IMAM MUHAMMAD IBN SAUD ISLAMIC UNIVERSITY
COLLEGE OF COMPUTER AND INFORMATION SCIENCES
DEPARTMENT OF COMPUTER SCIENCE

ARABIC TWEETS CLASSIFIER AND ASSESSMENT

Written By

Ahmed A. Alosaimi **&** Khalid M. Alnuaim

Supervised By

Dr. Adel Alsalem

May 2013

This work dedicated to my mother and father, for their unconditional support.

--Ahmed

This work dedicated to my parents for their love, endless support and encouragement.

--Khalid

Abstract

With the rapid growth of social networks like Twitter, there is a need to give it more focus. What we seek in this thesis is to direct the user to explore the aspects that have a major impact on his followers. This done by analyzing the user contribution thru many steps cleaning, classifying and measuring the impact.

Therefore, we have developed a web application that classifies the user's tweets and displays the impact makes on his followers.

Keywords- Twitter, Classification, Social network analysis.

Acknowledgments

First and foremost, we would like to take this opportunity to thank our supervisor **Dr. Adel Alsalem** for his great advices, support, and for his valuable time and effort which was the main influence in the success of this project.

Special thanks for **Dr. Ahmed Khorsi** and **Eng. Wael Alalwani** for their advice and support during our project.

We also would like to thanks everyone who help us to make this project a batter project and more efficient, by give us their advice, spreading and answering the questionnaire survey.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background of Study

Nowadays Social Networks become very important and part of our daily life. Day after day, social networks become bigger and bigger. Moreover, people are looking for ways to focus their effort and energy in something that meaningful and interesting for their followers. Therefore, knowing the tweets impact will give a clear picture about where to focus your energy.

1.2 Problem Statement

Due to the rapid growth of Twitter globally and, and the lack of system that measure the user's impact on has followers especially in the Arab world. That motivate us to develop an automated tweet classification system that are capable of automatically recognizing and classifying the user's tweets.

1.3 Objective of the project

The objective of the project is to develop a system that retrieves the user's tweets, then automatically recognizes and classifies it into predefine classes and measure the impact based on the follower reaction, and then present the result to the user.

1.4 Scope of the project

The application will only work in Arabic letters tweets. This is because we are focusing to enrich the Arabic content. Moreover, there is already a similar application targeted only the English tweets [13]. In addition, the application only will process the logged in user's tweets.

1.5 Thesis Outline

This thesis consists of five chapters. In the first chapter, we discuss the Introduction, Objectives and scope of the project. In chapter 2, we discuss more on theory and literature reviews that been done. Chapter 3, we discuss the proposed types of classification also the project main steps. In Chapter 4, we will show the implementation of the project and the tools used. A complete test of the system will be discussed in Chapter 5. Finally, in Chapter 6, conclusion and future work are presented.

Chapter 2

Literature Review and Theory

2.1 Twitter Research

Twitter is a popular social network site. Therefore, there is many researches talking about Twitter classification. Christopher Horn [15] develop a web application that classify tweets. Unfortunately, it does not support Arabic tweets.

2.2 Text Classification

Currently, there is a lot of research going on in the area of classification. However, they mostly been developed for English and other European languages. Moreover, the focused on long article classification rather than short sentences like “tweets”. Tarek Fouad Gharib et al. [3] used a support vector machines (SVM) to classify Arabic text. Laila Khreisat [9] classify Arabic text using N-Gram Frequency Statistics.

Chapter 3

Methodology

3.1 Twitter

Twitter [14] is a social networking application that allows people to micro-blog about different topics. The users are limited to only 140 character in each post (tweet). Followers is a concept implemented on Twitter, which is allow you to follow people and informed when they update their status.

3.2 Twitter API

One of the major components of our project is to retrieve the user's tweets, and this done using the Twitter Application Programming Interface (API). After the last API update, all the requests has to be authenticated using the open standard for authorization "*OAuth"* which provides a process for the end user to authorize third-party access to their server resources without sharing their credentials.

3.3 Types of proposed classification

When we started this project we tried to think of ways that help us classify tweets, and we proposed these types.

3.3.1 Based on Hashtags

Hashtags are words or phrases prefixed with the symbol #. It is provide a means of grouping messages. One can search for the hashtag and get the set of messages that contain it [6]. This is one of the methods we consider to classify the tweets, but we faced many problems. First, because not all tweets have a hashtag. Therefore, we cannot classify tweets without hashtag and this is not an efficient way to classify. In addition, hashtags develop very fast that will make it harder to classify. Finally, there could be more than one meaning for single hashtag or different hashtags for the same thing.

3.3.2 Based on pre classification

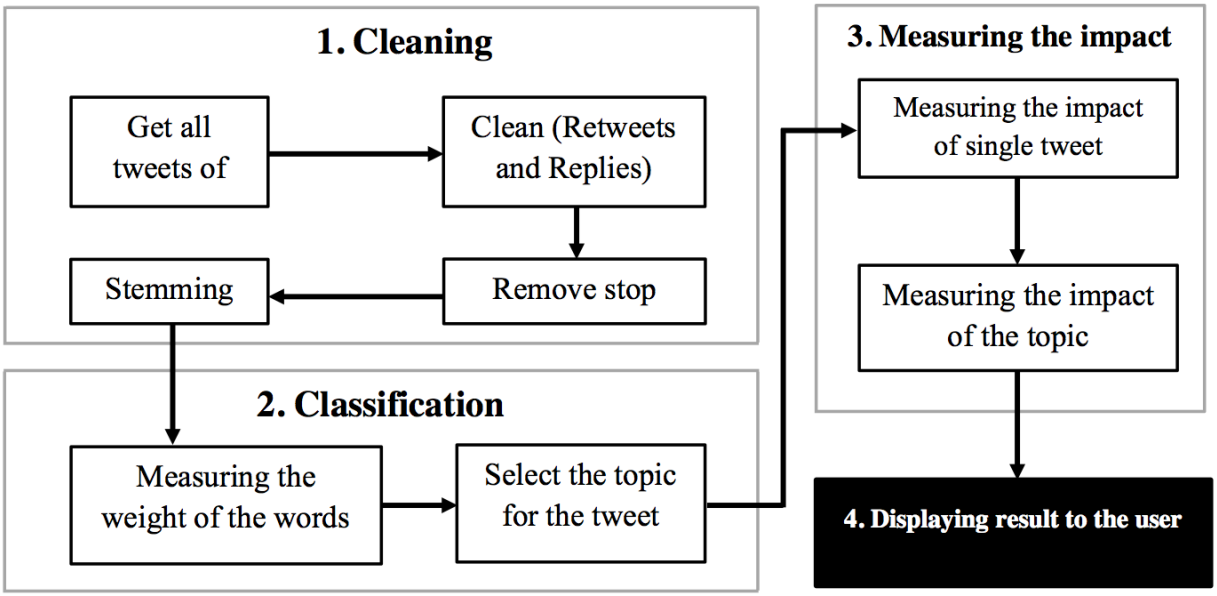
The idea is to ask the user to post all his new tweets using our site and classify it before publishing; however, it is not an efficient way because many of users do not prefer to tweet from specific site or application, also tweeting is usually spontaneous and adding more steps to it will make it not.

3.3.3 Based on words

Tweet classification based on words is the process of classifying tweets into a predefined set of classes based on their content, and this happens after we remove any stop word and stem all the remaining words, and we believe that is the best type for our project.

3.4 Project Steps

The tweets classifying system must pass through a set of steps. [Figure 1] shows the different phases of the system. We divide them into four major steps; Cleaning, Categorization, Measuring the impact and Displaying the result to the user. Moreover, we will explain them in detail. [3]



[Figure 1] Project Steps

3.4.1 Cleaning

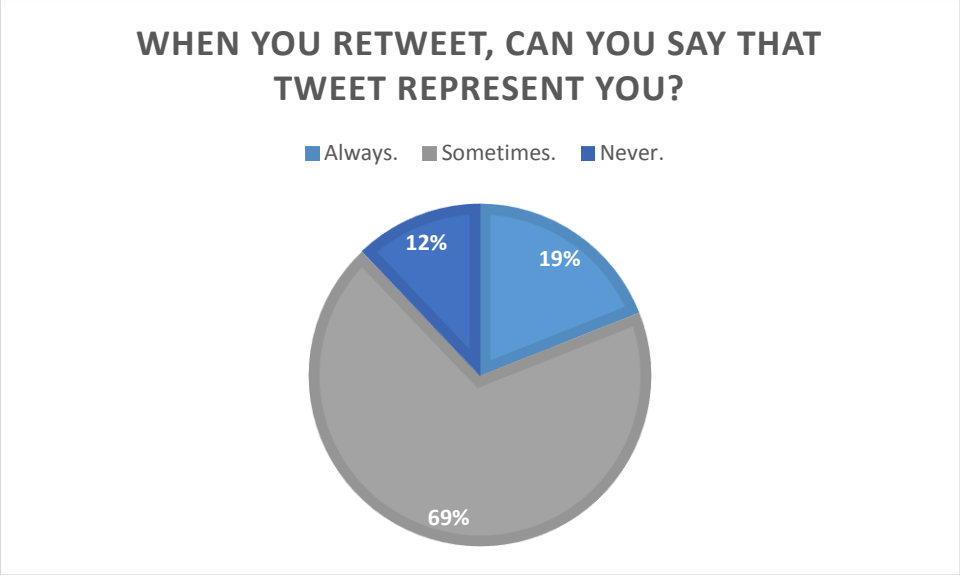
This step is important because it cleans and removes anything that does not represent the user's opinion to gives more accurate result, for example prepositions and particles. We can achieve this by:

3.4.1.1 Get all tweets of user

Retrieving the user's tweets is a simple task done by the "Twitter API". We can retrieve it after the authentication and the approval from the user.

3.4.1.2 Cleaning (Retweets and Replies)

Before moving to the next step, we have to remove the Retweets and Replies from the user tweets. Because it does not represent the opinion of the user and this is based on the results of a questionnaire we published[4] 69% are sometime using the Retweet function for retweeting a tweet that they do not agree with it just to make it visible for their follower or thy retweet something does not represent their opinion.



[Figure 2] Questionnaire result 1

3.4.1.3 Removing short tweets

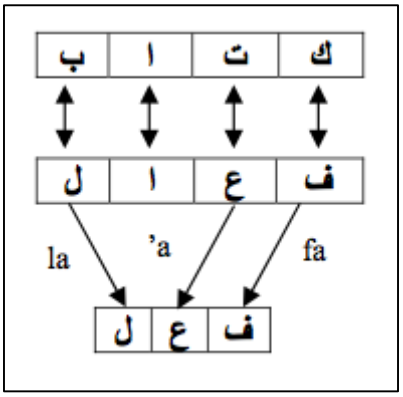
Short tweets it is very hard to classify or maybe it is impossible, for instance a tweet only contain one or couple words. Therefore, we are removing any tweet that only consume 15% of the maximum length or less.

3.4.1.4 Removing stop words

Stop words like prepositions (على، إلى، من...) and particles (إن، يا، و...) considered insignificant words and must be removed. It can be done by using a simple program that removes prepositions and particles from the text using a predefined list of prepositions and particles.

3.4.1.5 Stemming

Stemming is the process of removing all affixes from a word to extract its root. It is essential to improve performance in information retrieval tasks especially with highly inflected language like Arabic language. There are many different approaches for stemming: the root-based stemmer, the light stemmer, and the statistical stemmer. [3] [Figure 3] shows the steps to extract the root letters of the word (كتاب) by pattern matching. [5]



[Figure 3] Steps to extract the root

The stemming can be done by several methods. For example, a database that contain huge number of words with their root or using tools like Waikato Environment for Knowledge Analysis "WEKA" with Arabic stemming algorithms, or using an intelligent equation that calculate the root for a given word.

We decided of using "Al-Shalabi et al." root extractor, it work with three-letter roots, and over 80% of Arabic words have three-letter roots and the most commonly used words in Arabic writing have three-letter roots. The accuracy of the "Al-Shalabi et al." root extractor reported to be over 90%. [7]

"Al-Shalabi et al." root extractor works by given weight for certain Arabic letters as shown in [Figure 4].

Arabic Letters	Weight
ا ة	5
ء ى ي	3.5
ت ى و	3
أ إ م ن	2
ل س ه	1
Rest of the Arabic Alphabet	0

[Figure 4] Assignments of letters to weights

Then ranking each letter based on his position in the word and this is differ from add and even [Figure 5].

Letter Position From Right	Rank (If Word Length Is Even)	Rank (If Word Length Is Odd)
1	N	N
2	N - 1	N - 1
3	N - 2	N - 2
.	.	.
$\lceil N / 2 \rceil$	N / 2 + 1	$\lceil N / 2 \rceil$
$\lceil N / 2 \rceil + 1$	N / 2 + 1 - 0.5	$\lceil N / 2 \rceil + 1 - 1.5$
$\lceil N / 2 \rceil + 2$	N / 2 + 2 - 0.5	$\lceil N / 2 \rceil + 2 - 1.5$
$\lceil N / 2 \rceil + 3$	N / 2 + 3 - 0.5	$\lceil N / 2 \rceil + 3 - 1.5$
.	.	.
N	N - 0.5	N - 1.5

[Figure 5] Order of letters

The last step done by calculate the product of the weight for each letter and his rank, and then the root will be the three less product letters [Figure 6].

Word	المحافظة							
Letters	ة	ظ	ف	ا	ح	م	ل	ا
Weight	5	0	0	5	0	2	1	5
Rank	7.5	6.5	5.5	4.5	5	6	7	8
Product	37.5	0	0	22.5	0	12	7	40
Root	حفظ							

[Figure 6] Example of using the root extractor

We faced some problem with this algorithm, for instance, some words supposed to have the same root but they get different roots and this will have a little affect in classification.

3.4.2 Classification

Given the length of each tweet is 140 character; it has its advantages and disadvantages. One of the advantages, that it is usually focuses on one topic. This adds simplicity in the algorithm for the classification. The disadvantages is that it is hard to determine the topic because of the words used are not clear enough or sometime to short.

3.4.2.1 Measuring the weight of the words

The method we are using for measuring the weight of the words is by utilizing a predefined set of topics. Each topic has a group of words divided into three groups with values (3, 2, and 1). Each group is variable depending on the number of iterations for each word. The first 100 words that has more iterations will be in the first category with value of (3), next 100 words will be in next category with value of (2), the last category contains the next 100 words with value of (1), the rest will not have any value unless it exceeds one of the words of any categories.

After we have defined our categories and their words, we measure all the words and determine the topic based on the higher topic ranking. This done by using machine-learning techniques that figure the topic for each tweet and add all the word in this tweet to the database under this topic to improve it, with the ability to monitor each topic if need it to ensure it is working correctly.

3.4.2.2 Selecting the topic for the tweet

Due to the special nature of the Arabic users, general approaches are inadequate for them. Therefore, we published a questionnaire to know what are the topics they are interested in and tweeting about. Based on the results of a questionnaire we published [4] and a research paper [9], we found that people are usually tweeting under these topics:

1. Sports.
2. Politics.
3. Economy.
4. Religion.
5. Technology.
6. Art.
7. Fashion.
8. Literature.
9. Entertainment.
10. Personal matters (Miscellaneous).

We will only consider these topics and collect the important words that do not belong to any of the above topics, and then we will create new topics based on these words. We are also considering using a deeper classification, for example instead of sport we could have category for specific kind of sport. One of the approaches that we considered along the way, a tool that uses "Wikipedia" as source to determine the topic. The problem with this approach is that social networks are faster developing than Wikipedia. [8]

We faced some problem when classify by getting about 60% incorrect result, so improved the way we classify and it done through several stages. First, we selected 100 random tweets from the 52,000 tweets in the database to test the improvement, from these 100 tweets we kept only the tweets that belong to one class and we end up with 40 tweets.

First attempt is by reducing the number of words in each group from 100 words to 10 words, this attempt did not improve anything, so we moved to the next stage.

By duplicating the word in the Hashtag, we improve the result by 20%, but we still getting around 40% incorrect result. Therefore, we changed the value for each group, so the first word has a different value than the last word and this is improved the result about 10% using this equation:

Value of the word = (group value) + (1 - (position of the word / 100))

3.4.3 Measuring the impact

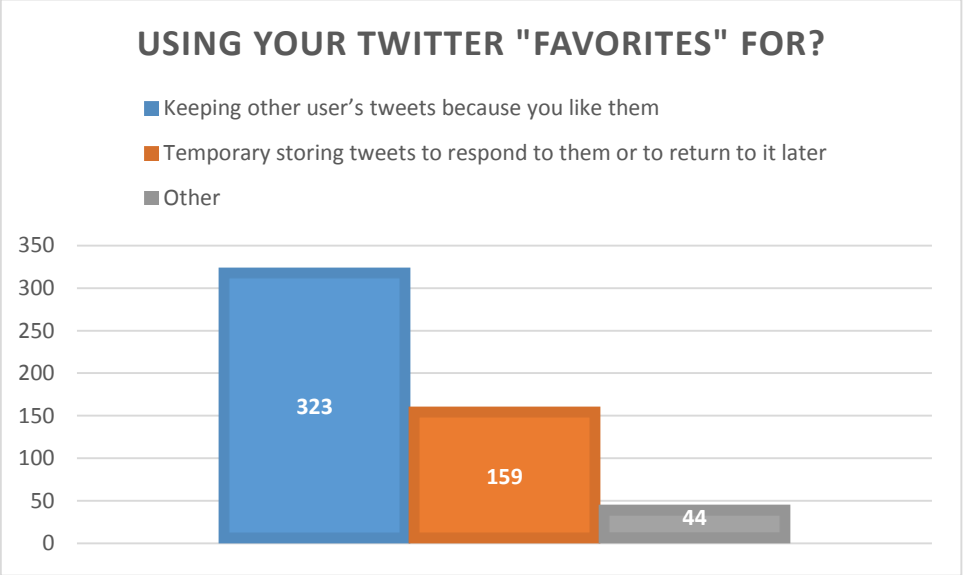
By using the data we collect, we now can measure the impact by following the next two steps.

3.4.3.1 Measuring the impact of single tweet

The impact and the strength of single tweet depends on three key factors Replies, Favorites and Retweets. Through these factors, we can measure the impact of the single tweet on followers. We will talk about each of these factors and its influence.

1. **Replies** has a huge impact on the user. In fact, there is one kind of impact measurements for users that depends entirely on replies.

2. **Favorites**, the misuse of favorites it considered the big disadvantage. Based on the results of a questionnaire we published [Figure 7] [4] 30% are misusing the favorites as a temporary repository for links or for a different use.



[Figure 7] Questionnaire result 2

3. **Retweets** is very important in measuring the impact of single tweet and helps spread the tweet to the largest possible number of people. Even though some people are using this feature in the wrong way, like retweeting a user just for mockery, but that does not mean that it does not have any effect. Once it reaches more people, it means more viewers.

After we discussed each factor, we come up with this equation to calculate the impact of single tweet.

Impact of single tweet = (# Replies * 0.4) + (# Retweets * 0.4) + (# Favorites * 0.2)

After the latest release for Twitter API, Twitter added more limitation to the API by removing the request for Replies and Favorites, which was done in the beginning of our research in this project by using a single request.

3.4.3.2 Measuring the impact of the topic

The impact of the topic depends entirely on the impact of single tweet. Therefore, to calculate the impact for a topic we add all the points for each tweet under this topic then divide it on the total number of tweets under the topic.

Impact of topic = (total point of single tweet for the topic) / total # of tweets for the topic

3.4.4 Displaying result to the user

After we calculate the impact for each topic, we present the result to the user in order. The topic with the most impact is placed on the top then the second one and so on.

3.5 Policies and privacy

It is important to consider these policies when developing a web application and how to deploy it in the application. Therefore, we divided this section to two parts, since we are using Twitter API we will first talk about it then we will talk about our policies and privacy.

3.5.1 Twitter API policies

Twitter API is very concerned about the user privacy. Therefore, it prevents the developer from facilitating or encouraging the publishing of private or confidential information. Also it prevents the developer form storing user's Twitter passwords, it also prevents from storing non-public Twitter Content except at the explicit direction of a Twitter end user. [12]

3.5.2 Our polices and privacy

- Since we are using Twitter API, the authentication done by Twitter. Therefore, we do not store any sensitive user credential like username and password.
- We could use the user data for future studies and statistics with the guarantee of anonymity.

Chapter 4

Implementation

In the implementation section, we will walk through the process and tools we used to develop the project that we built from scratch and wrote every function in it. We will talk about them with brief description.

4.1 ASP.NET

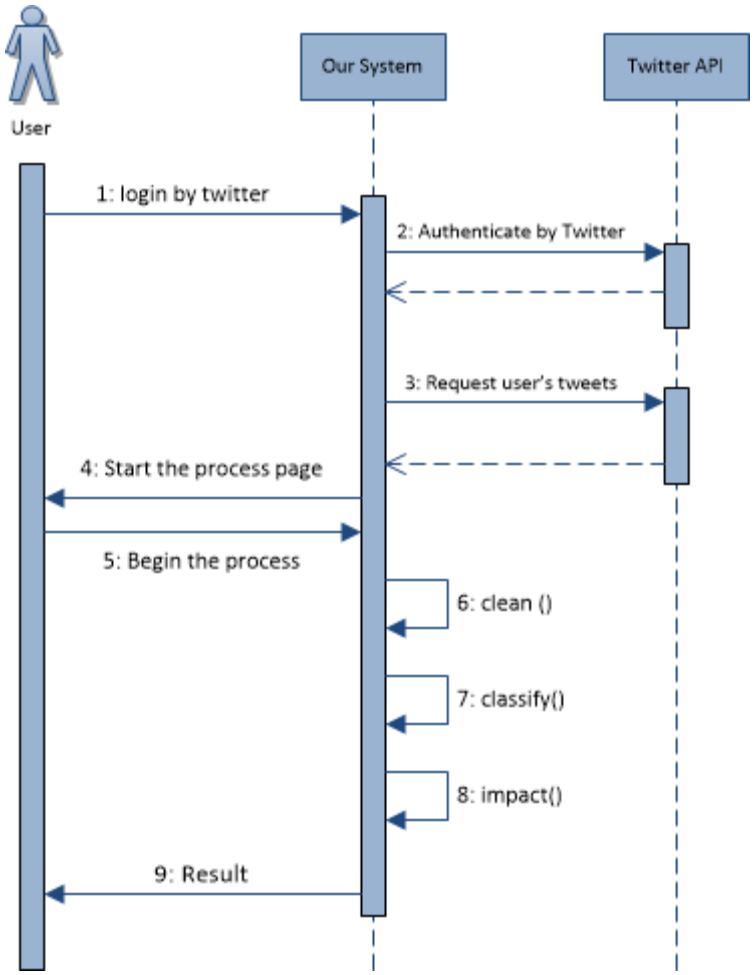
ASP.NET is a framework for developing dynamic web based application. It develop by Microsoft in 2002. [11] ASP.NET has a great community support and that will help us to concentrate in content rather than the environment.

4.2 LINQ to Twitter

LINQ to Twitter is open source third party library communicate with Twitter API using Language Integrated Query (LINQ) which is part of Microsoft .NET Framework. [10] We decided to use this library to add more focus to the main part of our project rather than to deal with Twitter authentication and handling the JASON file that return from each Twitter API request.

4.3 Sequence diagram

Figure 8 shows the Sequence diagram of the classification process for the user.

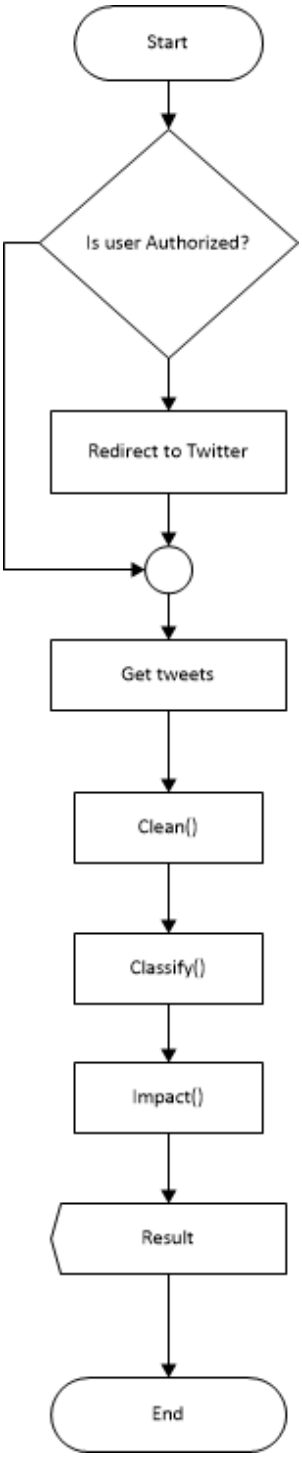


[Figure 8] UML Sequence diagram

1. The user start the login process.
2. The system redirect the user to complete the authentication process. If the user's authentication done correctly the system receive user's "Token" and "Token secret", else the system receive an error.
3. The system request user's tweets from Twitter API.
4. The system present "Start the process" page for the user.
5. The user start the process.
6. The system clean the user's tweets.
7. The system classify the user's tweets.
8. The system measure the impact for the user's tweets.
9. The system present the result to the user.

4.4 Components

As explained previously, the project is consist of four major components and we will explain each one of them from the implementation perspective.



[Figure 9] Flowchart diagram

4.4.1 Cleaning

- **Step 1:** Authenticate the user and retrieve the "user token" and the "user token secret" so we can use it in future request in behalf of the user.
- **Step 2:** Request the user tweets up to 3200 tweets (Due to Twitter API limitation).
- **Step 3:** Remove all Retweets and Replies, this done using the library "LINQ to Twitter" through the tweets request.
- **Step 4:** Remove short tweet**s,** which consists only about 15% of the maximum length.
- **Step 5:** Remove numbers and non-Arabic words using a function that compare each character in the tweet to the Arabic alphabet and remove any character that does not exist in it.
- **Step 6:** Remove stop words using a function that compare each word in the tweet to pre define list of stop words and remove any word that exist in the list.
- **Step 7:** Stem the rest of tweet words using a function based on "Al-Shalabi et al." root extractor algorithm.

4.4.2 Classification

- **Step 1:** Determine the class for each word in a single tweet by compare it to a database that contain pre-defined words with their classes.
- **Step 2:** Determine the class for the tweet by selecting the highest class for this tweet.
- **Step 3:** Adding all the words to selected class. if it already in the database increase the counter, if not add it as new word for this class.

4.4.3 Measuring the impact

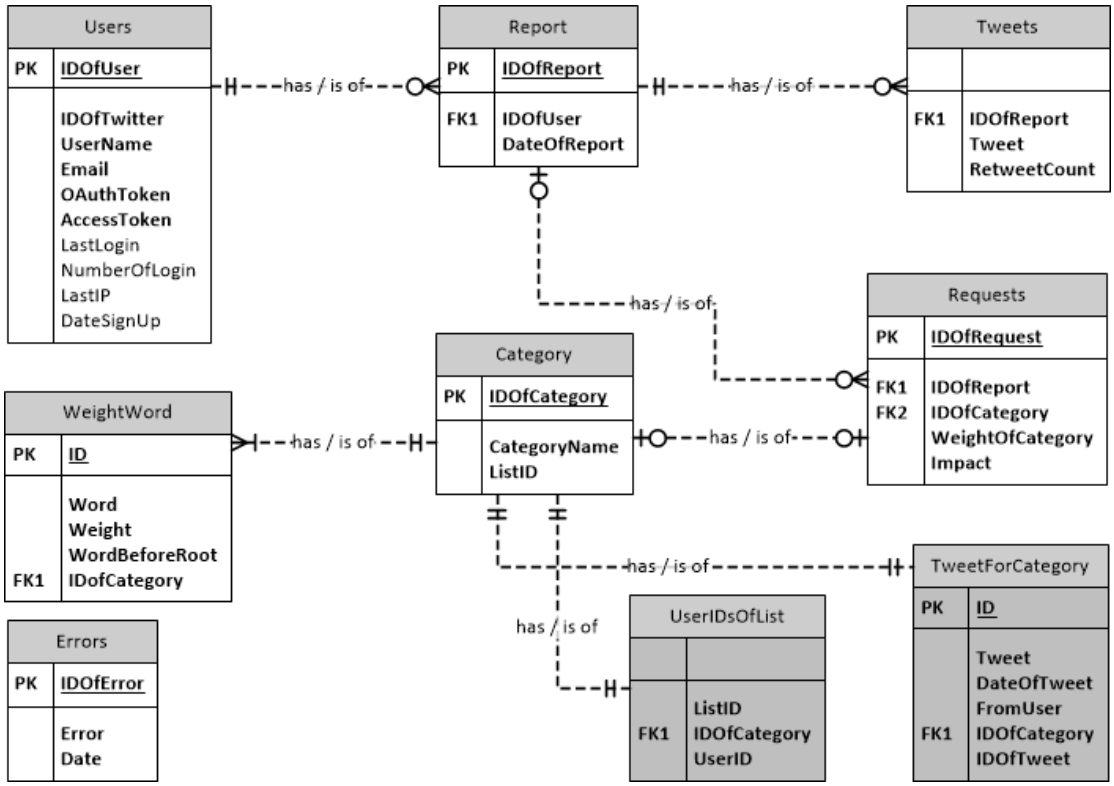
- **Step 1:** The impact for single tweet for now is simply the number of Retweets (Due to Twitter API limitation we exclude the number of Replies and the number of Favorites until farther update to the API).
- **Step 2:** Calculating the impact for a class by adding all the Retweets count under this class and dived on the number of tweets for this class.

4.4.4 Displaying result to the user

- **Step 1:** Displaying each class with the impact ordered by the highest.

4.5 Databases

We used SQL server to manage are database. In addition, we build our database to be dynamic as possible, for instance we made it possible to add more topics if needed in the future. [Figure 10] shows the ER diagram of the database we used.



[Figure 10] ER diagram

4.5.1 Main tables

Users table contains all users information, IDOfUser is unique garneted from our system, in the other hand IDOfTwitter is the user's unique ID on Twitter. In addition, it contains OAuthToken and AccessToken, which are reserved from Twitter after the authentication. The **category** table contains a unique ID and name, in addition to ListID, which is, refer to the list we create on Twitter that concern about this category (the list in Twitter is a placeholder for Twitter accounts, created by any one).

Report table contains unique ID, IDOfUser that request this report and DateOfReport, in order to decide if a new report need it or not (due to the limitation of requests in Twitter API). Each report a number of requests depending on how many topic we have and thy store in **Request** table with their weight between categories and their impacts, to make it more dynamic.

We stored the user's tweets in **Tweets** table, so we can retrieve without generating a new request to Twitter API. The reasons we stored the actual tweets rather than the results are because the weight for the word are updating constantly and to get an accurate result we need to process the tweets every time. In addition, we need the tweets to extract the Hashtags from the tweets.

WeightWord table contains a unique ID for each word and contains the weight of each word, which is the number of iterations. For this table we have two cases, first if the word is a new word and not in the table, in this case we use this query:

```
INSERT INTO WeightWord
            (Ward, Weight, CategoryID, WardNoRoot)
VALUES      (@Ward, 1,@CategoryID,@WardNoRoot)
```

The second case if the word already existed in the table, so we use this query:

```
UPDATE      WeightWord
SET         Weight = Weight + 1
WHERE       (Ward = @Ward) AND (CategoryID = @CategoryID)
```

The last table is **Errors** table, which store all the errors that faces the users, so we can fix it.

4.5.2 Supporting tables

UserIDsList and **TweetForCategory** tables are created to generate the initial values of words for each topic. The first table is for storing the Twitter users ID for each list so we can retrieve their tweets and store it the second table.

4.6 Security

We added an authentication function in each page that check if the user that logged in is the same user browsing to prevent session hijacking both in client interface and control panel.

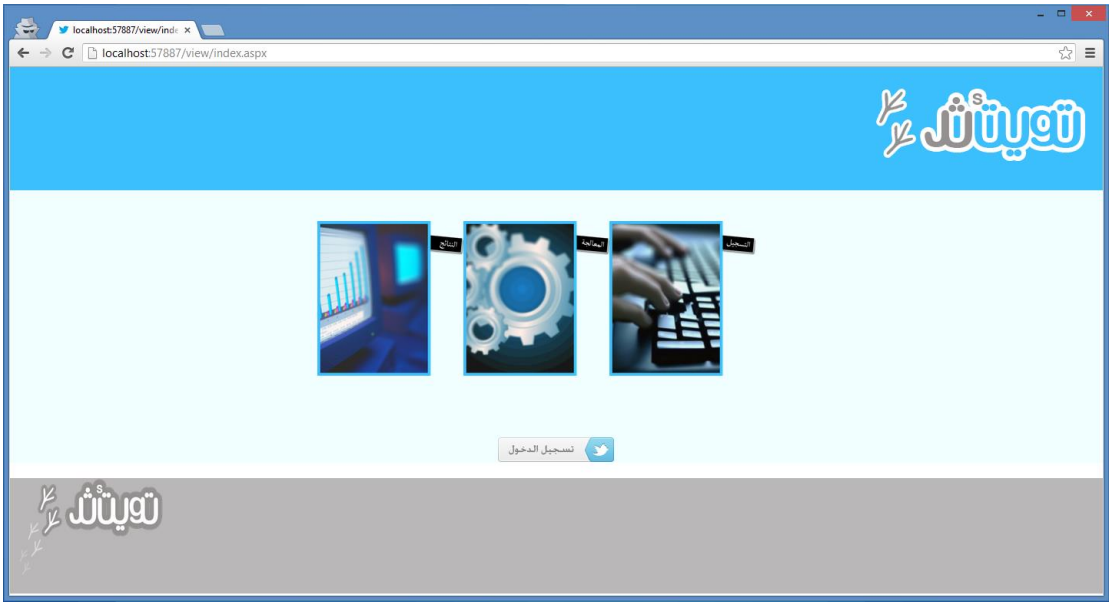
4.7 Supported Tools

The way we choose to classify is based on the word frequency for each topic. To start we needed initial words for each topic. Therefore, we create twitter account, then configured a list for each topic and added specialized accounts for each topic, then we develop a tool that retrieve tweets form these accounts, calculate the frequency, and store it in a database as initial value. We analyzed over 52,000 tweet in variety of topics to get the most accurate results as possible.

4.8 GUI

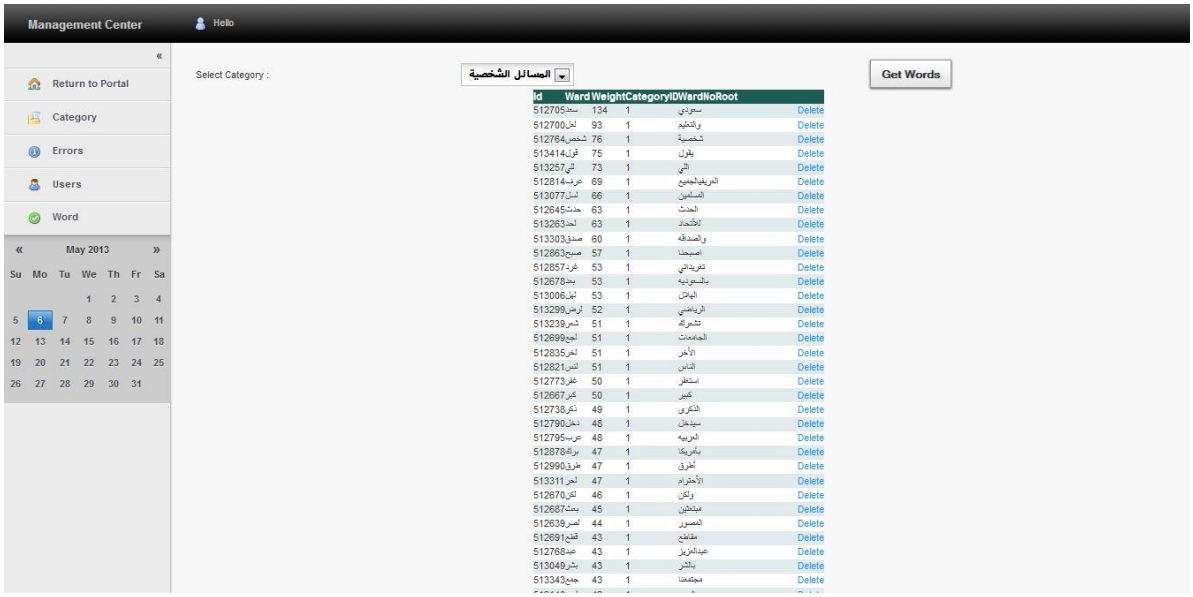
We aimed to make the graphical user interface simple and easy to use. We design and used open source “jQuery” functions, which is a multi-browser JavaScript library, designed to simplify the client-side scripting of HTML [16]. In addition, we used “Twitter Bootstrap” which is a free collection of tools for creating websites and web applications [17].

4.8.1 Client interface



[Figure 11] Print screen for client interface

4.8.2 Control panel

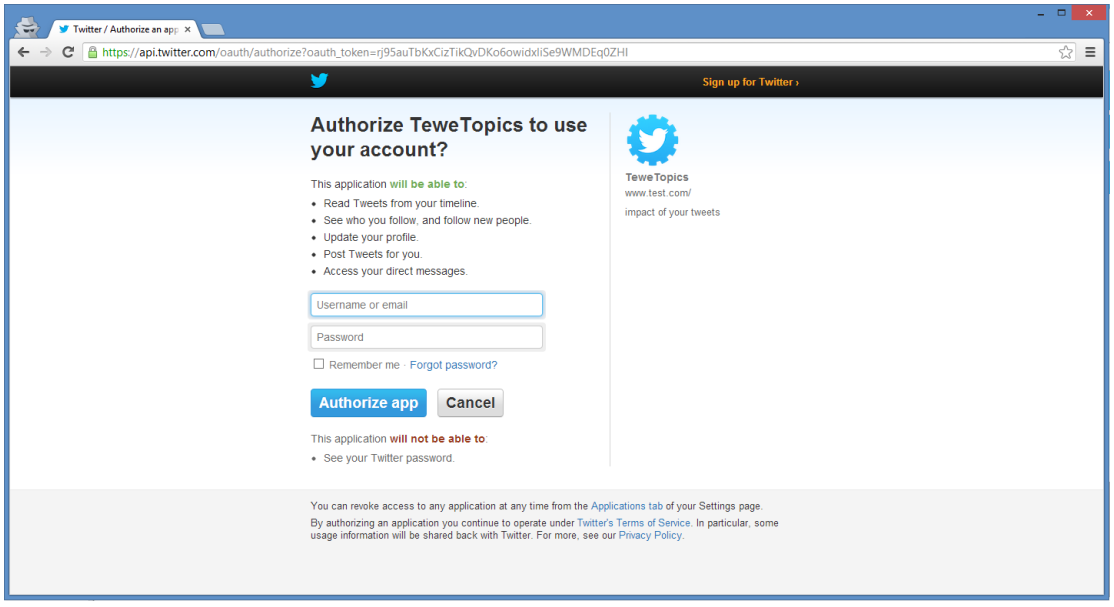


[Figure 12] Print screen of Control panel

Chapter 5

Testing

5.1 Authentication



[Figure 13] print screen of Authentication

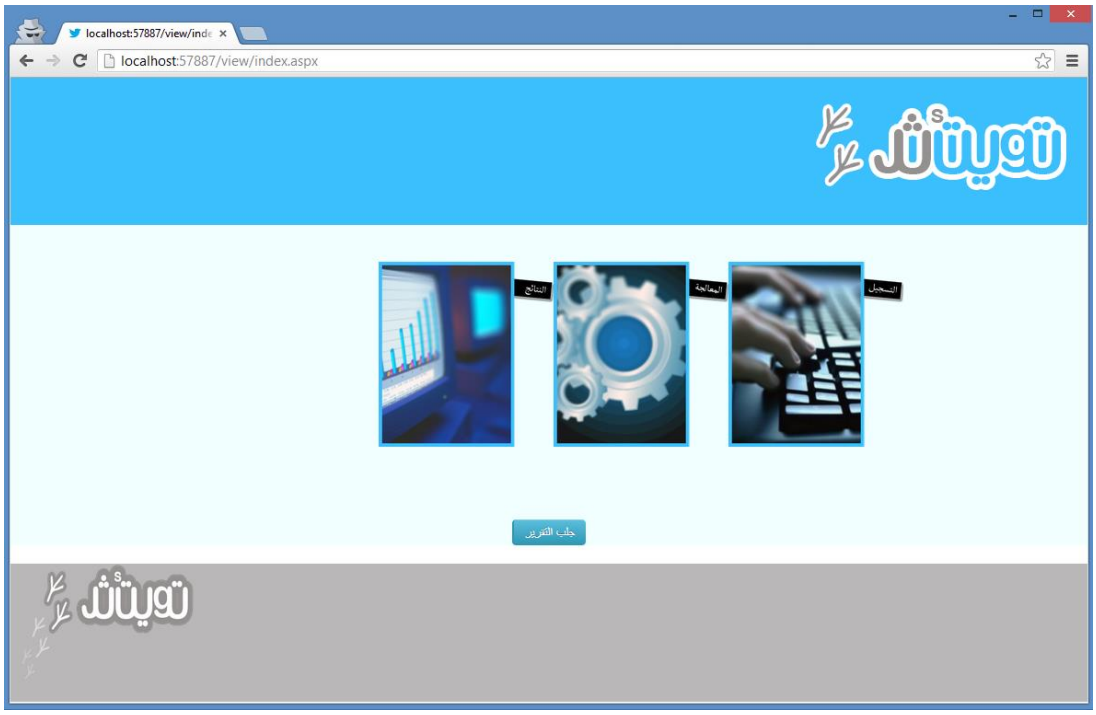
5.2 Registration

[Figure 14] Print screen of Registration

5.3 Polices and privacy

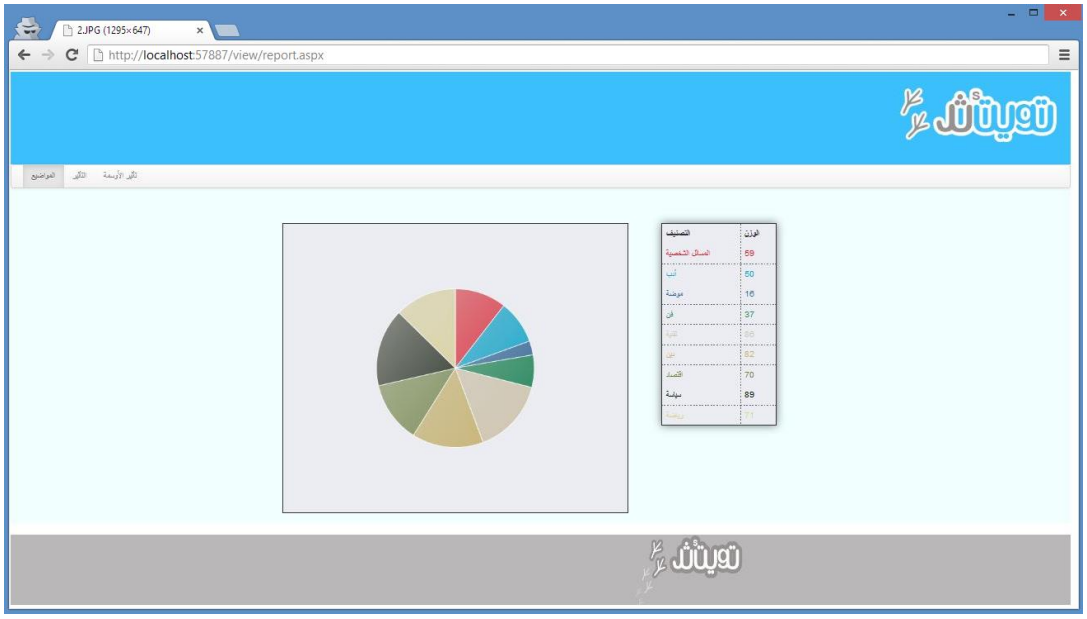
[Figure 15] Print screen of Polices and privacy

5.4 Begin the process



[Figure 16] Print screen of Begin the process

5.5 Result



[Figure 17] Print screen of Result

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this report, we used word-based classification to classify Arabic tweets after clean the tweet by removing any stop words and stem the remaining words. In addition to using, an equation for measuring the impact of the topics for the user based on the impact of his tweets.

6.2 Future Work

- Using the Hashtag in the classification process will grate improve to our system, especially of we figured the direction of this hashtag by taking a simple and run it through the system.
- Using the use's Bio (small biography present it in the user's profile) in the classification process, especially if there a point equality between two classes.
- Improving the classification algorithm also improving the stemming algorithm.
- Using database contain words dictionary to compare the rooted word, is it correct or not.

References

[1] Esam A. Alwagait “Conference of Analysis of information in social networks”, 2011.

[2] Social networking service – Wikipedia.

[3] Tarek F. Gharib, Mena B. Habib, Zaki T. Fayed “Arabic Text Classification Using Support Vector Machines”.

[4] Questionnaire taken by over 360 twitter users.

[5] M. M. Syiam, Z. T. Fayed, M. B. Habib “AN INTELLIGENT SYSTEM FOR ARABIC TEXT CATEGORIZATION” IJICIS, Vol.6, No. 1, JANUARY 2006.

[6] Hashtag - Wikipedia.

[7] Rehab Duwairi “Arabic Text Categorization” The International Arab Journal of Information Technology, Vol. 4, No. 2, April 2007.

[8] Arakawa, Y., Tagashira, S., Fukuda, A. “Hot topic detection in local areas using Twitter and Wikipedia”, ARCS Workshops (ARCS), 2012.

[9] Laila Khreisat “Arabic Text Classification Using N-Gram Frequency Statistics A Comparative Study” Conference on Data Mining, DMIN'06

[10] LINQ to Twitter - linqtotwitter.codeplex.com

[11] ASP.NET – Wikipedia.

[12] Developer Rules of the Road - dev.twitter.com.

[13] Example for English application - tweettopicexplorer.neoformix.com.

[14] Twitter – twitter.com.

[15] Christopher Horn “Analysis and Classification of Twitter messages” Master's Thesis at Graz University of Technology, 2010.

[16] JQuery – Wikipedia.

[17] Twitter Bootstrap – Wikipedia.

Appendix A

Questionnaire Survey

A.1 Questionnaire questions

Your use for Twitter's properties and services

A quick survey about your use for Twitter's properties and services, please kindly answer all questions.

You use the "Retweet" in Twitter for?*

Choose answers that suit you (more than one choice)

- ☐ "Retweet" tweet you like.
- ☐ "Retweet" tweet you disagree with it just to show it to your followers.
- ☐ Other: ……. .

Using your Twitter "Favorites" for?*

Choose answers that suit you (more than one choice)

- ☐ Keeping other user's tweets because you like them.
- ☐ Temporary storing tweets to respond to them or to return to it later.
- ☐ Other: …….. .

Do you use "Retweet" for tweet you disagree with it just to show it to your followers?*

- ○ Always.
- ○ Sometimes.
- ○ Never.

You mostly use Twitter for?*

- Writing tweets.
- "Retweets" others.
- Observing without participation.

When you retweet, can you say that tweet represent you?*

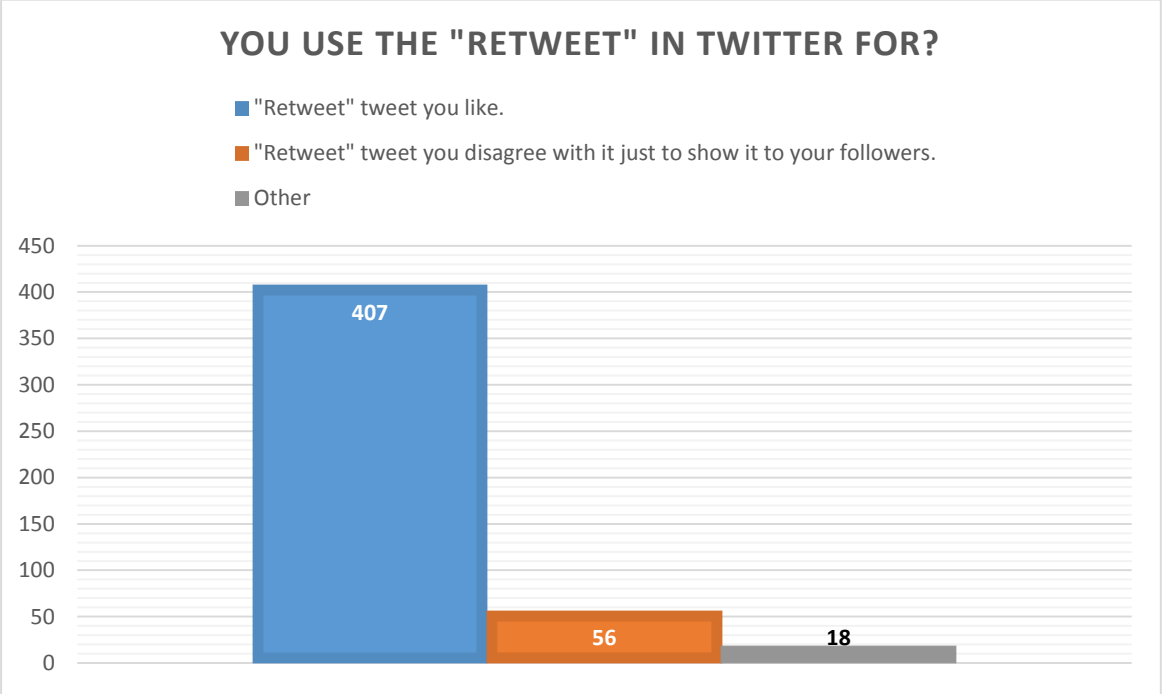
- Always.
- Sometimes.
- Never.

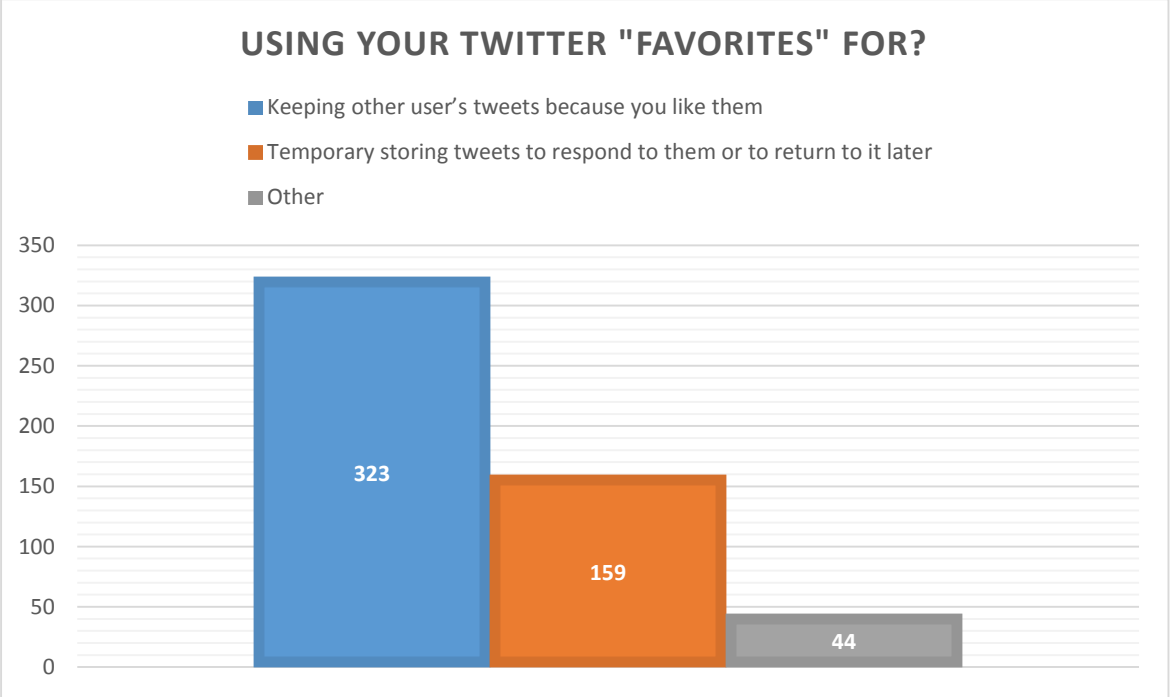
What are the areas that you often tweet about it?*

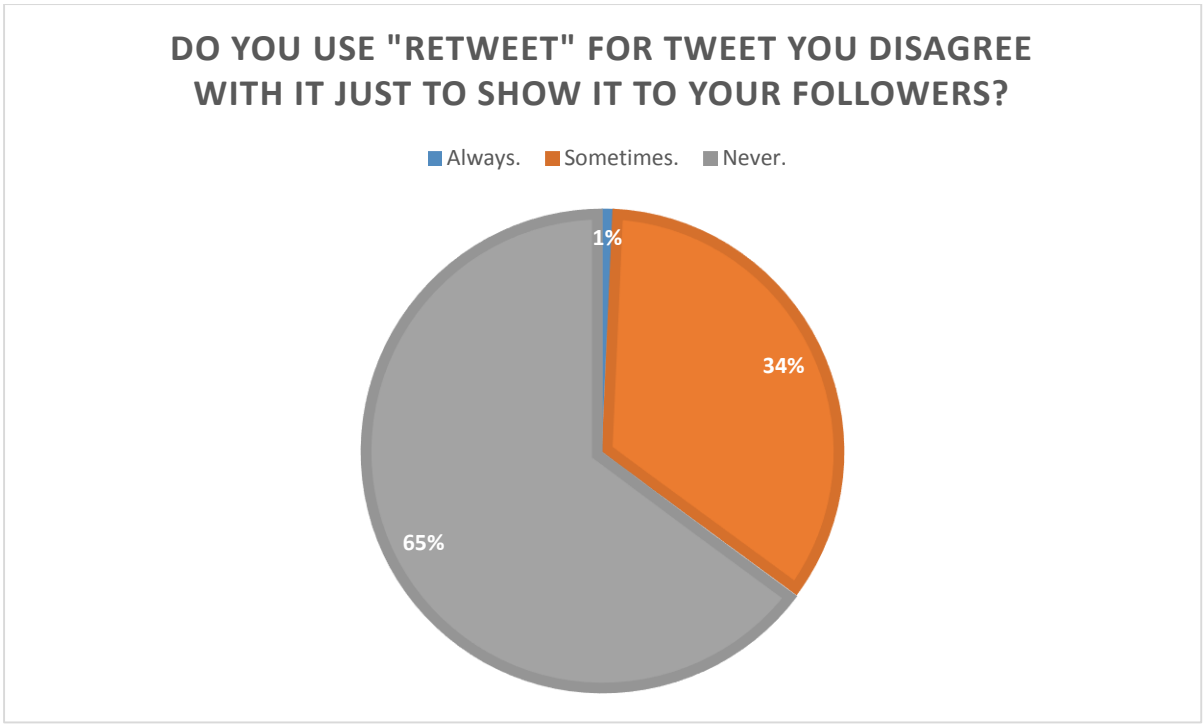
Choose answers that suit you (more than one choice)

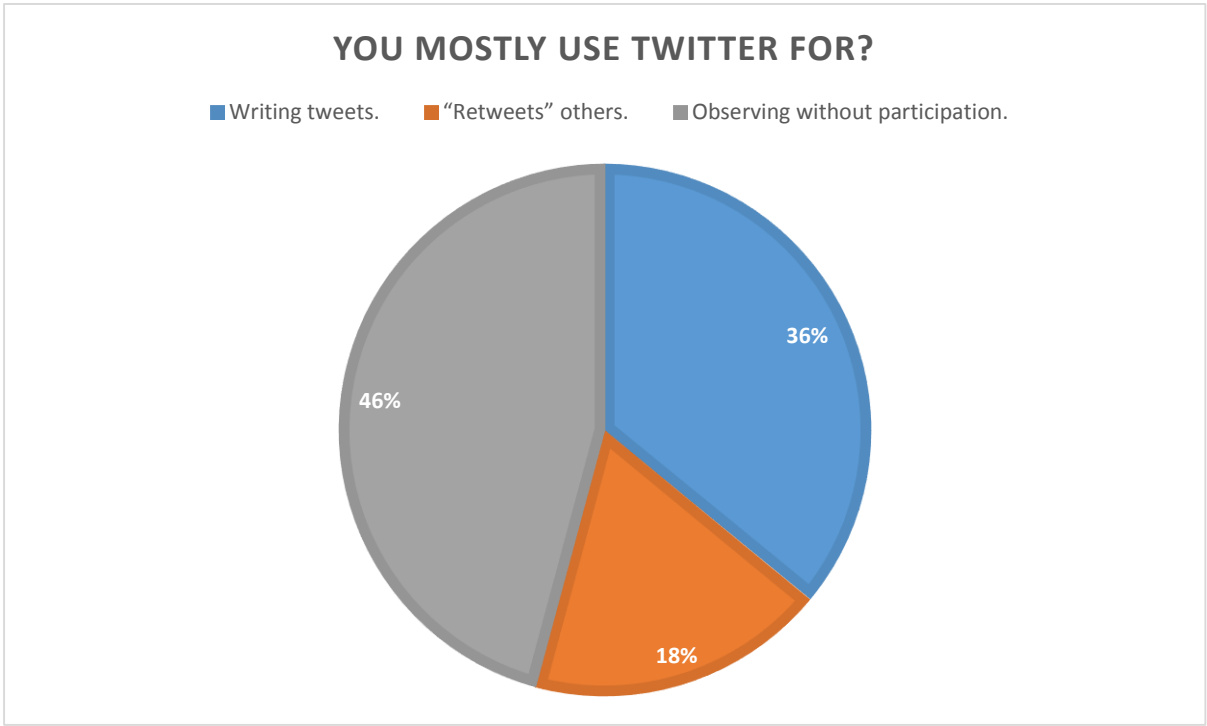
- ☐ Sports.
- ☐ Politics.
- ☐ Economy.
- ☐ Religion.
- ☐ Technology.
- ☐ Art.
- ☐ Fashion
- ☐ Language and Literature.
- ☐ Personal matters.
- ☐ Other: …….. .

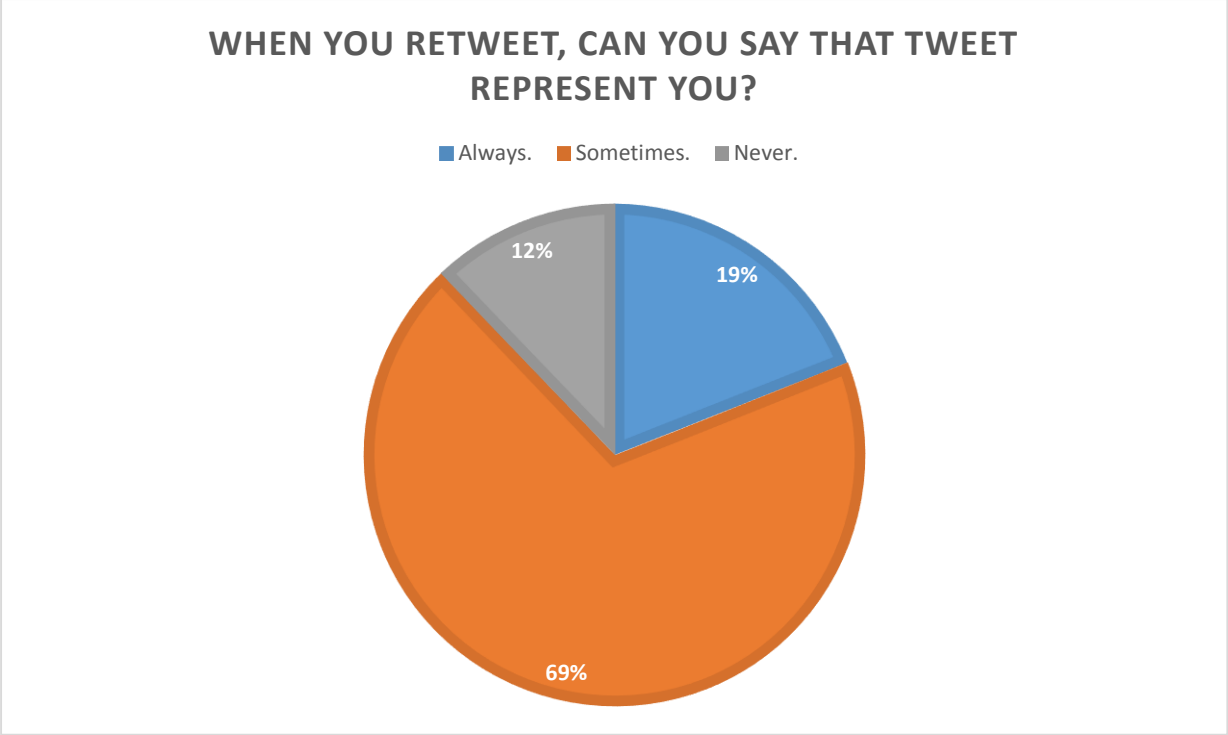
A.2 Questionnaire results

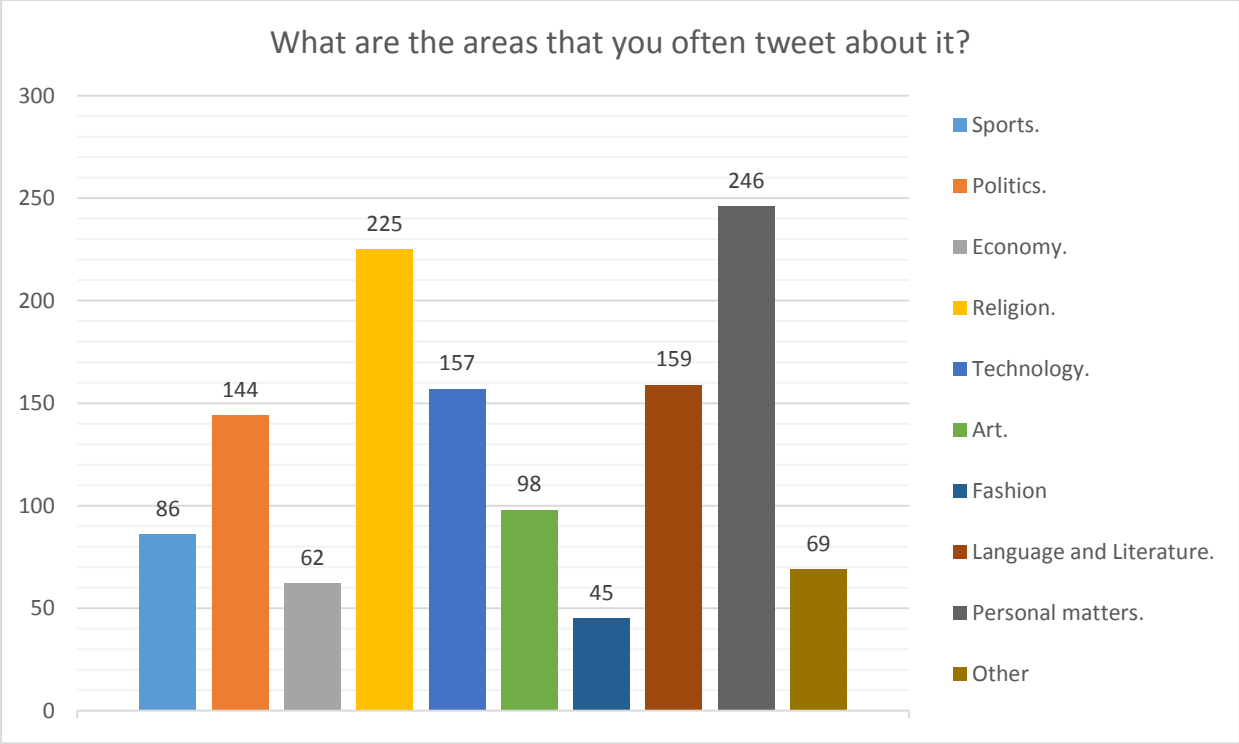





DO YOU USE "RETWEET" FOR TWEET YOU DISAGREE WITH IT JUST TO SHOW IT TO YOUR FOLLOWERS?
Always.
Sometimes.
Never.
1%
34%
65%


YOU MOSTLY USE TWITTER FOR?
Writing tweets.
"Retweets" others.
Observing without participation.
36%
46%
18%


WHEN YOU RETWEET, CAN YOU SAY THAT TWEET REPRESENT YOU?
Always.
Sometimes.
Never.
12%
19%
69%


What are the areas that you often tweet about it?
300
250
200
150
100
50
0
86
144
62
225
157
98
45
159
246
69
Sports.
Politics.
Economy.
Religion.
Technology.
Art.
Fashion
Language and Literature.
Personal matters.
Other

Appendix B

Source code (Selected Sample)

B.1 Clean

B.1.1 Stop word removal

```
//// import stopword from file
        static string stopwordList;
        public string cleanHashTag(string Tweet)
        {
            Tweet = Tweet.Replace('#', ' ');
            Tweet = Tweet.Replace('_', ' ');
            return Tweet;
        }
        //// counter of number stopword removed
        static int intNumWordRmov = 0;

        public int getNumWordRmov() { return intNumWordRmov; }


        //!!!!!!!!!!!!!!!!!!!!  ONLY CALL THIS FUNCTION  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        //============================================================================
        //######  START ###################### REMOVE STOPWORDS #######################
        public string stopwordsRemove(string strTXT)
        {
            strTXT = this.cleanHashTag(strTXT);
            intNumWordRmov = 0;

            //// read stopword.txt file
            stopwordList = File.ReadAllText(AppDomain.CurrentDomain.BaseDirectory +
"/extra/stopword.txt");

            //// removing hash and http and non arabic word from input txt
            strTXT = hashRemove(strTXT);
            strTXT = httpRemove(strTXT);
            strTXT = removeNonArabic(strTXT);

            //// replising lines and spaces with # and split each word into array
            strTXT = strTXT.Replace(Environment.NewLine, "#");
            strTXT = strTXT.Replace(" ", "#");
            string[] arrIN = strTXT.Split('#');

            //// spliting stopword string inro array
            stopwordList = stopwordList.Replace(Environment.NewLine, "#");
            string[] arrLST = stopwordList.Split('#');

            //// comper each input with all stopword if equl removit and add 1 to counter
            for (int i = 0; i < arrIN.Length; i++)
            {
                for (int j = 0; j < arrLST.Length; j++)
                {
                    if (arrIN[i] == arrLST[j]) { arrIN[i] = ""; intNumWordRmov++; }
                }
            }

            //// join input array into out string with speace and return
            string strOUT = "";
            strOUT = string.Join(" ", arrIN);
            strOUT = Regex.Replace(strOUT, @"^\s+$[\r\n]*", "", RegexOptions.Multiline);
            return strOUT;
        }
        //######  END ###################### REMOVE STOPWORDS #######################
        //============================================================================
        //!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
//############################ REMOVE HASH ##############################
static string hashRemove(string strIN)
{
    strIN = strIN.Replace(Environment.NewLine, " ");
    string[] arrIN = strIN.Split(' ');

    for (int i = 0; i < arrIN.Length; i++)
    {
        if (arrIN[i].StartsWith("#")) { arrIN[i] = ""; }
    }


    string strOUT = "";
    strOUT = string.Join(" ", arrIN);
    return strOUT;
}



//############################ REMOVE HTTP ##############################
static string httpRemove(string strIN)
{
    strIN = strIN.Replace(Environment.NewLine, " ");
    string[] arrIN = strIN.Split(' ');

    for (int i = 0; i < arrIN.Length; i++)
    {
        if (arrIN[i].StartsWith("http://")) { arrIN[i] = ""; }
    }


    string strOUT = "";
    strOUT = string.Join(" ", arrIN);
    return strOUT;
}



//############################ REMOVE NONE ARABIC ########################
static string removeNonArabic(string strIN)
{

    strIN = strIN.Replace(" ", "#");
    strIN = strIN.Replace(Environment.NewLine, "#");
    string strArabicChrcts = "ءاأإآئؤبتةثجحخدذرزسشصضطظعغفقكلمنهوي";

    // convert strIN to array char
    char[] buffer = strIN.ToCharArray();
    for (int i = 0; i < buffer.Length; i++)
    {
        //char letter = buffer[i];
        if (!(buffer[i] == '#'))
        {
            if (!(strArabicChrcts.Contains(buffer[i])))
            {
                buffer[i] = '&';
            }
        }

    }


    //removing
    // buffer = buffer.Except('&');

    string strOUT = new string(buffer);
    strOUT = strOUT.Replace("&", "");
    strOUT = strOUT.Replace("#", Environment.NewLine);
    // strIN = strIN.Replace("%", Environment.NewLine);
    return strOUT;

}
```

B.1.2 Stemming

```
// Stemming
        string[] Alphabet = { " ","س","ل","ن","م","إ","أ","و","ى","ت","ي","ئ","ة","ا
ه"" };
        double[] AlphabetWeight = { 5, 5, 3.5, 3.5, 3, 3, 3, 2, 2, 2, 2, 1, 1, 1 };
        string[] MyWordAlphabet;
        double[] MyWordAlphabetWeight;
        double[] MyWordAlphabetRank;
        double[] MyWordAlphabetProduct;
        double[,] RootProduct = new double[3, 2];


        // Stemming
        public string GetRoot(string word)
        {
            // if length of word is less then 4 char return word without edit as root
            if (word.Length == 2 || word.Length == 3)
            {
                return word;
            }

            string Root = " ";

            this.MyWordAlphabet = new string[word.Length];
            this.MyWordAlphabetWeight = new double[word.Length];
            this.MyWordAlphabetRank = new double[word.Length];
            this.MyWordAlphabetProduct = new double[word.Length];

            // get Weight
            for (int i = 0; i < MyWordAlphabet.Length; i++)
            {
                MyWordAlphabet[i] = word[i].ToString();
                MyWordAlphabetWeight[i] = GetAlphabetWeight(MyWordAlphabet[i]);
            }
            // get Rank
            if (word.Length % 2 == 1)
            {
                Root = "Word is odd :";
                for (int position = 0; position < word.Length / 2; position++)
                {
                    MyWordAlphabetRank[position] = word.Length - position;
                }
                MyWordAlphabetRank[word.Length / 2] = word.Length / 2;
                int j = 1;
                for (int position = (word.Length / 2) + 1; position < word.Length; position++)
                {
                    MyWordAlphabetRank[position] = (word.Length / 2) + j++ - 0.5;
                }
            }
            else
            {
                Root = "Word is even :";
                for (int position = 0; position < word.Length / 2; position++)
                {
                    MyWordAlphabetRank[position] = word.Length - position;
                }
                int j = 1;
                for (int position = (word.Length / 2); position < word.Length; position++)
                {
                    MyWordAlphabetRank[position] = (word.Length / 2) + j++ - 0.5;
                }
            }
            // get Product
            for (int i = 0; i < MyWordAlphabet.Length; i++)
            {
                MyWordAlphabetProduct[i] = MyWordAlphabetRank[i] * MyWordAlphabetWeight[i];
            }
            //get root from Product
            RootProduct[0, 0] = 0;
            RootProduct[1, 0] = 1;
            RootProduct[2, 0] = 2;
            RootProduct[0, 1] = MyWordAlphabetProduct[0];
```

```
RootProduct[1, 1] = MyWordAlphabetProduct[1];
            RootProduct[2, 1] = MyWordAlphabetProduct[2];
            for (int i = 3; i < word.Length; i++)
            {
                if (MyWordAlphabetProduct[i] < RootProduct[0, 1] || MyWordAlphabetProduct[i] <
RootProduct[1, 1] || MyWordAlphabetProduct[i] < RootProduct[2, 1])
                {
                    if ((RootProduct[0, 1] > RootProduct[1, 1] && RootProduct[0, 1] >
RootProduct[2, 1])/* && RootProduct[0, 1] < RootProduct[1, 1] && RootProduct[0, 1] <
RootProduct[2, 1]*/)
                    {
                        RootProduct[0, 0] = i;
                        RootProduct[0, 1] = MyWordAlphabetProduct[i];
                    }
                    else if ((RootProduct[1, 1] > RootProduct[0, 1] && RootProduct[1, 1] >
RootProduct[2, 1]))
                    {
                        RootProduct[1, 0] = i;
                        RootProduct[1, 1] = MyWordAlphabetProduct[i];
                    }
                    else //if (RootProduct[2, 1] > RootProduct[0, 1] && RootProduct[2, 1] >
RootProduct[1, 1])
                    {
                        RootProduct[2, 0] = i;
                        RootProduct[2, 1] = MyWordAlphabetProduct[i];
                    }
                }
            }
            // Sort Root

            double Temp1;
            double Temp2;
            for (int i = 0; i < 3; i++)
            {
                if (RootProduct[0, 0] > RootProduct[1, 0])
                {
                    Temp1 = RootProduct[1, 0];
                    Temp2 = RootProduct[1, 1];
                    RootProduct[1, 0] = RootProduct[0, 0];
                    RootProduct[1, 1] = RootProduct[0, 1];
                    RootProduct[0, 0] = Temp1;
                    RootProduct[0, 1] = Temp2;
                }
                if (RootProduct[1, 0] > RootProduct[2, 0])
                {
                    Temp1 = RootProduct[2, 0];
                    Temp2 = RootProduct[2, 1];
                    RootProduct[2, 0] = RootProduct[1, 0];
                    RootProduct[2, 1] = RootProduct[1, 1];
                    RootProduct[1, 0] = Temp1;
                    RootProduct[1, 1] = Temp2;
                }
            }

            string MyRoot = MyWordAlphabet[Convert.ToInt32(RootProduct[0, 0])] +
MyWordAlphabet[Convert.ToInt32(RootProduct[1, 0])] +
MyWordAlphabet[Convert.ToInt32(RootProduct[2, 0])];
            //prnt
            Root = Root + "{";
            for (int i = 0; i < MyWordAlphabet.Length; i++)
            {
                Root = Root + "\n";
                Root = Root + " " + MyWordAlphabet[i] + " = " +
MyWordAlphabetWeight[i].ToString() + " * " + MyWordAlphabetRank[i] + " = " +
MyWordAlphabetProduct[i] + ",";

            }
            Root = Root + "}";
            Root = Root + "\n" + MyRoot;
            return MyRoot;
        }
```

```
// Get Weight for one Alphabet from "AlphabetWeight" array
double GetAlphabetWeight(string alphabet)
{
    int index = -1;
    for (int i = 0; i < this.Alphabet.Length; i++)
    {
        if (alphabet.Equals(this.Alphabet[i]))
        {
            index = i;
        }
    }
    if (index != -1)
    {
        return this.AlphabetWeight[index];
    }
    else
    {
        return 0;
    }
}
public string GetAllHashFromTweet(string str)
{
    string Hashs = " ";
    for (int i = 0; i < str.Length; i++)
    {
        if (str[i] == '#')
        {
            for (int j = i; j < str.Length; j++)
            {
                if (str[j] == ' ')
                {
                    Hashs = Hashs + str.Substring(i, j - i) + " ";
                    i = j;
                }
                if (j == str.Length - 1)
                {
                    Hashs = Hashs + str.Substring(i, j - i + 1) + " ";
                    i = j;
                }
            }
        }
    }
    return Hashs;
}
public List<string> GetAllHashFromTweetAsList(string str)
{
    List<string> Hashs = new List<string>();
    for (int i = 0; i < str.Length; i++)
    {
        if (str[i] == '#')
        {
            for (int j = i; j < str.Length; j++)
            {
                if (str[j] == ' ')
                {
                    Hashs.Add(this.stopwordsRemove(str.Substring(i, j - i)));
                    i = j;
                }
                if (j == str.Length - 1)
                {
                    Hashs.Add(this.stopwordsRemove(str.Substring(i, j - i + 1)));
                    i = j;
                }
            }
        }
    }
    return Hashs;
}
```

B.2 Classification and measuring the impact

```
// Classification Tweets And get impact
            for (int indexTweets = 0; indexTweets < lstTempTweet.Count; indexTweets++)
            {
                if (lstTempTweet[indexTweets].Length < 20)
                {
                    continue;
                }

                // Get Tweets one by one
                string Tweet = lstTempTweet[indexTweets];
                // Reset "TotalWeightForTweet"
                TotalWeightForTweet = new List<double>();
                // Clean Hash Tags From '#' Or '_'
                string HashsOfTweet = myClean.GetAllHashFromTweet(Tweet);
                // Double Hash Tags Weight
                Tweet = Tweet + " " + HashsOfTweet;


                //Reset TotalWeightForTweet
                for (int i = 0; i < CategorysID.Count; i++)
                {
                    TotalWeightForTweet.Add(0);
                }
                //classificationOfOneTweetConst = classificationOfOneTweet;
                //Remove Stop Ward For Tweet
                Tweet = myClean.stopwordsRemove(Tweet);
                // Split Tweet To Words
                string[] WordsOfTweet = Tweet.Split(' ');
                // Get Total Weight For Tweet
                foreach (string Word in WordsOfTweet)
                {
                    if (Word.Length > 2)
                    {
                        // Get Word Categorys by Root of Word
                        WeightForWord = MyDBConnection.GetWordCategorys(myClean.GetRoot(Word),
CategorysID);
                        for (int i = 0; i < CategorysID.Count; i++)
                        {
                            TotalWeightForTweet[i] = TotalWeightForTweet[i] + WeightForWord[i];
                        }
                    }
                }
                //Reset biggest in "TotalWeightForTweet"
                biggest = 0;
                // Get biggest
                for (int i = 0; i < TotalWeightForTweet.Count; i++)
                {
                    if (TotalWeightForTweet[i] > biggest)
                    {
                        biggest = TotalWeightForTweet[i];
                    }
                }
                // Add One in Categorys if Equals biggest
                for (int i = 0; i < classificationOfTweets.Length; i++)
                {
                    if (TotalWeightForTweet[i] == biggest)
                    {
                        classificationOfTweets[i] = classificationOfTweets[i] + 1;
                        NumberOfTweetsForCategory[i] = NumberOfTweetsForCategory[i] + 1;
                        NumberOfRetweetsForCategory[i] =  NumberOfRetweetsForCategory[i] +
lstTempRTCount[indexTweets];
                    }
                }


            }


WeightForCategory[i] = NumberOfRetweetsForCategory[i] / NumberOfTweetsForCategory[i];
```

```
public List<double> GetWordCategorys(string word, List<int> CategorysID)
     {
         List<double> WordCategorys = new List<double>();
         List<int> MyCategorysID = CategorysID;
         MyWeightWordDataTable = MyWeightWordTableAdapter.GetDataByWard(word);
         int CategoryID;
         int WordWeight;
         //Reset WordCategorys
         for (int i = 0; i < MyCategorysID.Count; i++)
         {
             WordCategorys.Add(0);
         }

         foreach (DataSet1.WeightWordRow row in MyWeightWordDataTable)
         {
             CategoryID = row.CategoryID;
             WordWeight = row.Weight;
             for (int i = 0; i < MyCategorysID.Count; i++)
             {
                 if (CategorysID[i] == CategoryID)
                 {
                     int n = 1;
                     if (row.Counter < firstLevel)
                     {
                         WordCategorys[i] = 3 + (n * (1 - ((row.Counter/1) /100)));
                     }
                     else if (row.Counter < secondLevel)
                     {
                         WordCategorys[i] = 2 + (n * (1 - ((row.Counter / 2) / 100)));
                     }
                     else if (row.Counter < thirdLevel)
                     {
                         WordCategorys[i] = 1 + (n * (1 - ((row.Counter / 3) / 100)));
                     }
                 }
             }
         }
         return WordCategorys;
     }
```