Dissertation on

"Detection of Events and Emerging Themes from Social Media Streams "

Submitted in partial fulfillment of the requirements for the award of degree of

Bachelor of Technology

in

Computer Science & Engineering

Submitted by:

Varsha R	**<01FB15ECS337>**
Vaishnavi Rao	**<01FB15ECS334>**

Under the guidance of

Internal Guide

Prof. C O Prakash

Assistant Professor,

PES University

January – May 2019

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

FACULTY OF ENGINEERING

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)

100ft Ring Road, Bengaluru – 560 085, Karnataka, India

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)

100ft Ring Road, Bengaluru – 560 085, Karnataka, India

FACULTY OF ENGINEERING

CERTIFICATE

This is to certify that the dissertation entitled

'Detection of Events and Emerging Themes from Social Media Streams '

is a bonafide work carried out by

Varsha R	**<01FB15ECS337>**
Vaishnavi Rao	**<01FB15ECS334>**

In partial fulfilment for the completion of eighth semester project work in the Program of Study Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period Jan. 2019 – May. 2019. It is certified that all corrections / suggestions indicated for internal assessment have been incorporated in the report. The dissertation has been approved as it satisfies the 8th semester academic requirements in respect of project work.

Signature	Signature	Signature
Prof. C O Prakash	**Dr. Shylaja S S**	**Dr. B K Keshavan**
Assistant Professor	Chairperson	Dean of Faculty

External Viva

Name of the Examiners	**Signature with Date**
1. ______________________________	______________________________
2. ______________________________	______________________________

Acknowledgement

We would like to express my gratitude to Prof. C O Prakash, Dept. of Computer Science, PES University, for his continuous guidance, assistance and encouragement throughout the development of this project. We are grateful to the project coordinators, Prof. Preet Kanwal and Prof. Sangeeta ma'am for organizing, managing and helping out with the entire process. We want to take this opportunity to thank Dr. Shylaja S S, Chairperson, Department of Computer Science and Engineering, PES University, for all the knowledge and support we have received from the department. We would like to thank Dr. B.K. Keshavan, Dean of Faculty, PES University for his help. We are deeply grateful to Dr. M. R. Doreswamy, Chancellor, PES University, Prof. Jawahar Doreswamy, Pro Chancellor – PES University, Dr. K.N.B. Murthy, Vice-Chancellor, PES University for providing to me various opportunities and enlightenment every step of the way.

Finally, this project could not have been completed without the continual support and encouragement we have received from our parents and friends.

Abstract

The micro-blogging platform, Twitter, has opened people up to an entirely new world of information-gathering and news-sharing, including but not limited to detailing everyday events and activities of the users, the “Tweeple”. Twitter, then, becomes an exhaustive source of data for analysis as well as the first place people are increasingly turning to for daily alerts or updates, in the face of news organisations and other traditional forms of media people looked to earlier. A major challenge facing event detection using social media streams is to extract useful real-world events from the mundane and polluted text (abbreviated words, spelling and grammatical errors and text written in mixed languages). We intend to build a model that generates the trending topics of the day after being fed the tweets of the day. We want to figure out the best algorithm to use by testing it on various machine learning and data science models and comparing accuracies (eg:- naive bayes, RNN, Random Forest, SVM, etc). The solution proposed aims at a far-reaching collection and display of events detected in the news in a way that makes an everyday user’s feed more comprehensive and it will adhere to customization of tweets displayed on the feed based on the trending topics and to the importance of the trending topic and focus on topics by analysing the sentiment. Eg: Earthquake warning will be trending over a celebrity wedding

TABLE OF CONTENTS:

1. Introduction

The micro-blogging platform, Twitter, has opened people up to an entirely new world of information-gathering and news-sharing, including but not limited to detailing everyday events and activities of the users, the “Tweeple”. Twitter, then, becomes an exhaustive source of data for analysis as well as the first place people are increasingly turning to for daily alerts or updates, in the face of news organisations and other traditional forms of media people looked to earlier. A major challenge facing event detection using social media streams is to extract useful real-world events from the mundane and polluted text (abbreviated words, spelling and grammatical errors and text written in mixed languages). [3] Event detection requires the automatic answering of what, when, where, and by whom. After reporting on the most recent efforts in the area, it is clear that no method addressed all of these questions. [5] Investigating how to model the social streams together with other data sources, like news streams to better detect and represent events was another complication faced earlier. [6] A considerable effort is still required to achieve efficient, scalable and reliable systems for event detection, summarization, tracking and association. [7]

1.1 Scope

The project will depend on the efficient functioning of Twitter minus any crashes on any given day to be able to conduct real time analysis smoothly in the long run. A proper, working internet connection is required that keeps the end user connected with their surroundings in order to access Twitter easily. Additional limitations include false positives generated by the model and disinformation provided by people on Twitter. Twitter users are assumed to be sensors, who make observations, which are their actual tweets to detect a target event. The tweets are given a time values and a location as well. The detection of these events and other emerging topics is restricted to the Indian subcontinent.

2. Problem Statement

The haphazard nature of tweets can make a timeline quite messy. Twitter also happens to be one of the most important sources of news for people where they hear or read about a particular event first.
In the face of this, we propose a solution where the user's timeline also reflects trending topics based on national emergency and other filters among other trending topics.
This not only pushes events of national importance to the top of the user's timeline, it also filters out certain other trending topics that are given importance by media organisations in which the user might not be all that interested.

3. Literature Survey

3.1 Catching the Long-Tail: Extracting Local News Events from Twitter [1]

To detect the messages that report occurrence of an event, they have used a two step process. In the first step they reject tweets that follow a specific pattern using regular expressions, and the second step is supervised classification and boosting.
For detection of relevant tweets, the approach using supervised classification of individual tweets is able to catch the sparsely reported events in the 'long-tail'.
Improvised standard NLP techniques are also used so that they work on the informal language often used in Twitter

3.2 Topic Extraction from News Archive Using TF*PDF Algorithm [2]

Uses the novel TF*PDF algorithm to recognize the terms that try to explain the main topics. These would be the terms that appear frequently in many documents from many newswire sources concurrently. TF*PDF algorithm is designed in a way that it would assign heavy term weight to these kind of terms and thus reveal the main topics. TF*PDF algorithm performs well in extracting the terms explaining the main topics, by taking advantage on the concept that whenever there is a hot topic on air, the terms that explain the hot topics will appear frequently in many documents from multiple newswire sources.

3.3 A SURVEY OF TECHNIQUES FOR EVENT DETECTION IN TWITTER [3]

This paper discusses event detection techniques unique to Twitter data and classifies it according to event type, detection task, and detection method. It also includes a brief description of event detection as applicable to traditional media outlets in which some of them have been adapted to suit Twitter data. This is further classified into detection based on document or temporal features. For Twitter data specifically, unsupervised and supervised detection approaches have been elaborated upon, along with an explanation of sub-classification based on event type and detection methods.

3.4 Sentiment-Based Event Detection in Twitter [4]

This article aims to determine whether sentiment analysis can be used as a method to detect significant events occurring around the world.
Most methods for event detection concentrate solely on the increasing frequency of terms used in social media in event detection.
The results, the paper observes, tell us that sentiment-based solutions produce comparable performance with frequency-based approaches and are more effective in detecting them within a 1-day period.

A major challenge facing event detection using social media streams is to extract useful real-world events from the mundane and polluted text (abbreviated words, spelling and grammatical errors and text written in mixed languages). [3]
Event detection requires the automatic answering of what, when, where, and by whom. After reporting on the most recent efforts in the area, it is clear that no method addressed all of these questions. [5]

Another challenge is to investigate how to model the social streams together with other data sources, like news streams to better detect and represent events. [6]
A considerable effort is still required to achieve efficient, scalable and reliable systems for event detection, summarization, tracking and association. [7]
There is a growing need to zero in on a public benchmark to evaluate the performance of different detection approaches. [3]
Furthermore, there is a deficit of publicly available testbeds for objective comparison of different methods utilised. [3]

3.5 Tweet Analysis for Real-Time Event Detection and Earthquake Reporting System Development Ekta, Paahuni Khandelwal, Priya Bundela, Richa Dewan

This paper attempts to do an analysis/monitoring of tweets in order to be able to detect earthquakes in real time.
The approach taken is one of the classification of tweets by putting them through a Naive Bayes classifier, based on features as keywords, sentiment, number of words, and context. Post this classification, a probabilistic value is associated with the tweets under consideration which signifies the occurrence or non-occurrence of the earthquake.
The study hopes to further the results obtained to create an application to alert users of the disaster based on their geo-location.

4. Customer requirements specification

4.1 Product Perspective

- This project is dependent on Twitter and its users. It is not functionable without Twitter's huge fanbase and customers.
- Twitter in general has the following features :-

- o A home page where there are a bunch of tweets in textual format displayed to a Twitter user with a valid Twitter account.
- o This home page has tweets from people the user follows.
- o There is another section of each user's account with general trending tweets decided by Twitter's algorithm based on the most used hashtags and most liked/shared tweets.
- o There is also a user profile that displays activities of the user, the user's tweets and retweets.
- o Identifying the principal external interfaces of this software product.

- Python and a simple laptop with 8GB RAM is used for development
- Python, Bootstrap and a simple laptop with 8GB RAM is used for deployment

4.2 User Characteristics

The haphazard nature of tweets can make a timeline quite messy. Twitter also happens to be one of the most important sources of news for people where they hear or read about a particular event first. In the face of this, we propose a solution where the user's timeline also reflects trending topics based on national emergency and other filters among other trending topics. This not only pushes events of national importance to the top of the user's timeline, it also filters out certain other trending topics that are given importance by media organisations in which the user might not be all that interested.

4.3 General Constraints, Assumptions and Dependencies

Dependencies:

- The project will depend on the efficient functioning of Twitter minus any crashes on any given day to be able to conduct real time analysis smoothly in the long run. A proper, working internet connection that keeps the end user connected with their surroundings in order to access Twitter easily is another network-related issue the successful functioning of the model is dependent on.

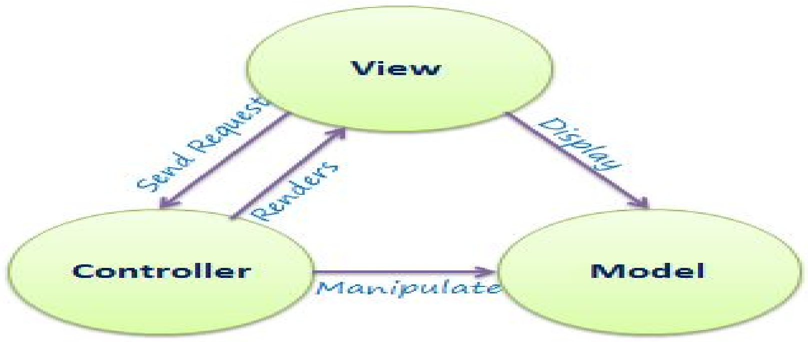
Assumptions:

- Each Twitter user is assumed to be a sensor, that detects a target event and makes an observation. This observation is the actual tweet.
- The tweets are given time values as well as a location (latitudinal and longitudinal coordinates).

4.4 Risks

- Misinformation given by people is the biggest threat that the project faces.
- Additionally, false positives generated by the model can pose a problem that will cause confusion among the users.

4. 5 System Architecture



- Preprocessing the tweets using simple nltk tools
- Tweets of that particular day will be displayed. This will be fetched directly from twitter.com from current 'yyyy:mm:dd 00:00:00' timestamp till the current time 'yyyy:mm:dd hh:mm:ss'
- on clicking on the trending topic, most relevant tweets from the residing country (India) with respect to natural disasters, health hazards (epidemic and breakouts),

terror attacks and severe weather forecasts will be displayed. These will be the only 4 categories.

- If no relevant trending tweets were found that belong to any of these categories, the tweets will be displayed as usual.
- We will plan on adding a default sorting of the tweets based on the likes/shares of the tweets.

Architecture:

We plan to follow the MVC system architecture

1. Model: we will have a business unit with basically our fixed twitter training data. The twitter live data fetched from the website will be our testing data, both stored in our database.
2. View: we do require a frontend view to display to the users. Although there isn't much interaction with the interface, it is required to hide the implementation from the end user and for simple display of resultant tweets
3. Controller: since we have number of algorithms that are being tested on the tweets, a voting mechanism is adopted at the backend and the best algorithm is chosen to display the outputs to the user interface.

4.6 Requirements List

4.7 Module / Scenario 1

Reqmt #	Requirement
CRS – 1	Generates the trending topics of the day after being fed the tweets of the day. Tweets of that particular day will be displayed. This will be fetched directly from twitter.com from current 'yyyy:mm:dd 00:00:00' timestamp till the current time 'yyyy:mm:dd hh:mm:ss'

4.8 Module / Scenario 2

Reqmt #	Requirement
CRS – 1	Figure out the best algorithm to use by testing it on various machine learning and data science models and comparing accuracies (eg:- naive bayes, RNN, Random Forest, SVM, etc

4.9 Module / Scenario n

Reqmt #	Requirement
CRS – 1	Adhere to customization of tweets displayed on the feed based on the trending topics. On clicking on the trending topic, most relevant tweets from the residing country (India) with respect to natural disasters, health hazards (epidemic and breakouts), terror attacks and severe weather forecasts will be displayed. These will be the only 4 categories.

4.10 External Interface Requirements

4.11 Hardware Requirements

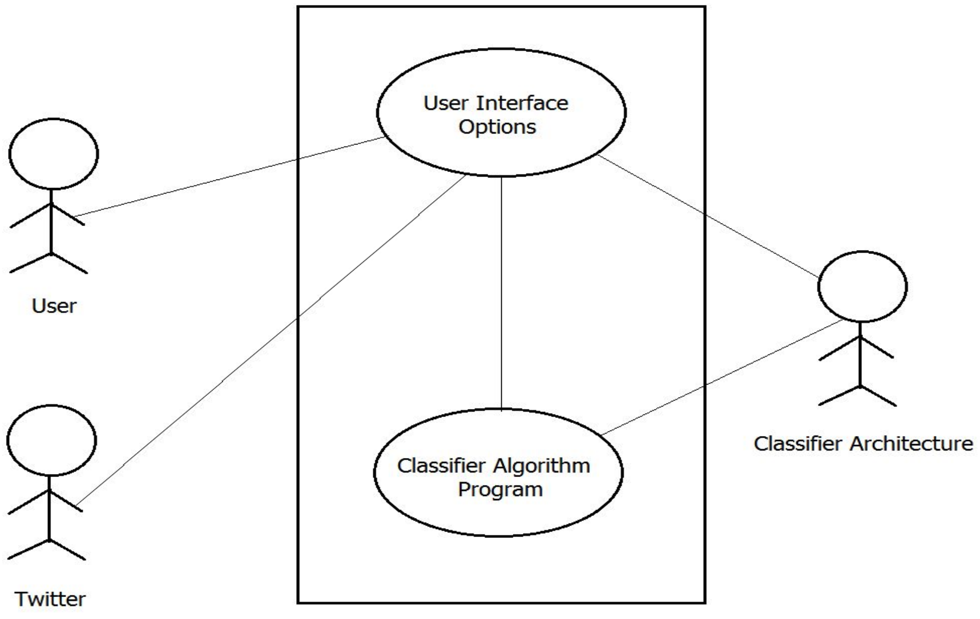
Any laptop with a decent internet connection will be able to use our software.

4.12 Software Requirements

- Python 3.7
- Anaconda navigator
- Pandas
- Scikit-learn

- Keras

4.13 User Interfaces



- We will be providing a simple user interface with the most recent tweets filtered in accordance with the option chosen by the user
- Twitter data from Kaggle is being used for training and live tweets from twitter.com are fetched for testing. We will be using Sentiment140 dataset. It contains 1,600,000 tweets extracted using the twitter api . The tweets have been annotated (0 = negative, 2 = neutral, 4 = positive) and they can be used to detect sentiment . It contains the following 6 fields:
    - target: the polarity of the tweet (0 = negative, 2 = neutral, 4 = positive)
    - ids: The id of the tweet (2087)
    - date: the date of the tweet (Sat May 16 23:58:44 UTC 2009)
    - flag: The query (lyx). If there is no query, then this value is NO_QUERY.

- user: the user that tweeted (robotickilldozr)
- text: the text of the tweet (Lyx is cool)

We will only be using the text column of the dataset

4.14 Performance Requirements

- 1 terminal for the user to look at comparison of different algorithms and their performances and 1 for display of tweets via the UI
- 1 user can use the UI. No login is required.
- The entire day's tweets need to be processed. Could be above 500MB to 4GB
- System only caters to tweets generated in the Indian subcontinent
- 1.6 million tweets will be used for training in the ratio of 80:20 for train test split. After validation, live tweets from twitter.com will be fetched for users.

4.15 Special Characteristics

- A user's login credentials with their username and password as will be provided in the UI will be the security check that will be required to prevent any unauthorized access. Additionally, the backend database will be accessible only to the managers on the backend.
- The dataset will be stored in the servers as mentioned, and the logs of the tweets will be maintained there itself.

4.16 Help

A simple about page will be added for users to understand what our website does and explains the idea behind this project.

4.17 Other Requirements

4.18 Site Adaptation Requirements

- The website that acts as the frontend to display the tweets to a user will be updated in a timely manner to collect all the tweets relevant to a given category.
- The website will also adhere to the latest standards and versions followed by *twitter.com*.

4.19 Safety Requirements

The final website, in keeping with Twitter India's safety standards, will adhere to policies recommended by CSR India. Furthermore, users will be prevented from giving away sensitive information such as their bank details and other national identity numbers. Repeated posting and duplicate tweets, as well as multiple tweets with unrelated links will be marked as spams.

4.20 Packaging

Given that the product aims to collate tweets from different users, the media and method of sharing is going to be packaged as a social media platform itself, and will maintain the look and feel of one as well.

5. High Level System Design

5.1 Design Constraints, Assumptions and Dependencies

TECHNICAL CONSTRAINTS:

Programming Language: The programming language of choice for the bulk of the project is Python, given its immense popularity in non-enterprise-related applications. It offers an easier and faster way to build high-performing algorithms, along with an extensive collection of specialized libraries available as well.

Operating Systems and Platforms Supported: The vision is to build a fully-functional project that works smoothly across platforms, and are currently implementing the project on a standard Windows device.

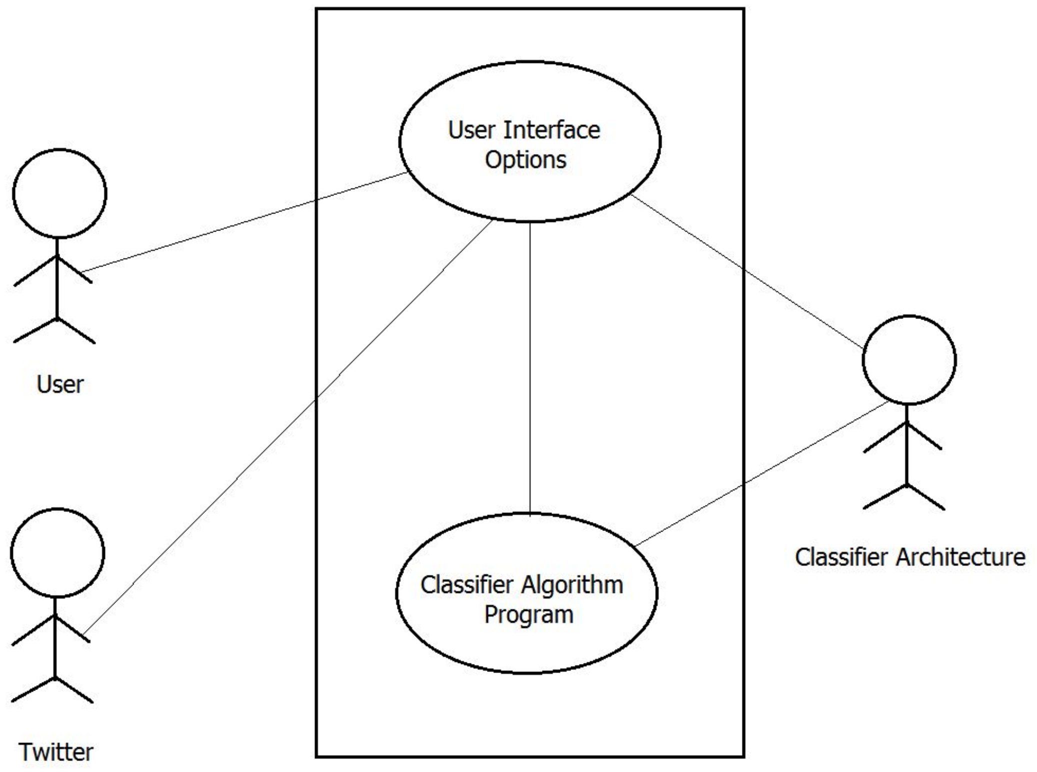
Frameworks: The front-end (user interface) is based on a standard MVC framework.

DEPENDENCIES:

A major portion of the project design and implementation is dependent on the availability of data in large numbers, and subsequently, necessary servers to be able to stream data without any hindrance.

5.2 Design Description

5.3 Use Case Diagram



Use Case Item	Description
User	The users who will use our system
Twitter	Twitter.com API
Classifier	The algorithm

5.4 Class Diagram

Here, a description of each class in this class diagram will be given. A diagram of the entire system will be given at a high level and then broken down into sub levels. Classes maybe repeated across class diagrams, to show the interfaces with other classes. The detailed explanation of each class with its methods will be covered in the low-level design document.

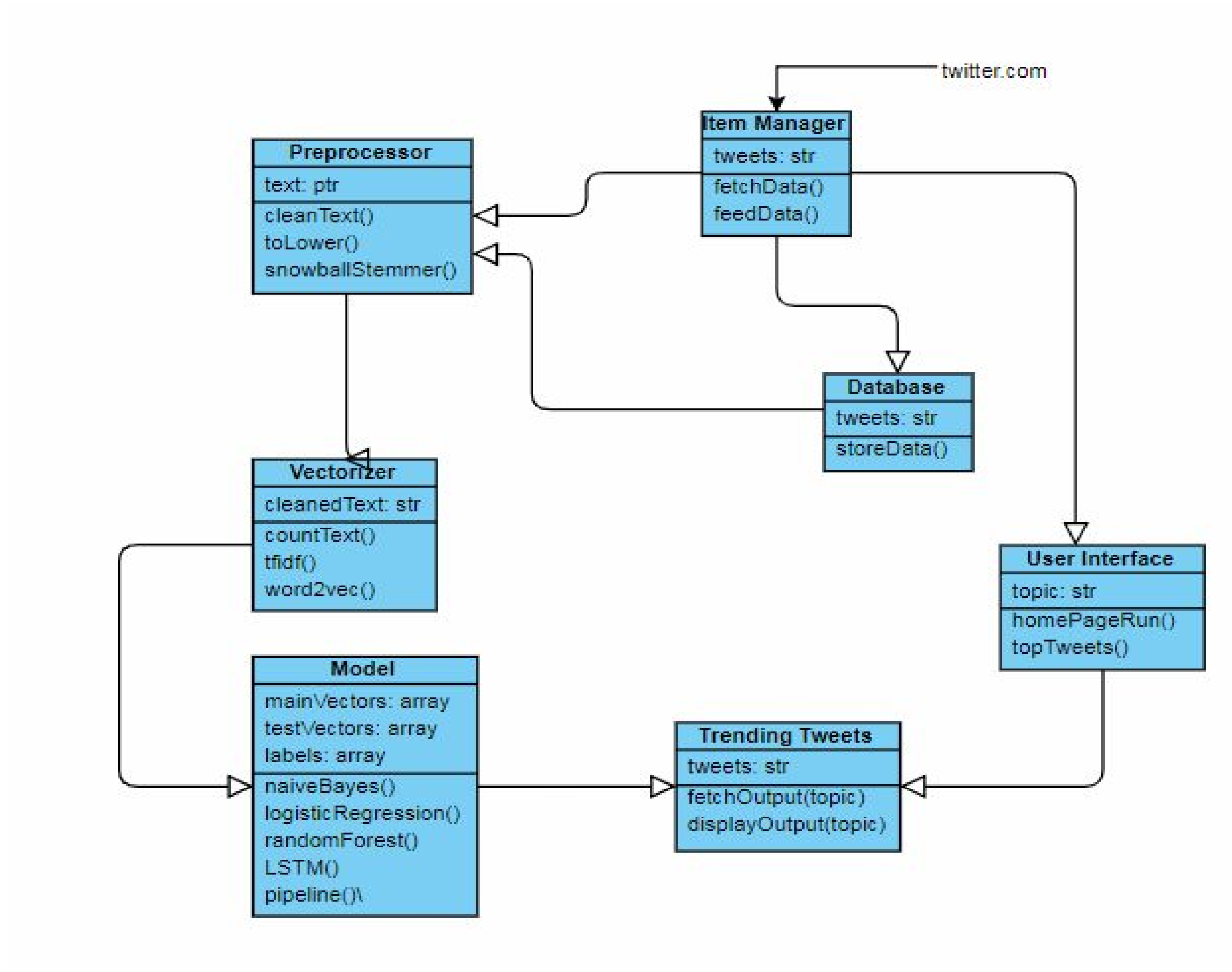
For Example,

Class	Properties	Methods
Preprocessing	Using the nltk library, we will preprocess the tweets by removing the stop words, reduce characters to lower case and tokenize	A simple preprocessing function that will be run on every tweet

Vectorization	Tweets cannot be utilized directly in string format. Hence tweets will need to be vectorized to a format the algorithm will understand	Direct modules to vectorize texts are used
Model ModelTraining ModelTesting EvaluationMetrics	Model Parameters and the subclasses of this class provide more model training and testing related parameters.	Train the model and test the model. Also the subclasses have methods that calculate the error of the methods
Item Manager (In this context refers to a Live incoming tweet)	This runs a code and fetches the top tweets of the day from that day 00:00	Fetch the top tweets of the day
Database	This basically has the entire training data as well as the new incoming testing data	Store the incoming and old tweets in a database
User	Every user is associated with an unique id and is displayed the home page when our website is opened	Display home page with trending tweets and drop down menu

Trending tweets	After user makes a choice, turn to another page, display tweets of the topic chosen. if no tweets for the topic are available, display trending tweets of the day	Displayed trending tweets by running the best performing algorithm on the live twitter data

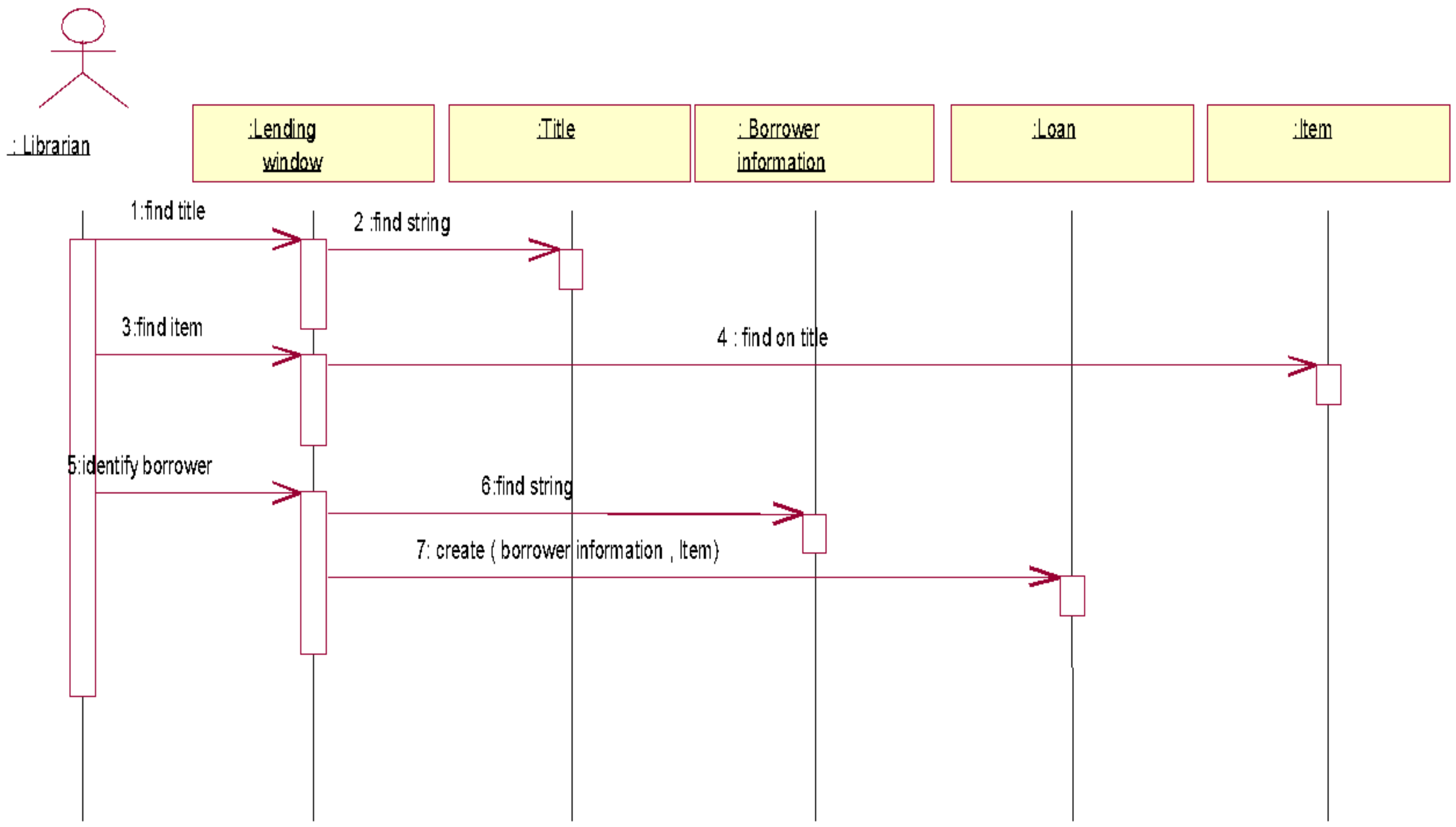
5.5 Class Description



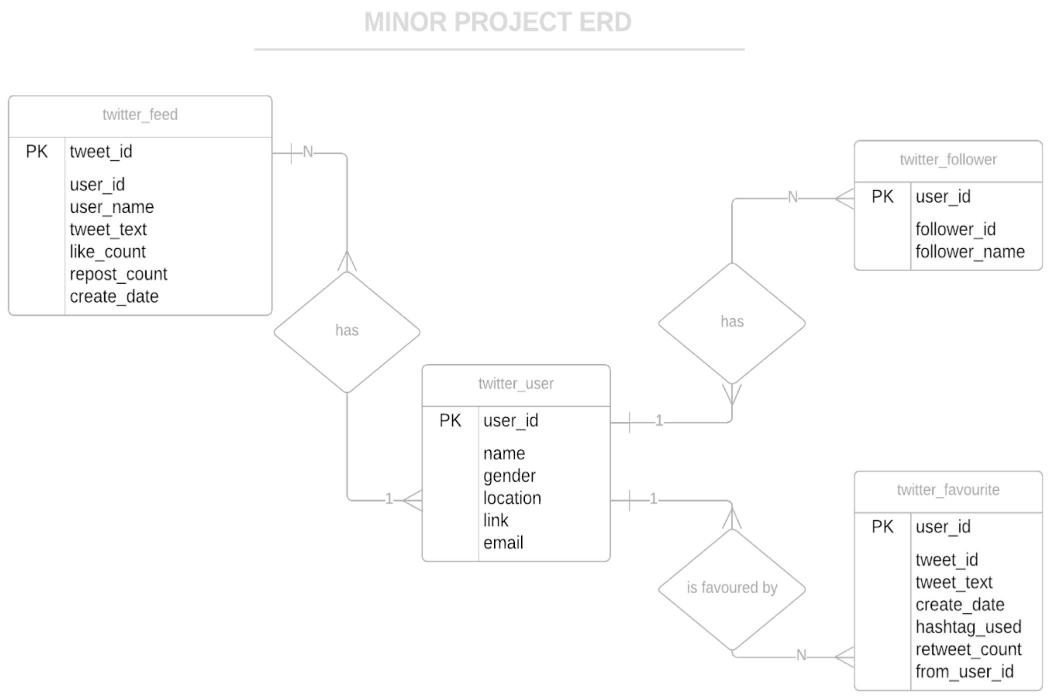
5.6 Sequence Diagram

The Sequence diagram for each module will be presented here.

For Example:


: Librarian
:Lending window
:Title
: Borrower information
:Loan
:Item
1:find title
2 :find string
3:find item
4 : find on title
5:identify borrower
6:find string
7: create (borrower information , Item)

5.7 ER Diagrams



This section will include the ER Diagram. The following table shall be filled for details of the entities and their data elements / attributes.

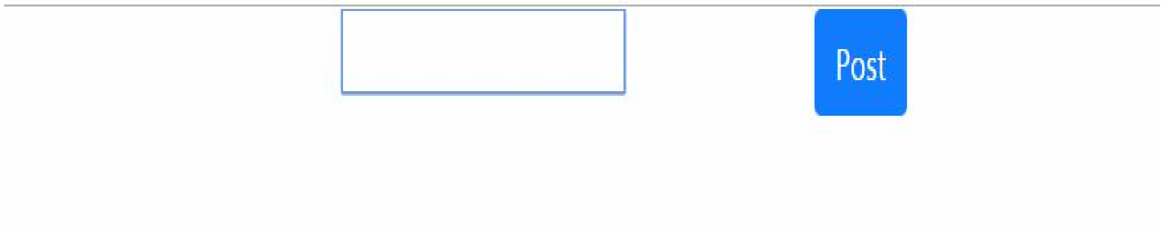
#	Entity	Name	Definition	Type
		ENTITIES		
1.	twitter_feed		Refers to the user's Twitter feed.	
2.	twitter_user		Refers to the Twitter user themselves.	
3.	twitter_follower		Refers to the Twitter follower(s).	

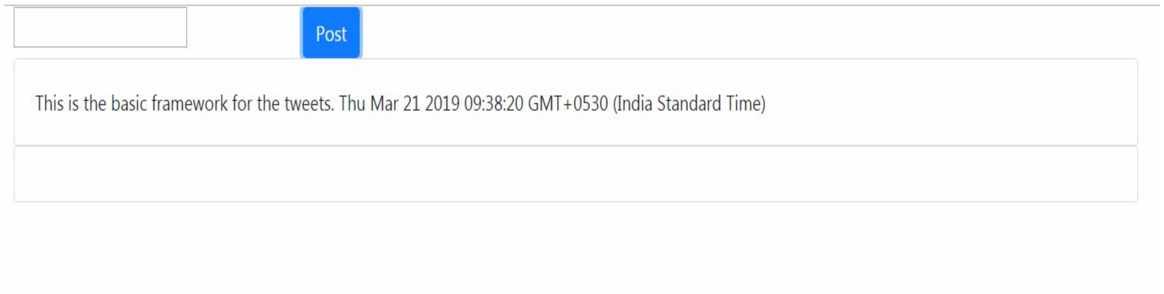
4.	twitter_favourite		Refers to the followers who engage most actively with a specific user.	
#	**Attribute**	**Name**	**Definition**	**Type (size)**
		DATA ELEMENTS		
1.	user_id □user_name □tweet_text □like_count □repost_count □create_date			int str str int int date
2.	user_id name □gender □location □link□ email			int str str str str str
3.	user_id follower_id □follower_name			int int str
4.	user_id tweet_id□ tweet_text			int int str

	create_date □hashtag_used □retweet_count □from_user_id			date str int int

User Interface Diagrams

The UI as it currently stands, looks like this:





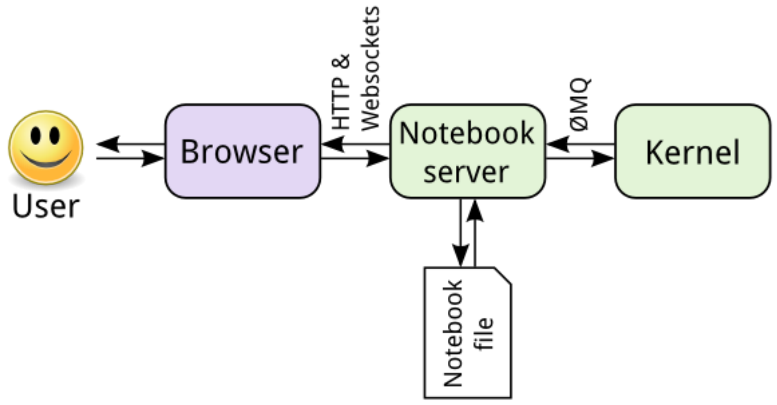
This is the basic framework for the tweets. Thu Mar 21 2019 09:38:20 GMT+0530 (India Standard Time)

5.8 Report Layouts

There is no reports requirements for this product as such.

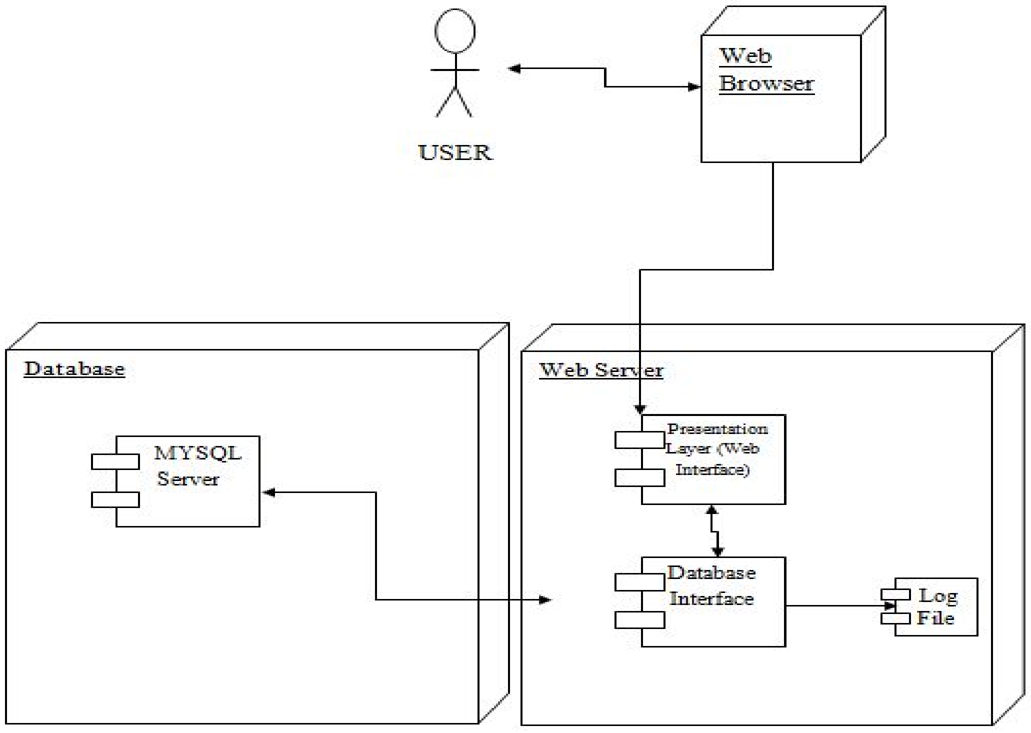
5.9 External Interfaces

Gives an overall diagram as to how the system with known interfaces will work. However, the description of the interface may or may not be covered in this document, depending on whether it is within the scope of the offshore development.



5.10 Packaging and Deployment Diagrams

Following is an abstract package and deployment diagram for the system that shows the most important modules of the system and how they interact with each other.



5.11 Help

Since the product is a user friendly webpage, we do not find the necessity to provide an explicit documentation or user guide. Basic introduction at the start on the home page that guide him through the page such that he can be familiar with all the features of the product should be enough.

5.12 Reusability Considerations

This project involves training machine learning models for twitter data analysis, these models are created once and reused, multiple times. We're using a simple Jupyter notebook interface where the algorithms can be run again and again and the training data is stored in a simple csv format and not using any database management systems. The live testing data from Twitter.com however is stored in a mySQL database.

6. Low Level Detailed Design

With this document we intend to present a picture of the internal description of how we are designing the various features of the project. The class and method descriptions along with the design methodology to be used is described in this document.

6.1 Design Constraints, Assumptions and Dependencies

This section provides the list of constraints, assumptions and dependencies.

6.2 Design Description

This section describes the design with respect to functional modules.

1.1 <u>Trending topic</u>

1.1.1 Preprocessor

Here, a detailed description of each class with its methods shall be described.

1.1.1.1 Class Description

This class handles the preprocessing of all the twitter text data (tweets) in order to be able to analyze it better.

Tweets are first converted to lower case

remove hash tags using BeautifulSoup

remove non-character such as digits and symbols

remove stop words such as "the" and "and"

convert to root words by stemming if needed

Tokenize the words

1.1.1.2 Data Members 1

Data Type	Data Name	Initial Value	Description
str	Text	null	Tweets

1.1.1.2.1 CleanText

The following details shall be defined for the methods:

- Purpose - To include all the sub methods and clean the tweet
- Input - text
- Output - cleaned text

- Parameters - raw_text, remove_stopwords=True, stemming=False, split_text=False
- Pseudo-code -

```
text = BeautifulSoup(raw_text, 'html.parser').get_text()  #remove html
        letters_only = re.sub("[^a-zA-Z]", " ", text)  # remove non-character
        words = letters_only.lower().split() # convert to lower case

        if remove_stopwords: # remove stopword
        stops = set(stopwords.words("english"))
        words = [w for w in words if not w in stops]

        if stemming==True: # stemming
#       stemmer = PorterStemmer()
        stemmer = SnowballStemmer('english')
        words = [stemmer.stem(w) for w in words]

        if split_text==True:  # split text
        return (words)

        return( " ".join(words))
```

1.1.2 <u>Vectorizer</u>

1.1.2.1 Class Description

This class will be used to convert the cleaned tweets in text format to a format the computer algorithm understands

1.1.2.2 Data Members 2

Data Type	Data Name	Initial Value	Description
str	CleanedText	null	pre-processed Tweets

1.1.2.2.1 CountVect()

Uses a method called count vectorization. These vectors are fed into Naive bayes and Logistic regression algorithms

1.1.2.2.2 tfidf()

Uses a method called tfidf vectorization. These vectors are used for Random Forest Classifier

1.1.2.2.3 Word2vec()

Uses a method called word2vec vectorization. These vectors are used for LSTM classifier.

1.1.3 <u>Model</u>

3.1.3.1 Class Description

This class runs all the algorithms on the tweet vectors and decided the best

3.1.3.2 Data Members 2

Data Type	Data Name	Initial Value	Description
array	train vectors	0.0	tweets used for training
array	test vectors	0.0	tweets used for testing

array	labels	0	corresponding labels - 0, 1, 2, 3, 4

3.1.3.2.1 Multinomial naive bayes

Uses a method called multinomial naive bayes by taking as input count vectorized train data

3.1.3.2.2 Logistic regression

Uses a method called logistic regression by taking as input count vectorized train data

3.1.3.2.3 Random Forest

Uses a method called random forest classification by taking as input tfidf vectorized train data

3.1.3.2.4 LSTM()

Uses a method called LSTM RNN by taking as input word2vec vectorized train data

3.1.3.2.5 Pipeline()

Uses a pipeline method with gridSearchCV to decide the best method among the 4 by comparing accuracies and using a voting mechanism.

<u>3.1.4 Item Manager</u>

3.1.4.1 Class Description

This class runs a code and fetches the top tweets of the day from that day 00:00

3.1.4.2 Data Members 2

Data Type	Data Name	Initial Value	Description
str	Tweets	null	Live Tweets from twitter.com

3.1.4.2.1 fetchData()

This fetched data by web scraping twitter.com from the "what's happening" page of twitter.

3.1.4.2.2 FeedData()

This feeds the data into the database after preprocessing it.

3.1.5 Database

3.1.5.1 Class Description

This basically has the entire training data as well as the new incoming testing data.

3.1.5.2 Data Members 2

Data Type	Data Name	Initial Value	Description
str	Tweets	null	training and live Tweets

3.1.5.2.1 storeData()

This basically stores any preprocessed tweets that are fed to it for further algorithm running

3.1.6 User Interface

3.1.6.1 Class Description

Every user is associated with an unique id and is displayed the home page when our website is opened.

3.1.6.2 Data Members 2

Data Type	Data Name	Initial Value	Description
str	Topic	default	4 classification topics

3.1.6.2.1 HomePageRun()

This basically runs the default homepage with the dropdown menu

3.1.6.2.2 TopTweets()

This displays the top tweets as it is on “what’s happening”.

<u>3.1.7 Trending Topic</u>

3.1.5.1 Class Description

After user makes a choice, turn to another page, display tweets of the topic chosen. if no tweets for the topic are available, display trending tweets of the day

3.1.5.2 Data Members 2

Data Type	Data Name	Initial Value	Description
str	Tweets	default	Tweets on the topic chosen

3.1.5.2.1 FetchOutput(topic)

This runs the algorithm to fetch the trending topic tweets as chosen by the user by using the best performer on the live tweets

3.1.5.2.2 DisplayOutput()

This displays the outputs received from 3.1.5.2.1 for the user on the UI.

7. Implementation/Pseudo code

## Twitter trending topic analysis

* #### [Part 1. Data Exploration](#part1)

* #### [Part 2. Data Preparation](#part2)

* #### [Part 3. Bag of Words](#part3)

* #### [Part 4. Word2Vec](#part4)

* #### [Part 5. LSTM](#part5)

* #### [Part 6. Word Cloud](#part6)

## Part 1. Data Exploration

```
import pandas as pd

import numpy as np

import sqlite3


import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

#from wordcloud import WordCloud


from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

from sklearn.naive_bayes import BernoulliNB, MultinomialNB

from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn import metrics

from sklearn.metrics import roc_auc_score, accuracy_score

from sklearn.pipeline import Pipeline


import re

import nltk

from bs4 import BeautifulSoup

import lxml

from textblob import TextBlob

from nltk.corpus import stopwords

from nltk.stem.porter import PorterStemmer

from nltk.stem import SnowballStemmer, WordNetLemmatizer

from nltk import sent_tokenize, word_tokenize, pos_tag


import logging

from gensim.models import word2vec

from gensim.models import Word2Vec

from gensim.models.keyedvectors import KeyedVectors


from keras.preprocessing import sequence
from keras.utils import np_utils

from keras.models import Sequential

from keras.layers.core import Dense, Dropout, Activation, Lambda,SpatialDropout1D

from keras.layers.embeddings import Embedding

from keras.layers.recurrent import LSTM, SimpleRNN, GRU

from keras.preprocessing.text import Tokenizer

from collections import defaultdict

from keras.layers.convolutional import Convolution1D

from keras import backend as K

from keras.layers.embeddings import Embedding
```

### Load Data

```
data_xls = pd.read_excel('Data.xlsx', index_col=None)

data_xls.to_csv('data.csv', encoding='utf-8')

df = pd.read_csv('data.csv')
```

### Data Exploration

```
df = pd.read_csv('train.csv')

print(df.shape)

print(df.head())

print(df.info())

print(df.describe())
```

### Data Visualization

```
# Plot distribution of rating
plt.figure(figsize=(12,8))
df['Category'].value_counts().sort_index().plot(kind='bar', color=(0.2, 0.4, 0.6, 0.6))
plt.title('Distribution of Labels')
plt.xlabel('Label')
plt.ylabel('Count')
```

## Part 2. Data Preparation

```
### Prepare Data
### Train Test Split
# Split data into training set and validation
X_train, X_test, y_train, y_test = train_test_split(df['Tweets'], df['Category'],
test_size=0.2, random_state=0)


print('Load %d training examples and %d validation examples. \n'
%(len(X_train),len(X_test)))


y_train
```

### Text Preprocessing

```
def cleanText(raw_text, remove_stopwords=True, stemming=False, split_text=False):
    '''
    Convert a raw review to a cleaned review
    '''
    text = BeautifulSoup(raw_text, 'html.parser').get_text()  #remove html
    letters_only = re.sub("[^a-zA-Z]", " ", text)  # remove non-character
    words = letters_only.lower().split() # convert to lower case

    if remove_stopwords: # remove stopword
        stops = set(stopwords.words("english"))
        words = [w for w in words if not w in stops]

    if stemming==True: # stemming
#         stemmer = PorterStemmer()
        stemmer = SnowballStemmer('english')
        words = [stemmer.stem(w) for w in words]

    if split_text==True:  # split text
        return (words)

    return( " ".join(words))
```

# Preprocess text data in training set and validation set

```
X_train_cleaned = []
X_test_cleaned = []
```

```
text = 'renewed call for early warning system after quake california quake lead to renewed
call'
for d in X_train:
    try:
        X_train_cleaned.append(cleanText(d))
    except:
        X_train_cleaned.append(text)
print('Show a cleaned Tweet in the training set : \n',  X_train_cleaned[0])
for d in X_test:
    X_test_cleaned.append(cleanText(d))
X_train_cleaned
```

### CountVectorizer with Multinomial Naive Bayes (Benchmark Model)

```
# Fit and transform the training data to a document-term matrix using CountVectorizer
countVect = CountVectorizer()
X_train_countVect = countVect.fit_transform(X_train_cleaned)
print("Number of features : %d \n" %len(countVect.get_feature_names())) #6378
print("Show some feature names : \n", countVect.get_feature_names()[::1000])


# Train MultinomialNB classifier
mnb = MultinomialNB()
mnb.fit(X_train_countVect, y_train)


def modelEvaluation(predictions):
    '''
    Print model evaluation to predicted result
    '''
          print ("\nAccuracy on validation set: {:.4f}".format(accuracy_score(y_test,
predictions)))
    print("\nAUC score : {:.4f}".format(roc_auc_score(y_test, predictions)))
    print("\nClassification report : \n", metrics.classification_report(y_test, predictions))
    print("\nConfusion Matrix : \n",metrics.confusion_matrix(y_test, predictions) )

def modelTrainEvaluation(predictions):
    '''
    Print model evaluation to predicted result
    '''
    print ("\nAccuracy on train set: {:.4f}".format(accuracy_score(y_train, predictions)))
    print("\nAUC score : {:.4f}".format(roc_auc_score(y_train, predictions)))
    print("\nClassification report : \n", metrics.classification_report(y_train, predictions))
    print("\nConfusion Matrix : \n",metrics.confusion_matrix(y_train, predictions) )

# Evaluate the model on validaton set
predictions = mnb.predict(countVect.transform(X_test_cleaned))
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, predictions))
from sklearn.metrics import confusion_matrix
```

```
cm1 = confusion_matrix(y_test,predictions)

print('Confusion Matrix : \n', cm1)
```

### TfidfVectorizer with Logistic Regression

```
# Fit and transform the training data to a document-term matrix using TfidfVectorizer

tfidf = TfidfVectorizer() #minimum document frequency of 5

X_train_tfidf = tfidf.fit_transform(X_train_cleaned)

print("Number of features : %d \n" %len(tfidf.get_feature_names())) #1722

print("Show some feature names : \n", tfidf.get_feature_names()[::1000])


# Logistic Regression

lr = LogisticRegression()

lr.fit(X_train_tfidf, y_train)


# Look at the top 10 features with smallest and the largest coefficients

feature_names = np.array(tfidf.get_feature_names())

sorted_coef_index = lr.coef_[0].argsort()

print('\nTop 10 features with smallest coefficients
:\n{}\n'.format(feature_names[sorted_coef_index[:10]]))

print('Top 10 features with largest coefficients :
\n{}'.format(feature_names[sorted_coef_index[:-11:-1]]))


# Evaluate on the validaton set

predictions = lr.predict(tfidf.transform(X_test_cleaned))
```

```
print(accuracy_score(y_test, predictions))
cm1 = confusion_matrix(y_test,predictions)
print('Confusion Matrix : \n', cm1)
```

### Pipeline and GridSearch

```
# Building a pipeline
estimators = [("tfidf", TfidfVectorizer()), ("lr", LogisticRegression())]
model = Pipeline(estimators)

# Grid search
params = {"lr__C":[0.1, 1, 10], #regularization param of logistic regression
        "tfidf__min_df": [1, 3], #min count of words
        "tfidf__max_features": [1000, None], #max features
        "tfidf__ngram_range": [(1,1), (1,2)], #1-grams or 2-grams
        "tfidf__stop_words": [None, "english"]} #use stopwords or don't

grid = GridSearchCV(estimator=model, param_grid=params, scoring="accuracy",
n_jobs=-1)
grid.fit(X_train_cleaned, y_train)
print("The best paramenter set is : \n", grid.best_params_)

# Evaluate on the validaton set
predictions = grid.predict(X_test_cleaned)
```

```
print(accuracy_score(y_test, predictions))
cm1 = confusion_matrix(y_test,predictions)
print('Confusion Matrix : \n', cm1)


#best model - store it
import pickle
filename = 'final.pkl'
pickle.dump(grid, open(filename, 'wb'))
```

## Part 4. Word2Vec

```
### Parsing Review into Sentences
# Split review text into parsed sentences uisng NLTK's punkt tokenizer
# nltk.download()
tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')


def parseSent(review, tokenizer, remove_stopwords=False):
    '''
    Parse text into sentences
    '''
    raw_sentences = tokenizer.tokenize(review.strip())
    sentences = []
    for raw_sentence in raw_sentences:
        if len(raw_sentence) > 0:
```

```
        sentences.append(cleanText(raw_sentence, remove_stopwords, split_text=True))
    return sentences


# Parse each tweet in the training set into sentences
sentences = []
for review in X_train_cleaned:
    sentences += parseSent(review, tokenizer)


print('%d parsed sentence in the training set\n'  %len(sentences))
print('Show a parsed sentence in the training set : \n',  sentences[10])


### Creating Volcabulary List usinhg Word2Vec Model
# Fit parsed sentences to Word2Vec model
#     logging.basicConfig(format='%(asctime)s     :     %(levelname)s     :
%(message)s',level=logging.INFO)


num_features = 300  #embedding dimension
min_word_count = 10
num_workers = 4
context = 10
downsampling = 1e-3


print("Training Word2Vec model ...\n")
```

```
w2v = Word2Vec(sentences, workers=num_workers, size=num_features, min_count = min_word_count, window = context, sample = downsampling)

w2v.init_sims(replace=True)

w2v.save("w2v_300features_10minwordcounts_10context") #save trained word2vec model

print("Number of words in the vocabulary list : %d \n" %len(w2v.wv.index2word)) #4016

print("Show first 10 words in the vocalbulary list  vocabulary list: \n", w2v.wv.index2word[0:10])
```

### Random Forest Classifer

```
# Random Forest Classifier

rf = RandomForestClassifier(n_estimators=100)

rf.fit(X_train_countVect, y_train)

X_test_countVect = countVect.transform(X_test_cleaned)

predictions = rf.predict(X_test_countVect)

print(accuracy_score(y_test, predictions))

cm1 = confusion_matrix(y_test,predictions)

print('Confusion Matrix : \n', cm1)
```

## Part 6. Word Cloud

In this part, we create word clouds for positive sentiment reviews and negative sentiment reviews of a selected brand, to get an intuition of words frequently appear in different sentiments.

```
from wordcloud import WordCloud
```

```
def create_word_cloud(sentiment):


    #df_brand = df.loc[df['Brand Name'].isin([brand])]

    df_brand_sample = df.sample(frac=0.01)

    word_cloud_collection = ''


    if sentiment == 0:

        df_reviews = df_brand_sample[df_brand_sample["Category"]==0]["Tweets"]


    if sentiment == 1:

        df_reviews = df_brand_sample[df_brand_sample["Category"]==1]["Tweets"]

    if sentiment == 2:

        df_reviews = df_brand_sample[df_brand_sample["Category"]==2]["Tweets"]

    if sentiment == 3:

        df_reviews = df_brand_sample[df_brand_sample["Category"]==3]["Tweets"]

    if sentiment == 4:

        df_reviews = df_brand_sample[df_brand_sample["Category"]==4]["Tweets"]


    for val in df_reviews.str.lower():

        tokens = nltk.word_tokenize(val)

            tokens = [word for word in tokens if word not in stopwords.words('english') and
word != "br"]

        for words in tokens:

            word_cloud_collection = word_cloud_collection + words + ' '
```

```
wordcloud = WordCloud(max_font_size=50, width=500, height=300).generate(word_cloud_collection)
    plt.figure(figsize=(20,20))
    plt.imshow(wordcloud)
    plt.axis("off")
    plt.show()
create_word_cloud(sentiment=0)
create_word_cloud(sentiment=1)
create_word_cloud(sentiment=2)
create_word_cloud(sentiment=3)
create_word_cloud(sentiment=4)
```

# Keras

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Flatten, LSTM, Conv1D, MaxPooling1D, Dropout, Activation
from keras.layers.embeddings import Embedding
import nltk
import string
import numpy as np
import pandas as pd
from nltk.corpus import stopwords
from sklearn.manifold import TSNE


import itertools

import os


%matplotlib inline

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import tensorflow as tf


from sklearn.preprocessing import LabelBinarizer, LabelEncoder

from sklearn.metrics import confusion_matrix


from tensorflow import keras

from keras.models import Sequential

from keras.layers import Dense, Activation, Dropout

from keras.preprocessing import text, sequence

from keras import utils


max_words = 1000

tokenize = text.Tokenizer(num_words=max_words, char_level=False)

tokenize.fit_on_texts(X_train_cleaned) # only fit on train

tokenize.fit_on_texts(X_test_cleaned) # only fit on train
```

```
x_train = tokenize.texts_to_matrix(X_train_cleaned)
x_test = tokenize.texts_to_matrix(X_test_cleaned)


encoder = LabelEncoder()
encoder.fit(X_train_cleaned)
encoder.fit(X_test_cleaned)
num_classes = np.max(y_train)+1
print(num_classes)
Y_train = utils.to_categorical(y_train, num_classes)
Y_test = utils.to_categorical(y_test, num_classes)


batch_size = 32
epochs = 100
```

# Build the model

```
model = Sequential()
model.add(Dense(512, input_shape=(max_words,)))
#model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])


history = model.fit(x_train, Y_train,
                    batch_size=batch_size,
                    epochs=epochs,
verbose=1,
                    validation_split=0.2)
score = model.evaluate(x_test, Y_test,
                       batch_size=batch_size, verbose=1)
print('Test accuracy:', score[1])


preds = model.predict_classes(x_test)
print(accuracy_score(y_test, preds))
cm1 = confusion_matrix(y_test,list(preds))
print('Confusion Matrix : \n', cm1)
```

#testing

```
df2 = pd.read_csv('test_data.csv')
cleaned = []
text = 'renewed call for early warning system after quake california quake lead to renewed
call'
for d in df2['text']:
    try:
        cleaned.append(cleanText(d))
    except:
        cleaned.append(text)
print('Show a cleaned Tweet in the training set : \n',  cleaned[2])


tfidf = TfidfVectorizer() #minimum document frequency of 5
X_tfidf = tfidf.fit_transform(cleaned)
print("Number of features : %d \n" %len(tfidf.get_feature_names())) #1722
print("Show some feature names : \n", tfidf.get_feature_names()[::1000])


final_model = pickle.load(open(filename, 'rb'))
label = final_model.predict(cleaned)
df2['Category'] = label
df2.to_csv('test_data2.csv')
df2.tail(25)


for i in range(len(df2)):
    if(df2['Category'][i] == 3):
        print(df2['text'][i])
```

#display to the user

```
print("Top 10 tweets for label 1: natural disaster")
count = 0
```

```
for i in range(len(df2)):
    if count < 11:
        if(df2['Category'][i] == 1):
            print(df2['text'][i])
            count = count + 1
```

#display to the user

```
print("Top 10 tweets for label 2: disease outbreaks")
count = 0
for i in range(len(df2)):
    if count < 11:
        if(df2['Category'][i] == 2):
            print(df2['text'][i])
            count = count + 1
```

#display to the user

```
print("Top 10 tweets for label 3: terror attacks")
count = 0
for i in range(len(df2)):
    if count < 11:
        if(df2['Category'][i] == 3):
            print(df2['text'][i])
            count = count + 1
```

#display to the user

```
print("Top 10 tweets for label 4: extreme weather")
count = 0
for i in range(len(df2)):
    if count < 11:
        if(df2['Category'][i] == 4):
            print(df2['text'][i])
            count = count + 1
```

8. Testing

8.1 Test Strategies

1. Unit testing:

Every single element of our project needs to be tested. For this, we first prepare a test dataset with the relevant tweets belonging to all categories 0, 1, 2, 3, 4. We generate a label for each of these tweets using all the algorithms available.

We then display a few tweets belonging to each category. We manually analyze the tweets and finalize on the algorithm to be chosen.

2. Integration testing:

This is the phase where we need to merge both the front-end with the Python back-end. We will then feed tweets whose labels we know and generate labels and compare with the actual labels. This will also help us in generating an accuracy for better understanding of how the algorithm chosen performs.

This stage hasn’t been implemented yet because the front-end is yet to be completed.

3. System Testing

There really isn’t a system testing stage as after integration, the entire system is completely built and hence becomes redundant.

The testing then happens in these few steps:

- Test on validation data (train_test_split)
- Test on test dataset
- Manually evaluate performance
- Test on live stream Twitter data
- Manually evaluate performance
- Perform integration testing to evaluate the performance of the front-end and correct errors if any
- Beta test data by displaying options to various users via the front-end
- Add any user requirements and send for production

8.2 Performance Criteria

1 terminal for the user to look at comparison of different algorithms and their performances and 1 for display of tweets via the UI

- 1 user can use the UI. No login is required.
- The entire day’s tweets need to be processed. Could be above 500MB to 4GB
- System only caters to tweets generated in the Indian subcontinent
- 1.6 million tweets will be used for training in the ratio of 80:20 for train test split.

After validation, live tweets from twitter.com will be fetched for users.

8.3 Test Environment

The test environment is basically right now jupyter notebook. The accuracy and the top results from running the algorithm in the test dataset has given us these outputs. These outputs have been manually analyzed to conclude the performance of our algorithm. The hardware environment is nothing but a PC with jupyter notebook installed. This will work with both Ubuntu and Windows systems.

8.4 Roles and Responsibilities

Data mining: Gathering data suitable for testing. This testing data should not be labelled and must have equal number of tweets belonging to all 5 categories.

Database management: Here, we need to organize the data properly and store them

Pre-processing: The test data also has to be pre-processed

Model Evaluation: The test data after being pre-processed and converted to the right vector form is passed into the model to obtain results

Testing: we use different models, observe results and conclude the performance and draw justifications for the behaviour of the results.

8.5 Test Schedule

1. Test on validation data (train_test_split)
2. Test on test dataset
3. Manually evaluate performance
4. Test on live stream Twitter data
5. Manually evaluate performance
6. Perform integration testing to evaluate the performance of the front-end and correct errors if any
7. Beta test data by displaying options to various users via the front-end

8. Add any user requirements and send for production

8.6 Test Tools Used

The test tools used are:

Python3

Jupyter Notebook

Pandas

Scikit-learn

Keras

Numpy

8.6 Acceptance Criteria

The test data is said to have passed the acceptance criteria if we can manually verify the validity of the data it is predicting

The test data was run on the best algorithm we found, the logistic regression algorithm with fine tuned parameters and the results were good for 0, 1, 2 and 4 and were weak for 3

8.7 Test Case List

This section shall clearly define the test cases that are planned for testing. The following information shall be mentioned: -

Test Case Number	**Test Case**	**Required Output**

0	None of the above	0
1	Natural Disasters	1
2	Disease Outbreaks	2
3	Terror Attacks	3
4	Extreme Weather	4

8.8 Test Data

Just happened a terrible car crash

Our Deeds are the Reason of this #earthquake May ALLAH Forgive us all

Heard about #earthquake is different cities, stay safe everyone.

Forest fire near La Ronge Sask. Canada

All residents asked to 'shelter in place' are being notified by officers. No other evacuation or shelter in place orders are expected

13,000 people receive #wildfires evacuation orders in California

Just got sent this photo from Ruby #Alaska as smoke from #wildfires pours into a school

#RockyFire Update => California Hwy. 20 closed in both directions due to Lake County fire - #CAfire #wildfires

#flood #disaster Heavy rain causes flash flooding of streets in Manitou, Colorado Springs areas

Typhoon Soudelor kills 28 in China and Taiwan

We're shaking...It's an earthquake

9. Results and Discussion

We finally observed that Logistic regression performs the best in predicting the right labels. We analyzed why this could be the case even when there are more complicated algorithms like Neural Networks and Random Forest classifier. The Deep Neural Network is estimating many more parameters and even more permutations of parameters than the logistic regression. You'd be estimating lots of parameters with little data per parameter and get a bunch of spurious results. Therefore depending upon the situation, the additional granularity of the Deep Neural Network would either represent a treasure trove of additional detail and value, or an error prone and misleading representation of the situation. Logistic Regression is one of the most used Machine Learning algorithms for binary classification. It is a simple Algorithm that you can use as a performance baseline, it is easy to implement and it will do well enough in many tasks. logistic regression does work better when you remove attributes that are unrelated to the output variable as well as attributes that are very similar (correlated) to each other. Therefore Feature Engineering plays an important role in regards to the performance of Logistic and also Linear Regression. Another advantage of Logistic Regression is that it is incredibly easy to implement and very efficient to train.

After choosing Logistic Regression as our final model, we fine tune it using GridSearchCV and finding the best parameters. After running this model on our test data we get outputs for each label which is clearly seen in the test outputs displayed.

Results of the training:

1. Multinomial Naive Bayes:

```
With count vectorization
0.9408424041646948
Confusion Matrix :
 [[ 52  18  15  14]
 [  0 784   0   1]
 [  2   2 819   2]
 [ 19  29  23 333]]
```

2. Logistic regression:
With tfidf vectorization
0.9522006625650734
Confusion Matrix :

```
[[ 51  19   5  24]
 [  0 785   0   0]
 [  0   0 821   4]
 [ 18  26   5 355]]
```

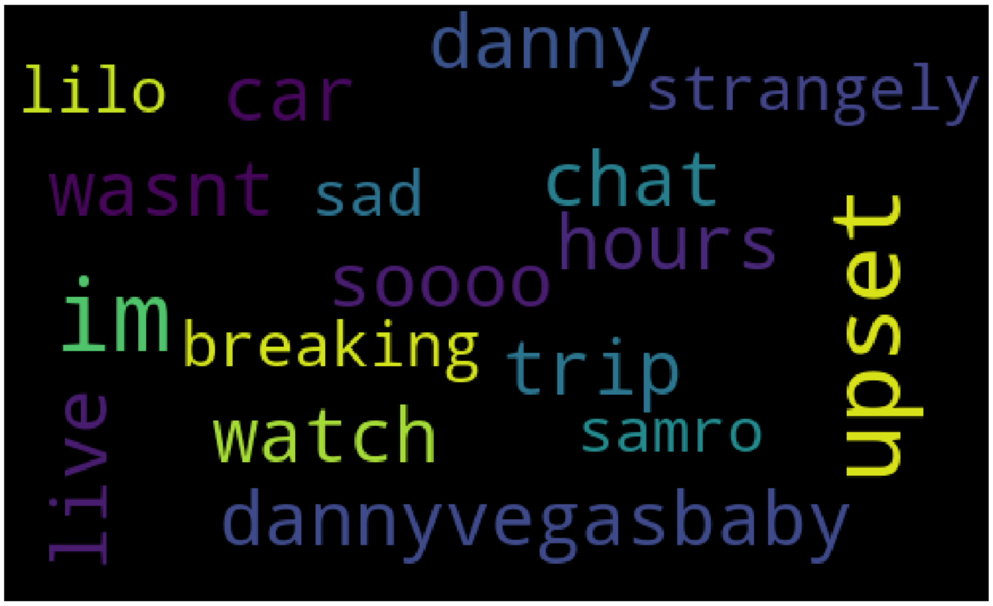
3. Random Forest classifier
With word2vec vectorization
0.9654519640321817
Confusion Matrix :

```
[[ 76  18   0   5]
 [  0 782   0   3]
 [  2   2 819   2]
 [ 31   9   1 363]]
```

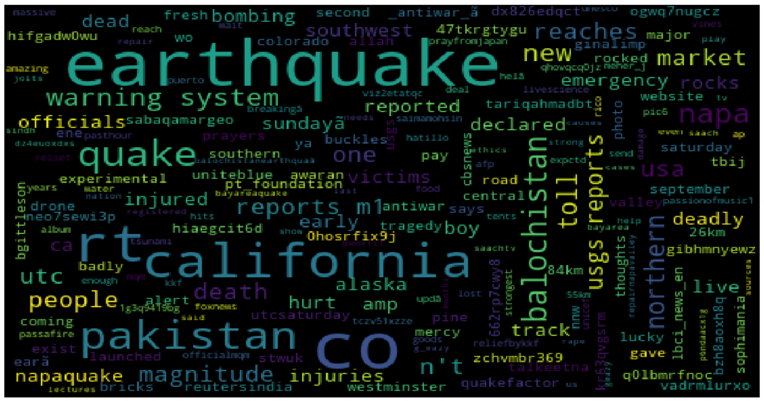
4. Artificial Neural Networks
With Text Tokenizer
0.9657610718273167
Confusion Matrix :

```
[[399   0   1   1  16]
 [  3 759   1   0  10]
 [  4   0 835   0   5]
 [  4   0   0 267   2]
 [ 38   2   3   2 335]]
```

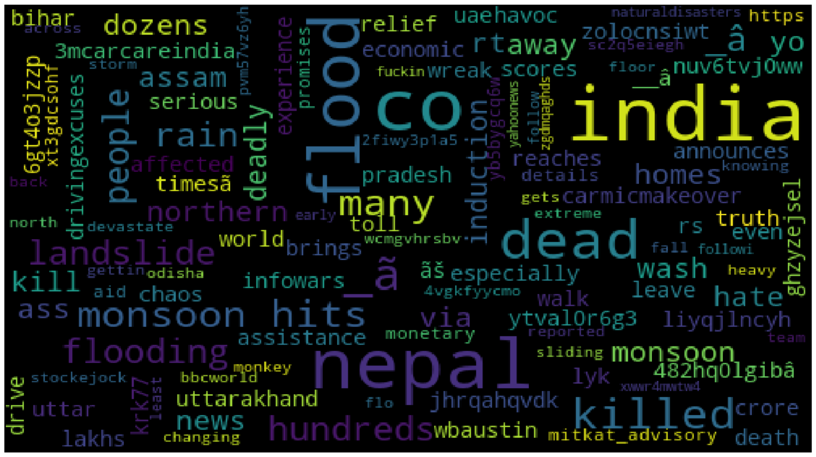
10. Snapshots



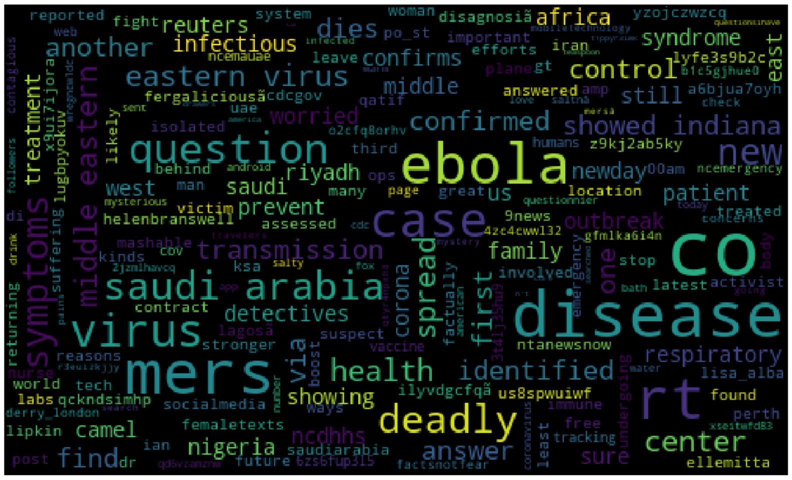
Label 0



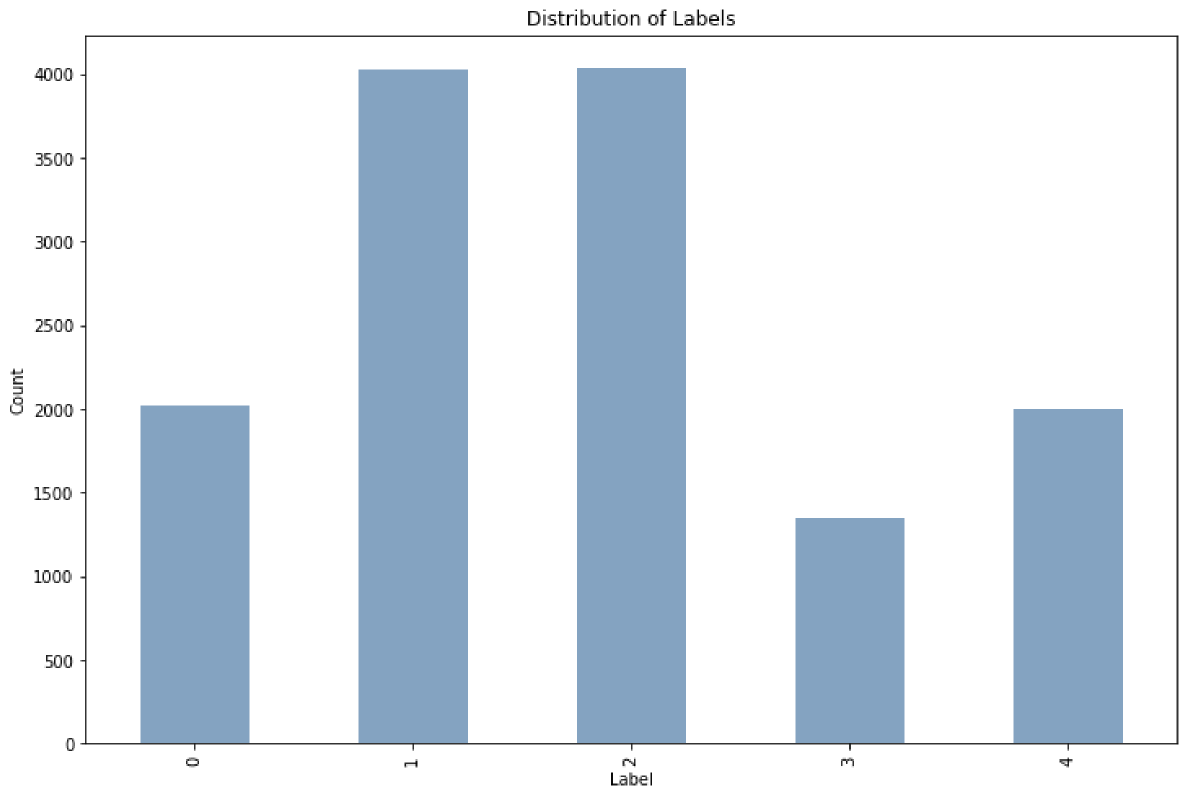
Label 1



Label 2



Label 4



Dataset Description:

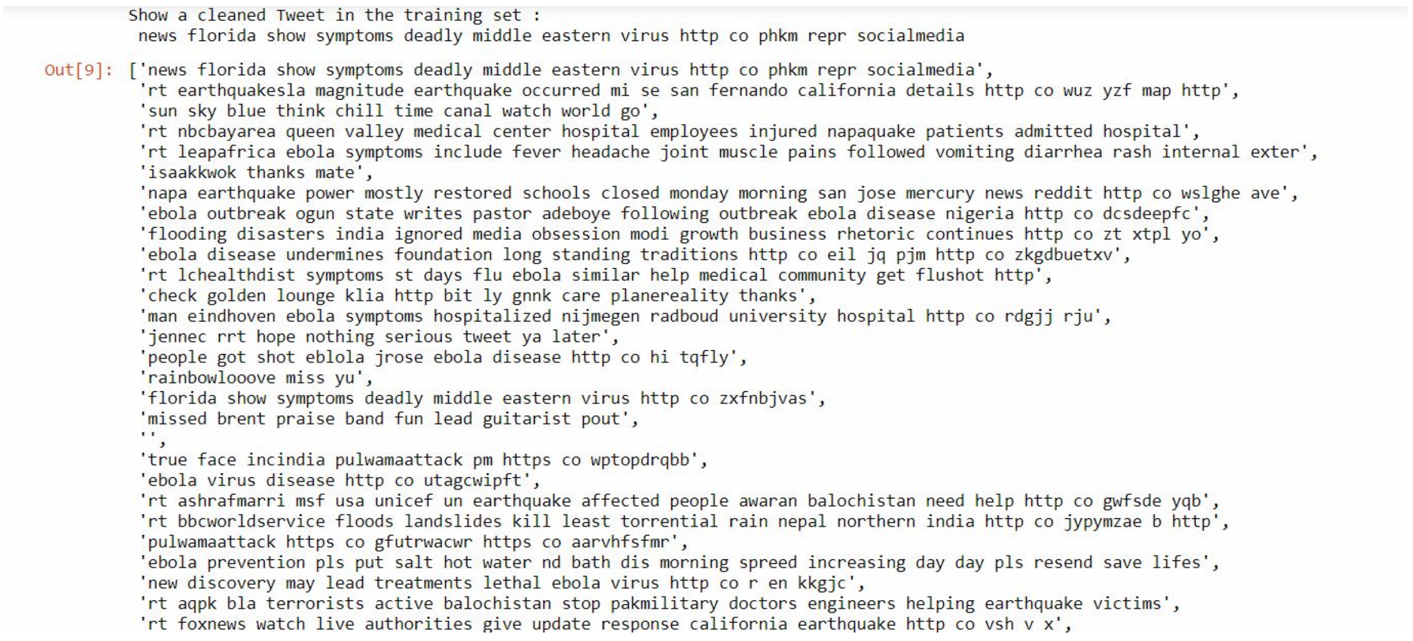
```
shape: (10564, 3)

   Unnamed: 0                                    Tweets  Category

0          0  USGS reports a M1.7 #earthquake 70km ESE of He...         1

1          1  QuakeFactor M 3.0, Southern Alaska: Sunday, Se...        1

RangeIndex: 10564 entries, 0 to 10563

Data columns (total 3 columns):

        Unnamed: 0      Category

count  10564.000000  10564.000000

mean    5281.500000      1.903256

std     3049.708456      1.148911

min        0.000000      0.000000
```

```
25%     2640.750000     1.000000
50%     5281.500000     2.000000
75%     7922.250000     2.000000
max    10563.000000     4.000000
```

Loading 8451 training examples and 2113 validation examples.

After cleaning and preprocessing data,

napa looks like quake strong earthquake rocked northern california early sunday morning inj http co wr n bshu



```
Show a cleaned Tweet in the training set :
 news florida show symptoms deadly middle eastern virus http co phkm repr socialmedia

Out[9]: ['news florida show symptoms deadly middle eastern virus http co phkm repr socialmedia',
         'rt earthquakesla magnitude earthquake occurred mi se san fernando california details http co wuz yzf map http',
         'sun sky blue think chill time canal watch world go',
         'rt nbcbayarea queen valley medical center hospital employees injured napaquake patients admitted hospital',
         'rt leapafrica ebola symptoms include fever headache joint muscle pains followed vomiting diarrhea rash internal exter',
         'isaakkwok thanks mate',
         'napa earthquake power mostly restored schools closed monday morning san jose mercury news reddit http co wslghe ave',
         'ebola outbreak ogun state writes pastor adeboye following outbreak ebola disease nigeria http co dcsdeepfc',
         'flooding disasters india ignored media obsession modi growth business rhetoric continues http co zt xtpl yo',
         'ebola disease undermines foundation long standing traditions http co eil jq pjm http co zkgdbuetxv',
         'rt lchealthdist symptoms st days flu ebola similar help medical community get flushot http',
         'check golden lounge klia http bit ly gnnk care planereality thanks',
         'man eindhoven ebola symptoms hospitalized nijmegen radboud university hospital http co rdgjj rju',
         'jennec rrt hope nothing serious tweet ya later',
         'people got shot eblola jrose ebola disease http co hi tqfly',
         'rainbowlooove miss yu',
         'florida show symptoms deadly middle eastern virus http co zxfnbjvas',
         'missed brent praise band fun lead guitarist pout',
         '',
         'true face incindia pulwamaattack pm https co wptopdrqbb',
         'ebola virus disease http co utagcwipft',
         'rt ashrafmarri msf usa unicef un earthquake affected people awaran balochistan need help http co gwfsde yqb',
         'rt bbcworldservice floods landslides kill least torrential rain nepal northern india http co jypymzae b http',
         'pulwamaattack https co gfutrwacwr https co aarvhfsfmr',
         'ebola prevention pls put salt hot water nd bath dis morning spreed increasing day day pls resend save lifes',
         'new discovery may lead treatments lethal ebola virus http co r en kkgjc',
         'rt aqpk bla terrorists active balochistan stop pakmilitary doctors engineers helping earthquake victims',
         'rt foxnews watch live authorities give update response california earthquake http co vsh v x',
```

11. Conclusions

To conclude. we aimed at building a model that analyses Twitter data (Tweets) and categorizes haphazard tweets into 4 categories: natural disasters, disease outbreaks, terror attacks and extreme weather. We believe that this news is essential and should be prioritized over other irrelevant tweets (like about a celebrity's wedding). We achieved this by a Python backend that uses 4 algorithms- Multinomial Naive Bayes, Logistic Regression, Random Forests and Artificial Neural Networks. They train on our labeled train dataset and achieved validation results as mentioned above. We also had a front end that displays the data according to the option chosen by the user to display.

12. Further Enhancements

We intend to further take this project further and convert it into a full fledged news reporting site that not only analyses data from Twitter.com but also from Facebook status updates, inshorts, etc and possibly have more categories than the 4 we have now. We could have a UI with a lot more features where the user can also tweet or comment on the tweet along with repost and like features. Social media is always evolving and having a specific site only for urgent important news display will become quickly popular among the masses.

13. References

[1] Catching the Long-Tail: Extracting Local News Events from Twitter; Puneet Agarwal, Rajgopal Vaithiyanathan, Saurabh Sharma and Gautam Shroff

[2] Topic Extraction from News Archive Using TF*PDF Algorithm; Khoo Khyou Bun Mitsuru Ishizuka

[3] A Survey of Techniques for Event Detection in Twitter; Farzindar Atefeh and Wael Khreich

[4] Sentiment-Based Event Detection in Twitter; Georgios Paltoglou

[5] Emerging Event Detection in Social Networks with Location Sensitivity; Unankard Sayan, Xue Li, and Mohamed A Sharaf

[6] Predicting Crowd Behaviour with Big Public Data; N. Kallus

[7] Event Detection in Social Media Data; Wenwen Dou, Xiaoyu Wang, William Ribarsky, Michelle Zhou

[8] Tweet Analysis for Real-Time Event Detection and Earthquake Reporting System Development Ekta, Paahuni Khandelwal, Priya Bundela, Richa Dewan