EMOTION ANALYSIS USING OPINION MINING

B.Tech. Minor Project - II Report

BY:

PRAKHAR DOGRA	2K12/CO/86
SHUBHAM VARSHNEY	2K12/CO/126
VIKAS	2K12/CO/139
VIKRANT DABAS	2K12/CO/140

DEPARTMENT OF COMPUTER ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

SHAHBAD DAULATPUR, MAIN BAWANA ROAD, DELHI – 110042, INDIA

MAY, 2015

CERTIFICATE

We hereby certify that the work which is being presented in this B.Tech. Minor Project Report entitled **"EMOTION ANALYSIS USING OPINION MINING",** in partial fulfilment of the requirements for the award of the **Bachelor of Technology in Computer Engineering** and submitted to the Department of Computer Engineering of Delhi Technological University is an authentic record of our own work carried out during a period from Feburary 2015 to April 2015 under the supervision of **Mr. Rahul Gupta, Assistant Professor, Computer Engineering Department**.

The matter presented in this report has not been submitted by us for the award of any other degree elsewhere.

VIKRANT DABAS | SHUBHAM VARSHNEY |PRAKHAR DOGRA | VIKAS
(2K12/CO/140) | (2K12/CO/126) | (2K12/CO/086) | (2k12/CO/139)

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Date:

Mr. Rahul Gupta
(Project Supervisor)
(Assistant Professor)

INDEX

I.) Introduction

The demand of Information on Opinions and Sentiments.

"Romance should never begin with sentiment. It shouldbegin with science and end with a settlement." - Oscar Wilde, *An Ideal Husband*

"What other people think" has always been an important piece of informationfor most of us during the decision-making process. Long beforeawareness of the World Wide Web became widespread, many of usasked our friends to recommend an auto mechanic or to explain whothey were planning to vote for in local elections, requested referenceletters regarding job applicants from colleagues, or consulted *ConsumerReports* to decide what dishwasher to buy. But the Internet and theWebhave now (among other things) made it possible to find out about theopinions and experiences of those in the vast pool of people that are neitherour personal acquaintances nor well-known professional critics - that is, people we have never heard of. And conversely, more and morepeople are making their opinions available to strangers via the Internet.Indeed, according to two surveys of more than 2000 American adultsEach :- [1,2]

- 81% of Internet users (or 60% of Americans) have done onlineresearch on a product at least once;
- 20% (15% of all Americans) do so on a typical day;
- among readers of online reviews of restaurants, hotels, andvarious services (e.g., travel agencies or doctors), between73% and 87% report that reviews had a significant influenceon their purchase;
- consumers report being willing to pay from 20% to 99% morefor a 5-star-rated item than a 4-star-rated item (the variancestems from what type of item or service is considered);
- 32% have provided a rating on a product, service, or personvia an online ratings system, and 30% (including 18%of online senior citizens) have posted an online comment orreview regarding a product or service.

We hasten to point out that consumption of goods and servicesis not the only motivation behind people's seeking out or expressingopinions online. A need for political information is another importantfactor. For example, in a survey of over 2500 American adults, RainieandHorrigan[3] studied the 31% of Americans — over 60 millionpeople — that were

2006 *campaign internet users*, defined as those whogathered information about the 2006 elections online and exchangedviews via email. Of these,

- 28% said that a major reason for these online activities wasto get perspectives from within their community, and 34%said that a major reason was to get perspectives from outsidetheir community;
- 27% had looked online for the endorsements or ratings ofexternal organizations;
- 28% said that most of the sites they use share their pointof view, but 29% said that most of the sites they use challengetheir point of view, indicating that many people are notsimply looking for validations of their pre-existing opinions;
- 8% posted their own political commentary online.

The user hunger for and reliance upon online advice and recommendationsthat the data above reveals is merely one reason behindthe surge of interest in new systems that deal directly with opinions asa first-class object. But, Horrigan reports that while a majority ofAmerican internet users report positive experiences during online productresearch, at the same time, 58% also report that online informationwas missing, impossible to find, confusing, and/or overwhelming. Thus,there is a clear need to aid consumers of products and of informationby building better information-access systems than are currently inexistence.

➢ A note on Terminology: Opinion Mining, Sentiment Analysis, Subjectivity and all that.

"The beginning of wisdom is the definition of terms"

- Socrates.

The body of work we review is that which deals with the computational treatment of (in alphabetical order) *opinion*, *sentiment*, and *subjectivity* in text. Such work has come to be known as *opinion mining*, *sentiment analysis*, and/or *subjectivity analysis*. The phrases *review mining* and *appraisal extraction* have been used, too, and there are some connections to *affective computing*, where the goals include enabling computers to recognize and express emotions. This proliferation of terms reflects differences in the connotations that these terms carry, both in their original general-discourse usages4 and in the usages that have evolved in the technical literature of several communities. In 1994, Wiebe[4], influenced by the writings of the literary theorist Banfield, centered the idea of *subjectivity* around that of *private states*, defined by Quirk et al. as states that are not open to objective observation or verification.

Opinions, evaluations, emotions, and speculations all fall into this category; but a canonical exampleof research typically described as a type of subjectivity analysis is the recognition of opinion-oriented language in order to distinguish it from objective language. While there has been some research self-identified as subjectivity analysis on the particular application area of determining the value judgments (e.g., "four stars" or "C+") expressed in the evaluative opinions that are found, this application has not tended to be a major focus of such work. The term *opinion mining* appears in a paper by Dave et al.[5] that was published in the proceedings of the 2003 WWW conference; the publication venue may explain the popularity of the term within communities strongly associated with Web search or information retrieval. According to Dave et al., the ideal opinion-mining tool would "process a set of search results for a given item, generating a list of product attributes (quality, features, etc.) and aggregating opinions about each of them (poor, mixed, good)." Much of the subsequent research self-identified as opinion mining fits this description in its emphasis on extracting and analyzing judgments on various aspects of given items. However, the term has recently also been interpreted more broadly to include many different types of analysis of evaluative text. The history of the phrase *sentiment analysis* parallels that of "opinion mining" in certain respects. The term "sentiment" used in reference to the automatic analysis of evaluative text and tracking of the predictive judgments therein appears in 2001 papers by Das and Chen and Tong[6], due to these authors' interest in analyzing market sentiment. It subsequently occurred within 2002 papers by Turney and Pang et al., which were published in the proceedings of the annual meeting of the Association for Computational Linguistics (ACL) and the annual conference on Empirical Methods in Natural Language Processing (EMNLP). Moreover, Nasukawa and Yi entitled their 2003 paper, "Sentiment analysis: Capturing favorability using natural language processing", and a paper in the same year by Yi et al.[7] was named "Sentiment Analyzer: Extracting sentiments about a given topicusing natural language processing techniques." These events togethermay explain the popularity of "sentiment analysis" among communitiesself-identified as focused on NLP. A sizeable number of papersmentioning "sentiment analysis" focus on the specific application ofclassifying reviews as to their polarity (either positive or negative), afact that appears to have caused some authors to suggest that thephrase refers specifically to this narrowly defined task. However, nowadaysmany construe the term more broadly to mean the computationaltreatment of opinion, sentiment, and subjectivity in text.**Thus, when broad interpretations are applied, "sentiment analysis"and "opinion mining" denote the same**

field of study (which itself canbe considered a sub-area of subjectivity analysis). We have attemptedto use these terms more or less interchangeably in this survey. This is inno small part because we view the field as representing a unified bodyof work, and would thus like to encourage researchers in the area toshare terminology regardless of the publication venues at which theirpapers might appear.

II.) Problem Statement

The goal of our project is to do emotion analysis on tweets using opinion mining. The main idea of the project is to create a model that is able to predict the type of emotion of a tweet using web mining to initially extract tweets using the Twitter API and then further using sentiment analysis to analyze the emotions. The model is made using three modules that are named as follows:

- Pre-Processing Module (Data Cleaning)
- Scoring Module
- Dictionary Expansion Module

The third component is used for creation and further expansion of an independent dictionary that stores the emotion values of adjectives, adverbs and verbs that are not identified by the original dictionary.

III.) Implementation

➢ Software Tools Used

1. 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 is multipurpose, ie, it is not specialized to a specific target of users (like R for statistics, or PHP for web programming). It is extended through modules and libraries that hook very easily into the C programming language. For our purpose, we used a bunch of python libraries like tweepy, tkinter etc. A brief explanation for all of them has been given below.Python can be used for any programming task, from GUI programming to web programming with everything else in between. It's quite efficient, as much of its activity is done at the C level.For our purpose we have used the Python 3.4.3.

2. Python csv module

The so-called CSV (Comma Separated Values) format is the most common import and export format for spreadsheets and databases. There is no “CSV standard”, so the format is operationally defined by the many applications which read and write it. The lack of a standard means that subtle differences often exist in the data produced and consumed by different applications. These differences can make it annoying to process CSV files from multiple sources. Still, while the delimiters and quoting characters vary, the overall format is similar enough that it is possible to write a single module which can efficiently manipulate such data, hiding the details of reading and writing the data from the programmer.The **csv** module implements classes to read and write tabular data in CSV format. It allows programmers to say, “Write this data in the format preferred by Excel,” or “read data from this file which was generated by Excel,” without knowing the precise details of the CSV format used by Excel. Programmers can also describe the CSV formats understood by other applications or define their own special-purpose CSV formats.The **csv** module’s **reader** and **writer** objects read and write sequences. Programmers can also read and write data in dictionary form using the DictReaderand DictWriter classes.

3. PyCharm: Integrated Development Environment

PyCharm is an Integrated Development Environment (IDE) used for programming in Python. It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems (VCSes), and supports web development with Django. PyCharm is developed by the Czech company Jet Brains.It is cross-platform working on Windows, Mac OS X and Linux. PyCharm has a Professional Edition, released under a proprietary license and a Community Edition released under the Apache License.PyCharm Community Edition is less extensive than the Professional Edition.

4. NLTK

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet , along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, and an active discussion forum.NLTK is suitable for linguists, engineers, students, educators, researchers, and industry users alike. NLTK is available for Windows, Mac OS X, and Linux. Best of all, NLTK is a free, open source, community-driven project.NLTK has been called "a wonderful tool for teaching, and working in, computational linguistics using Python," and "an amazing library to play with natural language."In our project, we used NLTK in cleaning module as well as Part of Speech Tagging.NLTK helped in data cleaning by providing the list of stopwords.NLTK further helped in the tagging of Part of speech such as verbs , adjectives etc.NLTK provides several modules for these type of tasks, such as PunktTokenizer module for tokenizing the given string.

5. Tkinter

The **Tkinter** module ("Tk interface") is the standard Python interface to the Tk GUI toolkit.BothTk and Tkinter are available on most Unix platforms, as well as on Windows and Macintosh systems. Starting with the 8.0 release, Tk offers native look and feel on all platforms.Tkinter consists of a number of modules. The Tk interface is provided by a binary extension module named **_tkinter**. This module contains the low-level interface to Tk, and should never be used directly by application programmers. It is usually a shared library (or

DLL), but might in some cases be statically linked with the Python interpreter.The public interface is provided through a number of Python modules. The most important interface module is the Tkinter module itself. To use Tkinter, all we need to do is to import the Tkinter module. We made extensive use of Tkinter to provide an interactive graphical user interface to the user.

6. **Tweepy**

The python tweepy library has been used for interacting with the twitter and to download the twitter data.It is basically a data streaming library for tweeter.Tweepy supports oauth authentication. Authentication is handled by the tweepy.AuthHandler class.Tweepy provides access to the well documented Twitter API. With tweepy, it's possible to get any object and use any method that the official Twitter API offers.Main Model classes in the Twitter API are Tweets, Users, Entities and Places. Access to each returns a JSON-formatted response and traversing through information is very easy in Python.

7. **TweepyStreamingAPI**

One of the main usage cases of tweepy is monitoring for tweets and doing actions when some event happens. Key component of that is the StreamListener object, which monitors tweets in real time and catches them.

8. **Tweepy REST API**

Twitter provides the REST search API for searching tweets from Twitter's search index. This is different than using thestreaming filter API, in that the later is real-time and starts giving you results from the point of query, while the former is retrospective and will give you results from past, up to as far back as the search index goes (usually last 7 days). While the streaming API seems like the thing to use when you want to track a certain query in real time, there are situations where you may want to use the regular REST search API. We may also want to combine the two approaches, i.e. start 2 searches, one using the streaming filter API to go forward in time and one using the REST search API to go backwards in time, in order to get some on-going and past context for your search term.

➢ About the Implementation

Twitter is a social networking and microblogging service that lets its users post real time messages, called tweets. Tweets have many unique characteristics, which implicates new challenges and shape up the means of carrying sentiment analysis on it as compared to other domains. Following are some key characteristics of tweets: [8]

- *Message Length:* The maximum length of a Twitter message is 140 characters. This is different from previous sentiment classification research that focused on classifying longer texts, such as product and movie reviews.
- *Writing technique:* The occurrence of incorrect spellings and cyber slang in tweets is more often in comparison with other domains. As the messages are quick and short, people use acronyms, misspell, and use emoticons and other characters that convey special meanings.
- *Availability:* The amount of data available is immense. More people tweet in the public domain as compared to Facebook (as Facebook has many privacy settings) thus making data more readily available. The Twitter API facilitates collection of tweets for training.
- *Topics:* Twitter users post messages about a range of topics unlike other sites which are designed for a specific topic. This differs from a large fraction of past research, which focused on specific domains such as movie reviews.
- *Real time:* Blogs are updated at longer intervals of time as blogs characteristically are longer in nature and writing them takes time. Tweets on the other hand being limited to 140 letters and are updated very often. This gives a more real time feel and represents the first reactions to events.

We now describe some basic terminology related to twitter:

- *Emoticons:* These are pictorial representations of facial expressions using punctuation and letters. The purpose of emoticons is to express the user'smood.
- *Target:* Twitter users make use of the "@" symbol to refer to other users on Twitter. Usersare automatically alerted if they have been mentioned in this fashion.
- *Hash tags:* Users use hash tags "#" to mark topics. It is used by Twitter users to make their tweets visible to a greater audience.
- *Special symbols*: "RT" is used to indicate that it is a repeat of someone else's earlier tweet.

❖ ***Program Architecture***

Opinion words are the words that people use to express their opinion (positive, negative or neutral). To find the semantic orientation of the opinion words in tweets, we propose a novel hybrid approach involving both corpus-based and dictionary-based techniques. We also consider features like emoticons and capitalization as they have recently become a large part of the cyber language. To uncover the opinion direction, we will first extract the opinion words in the tweets and then find out their orientation, i.e., to decide whether each opinion word reflects a positive sentiment, negative sentiment or a neutral sentiment. In our work, we are considering the opinion words as the combination of the adjectives along with the verbs and adverbs. The corpus-based method is then used to find the semantic orientation of adjectives and the dictionary-based method is employed to find the semantic orientation of verbs and adverbs. The overall tweet sentiment is then calculated using a linear equation which incorporates emotion intensifiers too. The components of the model are:-

1.) Downloading of Data:

To get large publically available Twitter datasets, we use Twitter API.

Obtaining access tokens

In order to make authorized calls to Twitter's APIs, your application must first obtain an OAuth **access token** on behalf of a Twitter user or you could issue Application-only authenticated requests when user context is not required. The way you will obtain such tokens will depend on your use case.[12]

The Twitter API has two different flavors: RESTful and Streaming.

- REST APIs

The REST APIs provide programmatic access to read and write Twitter data. Author a new Tweet, read author profile and follower data, and more. The REST API identifies Twitter applications and users using OAuth; responses are available in JSON. The RESTful API is useful for getting things like lists of followers and those who follow a particular user, and is what most Twitter clients are built off of.[9]

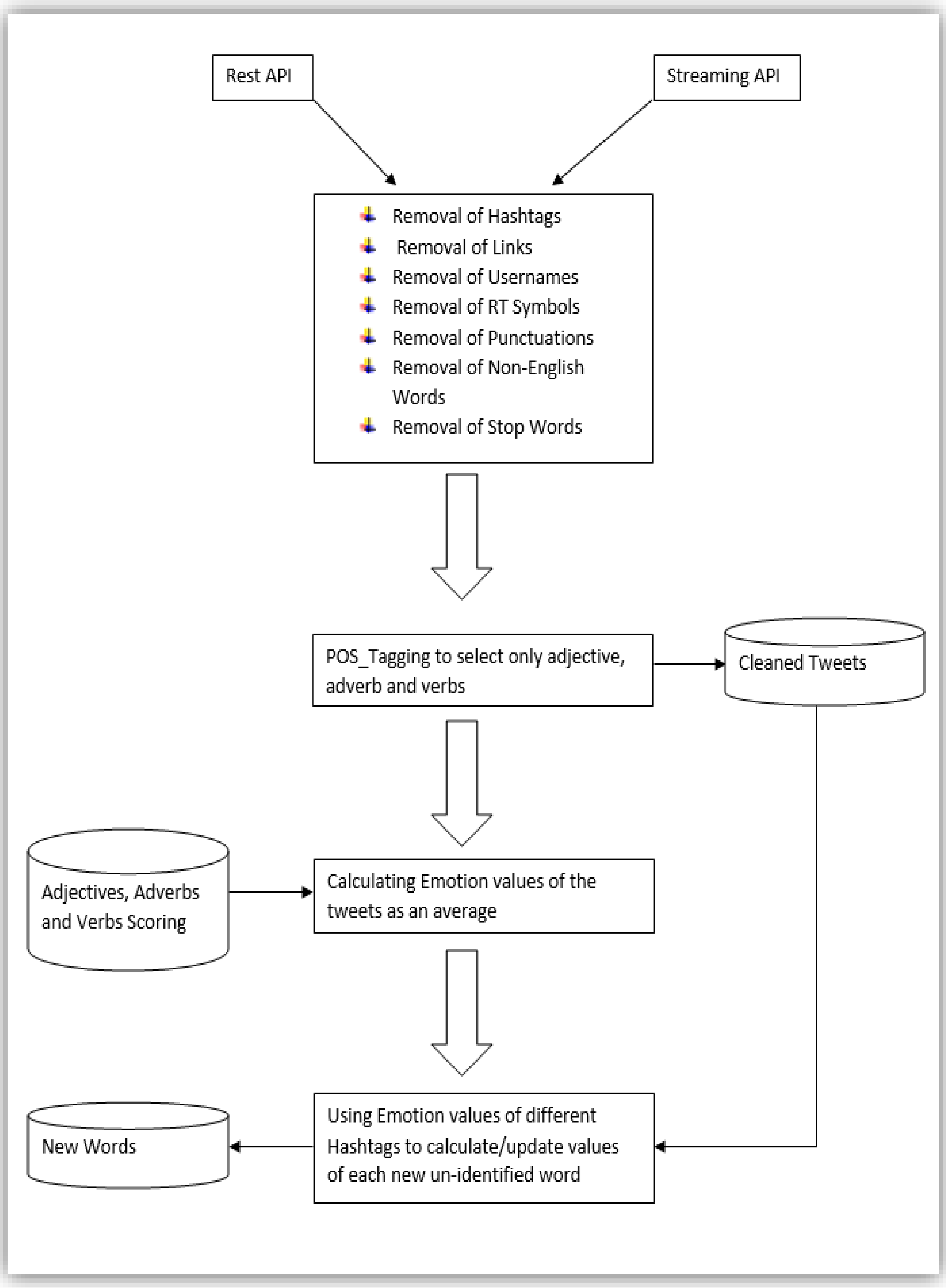

Rest API
Streaming API
Removal of Hashtags
Removal of Links
Removal of Usernames
Removal of RT Symbols
Removal of Punctuations
Removal of Non-English Words
Removal of Stop Words
POS_Tagging to select only adjective, adverb and verbs
Cleaned Tweets
Adjectives, Adverbs and Verbs Scoring
Calculating Emotion values of the tweets as an average
New Words
Using Emotion values of different Hashtags to calculate/update values of each new un-identified word

FIGURE: PROPOSED ARCHITECTURE

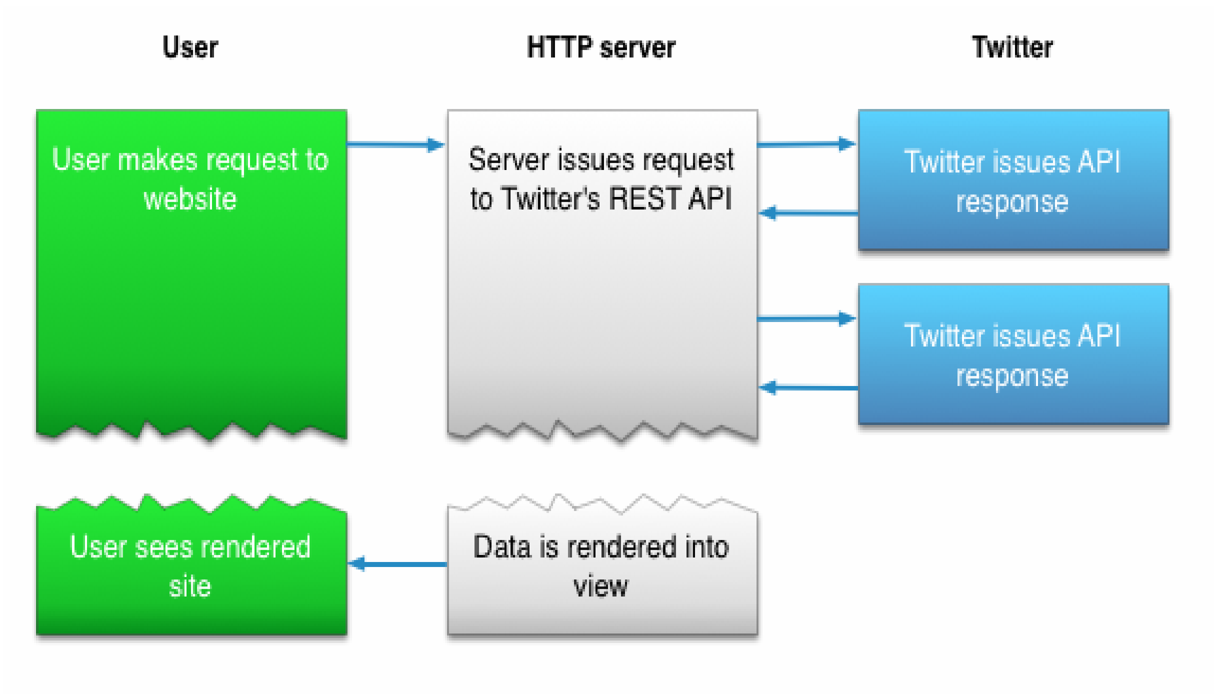
- The Streaming APIs

The Streaming APIs give developers low latency access to Twitter's global stream of Tweet data. A proper implementation of a streaming client will be pushed messages indicating Tweets and other events have occurred, without any of the overhead associated with polling a REST endpoint.Twitter offers several streaming endpoints, each customized to certain use cases.The Streaming API works by making a request for a specific type of data — filtered by keyword, user, geographic area, or a random sample — and then keeping the connection open as long as there are no errors in the connection.[10]

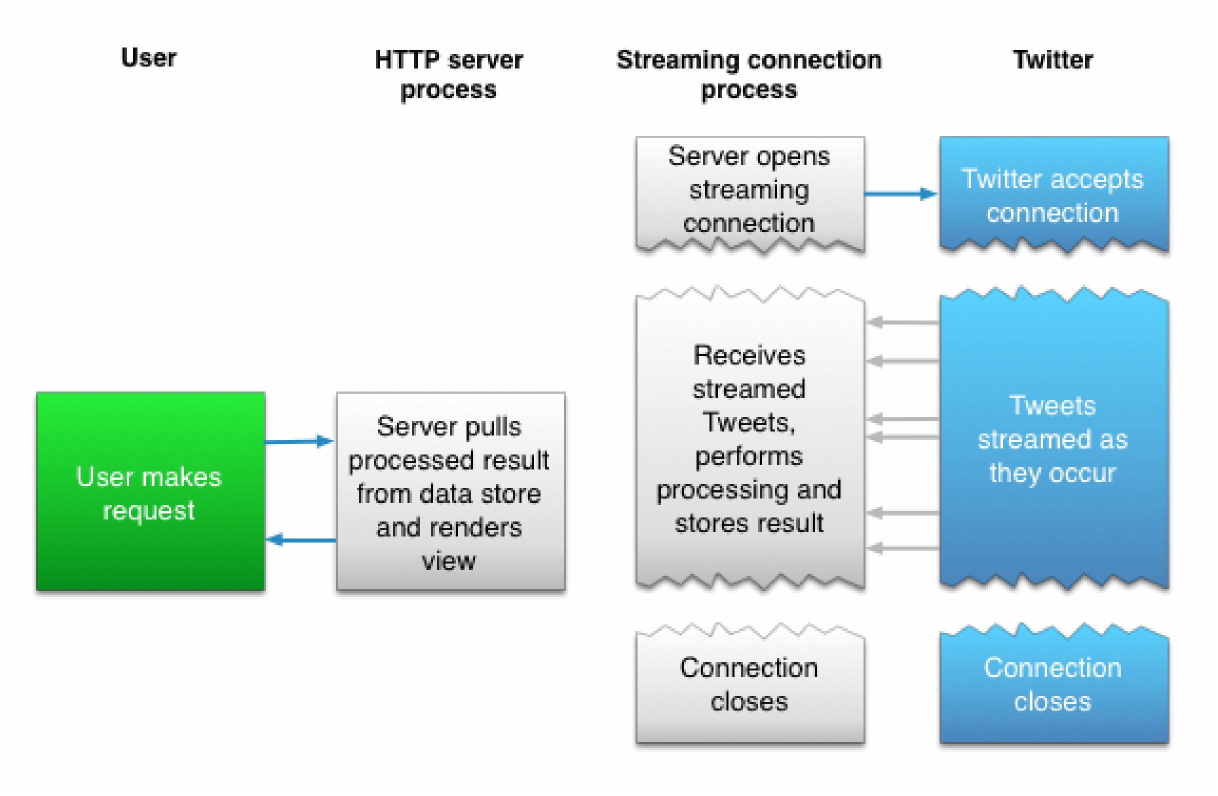
Public streams	Streams of the public data flowing through Twitter. Suitable for following specific users or topics, and data mining.
User streams	Single-user streams, containing roughly all of the data corresponding with a single user's view of Twitter.
Site streams	The multi-user version of user streams. Site streams are intended for servers which must connect to Twitter on behalf of many users.

Differences between Streaming and REST

Connecting to the streaming API requires keeping a persistent HTTP connection open. In many cases this involves thinking about your application differently than if you were interacting with the REST API. For an example, consider a web application which accepts user requests, makes one or more requests to Twitter's API, then formats and prints the result to the user, as a response to the user's initial request:[10]


User
HTTP server
Twitter
User makes request to website
Server issues request to Twitter's REST API
Twitter issues API response
Twitter issues API response
User sees rendered site
Data is rendered into view

An app which connects to the Streaming APIs will not be able to establish a connection in response to a user request, as shown in the above example. Instead, the code for maintaining the Streaming connection is typically run in a process separate from the process which handles HTTP requests:


User
HTTP server process
Streaming connection process
Twitter
Server opens streaming connection
Twitter accepts connection
Receives streamed Tweets, performs processing and stores result
Tweets streamed as they occur
User makes request
Server pulls processed result from data store and renders view
Connection closes
Connection closes

The streaming process gets the input Tweets and performs any parsing, filtering, and/or aggregation needed before storing the result to a data store. The HTTP handling process queries the data store for results in response to user requests. While this model is more complex than the first example, the benefits from having a realtime stream of Tweet data make the integration worthwhile for many types of apps.To access the twitter API we have used tweepy python package.[11]

2.) <u>**Cleaning of Downloaded Data**</u>

Raw data is the original data and is often hard to use for analysis directly. Data processed after applying the data cleaning step is ready for analysis. For this reason data cleaning process plays a very crucial in our Project. The data we extracted from twitter contains many undesirable symbols, words, and punctuations etc. that may slow down our applied algorithms or even make them to perform incorrectly. Hence, it becomes very necessary to write data cleaning modules or in some cases use the language library for the same purpose.

We present some of the differences between the rawdata and the tidy data obtained after the cleaning module is applied on it:

- Raw data may only be need to process only once (during cleaning).
- Raw data is the original data and is often hard to use for analysis directly.
- Processed Data is ready for analysis.
- Processing of raw data includes removal of hashtags, links, usernames, punctuations, non-English words and symbols like RT.

Pre-processing of Tweets

To prepare the transaction file that contains opinion indicators, namely the adjective, adverb and verb .

Thus, we pre-process all the tweets as follows:

i. Remove all URLs (e.g. www.example.com), hash tags (e.g. #topic), targets (@username), special Twitter words (“e.g. RT”).
ii. Calculate the percentage of the tweet in Caps.
iii. Correct spellings: A sequence of repeated characters is tagged by a weight. We do this to differentiate between the regular usage and emphasized usage of a word.

iv. Using a POS tagger, the NL Processor linguistic Parser, we tag the adjectives, verbs and adverbs.

The following strategy has been followed to make the downloaded twitter data tidy

i. For Removal of hashtags, a substring removal python function is used.
ii. Similar to the removal of hashtags, substring removal function is used to remove the links in the tweets.
iii. Each punctuation is replaced by an empty substring
iv. Removal of strings generally present in tweets such as RT, @someUser etc. through the substring removal function.
v. Removal of non-English tweets by encoding them into ASCII and if encoding produces any exception, then reject them, otherwise, select the tweets.
vi. Removal of stopping words through nltk library.
vii. The substring removal function removes the part of the string that contains a substring e.g. if substring = 'http', then http://www.google.com is removed, that means, remove until a space is found.
viii. Substring removal function has been very useful in removing the links, as well as retweets, hashtags and some other undesired content from the tweets.
ix. NLTK is a leading platform for building Python programs to work with human language data.
x. It provides easy-to-use suite of text processing libraries for tokenization and tagging.
xi. NLTK has been used to remove the stopping words from all the tweets.
xii. Words are fed to the Part of Speech tagger after Sentence Tokenizer and Word Tokenizer is applied.
xiii. Only adjectives, adverbs and verbs have been retained.All other words are discarded.

3.) **Proposed Scoring Modules**

We have proposed several scoring methods/formulas to calculate the average emotion value of tweets of a handle(username) or hashtag:

1. Emotion value = $\frac{\text{sum of adjective values} + \text{sum of adverb and verb values}}{(\text{no.of adjectives} * 5) + \text{no.of adverb and verb}}$

2. Multiply the value of adverb/verb with the upcoming adjective .If two verb /adverb

are next to each other, simply multiply them. And then all these products are added. Further divided by 5*number of adjectives encountered.

3. If the value of adverb/verb is less than 0 i.e, negative, then for the upcoming adjective, subtract its value from 5 instead of multiplying. And if value of verb/adverb is positive and >= 0.5 then multiply it with the upcoming adjective else multiply 0.5 with upcoming adjective. Later these products are added and the sum is divided by 5*number of adjectives encountered.
4. Since the emotions are independent of each other, we should treat them independently and use a different formula for each emotion. We have used normalising values of different emotions. These predefined values are used to normalize the emotion values. In this method an assumption is taken that if an emotion is strong it's respective value is > 3 (out of a scale of 5).

<u>Scoring Module-1:</u>

- This algorithm is based on the idea that every emotion is distinct from one another and thus the formula used to calculate the required emotion value is different from one another. The general method used is:

if sum of emotion values of all adjectives/number of adjectives > 3 then :

$$\text{emotion value} = \frac{sum\ of\ adjectives/normalizing\ factor + sum\ of\ adverb\ \&\ verbs}{total\ number\ of\ adjectives, adverbs\ and\ verbs}$$

else :

$$\text{emotion value} = \frac{sum\ of\ adjectives/normalizing\ factor + sum\ of\ adverb\ \&\ verbs}{total\ number\ of\ adjectives, adverbs\ and\ verbs}$$

- The normalizing factor varies for each emotion and hence gives a different formula for each emotion. This normalizing factor has been calculated on a common scale of 0 to 5 using the images of brain scans of prefrontal cortex.
- There is an assumption that words with emotion value(0-5) > 3 for an adjective express stronger feelings. Whereas the other express weaker feelings.

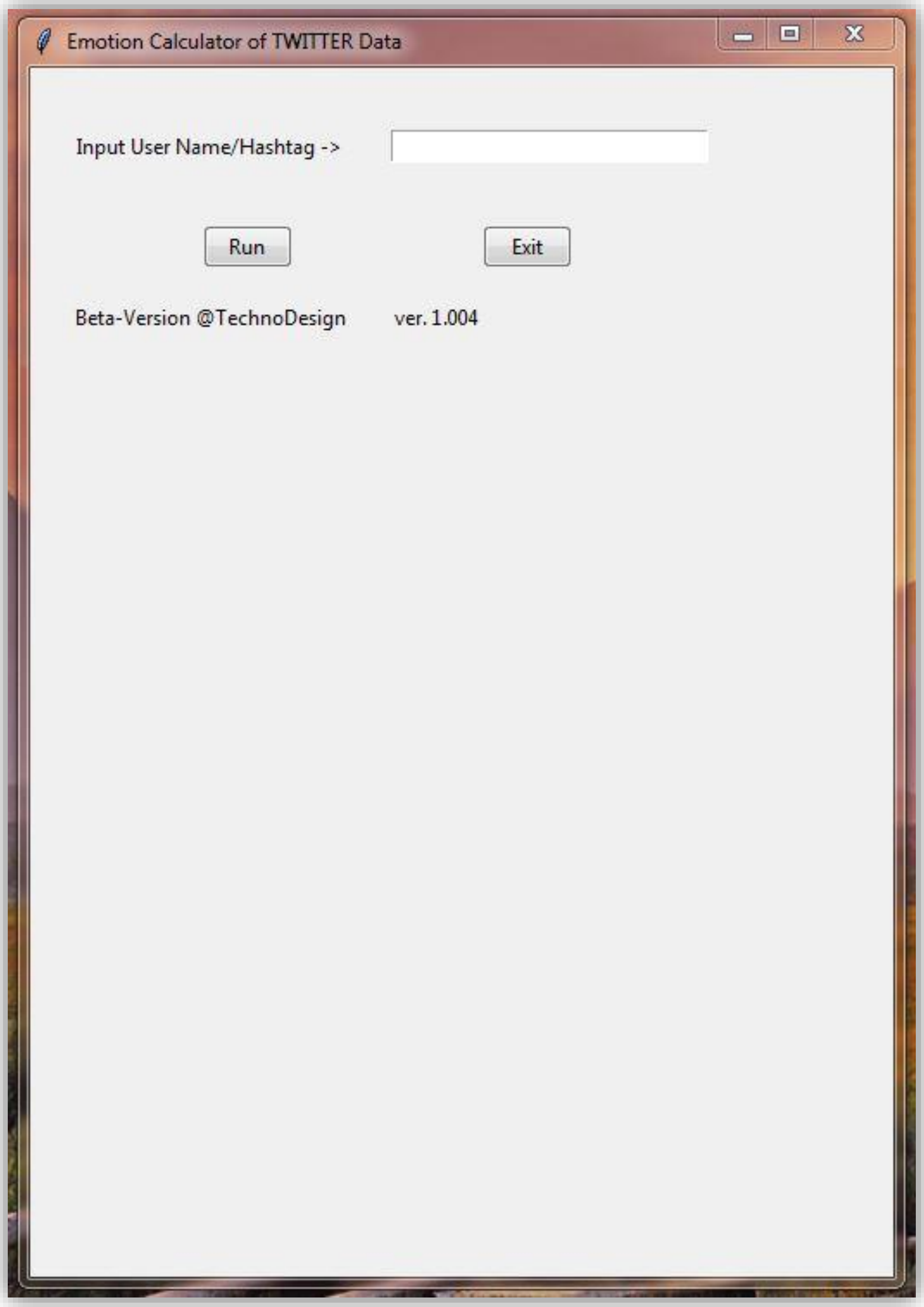
<u>Scoring Module-2:</u>

- This algorithm is based on the idea that whenever there is an adverb/verb in front of an adjective there is bound to be an effect on the emotion expressed by the same adjective. It can be either negative or negative. Keeping this in mind we have created method to calculate the emotion values.
- If the value of adverb/verb is less than 0 i.e, negative, then for the upcoming adjective, subtract its value from 5 instead of multiplying . And if value of verb/adverb is positive and >= 0.5 then multiply it with the upcoming adjective else multiply 0.5 with upcoming adjective. Later these products are added and the sum is divided by 5*number of adjectives encountered.
- And if there are more than 1 adverbs/verbs before an adjective then they are multiplied together.

4.) <u>Dictionary Expansion Module:</u>

- The main idea of this module is to create a dictionary apart from the original dictionary being used.
- This module will be run after the scoring module for a hashtag or user-handle has been run.
- The emotion values from the scoring modules will be fed into this module and those values will be assigned to words (adjective/verb/adverb) that haven't been identified by the original dictionary. Now this module is separate from the model and uses emotion values of every iteration of the model to expand it's dictionary and update already used words.
- This module is designed to input emotion values of a hashtag or a user-handle. But it can be modified to process the tweets one by one.

IV.) Screenshots and Some Analysis



Emotion Calculator of TWITTER Data
Input User Name/Hashtag ->
Run
Exit
Beta-Version @TechnoDesign
ver. 1.004

FIGURE: STARTUP SCREEN

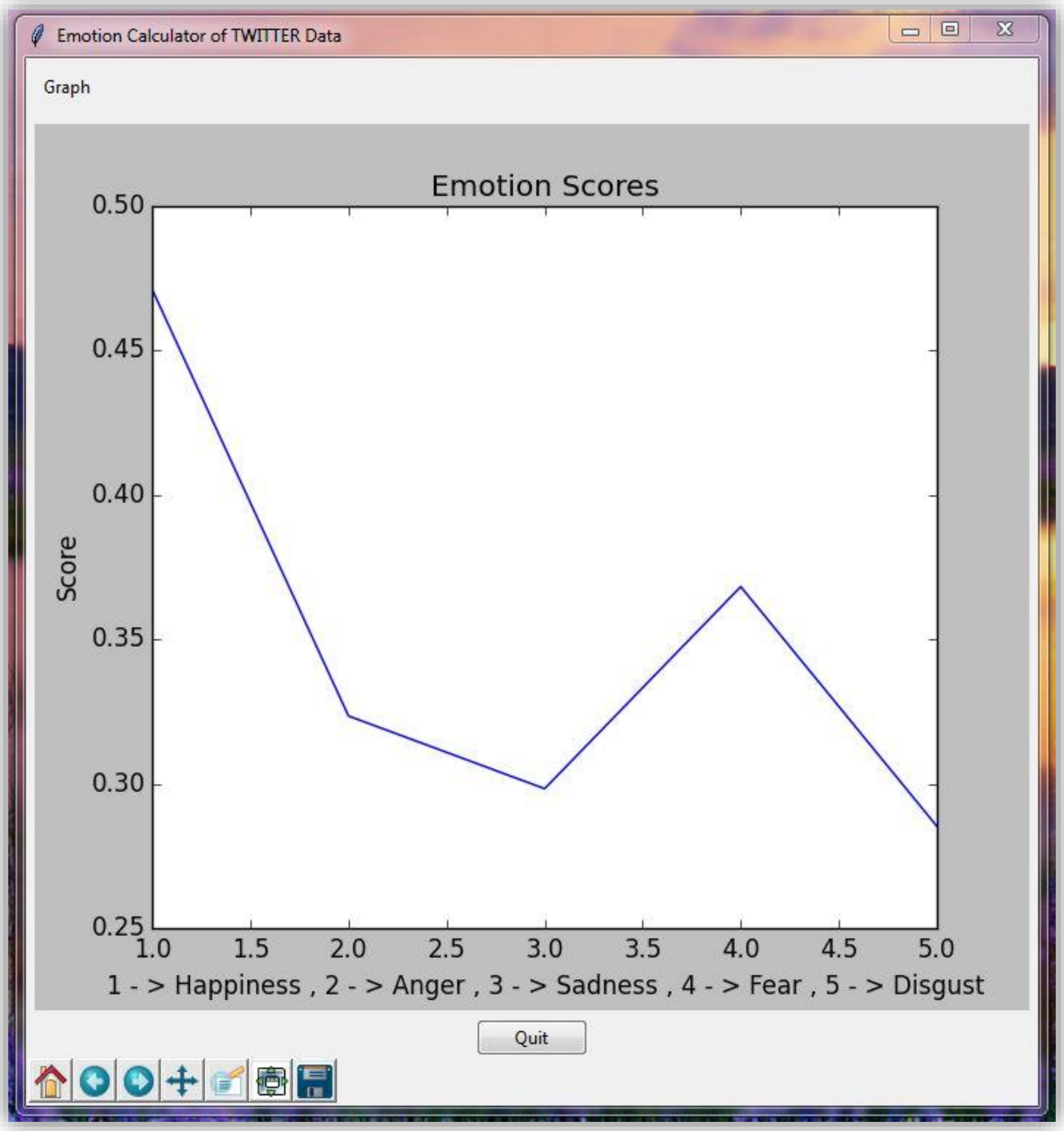
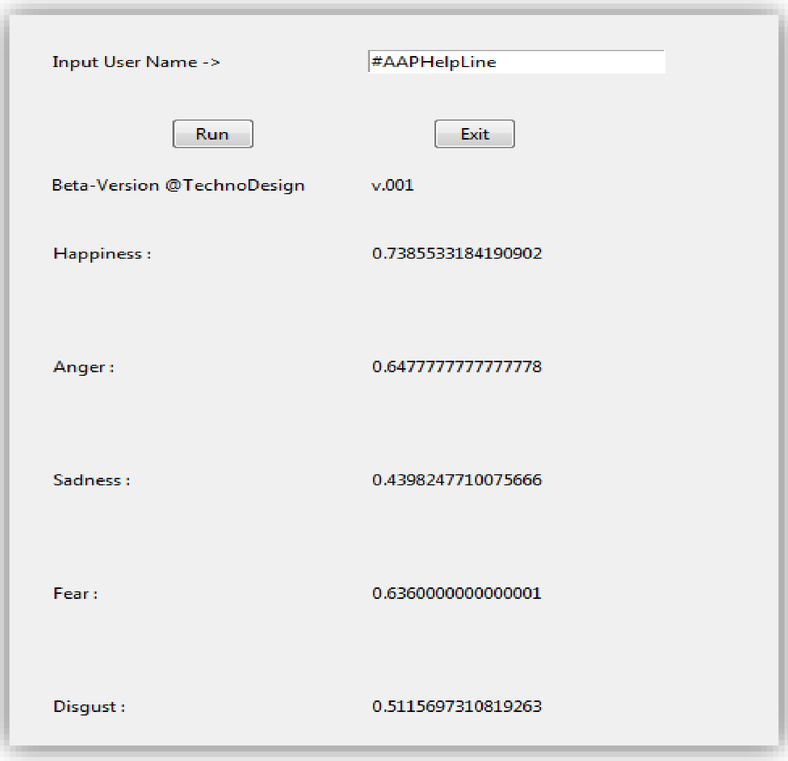

Emotion Calculator of TWITTER Data
Graph
Emotion Scores
Score
0.50
0.45
0.40
0.35
0.30
0.25
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
1 - > Happiness , 2 - > Anger , 3 - > Sadness , 4 - > Fear , 5 - > Disgust
Quit

FIGURE: GRAPH VIEWER

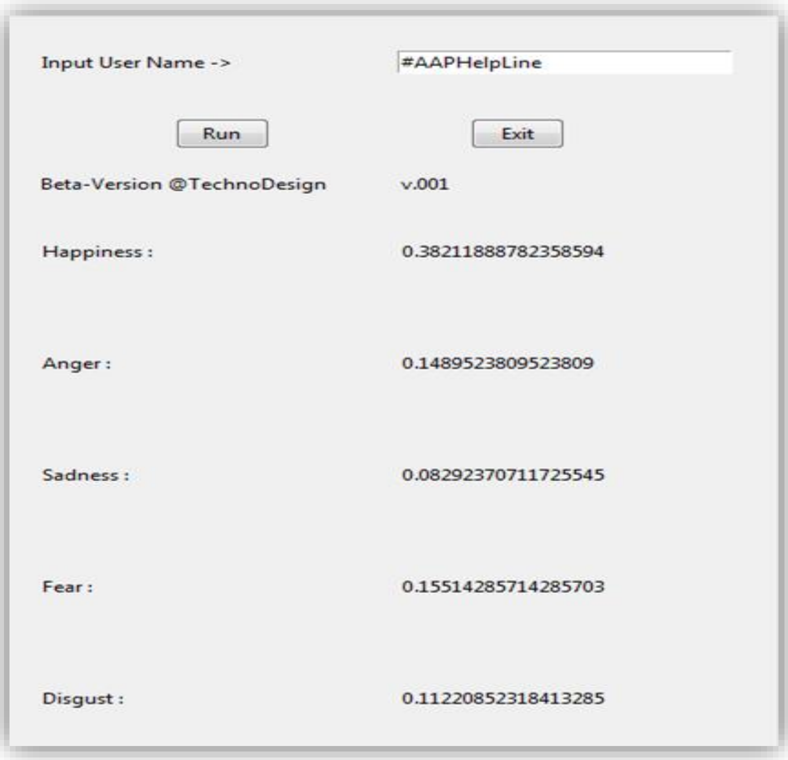
The above figures show the introduction window when the script is executed for the first time. After the user inputs a user name or hashtag, the scores for each emotion are indicated and the graph corresponding to the values can be displayed. Some case values are studied in the following pages.

Working of Scoring Module 1



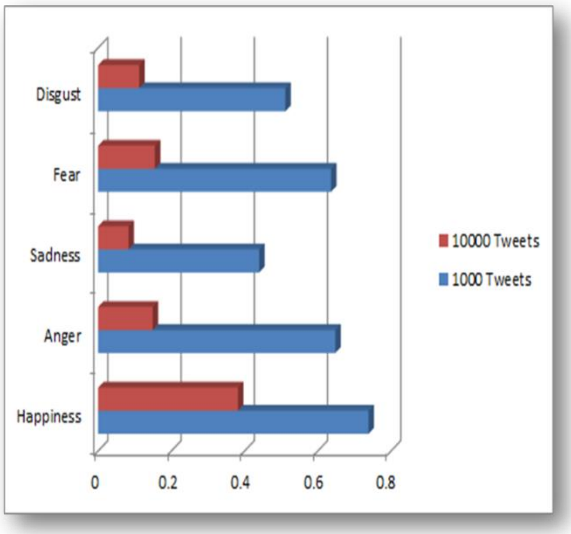
```
Input User Name ->              #AAPHelpLine

        Run                         Exit

Beta-Version @TechnoDesign      v.001

Happiness :                     0.7385533184190902

Anger :                         0.6477777777777778

Sadness :                       0.4398247710075666

Fear :                          0.6360000000000001

Disgust :                       0.5115697310819263
```

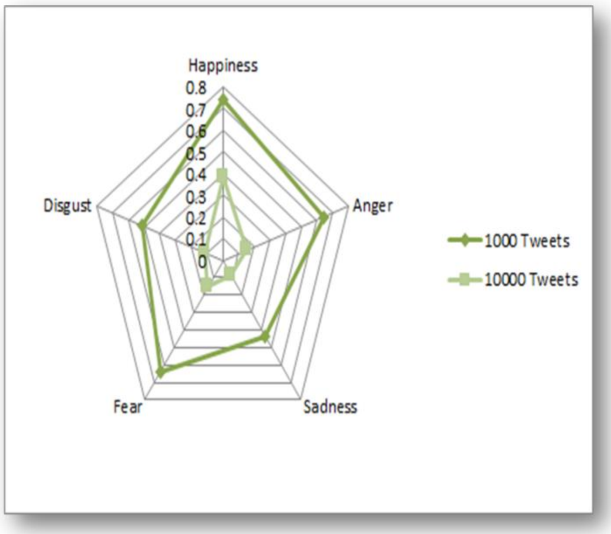
For 100 Tweets



```
Input User Name ->              #AAPHelpLine

        Run                         Exit

Beta-Version @TechnoDesign      v.001

Happiness :                     0.38211888782358594

Anger :                         0.1489523809523809

Sadness :                       0.08292370711725545

Fear :                          0.15514285714285703

Disgust :                       0.11220852318413285
```

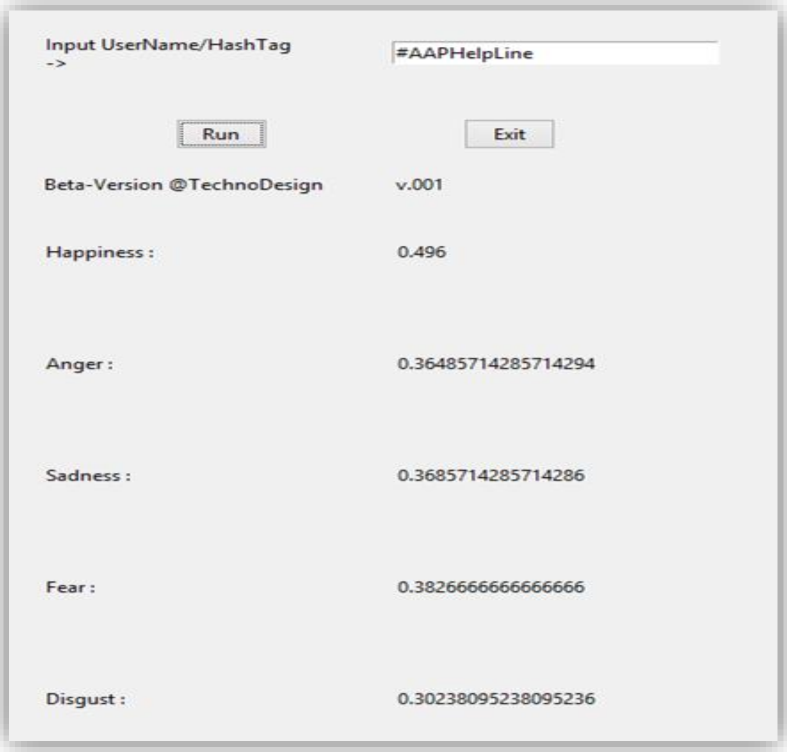
For 1000 Tweets





PLOTTED EMOTION VALUES

Working of Scoring Module 2

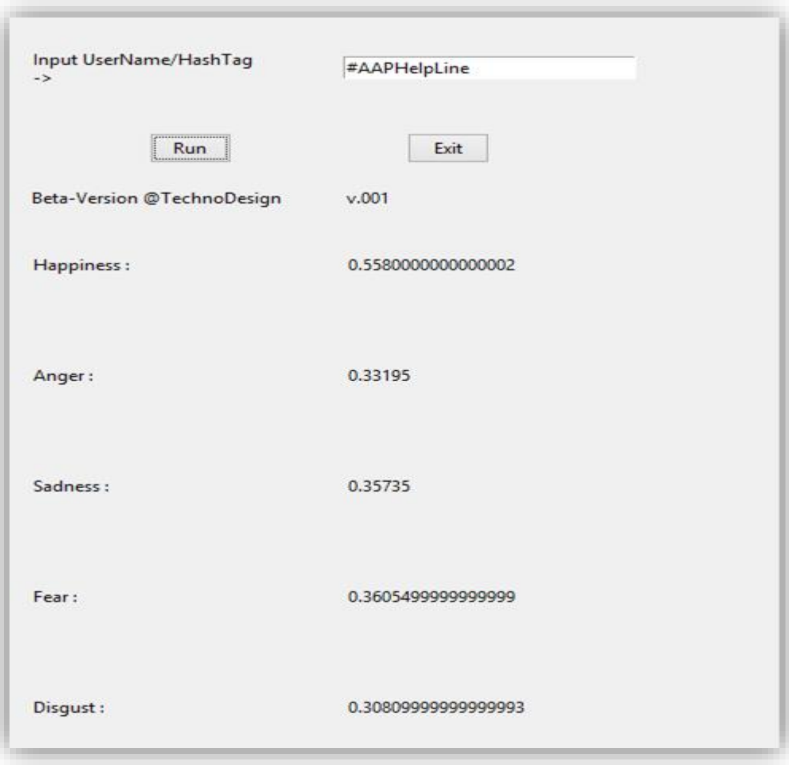


Input UserName/HashTag -> #AAPHelpLine

Beta-Version @TechnoDesign v.001

Happiness :	0.496
Anger :	0.36485714285714294
Sadness :	0.3685714285714286
Fear :	0.3826666666666666
Disgust :	0.30238095238095236

FOR 100 TWEETS

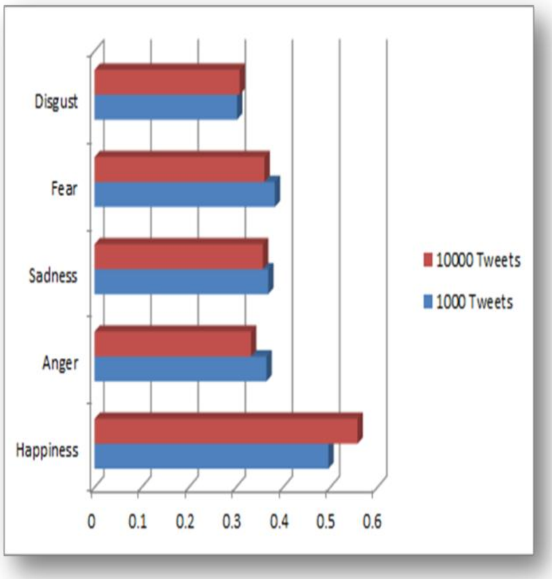


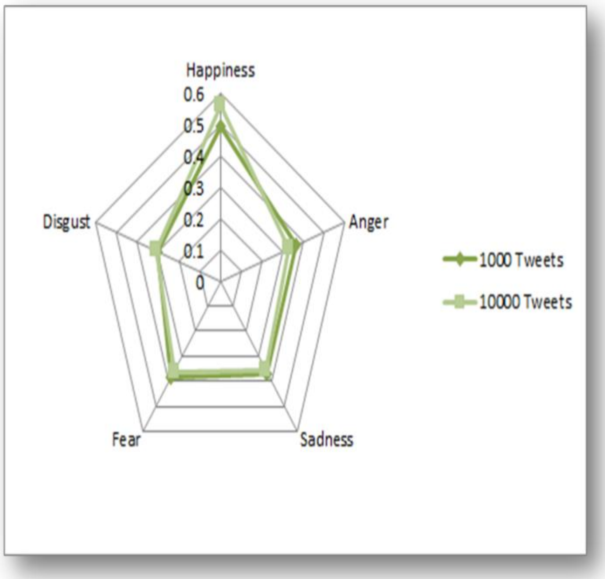
Input UserName/HashTag -> #AAPHelpLine

Beta-Version @TechnoDesign v.001

Happiness :	0.5580000000000002
Anger :	0.33195
Sadness :	0.35735
Fear :	0.3605499999999999
Disgust :	0.3080999999999993

FOR 1000 TWEETS

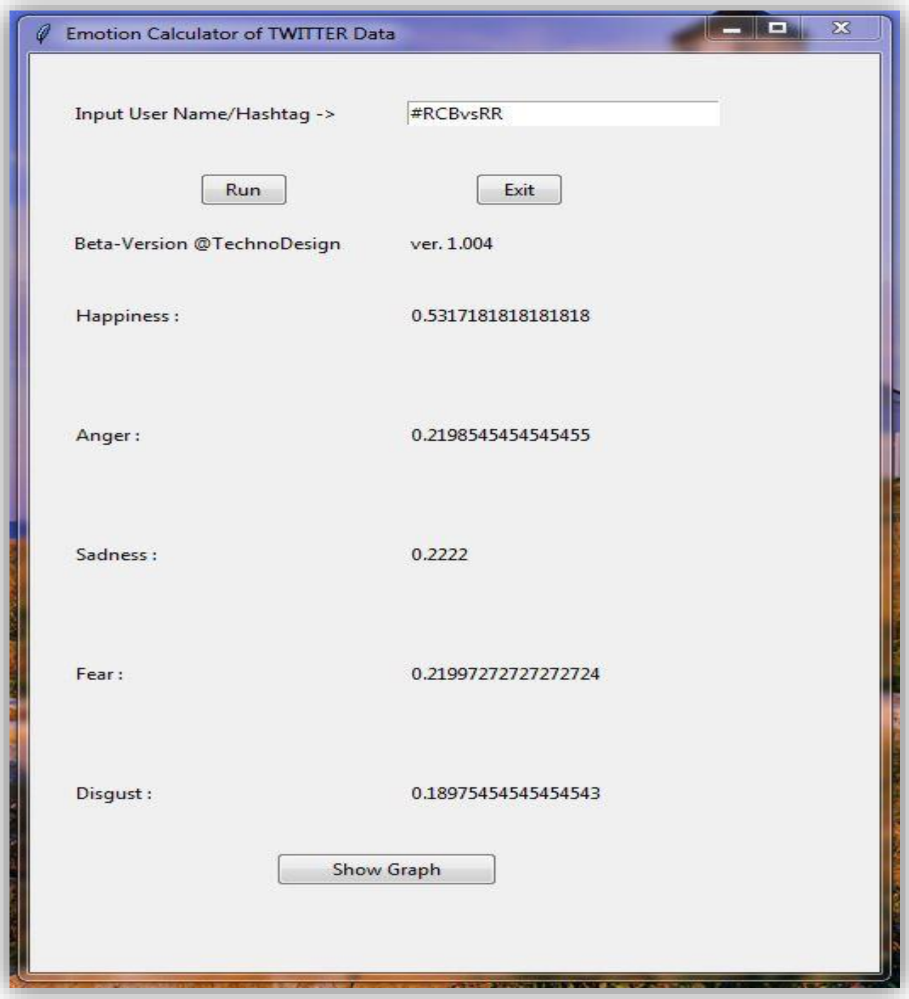




PLOTTED EMOTION VALUES

From the above plots, we can see that Scoring Module 1 has given better results and thus we have continued our work with Scoring Module 1. This was evident when we read some of the tweets downloaded to cross verify.

#RCBvsRR



Input User Name/Hashtag -> #RCBvsRR

Beta-Version @TechnoDesign ver. 1.004

Happiness : 0.5317181818181818

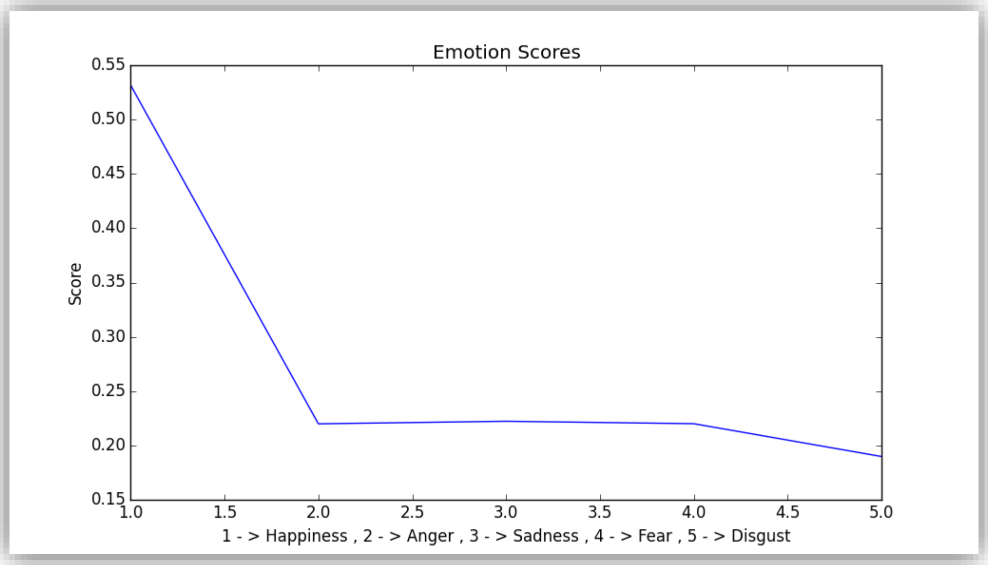
Anger : 0.2198545454545455

Sadness : 0.2222

Fear : 0.21997272727272724

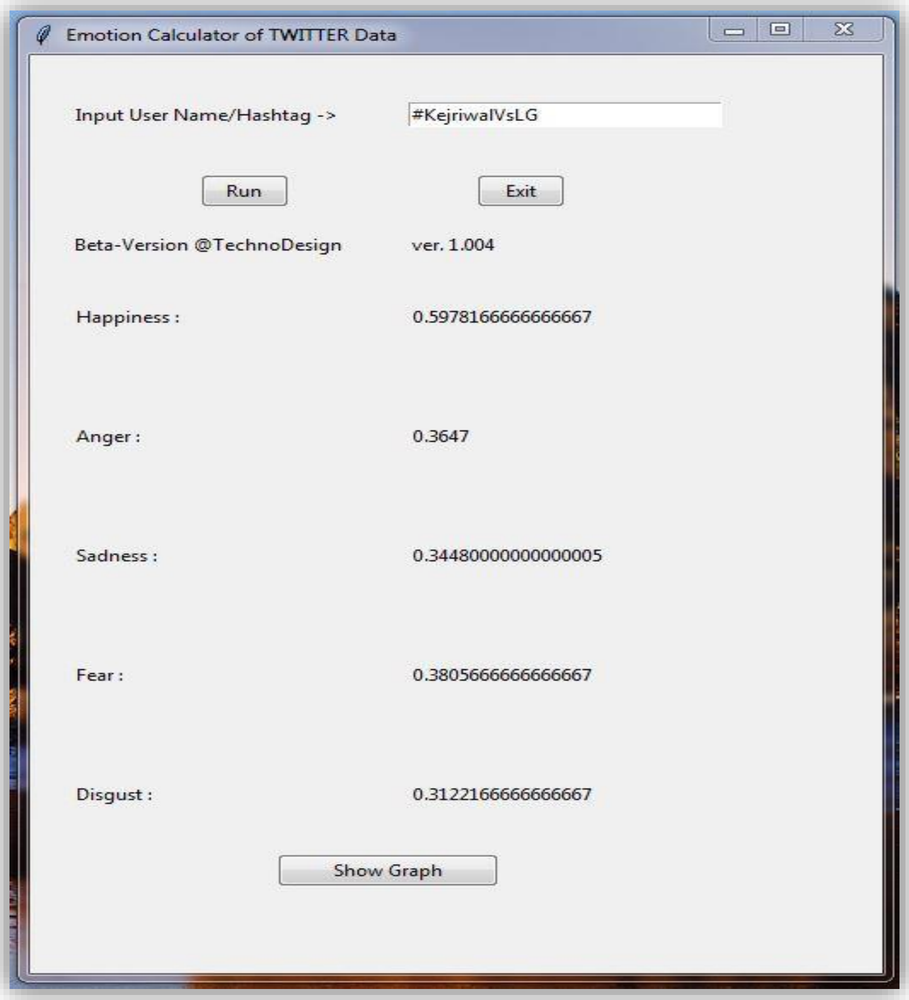
Disgust : 0.18975454545454543

Emotion Scores of hashtag RCBvsRR



Graph showing that The trend was towards emotion being happiness

- #KejriwalVsLG

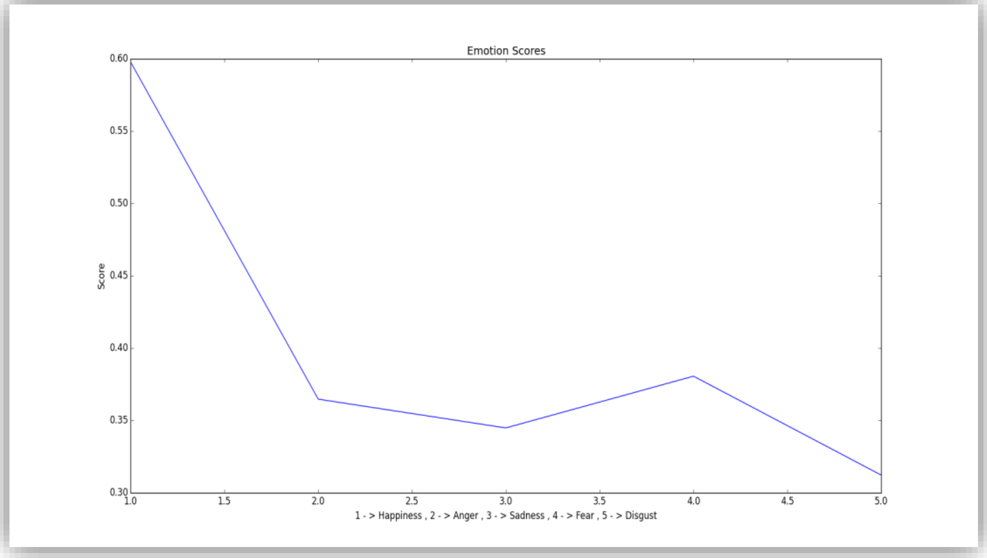


Input User Name/Hashtag -> #KejriwalVsLG

Beta-Version @TechnoDesign ver. 1.004

Happiness :	0.5978166666666667
Anger :	0.3647
Sadness :	0.34480000000000005
Fear :	0.3805666666666667
Disgust :	0.3122166666666667

EMOTION SCORES OF HASHTAG KEJRIWALVSLG



GRAPH SHOWING THAT THE TREND WAS TOWARDS EMOTION BEING HAPPINESS

V.) Applications and Future Work

Applications:

- Review-Related Websites
  - Movie Reviews
  - Product Reviews

- Business and Government Intelligence
  - Knowing Consumer attitudes
  - Trends

- Knowing public opinions for political leaders and work being done by government.

Following is the list of functionalities/modules that will be added in the model:

- **Emoticon processor:** Emoticons are heavily used in tweets but since they are non-ASCII characters so they are removed in the cleaning module. So they need to be identified before that. Right now we don't have pre-defined values for these emoticons so we haven't added such module yet.
- **Apriori Association:** We haven't used any association rule between adverb/verb-adjective pair (or triplet). This method will provide a method that shows how much the adverb/verb affects the following adjective. It will provide stronger lexical affinity between the words and hence better contextual understanding by the machine.
- **KNN Classifier:** This classifier has to be used after the scoring module. When we encounter a tweet from the same hashtag we will classify using the kNN classification method into one of the pre-defined classes. The actual use of this kNN classifier will be in a separate application that will use the functionality of all the modules. We can enter a tweet and then compare it with other tweets of the same hash tags and then classify that tweet into one of the five emotion classes.

VI.) **References**

1)ComScore/the Kelsey group, “Online consumer-generated reviews have significant impact on offline purchase behavior,” Press Release,http://www.comscore.com/press/release.asp?press=1928, November 2007 [March 2015]

2) M. Bansal, C. Cardie, and L. Lee, “The power of negative thinking: Exploiting label disagreement in the min-cut classification framework,” in *Proceedings of the International Conference on Computational Linguistics (COLING)*, 2008. (Poster paper) [March 2015]

3) L. Rainie and J. Horrigan, “Election 2006 online,” Pew Internet &AmericanLife Project Report, January 2007. [March 2015]

4) R. Picard, *Affective Computing*. MIT Press, 1997 [March 2015]

5) B. Liu, “Web data mining; Exploring hyperlinks, contents, and usage data,” *Opinion Mining*. Springer, 2006 [March 2015]

6) T. Nasukawa and J. Yi, “Sentiment analysis: Capturing favorability using natural language processing,” in *Proceedings of the Conference on Knowledge Capture (K-CAP)*, 2003 [March 2015]

7)J. Yi, T. Nasukawa, R. Bunescu, and W. Niblack, “Sentiment analyzer: Extracting sentiments about a given topic using natural language processing techniques,” in *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, 2003. [March 2015]

8)Akshi Kumar, Teeja Mary Sebastian. Sentiment Analysis on Twitter, 2012 [March 2015]

9) Twitter. “REST API”. Internet: https://dev.twitter.com/rest/public, February 2015 [March 2015]

10) Twitter. “Streaming API”. Internet: https://dev.twitter.com/streaming/overview, February 2015 [March 2015]

11) ReadTheDocs. “Tweepy”. Internet: https://media.readthedocs.org/pdf/tweepy/latest/tweepy.pdf, February 2015 [March 2015]

12) Twitter. “Downloading Tweets”. Internet: https://dev.twitter.com/oauth, February 2015 [March 2015]

VII.) Appendix

A.) Some Common Terms

To see that the distinctions in common usage can be subtle, consider how interrelated the following set of definitions given in *Merriam-Webster's Online Dictionary* are:

[Synonyms: opinion, view, belief, conviction, persuasion, sentiment mean a judgment one holds as true.]

- Opinion implies a conclusion thought out yet open to dispute each expert seemed to have a different opinion
- View suggests a subjective opinion, very assertive in stating his views.
- Belief implies often deliberate acceptance and intellectual assent ; a firm belief in her party's platform.
- Conviction applies to a firmly and seriously held belief; the conviction that animal life is as sacred as human.
- Persuasion suggests a belief grounded on assurance (as by evidence) of its truth; was of the persuasion that everything changes.
- Sentiment suggests a settled opinion reflective of one's feelings; her feminist sentiments are well-known.

B.) Source Code

<u>The Twitter Application credentials</u>

```
consumer_key = "RGVO3CTujE60TW5IQy1JwmyxF"

consumer_secret = "ziWzApZCAqlwOt3xK3L0B02VjEsDFZg4Fniy76TsTLKgtnjqlG"

access_key = "1587880604-1tjWpdETzVE4fPALCGeNs6O2oHi4y8ShwIsDQSl"

access_secret = "wJHBahm2y3KnTXuuj2JX18GAolaHVMnLKZ7Ygc0LxnQMH"

auth = tweepy.OAuthHandler(consumer_key, consumer_secret)

auth.set_access_token(access_key, access_secret)
```

<u>Function to download tweets corresponding to a hash tag.</u>

```
hashtag = name


api = tweepy.API(auth)

print("FYI , the following trending topics are available")

   trends1 = api.trends_place(1)

trends = set([trend['name'] for trend in trends1[0]['trends']])

print (trends)
```

```
print("Tweets downloading has started !!")

    '''
cricTweet = tweepy.Cursor(api.search, q=hashtag).items(179)
for tweet in cricTweet:
print (tweet.text)
with open('hashtag_tweets','w') as f:
        for tweet in cricTweet:
        f.write(tweet.text)
    '''

searchQuery = '#someHashtag'  # this is what we're searching for
maxTweets = 5000 # Some arbitrary large number
tweetsPerQry = 100  # this is the max the API permits
fName = 'tweets.txt' # 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 = -1

tweetCount = 0
print("Downloading max {0} tweets".format(maxTweets))
with open(fName, 'w') as f:
whiletweetCount<maxTweets:
try:
if (max_id<= 0):
if (not sinceId):
new_tweets = api.search(q=hashtag, count=tweetsPerQry)
else:
new_tweets = api.search(q=hashtag, count=tweetsPerQry, since_id=sinceId)
```

```
else:
if (not sinceId):
new_tweets = api.search(q=hashtag, count=tweetsPerQry, max_id=str(max_id - 1))
else:
new_tweets = api.search(q=hashtag, count=tweetsPerQry, max_id=str(max_id - 1), since_id=sinceId)
if not new_tweets:
print("No more tweets found")
break
list_tweets = []
for tweet in new_tweets:
list_tweets.append(tweet.text)
tweetCount += len(new_tweets)
print("Downloaded {0} tweets".format(tweetCount))
max_id = new_tweets[-1].id
excepttweepy.TweepError as e:
            # Just exit if any error
print("some error : " + str(e))
break
  z=open(hashtag+'.txt','w')
for tweet in list_tweets:
tweet=tweet.encode('ascii','ignore')
z.write(tweet.decode())
z.write("\n")
z.close()
returnlist_tweets
    #print ("Downloaded {0} tweets, Saved to {1}".format(tweetCount, fName))
```

Function to download tweets corresponding to an username.

```
defget_all_tweets(username):

alltweets = []
user_timeline = twitter.get_user_timeline(screen_name=username,count=200,include_rts=1)
alltweets.extend(user_timeline)
oldest = alltweets[-1]['id'] - 1

  #keep grabbing tweets until there are no tweets left to grab
```

```
whilelen(user_timeline) > 0:
user_timeline =
twitter.get_user_timeline(screen_name=username,count=200,include_rts=1,max_id=oldest)
alltweets.extend(user_timeline)
oldest = alltweets[-1]['id']

tweets = []
for tweet in alltweets:
tweets.append(tweet['text'])

return tweets
```

<u>Data cleaning Functions:</u>

```
defisEnglish(s):
try:
s.encode('ascii')
exceptUnicodeEncodeError:
return False
else:
return True

#The following function removes the part of the string that contains the substring eg. if
#substring = 'http' , then http://www.google.com is removed, that means, remove until a space is
found
defrem_substring(tweets,substring):
   m=0;
   #print(len(tweets))
fori in tweets:
whilei.find(substring)!=-1:
         k=i.find(substring)
         d=i.find(' ',k,len(i))
if d!=-1:              #substring is present somwhere in the middle(not the end of the string)
i=i[:k]+i[d:]
else:                  #special case when the substring is present at the end, we needn't append the
i=i[:k]            #substring after the junk string to our result
```

```
tweets[m]=i #store the result in tweets "list"
              #print(i)
        m += 1
return tweets

#The following function removes the non English tweets .Makes use of the above written isEnglish
Function
defremoveNonEnglish(tweets):
result=[]
fori in tweets:
ifisEnglish(i):
result.append(i)
return result



#the following function converts all the text to the lower case
deflower_case(tweets):
result=[]
fori in tweets:
result.append(i.lower())
return result

defrem_punctuation(tweets,punct):
    #print(len(tweets))
    m=0
fori in tweets:
i=i.replace(punct,'')
tweets[m]=i
        m=m+1
return tweets
```

<u>Scoring Module 1:</u>

```
def score(name):
filename = 'data_emotions_words_list.csv'
count_adj=0
```

```
count=0
   #ans=[]
ans.append(0)
ans.append(0)
ans.append(0)
ans.append(0)
ans.append(0)
special_word=None
special_score=None

   #r=open(sys.argv[1],'r')
for line in name:
line=line.lower()
line=line.replace("."," ")
line=line.split(" ")
list_words=line
      #print(list_words[0])
for word in list_words:
with open(filename,encoding="ISO-8859-1",newline='') as f:
reader = csv.reader(f)
for row in reader:
my_list=row
word=word.lower()
if (word == my_list[0] and len(word)>=1):
ans_list=[]
count_adj+=1
                  #print(count)
ans_list.append(word)
ans_list.append(my_list[1])
ans_list.append(my_list[3])
ans_list.append(my_list[5])
ans_list.append(my_list[7])
ans_list.append(my_list[9])
ifspecial_word is None:
ans[0]+=float(my_list[1])
```

```
ans[1]+=float(my_list[3])
ans[2]+=float(my_list[5])
ans[3]+=float(my_list[7])
ans[4]+=float(my_list[9])
else:
print(special_word,word)
print(special_score)
ifspecial_score>=0:
ans[0]+=float(my_list[1])*max(0.5,float(special_score))
ans[1]+=float(my_list[3])*max(0.5,float(special_score))
ans[2]+=float(my_list[5])*max(0.5,float(special_score))
ans[3]+=float(my_list[7])*max(0.5,float(special_score))
ans[4]+=float(my_list[9])*max(0.5,float(special_score))
else:
ans[0]+=5-float(my_list[1])
ans[1]+=5-float(my_list[3])
ans[2]+=5-float(my_list[5])
ans[3]+=5-float(my_list[7])
ans[4]+=5-float(my_list[9])
special_word=None
            #final_list.append(ans_list)
            #print(ans_list)
break

with open('adverb.csv',encoding="ISO-8859-1",newline='') as a:
reader=csv.reader(a)
for row in reader:
list_adverb=row
          #print(list_adverb)
word=word.lower()
if (word==list_adverb[0]):
count+=1
ans_list=[]
ans_list.append(word)
ans_list.append(list_adverb[1])
ans_list.append(list_adverb[1])
ans_list.append(list_adverb[1])
ans_list.append(list_adverb[1])
ans_list.append(list_adverb[1])
            #ans[0]+=float(list_adverb[1])
            #ans[1]+=float(list_adverb[1])
            #ans[2]+=float(list_adverb[1])
            #ans[3]+=float(list_adverb[1])
            #ans[4]+=float(list_adverb[1])
ifspecial_word is None:
special_word=word
special_score=float(list_adverb[1])
else:
special_word=word
special_score=float(special_score)*float(list_adverb[1])

            #print(ans_list)
break

with open('verb.csv',encoding="ISO-8859-1",newline='') as v:
reader=csv.reader(v)
for row in reader:
list_verb=row
word=word.lower()
          #print(list_adverb)
if (word==list_verb[0]):
count+=1
ans_list=[]
ans_list.append(word)
ans_list.append(list_verb[1])
ans_list.append(list_verb[1])
ans_list.append(list_verb[1])
ans_list.append(list_verb[1])
ans_list.append(list_verb[1])
            #ans[0]+=float(list_verb[1])
```

```
                        #ans[1]+=float(list_verb[1])
                        #ans[2]+=float(list_verb[1])
                        #ans[3]+=float(list_verb[1])
                        #ans[4]+=float(list_verb[1])
ifspecial_word is None:
special_word=word
special_score=float(list_verb[1])
else:
special_word=word
special_score=float(special_score)*float(list_verb[1])

                        #print(ans_list)
break



    #print(count)
    #print(ans)
fori in range(0,5):
        #ans[i]=ans[i]/((5*count_adj)+count)
ans[i]=ans[i]/(5*count_adj)
print(ans)
returnans
```

<u>Scoring Module 2:</u>

```
def score(name):
    #count_adj=0
count=0
ans=[]
ans.append(0)
ans.append(0)
ans.append(0)
ans.append(0)
ans.append(0)
    #r=open(name,'r')
count_adj=0
    #count_verb=0
```

```
adj_ans=[]
adj_ans.append(0)
adj_ans.append(0)
adj_ans.append(0)
adj_ans.append(0)
adj_ans.append(0)
adv_ans=[]
adv_ans.append(0)
adv_ans.append(0)
adv_ans.append(0)
adv_ans.append(0)
adv_ans.append(0)



  #r=open(sys.argv[1],'r')
for line in name:
line=line.lower()
line=line.replace("."," ")
line=line.split(" ")
list_words=line
  #    print(list_words[0])
for word in list_words:
with open("data_emotions_words_list.csv",encoding="ISO-8859-1",newline='') as f:
reader = csv.reader(f)
for row in reader:
my_list=row
word=word.lower()
if (word == my_list[0] and len(word)>=1):
ans_list=[]
count_adj+=1
            #print(count)
ans_list.append(word)
ans_list.append(my_list[1])
ans_list.append(my_list[3])
ans_list.append(my_list[5])
ans_list.append(my_list[7])
ans_list.append(my_list[9])
adj_ans[0]+=float(my_list[1])
adj_ans[1]+=float(my_list[3])
adj_ans[2]+=float(my_list[5])
adj_ans[3]+=float(my_list[7])
adj_ans[4]+=float(my_list[9])

                    #final_list.append(ans_list)
                    #print(ans_list)
break

with open('adverb.csv',encoding="ISO-8859-1",newline='') as a:
reader=csv.reader(a)
for row in reader:
list_adverb=row
                    #print(list_adverb)
word=word.lower()
if (word==list_adverb[0]):
count+=1
ans_list=[]
ans_list.append(word)
ans_list.append(list_adverb[1])
ans_list.append(list_adverb[1])
ans_list.append(list_adverb[1])
ans_list.append(list_adverb[1])
ans_list.append(list_adverb[1])
adv_ans[0]+=float(list_adverb[1])
adv_ans[1]+=float(list_adverb[1])
adv_ans[2]+=float(list_adverb[1])
adv_ans[3]+=float(list_adverb[1])
adv_ans[4]+=float(list_adverb[1])


                        #print(ans_list)
```

```
break

with open('verb.csv',encoding="ISO-8859-1",newline='') as v:
reader=csv.reader(v)
for row in reader:
list_verb=row

word=word.lower()
                    #print(list_adverb)
if (word==list_verb[0]):
count+=1
ans_list=[]
ans_list.append(word)
ans_list.append(list_verb[1])
ans_list.append(list_verb[1])
ans_list.append(list_verb[1])
ans_list.append(list_verb[1])
ans_list.append(list_verb[1])
adv_ans[0]+=float(list_verb[1])
adv_ans[1]+=float(list_verb[1])
adv_ans[2]+=float(list_verb[1])
adv_ans[3]+=float(list_verb[1])
adv_ans[4]+=float(list_verb[1])

                        #print(ans_list)
break



ifadj_ans[0]/count_adj>3:
            ans[0]=((adj_ans[0]/3.725)+adv_ans[0])/(count+count_adj)
else:
            ans[0]=((adj_ans[0]/3.725)-adv_ans[0])/(count+count_adj)

ifadj_ans[2]/count_adj>3:
            ans[2]=((adj_ans[2]/3.4875)+adv_ans[2])/(count+count_adj)
```

```
else:
        ans[2]=((adj_ans[2]/3.4875)-adv_ans[2])/(count+count_adj)

ifadj_ans[4]/count_adj>3:
        ans[4]=((adj_ans[4]/2.665)+adv_ans[4])/(count+count_adj)
else:
        ans[4]=((adj_ans[4]/2.665)-adv_ans[4])/(count+count_adj)

ifadj_ans[1]/count_adj>3:
        ans[1]=((adj_ans[1]/2.5)+adv_ans[1])/(count+count_adj)
else:
        ans[1]=((adj_ans[1]/2.5)-adv_ans[1])/(count+count_adj)

ifadj_ans[3]/count_adj>3:
        ans[3]=((adj_ans[3]/2.5)+adv_ans[3])/(count+count_adj)
else:
        ans[3]=((adj_ans[3]/2.5)-adv_ans[3])/(count+count_adj)

print(ans)
returnans
```

POS Tagger Function used to identify the adjectives,verbs,adverbs.

```
defPOS_tagger(tweets,username):
        x=[]
#for each line in tweets list
        for line in tweets:
                tokenized=nltk.sent_tokenize(line)
                t=""
                #for each sentence in the line
                for sent in tokenized:
                        #tokenize this sentence
                        text=nltk.word_tokenize(sent)
                        k=nltk.pos_tag(text)
                        fori in k:
                                if (i[1][:2]=="VB" or i[1][:2]=="JJ") or i[1][:2]=="RB":
                                    t=t+i[0]+' '

                x.append(t)
        filename="pos_tagged_"+username+".txt"
        handle=open(filename,"w")
        fori in x:
                handle.write(i+'\n')
```

Function to expand the dictionary:

```
ef learn(name):
list = []
filename = 'data_emotions_words_list.csv'
fname = 'new_dict.csv'
dict_list = []
new_list = []
   l = 0
oh = []
oa = []
os = []
od = []
of = []
oc = []

with open(fname,encoding="ISO-8859-1",newline='') as n: #new dictionary
new_reader = csv.reader(n)
fornew_row in new_reader:
my_new_list = new_row
         #new_list.append(my_new_list[0])
dict_list.append(my_new_list[0])
oh.append(float(my_new_list[1]))
oa.append(float(my_new_list[2]))
os.append(float(my_new_list[3]))
of.append(float(my_new_list[4]))
od.append(float(my_new_list[5]))
oc.append(float(my_new_list[6]))
```

```
print(dict_list[l]+","+str(oh[l]))



            l += 1


list_evalue = []
my_new_list = []
    b = []

    j = 0



words = []
    #ans = [0.52,0.32,0.3,0.36,0.28]
    '''
ans[0] = 0.48
ans[1] = 0.32
ans[2] = 0.3
ans[3] = 0.36
ans[4] = 0.28
    '''
fi = open(name,'r')
tweets = fi.readlines()
for tweet in tweets:
if(tweet == '\n'):
continue
words[:] = []
words = tweet.split(' ')
for word in words:
if(word == '\n'):
continue
            #print(word)
word=word.lower()
with open(filename,encoding="ISO-8859-1",newline='') as di:  #original dictionary
reader = csv.reader(di)
```

```
for row in reader:
my_list=row
if(word != my_list[0]):                    # word not in original dictionary

if(word in new_list):
if(word in dict_list):             # word in new dictionary and in tweet
continue
else:
                              j = new_list.index(word)       # repeated words
c[j] += 1
else:
                          l += 1
new_list.append(word)              # word not in new dictionary
h.append(ans[0])
a.append(ans[1])
s.append(ans[2])
f.append(ans[3])
d.append(ans[4])
c.append(1)

                          #print(new_list[l-1]+","+str(h[l-1])+","+str(a[l-1])+","+str(s[l-1])+","+str(f[l-
1])+","+str(d[l-1])+","+str(c[l-1]))



total = l
    z = 0
    '''
cnt = 0
with open(fname,encoding="ISO-8859-1",newline='') as n: #new dictionary
new_reader = csv.reader(n)
fornew_row in new_reader:
cnt += 1
    '''

while(z<len(new_list)):
```

```
if(c[z]>=1000):
c[z] = round((c[z]/1036),1)
        z = z + 1



print("\n"+str(len(dict_list))+"\n")
print(len(new_list))
print("\n")
print(l)
i = 0

    k = 0
    z = 0
    '''
    flag2 = []
while(i<len(dict_list)):
flag2[i] = 0
i += 1
    '''
    flag2 = [0]*len(dict_list)
    flag3 = [0]*len(new_list)
while(z<len(new_list)):
i = 0
flag = 0
while(i<len(dict_list) and flag == 0):



if(new_list[z] == dict_list[i] and flag3[z] == 0):        # in tweet and in dictionary
flag = 1
flag2[i] = 1
flag3[z] = 1
list.append(new_list[z])
dh.append((h[z]*c[z] + oh[i]*oc[i])/(c[z]+oc[i]))
da.append((a[z]*c[z] + oa[i]*oc[i])/(c[z]+oc[i]))
ds.append((s[z]*c[z] + os[i]*oc[i])/(c[z]+oc[i]))
```

```
df.append((f[z]*c[z] + of[i]*oc[i])/(c[z]+oc[i]))
dd.append((d[z]*c[z] + od[i]*oc[i])/(c[z]+oc[i]))
dc.append(c[z]+oc[i])

else:
if(flag2[i] == 0):                    # only in dictionary
list.append(dict_list[i])
dh.append(oh[i])
da.append(oa[i])
ds.append(os[i])
df.append(of[i])
dd.append(od[i])
dc.append(oc[i])
flag2[i] = 1
i += 1

if(flag == 0 and flag3[z] == 0):
list.append(new_list[z])
dh.append(h[z])              # in tweet only
da.append(a[z])
ds.append(s[z])
df.append(f[z])
dd.append(d[z])
dc.append(c[z])

#print(list[z]+","+str(dh[z])+","+str(da[z])+","+str(ds[z])+","+str(df[z])+","+str(dd[z])+","+str(dc[z]))
        z += 1
    z = 0
    y = open("new_dict.csv",'w')
while(z<len(list)):
print(list[z]+","+str(dh[z])+","+str(da[z])+","+str(ds[z])+","+str(df[z])+","+str(dd[z])+","+str(dc[z]))

y.write(list[z]+","+str(dh[z])+","+str(da[z])+","+str(ds[z])+","+str(df[z])+","+str(dd[z])+","+str(dc[z])
)
y.write("\n")
        z = z + 1
```

```
y.close()
```

Main Program:

```
def run(username, master):
    column0_padx = 24
row_pady = 36
tweets = []
print (username)
if(username[0] == "@"):
tweets = get_all_tweets(username)
print("Downloading of tweets of user has started !!")
if(username[0] == "#"):
tweets = get_all_hash(username)
print("Downloading of tweets of hashtag has started !!")

print("Tweets have been downloaded !!")
print("Now Cleaning of tweets starts !!")

    #time.sleep(1)
    '''
filename = username+"tweets.txt"

with open(filename) as f:
for line in f:
tweets.append(line)'''

tweets=rem_substring(tweets,'#')
tweets=rem_substring(tweets,'http')
tweets=rem_substring(tweets,'@')
tweets=rem_substring(tweets,'RT')
tweets=rem_punctuation(tweets,'\"')
tweets=rem_punctuation(tweets,'-')
tweets=rem_punctuation(tweets,'!')
tweets=rem_punctuation(tweets,':')
tweets=removeNonEnglish(tweets)
```

```
    #tweets.replace("."," ")
for tweet in tweets:
tweet=tweet.replace("."," ")



    z=open('cleaned_'+username+'.txt','w')
for tweet in tweets:
tweet=tweet.encode('ascii','ignore')
z.write("\n")
z.write(tweet.decode())
z.close()
    #filename = username+"_cleaned_tweets.txt"
    #x = open(filename,"a")
    '''for i in tweets:
x.write(i+'\n')
    #time.sleep(2)'''
POS_tagger(tweets,username)
print("Tweets have now been cleaned !!")
evalue = score(tweets)

learn("pos_tagged_"+username+".txt")
    L3 = Label(master, text="Happiness : ", wraplength=150, justify='left', pady=row_pady)
L3.grid(row=5, column=0, sticky='w', padx=column0_padx)
    L8 = Label(master, text=str(evalue[0]))
L8.grid(row=5, column=1, sticky='w')
    L4 = Label(master, text="Anger : ", wraplength=150, justify='left', pady=row_pady)
L4.grid(row=6, column=0, sticky='w', padx=column0_padx)
    L9 = Label(master, text=str(evalue[1]))
L9.grid(row=6, column=1, sticky='w')
    L5 = Label(master, text="Sadness : ", wraplength=150, justify='left', pady=row_pady)
L5.grid(row=7, column=0, sticky='w', padx=column0_padx)
    L10 = Label(master, text=str(evalue[2]))
L10.grid(row=7, column=1, sticky='w')
    L6 = Label(master, text="Fear : ", wraplength=150, justify='left', pady=row_pady)
L6.grid(row=8, column=0, sticky='w', padx=column0_padx)
```

```
   L11 = Label(master, text=str(evalue[3]))
L11.grid(row=8, column=1, sticky='w')
   L7 = Label(master, text="Disgust : ", wraplength=150, justify='left', pady=row_pady)
L7.grid(row=9, column=0, sticky='w', padx=column0_padx)
   L12 = Label(master, text=str(evalue[4]))
L12.grid(row=9, column=1, sticky='w')

bottom_fram = Frame(master)
bottom_fram.grid(row=10, column=0, columnspan=2, sticky='w')

btn_start = ttk.Button(bottom_fram, text = "Show Graph", width=20, command= lambda:
new_window())
btn_start.pack(side='left', padx=145)
print("Tweets have now been cleaned !!")

defnew_window():
id = "Graph"
window = Toplevel(master)
label = ttk.Label(window, text=id)
label.pack(side="top", fill="both", padx=10, pady=10)

     f = Figure(figsize=(5,5), dpi=100)
     a = f.add_subplot(111)
     #t = arange(0.0,3.0,0.01)
     #s = sin(2*pi*t)
     #a.plot(t,s)
a.plot([1,2,3,4,5],[evalue[0], evalue[1], evalue[2], evalue[3], evalue[4]])
a.set_title('Emotion Scores')
a.set_xlabel('1 - > Happiness , 2 - > Anger , 3 - > Sadness , 4 - > Fear , 5 - > Disgust ')
a.set_ylabel('Score')

     # atk.DrawingArea
canvas = FigureCanvasTkAgg(f, master=window)
canvas.show()
canvas.get_tk_widget().pack(side=TOP, fill=BOTH, expand=1)
```

```
toolbar = NavigationToolbar2TkAgg( canvas, window )
toolbar.update()
        canvas._tkcanvas.pack(side=TOP, fill=BOTH, expand=1)

defon_key_event(event):
print('you pressed %s'%event.key)
key_press_handler(event, canvas, toolbar)

canvas.mpl_connect('key_press_event', on_key_event)

def _quit():
window.quit()     # stops mainloop
window.destroy()  # this is necessary on Windows to prevent
                            # Fatal Python Error: PyEval_RestoreThread: NULL tstate

button = ttk.Button(master=window, text='Quit', command=_quit)
button.pack(side=BOTTOM)

class App:
def __init__(self, master):
self.root = Frame(master)
        column0_padx = 24
row_pady = 36

        #Label entry
userart = Label(
master, text="Input User Name -> ",
wraplength=150, justify='left', pady=row_pady)
entry_point = Entry(master, width=30)
userart.grid(row=1, column=0, sticky='w', padx=column0_padx)
entry_point.grid(row=1, column=1, sticky='w')

        # version
lbl_version = ttk.Label(master, text="Beta-Version @TechnoDesign")
```

```
version = ttk.Label(master, text="ver. 1.004")
lbl_version.grid(row=4, column=0, sticky='w', padx=column0_padx)
version.grid(row=4, column=1, sticky='w')

sep = ttk.Label(master)
sep.grid(row=3, column=0, sticky='w')


    #progress_bar
    #progressbar = ttk.Progressbar(orient='horizontal', length=200, mode='determinate')
    #progressbar.grid(row=5, column=0, sticky='w', padx=column0_padx)
    #progressbar.start()

    # buttons
bottom_frame = Frame(master)
bottom_frame.grid(row=2, column=0, columnspan=2, sticky='w')

btn_start = ttk.Button(bottom_frame, text = "Run", width=7, command=lambda:
run(entry_point.get(), master))
btn_start.pack(side='left', padx=100)
btn_exit = ttk.Button(bottom_frame, text="Exit", width=7, command=self.root.quit)
btn_exit.pack(side='left', padx=10)

root = Tk()
root.title("Emotion Calculator of TWITTER Data")
root.minsize(500, 700)
app = App(root)
root.mainloop()
```