Chapter 3: Twitter Terrorism Detection Framework

In this chapter we describe the architecture of Twitter Terrorism Detection Framework. There are mainly three sections in this chapter. First section 3.1 is about the system architecture of the Twitter Terrorism Detection Framework where different modules of the architecture and relationship among them are described briefly. Among the main modules of the architecture like Twitter data crawling module, storage module, tweet analysis module, output module. In section 3.2 we discuss about the analytical representation of our system which gives the details of the developed system with different algorithms, flowcharts and tables required for analysis. Section 3.3 is about the complexity analysis of Twitter Terrorism Detection Framework.

3.1 System Architecture

The system architecture of Twitter Terrorism Detection Framework comprises five basic modules: Twitter data crawler module, storage module, tweet classification module, output module and training module. The function of twitter data crawler module is to crawl real-time tweets from twitter using Twitter Streaming API. Storage module stores the tweets temporarily. Tweet classification module predict the category of the tweet. The output module shows the output of the system. The training module builds the classification model which is used to predict the category of each tweet. The architecture is shown in the following figure (fig 3.1).

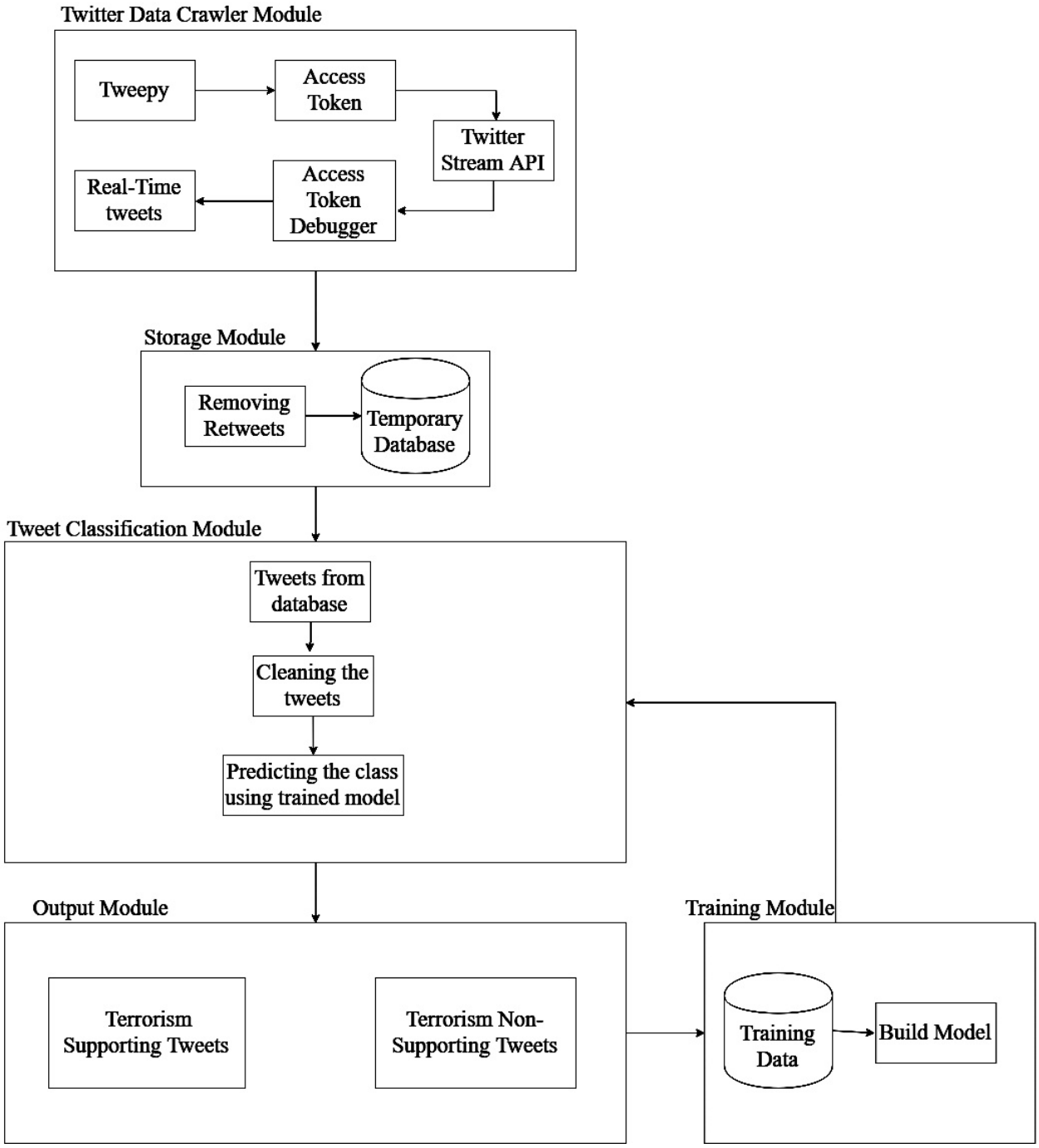

Figure 3.1: Twitter Terrorism Detection Framework Architecture

3.1.1 Twitter Data Crawler Module

First module is *Twitter Data Crawler Module*. This module is used for fetching the real-time tweets from Twitter. First, we need 4 API key from the twitter developer console. The 4 keys are:

1. Access Token
2. Access Secret Key
3. Consumer Token
4. Consumer Secret Key

These 4 key are needed to get access to Twitter API. Using these 4 key we can setup a twitter real-time tweet listener. This listener will allow us to collect real-time tweets from the twitter.

As we have used python to build our framework, we have used a python library that helped us getting access to the Twitter streaming API. We have used a python library 'Tweepy' to access Twitter streaming API.[i]

While collecting real-time tweets we have made sure to handle all kinds of error that would break the connection. If the connection breaks, the crawler module will stop working. That's why we have checked for several exception such as database exception, limit exceeded exception etc.

3.1.2 Storage Module

In this module we have stored the tweet that are coming from the twitter Streaming API. The Tweets are stored here temporarily. In the storage module there is facility to store last 1 million tweets for further analysis. Also tweets after categorization is stored in the storage module to show them in our user interface. Before storing the tweets into the database we first removed the retweets from the tweet collection. Because if we keep retweets the database will be heavy as the original tweet is already stored and the retweet would only be a duplicate. Because of this we have removed all the retweets from the database by checking the retweet symbol (RT).

3.1.3 Classification Module

In the classification module we have retrieved the real-time tweet from the database and then used our classifier model to predict the category of each tweets. According to the prediction the tweet is stored in different table of the database. If the tweet is terrorism supporting then the tweet is stored in a table named *matched_tweets*. Otherwise the tweet is stored in a table called *all_tweets*.

3.1.4 Output Module

The output module is a user interface, which is a web application. It shows the categorized output of the real-time tweets which is dynamically updated. Also it shows the tweets that are marked as terrorism supporting tweets.

3.1.5 Training Module

This module is used to train the classification model which is generated by using the training dataset and learning algorithms.

3.1.5.1 Dataset

In this study we have used a dataset which we have collected via twitter streaming API and tweets from some specific profile.

The dataset that we have used are stored in the form of a comma-separated values files with tweets and their corresponding category. Category is defined by 0 or 1 or 2.

The dataset is a mixture of words, emoticons, symbols, URLs and references to people.

First category is terrorism supporting tweets. Which is marked by 0 in the data set. These are the tweets which supports terrorism, spreads fear among general people or spreads their views by brain washing people. Figure 3. Shows a tweet that falls in this category.

Figure 3.: A terrorism supporting tweet

Second Category is terrorism related but non-supporting tweets. This category is marked by 1 in the dataset. This category contain tweets that are related to terrorism but not supporting it or against it. Figure 3. shows a tweet that falls in this category

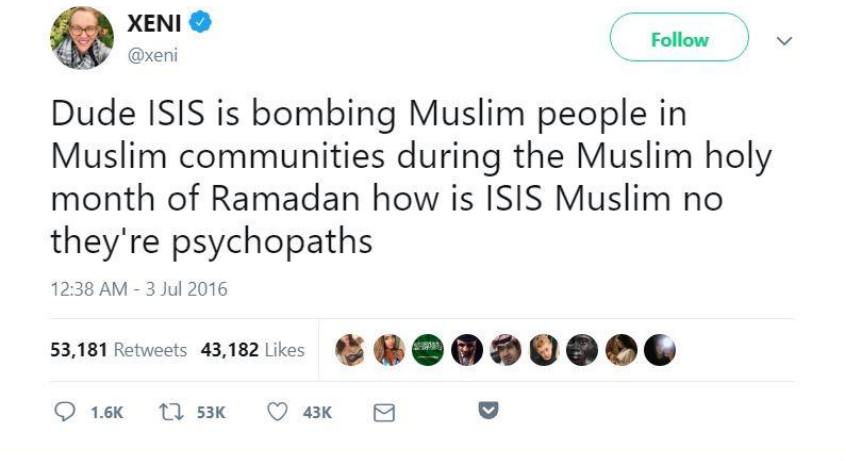

Figure: A terrorism non-supporting tweet

Second Category is random tweets. This category is marked by 2 in the dataset. This category contain tweets that are randomly posted by the users. Figure 3. shows a tweet that falls in this category.

Figure 3.: A random tweet

Figure 3. shows the head of our dataset, which is stored in a CSV file.

tweets	target
#IS Official Media claims that #ISIS Militants have Destroyed 3 Turkish Army Tanks while backing rebels in Nibra... https://twitter.com/Nidalgazaui/status/725675000213200898/photo/1	1
RT @hrw: #ISIS rule in Iraq marked by summary executions, " disappearances " , torture https://www.hrw.org/news/2016/07/10/iraq-isis-rule-marked-executions-cruelty pic.twitter.com/O5hiiMJ0Fn	1
correction my 1st day to sleep in @FevEligante	2
RT @BoArhama: #ISIS criminal groups used to target Muslims in the mosques ! #ISISAttackingMuslims pic.twitter.com/PKaAhY4QSP	1
RT @SOFREP: Man charged with marking targets in DC for IslamicÃ‚Â State https://sofrep.com/58875/man-charged-marking-targets-dc-islamic-state/ pic.twitter.com/BhyUQtZyUz	1
U.S. Will Deploy 560 Troops to Iraq to Help Retake Mosul From ISIS http://www.nytimes.com/2016/07/12/world/middleeast/us-iraq-mosul.html	1
We should get revenge of our brothers who are killed in the battlefiled!! Be ready US army! #ISIS #Revenge #Mujahideen #Jihad	0
@13Christina well tomorrow i'm gonna hopefully be getting good news Jeep-wise. I gotta get up early and go for a run. Didn't swim 2day.	2
#ISIS soldiers managed to kill a colonel in the apostate Afghan army in the city of KÃ„Âbul by detonating an explosive device against him. Alhamdulillah!	0

Figure: A peek into our dataset

3.1.5.2 Data Pre-Processing

As we crawled raw tweets from the twitter, it contains some noises. Noises in tweets are natural, because people express their feelings informally in the social media. In Twitter, tweets contains some special component such as retweet, mentions, hash-tags, URLs, emoticons etc. which have to be suitably extracted. Therefore, to train our classifier using the datasets, the raw tweets must be cleaned before applying the classification algorithms. To get the best out of our dataset, we applied a number of data cleaning processes. At first some general cleaning are done, such as:

- Every tweet is first converted into lowercase format.
- Two or more spaces are replaced with a single space
- Quotes (" and ’), extra dots (.) and spaces are stripped from the ends of tweet.
- Replace 2 or more spaces with a single space.

To handle the special component of a tweet, we have done the following pre-processing tasks:

I. ***URL:*** Twitter users often shares URL in their tweets. In our training, any particular URL doesn’t contain any special feature and if we kept the URLs in the tweet, that would have been leaded to sparse feature. Therefore, we remove all the URL from the tweets. To match the URLs we have used this regular expression *((www\.[\S]+)|(https?://[\S]+)).*

II. ***User Mention:*** A handle is associated with each twitter user account. Twitter supports mentioning other users in a tweet. A user can mention other user by putting an ‘at’ symbol (@) before the handle, i.e. @handle_of_the_user. User mentions are irrelevant in our classifier, so we remove the user mentions from the tweet. To remove the user mentions, we have used this regular expression *@[\S]+*.

III. ***Hashtag:*** To mention a trending topic on twitter, a special component called Hashtags are used. Hashtags don’t contain any space and starts with a hash symbol (#). In twitter, Hashtags are frequently used by users. We removed the hash symbol from the hashtags. For example, *#isis* is replaced by *isis*. To match the hashtags we have used this regular expression *#(\S+).*

IV. ***Retweet:*** The tweets that are updated by one user and shared by other users these tweets are known as retweets. In every retweet, there is a symbol RT in the beginning of the tweet. As RT is not an important feature in our system, we have removed the word RT from the tweets. The regular expression used to match retweets is \brt\b.

After Tweet level cleaning is done, individual words of the tweets are processed as follows:

- Any punctuation [’"?!,.():;] from the word is stripped
- Words with three or more letter repetitions are converted to two letters. Some people update their tweets like I *am happppppy* which adds multiple characters on a certain words. Tweets containing this type of words are handled by converting the word *happppppy* to *happy*.
- To handle the words like *sugar-free* and *our’s*, we have removed - and ’. This type of words are converted into a more general form like *sugarfree* and *ours*.
- Then we checked for valid word by checking successive alphabets, if it is not valid then we have stripped them.

V. ***Stop word:*** There are some words which do not make any significant change in absence of them. Those words are called stop word. Our current step is to remove those words from the document. There are a lot of stop words in English language such as: *him, about, ours, those, me, few, how, being, off, again, yourselves, its, once, below, any, yourself, is, from, do, can, until, all, hers, our, just, further, then, above, into, theirs, in, i, who, for, more, each, doing, with, against, o, of, during, as, there, some, are, while, and, only, if, where, were, so, having, these, before, myself, under, very* etc.

VI. ***Contracted Word Handling:*** Users often sends tweets containing words in contracted form. Like *are not* is written as *aren’t, I am* is written as *I’m* etc. We converted the contracted word to their long form. A list of contracted word and their long form are given in Table 3.1:

Table 3.1: Contracted word and long form[ii]

Contracted form	Long form	Contracted form	Long Form
Aren’t	Are not	I am	I’m
Isn’t	Is not	Weren’t	Were not
Haven’t	Have not	Hasn’t	Has not
Hadn’t	Had not	Won’t	Will not
Don’t	Do not	Doesn’t	Does not
I’d	I had	I’ll	I shall
They’ll	They will	He’ll	He will
She’ll	She will	Can’t	Can not
Mustn’t	Must not	Shouldn’t	Should not
Couldn’t	Could not	Wouldn’t	Wouldn’t

VII. ***Tokenization:*** Given a character sequence and a defined document unit, tokenization is the task of chopping it up into pieces, called *tokens*, perhaps at the same time throwing away certain characters, such as punctuation. Here is an example of tokenization:

Input : We are going to USA! I am so happy today!

Output : We, are, going, to, USA, I, am, so, happy, today

These tokens are often loosely referred to as terms or words, but it is sometimes important to make a type/token distinction. A *token* is an instance of a sequence of characters in some particular document that are grouped together as a useful semantic unit for processing. A *type* is the class of all tokens containing the same character sequence. A *term* is a (perhaps normalized) type that is included in the IR system's dictionary.[iii]

These tokens are further used to make feature vector.

VIII. ***Stemming:*** Stemming is a technique to remove affixes from a word, ending up with the stem. For example, the stem of killing is kill. Stemming is most commonly used by search engines for indexing words. Instead of storing all forms of a word, a search engine can store only the stems, greatly reducing the size of index while increasing retrieval accuracy.[iv] Similarly we have applied stemming in our dataset to reduce the number of words and made our feature vector size small.

Some tweets from our training dataset in their raw form and pre-processed form are shown in table 3.2:

Table 3.2: Tweets in raw form and processed form

Raw Form	**Processed Form**
RT @NihadAwad: #ISIS is attacking the holy city of #Medina ISIS is attacking mainly Muslims ISIS is being fought mainly by Muslims #StopISIS	isis is attack holy city medina isis attack main muslims isis fight main muslims stopisis
RT @TexasCruzn: Moderate Muslims Offended By burning ISIS Flag http://ow.ly/rOuz5023UmB	moderate muslims offend burn isis flag
@missy: good for you. i can’t drink just water.	good drink just water

@Flik just filter out the follows	just filter out follow
@skp time to die. Let's jump off the top of a hill , life is pointless if @gulpanang is following @vishurao	time die let us jump top hill life point follow

3.1.5.3 Feature Vector

In this study we have used style based features of text. As we are using tweets in our analysis we could also use time based features.[v] But due to our limitation of the collecting data, we only collected data in a limited time frame. So, our tweets time is limited in a time-frame. That's why we could not use time based features.

Style based features depends on the variations of literary style between different writers. Usually it includes statistics about the frequency of specific items or the length of words or sentences.

A common style based analysis method is called writer invariant or author invariant. It claims that all texts written by the same author are similar or invariant. In other words texts written by the same author will be more similar than those written by different ones. Even though we are not interested in the authors of tweets, style based features is an important analysis method, since the topic all terrorist authors write about is similar and the purpose of the messages is the same, mainly to spread their terrorist propaganda, it is reasonable to believe that the style of writing they have might be similar. We used frequency of words, presence of words, count of words and use of punctuations as style based features.

We have used TF*IDF to represent our features. TF*IDF is an information retrieval technique that weighs a term's frequency (TF) and its inverse document frequency (IDF). Each word or term has its respective TF and IDF score. The product of the TF and IDF scores of a term is called the TF*IDF weight of that term. It shows the relative importance of a feature in a document or text.[vi]

3.2 Flow Graph of Twitter Terrorism Detection Framework

3.3 Analytical Representation of the Architecture

This section gives an analytical description of the system architecture given in previous sections. The system architecture above illustrates the internal and external structure of system modules integrated together in one package to form one system. The following subsections provide a brief background overview of the tools used and the implementation details of the different modules of the developed system starting from the back-end to the front-end. The whole system was developed Windows operating system, PhpStorm IDE and PyCharm IDE platforms.

3.3.1 Crawling Data from Twitter

We have crawled real-time tweets from the twitter by using twitter streaming API. To do this we have provided four keys that we have collected from the twitter developer website.

Algorithm 3.1: Crawl Real-Time Tweets

Input: Developer Access keys

Require: Real-Time tweets streaming from twitter

1. **Begin**
2. **Call** Twitter API
3. **Call** Twitter Streaming API
4. **Set** accessTkn = “”
5. **Set** accessTknSec = “”
6. **Set** consumerKey = “”
7. **Set** consumerSec = “”
8. tweets = STREAM-LISTENER(accessKeys)
9. **Create** a table named all_tweets having the field username, tweet, tweet_id, type
9. **if** tweets != null **then**
10. **if** tweets[‘retweet’] = False **then**

10. **Insert** tweets[‘text’] , tweets[‘username’] and tweets[‘tweet_id’] into database
11. **End**

3.3.2 Cleaning Crawled Data

After data is crawled from the twitter the tweet are in raw form. We can’t use these tweets to classify or train. So, we have cleaned the tweet before using them in classification or training. We have performed several cleaning like removing url, user mentions, removing hash (#) from hashtags, removing retweets, removing stop words etc.

Algorithm 3.2: Cleaning raw tweets

Input: raw tweets

Require: clean the raw tweets

1. **Begin**
2. remove url from raw tweets
3. remove hash (#) symbol of hashtags from raw tweets
4. remove user mentions form raw tweets
5. remove retweet symbol RT from raw tweets
6. convert the raw tweets into lowercase form
7. search for contracted from in tweets
8. **if** contracted form found **then**
9. replace it with long form
10. search for stop words in tweets
11. **if** stop words found **then**
12. remove the stop words
13. tokenize the tweets
14. apply stemming on the tweets
15. **End**

3.3.3 Building Model

To predict the class of the tweet we needed a mathematical model which can classify the tweets based on their features. We have used three classification algorithm. These are SVM (Support Vector Machine), Logistic Regression and AdaBoost. The below subsection

3.3.3.1 SVM

Algorithm 3.3: SVM learning algorithm

Inputs: Training data

Require: Train model to classify

1. **Begin**
2. Find closest pair from opposite class from training data
2. **while** there are violating points in the closest pair **do**
3. Find a violator
4. make union of the closest pair set and violator
5. **if** any $\alpha(i) < 0$ due to the union **then**
6. divide the set of closest pair by i
7. repeat till all such points are pruned
8. **end if**
9. **end while**
10. **End**

3.3.3.2 Logistic Regression

Algorithm 3.4: Logistic Regression learning algorithm

Inputs: Training data, *x*
Require: Train model to classify
1. Begin
2. Initialize w
3. **for** i=1 to n **do**
4. $z(i)=\sum w(i)*x(i)$
5. **end for**
6. **for** j = 0 to d **do**
8. **for** i = 1 to n **do**
9. θ (j) = SOFT-MAX(z(i))

10. **end for**
11. **end for**
12. **End**

3.3.3.3 AdaBoost

The ADABOOST variant of the boosting method for ensemble learning. The algorithm generates hypothesis by successively reweighting the training examples. The function WEIGHTED-MAJORITY generates a hypothesis that returns the output value with the highest vote from the hypotheses in h, with votes weighted by z.

Algorithm 3.5: AdaBoost learning algorithm

Inputs: examples, *K* no of hypotheses

Require: Train model to classify

1. **Begin**
2. **Set** w = 1/N
3. **Set** h = Vector of K hypothesis
4. **Set** z = Vector of K hypothesis weights
5. **for** $k = 1$ **to** K **do**
6. **h**[k] ← *LEARNING*(*examples*, **w**)
7. *error* ← 0
8. **for** $j = 1$ **to** N **do**
9. **if h**[k](x_j) ≠ y_j **then** *error* ← *error* + **w**[j]
10. **for** $j = 1$ **to** N **do**
11. **if h**[k](x_j) = y_j **then w**[j] ← **w**[j] · *error* / (1 − *error*)
12. **w** ← NORMALIZE(**w**)
13. **Z**[k] ← log(1 − *error*) / *error*
14. WEIGHTED-MAJORITY(**h**, **z**)
15. **End**

3.3.4 Collecting Tweet Data

We have collected our dataset from two sources, one is by searching some particular words and another is crawling tweets from a user's profile. In searching tweets, we supplied a list of words. If one or more words from the set contains in a tweet, the tweet is stored in the database. In crawling from user's profile, we supplied a username and all the tweets from the profile are collected and stored in the database.

Algorithm 3.6: Collect tweets containing any of the word from a list

Inputs: Developer Access keys and wordList

Require: Write tweets into CSV file

1. **Begin**
2. **Call** Twitter API
3. **Call** Twitter Streaming API
4. **Set** accessTkn = “”
5. **Set** accessTknSec = “”
6. **Set** consumerKey = “”
7. **Set** consumerSec = “”
8. tweets = STREAM-LISTENER(wordlist, accessKeys)
9. create and open a CSV file having column name tweets and type
10. **if** tweets != null **then**
11. **if** tweets[‘retweet’] = False **then**
12. **Write** tweets[‘text’] into the CSV file
13. **End**

Algorithm 3.7: Collecting tweets form a profile

Inputs: Developer Access keys and Username

Require: Write tweets into CSV file

1. **Begin**
2. **Call** Twitter API
3. **Set** accessTkn = “”
4. **Set** accessTknSec = “”
5. **Set** consumerKey = “”
6. **Set** consumerSec = “”
7. tweets = STREAM-LISTENER(username, accessKeys)

8. create and open a CSV file having column name tweets and type
9. **if** tweets != null **then**
10. **Write** tweets[‘text’] into the CSV file
11. **End**

3.3.4 Generating Output

By using the model that we have built in the previous steps, we can classify a tweet. The classification result is 0 or 1 or 2. According to this result we can show the type of the tweets.

Algorithm 3.8: Classification of real-time tweets

Inputs: model file

Require: Classification of the tweets

1. **Begin**
2. *classifier* = load(*model*)
3. **for** each tweets in the all_tweets table in database **do**
4. clean the tweets
5. *type* = *classifier*.predict(clean_tweets)
6. **if** *type* = 0 **then**
7. *result* = “Terrorism Supporting”
8. **else if** *type* = 1 **then**
9. *result* = “Terrorism Non-Supporting”
10. **else if** *type* = 2 **then**
11. *result* = “Random”
12. **show the *result***
11. **End**

3.4 Complexity Analysis

Our system has five parts keeping the issue of time in concern. The time complexity of our system may be described as follows:

Complexity of Data Storing

Let, n is the number of tweets
Q is the time required to crawl and store a tweet in the database
So time require to store all the tweets in the database is *O(nQ)*.

Complexity of Tweet Cleaning

Let, n is the number of letter in a tweet
m is the number of tweets in the dataset
So time require to clean all the tweets are *O(nm)*.

Complexity of Predicting a tweet

Let, n is the number of word in a tweet
C is the number of words in the feature vector
So time require to clean all the tweets are *O(nC)*.

[i] Link of Tweepy

[ii] https://www.englisch-hilfen.de/en/grammar/kurzformen.htm

[iii] https://nlp.stanford.edu/IR-book/html/htmledition/tokenization-1.html

[iv] https://www.packtpub.com/mapt/book/big_data_and_business_intelligence/9781787285101/12/ch02lvl1sec019/stemming-words

[v] Write name of the twitter feature paper

[vi] https://www.elephate.com/blog/what-is-tf-idf/