# INTERNET MONITORING - DATA ANALYSIS - SENTIMENTAL ANALYSIS - MARKET ANALYSIS ON HASHTAGS IN TWITTER AND GOOGLE USING PYTHON

|  |  |
| --- | --- |
| **A PROJECT REPORT** | |
| ***Submitted by*** |  |
| **SANGEETH SUBRAMONIAM** | **( 210415104093 )** |
| **COMPUTERSCIENCE ENGINEERING** |  |

***in partial fulfillment for the award of the degree***

*of*

# BACHELOR OF ENGINEERING

*in*

## COMPUTER SCIENCE AND ENGINEERING CHENNAI INSTITUTE OF TECHNOLOGY, CHENNAI

**ANNA UNIVERSITY::CHENNAI 600025**

**MARCH 2019**



**ANNA UNIVERSITY:CHENNAI 600025**

**BONAFIDE CERTIFICATE**

Certified that this project report **“INTERNET MONITORING-DATA ANALYSIS -SENTIMENTAL ANALYSIS - STOCK MARKET ANALYSIS ON HASHTAGS IN TWITTER USING PYTHON”** is the bonafide work done by **SANGEETH SUBRAMONIAM,** Who carried out the work under my supervision. Certified further that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

## SIGNATURE SIGNATURE

**DR.B.SUNDARAMBAL, Mrs.K.RAJALAKSHMI., M.E.,**

**HEAD OF THE DEPARTMENT SUPERVISOR**

Associate Professor Associate Professor

Computer Science and Engineering Computer Science and Engineering Chennai Institute of Technology Chennai Institute of Technology Kundrathur Kundrathur

Chennai – 600 069 Chennai – 600 069

Submitted for University viva voice examination held on ………………

## INTERNAL EXAMINER EXTERNAL EXAMINER

**ACKNOWLEDGEMENT**

We convey our profound thanks and gratitude to our honorable **Chairman Mr.P.SRIRAM,** Chennai Institute of Technology, Chennai for providing me an excellent academic climate, which made this endeavor possible.

We also express our gratitude to our beloved **Principal Dr.K.S.SRINIVASAN, Ph.D.,** who constantly nurtured our standard of education and devoted their precious time for our needs.

We are deeply indebted to pay our sincere thanks to our respectable **Head of the Department, Mrs.SUNDARAMBAL, Ph.D.,** Department of Computer Science Engineering for showing some exuberant consent for our project and providing us with all the facilities in the department to complete the project.

We take immense pleasure to express our heartfelt thanks to our **Guide Mrs.K.RAJALAKSHMI M.E.,** for motivating us to study in this field and for her illuminating guidance and continuous support in the planning and execution of this project.

We thank all our department teaching and non-teaching staffs, who have helped directly and indirectly to complete this project in time.

Last but not least we extend our deep gratitude to our beloved family members for their moral coordination, encouragement and financial support to carry out this project.

# ABSTRACT

This project addresses the problem of sentiment analysis in twitter and Headlines analysis in Google; That is classifying tweets and the Headlines according to the sentiment expressed in them: positive, negative or neutral. Twitter is an online micro-blogging and social-networking platform which allows users to write short status updates of maximum length 140 characters. It is a rapidly expanding service with over 275 million registered users - out of which 100 million are active users and half of them log on twitter on a daily basis - generating nearly 250 million tweets per day. Due to this large amount of usage we hope to achieve a reflection of public sentiment by analysing the sentiments expressed in the tweets. Analysing the public sentiment is important for many applications such as firms trying to find out the response of their products in the market, predicting political elections and predicting socioeconomic phenomena like stock exchange. Also taking into consideration the Critic reviews from various sources including the newspapers and digital media’s by the use of the Google search engine. The aim of this project is to develop a functional classifier for accurate and automatic sentiment classification of an unknown tweet stream and the headline analysed.

# TABLE OF CONTENTS

**CHAPTER NO TITLE PAGE NO**

[ABSTRACT 4](#_TOC_250017)

[LIST OF FIGURES 7](#_TOC_250016)

LIST OF ABBREVATION 8

[LIST OF TABLES 9](#_TOC_250015)

1. INDRODUCTION 10
   1. [AIM 10](#_TOC_250014)
   2. [SYNOPSIS 10](#_TOC_250013)
2. LITERATURE SURVEY 11
3. SYSTEM ANALYSIS 13
   1. [EXISTING SYSTEM 13](#_TOC_250012)
   2. DISADVANTAGES 13
   3. [PROPOSED SYSTEM 14](#_TOC_250011)
   4. ADVANTAGES 17
   5. FUNCTIONAL REQUIREMENT 17
   6. DATA COLLECTION 18
   7. AUTHENTICAION 18
   8. TWITTER API 18
4. SYSTEM REQUIREMENTS 20
   1. [INTRODUCTION 20](#_TOC_250010)
   2. [HARDWARE REQUIREMENTS 20](#_TOC_250009)
   3. [SOFTWARE REQUIREMENTS 20](#_TOC_250008)
5. SYSTEM DESIGN 21
   1. [USE CASE DIAGRAM 21](#_TOC_250007)
   2. [E-R DIAGRAM 22](#_TOC_250006)
   3. [CLASS DIAGRAM 23](#_TOC_250005)
   4. DATATYPES 24
6. SYSTEM STUDY 27
   1. [FEASIBILITY STUDY 27](#_TOC_250004)
      1. ECONOMIC FEASIBILITY 27
      2. [TECHNICAL FEASIBILITY 28](#_TOC_250003)
      3. SCHEDULE FEASIBILITY 28
7. IMPLEMENTATION 29
   1. [MODULES 29](#_TOC_250002)
   2. PROGRAMMING 35
   3. PROGRAMMING STANDARDS 35

7.4[NAMING CONVENTIONS 36](#_TOC_250001)

|  |  |  |
| --- | --- | --- |
|  | 7.4.1RULES  7.4.2TYPE | 36 |
| 7.4.2.1 PACKAGE NAMES | 37 |
| 7.4.2.2 CLASS NAME | 37 |
| 7.4.2.3 METHOD NAMES | 37 |
| 7.4.2.4 CONSTANT NAMES | 38 |
| 7.4.2.5 PARAMETER NAMES | 38 |
| 7.4.3 CAMELCASE DEFINED | 38 |
| **8** | **SYSTEM TESTING** | **40** |
|  | 8.1 TEST DATA AND OUTPUT | 41 |
|  | 8.2 TESTING STRATERGIES | 30 |
|  | 8.2.1 TESTING  8.2.2WHITE BOX TESTING  8.2.3BLACK BOX TESTING  8.2.4INTEGRATION TESTING | 30 |
| **9** | **CONCLUSION** | **50** |
| **10** | **FUTURE ENHANCEMENT** | **51** |
| **11** | **APPENDIX A** | **53** |
| **12** | **SAMPLE CODING** | **53** |
| **13** | **APPENDIX B** | **58** |
| **14** | **SCREENSHOTS** | **58** |
| **15** | **REFERENCES** | **61** |

# LIST OF FIGURES

**FIGURE NO NAME OF THE FIGURE PAGE NO**

|  |  |  |
| --- | --- | --- |
| 5.1 | Use Case Diagram | 21 |
| 5.2 | E-R Diagram | 22 |
| 5.3 | Class Diagram | 23 |
| 7.1.1 | Tweepy Logo | 29 |
| 7.1.1.3 | Pattern Classification | 34 |
| B.1 | User Interface | 58 |
| B.2 | Entering the terms | 59 |
| B.3 | Analyzing and plotting Graph | 60 |

**LIST OF ABBREVIATIONS**

|  |  |
| --- | --- |
|  |  |
| **GUI** | Graphical User interface |
| **API** | Application Program Interface |
| **SQL** | Structured Query language |
| **UML** | Unified Modeling language |
| **ERD** | Entity relationship diagram |
| **IDE** | Integrated Development environment |
| **HTML** | Hyper Text Markup Language |
|  |  |

# LIST OF TABLES

**SNO TABLE NAME PAGE NO**

* 1. FUNCTIONAL REQUIREMENT 17
     1. USER DATA 24
     2. SENTIMENT 24
     3. WATCH 25
     4. RECORDS 25

5.4.5 TWITTER TOKEN 26

**CHAPTER 1 INTRODUCTION**

## AIM

The Aim of this project is to provide an efficient data hiding technique and Video Encryption in which the data and the Data can be retrieved independently

## SYNOPSIS

Sentiment Analysis is a Natural Language Processing and Information Extraction task that aims to obtain writer’s feelings expressed in positive or negative comments, questions and requests, by analyzing a large numbers of documents. Generally speaking, sentiment analysis aims to determine the attitude of a speaker or a writer with respect to some topic or the overall tonality of a document. In recent years, the exponential increase in the Internet usage and exchange of public opinion is the driving force behind Sentiment Analysis today. The Web is a huge repository of structured and unstructured data.

The analysis of sentiments may be document based where the sentiment in the entire document is summarized as positive, negative or objective. It can be sentence based where individual sentences, bearing sentiments, in the text are classified. SA can be phrase based where the phrases in a sentence are classified according to polarity. Sentiment Analysis identifies the phrases in a text that bears some sentiment. The author may speak about some objective facts or subjective opinions. It is necessary to distinguish between the two. SA finds the subject towards whom the sentiment is directed. A text may contain many entities but it is necessary to find the entity towards which the sentiment is directed. It identifies the polarity and degree of the sentiment.

# CHAPTER 2 LITERATURE SURVEY

## Sentiment Knowledge Discovery in Twitter Streaming Data

Micro-blogs are a challenging new source of information for data mining techniques. Twitter is a micro-blogging service built to discover what is happening at any moment in time, anywhere in the world. Twitter messages are short, and generated constantly, and well suited for knowledge discovery using data stream mining. We briefly discuss the challenges that Twitter data streams pose, focusing on classification problems, and then consider these streams for opinion mining and sentiment analysis. To deal with streaming unbalanced classes, we propose a sliding window Kappa statistic for evaluation in time-changing data streams. Using this statistic we perform a study on Twitter data using learning algorithms for data streams.

## Twitter Sentiment Classification using Distant Supervision

We introduce a novel approach for automatically classifying the sentiment of Twitter messages. These messages are classified as either positive or negative with respect to a query term. This is useful for consumers who want to re-search the sentiment of products before purchase, or companies that want to monitor the public sentiment of their brands. There is no previous research on classifying sentiment of messages on microblogging services like Twitter. We present the results of machine learning algorithms for classifying the sentiment of Twitter messages using distant supervision. Our training data consists of Twitter messages with emoticons, which are used as noisy labels. This type of training data is abundantly available and can be obtained through automated means. We show that machine learning algorithms (Naive Bayes, Maximum Entropy, and SVM) have

accuracy above 80% when trained with emoticon data. This paper also describes the preprocessing steps needed in order to achieve high accuracy. The main contribution of this paper is the idea of using tweets with emoticons for distant supervisedlearning.

## Twitter as a Corpus for Sentiment Analysis and Opinion Mining

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 and the google Headlines analysis.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.

# CHAPTER 3 SYSTEM ANALYSIS

## EXISTING SYSTEM

Sentiment analysis of data in the domain of micro-blogging is a relatively new research topic, so there is still a lot of room for further research in this area. Decent amount of related prior work has been done on sentiment analysis of user reviews, documents, web blogs/articles and general phrase level sentiment analysis. These differ from twitter mainly because of the limit of 280 characters per tweet which forces the user to express opinion compressed in very short text. The best results reached in sentiment classification use supervised learning techniques such as Naive Bayes and Support Vector Machines, but the manual labelling required for the supervised approach is very expensive. Some work has been done on unsupervised and semi-supervised approaches, and there is a lot of room of improvement. Various researchers testing new features and classification techniques often just compare their results to baseline performance. There is a need of proper and formal comparisons between these results arrived through different features and classification techniques in order to select the best features and most efficient classification techniques for particular applications

## 3.1.1 Disadvantages

* + - Sentiment analysis can be applied to many areas but arriving at whether a statement is positive or negative can be difficult. The categorization is mainly split into two types: facts and opinion.
    - Facts are expressed about entities, whereas opinions are about their properties.
    - Furthermore, opinions are completely subjective and describe people’s sentiments, appraisals or general feeling towards entities and their properties.The human language can be complex for machine based learning systems to interpret. For example, opinions can be expressed with sarcasm or irony, and the order of words can add even more confusion.
    - It’s an incredibly difficult issue, and sarcasm and other types of ironic language are inherently problematic for machines to detect when looked at in isolation.

## PROPOSED SYSTEM

The bag-of-words model is one of the most widely used feature model for almost all text classification tasks due to its simplicity coupled with good performance. The model represents the text to be classified as a bag or collection of individual words with no link or dependence of one word with the other, i.e. it completely disregards grammar and order of words within the text. This model is also very popular in sentiment analysis and has been used by various researchers.

The simplest way to incorporate this model in our classifier is by using unigrams as features. Generally speaking n-grams is a contiguous sequence of “n” words in our text, which is completely independent of any other words or grams in the text. So unigrams is just a collection of individual words in the text to be classified, and we assume that the probability of occurrence of one word will not be affected by the presence or absence of any other word in the text. This is a very simplifying assumption but it has been shown to provide rather good performance. One simple way to use unigrams as features is to assign them with a certain prior polarity, and take the average of the overall polarity of the text, where the overall polarity of the text could simply be calculated by summing the prior polarities of individual unigrams. Prior polarity of the word would be positive if the word is

generally used as an indication of positivity, for example the word “sweet”; while it would be negative if the word is generally associated with negative connotations, for example “evil”. There can also be degrees of polarity in the model, which means how much indicative is that word for that particular class. A word like “awesome” would probably have strong subjective polarity along with positivity, while the word “decent” would although have positive prior polarity but probably with weak subjectivity. There are three ways of using prior polarity of words as features. The simpler un-supervised approach is to use publicly available online lexicons/dictionaries which map a word to its prior polarity. The Multi- Perspective-Question-Answering (MPQA) is an online resource with such a subjectivity lexicon which maps a total of 4,850 words according to whether they are “positive” or “negative” and whether they have “strong” or “weak” subjectivity

. The SentiWordNet 3.0 is another such resource which gives probability of each word belonging to positive, negative and neutral classes .

The second approach is to construct a custom prior polarity dictionary from our training data according to the occurrence of each word in each particular class. For example if a certain word is occurring more often in the positive labelled phrases in our training dataset (as compared to other classes) then we can calculate the probability of that word belonging to positive class to be higher than the probability of occurring in any other class. However, the latter is a supervised approach because the training data has to be labeled in the appropriate classes before it is possible to calculate the relative occurrence of a word in each of the class. Kouloumpis et al. noted a decrease in performance by using the lexicon word features along with custom n-gram word features constructed from the training data, as opposed to when the n-grams were used alone .

Many of the researchers in this field have used already constructed publicly available lexicons of sentiment bearing words while many others have also explored building their own prior polarity lexicons. The basic problem with the approach of prior polarity approach has been identified by Wilson et al. who distinguish between prior polarity and contextual polarity. They say that the prior polarity of a word may in fact be different from the way the word has been used in the particular context.

The paper presented the following phrase as an example: Philip Clapp, president of the National Environment Trust, sums up well the general thrust of the reaction of environmental movements: “There is no reason at all to believe that the polluters are suddenly going to become reasonable.” In this example all of the four underlined words “trust”, “well”, “reason” and “reasonable” have positive polarities when observed without context to the phrase, but here they are not being used to express a positive sentiment. This concludes that even though generally speaking a word like “trust” may be used in positive sentences, but this doesn’t rule out the chances of it appearing in non-positive sentences as well.

Henceforth prior polarities of individual words (whether the words generally carry positive or negative connotations) may alone not enough for the problem. The paper explores some other features which include grammar and syntactic relationships between words to make their classifier better at judging the contextual polarity of the phrase. The task of twitter sentiment analysis can be most closely related to phrase level sentiment analysis. A seminal paper on phrase level sentiment analysis was presented in 2005 by Wilson et al. which identified a new approach to the problem by first classifying phrases according to subjectivity (polar) and objectivity (neutral) and then further classifying the subjective-

classified phrases as either positive or negative. The paper noticed that many of

the objective phrases used prior sentiment bearing words in them, which led to poor classification of especially objective phrases. It claims that if we use a simple classifier which assumes that the contextual polarity of the word is merely equal to its prior polarity gives a result of about 48%. The novel classification process proposed by this paper along with the list of ingenious features which include information about contextual polarity resulted in significant improvement in performance (in terms of accuracy) of the classification process.

## Advantages

* Sentiment analysis has many applications and benefits to a business and organization. It can be used to give your business valuable insights into how people feel about your product brand or service.
* When applied to social media channels, it can be used to identify spikes in sentiment, thereby allowing you to identify potential product advocates or social media influencers.
* It can be used to identify when potential negative threads are emerging online regarding your business, thereby allowing you to be proactive in dealing with it more quickly.
* Sentiment analysis could also be applied to your corporate network, for example, by applying it to your email server, emails could be monitored for their general “tone”.

## FUNCTIONAL REQUIREMENT

|  |  |
| --- | --- |
| Functional Requirement | Priority (Moscow) |
| The application must connect to twitter API to get the most recent tweet about a specific topic. | Must |
| The application should contain an interface to add or remove a watch and select number of related words. | Should |
| The application data could be persistent. | Could |
| The application must display the related words for a particular watch. | Must |

**Table 3.2 Functional Requirements**

* 1. **DATA COLLECTION**

The data collection of Twitter was abstracted for the rest of the application by creating the class Twitter Client that has method to authenticate and to stream twitter messages in really time. When this client is created it receives a reference to the analyser class that is used to retrieve the current watched words. The initial step is to call the authenticate method from the Twitter Client in order to retrieve the access token required to perform further requests to the Twitter API.

## AUTHENTHICATION

In order to authenticate, first URL encode the API secret. Both API Key and API secret are given to us by registering the application on the Twitter developer website. The result as it follow:

API Key: '1092682800078061568-ZJ00WOfFrxTh7abjiSytW2qe8vE6dy'

API secret: 'VDmXoHWlsh4m91zoZ7vDHJNTN2CiOurolMtTvXFYLvojr'

## TWITTER API

In order to retrieve data method stream is called it retrieves the latest public message for the watched word once.

Assuming the authentication has been performed, it get an update list of the current watched words. If the list is empty then there is nothing to do. Otherwise, create a query string by concatenating each word separated by the string +OR+. One example for the words: football; politics and love. Furthermore the constructor is there as string: football+OR+politics+OR+love.

There are limits imposed by Twitter on how many times user can call the API. This means that we cannot call constantly the API to retrieve the latest message and instead require to wait between each request.

# CHAPTER 4 SYSTEM REQUIREMENT

## INTRODUCTION

The system requirement is a technical specification of requirements for the software products. It is the first step in the requirements analysis process it lists the requirements of a particular software system including functional, performance and security requirements. The requirements also provide usage scenarios from a user, an operational and an administrative perspective. The purpose of software requirements specification is to provide a detailed overview of the software project, its parameters and goals. This describes the project target audience and its user interface, hardware and software requirements. It defines how the client, team and audience see the project and its functionality.

## HARDWARE REQUIREMENTS

* System : Pentium IV 2.4 GHz.
* Hard Disk : 40 GB.
* Removable Disk : 8 GB
* Ram : 256 Mb.

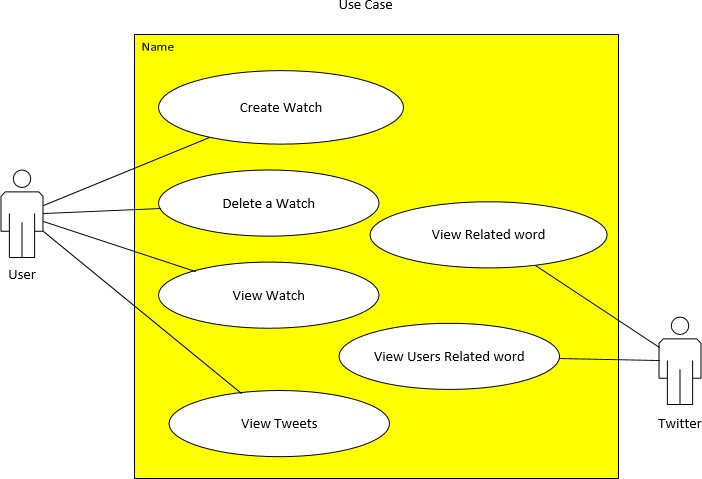
## SOFTWARE REQUIREMENTS

* Operating System : Windows 7
* Front-End Tool : Python
* Back-End Tool : MYSQL-Server
* IDE : Pycharm

# CHAPTER 5 SYSTEM DESIGN

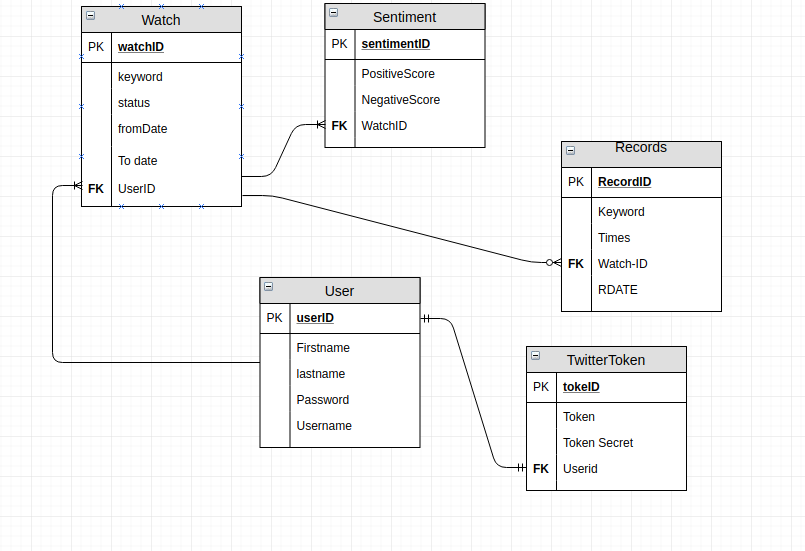
This section of the report will discuss the design proposed for this application. The proposed application contains three basic components. A data collection which will be responsible to retrieve data from Twitter really time messages related to the topic that the user intend to analyse and then there are two analyses that will be performed for each message. The first is the sentiment analysis which tried to determine if the message is positive or negative. The second is the analysis of the related words for a topic in order to understand how they evolve over time.

## USE CASE DIAGRAM



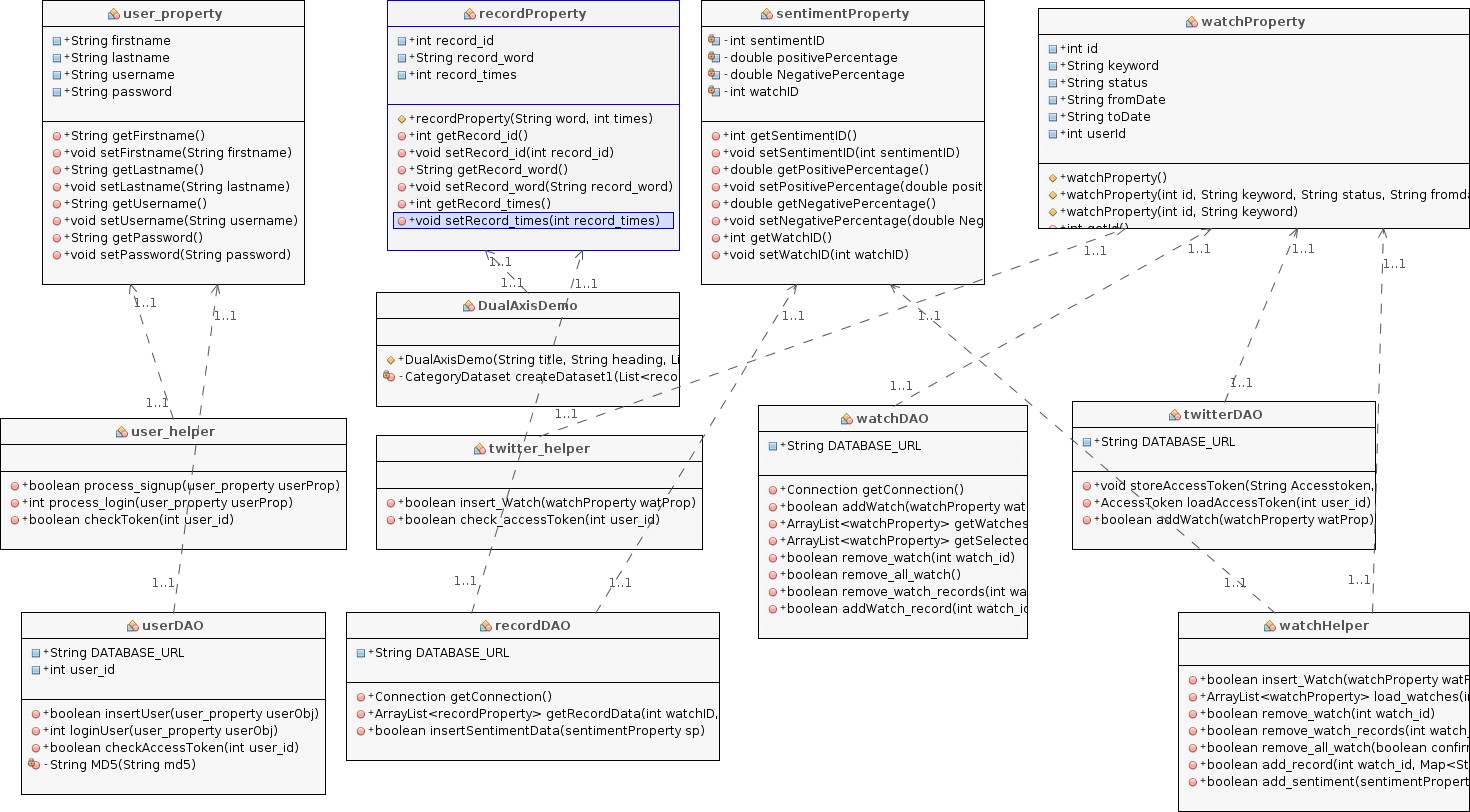
**FIG 5.1. USE CASE DIAGRAM FOR THE SYSTEM**

## E-R DIAGRAM



**FIG 5.2 E-R DIAGRAM FOR THE SYSTEM**

## CLASS DIAGRAM



**FIG 5.3 CLASS DIAGRAM**

* 1. **DATATYPES**

|  |  |  |
| --- | --- | --- |
| S.no | Field name | Type |
| 1 | UserID | int |
| 2 | Username | varchar(250) |
| 3 | First Name | varchar(250) |
| 4 | Last Name | varchar(250) |
| 5 | Password (Hashed MD5) | varchar(250) |

**USER DATA**

**Table 5.4.1**

**SENTIMENT**

|  |  |  |
| --- | --- | --- |
| S.no | Field name | Type |
| 1 | Sentiment ID | int |
| 2 | Positive Score | double |
| 3 | Negative Score | double |
| 4 | WatchId | int |

**Table 5.4.2**

**WATCH**

|  |  |  |
| --- | --- | --- |
| S.no | Field name | Type |
| 1 | Watch ID | Int |
| 2 | Keyword | Varchar |
| 3 | Status | Varchar |
| 4 | From date | Date |
| 5 | To date | Date |
| 6 | UserID | Int |

**Table 5.4.3**

**RECORDS**

|  |  |  |
| --- | --- | --- |
| S.no | Field name | Type |
| 1 | Record ID | Int |
| 2 | Keyword | Varchar |
| 3 | Times | Int |
| 4 | Record date | Date |
| 6 | WatchID | Int |

**Table 5.4.4**

**TWITTER TOKEN**

|  |  |  |
| --- | --- | --- |
| S.no | Field name | Type |
| 1 | Token ID | int |
| 2 | Token | varchar |
| 3 | Token Secret | varchar |
| 4 | UserID | int |

**Table5.4.5**

**CHAPTER 6 SYSTEM STUDY**

## FEASIBILITY STUDY

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

* + - ECONOMICAL FEASIBILITY
    - TECHNICAL FEASIBILITY
    - SOCIAL FEASIBILITY

## Economic Feasibility

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into their search and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

## Technical Feasibility

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement; as only minimal or null changes are required for implementing this system.

## Social Feasibility

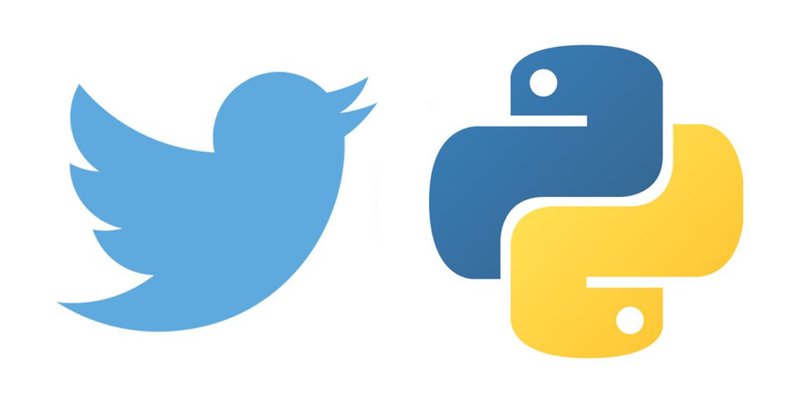
The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

# CHAPTER 7 IMPLEMENTATION

## 7.1 MODULES

**7.1.1 Tweepy Framework:**

The application performs real-time sentiment analysis on Twitter on tweets that matched particular keywords provided by the user. For example if a user is interested in performing sentiment analysis on tweets which contain the word “Obama” he / she will enter that keyword and the web application will perform the appropriate sentiment analysis and display the results for the user.



**FIG 7.1.1 TWEEPY LOGO**

The web application has been implemented using the Local host service because it can be used as a free web hosting service and it provides a layer of abstraction to the developer from the low level web operations so it is easier to learn. We implemented our algorithm in python and integrated it with GUI for our website using HTML and Javascript using the jinja2 template. We used the Google Visualization Chart API for presenting our results in a graphical, easy-to understand manner.

For acquiring tweets from Twitter we used the TWEEPY API in this case. Tweepy API provides access to tweets up to around a week in past according to the search query we specify. If we use the Twitter Streaming API and the user specifies a keyword which is not very common in Twitter, the web application may have to wait for a long time to acquire enough tweets to display reasonable results and also Twitter has stopped using the Basic authentication and switched to the OAuthentication using secret keys . In contrast to this it is much simpler to acquire the tweets in a couple of simple URL calls to the Tweepy API. One limitation of The Tweepy API however is that one call can only give us a maximum of 100 results. Since we apply sentiment analysis on the past 1,000 tweets on any search query (given that there are that many tweets matching with the keyword available), so we have to basically call the API 10 times to get the required number of tweets. This is the basic source of processing delay in our web application.

## Feature Extraction

Now that we have arrived at our training set we need to extract useful features from it which can be used in the process of classification. But first we will discuss some text formatting techniques which will aid us in feature extraction:

Tokenization: It is the process of breaking a stream of text up into words, symbols and other meaningful elements called “tokens”. Tokens can be separated by whitespace characters and/or punctuation characters. It is done so that we can look at tokens as individual components that make up a tweet.

Urls and user references (identified by tokens “http” and “@”) are removed if we are interested in only analyzing the text of the tweet.

Punctuation marks and digits/numerals may be removed if for example we wish to compare the tweet to a list of English words.

Lowercase Conversion: Tweet may be normalized by converting it to lowercase which makes its comparison with an English dictionary easier.

Stemming: It is the text normalizing process of reducing a derived word to its root or stem. For example a stemmer would reduce the phrases “stemmer”, “stemmed”, “stemming” to the root word “stem”. Advantage of stemming is that it

makes comparison between words simpler, as we do not need to deal with complex

grammatical transformations of the word. In our case we employed the algorithm of “porter

stemming” on both the tweets and the dictionary, whenever there was a need of comparison.

Stop-words removal: Stop words are class of some extremely common words which hold no additional information when used in a text and are thus claimed to be useless. Examples include “a”, “an”, “the”, “he”, “she”, “by”, “on”, etc. It is sometimes convenient to remove these words because they hold no additional information since they are used almost equally in all classes of text.

Now that we have discussed some of the text formatting techniques employed by us, we will move to the list of features that we have explored. As we will see below a feature is any variable which can help our classifier in differentiating between the different classes. There are two kinds of classification in our system (as will be discussed in detail in the next section), the objectivity / subjectivity classification and the positivity / negativity classification. As the name suggests the former is for differentiating between objective and subjective classes while the latter is for differentiating between positive and negative classes. The list of features explored for objective / subjective classification is as below:

* + - * + Number of exclamation marks in a tweet
        + Number of question marks in a tweet
        + Presence of exclamation marks in a tweet
        + Presence of question marks in a tweet
        + Presence of url in a tweet
        + Presence of emoticons in a tweet
        + Unigram word models calculated using Naive Bayes
        + Prior polarity of words through online lexicon MPQA
        + Number of digits in a tweet • Number of capitalized words in a tweet
        + Number of capitalized characters in a tweet
        + Number of punctuation marks / symbols in a tweet

The final issue we have in feature selection is choosing the best features from a large number of features. Our ultimate aim is to achieve the greatest accuracy of our classifier while using least number of features. This is because adding new feature add to the dimensionality of our classification problem and thus add to the complexity of our classifier. This increase in complexity may not necessarily be linear and may even be quadratic so it is preferred to keep the features at a minimum low. Another issue we have with too many features is that our training data may be over-fit and it may confuse the classifier when doing classification on an unknown test set, so the accuracy of the classifier may even decrease.

## Classification

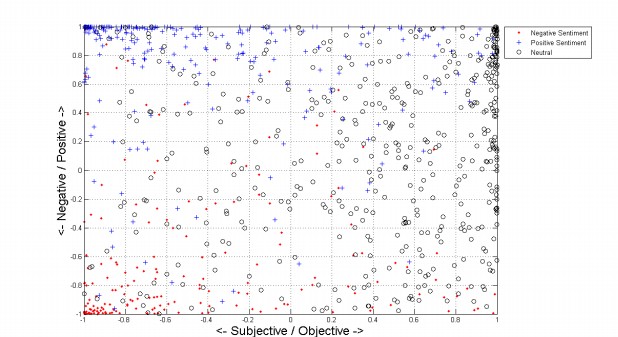
Pattern classification is the process through which data is divided into different classes according to some common patterns which are found in one class which differ to some degree with the patterns found in the other classes. The ultimate aim of our project is to design a classifier which accurately classifies tweets in the following four sentiment classes: positive, negative, neutral and ambiguous. There can be two kinds of sentiment classifications in this area: contextual sentiment analysis and general sentiment analysis. Contextual sentiment analysis deals with classifying specific parts of a tweet according to the context provided, for example for the tweet “4 more years of being in shithole Australia then I move to the USA :D” a contextual sentiment classifier would identify Australia with negative sentiment and USA with a positive sentiment. On the other hand general sentiment analysis deals with the general sentiment of the entire text

(tweet in this case) as a whole. Thus for the tweet mentioned earlier since there is an overall positive attitude, an accurate general sentiment classifier would identify it as positive. For our particular project we will only be dealing with the latter case,

i.e. of general (overall) sentiment analysis of the tweet as a whole. The classification approach generally followed in this domain is a two-step approach. First Objectivity Classification is done which deals with classifying a tweet or a phrase as either objective or subjective. After this we perform Polarity Classification (only on tweets classified as subjective by the objectivity classification) to determine whether the tweet is positive, negative or both (some researchers include the both category and some don’t). This was presented by Wilson et al. and reports enhanced accuracy than a simple one-step approach .

We propose a novel approach which is slightly different from the approach proposed by Wilson et al. . We propose that in first step each tweet should undergo two classifiers: the objectivity classifier and the polarity classifier. The former would try to classify a tweet between objective and subjective classes, while latter would do so between the positive and negative classes. We use the short-listed features for these classifications and use the Naive Bayes algorithm so that after the first step we have two numbers from 0 to 1 representing each tweet. One of these numbers is the probability of tweet belonging to objective class and the other number is probability of tweet belonging to positive class. Since we can easily calculate the two remaining probabilities of subjective and negative by simple subtraction by 1.

## 7.1.1.3 PATTERN ANALYSIS



**FIG 7.1.1.3**

* 1. **Programming**

The objective of the coding phase is to transform the design of a system into code in a high level language and then to unit test this code. The programmers adhere to standard and well defined style of coding which they call their coding standard. The main advantages of adhering to a standard style of coding are as follows:

* + - A coding standard gives uniform appearances to the code written by different engineers
    - It facilitates code of understanding
    - Promotes good programming practices

For implementing our design into a code, we require a good high level language.

A programming language should have the following features:

* + - Readability
    - Portability
    - Brevity
    - Error checking
    - Familiar notation
    - Quick translation
    - Efficiency

## Programming STANDARDS

Good software development organizations usually develop their own coding standards and guidelines depending on what best suits their organization and the type of products they develop.

The following are some representative coding standards.

1. **Rules for limiting the use of global**: These rules list what types of data can be declared global and what cannot.
2. **Contents of the headers preceding codes for different modules**: The information contained in the headers of different modules should be standard for an organization. The exact format in which the header information is organized in the header can also be specified. The following are some standard header data:
   * Name of the module.
   * Date on which the module was created.
   * Author’s name.
   * Modification history.
   * Synopsis of the module.
   * Different functions supported, along with their input/output parameters.
   * Global variables accessed/modified by the module.
3. **Naming conventions for global variables, local variables, and constant identifiers**: A possible naming convention can be that global variable names always start with a capital letter, local variable names are made of small letters, and constant names are always capital letters.
4. **Error return conventions and exception handling mechanisms**: The way error conditions are reported by different functions in a program are handled should be standard within an organization.

## NAMING CONVENTIONS

## Rules Common to All Identifiers

Identifiers use only ASCII letters and digits, and, in a small number of cases noted below, underscores and very rarely (when required by frameworks like Angular) dollar signs.

Give as descriptive a name as possible, within reason. Do not worry about saving horizontal space as it is far more important to make your code immediately understandable by a new reader. Do not use abbreviations that are ambiguous or unfamiliar to readers outside your project, and do not abbreviate by deleting letters within a word.

## Rules By Identifier Type

* + - 1. **Package Names**

Package names are all lowerCamelCase. For example, my.exampleCode.Hiding, but not my.examplecode.Hiding or my.example\_code.Hiding.

## Class Names

Class, interface, record, and typedef names are written in UpperCamelCase. Unexported classes are simply locals: they are not marked @private and therefore are not named with a trailing underscore.

Type names are typically nouns or noun phrases. For example, Request, ImmutableList, or VisibilityMode. Additionally, interface names may sometimes be adjectives or adjective phrases instead (for example, Readable).

## Method Names

Method names are written in lowerCamelCase. Private methods’ names must end with a trailing underscore.

Method names are typically verbs or verb phrases. For example, sendMessage or stop\_. Getter and setter methods for properties are never required, but if they are used they should be named getFoo (or optionally isFoo or hasFoo for booleans), or setFoo (value) for setters.

Underscores may also appear in JsUnit test method names to separate logical components of the name. One typical pattern is test<MethodUnderTest>\_<state>, for example testPop\_emptyStack. There is no One Correct Way to name test methods.

## Constant Names

Constant names use CONSTANT\_CASE: all uppercase letters, with words separated by underscores. There is no reason for a constant to be named with a trailing underscore, since private static properties can be replaced by (implicitly private) module locals.

## Parameter Names

Parameter names are written in lowerCamelCase. Note that this applies even if the parameter expects a constructor.

One-character parameter names should not be used in public methods.

Exception: When required by a third-party framework, parameter names may begin with a $. This exception does not apply to any other identifiers (e.g. local variables or properties).

7.4.3 **Camel Case Defined**

Sometimes there is more than one reasonable way to convert an English phrase into camel case, such as when acronyms or unusual constructs like IPv6 or iOS are present. To improve predictability, Google Style specifies the following (nearly) deterministic scheme.

Beginning with the prose form of the name:

1. Convert the phrase to plain ASCII and remove any apostrophes. For example, EINSTEIN’S algorithm might become EINSTEINS algorithm.
2. Divide this result into words, splitting on spaces and any remaining punctuation (typically hyphens).
   1. Recommended: if any word already has a conventional camel case appearance in common usage, split this into its constituent parts (e.g., AdWords becomes ad words). Note that a word such as iOS is not really in camel case per se; it defies any convention, so this recommendation does not apply.
3. Now lowercase everything (including acronyms), then uppercase only the first character of:
   1. … each word, to yield upper camel case, or… each word except the first, to yield lower camel case
4. Finally, join all the words into a single identifier.

Note that the casing of the original words is almost entirely disregarded.

# CHAPTER 8 SYSTEM TESTING

Testing is the process of evaluating a system or its component(s)

With the intent to find whether it satisfies the specified requirements or not. In simple words, testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

According to ANSI/IEEE 1059 standard, Testing can be defined as - A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item.

An early start to testing reduces the cost and time to rework and produce error-free software that is delivered to the client. However, in Software Development Life Cycle (SDLC), testing can be started from the Requirements Gathering phase and continued till the deployment of the software. It also depends on the development model that is being used. For example, in the Waterfall model, formal testing is conducted in the testing phase; but in the incremental model, testing is performed at the end of every increment/iteration and the whole application is tested at the end.

Testing is done in different forms at every phase of SDLC:

* + - * + During the requirement gathering phase, the analysis and verification of requirements are also considered as testing.
        + Reviewing the design in the design phase with the intent to improve the design is also considered as testing.
        + Testing performed by a developer on completion of the code.

It is difficult to determine when to stop testing, as testing is a never-ending process and no one can claim that a software is 100% tested. The following aspects are to be considered for stopping the testing process:

* + - * + Testing Deadlines
        + Completion of test case execution
        + Completion of functional and code coverage to a certain point
        + Bug rate falls below a certain level and no high-priority bugs are identified
        + Management decision

## TESTING

* + 1. **UNIT TESTING**

This type of testing is performed by developers before the setup is handed over to the testing team to formally execute the test cases. Unit testing is performed by the respective developers on the individual units of source code assigned areas. The developers use test data that is different from the test data of the quality assurance team.

The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality.

## FUNCTIONAL TESTING

This is a type of black-box testing that is based on the specifications of the software that is to be tested. The application is tested by providing input and then the results are examined that need to conform to the functionality it was intended for. Functional testing of a software is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.

There are five steps that are involved while testing an application for functionality.

* + - * The determination of the functionality that the intended application is meant to perform.
      * The creation of test data based on the specifications of the application.
      * The output based on the test data and the specifications of the application.
      * The writing of test scenarios and the execution of test cases.
      * The comparison of actual and expected results based on the executed test cases.

## PERFORMANCE TESTING

It is mostly used to identify any bottlenecks or performance issues rather than finding bugs in a software. There are different causes that contribute in lowering the performance of a software:

* Network delay
* Client-side processing
* Database transaction processing
* Load balancing between servers
* Data rendering

Performance testing is considered as one of the important and mandatory testing type in terms of the following aspects:

* Speed (i.e. Response Time, data rendering and accessing)
* Capacity
* Stability
* Scalability

Performance testing can be either qualitative or quantitative and can be divided into different subtypes such as Load testing and Stress testing.

## STRESS TESTING

Stress testing includes testing the behavior of a software under abnormal conditions. For example, it may include taking away some resources or applying a load beyond the actual load limit.

The aim of stress testing is to test the software by applying the load to the system and taking over the resources used by the software to identify the breaking point. This testing can be performed by testing different scenarios such as:

* Shutdown or restart of network ports randomly
* Turning the database on or off
* Running different processes that consume resources such as CPU, memory, server, etc.

## INTEGRATION TESTING

**‘**

Integration testing is defined as the testing of combined parts of an application to determine if they function correctly. Integration testing can be done in two ways: Bottom-up integration testing and Top-down integration testing.

## ALPHA TESTING

This test is the first stage of testing and will be performed amongst the teams (developer and QA teams). Unit testing, integration testing and system testing when combined together is known as alpha testing. During this phase, the following aspects will be tested in the application:

* + - * Spelling Mistakes
      * Broken Links
      * Cloudy Directions
      * The Application will be tested on machines with the lowest specification to test loading times and any latency problems.

## TESTING TECHNIQUES / TESTING STRATEGIES

* + 1. **TESTING**

Testing depends on the process and the associated stakeholders of the project(s). In the IT industry, large companies have a team with responsibilities to evaluate the developed software in context of the given requirements. Moreover, developers also conduct testing which is called Unit Testing. In most cases, the following professionals are involved in testing a system within their respective capacities:

* Software Tester
* Software Developer
* Project Lead/Manager
* End User

Different companies have different designations for people who test the software on the basis of their experience and knowledge such as Software Tester, Software Quality Assurance Engineer, QA Analyst, etc.

It is not possible to test the software at any time during its cycle. The next two sections state when testing should be started and when to end it during the

SDLC.

Testing is the process of executing the program with the intent of finding the error. A good test case design is one that as probability of finding a yet undiscovered error. A successful test is one that uncovers a yet undiscovered error. Any engineering product can be tested in one of the two ways:

## WHITE BOX TESTING

White-box testing is the detailed investigation of internal logic and structure of the code. White-box testing is also called glass testing or open-box testing. In order to perform white-box testing on an application, a tester needs to know the internal workings of the code.

The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately. Basic path testing is a white box testing.

Basic path testing:

* + - * + Flow graph notation
        + Cyclometric complexity
        + Deriving test cases
        + Graph matrices control

## BLACK BOX TESTING

The technique of testing without having any knowledge of the interior workings of the application is called black-box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, while performing a black-box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

The steps involved in black box test case design are:

* + - * + Graph based testing methods
        + Equivalence partitioning
        + Boundary value analysis
        + Comparison testing

## SOFTWARE TESTING STRATEGIES

Software Testing is evaluation of the software against requirements gathered from users and system specifications. Testing is conducted at the phase level in software development life cycle or at module level in program code. Software testing comprises of Validation and Verification.

Target of the test are -

* + - * + **Errors** - These are actual coding mistakes made by developers. In addition, there is a difference in output of software and desired output, is considered as an error.
        + **Fault** - When error exists fault occurs. A fault, also known as a bug, is a result of an error which can cause system to fail.

**Failure** - failure is said to be the inability of the system to perform the desired task. Failure occurs when fault exists in the system.

A test needs to check if a webpage can be opened in Internet Explorer. This can be easily done with manual testing. But to check if the web-server can take the load of 1 million users, it is quite impossible to test manually.

There are software and hardware tools which helps tester in conducting load testing, stress testing, regression testing.

## INTEGRATION TESTING

Integration testing is a level of software testing where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assist in Integration Testing.

In a comprehensive software development environment, bottom-up testing is usually done first, followed by top-down testing. The process concludes with multiple tests of the complete application, preferably in scenarios designed to mimic actual situations.

## ACCEPTANCE TESTING

An acceptance test is performed by the client and verifies whether the end to end the flow of the system is as per the business requirements or not and if it is as per the needs of the end user. Client accepts the software only when all the features and functionalities work as expected. It is the last phase of the testing, after which the software goes into production. This is also called as User Acceptance Testing (UAT).

## BACK END TESTING

Whenever an input or data is entered on front-end application, it stores in the database and the testing of such database is known as Database Testing or Backend testing. There are different databases like SQL Server, MySQL, and Oracle etc. Database testing involves testing of table structure, schema, stored procedure, data structure and so on.

In back-end testing GUI is not involved, testers are directly connected to the database with proper access and testers can easily verify data by running few queries on the database. There can be issues identified like data loss, deadlock, data corruption etc. during this back-end testing and these issues are critical to fixing before system goes live into the production environment.

## BROWSER COMPATIBILITY TESTING

It is a subtype of Compatibility Testing (which is explained below) and is performed by the testing team. Browser Compatibility Testing is performed for web applications and it ensures that the software can run with the combination of different browser and operating system. This type of testing also validates whether web application runs on all versions of all browsers or not.

## GRAPHICAL USER INTERFACE (GUI) TESTING

The objective of this GUI testing is to validate the GUI as per the business requirement. The expected GUI of the application is mentioned in Detailed Design Document and GUI mockup screens. The GUI testing includes size of the buttons and input field present on the screen, alignment of all text, tables and content in the tables.

It also validates the menu of the application, after selecting different menu and menu items, it validates that the page does not fluctuate and the alignment remains same after hovering the mouse on the menu or sub-menu.

## SECURITY TESTING

It is a type of testing performed by a special team of testers. A system can be penetrated by any hacking way. Security Testing is done to check how the software or application or website is secure from internal and external threats. This testing includes how much software is secure from the malicious program, viruses and how secure and strong the authorization and authentication processes are.

It also checks how software behaves for any hackers attack and malicious programs and how software is maintained for data security after such hacker attack.

## ACCESSIBILITY TESTING

The aim of accessibility testing is to determine whether the software or application is accessible for disabled people or not. Here disability means deaf, color blind, mentally disabled, blind, old age and other disabled groups. Various checks are performed such as font size for visually disabled, color and contrast for color blindness etc.

**CHAPTER 9**

**CONCLUSION**

The task of sentiment analysis, especially in the domain of micro-blogging, is still in the developing stage and far from complete. So we propose a couple of ideas which we feel are worth exploring in the future and may result in further improved performance. Right now we have worked with only the very simplest unigram models; we can improve those models by adding extra information like closeness of the word with a negation word. We could specify a window prior to the word (a window could for example be of 2 or 3 words) under consideration and the effect of negation may be incorporated into the model if it lies within that window. The closer the negation word is to the unigram word whose prior polarity is to be calculated, the more it should affect the polarity. For example if the negation is right next to the word, it may simply reverse the polarity of that word and farther the negation is from the word the more minimized ifs effect should be. Apart from this, we are currently only focusing on unigrams and the effect of bigrams and trigrams may be explored. As reported in the literature review section when bigrams are used along with unigrams this usually enhances performance.

# CHAPTER 10 FUTURE ENHANCEMENT

Right now we are exploring Parts of Speech separate from the unigram models, we can try to incorporate POS information within our unigram models in future. So say instead of calculating a single probability for each word like P(word

| obj) we could instead have multiple probabilities for each according to the Part of Speech the word belongs to. For example we may have P(word | obj, verb), P(word

| obj, noun) and P(word | obj, adjective). Pang et al. used a somewhat similar approach and claims that appending POS information for every unigram results in no significant change in performance (with Naive Bayes performing slightly better and SVM having a slight decrease in performance), while there is a significant decrease in accuracy if only adjective unigrams are used as features. However these results are for classification of reviews and may be verified for sentiment analysis on micro blogging websites like Twitter.

One potential problem with our research is that the sizes of the three classes are not equal. The objective class which contains 4,543 tweets is about twice the sizes of positive and negative classes which contain 2,543 and 1,877 tweets respectively. The problem with unequal classes is that the classifier tries to increase the overall accuracy of the system by increasing the accuracy of the majority class, even if that comes at the cost of decrease in accuracy of the minority classes. That is the very reason why we report significantly higher accuracies for objective class as opposed to positive or negative classes. To overcome this problem and have the classifier exhibit no bias towards any of the classes, it is necessary to label more data (tweets) so that all three of our classes are almost equal.

Last but not the least, we can attempt to model human confidence in our system. For example if we have 5 human labellers labelling each tweet, we can plot the tweet in the 2-dimensional objectivity / subjectivity and positivity / negativity plane while differentiating between tweets in which all 5 labels agree, only 4 agree, only 3 agree or no majority vote is reached. We could develop our custom cost function for coming up with optimized class boundaries such that highest weightage is given to those tweets in which all 5 labels agree and as the number of agreements start decreasing, so do the weights assigned. In this way the effects of human confidence can be visualized in sentiment analysis.

**APPENDIX ‘A’ SOURCE CODE**

1. **Code for calculating the sentiment of each tweet.**

**import** sys,tweepy,csv,re  
**from** textblob **import** TextBlob  
**from** bs4 **import** BeautifulSoup  
**import** matplotlib.pyplot **as** plt  
**import** requests  
**import** re  
  
**from** flask **import** Flask, render\_template, request  
app = Flask(\_\_name\_\_)  
  
  
**class** Analysis:  
 **def** \_\_init\_\_(self, term):  
 self.term = term  
 self.sentiment = 0  
 self.subjectivity = 0  
 self.url = **'https://www.google.com/search?q={0}&source=lnms&tbm=nws'**.format(self.term)  
 self.tweets = []  
 self.tweetText = []

**def** run(self):  
 response = requests.get(self.url)  
 **print**(response.text)  
 soup = BeautifulSoup(response.text, **'html.parser'**)  
 headline\_results = soup.find\_all(**'div'**, class\_=**'st'**)  
  
 polarity = 0  
 positive = 0  
 wpositive = 0  
 spositive = 0  
 negative = 0  
 wnegative = 0  
 snegative = 0  
 neutral = 0  
  
 **for** tweet **in** headline\_results:  
 *# Append to temp so that we can store in csv later. I use encode UTF-8* self.tweetText.append(self.cleanTweet(str(tweet.text)).encode(**'utf-8'**))  
 *# print (tweet.text.translate(non\_bmp\_map)) #print tweet's text* analysis = TextBlob(tweet.text)  
 *# print(analysis.sentiment) # print tweet's polarity* polarity += analysis.sentiment.polarity *# adding up polarities to find the average later*

**if** (  
 analysis.sentiment.polarity == 0): *# adding reaction of how people are reacting to find average later* neutral += 1  
 **elif** (analysis.sentiment.polarity > 0 **and** analysis.sentiment.polarity <= 0.3):  
 wpositive += 1  
 **elif** (analysis.sentiment.polarity > 0.3 **and** analysis.sentiment.polarity <= 0.6):  
 positive += 1  
 **elif** (analysis.sentiment.polarity > 0.6 **and** analysis.sentiment.polarity <= 1):  
 spositive += 1  
 **elif** (analysis.sentiment.polarity > -0.3 **and** analysis.sentiment.polarity <= 0):  
 wnegative += 1  
 **elif** (analysis.sentiment.polarity > -0.6 **and** analysis.sentiment.polarity <= -0.3):  
 negative += 1  
 **elif** (analysis.sentiment.polarity > -1 **and** analysis.sentiment.polarity <= -0.6):  
 snegative += 1

*# finding average of how people are reacting* positive = self.percentage(positive, len(headline\_results))  
 wpositive = self.percentage(wpositive, len(headline\_results))  
 spositive = self.percentage(spositive, len(headline\_results))  
 negative = self.percentage(negative, len(headline\_results))  
 wnegative = self.percentage(wnegative, len(headline\_results))  
 snegative = self.percentage(snegative, len(headline\_results))  
 neutral = self.percentage(neutral, len(headline\_results))  
  
*# finding average reaction* polarity = polarity / len(headline\_results)  
  
  
 **if** (polarity == 0):  
 **print**(**"Neutral"**)  
 **elif** (polarity > 0 **and** polarity <= 0.3):  
 **print**(**"Weakly Positive"**)  
 **elif** (polarity > 0.3 **and** polarity <= 0.6):  
 **print**(**"Positive"**)  
 **elif** (polarity > 0.6 **and** polarity <= 1):  
 **print**(**"Strongly Positive"**)  
 **elif** (polarity > -0.3 **and** polarity <= 0):  
 **print**(**"Weakly Negative"**)  
 **elif** (polarity > -0.6 **and** polarity <= -0.3):  
 **print**(**"Negative"**)  
 **elif** (polarity > -1 **and** polarity <= -0.6):  
 **print**(**"Strongly Negative"**)  
  
 **print**()  
 **print**(**"Detailed Report: "**)  
 **print**(str(positive) + **"% people thought it was positive"**)  
 **print**(str(wpositive) + **"% people thought it was weakly positive"**)  
 **print**(str(spositive) + **"% people thought it was strongly positive"**)  
 **print**(str(negative) + **"% people thought it was negative"**)  
 **print**(str(wnegative) + **"% people thought it was weakly negative"**)  
 **print**(str(snegative) + **"% people thought it was strongly negative"**)  
 **print**(str(neutral) + **"% people thought it was neutral"**)  
  
 *# self.plotPieChart(positive, wpositive, spositive, negative, wnegative, snegative, neutral, searchTerm, len(headline\_results))* **return** [positive, wpositive, spositive, negative, wnegative, snegative, neutral]  
  
**def** cleanTweet(self, tweet):  
 *# Remove Links, Special Characters etc from tweet* **return ' '**.join(re.sub(**"(@[A-Za-z0-9]+)|([^0-9A-Za-z \t]) | (\w +:\ / \ / \S +)"**, **" "**, tweet).split())  
 *# function to calculate percentage***def** percentage(self, part, whole):  
 temp = 100 \* float(part) / float(whole)  
 **return** format(temp, **'.2f'**)

**class** SentimentAnalysis:  
  
 **def** \_\_init\_\_(self):  
 self.tweets = []  
 self.tweetText = []  
  
 **def** DownloadData(self,searchTerm,NoOfTerms):  
 *# authenticating* consumerKey = **'jkupWshY89FlvOuLX0hKtDrKz'** consumerSecret = **'bDIM0p6lnYipKWLi2gyH1MbUv1XXffZzMXMi9j7BWcy3X0Zlmp'** accessToken = **'1092682800078061568-ZJ00WOfFrxTh7abjiSytW2qe8vE6dy'** accessTokenSecret = **'VDmXoHWlsh4m91zoZ7vDHJNTN2CiOurolMtTvXFYLvojr'** auth = tweepy.OAuthHandler(consumerKey, consumerSecret)  
 auth.set\_access\_token(accessToken, accessTokenSecret)  
 api = tweepy.API(auth)

*# input for term to be searched and how many tweets to search  
#searchTerm = input("Enter Keyword/Tag to search about: ")  
  
#NoOfTerms = int(input("Enter how many tweets to search: "))  
  
  
# searching for tweets*self.tweets = tweepy.Cursor(api.search, q=searchTerm, lang = **"en"**).items(NoOfTerms)  
  
*# Open/create a file to append data to*csvFile = open(**'result.csv'**, **'a'**)  
  
*# Use csv writer*csvWriter = csv.writer(csvFile)  
  
  
*# creating some variables to store info*polarity = 0  
positive = 0  
wpositive = 0  
spositive = 0  
negative = 0  
wnegative = 0  
snegative = 0  
neutral = 0  
  
  
*# iterating through tweets fetched***for** tweet **in** self.tweets:  
 *#Append to temp so that we can store in csv later. I use encode UTF-8* self.tweetText.append(self.cleanTweet(tweet.text).encode(**'utf-8'**))  
 *# print (tweet.text.translate(non\_bmp\_map)) #print tweet's text* analysis = TextBlob(tweet.text)  
 *# print(analysis.sentiment) # print tweet's polarity* polarity += analysis.sentiment.polarity *# adding up polarities to find the average later* **if** (analysis.sentiment.polarity == 0): *# adding reaction of how people are reacting to find average later* neutral += 1  
 **elif** (analysis.sentiment.polarity > 0 **and** analysis.sentiment.polarity <= 0.3):  
 wpositive += 1  
 **elif** (analysis.sentiment.polarity > 0.3 **and** analysis.sentiment.polarity <= 0.6):  
 positive += 1  
 **elif** (analysis.sentiment.polarity > 0.6 **and** analysis.sentiment.polarity <= 1):  
 spositive += 1  
 **elif** (analysis.sentiment.polarity > -0.3 **and** analysis.sentiment.polarity <= 0):  
 wnegative += 1  
 **elif** (analysis.sentiment.polarity > -0.6 **and** analysis.sentiment.polarity <= -0.3):  
 negative += 1  
 **elif** (analysis.sentiment.polarity > -1 **and** analysis.sentiment.polarity <= -0.6):  
 snegative += 1  
  
  
*# Write to csv and close csv file*csvWriter.writerow(self.tweetText)  
csvFile.close()  
  
*# finding average of how people are reacting*positive = self.percentage(positive, NoOfTerms)  
wpositive = self.percentage(wpositive, NoOfTerms)  
spositive = self.percentage(spositive, NoOfTerms)  
negative = self.percentage(negative, NoOfTerms)  
wnegative = self.percentage(wnegative, NoOfTerms)  
snegative = self.percentage(snegative, NoOfTerms)  
neutral = self.percentage(neutral, NoOfTerms)

*# finding average reaction* polarity = polarity / NoOfTerms  
  
 *# printing out data* **print**(**"How people are reacting on "** + searchTerm + **" by analyzing "** + str(NoOfTerms) + **" tweets."**)  
 **print**()  
 **print**(**"General Report: "**)  
  
 **if** (polarity == 0):  
 **print**(**"Neutral"**)  
 **elif** (polarity > 0 **and** polarity <= 0.3):  
 **print**(**"Weakly Positive"**)  
 **elif** (polarity > 0.3 **and** polarity <= 0.6):  
 **print**(**"Positive"**)  
 **elif** (polarity > 0.6 **and** polarity <= 1):  
 **print**(**"Strongly Positive"**)  
 **elif** (polarity > -0.3 **and** polarity <= 0):  
 **print**(**"Weakly Negative"**)  
 **elif** (polarity > -0.6 **and** polarity <= -0.3):  
 **print**(**"Negative"**)  
 **elif** (polarity > -1 **and** polarity <= -0.6):  
 **print**(**"Strongly Negative"**)

**print**()  
 **print**(**"Detailed Report: "**)  
 **print**(str(positive) + **"% people thought it was positive"**)  
 **print**(str(wpositive) + **"% people thought it was weakly positive"**)  
 **print**(str(spositive) + **"% people thought it was strongly positive"**)  
 **print**(str(negative) + **"% people thought it was negative"**)  
 **print**(str(wnegative) + **"% people thought it was weakly negative"**)  
 **print**(str(snegative) + **"% people thought it was strongly negative"**)  
 **print**(str(neutral) + **"% people thought it was neutral"**)  
  
 *#self.plotPieChart(positive, wpositive, spositive, negative, wnegative, snegative, neutral, searchTerm, NoOfTerms)* **return** [positive,wpositive,spositive,negative,wnegative,snegative,neutral]  
  
  
**def** cleanTweet(self, tweet):  
 *# Remove Links, Special Characters etc from tweet* **return ' '**.join(re.sub(**"(@[A-Za-z0-9]+)|([^0-9A-Za-z \t]) | (\w +:\ / \ / \S +)"**, **" "**, tweet).split())

*# function to calculate percentage* **def** percentage(self, part, whole):  
 temp = 100 \* float(part) / float(whole)  
 **return** format(temp, **'.2f'**)  
  
 **def** plotPieChart(self, positive, wpositive, spositive, negative, wnegative, snegative, neutral, searchTerm, noOfSearchTerms):  
 labels = [**'Positive ['** + str(positive) + **'%]'**, **'Weakly Positive ['** + str(wpositive) + **'%]'**,**'Strongly Positive ['** + str(spositive) + **'%]'**, **'Neutral ['** + str(neutral) + **'%]'**,  
 **'Negative ['** + str(negative) + **'%]'**, **'Weakly Negative ['** + str(wnegative) + **'%]'**, **'Strongly Negative ['** + str(snegative) + **'%]'**]  
 sizes = [positive, wpositive, spositive, neutral, negative, wnegative, snegative]  
 colors = [**'yellowgreen'**,**'lightgreen'**,**'darkgreen'**, **'gold'**, **'red'**,**'lightsalmon'**,**'darkred'**]  
 patches, texts = plt.pie(sizes, colors=colors, startangle=90)  
 plt.legend(patches, labels, loc=**"best"**)  
 plt.title(**'How people are reacting on '** + searchTerm + **' by analyzing '** + str(noOfSearchTerms) + **' Tweets.'**)  
 plt.axis(**'equal'**)  
 plt.tight\_layout()  
 plt.show()  
  
  
  
*#if \_\_name\_\_== "\_\_main\_\_":  
 #sa = SentimentAnalysis()  
 #sa.DownloadData()*@app.route(**'/index'**)  
**def** hello\_world():  
 **return**(**"Hello World"**)  
  
@app.route(**'/getForm'**)  
**def** getForm():  
 **return** render\_template(**"TweetFormTemp.html"**)  
  
@app.route(**'/getTweets'**, methods=[**'POST'**])  
**def** getTweets():  
 **if** request.method ==**'POST'**:  
 subject = request.form[**'subject'**]  
 tweets = request.form[**'tweets'**]  
 sa = SentimentAnalysis()  
 arr = sa.DownloadData(subject,int(tweets))  
 a = Analysis(**'samsung S10'**)  
 b = a.run()  
 **print**(b)  
 **print**(**"sdads"**)  
 **return** render\_template(**"result.html"**, result = arr,res = b)  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 app.run()

**HTML FOR THE USER INTERFACE**

<html>

<head>

<title>Give Data</title>

</head>

<meta charset="UTF-8">

<title>Analyser</title>

<link rel='stylesheet' href='http://codepen.io/assets/libs/fullpage/jquery-ui.css'>

<link rel="stylesheet" href="{{url\_for('static',filename='styles/style.css')}}" media="screen" type="text/css" />

</head>

<body>

<body style="background-image:url({{url\_for('static',filename='bgimage.jpg')}})">

<div class="login-card">

<h1>Analyser Input</h1><br>

<form method="post" action="http://localhost:5000/getTweets">

<input type="text" name="subject" placeholder="INPUT TERM / HASHTAG">

<input type="text" name="tweets" placeholder="Number of Search">

<input type="submit" name="login" class="login login-submit" value="Search the internet">

</form>

<!-- <div id="error"><img src="https://dl.dropboxusercontent.com/u/23299152/Delete-icon.png" /> Your caps-lock is on.</div> -->

<script src='http://codepen.io/assets/libs/fullpage/jquery\_and\_jqueryui.js'></script>

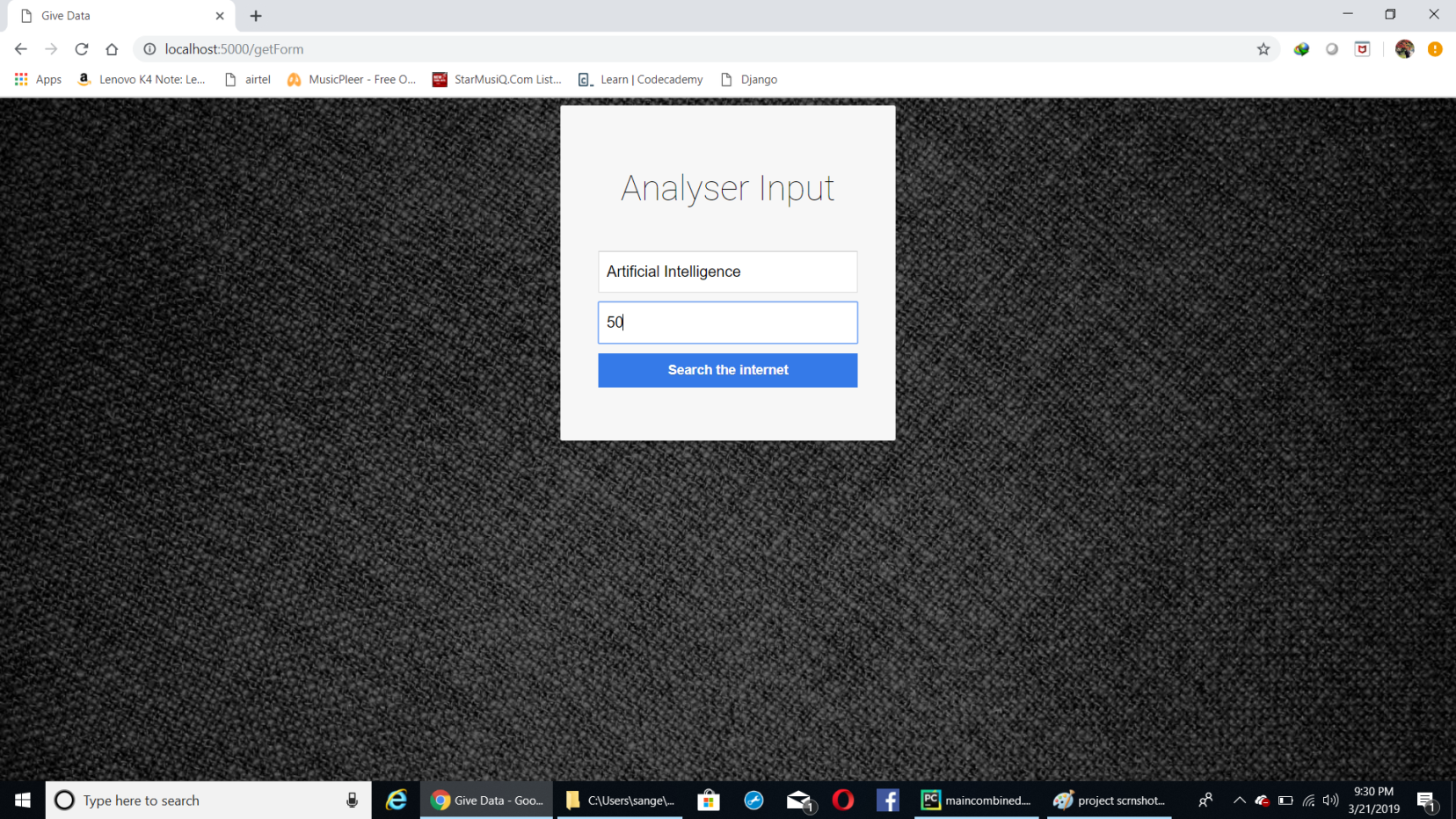
</body>

</html>

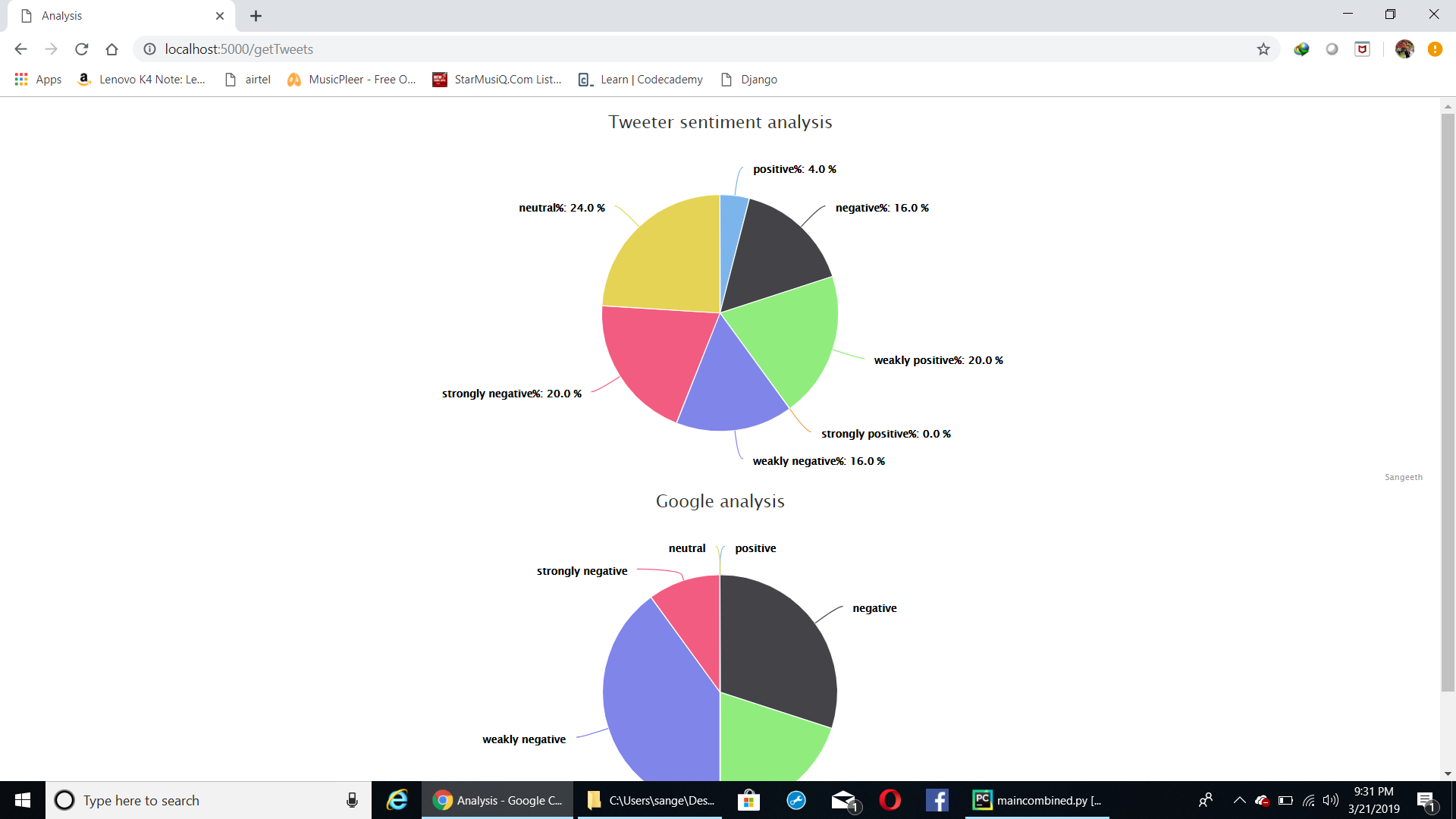
# APPENDIX B SCREENSHOTS

# C:\Users\sange\Desktop\proj scrnshot.jpg

## Fig B.1 Interactive form which lets the user enter the keyword to search the system

****

**Fig B.2 Form where the user the user can enter the search term and the number of searches**

****

**Fig B.3 Graphical representation of the keywords trending on a particular watch on a particular time.**

**REFERENCES**

* 1. Albert Biffet and Eibe Frank. Sentiment Knowledge Discovery in Twitter Streaming Data. Discovery Science, Lecture Notes in Computer Science, 2010,

Volume 6332/2010, 1-15, DOI: 10.1007/978-3-642-16184-1\_1

* 1. Alec Go, Richa Bhayani and Lei Huang. Twitter Sentiment Classification using Distant Supervision. Project Technical Report, Stanford University, 2009.
  2. Alexander Pak and Patrick Paroubek. Twitter as a Corpus for Sentiment Analysis and Opinion Mining. In Proceedings of international conference on Language Resources and Evaluation (LREC), 2010.
  3. Andranik Tumasjan, Timm O. Sprenger, Philipp G. Sandner and Isabell M. Welpe. Predicting Elections with Twitter: What 140 Characters Reveal about Political Sentiment. In Proceedings of AAAI Conference on Weblogs and Social Media (ICWSM), 2010.
  4. Bo Pang, Lillian Lee and Shivakumar Vaithyanathan. Thumbs up? Sentiment Classification using Machine Learning Techniques. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), 2002.
  5. Chenhao Tan, Lilian Lee, Jie Tang, Long Jiang, Ming Zhou and Ping Li. User Level Sentiment Analysis Incorporating Social Networks. In Proceedings of ACM Special Interest Group on Knowledge Discovery and Data Mining (SIGKDD), 2011.
  6. Efthymios Kouloumpis, Theresa Wilson and Johanna Moore. Twitter Sentiment Analysis: The Good the Bad and the OMG! In Proceedings of AAAI Conference on Weblogs and Social Media (ICWSM), 2011.
  7. Hatzivassiloglou, V., & McKeown, K.R.. Predicting the semantic orientation of adjectives. In Proceedings of the 35th Annual Meeting of the ACL and the 8th Conference of the European Chapter of the ACL, 2009.
  8. Johann Bollen, Alberto Pepe and Huina Mao. Modelling Public Mood and Emotion: Twitter Sentiment and socio-economic phenomena. In Proceedings of AAAI Conference on Weblogs and Social Media (ICWSM), 2011.
  9. Luciano Barbosa and Junlan Feng. Robust Sentiment Detection on Twitter from Biased and Noisy Data. In Proceedings of the international conference on Computational Linguistics (COLING), 2010.
  10. Peter D. Turney. Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews. In Proceedings of the Annual Meeting of the Association of Computational Linguistics (ACL), 2002.
  11. Rudy Prabowo and Mike Thelwall. Sentiment Analysis: A Combined Approach. Journal of Infometrics, Volume 3, Issue 2, April 2009, Pages 143-157,2009.
  12. Samuel Brody and Nicholas Diakopoulus, Cooooooooooooooollllllllllllll!!!!!!!!!!!!!! Using Word Lengthening to Detect Sentiment in Microblogs. In Proceedings of Empirical Methods on Natural Language Processing (EMNLP), 2011.
  13. Soo-Min Kim and Eduard Hovy. Determining the Sentiment of Opinions. In Proceedings of International Conference on Computational Linguistics (ICCL), 2004.
  14. Stefano Baccianella, Andrea Esuli, Fabrizio Sebastiani. SENTIWORDNET 3.0: An Enhanced Lexical Resource for Sentiment Analysis and Opinion Mining. In Proceedings of international conference on Language Resources and Evaluation (LREC), 2010.
  15. Theresa Wilson, Janyce Wiebe and Paul Hoffmann. Recognizing Contextual Polarity in Phrase-Level Sentiment Analysis. In the Annual Meeting of Association of Computational Linguistics: Human Language Technologies (ACL-HLT), 2005.
  16. Ian H. Witten, Eibe Frank & Mark A. Hall. Data Mining – Practical Machine Learning Tools and Techniques.

1. Ricgard O. Duda, Peter E. Hart & David G. Stork: Pattern Classification. Steven Bird, Even Klein & Edward Loper. Natural Language Processing with Python.
2. Ben Parr. Twitter Has 100 Million Monthly Active Users; 50% Log In Everyday.