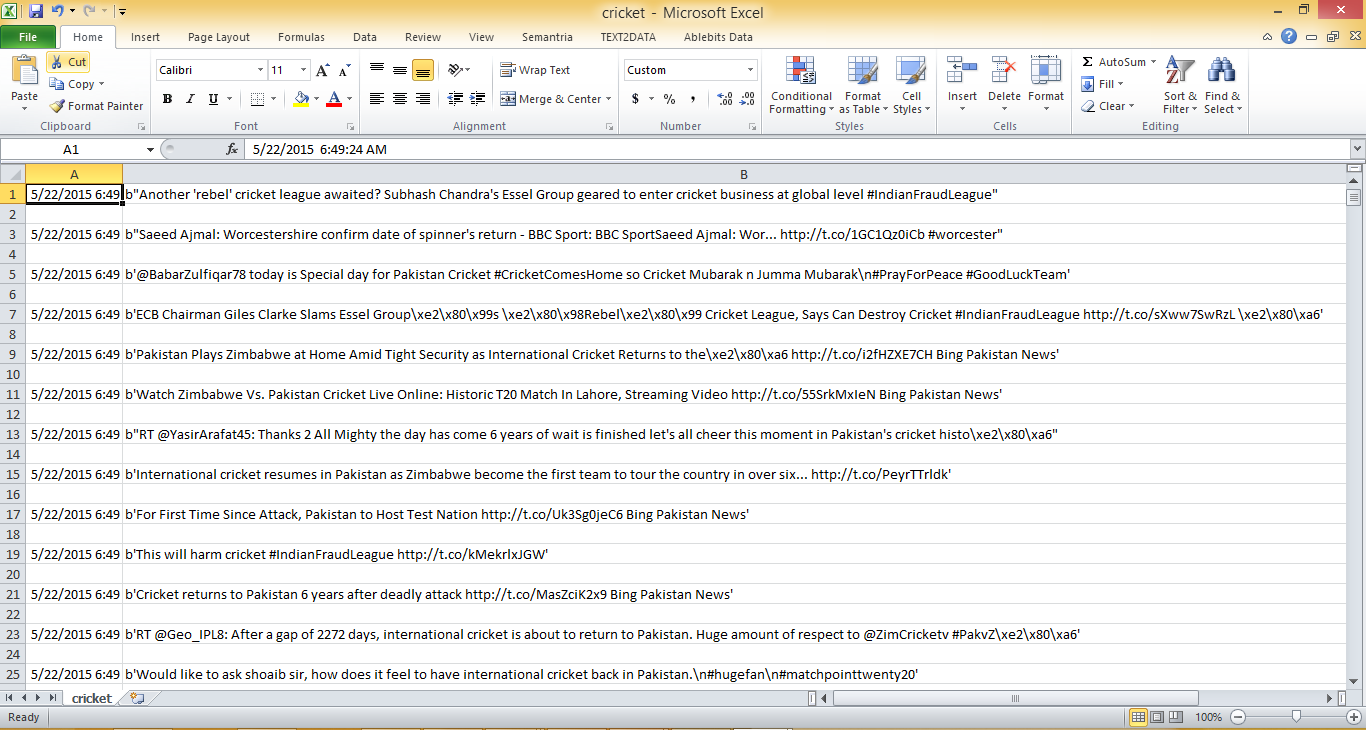


Fig. 4.2.1.1 Mined CSV File

* + 1. **Un-mined CSV Files**

These files are of **CSV** format and in the file the tweets are un-mined, that is, they contain all the information related to the tweet which includes user-id, hash-tags, tweet data, number of re-tweets, date and time, location, etc.

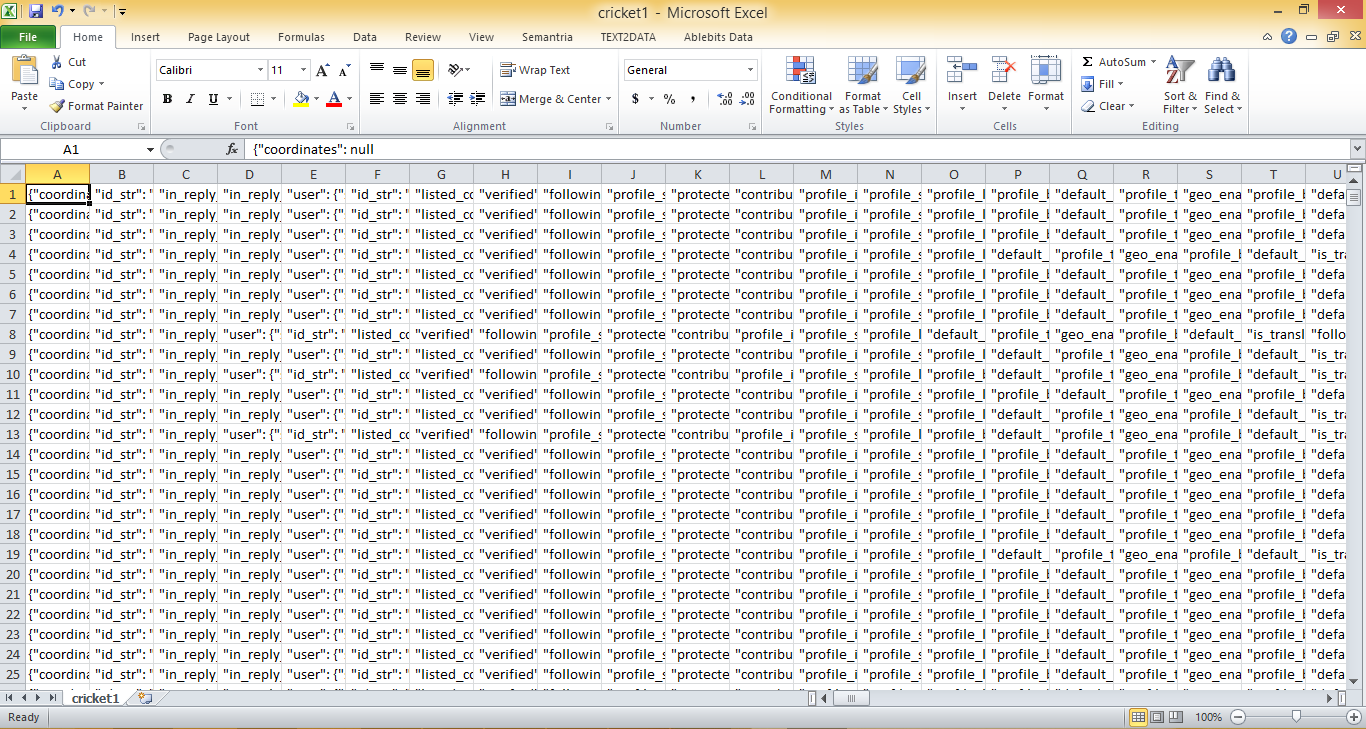


Fig.4.2.2.1 Un-mined CSV file

* + 1. **Sentimental Analysis of the Tweets**

Once the tweets are fetched, then analysis of tweets is done and it is done by matching those tweets with the online dictionary to check whether they are positive, negative or neutral. The screenshot shows the sentimental analysis of the polish presidential elections.

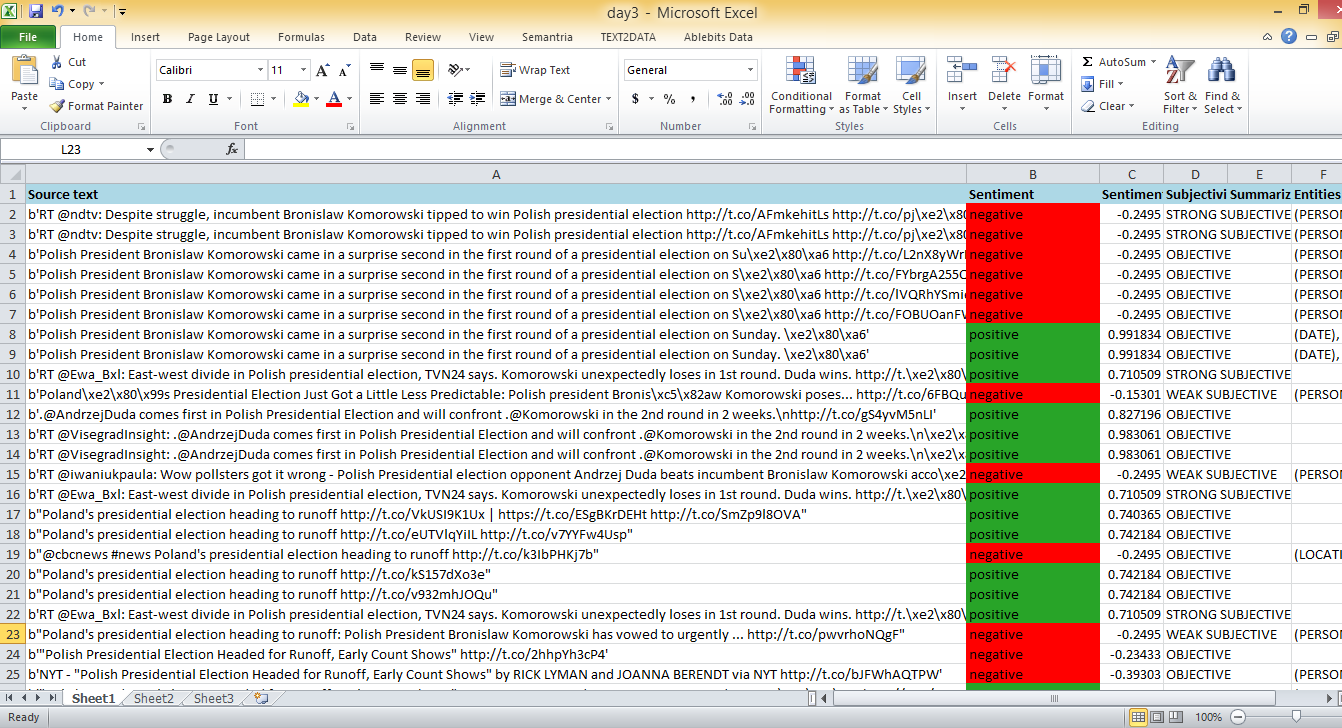


Fig.4.2.3.1 Sentimental Analysis of the Tweets Fetched

* + 1. **Graphical Representation of the Analyzed Tweets**

The tweets whose sentiments are analyzed are then used for the graphical representation. Graphs generated can be a bar graph or line graph based on the user requirement and also details are displayed on the command window accordingly.

* + - 1. **Representation of Sentiments using Bar Graph**

This includes the representation of the analyzed tweets is represented as Bar Graph which includes numerical representation of the tweets which are positive, negative and neutral.

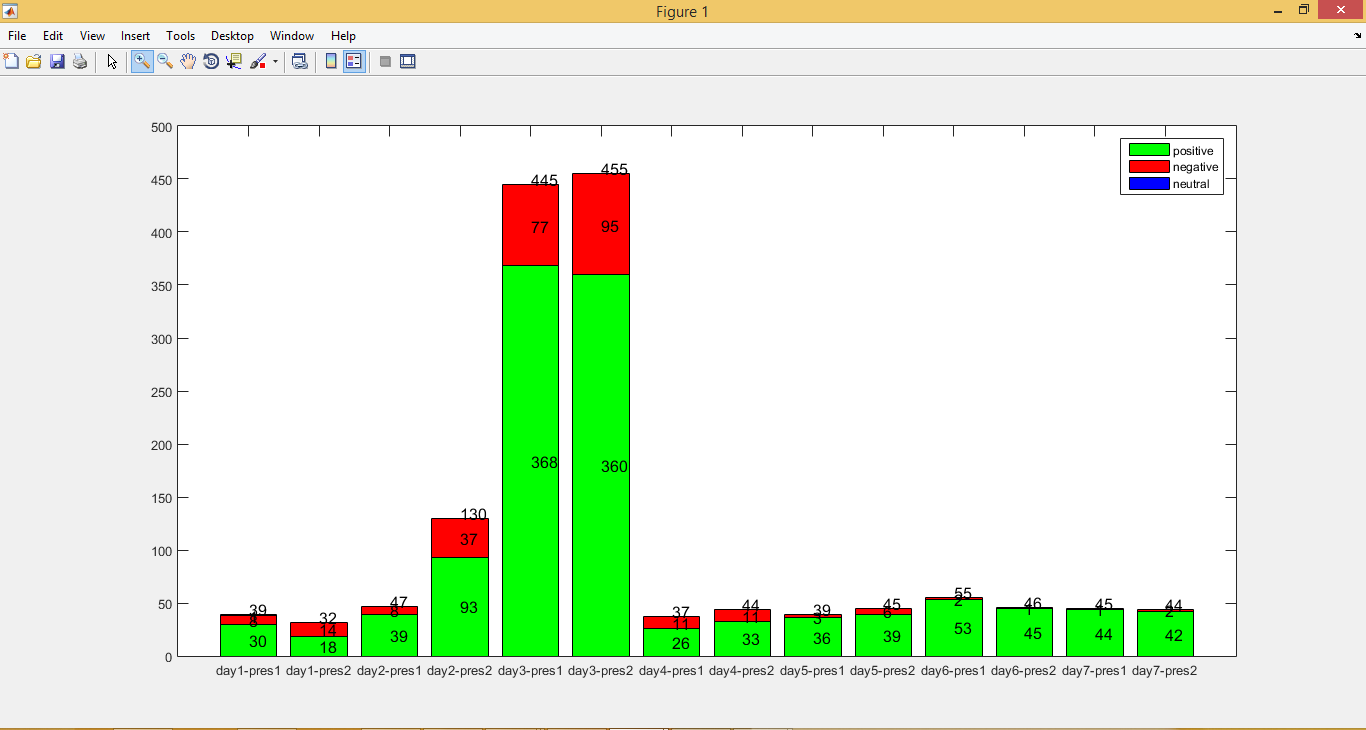


Fig. 4.2.4.1.1 Bar Graph representation day wise for President-1 and President-2 and the analysis was done for one week from 08-05-2015 to 15-05-2015.

* + - 1. **Representation of Sentiments using Line Graph**

This includes the representation of the analyzed tweets is represented as Line Graph which includes numerical representation of the tweets which are positive, negative and neutral.

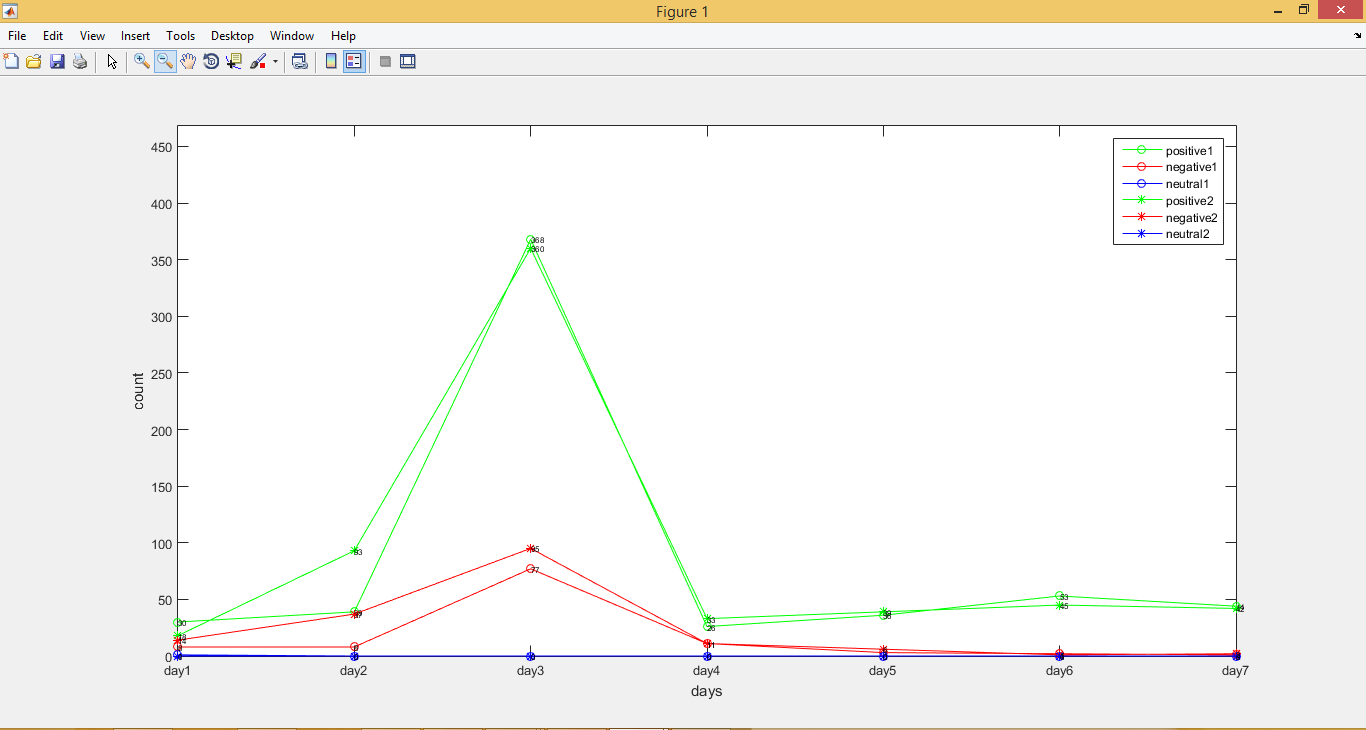


Fig. 4.2.4.2.1 Line Graph representation day wise for President-1 and President-2 and the analysis was done for one week from 08-05-2015 to 15-05-2015.

* + - 1. **Representation of Sentiments in Tabular Format**

This is the tabular representation of the analyzed tweets, which includes table stating the number of positive tweets, negative tweets and neutral tweets.

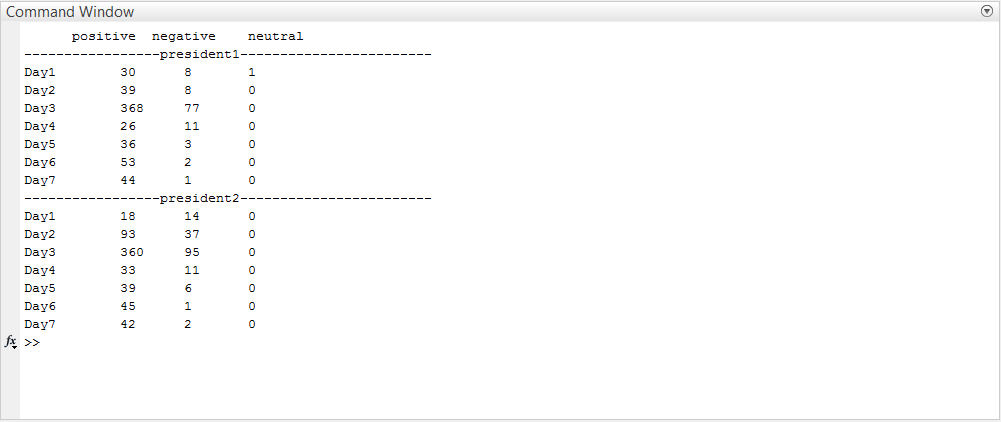


Fig. 4.2.4.3.1 Tabular Representation of the analyzed tweets as positive, negative or neutral for presidents day-wise.

* 1. **Codes and Standards**

|  |  |
| --- | --- |
| **Classes and Objects** | The class statement creates a new class definition. The name of the class immediately follows the keyword class followed by a colon as follows −  class ClassName:  'Optional class documentation string'  class\_suite   * The class has a documentation string, which can be accessed via ClassName.\_\_doc\_\_. * The class\_suite consists of all the component statements defining class members, data attributes and functions. |
| *//Example of the class* |
| class Employee:  'Common base class for all employees'  empCount = 0  def \_\_init\_\_(self, name, salary):  self.name = name  self.salary = salary  Employee.empCount += 1    def displayCount(self):  print ("Total Employee %d" % Employee.empCount)  def displayEmployee(self):  print ("Name : ", self.name, ", Salary: ", self.salary)   * The variable *empCount* is a class variable whose value is shared among all instances of a this class. This can be accessed as *Employee.empCount* from inside the class or outside the class. * The first method *\_\_init\_\_()* is a special method, which is called class constructor or initialization method that Python calls when you create a new instance of this class. * You declare other class methods like normal functions with the exception that the first argument to each method is *self*. Python adds the *self* argument to the list for you; you do not need to include it when you call the methods.   To create instances objects of a class, you call the class using class name and pass in whatever arguments its *\_\_init\_\_* method accepts.  "This would create first object of Employee class"  emp1 = Employee("Zara", 2000)  "This would create second object of Employee class"  emp2 = Employee("Manni", 5000) |
| **Interfaces** | Class defined for interface is Ui\_Form.  The class must include functions:   * \_\_init\_\_() * setupUi() * retranslateUi()   //*Example for interface*  class Ui\_Form(QtGui.QWidget):  def \_\_init\_\_(self):  *#include the function definition*  def setupUi(self, Form):  *#include the function definition*  def retranslateUi(self, Form):  *#include the function definition*  if \_\_name\_\_ == '\_\_main\_\_':  app=QtGui.QApplication(sys.argv)  ex = Ui\_Form()#start\_time,time\_limit=300)  ex.show()  *#main() function calling the form to disply GUI.* |
| **Derived Class** | The name BaseClassName must be defined in a scope containing the derived class definition.  //*Example of derived class*  class Parent(object):  def \_\_init\_\_(self): #public data members representation  pass  def \_protected(self):  pass  def \_\_private(self):  pass  class Child(Parent):  def foo(self):  self.\_protected() # This works  def bar(self):  self.\_\_private() # This doesn't work |
|  |  |

Table 4.3.1 Python Coding Standards

* 1. **Constraints and Tradeoffs**
* Setting up system for fetching any number of mined tweets:

We developed the application such that it fetches all the existing tweets related to the keyword from start date to end date.

* Analyzing sentiments of the tweets fetched:

We analyzed the tweets based to get the sentiments of the user posted the tweet. This helped in opinion analysis of people.

* Getting mined tweets for bulk data:

We tried to mine the tweets in such a way that database contains only the required details required for sentimental analysis. Mining the tweets minimizes the storage space and also makes data clear and easy to understand.

* MATLAB vs Python:

Python can be used for graphical representation but it would have made things more complex. So instead of using Python we used MATLAB for the graphical representation of the analyzed tweets. As in MATLAB there are various predefined functions for graph generation which made it easy to use and analyze tweets more efficiently and easily.

* 1. **Test Case Generation**

Testing of the modules is done in python. Each of the modules that are inherited in the final framework is tested separately. Each of the modules functionality is defined separately and tested separately in python 3.4 and simultaneous modifications are done to overcome the problems faced during testing.

Classes are the only ones that return some values or set values to attributes. Hence, unit tests are written are written for the methods of these classes. Test cases are usually written to test the return parameter of any file and check if the functions are going to return those values are not. This gives an idea about the conformance of any function and checks the validity of the return. In a UI intensive application, there is very little scope for checking the return using Test cases. Rather than checking using test cases, there are tests for usability, conformance with standards, responsive behavior and code portability. Despite these, the test cases are written for several functions, albeit to verify rudimentary behavior.

**CHAPTER 5**

**SYSTEM IMPLEMENTATION**

* 1. **Code for Application Development**

The GUI and the fetching of tweets are done using Python 3.4 coding. The graphical representation of the analyzed tweets is done using MATLAB coding.

* + 1. **Python Code**

from PyQt4 import QtCore, QtGui

import sys

from tweepy import Stream

from tweepy import OAuthHandler

from streaming import StreamListener

import time

import threading

import tweepy

import jsonpickle

import os

import csv

start\_time=time.time()

ckey = 'fR5FnN3RmBZbAUNrdrGSjC40S'

csecret = 'neFWC5k1jhxWAUJ8jOaVbgjR2EAsskaCVJWBtwZzwxQboKI9GS'

atoken = '3088495247-4WfVTQSZBfT5DbT6tOYzDMAuhINfgUaBhG0ZQS4'

asecret = '0P2mg7tw8aUYqaeJ7jtaJLXqvnGDQBUEfkwy6vgoEIWRI'

auth = tweepy.OAuthHandler(consumer\_key=ckey, consumer\_secret=csecret)

api = tweepy.API(auth)

lock=threading.Lock()

def key():

print("enter the keyword")

keyword=input()

print("enter the language")

language=input()

try:

auth = OAuthHandler(ckey, csecret)

auth.set\_access\_token(atoken, asecret)

twitterStream = Stream(auth, listener(start\_time,time\_limit=300))

twitterStream.filter( track=[keyword], languages=[language])

except Exception as e:

time.sleep(5)

def loc():

locat=[]

print("enter the location")

locat1=input()

loca1=float(locat1)

locat2=input()

loca2=float(locat1)

locat3=input()

loca3=float(locat1)

locat4=input()

loca4=float(locat1)

locat.append(locat1)

locat.append(locat2)

locat.append(locat3)

locat.append(locat4)

print("enter the language")

language=input()

locat=list(map(float,locat))

print(locat)

auth = OAuthHandler(ckey, csecret)

auth.set\_access\_token(atoken, asecret)

twitterStream = Stream(auth, listener1(start\_time,time\_limit=300))

twitterStream.filter( languages=[language], locations=locat)

def lockey():

print("enter the file name")

filename=input()

csvFile = open(filename, 'a')

#Use csv Writer

csvWriter = csv.writer(csvFile)

start\_time=time.time()

print("enter the keyword")

keyword=input()

print("enter the language")

language=input()

print("enter max tweets")

maxtw=input()

max\_tweets=int(maxtw)

print("enter the from date")

sinc=input()

print("enter the to date")

too=input()

print("enter the location")

location=input()

count=1

c=1

'''t=0

a=2014

while t < 10:

auth = tweepy.OAuthHandler(consumer\_key=ckey, consumer\_secret=csecret)

api = tweepy.API(auth)

results = api.search(q=keyword,lang=language,geo='48.85681, 2.34760,15km',since\_id=a)

a=resu.getMaxId()

t=t+1

'''

#max\_tweets=1000

auth = tweepy.OAuthHandler(consumer\_key=ckey, consumer\_secret=csecret)

api = tweepy.API(auth)

searched\_tweets = [status for status in tweepy.Cursor(api.search, q=keyword,language=language,since\_id=sinc,until=too,location=location).items(max\_tweets)]

searched\_tweets = []

last\_id = -1

while len(searched\_tweets) < max\_tweets:

count = max\_tweets - len(searched\_tweets)

try:

new\_tweets = api.search(q=keyword, count=count, max\_id=str(last\_id - 1))

if not new\_tweets:

break

searched\_tweets.extend(new\_tweets)

last\_id = new\_tweets[-1].id

except tweepy.TweepError as e:

# depending on TweepError.code, one may want to retry or wait

# to keep things simple, we will give up on an error

time.sleep(30)

continue

for tweets in searched\_tweets:

print (c,tweets.created\_at.date,tweets.text.encode('utf-8'))

c=c+1

csvWriter.writerow([tweets.created\_at, tweets.text.encode('utf-8')])

#print tweet.created\_at, tweet.text

csvFile.close()

def lockey1():

counter=0

print("enter the file name")

filename=input()

csvFile = open(filename, 'a')

#Use csv Writer

csvWriter = csv.writer(csvFile)

start\_time=time.time()

print("enter the keyword")

keyword=input()

print("enter the language")

language=input()

print("enter the from date")

sinc=input()

print("enter the to date")

too=input()

print("enter the location")

location=input()

auth = tweepy.OAuthHandler(consumer\_key=ckey, consumer\_secret=csecret)

api = tweepy.API(auth)

count=1

for tweet in tweepy.Cursor(api.search,q=keyword,language=language,location=location,since\_id=sinc).items():

try:

#Write a row to the csv file/ I use encode utf-8

csvWriter.writerow([tweet.created\_at, tweet.text.encode('utf-8')])

print(tweet.created\_at, tweet.text.encode('utf-8'))

counter = counter + 1

print(counter)

if counter == 2000:

time.sleep(60\*20) # wait for 20 min everytime 2,000 tweets are extracted

counter = 0

continue

except tweepy.error.TweepyError as e:

time.sleep(60\*20)

continue

csvFile.close()

def multi1():

t=d1()

t1=d1()

lock.acquire()

try:

t.start()

finally:

lock.release()

lock.acquire()

try:

t1.start()

finally:

lock.release()

t.sleep2()

t1.sleep()

def unmined1():

print("enter the keyword")

keyword=input()

print("enter the language")

language=input()

print("enter the from date")

sinc=input()

print("enter the to date")

too=input()

print("enter the location")

location=input()

print("enter the filename")

filename=input()

#searchQuery = input() # this is what we're searching for

maxTweets = 10000000 # Some arbitrary large number

tweetsPerQry = 100 # this is the max the API permits

fName = filename # We'll store the tweets in a text file.

# If results from a specific ID onwards are reqd, set since\_id to that ID.

# else default to no lower limit, go as far back as API allows

sinceId = None

# If results only below a specific ID are, set max\_id to that ID.

# else default to no upper limit, start from the most recent tweet matching the search query.

max\_id = 0

tweetCount = 0

auth = tweepy.OAuthHandler(consumer\_key=ckey, consumer\_secret=csecret)

api = tweepy.API(auth)

print("Downloading max {0} tweets".format(maxTweets))

with open(fName, 'w') as f:

while tweetCount < maxTweets:

try:

if (max\_id <= 0):

if (not sinceId):

new\_tweets = api.search(q=keyword,language=language,location=location,since\_id=sinc, count=tweetsPerQry)

else:

new\_tweets = api.search(q=keyword,language=language,location=location,since\_id=sinc, count=tweetsPerQry)

else:

if (not sinceId):

new\_tweets = api.search(q=keyword,language=language,location=location,since\_id=sinc, count=tweetsPerQry,max\_id=str(max\_id - 1))

else:

new\_tweets = api.search(q=keyword,language=language,location=location,since\_id=sinc, count=tweetsPerQry,max\_id=str(max\_id - 1))

if not new\_tweets:

print("No more tweets found")

break

for tweet in new\_tweets:

f.write(jsonpickle.encode(tweet.\_json, unpicklable=False) +'\n')

tweetCount += len(new\_tweets)

print("Downloaded {0} tweets".format(tweetCount))

max\_id = new\_tweets[-1].id

except tweepy.TweepError as e:

# Just exit if any error

print("some error : " + str(e))

break

print ("Downloaded {0} tweets, Saved to {1}".format(tweetCount, fName))

try:

\_fromUtf8 = QtCore.QString.fromUtf8

except AttributeError:

def \_fromUtf8(s):

return s

try:

\_encoding = QtGui.QApplication.UnicodeUTF8

def \_translate(context, text, disambig):

return QtGui.QApplication.translate(context, text, disambig, \_encoding)

except AttributeError:

def \_translate(context, text, disambig):

return QtGui.QApplication.translate(context, text, disambig)

class listener(StreamListener):

def \_\_init\_\_(self,start\_time,time\_limit=300):

self.time=start\_time

self.limit=time\_limit

'''def \_\_init\_\_data(keyword,location):

keyword=self.keyword.text()

location=self.location.text()'''

def on\_data(self, data):

while(time.time()-self.time)<self.limit:

try:

#print (data)

tweet = data.split(',"text":"')[1] .split('","source')[0]

print (tweet)

saveThis = str(time.time())+'::'+tweet

saveFile = open('file1.csv','a')

saveFile.write(tweet)

saveFile.write('\n')

saveFile.close()

return True

except BaseException (e):

print ('failed ondata,',str(e))

time.sleep(5)

exit()

def on\_error(self, status):

print (status)

class listener1(StreamListener):

def \_\_init\_\_(self,start\_time,time\_limit=300):

self.time=start\_time

self.limit=time\_limit

'''def \_\_init\_\_data(keyword,location):

keyword=self.keyword.text()

location=self.location.text()'''

def on\_data(self, data):

while(time.time()-self.time)<self.limit:

try:

#print (data)

tweet = data.split(',"text":"')[1] .split('","source')[0]

print (tweet)

saveThis = str(time.time())+'::'+tweet

saveFile = open('file.csv','a')

saveFile.write(tweet)

saveFile.write('\n')

saveFile.close()

return True

except BaseException (e):

print ('failed ondata,',str(e))

time.sleep(5)

exit()

def on\_error(self, status):

print (status)

class d1(threading.Thread):

def \_\_init\_\_(self):

super(d1,self).\_\_init\_\_()

#time.sleep(10)

def sleep(self):

time.sleep(2)

def sleep2(self):

time.sleep(2)

def run(self):

counter=0

print("enter the filename")

filename=input()

csvFile = open(filename, 'a')

#Use csv Writer

csvWriter = csv.writer(csvFile)

start\_time=time.time()

print("enter the keyword")

keyword=input()

print("enter the language")

language=input()

print("enter the from date")

sinc=input()

print("enter the to date")

too=input()

print("enter the location")

location=input()

auth = tweepy.OAuthHandler(consumer\_key=ckey, consumer\_secret=csecret)

api = tweepy.API(auth)

count=1

for tweet in tweepy.Cursor(api.search,q=keyword,language=language,location=location,since\_id=sinc).items():

try:

auth = tweepy.OAuthHandler(consumer\_key=ckey, consumer\_secret=csecret)

api = tweepy.API(auth)

#Write a row to the csv file/ I use encode utf-8

csvWriter.writerow([tweet.created\_at, tweet.text.encode('utf-8')])

print(tweet.created\_at, tweet.text.encode('utf-8'))

counter = counter + 1

#print(counter)

if counter == 2000:

time.sleep(60\*20) # wait for 20 min everytime 2,000 tweets are extracted

counter = 0

continue

except tweepy.error.TweepyError as e:

time.sleep(60\*20)

continue

except BaseException as e:

time.sleep(60\*2.5)

continue

except Exception as e:

time.sleep(60\*2.5)

continue

csvFile.close()

class Ui\_Form(QtGui.QWidget):

def \_\_init\_\_(self):#,start\_time,time\_limit=300):

#self.time=start\_time

#self.limit=time\_limit

QtGui.QWidget.\_\_init\_\_(self)

self.setupUi(self)

def setupUi(self, Form):

Form.setObjectName(\_fromUtf8("Form"))

Form.resize(409, 337)

self.keyword = QtGui.QPushButton(Form)

self.keyword.setGeometry(QtCore.QRect(130, 40, 141, 23))

self.keyword.setObjectName(\_fromUtf8("keyword"))

self.pushButton\_2 = QtGui.QPushButton(Form)

self.pushButton\_2.setGeometry(QtCore.QRect(130, 80, 141, 23))

self.pushButton\_2.setObjectName(\_fromUtf8("pushButton\_2"))

self.keyloc = QtGui.QPushButton(Form)

self.keyloc.setGeometry(QtCore.QRect(130, 120, 141, 23))

self.keyloc.setObjectName(\_fromUtf8("keyloc"))

self.keyloc1 = QtGui.QPushButton(Form)

self.keyloc1.setGeometry(QtCore.QRect(130, 160, 141, 23))

self.keyloc1.setObjectName(\_fromUtf8("keyloc1"))

self.multi = QtGui.QPushButton(Form)

self.multi.setGeometry(QtCore.QRect(130, 200, 141, 23))

self.multi.setObjectName(\_fromUtf8("multi"))

self.unmined = QtGui.QPushButton(Form)

self.unmined.setGeometry(QtCore.QRect(130, 240, 141, 23))

self.unmined.setObjectName(\_fromUtf8("unmined"))

self.retranslateUi(Form)

QtCore.QMetaObject.connectSlotsByName(Form)

def retranslateUi(self, Form):

Form.setWindowTitle(\_translate("Form", "Form", None))

self.keyword.setText(\_translate("Form", "Keyword", None))

self.pushButton\_2.setText(\_translate("Form", "Location", None))

self.keyloc.setText(\_translate("Form", "Keyword Location", None))

self.keyloc1.setText(\_translate("Form", "Keyword Location(infinite)", None))

self.multi.setText(\_translate("Form", "Multithreading", None))

self.unmined.setText(\_translate("Form", "Un-mined Tweets(infinite)", None))

if \_\_name\_\_ == '\_\_main\_\_':

app=QtGui.QApplication(sys.argv)

ex = Ui\_Form()#start\_time,time\_limit=300)

ex.show()

#keyword=keyword.get()

ex.keyword.clicked.connect(key)

ex.pushButton\_2.clicked.connect(loc)

ex.keyloc.clicked.connect(lockey)

ex.keyloc1.clicked.connect(lockey1)

ex.multi.clicked.connect(multi1)

ex.unmined.clicked.connect(unmined1)

sys.exit(app.exec\_())

* + 1. **MATLAB Code**
       1. **Code for Bar Graph Representation**

clc;

clear all;

close all;

P= 2;%number of presidents

N=7;%number of days

count=zeros(P,N,3);

for k=1:P

for j=1:N

[num, txt, raw] =xlsread(sprintf('day%d%d.xlsx',k,j));

for i= 1:length(txt(:,2)) -1

if(strcmp(txt(i+1,2),'positive'))

count(k,j,1)=count(k,j,1)+1;

elseif(strcmp(txt(i+1,2),'negative'))

count(k,j,2)=count(k,j,2)+1;

elseif(strcmp(txt(i+1,2),'neutral'))

count(k,j,3)=count(k,j,3)+1;

end

end

end

end

count1=0;

for k=1:N

for j=1:P

count1=count1+1;

for n=1:3

count\_temp(count1,n)=count(j,k,n);

end

end

end

b = bar(count\_temp,'stacked');

mycolor=[0 1 0;1 0 0;0 0 1];% u can change this to change colours

colormap(mycolor)

for i=1:P\*N

if(count\_temp(i,1)~=0)

text(i,round(count\_temp(i,1)/2),num2str(count\_temp(i,1)),'FontSize',12);%u can change font size

end

if(count\_temp(i,2)~=0)

text(i,floor(count\_temp(i,1)+count\_temp(i,2)/2),num2str(count\_temp(i,2)),'FontSize',12);

end

if(count\_temp(i,3)~=0)

text(i,floor(count\_temp(i,1)+count\_temp(i,2)+count\_temp(i,3)/2),num2str(count\_temp(i,3)),'FontSize',12);

end

text(i,count\_temp(i,1)+count\_temp(i,2)+count\_temp(i,3)+5,num2str(sum(count\_temp(i,:))),'FontSize',12);

end

set(gca,'XTickLabel',{'day1-pres1','day1-pres2','day2-pres1','day2-pres2','day3-pres1','day3-pres2','day4-pres1','day4-pres2','day5-pres1','day5-pres2','day6-pres1','day6-pres2','day7-pres1','day7-pres2',})

legend('positive', 'negative', 'neutral');%,'Location','NorthEastOutside')

%ylim([0 max(max(count))+20])

zoom on

hold on

fprintf('\t positive\tnegative\tneutral\n');

for j=1:P

fprintf('-----------------president%d------------------------\n',j);

for i =1:N

fprintf('Day%d\t\t%d\t\t%d\t\t%d\n',i,count(j,i,1),count(j,i,2),count(j,i,3));

end

end

* + - 1. **Code for Line Graph Representation**

clc;

clear all;

close all;

P= 2;%number of presidents

N=7;%number of days

count=zeros(P,N,3);

for k=1:P

for j=1:N

[num, txt, raw] =xlsread(sprintf('day%d%d.xlsx',k,j));

for i= 1:length(txt(:,2)) -1

if(strcmp(txt(i+1,2),'positive'))

count(k,j,1)=count(k,j,1)+1;

elseif(strcmp(txt(i+1,2),'negative'))

count(k,j,2)=count(k,j,2)+1;

elseif(strcmp(txt(i+1,2),'neutral'))

count(k,j,3)=count(k,j,3)+1;

end

end

end

end

figure(1)

%colours =['g','r','b'];

markers = ['o','\*','+','.','x','s','d'];

m=1;

for j=1:P

h(m)=plot(count(j,:,1),sprintf('-%sg',markers(j)));

hold on

h(m+1)=plot(count(j,:,2),sprintf('-%sr',markers(j)));

h(m+2)=plot(count(j,:,3),sprintf('-%sb',markers(j)));

xlabel('days');

ylabel('count');

for i=1:N

text (i,count(j,i,1),num2str(count(j,i,1)),'FontSize',6);

text (i,count(j,i,2),num2str(count(j,i,2)),'FontSize',6);

text (i,count(j,i,3),num2str(count(j,i,3)),'FontSize',6);

end

set(gca,'XTick',[1:1:N])

for i=1:N

temp(i,:) = sprintf('day%d',i);

end

%set(gca,'XTickLabel',['day1';'day2';'day3';'day4';'day5'])

set(gca,'XTickLabel',temp)

m=m+3;

end

ylim([0 max(max(max(count)))+100])

legend(h,'positive1', 'negative1', 'neutral1','positive2', 'negative2', 'neutral2');

fprintf('\t positive\tnegative\tneutral\n');

for j=1:P

fprintf('-----------------president%d------------------------\n',j);

for i =1:N

fprintf('Day%d\t\t%d\t\t%d\t\t%d\n',i,count(j,i,1),count(j,i,2),count(j,i,3));

end

end

* 1. **Unit Testing**

Unit testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use. Unit tests are of the pattern represented in the code below.

* + 1. **Keyword module Unit Test**

Keyword Module is tested by creating a separate unit test .py file containing the python code for fetching data from twitter using Keyword.

def key():

print("enter the keyword")

keyword=input()

print("enter the language")

language=input()

try:

auth = OAuthHandler(ckey, csecret)

auth.set\_access\_token(atoken, asecret)

twitterStream = Stream(auth, listener(start\_time,time\_limit=300))

twitterStream.filter( track=[keyword], languages=[language])

except Exception as e:

time.sleep(5)

class listener(StreamListener):

def \_\_init\_\_(self,start\_time,time\_limit=300):

self.time=start\_time

self.limit=time\_limit

'''def \_\_init\_\_data(keyword,location):

keyword=self.keyword.text()

location=self.location.text()'''

def on\_data(self, data):

while(time.time()-self.time)<self.limit:

try:

#print (data)

tweet = data.split(',"text":"')[1] .split('","source')[0]

print (tweet)

saveThis = str(time.time())+'::'+tweet

saveFile = open('file1.csv','a')

saveFile.write(tweet)

saveFile.write('\n')

saveFile.close()

return True

except BaseException (e):

print ('failed ondata,',str(e))

time.sleep(5)

exit()

def on\_error(self, status):

print (status)

* + 1. **Location Module unit test**

Location module is tested by creating a separate .py file containing the python code for fetching tweets from particular location.

def loc():

locat=[]

print("enter the location")

locat1=input()

loca1=float(locat1)

locat2=input()

loca2=float(locat1)

locat3=input()

loca3=float(locat1)

locat4=input()

loca4=float(locat1)

locat.append(locat1)

locat.append(locat2)

locat.append(locat3)

locat.append(locat4)

print("enter the language")

language=input()

locat=list(map(float,locat))

print(locat)

auth = OAuthHandler(ckey, csecret)

auth.set\_access\_token(atoken, asecret)

twitterStream = Stream(auth, listener1(start\_time,time\_limit=300))

twitterStream.filter( languages=[language], locations=locat)

class listener1(StreamListener):

def \_\_init\_\_(self,start\_time,time\_limit=300):

self.time=start\_time

self.limit=time\_limit

'''def \_\_init\_\_data(keyword,location):

keyword=self.keyword.text()

location=self.location.text()'''

def on\_data(self, data):

while(time.time()-self.time)<self.limit:

try:

#print (data)

tweet = data.split(',"text":"')[1] .split('","source')[0]

print (tweet)

saveThis = str(time.time())+'::'+tweet

saveFile = open('file.csv','a')

saveFile.write(tweet)

saveFile.write('\n')

saveFile.close()

return True

except BaseException (e):

print ('failed ondata,',str(e))

time.sleep(5)

exit()

def on\_error(self, status):

print (status)

* + 1. **Keyword Location Module Unit Test**

Keyword Location module is tested by creating a separate .py file containing the python code for fetching tweets based on keyword, location, maximum number of tweets, start date, end date and language.

def lockey():

print("enter the file name")

filename=input()

csvFile = open(filename, 'a')

#Use csv Writer

csvWriter = csv.writer(csvFile)

start\_time=time.time()

print("enter the keyword")

keyword=input()

print("enter the language")

language=input()

print("enter max tweets")

maxtw=input()

max\_tweets=int(maxtw)

print("enter the from date")

sinc=input()

print("enter the to date")

too=input()

print("enter the location")

location=input()

count=1

c=1

'''t=0

a=2014

while t < 10:

auth = tweepy.OAuthHandler(consumer\_key=ckey, consumer\_secret=csecret)

api = tweepy.API(auth)

results = api.search(q=keyword,lang=language,geo='48.85681, 2.34760,15km',since\_id=a)

a=resu.getMaxId()

t=t+1

'''

#max\_tweets=1000

auth = tweepy.OAuthHandler(consumer\_key=ckey, consumer\_secret=csecret)

api = tweepy.API(auth)

searched\_tweets = [status for status in tweepy.Cursor(api.search, q=keyword,language=language,since\_id=sinc,until=too,location=location).items(max\_tweets)]

searched\_tweets = []

last\_id = -1

while len(searched\_tweets) < max\_tweets:

count = max\_tweets - len(searched\_tweets)

try:

new\_tweets = api.search(q=keyword, count=count, max\_id=str(last\_id - 1))

if not new\_tweets:

break

searched\_tweets.extend(new\_tweets)

last\_id = new\_tweets[-1].id

except tweepy.TweepError as e:

# depending on TweepError.code, one may want to retry or wait

# to keep things simple, we will give up on an error

time.sleep(30)

continue

for tweets in searched\_tweets:

print (c,tweets.created\_at.date,tweets.text.encode('utf-8'))

c=c+1

csvWriter.writerow([tweets.created\_at, tweets.text.encode('utf-8')])

#print tweet.created\_at, tweet.text

csvFile.close()

* + 1. **Keyword Location Infinite Module Unit Test**

Keyword Location(infinite) module is tested by creating a separate .py file containing the python code for fetching tweets based on keyword, location, start date, end date and language.

def lockey1():

counter=0

print("enter the file name")

filename=input()

csvFile = open(filename, 'a')

#Use csv Writer

csvWriter = csv.writer(csvFile)

start\_time=time.time()

print("enter the keyword")

keyword=input()

print("enter the language")

language=input()

print("enter the from date")

sinc=input()

print("enter the to date")

too=input()

print("enter the location")

location=input()

auth = tweepy.OAuthHandler(consumer\_key=ckey, consumer\_secret=csecret)

api = tweepy.API(auth)

count=1

for tweet in tweepy.Cursor(api.search,q=keyword,language=language,location=location,since\_id=sinc).items():

try:

#Write a row to the csv file/ I use encode utf-8

csvWriter.writerow([tweet.created\_at, tweet.text.encode('utf-8')])

print(tweet.created\_at, tweet.text.encode('utf-8'))

counter = counter + 1

print(counter)

if counter == 2000:

time.sleep(60\*20) # wait for 20 min everytime 2,000 tweets are extracted

counter = 0

continue

except tweepy.error.TweepyError as e:

time.sleep(60\*20)

continue

csvFile.close()

* + 1. **Multithreading Module Unit Test**

Multithreading module is tested by creating a separate .py file containing the python code for fetching tweets based on keyword, location, start date, end date and language.

def multi1():

t=d1()

t1=d1()

lock.acquire()

try:

t.start()

finally:

lock.release()

lock.acquire()

try:

t1.start()

finally:

lock.release()

t.sleep2()

t1.sleep()

class d1(threading.Thread):

def \_\_init\_\_(self):

super(d1,self).\_\_init\_\_()

#time.sleep(10)

def sleep(self):

time.sleep(2)

def sleep2(self):

time.sleep(2)

def run(self):

counter=0

print("enter the filename")

filename=input()

csvFile = open(filename, 'a')

#Use csv Writer

csvWriter = csv.writer(csvFile)

start\_time=time.time()

print("enter the keyword")

keyword=input()

print("enter the language")

language=input()

print("enter the from date")

sinc=input()

print("enter the to date")

too=input()

print("enter the location")

location=input()

auth = tweepy.OAuthHandler(consumer\_key=ckey, consumer\_secret=csecret)

api = tweepy.API(auth)

count=1

for tweet in tweepy.Cursor(api.search,q=keyword,language=language,location=location,since\_id=sinc).items():

try:

auth = tweepy.OAuthHandler(consumer\_key=ckey, consumer\_secret=csecret)

api = tweepy.API(auth)

#Write a row to the csv file/ I use encode utf-8

csvWriter.writerow([tweet.created\_at, tweet.text.encode('utf-8')])

print(tweet.created\_at, tweet.text.encode('utf-8'))

counter = counter + 1

#print(counter)

if counter == 2000:

time.sleep(60\*20) # wait for 20 min everytime 2,000 tweets are extracted

counter = 0

continue

except tweepy.error.TweepyError as e:

time.sleep(60\*20)

continue

except BaseException as e:

time.sleep(60\*2.5)

continue

except Exception as e:

time.sleep(60\*2.5)

continue

csvFile.close()

* + 1. **Un-mined Tweets Module Unit Test**

Un-mined tweets module is tested by creating a separate .py file containing the python code for fetching tweets based on keyword, location, start date, end date and language. Only difference is that it stores tweets without mining them.

def unmined1():

print("enter the keyword")

keyword=input()

print("enter the language")

language=input()

print("enter the from date")

sinc=input()

print("enter the to date")

too=input()

print("enter the location")

location=input()

print("enter the filename")

filename=input()

#searchQuery = input() # this is what we're searching for

maxTweets = 10000000 # Some arbitrary large number

tweetsPerQry = 100 # this is the max the API permits

fName = filename # We'll store the tweets in a text file.

# If results from a specific ID onwards are reqd, set since\_id to that ID.

# else default to no lower limit, go as far back as API allows

sinceId = None

# If results only below a specific ID are, set max\_id to that ID.

# else default to no upper limit, start from the most recent tweet matching the search query.

max\_id = 0

tweetCount = 0

auth = tweepy.OAuthHandler(consumer\_key=ckey, consumer\_secret=csecret)

api = tweepy.API(auth)

print("Downloading max {0} tweets".format(maxTweets))

with open(fName, 'w') as f:

while tweetCount < maxTweets:

try:

if (max\_id <= 0):

if (not sinceId):

new\_tweets = api.search(q=keyword,language=language,location=location,since\_id=sinc, count=tweetsPerQry)

else:

new\_tweets = api.search(q=keyword,language=language,location=location,since\_id=sinc, count=tweetsPerQry)

else:

if (not sinceId):

new\_tweets = api.search(q=keyword,language=language,location=location,since\_id=sinc, count=tweetsPerQry,max\_id=str(max\_id - 1))

else:

new\_tweets = api.search(q=keyword,language=language,location=location,since\_id=sinc, count=tweetsPerQry,max\_id=str(max\_id - 1))

if not new\_tweets:

print("No more tweets found")

break

for tweet in new\_tweets:

f.write(jsonpickle.encode(tweet.\_json, unpicklable=False) +'\n')

tweetCount += len(new\_tweets)

print("Downloaded {0} tweets".format(tweetCount))

max\_id = new\_tweets[-1].id

except tweepy.TweepError as e:

# Just exit if any error

print("some error : " + str(e))

break

print ("Downloaded {0} tweets, Saved to {1}".format(tweetCount, fName))

* 1. **Integration Testing**

Integration testing involves individual unit test modules are merged together into a larger module with more functionality which is known as integrated testing and this end up giving us the final application with all modules in it. Integration testing is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

* 1. **Test Results**

Testing generates highly functional and appropriate code which fulfills all user input and produces desired output. The integrated behaviour as well as proper return values for the functions determined that the code was functioning properly. Integration was also properly functional, along with conformance with the architecture. With all test cases satisfied and usability testing, the only testing left is system testing, which happens in the environment where it will deployed. Test results are the deciding factor for the efficiency and accuracy of the code written. Therefore, testing is the most important and essential part to check the code functionality and efficiency.

**CHAPTER 6**

**RESULTS AND DISCUSSION**

* 1. **Output/Results**

The application functions as an automation tool and it help to minimize labour work and efforts and lose data from variety of failures. Apart from its automation another advantage is its seamless execution of program to fetch any number of tweets existing which was previously limited to 100 and after that user has to restart the process. This application can be used for any survey purpose and provides user a simple and efficient and easy to execute application. Another advantage of this application is, its generalization which made the application to be used for any keyword and for any location and also for any time period.

* 1. **Result Analysis**

The application was developed successfully meeting all the requirements and overcoming all the backlogs in previous application. All the functionality were defined in the requirement phase and further enhancements will be proposed and taken into consideration once the application is deployed and feedbacks gathered. The application was used for analysis of polish presidential election and it the result gathered from the analysis were very close to the actual result obtained.

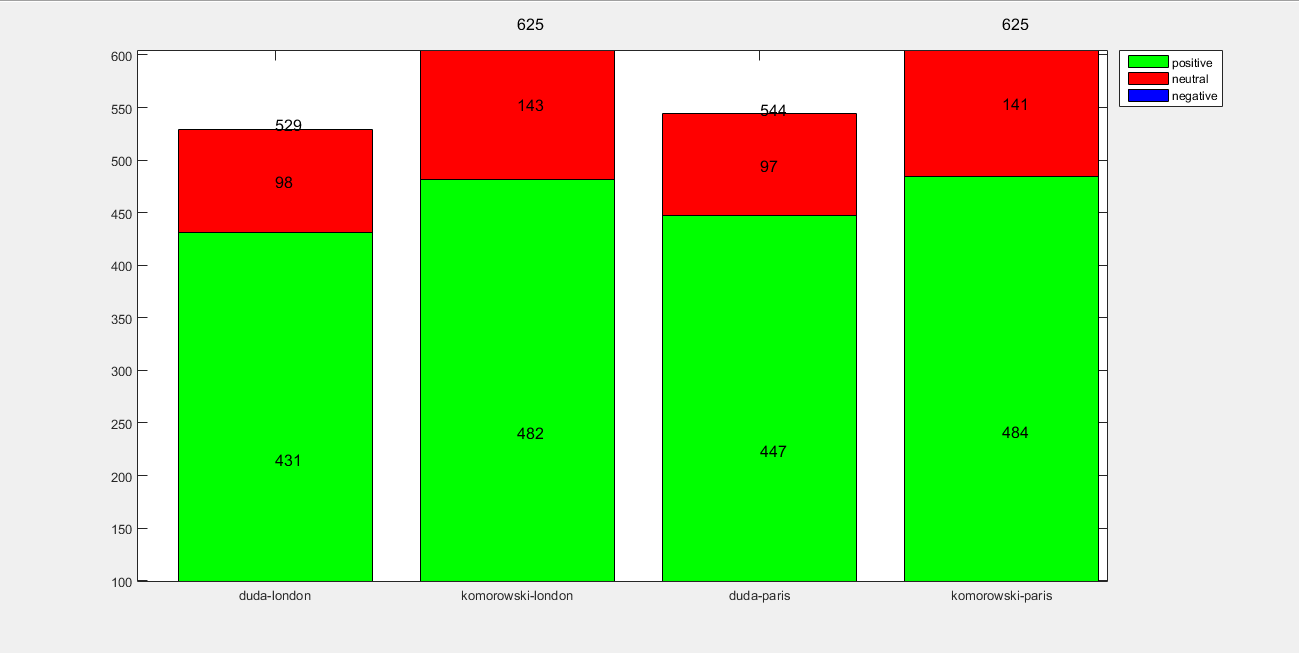


Fig. 6.2.1 Polish Presidential election analysis for final nominees and also showing survey results for 2 countries.

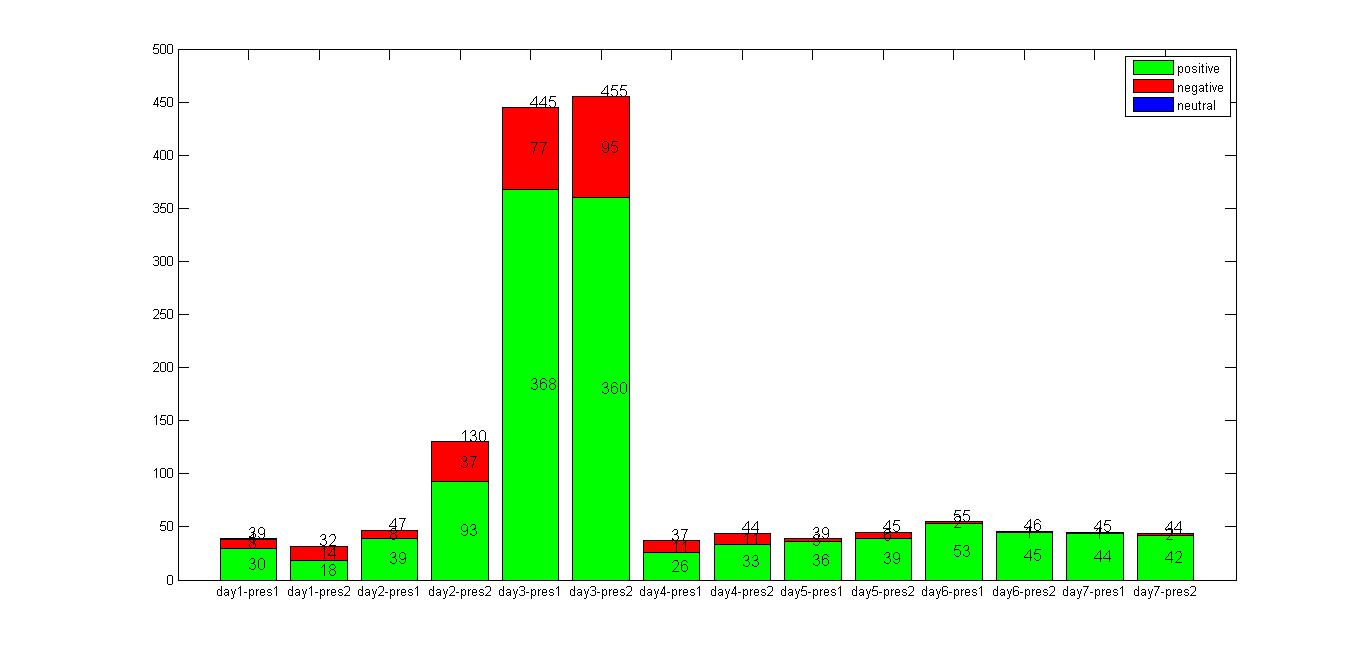


Fig. 6.2.2 Bar Graph representing Day-wise analysis for both the presidents within a week in London.

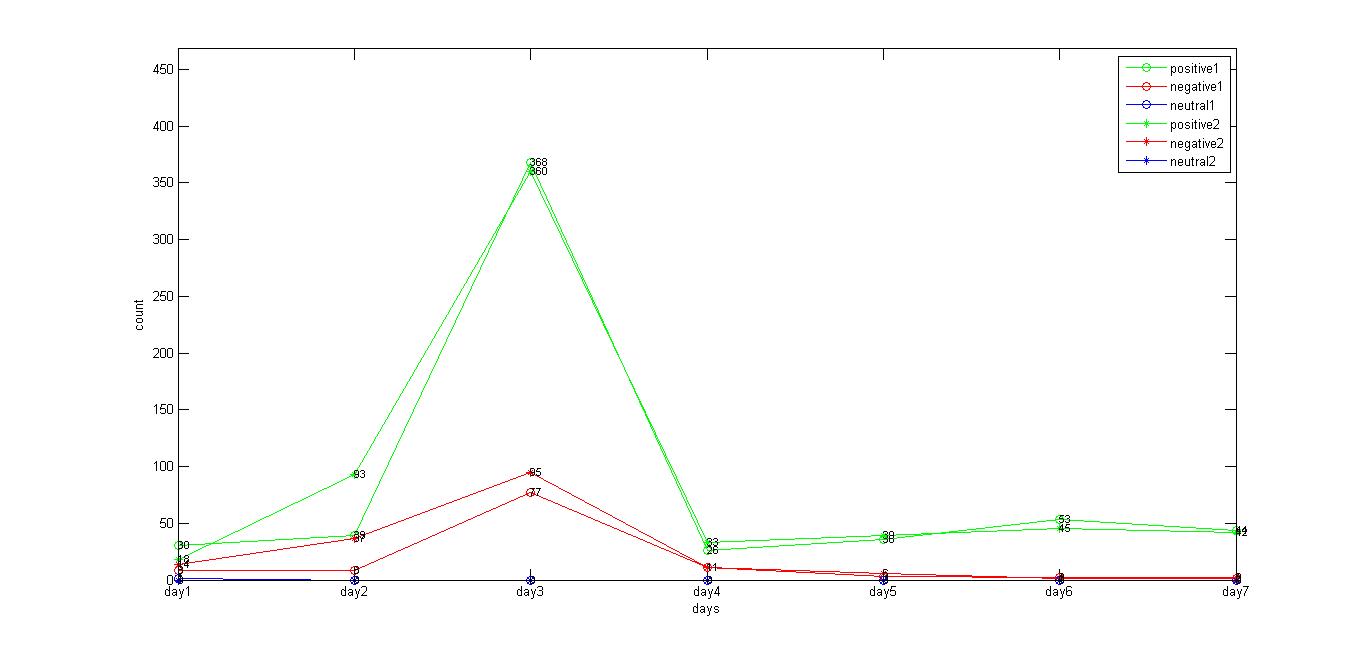


Fig. 6.2.3 Line Graph representing the Day-wise analysis for both the presidents within a week in London.

Similar analysis can be done for any country and for any keyword. The results obtained showed more positive:negative support to Duda and slightly less positive:negative support to Komorowski. Final result of the election was also 52% to Duda and 48% to Komorowski.

Performace of application was also analysed. With time as the number of tweets fetched increases, it becomes slightly slow because of the sleep function called in the program and also because the data in the database after time t becomes too much for it to handle. In this case we required the concept of Big-Data.

* 1. **Discussion**

This project was developed out of need for a faster and efficient way to carry out all the manual tasks previously carried out by research students at EFREI, France. The need to automate the fetching of tweets made this project come into existence. Another requirement which was essential was the need for fetching any number of tweets from twitter for a given period of time. As the twitter permits only 100 to be fetched at a time, so the user-id of the 100th tweet can be used as a reference id for the next set of 100 tweets and this runs in loop in order to fetch any number of tweets. Another need that led this project was the representation of the analyzed tweets. MATLAB was used for proper graphical representation of the analyzed tweets and easy for user to understand.

Also the previous system which was designed, there the user has to make changed to the code in order to change location and other functionality as a result user must have knowledge of all programming language. But the currently designed system is general and the user doesn’t require programming language knowledge in order to do a survey. All the data needed to be input at the run-time and no modification to the code is required.

**CHAPTER 7**

**CONCLUSION AND FUTURE WORK**

* 1. **Conclusion**

Efficient and reliable system was the key for this project. This project also provides a user-friendly interface and easy to find sentiments based on the tweets fetch regarding certain keyword. This project also fulfills the loopholes in previous system developed; it enables to fetch any number of tweets without restriction. This project also meets the initial goal of automating everything and also generalizing all the working by making it efficient to be used for any keyword.

* 1. **Scope of Future Work**

In order to enhance the scope of the project what can be done is, on deployment of the system, based on the feedback received, there will be changes made to the system. UI changes will be made based on user reviews. Also, based on review, modifications and enhancements will be made to widen the functionality of the tool. **Casandra** can be used in order to store large amount data in database. And also, adding feature like runtime initialization of threads.

**CHAPTER 8**

**REFERENCES**

# Book:

* [Bo Pang](https://sites.google.com/site/bopang42/) and [Lillian Lee](http://www.cs.cornell.edu/home/llee): Opinion mining and Sentiment analysis

**Research Paper:**

An Analysis on Opinion Mining: Techniques and Tools

-G.Angulakshmi, Dr.R.ManickaChezian

*International Journal of Advanced Research in Computer and*

*Communication Engineering*

*Vol. 3, Issue 7, July 2014*

**Reference Links**

<https://docs.python.org/2/tutorial/>

[www.pythoncentral.io/introduction-to-tweepy-twitter-for-python/](http://www.pythoncentral.io/introduction-to-tweepy-twitter-for-python/)

<https://dev.twitter.com/streaming/>

[www.tutorialspoint.com/matlab/](http://www.tutorialspoint.com/matlab/)

[www.stackoverflow.com/](http://www.stackoverflow.com/)

[www.text2data.org/](http://www.text2data.org/)