

A PROJECT REPORT ON

RUMOUR DETECTION ON TWITTER SITES USING LSTM

SUBMITTED TO THE SAVITRIBAI PHULE PUNE UNIVERSITY, PUNE
IN THE PARTIAL FULFILLMENT FOR THE AWARD OF THE DEGREE
OF

BACHELOR OF ENGINEERING
IN
INFORMATION TECHNOLOGY

BY

APEKSHA AGASE: 71706633K

APARNA BINDAGE: 71706686L

SAI PANDE: 71707382D

DATTA DHEBE: 71812980G

UNDER THE GUIDANCE OF
Mrs. S. M. JAYBHAYE



DEPARTMENT OF INFORMATION TECHNOLOGY
SINHGAD COLLEGE OF ENGINEERING,
PUNE VADGAON (BK), PUNE 411041
2019-20

CERTIFICATE

This is to certify that the project report entitled

RUMOUR DETECTION ON TWITTER USING LONG SHORT MEMORY

Submitted by

Apeksha Agase: 71706633K
Aparna Bindage: 71706686L
Sai Pande: 71707382D
Datta Dhebe: 71812980G

Is a bonafide work carried out by them under the supervision of Mrs. S. M. Jaybhaye and it is approved for the partial fulfillment of the requirement of Savitribai Phule Pune University for the award of the Degree of Bachelor of Engineering (Information Technology).

This project report has not been earlier submitted to any other Institute or University for the award of any degree or diploma

Mrs. S. M. Jaybhaye
Internal Guide

Department of Information Technology

Prof. G. R. Pathak
Head Of Department

Department of Information Technology

External Examiner

Date:

Place:

Principal

Sinhgad College of engineering, Pune

ACKNOWLEDGEMENT

We are highly indebted of our guide Mrs. S.M. Jaybhaye for her constant support, guidance and supervision throughout the course of project. We are also thankful to our guide for providing valuable inputs for completion of report. We would like to express our gratitude an appreciation to Mr. N.H. Deshpande for reviewing our project's progress and providing us valuable suggestions. We would like to express our special gratitude towards the staff members of the Department of Information Technology for giving such attention and time. This acknowledgment would be incomplete without expressing our thanks to Prof. G. R. Pathak, Head of the Department (Information Technology) for his support during the work. We are very grateful to our Principal, Dr. S.D. Lokhande who provided a lot of valuable support. We are thankful to our family and friends who encouraged and co-operated us which helped us in the completion of this report.

ABSTRACT

Microblogging platforms like Twitter serves as an ideal place for fabricating and diffusing rumours. These rumours disseminate at a rapid pace and lead to grievous social issues. Rumours can cause social panic and adversely affect people and organizations. Therefore, detecting rumours accurately, quickly and automatically is important. Existing models detect rumours on social media sites by employing feature extraction which is time-consuming, biased and labour-intensive. Even though, recent studies use machine learning-based methods for automatic rumour detection by extracting features of rumour contents (e.g., people's opinions, questions, etc.) and static spreading processes, early detection of rumour remains a challenge. The proposed system employs a deep learning model, Long Short-Term Memory (LSTM) in collaboration with pooling function of Convolutional Neural Network (CNN) to quickly detect rumour. LSTM networks makes predictions based on time series data. It also overcomes problems faced while training traditional Recurrent Neural Network (RNN), like, exploding and vanishing gradient. The dynamic changes of forwarding contents are taken into consideration using LSTM based model.

Keywords— *Rumour identification, Twitter, Long Short-Term Memory, Microblogging sites, Early detection.*

LIST OF FIGURES

1	Rumour example	1
2	DFD Level 0	10
3	DFD Level 1	10
4	DFD Level 2	11
5	Use Case Diagram for Rumour Detection System	12
6	Class Diagram for Rumour Detection System	13
7	Sequence Diagram for Rumour Detection System	14
8	Activity Diagram for Rumour Detection System	15
9	State Diagram for Rumour Detection System	16
10	Block Diagram for Rumour Detection System	18
11	Forget Gate Layer	21
12	Input Gate Layer	22
13	Cell State	23
14	Output Gate Layer	24
15	The LSTM-based Rumour Detection Model	25
16	Confusion matrix plot	49
17	Train accuracy vs. epoch	50
18	Train loss vs. epoch	50
19	Accuracy of different methods	51
20	Login Interface of Rumour Detection Model	53
21	Alter user for number of attempts left to login successfully	53
22	Interface to check rumour	54
23	Tweets retrieved related to keywords	54
24	Result obtained for the tweet	55
25	Interface to view graph	55

LIST OF TABLES

1	Comparison of existing system and proposed system	4
2	Software Requirements	9
3	Hardware Requirements	9
4	Unit testing table	44
5	Integration testing table	45
6	System testing table	46
7	Acceptance testing table	47
8	Summary of Input	48
9	Results of Model	49

CONTENTS

CERTIFICATE	I
ACKNOWLEDGEMENT	II
ABSTRACT	III
LIST OF FIGURES	IV
LIST OF TABLES	V

CHAPTER	TITLE	PAGE NO.
1.	INTRODUCTION TO RUMOUR DETECTION ON TWITTER USING LONG SHORT-TERM MEMORY	
1.1	Introduction To Project.....	1
1.2	Project Idea.....	2
1.3	Motivation Behind The Project Idea.....	3
1.4	Aim And Objectives.....	3
1.5	Project Goals.....	3
2.	BACKGROUND REVIEW AND LITERATURE SURVEY	
2.1	Background Review.....	4
2.2	Literature Survey.....	4
3.	REQUIREMENT AND ANALYSIS	
3.1	Problem Definition.....	8
3.2	Requirements specification.....	8
3.3	Software And Hardware Requirement.....	9
4.	DESIGN	
4.1	Data Flow Diagrams.....	10
4.2	UML Diagrams.....	12
4.2.1	Use Case Diagram.....	12
4.2.2	Class Diagram.....	13
4.2.3	Sequence Diagram.....	14
4.2.4	Activity Diagram.....	15
4.2.5	State Diagram.....	16
4.3	Functionality Of System.....	17
4.4	Block Diagram Of System.....	18
4.5	Long Short-Term Memory.....	20
5.	IMPLEMENTATION	
5.1	System Architecture.....	25
5.2	Code.....	28
6.	TESTING.....	43
7.	RESULTS AND EVALUATION.....	48
8.	GRAPHICS USER INTERFACE OF THE SYSTEM.....	53
9.	CONCLUSION AND FUTURE WORK.....	56
10.	PAPER PUBLICATION AND SPONSORSHIP DETAILS	57
11.	REFERENCES.....	60

1. INTRODUCTION TO RUMOUR DETECTION ON TWITTER USING LONG SHORT-TERM MEMORY

1.1 INTRODUCTION

Social media has become one of the major platforms which facilitate people and organizations to create or share information, career interests, ideas, news and much more. Social media is blooming, particularly microblogging sites like Twitter, Tumblr, and Sina Weibo are admired widely because these sites provides fast propagation and acquisition of information. Nowadays, social media is easily available which helps us to connect people across the globe. Twitter has become one of the popular social networking site on which users interact and post messages called “tweets”. These tweets comprises of user’s opinion, photos, videos, article links and quotes. Twitter gained popularity as a microblogging service because of its 280-character messages called tweets. These short message service provided by Twitter became extensively popular amongst people. These microblogging sites generate a large quantity of multimedia content which is crucial in many important applications. In spite of Twitter being a great platform for sharing and gaining information, it is vulnerable to quick dissemination of rumours which can cause damage to the society.

A rumour is a statement which is circulated without confirming the facts [1]. Rumours arise in the context of ambiguity or when the situation is not clear to people [2]. Rumours hence are harmful force that affects people and organizations [3]. For example, a rumoured tweet saying two explosions in White House and Barack Obama is injured was released from the official Twitter account of the Associated Press (AP) which was hacked on April 23, 2013.



Figure 1: Rumor example

This rumour created a huge havoc and social panic immediately and was extensively reposted. The aftermath of the rumour was massive leading to S&P 500 index instantly dropped by 14 points, knocking down \$136.5 billion in seconds and the Dow Jones Industrial Average also dropped around 145 in three minutes.

To reduce the negative and adverse impact of rumours on society, detecting and debunking them at an initial stage of diffusion should be the topmost priority. To check the credibility of the tweet, some organizations have provided services like rumour querying and rumour denial websites. Some websites like factcheck.org, snopes.com are made available for people to substantiate that the tweet is a truth or a rumour. These websites debunk rumours by manually verifying them or by referring public reports. This process requires a lot of time, funding and manpower. Twitter also implements a semi-automatic strategy which combines the automatic evaluation and crowd-sourcing annotation to flag possible rumoured tweets. The algorithm generates automatic credibility rating for each tweet to determine the tweet's credibility. Twitter allows users to give their feedback or response if their opinion contradicts with the rating generated by the algorithm but this process can let to biased decisions. Apart from these drawbacks, authenticating rumours is a time-consuming process. To overcome these drawbacks of the existing systems, a rumour detection model is proposed and built using LSTM networks integrated with the pooling operation of CNN. This model anticipates the output based on the contents of tweets.

1.2 PROJECT IDEA

- The goal is to build an automated rumour detection system for precise detection of rumours.
- LSTM-based model executes remarkably better than feature-based learning methods. Therefore, the proposed model can easily learn the hidden features and the local information very well.
- Further, the model allows early discovery of rumours and non-rumours, which on comparing to existing methods of rumour detection serves efficiently.

1.3 MOTIVATION BEHIND PROJECT

- Traditional methods like feature engineering is biased, time consuming and labour intensive.
- Websites like twittertrails.com and snopes.com requires financial resources, manpower and are time consuming.
- To overcome problem of manual labeling.
- To detect fake news quickly and accurately.

1.4 PROJECT AIM AND OBJECTIVE

1.4.1 Aim

- The aim of this project is to automatically detect rumour on microblogging site using CNN as well as LSTM and to increase the accuracy of learning model than the existing models.

1.4.2 Objectives

- To detect rumours automatically using deep learning.
- To increase the accuracy of learning model as compared to the existing models.
- To detect rumours as early as possible.

1.5 PROJECT GOAL

- To provide a mechanism that will detect rumour as early as possible.

2. BACKGROUND REVIEW AND LITERATURE SURVEY

2.1 COMPARISON OF EXISTING SYSTEM AND PROPOSED SYSTEM

Existing System	Proposed System
Rumour querying websites such as snopes.com, twittertrails.com are available.	Automated system using unsupervised approach is proposed.
Rely on public reports and manual check for identifying whether tweet is a rumour or not.	LSTM model automatically detects whether tweet is a rumour or not.
Takes long time to detect rumour.	Early detection is possible.

Table 1: Comparison of Existing System and Proposed System

2.2 LITERATURE SURVEY

Ma et al. [4] was first to apply RNN for finding rumours. It was noticed that an event comprises of the original message or tweet and some other messages like comments related to the event and reposts of the messages and this created a continuous stream of messages. Therefore, to extract the features efficiently from these messages, they were batched into time intervals with variable length, and contemplate these as a unit. The proposed model evaluates the RNN-based model using three recurrent units namely GRU, LSTM and tanh. GRU as well as LSTM shows potential to remarkably capture the long-term dependencies of messages and the performance of the detection model is high in the initial stage. It was also observed that as the user's post changes with respect to time. The change was observed in textual features of forwarding contents over time. Therefore, it becomes crucial to find which features are salient for detection purpose. The model used labeled data for detection of rumours and the future work will consists of developing an unsupervised model that will use unlabeled data as the social media generates a large amount of unlabeled data.

Chen et al. [5] proposed a model which uses deep-attention for early identification of rumours. The model uses RNN for identifying rumours. The model CallAtRumours(Call Attention to Rumours) detects rumours by learning temporary hidden characteristics and representations from the messages. Initially the model batches the continuous streams of messages into variable length units of time series and then soft-attention mechanism is incorporated into recurrence to draw out distinct features and shun duplicity. In each timestep, the hidden features will be assigned a weight parameter that determines its importance for detection process i.e. the unrelated words are disregarded whereas the words expressing user's emotions like rage, irritation or uncertainty are given more weightage. The future work is to inspect the probability of assimilating complex features with the detection model and study the problem of efficiency by using hashing techniques.

Weiling Chen, Chai Kiat Yeo, Chiew Tong Lau, Yan Zhang, Bu Sung Lee [6] proposed a detection model which uses combination of RNN and Autoencoder. The user's behaviour is seen to vary while commenting on a rumour message and on a genuine message, this is reported with the use of comment-based features. RNN helps in analyzing the features which change over time. The time-dependent and time-independent properties are merged and passed to an autoencoder for identifying rumour. This unsupervised model built using combination of RNN and autoencoder helps in achieving good accuracy but the model fails in taking into consideration the diffusion structure of the spreading process and the dynamic variation amongst spreaders.

Jing Ma, Wei Gao, Kam-Fai Wong [7] proposed a model namely RvNN, which is a type of tree structured neural network that links content's semantics and dissemination hints. Various classes of rumours are classified using neural networks to learn the distinct features of tweets. The distinct features are learned by following the dissemination structure of a tweet which happens to be non-sequential in nature. Two models were proposed for finding rumours namely, topdown-RvNN and bottomup-RvNN. Valuable results were produced by capturing the textual as well as structural traits of the tweet which indicated rumours. The model achieved early

identification of rumours but it failed to notice the dynamic differences between the dissemination structure and the spreaders. In future, more features like spreader's information, etc. will be consolidated with the neural structures to enhance accuracy.

Oluwaseun Ajao, Deepayan Bhowmik, Shahrzad Zargari [8] provides a structure that identifies rumoured messages and further classifies them by employing combination of LSTM and CNN models on Twitter. CNN provides a pooling function which assists in circumventing over-fitting and lowers the cost of detection by lowering the dimensionality. The model used texts and images to learn in an unsupervised manner and classifies the tweet accordingly. The model attained accuracy of 82% on a dataset called PHEME. Due to inadequate training data, the performance of the model was impeded. In future, efforts to discover the origin of rumours will be made.

N. Ruchansky et al. [9] proposed a model that combined the properties like the message, the response and the source of message for accurate results and predictions. The behavior of the spreaders was monitored to detect rumours efficiently. Using the motivation of the properties mentioned above, a CSI model was proposed which consisted modules namely, capture, score and integrate. RNN is used to capture the time-related patterns of messages and comments of the user's activity on the related event. The following module learns from the behavior of source and its characteristics with the help of neural networks and an implicit user graph which then draws out the representations and allocate a score to all the users participating in the event to prompt the rumoured message. Ultimately, the last module integrates the message, comments, response and source's information to classify if the event is a rumoured event or not. The disadvantage of this model is it considers only the textual information.

Tian Lan, Chen Li and Jianxin Li [12] proposed a model called HARRD(Hierarchical attention RNN model for rumour detection). The rumours are batched according to the timestamps and a Bidirectional-GRU combined with the hierarchical attention apparatus is employed to learn the representations of rumour. The model ignores the noisy, unimportant or irrelevant words and pays attention to the most important,

unique and informative words related to the event. The efficiency of model was validated using two datasets namely, Weibo and Twitter and the model demonstrates early identification of the rumours. The model focuses on the post level as well as the word level to extract salient features for detection purpose. In future, the aim is to extract features from images and add complex traits to enhance accuracy.

Feng Yu, Liang Wang, Shu Wu, Tieniu Tan, Qiang Liu [11] proposed a model namely, CAMI which uses the convolutional approach which identifies the misinformation. The model observes the properties of the rumoured and non-rumoured events by investigating the datasets and then splits the dataset into several categories. CAMI extracts the local-global and scattered salient features from the input messages. Paragraph vector is used by the model which is an unsupervised technique and it helps in learning the fixed-length characteristics from the variable-length messages. The CNN helps shrink the dimensionality of the matrix and minimize the cost.

William Yang Wang [12] put forward a design of a hybrid CNN that merges meta-data and text which leads to remarkable enhancement for detecting close-grained fake news. This paper also introduces a dataset called LIAR, for detecting rumours. This dataset can be easily accessed by users as it is a public dataset. This dataset comprises of short messages which are manually labeled in diverse areas like economy, healthcare, politics, education, etc. Moreover LIAR dataset can be used for political natural language processing, argument mining, rumour detection, topic modeling and stance classification.

Wadii Boulila, Abdullah Alsaedi, Muna Al-Harby , Junaid Qadir, Mohammed Al-Sarem [13] carried out a review for detection of fake news by employing the deep neural networks. Study materials sources like ACM Digital Library, IEEE Explore, etc. were utilized to recover important study material. To find answers to the research questions, the trends and the patterns in the learning techniques for fake news detection were comprehended. This review presents a short introduction of various learning models like LSTM, ANN, autoencoder, etc. and briefs about available datasets for rumour detection.

3. REQUIREMENT AND ANALYSIS

3.1 PROBLEM STATEMENT

Twitter uses “tweets” as a medium to communicate on the social media platform. These tweets carry some implicit hints which are useful for separating a rumour and a non-rumour. Several tweets can be related to an occasion or an event. The detection model can be trained and tested on these occasions. The project aims in determining whether the occasion is a rumour or not. The administrator is immediately informed about the credibility of the occasion which the administrator can easily search in the GUI of the model.

3.2 REQUIREMENTS SPECIFICATION

3.2.1 Functional Requirements

This section describes the functional requirements of the project. The functional requirement states the functions which the system must perform. These requirements aids the user capture the conduct of the system.

- To check the credibility of tweet.
- Authentication of administrator.
- Administrator should get all the tweets as per mentioned keyword.

3.2.2 Non-Functional Requirements

This section describes the non-functional requirements of the project. The non-functional requirements states the standards to which the systems must perform. They can also be used for imposing restrictions on the system.

- The model should detect rumour within minutes, i.e. the response time of the model’s functions should be less.
- The performance of model should be good. i.e. the accuracy should be high
- The model should be available 24X7.
- Only administrator should be allowed to retrieve tweets with respect to the tweet or mentioned keywords.

- The administrator should be available 24X7.
- The software should be easy to use with easy graphics user interface.
- The system should be easy to maintain.

3.3 SOFTWARE AND HARDWARE REQUIREMENTS

3.3.1 Software Requirements

Operating System	Windows 10
Data Mining Tool	Python 3.6
Cuda Version	10.1
Libraries	Tensorflow –gpu – 2.1.0 Keras – 2.3.1 Tweepy – 3.8.0 Sklearn – 0.22.1 Matplotlib – 3.1.3 Django – 3.0.5
IDE	Pycharm
API	Twitter API
Model	Word2Vec

Table 2: Software Requirements

3.3.2 Hardware Requirements

System	Intel core i5
Hard Disk	1 TB
RAM	8 GB
GPU	Nvidia GTX Series

Table 3: Hardware Requirements

4. DESIGN

4.1 DATA FLOW DIAGRAMS

A data flow diagram (DFD) is the diagram which gives the graphical depiction of the flow of data in a system or a process. For creating a synopsis of the system, DFD are commonly used in the primitive stage of development of the system. Basically, DFD provides the details of what should be the input and the output of each unit of the system. A DFD does not show any decision rules or loops like the traditional flowchart. Figure 2 shows Level 0 Data Flow Diagram.

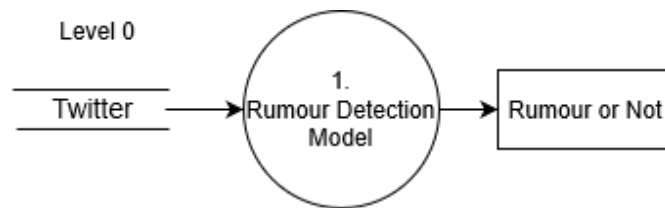


Figure 2: DFD Level 0

Figure 3 shows Level 1 Data Flow Diagram.

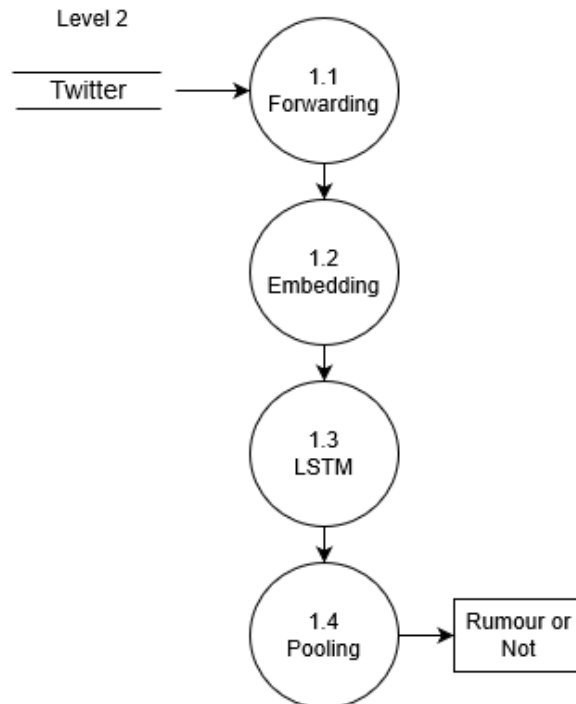


Figure 3: DFD Level 1

Figure 4 shows Level 2 Data Flow Diagram.

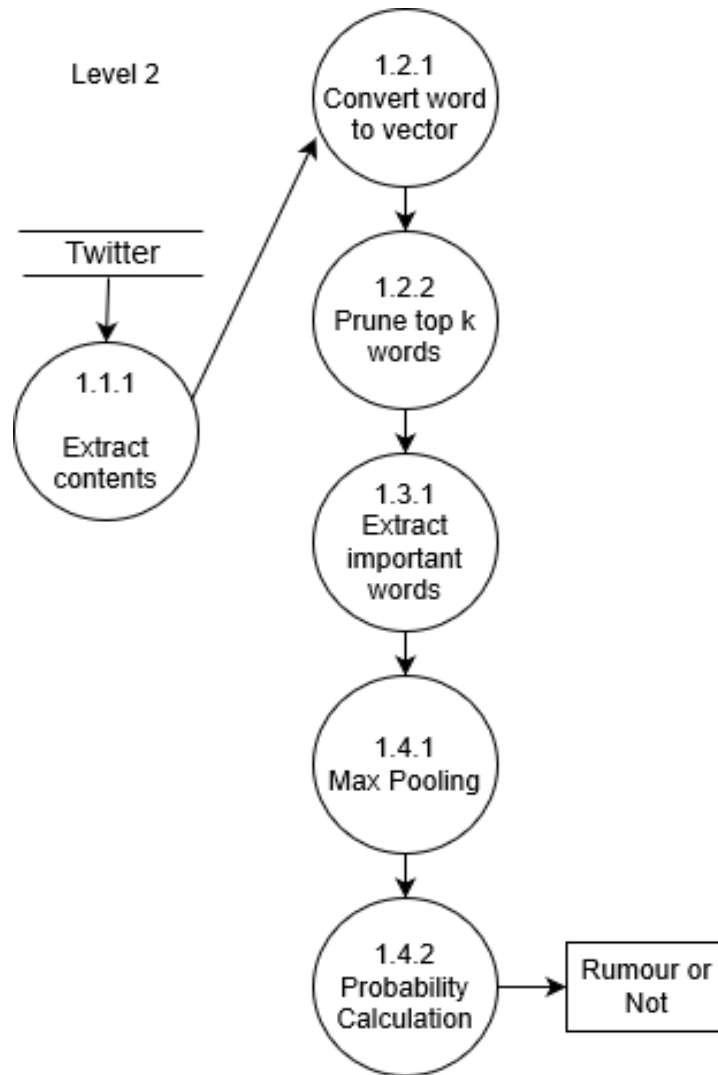


Figure 4: DFD Level 2

4.2 UML DIAGRAMS

4.2.1 Use Case Diagram

Use case diagram represents user's interaction with the system. It depicts the functional requirement of the proposed system using actor and use cases. Use case offers a detailed description of how the system is used. This diagram is used in primary phase of software system design to define input and output from proposed software system. Figure 5 exhibits actors like pre-processor, detection model, user, etc. and their functionalities like conversion of raw data, preparation of dimensional vector, detecting rumour, etc. that interact with rumour detection system.

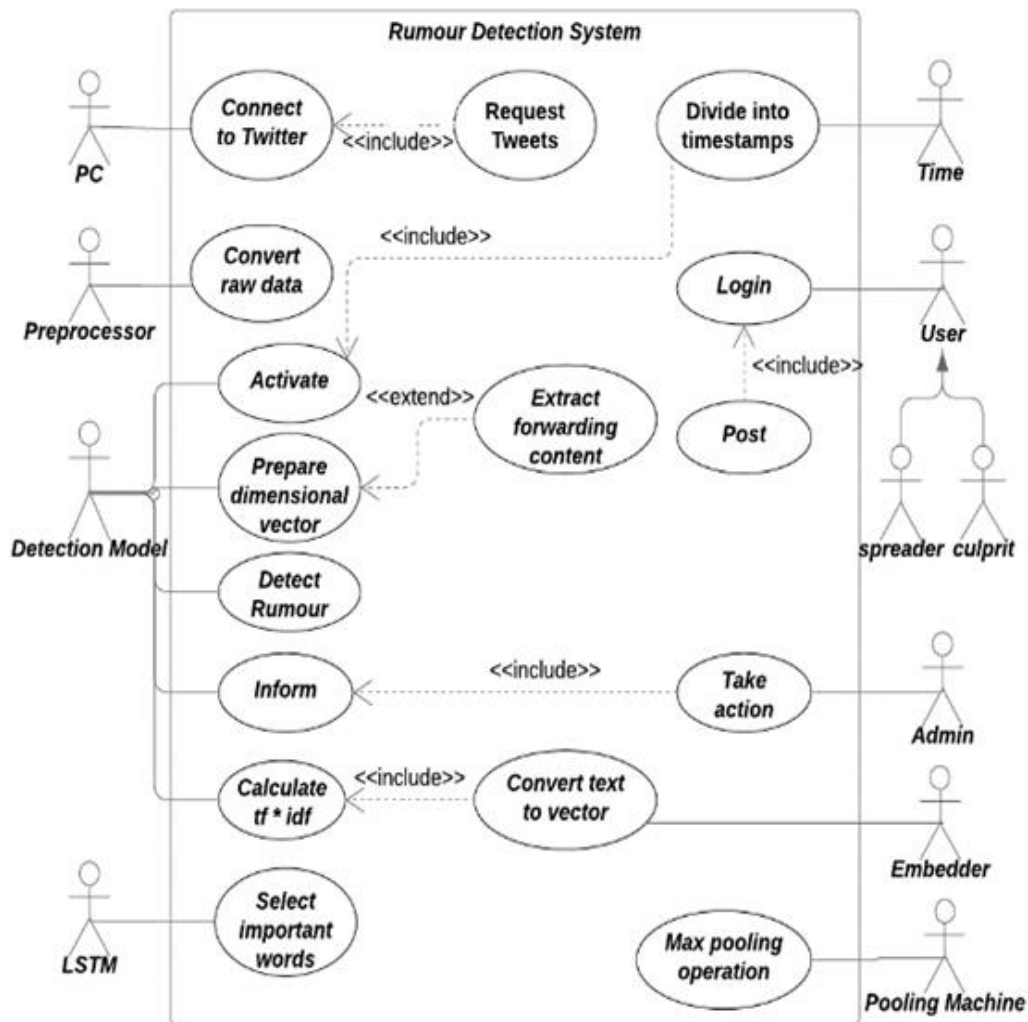


Figure 5: Use Case Diagram for Rumour Detection System

4.2.2 Class Diagram

Class diagram represents a group of classes, interfaces their interrelationships and collaboration of classes. It is used for modeling the static design archetype of system. Class diagram effectively shows how things are organized collectively in given scenario. They are very useful and therefore suitable for complete execution, specification and documentation of a system. Figure 6 shows classes like detection model, embedder, LSTM, etc along with their attributes and operations.

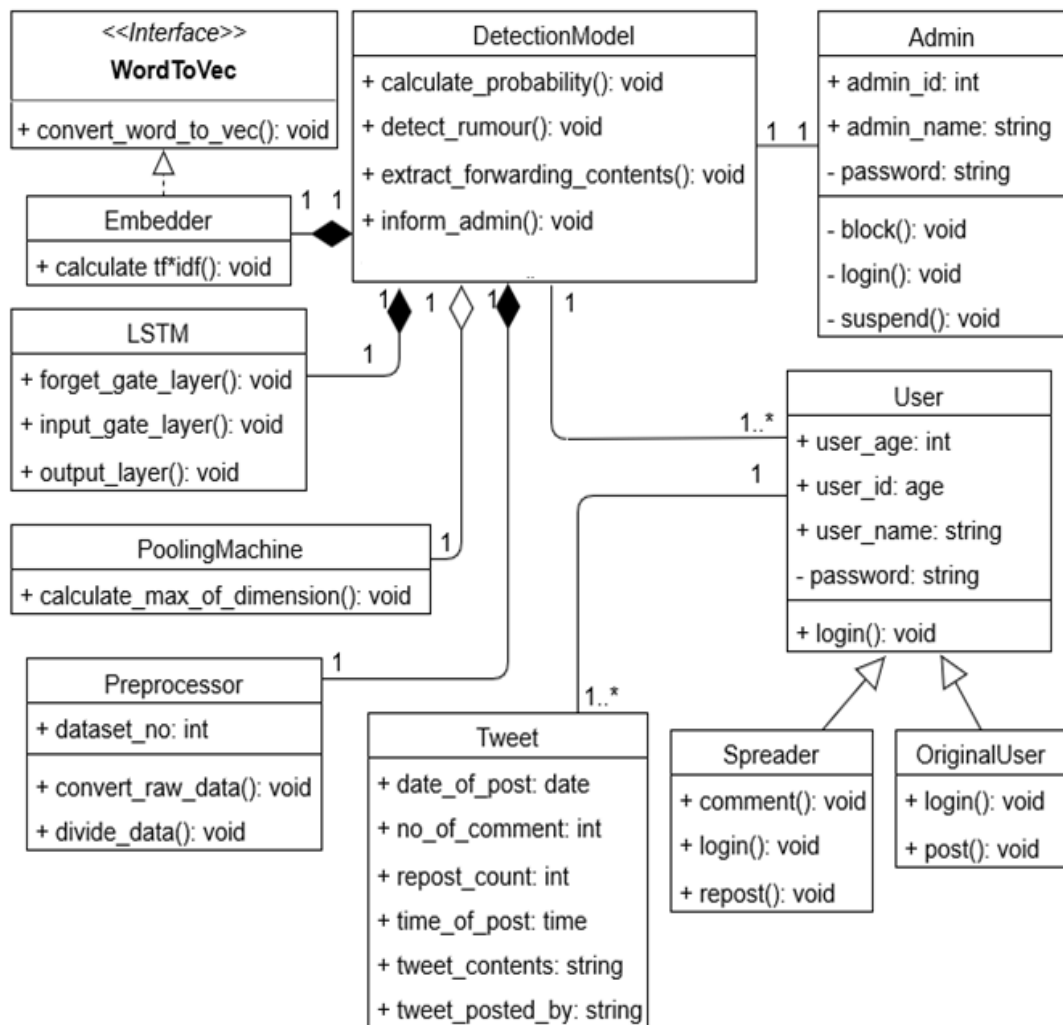


Figure 6: Class Diagram for Rumour Detection System

4.2.3 Sequence diagram

The sequence diagram is meant for showing communication and coordination among two or more objects. It primarily focuses on time ordering of messages and for the same it uses object timeline. It is meant for modelling dynamic aspect of software system. The objects present in sequence diagram are not only instances of class but they can be instances of nodes, components and collaborations involved in scenario. Figure 7 shows objects like detection model, embedder, LSTM, etc and interaction between those objects.

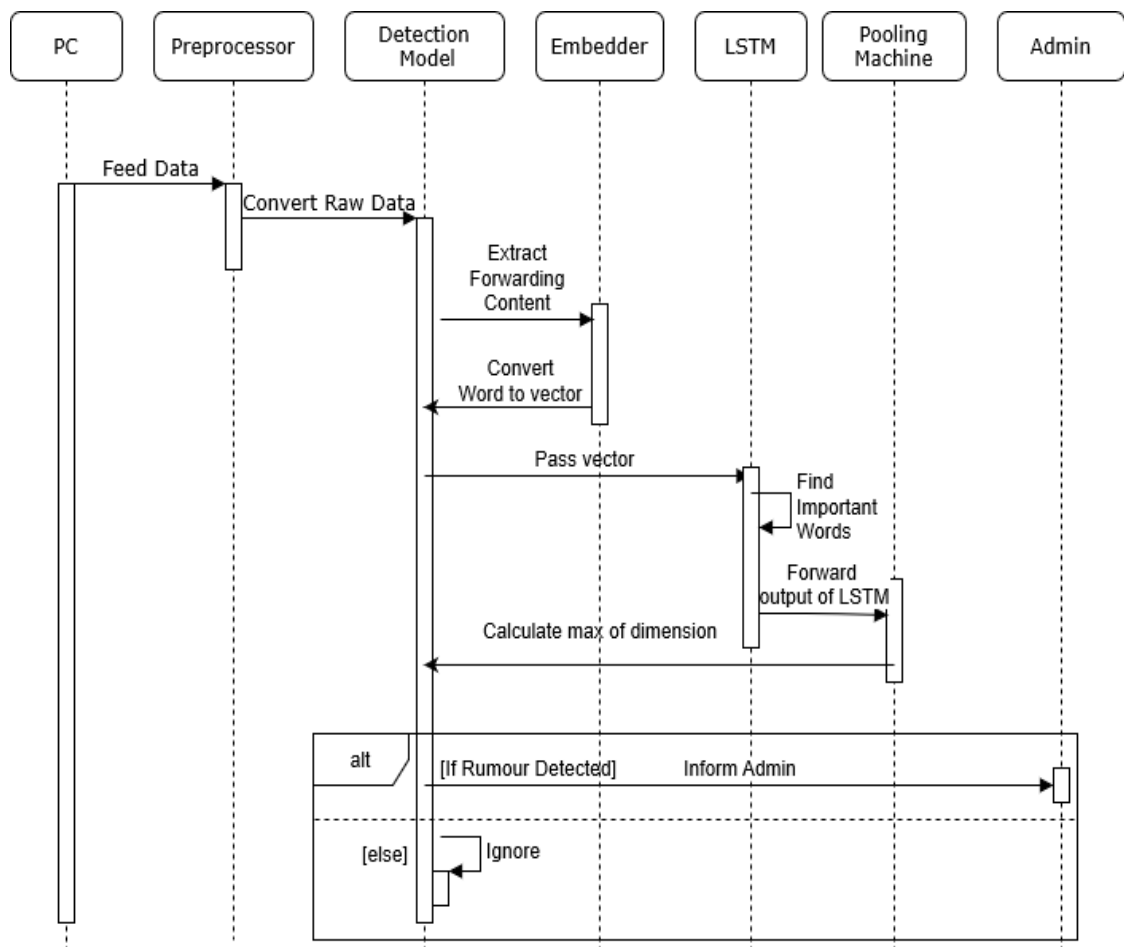


Figure 7: Sequence Diagram for Rumour Detection System

4.2.4 Activity Diagram

An activity diagram exhibits flow of control in a system. These diagrams are generally utilized for modeling business processes. Activities can be modeled as concurrent or sequential. The activity diagram has an initial state, activities that are to be performed by the system and a final state. It is a crucial behavioral diagram available in UML diagrams. Figure 8 shows the flow of control of rumour detection system where the process starts from PC and it does the activity of feeding data to preprocessor, which then converts this raw data to detection model for further processing. embedder, LSTM, pooling machine helps find the rumour correctly.

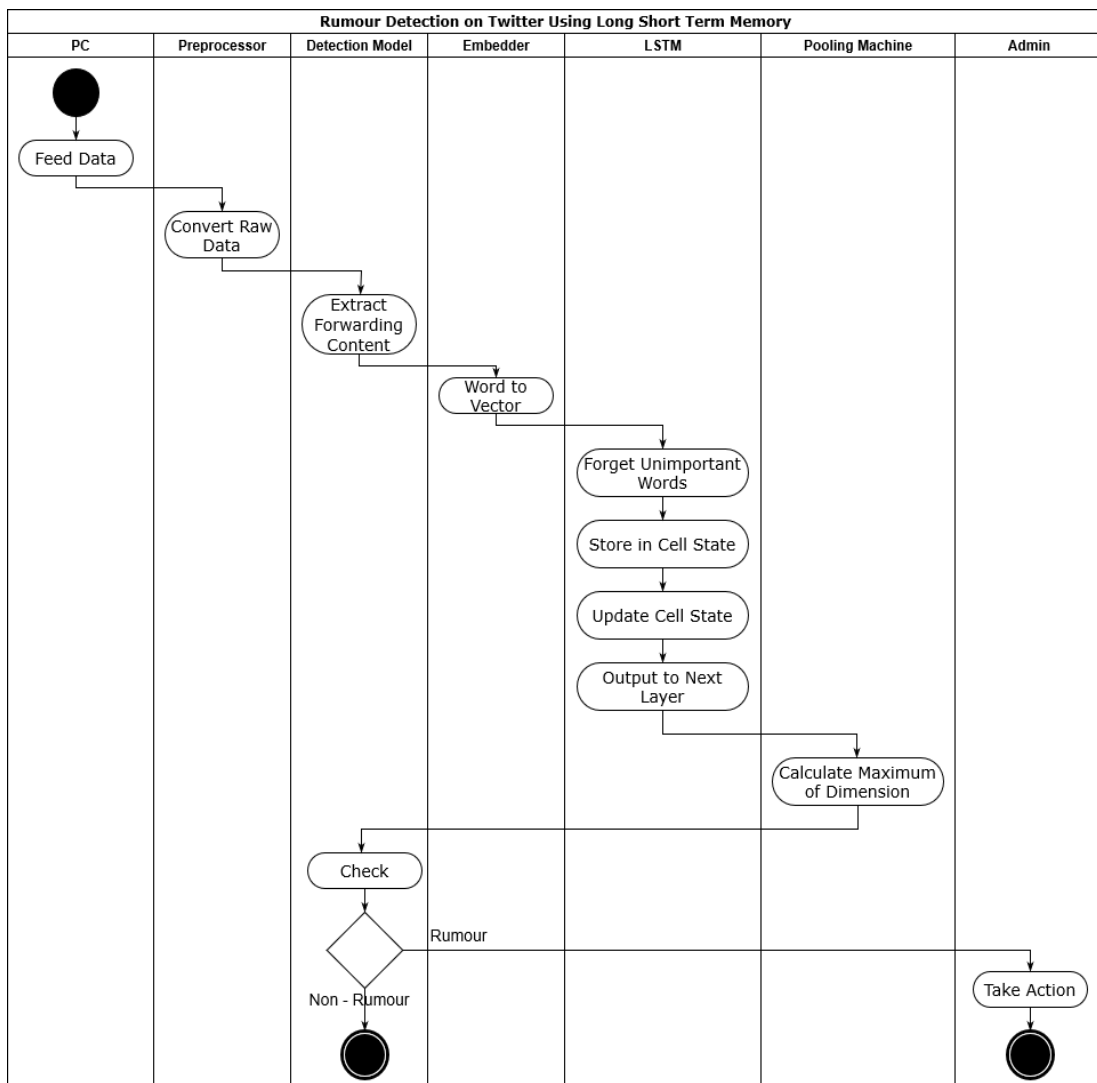


Figure 8: Activity diagram for Rumour Detection System

4.2.5 State Diagram

A state diagram is a vital behavioral diagram available in UML which represents the state or condition of the system at finite time. The behavior of system is described using finite state transitions. These diagrams are also referred as State-chart Diagrams and State machines. Figure 9 shows the dynamic behavior of the rumour detection model.

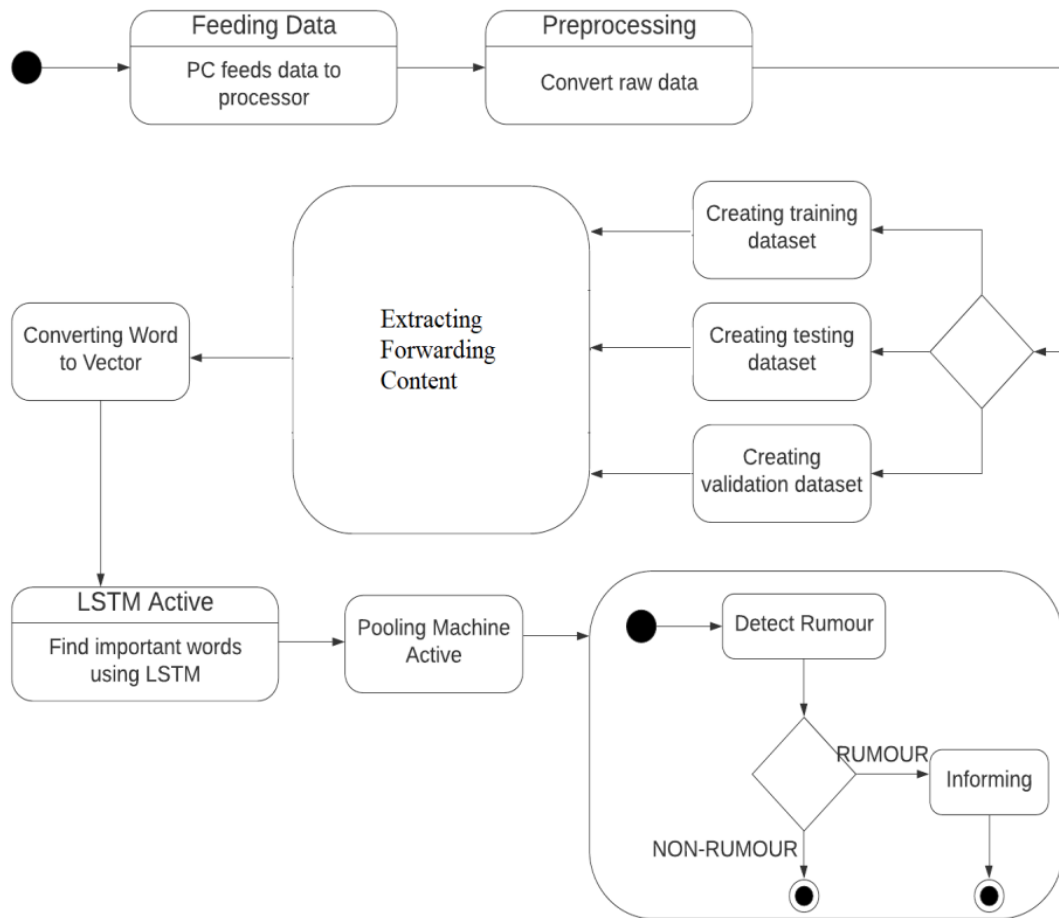


Figure 9: State Diagram for Rumour Detection System

4.3 Functionality of System:

- Preprocessor

- 1) Convert raw data by filtering according to machine requirement.
- 2) Divide dataset into training, testing and validation sets.

- PC

- 1) Request database administrator to connect to microblogging sites.

- Model

- 1) Activate system and divide datasets according to timestamps.
- 2) Prepare dimensional vector by extracting forwarding contents.
- 3) Calculate probability.
- 4) Detect if microblog is rumour or not.
- 5) If rumor, inform the administrator.

- Embedder

- 1) Calculate $tf*idf$ (Term Frequency – Inverse Document Frequency) i.e weight used to evaluate how important a word is to document in a collection.
- 2) Prune the top k words.
- 3) Calculate input to LSTM.

- LSTM

- 1) Forgets irrelevant information and keeps only important information.

- Pooling Machine

- 1) Calculate maximum of dimension.
- 2) Avoids overfitting.

- Database Administrator

- 1) Authenticates PC.
- 2) Provides database to PC.

- User

- 1) User has to login for accessing Twitter.
- 2) User can post tweets.
- 3) User can forward messages with or without commenting.

4.4 BLOCK DIAGRAM OF SYSTEM

A rumour is a doubtful and unverified statement, which on investigation, can come out as a legitimate fact or an actual misinformation (rumour). The system aims in debunking actual misinformation. The misinformation can be deliberately fabricated to mislead public or it can emerge out of a misunderstanding. Whatever maybe the cause of the diffusion of misinformation, it ultimately harms society emotionally and financially. To avoid confusion regarding the rumour coming out as a legitimate fact or an actual misinformation, the later sections follows the convention of referring a legitimate fact as “non-rumour” and an actual misinformation as “rumour”.

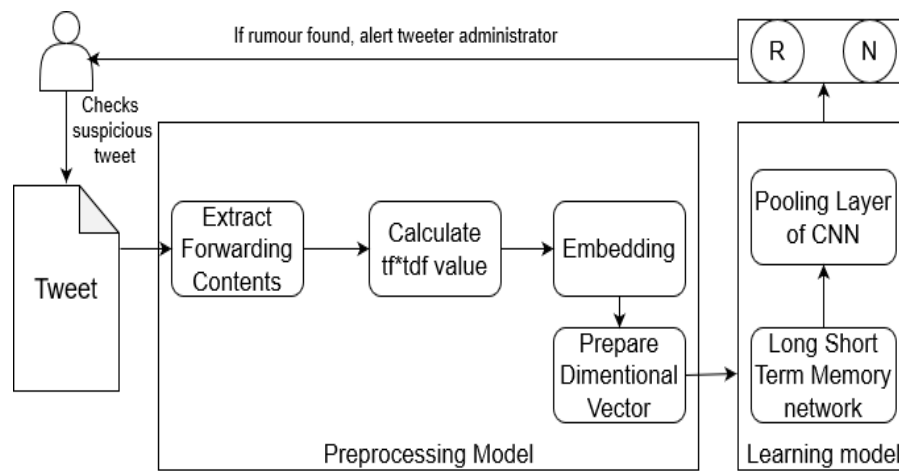


Figure 10: Block Diagram of Rumour Detection System

Figure 10 displays the block diagram for proposed rumour detection model. If the administrator comes across a dubious tweet, then this tweet can be fed to the model for inspection of its credibility. Using the information gained by extracting forwarding contents of the tweet, the results are produced. The model is divided into two phases namely, the processing phase and the learning phase.

Preprocessing phase: It is observed that the tweet’s content gets altered over the course of time. In this phase, extraction of the key features of the forwarding content is the major goal. To do the needful, first, the common and unimportant words in the tweet are given less weightage by calculating the $tf*idf$ values. These values help in determining the stop words which are then eliminated. Next is the embedding process which is prominently used for the purpose of transforming a word to its vector

equivalent. The neural network like LSTM requires a numeric data for processing. As the LSTM cannot process words directly, embedding plays a major role in mapping the words to their respective vector values. To perform this mapping, a shallow two-layered neural network called Word2Vec model is used to create word embedding for representing words as numeric values. A dimension D_c is created from the vector of vocabulary and the $tf*idf$ values and fed to the LSTM in the next phase.

Learning phase: this phase comprises of the LSTM and the pooling function of CNN. LSTM aids in capturing key features that will help distinguish rumours and non-rumours. LSTM overcomes the drawback of traditional neural networks as it captures the long-term dependencies of the tweets. LSTM comprises of gates and memory cell state s_t which controls the flow of information. The output h_t of the LSTM unit can be computed using the below equations [6].

$$i_t = \sigma(U_i h_{t-1} + W_i x_t + V_i s_{t-1} + b_i) \quad (1)$$

$$f_t = \sigma(U_f h_{t-1} + W_f x_t + V_f s_{t-1} + b_f) \quad (2)$$

$$\bar{s}_t = \tanh(U_s h_{t-1} + W_s x_t + b_c) \quad (3)$$

$$s_t = f_t x_{t-1} + i_t \bar{s}_t \quad (4)$$

$$o_t = \sigma(U_o h_{t-1} + W_o x_t + V_o s_t + b_o) \quad (5)$$

$$h_t = o_t \tanh(s_t) \quad (6)$$

Where,

x_t = input (forwarding content)

\tanh and σ (logistic sigmoid) = activation functions.

f_t = forget gate (removes unwanted information from the cell state)

i_t = input gate (keeps information in the cell state)

s_t = memory cell state

\bar{s}_t = new memory cell state

o_t = output gate

h_t = output of LSTM model

To downsample the input matrix and to efficiently capture the hidden clues, pooling function is used. Pooling function proves beneficial acquiring influential local information. Lastly, the output of model is displayed as “rumour” or “non-rumour”.

4.5 LONG SHORT-TERM MEMORY

LSTM is an artificial RNN architecture which is extensively used in the deep learning domain. The standard neural networks are generally feed forward networks but LSTM used feedback networks. LSTM is useful in various fields because it can be used for processing single data points like an image or a complete sequence of data like a video or speech. LSTM was once stated by a business magazines called Bloomberg Businessweek as the greatest commercial artificial intelligence achievement which can be used for almost everything, from composing music to predicting diseases.

LSTM networks can be used for processing, making predictions and classifications. LSTM classifies and makes predictions based on the time series data because there can be idle time for unknown duration between significant events. LSTM also deals with the vanishing and exploding gradient which were the problems found in training the conventional RNNs.

LSTM Architecture –

LSTM's architecture comprises of the regulators called gates, namely forget gate, input gate and output gate to control the flow of information and cell (it is the memory part which audits the dependencies of the elements present in the input). The forget gate regulates the amount of unwanted values which can be discarded, the input gate regulates the extent to which important values should be kept and lastly the output gate regulates to amount of information to be outputted by the LSTM unit to meet the purpose of the problem. The gates basically use the activation functions like 'logistic sigmoid function' or 'tanh function' to regulate the information. Weights are assigned to these connections during training to determine the operations of gate.

Core Concept

The core of LSTM lies in its gates and cell state. The cell state reserves memory from previous time steps and this memory is easily transported and used for processing the next time step. This helps the network to build long-term dependencies and make

accurate predictions. The gates decide which information stays and other irrelevant information gets discarded. The gates learn which information to keep or forget irrelevant information during the training. Every LSTM unit is fed with the current time step's input, previous LSTM unit's output and previous LSTM unit's memory. The unit generates new output and also alters the memory.

Working of LSTM

STEP 1: Forget Gate

In first step, LSTM identifies which information is useless and is discarded from the cell state. The decision of forgetting irrelevant and useless information is made by the activation function called sigmoid function. In the forget gate layer, information from the current input and previous hidden state is given as an input to the sigmoid function. The function outputs value between 0 and 1 where 1 means to keep the information completely and 0 means to forget it completely.

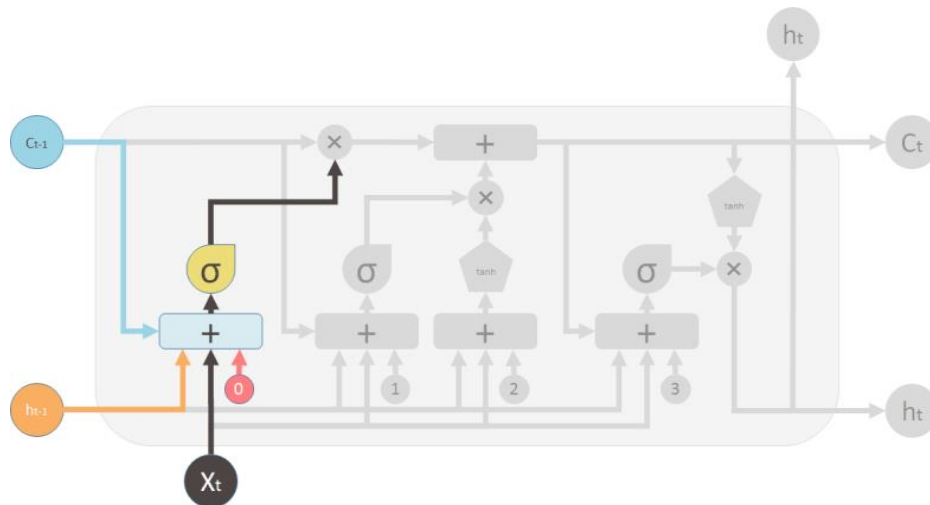


Figure 11: Forget Gate Layer

$$f_t = \sigma(w_f[h_{t-1}, x_t] + b_f)$$

Where,

w_f = Weight

h_{t-1} = Output from previous time stamp

x_t = New input

b_f = Bias

STEP 2: Input Gate

In the second step, the input gate layer decides which information to reserve in cell state. The current input and previous hidden state is fed to the sigmoid function and tanh function. The sigmoid function keeps only the relevant information by assigning it 1 or values near to 1 and values near to 0 are discarded. The tanh function squishes values between -1 and 1 for regulating network and it also creates a vector of new candidate values, which could be added to the state. The input gate layer is the point wise product of output of sigmoid and tanh function where the sigmoid function decides which information to keep from the tanh's output.

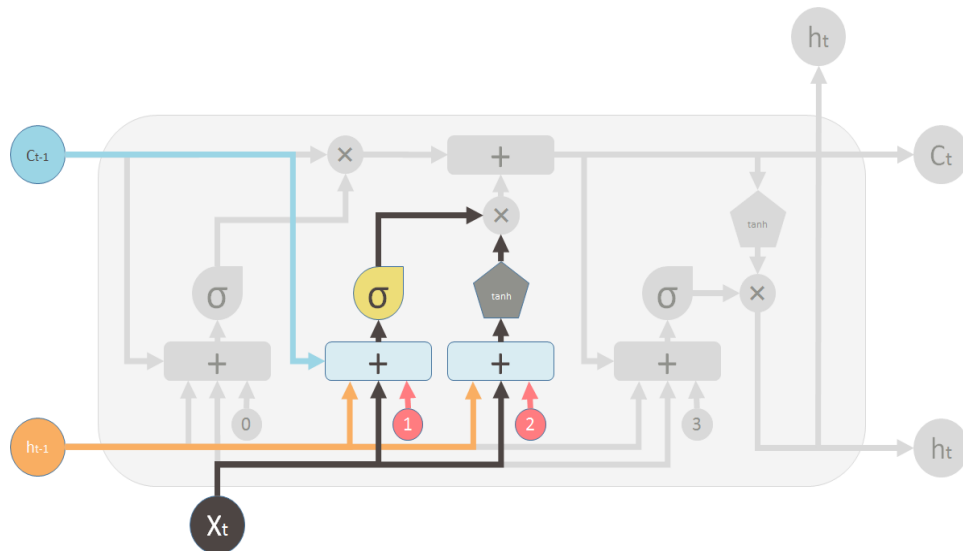


Figure 12: Input Gate Layer

$$i_t = \sigma(w_i[h_{t-1}, x_t + b_i])$$

$$\bar{s}_t = \tanh(w_s[h_{t-1}, x_t + b_s])$$

Where,

w_i = Weight

h_{t-1} = Output from previous time stamp

x_t = New input

STEP 3: Cell State

In this step, the old state s_{t-1} is point wise multiplied by f_t (forget vector). Basically, the forget gate controls the flow of the old memory to pass through for further processing. Further, the output from the input gate is point wise added with the $s_{t-1} * f_t$. This gives the new candidate values, which is scaled by upto what extent it decides to update each state value. This results in new cell state.

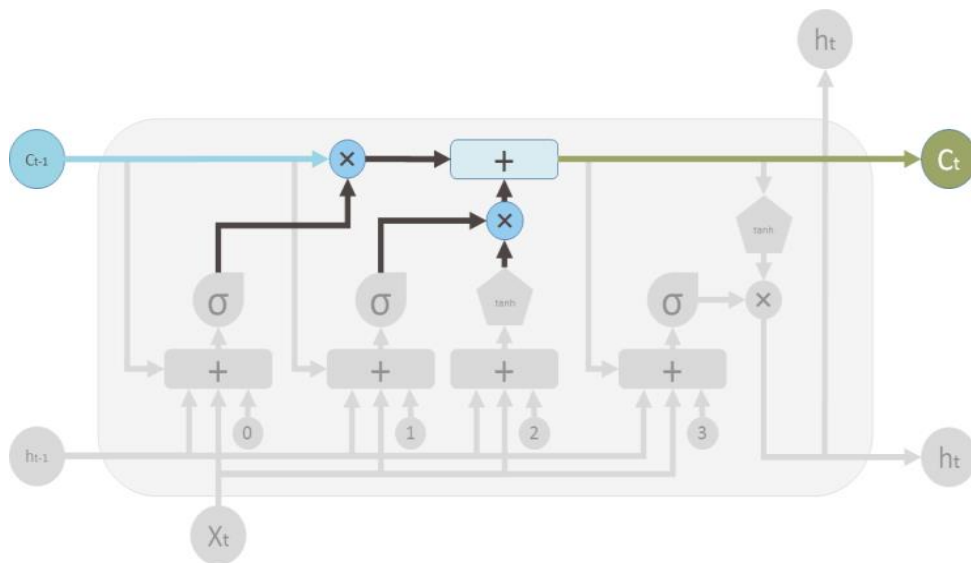


Figure 13: Cell State

$$s_t = f_t * s_{t-1} + i_t * \bar{s}_t$$

Where,

s_t = new cell state

f_t = Output of forget gate

\bar{s}_t = Vector Of candidate values

i_t = Output of input gate

STEP 4: Output gate

The output gate determines the next hidden state. First, the current input, the previous hidden state and the newly modified cell state are passed into a sigmoid function. The new modified cell state is again passed to tanh function and the final output is obtained by point wise multiplication of the tanh function's output and sigmoid function's output to decide what information the hidden state should carry. The output is the hidden state. The new hidden output and the new cell state are passed to the next time step.

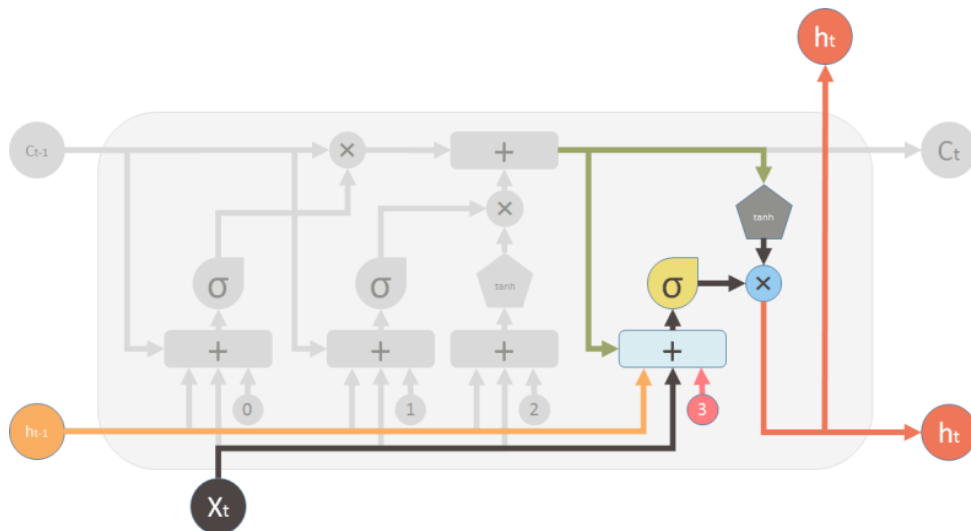


Figure 14: Output Gate Layer

$$o_t = \sigma(w_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(s_t)$$

Where,

o_t = Output of output gate

h_t = Final output of current timestamp

5. IMPLEMENTATION

5.1 SYSTEM ARCHITECTURE

5.1.1 Problem Statement

Twitter uses “tweets” as a medium to communicate on the social media platform. These tweets carry some implicit hints which are useful for separating a rumour and a non-rumour. Several tweets can be related to an occasion or an event. These events are collected together and are defined as $O = \{O_i\}$, where each event is represented by $O_i = \{(c_{i,j}, t_{i,j})\}$. Each event consists of tweets $c_{i,j}$ posted at time $t_{i,j}$, where $j=0$ denotes the original tweet $c_{i,0}$ posted at time $t_{i,0}$ by the user. The project aims in determining whether the event is a rumour or not.

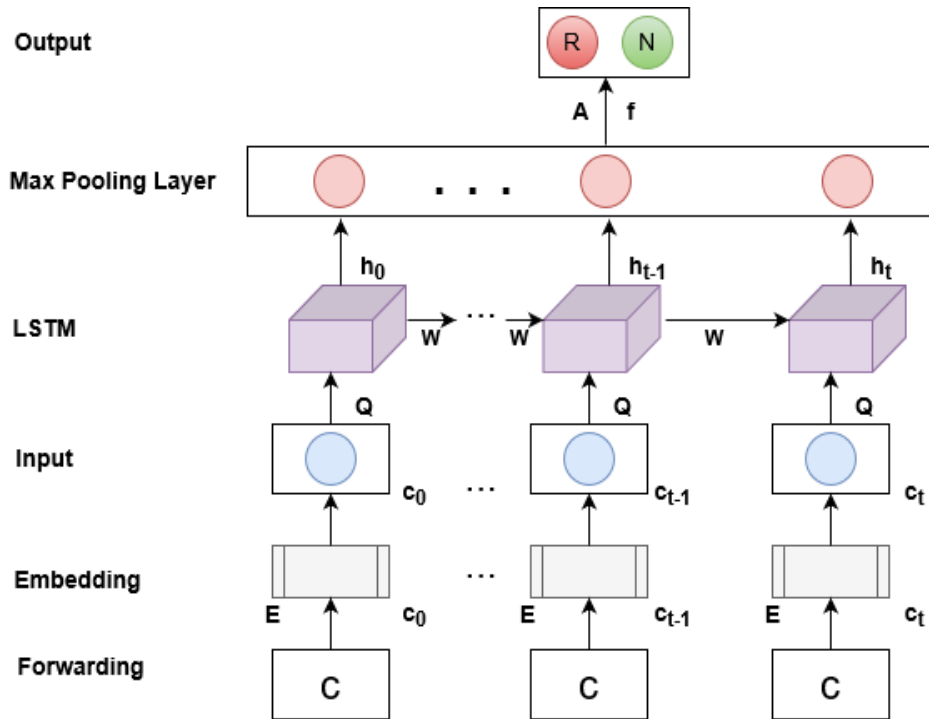


Figure 15: The LSTM-based Rumour Detection Model

5.1.2 Variable Length Time-Series

Each event has many tweets but only a single output neuron is used to represent the class of events. This process is expensive as well as carrying out back-propagation through many time steps is ineffective and results in final stage loss. To overcome

this drawback, the tweets are first batched into varying length time steps and then modeled using LSTM. Time-span which consist large amount of tweets is useful and their spreading should be captured with precision. The procedure to formulate varying length time series is described in Algorithm I.

At first, the total timeline is divided in equal timesteps L , where L is the reference length. Some timesteps may have large amount of tweets, some timesteps may have few number of tweets while some timesteps may not have even a single tweet. These empty timesteps denoted by U_0 from the set U_r to get a set of non-empty timesteps U'_r . Furthermore, from a set of continuous timesteps with largest comprehensive time-span is selected from U'_r and stored in \bar{U}_r . If the timesteps in set \bar{U}_r is lesser in number than L and more than that of previous round, timesteps are halved and are continued to partition; or else, the continuous timesteps is returned that is given by set \bar{U}_r .

```

Input : Tweets of incident  $O_i = \{(c_{i,j}, t_{i,j})\}_{j=1}^{m_i}$ ,
          Reference length of LSTM  $L$ 
Output: Time steps  $S = \{S_1, S_2, \dots\}$ 

 $P(i) = t_{i,m_i} - t_{i,1}; p = \frac{P(i)}{L}; r = 0;$ 
while true do
     $r++;$ 
     $U_r \leftarrow \text{Equipartition}(P(i), p);$ 
     $U_0 \leftarrow \{\text{empty intervals}\} \subseteq U_r;$ 
     $U'_r \leftarrow U_r - U_0;$ 
    Find  $\bar{U}_r \subseteq U'_r$  so that  $\bar{U}_r$  consists of continuous timesteps
    which cover the largest time-span;
    if  $|\bar{U}_r| < L \ \&\& \ |\bar{U}_r| > |\bar{U}_{r-1}|$  then
        /* Halve the timestep */
         $p = \frac{p}{2};$ 
    else
        /* Give output */
         $S = \{S_o \subseteq \bar{U}_r / S_1, \dots, S_{|\bar{U}_r|}\};$ 
        return  $S;$ 
    end
end
return  $S;$ 

```

Algorithm I: Algorithm to construct time series of variable length.

5.1.3 Model Structure

The timeseries formed using the Algorithm I is suitable for LSTM's recurrent units. The dimensional vector D_c is generated in each timestep.

The LSTM is fed c_t as an input. LSTM keeps only the key information and discards the rest irrelevant information. Using pooling function of CNN the implicit hints in the posts are extracted. Therefore, max pooling is used to improve accuracy and performance.

$$f_m = \max_{i \in 0 \dots t} \{h_{i, m}\}, \quad (7)$$

Equation (7) shows use of max pooling where the maximum value of h_i is selected from all the dimensions. f_m is calculated by max pooling the m^{th} dimension, i goes from $0 \dots t$, and m is the m^{th} dimension of h_i . Softmax layer helps in acquiring the probabilistic output of two classes, namely rumour and non-rumour.

$$V = Af + b_v, \quad (8)$$

Softmax layer is an activation function and the input fed to it is shown in (8). Here, f is the output max pooling function, b_v is bias, A represents the weight matrix and V represents the hidden layer's output.

$$p_i = \frac{e^{V_i}}{\sum_{k=1}^n e^{V_k}}, \quad (9)$$

In (9), p_i represents the probability related to the class i and n tells the number of classes. The model outputs $[1,0]$ for rumoured class and $[0,1]$ for non-rumoured class.

5.1.4 Model Training

The model trains the parameters using backpropagation along with stochastic gradient descent (SGD) with mini-batch. Using Ada Grad algorithm, the parameters are updated in training. The error percentage between the actual value and predicted value should be reduced to improve accuracy. Here, the loss function called the cross entropy is shown by (10).

$$L(p, q) = - \sum_{i=1}^c p_i \log q_i + \lambda \|\theta\|_2^2, \quad (10)$$

Here, p , q is predicted and actual value respectively, λ corresponds to regularization coefficient and θ represents the model parameters. Ultimately, in each epoch, model iterates for all the training events and execution is carried out till it reaches the maximum epoch number or the until the loss value converges.

5.2 CODE

5.2.1 Code to extract tweets using twitter API and store it in the json document.

```
import tweepy
import json, os
from os import listdir
import tweepy
import re
import numpy as np

access_token = '1185203655705653249-MI55LF0YDXJHU06sATMMQtHjfWrkKF'
access_token_secret = '9PCjIq70gj0o8zHgrtkvr1K6vITo5H2P1TyQcfOqyh7B3'
consumer_key = 'kkEHLGQM2iRKAIRzh33eAQ2RN'
consumer_secret =
'pNa2PFNYUVWSnUPYUxAFPYbExsl3ABY1oLHBkPTtY1AZIJpFd7'
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
# Setting your access token and secret
auth.set_access_token(access_token, access_token_secret)
# Creating the API object while passing in auth information
api = tweepy.API(auth)
Evenlist =[]
with open("twitter\Twitter_event_claims.txt", "r") as f:
    for line in f:
        txt = line.split()[0][1:]
        eid = txt.split(":")
        Evenlist.append(eid[1])
counter =0

def remove_pattern(input_txt,pattern):
    r= re.findall(pattern,input_txt)
    for i in r:
```

```

    input_txt = re.sub(i, "", input_txt)
    return input_txt

def remove_link(input_text):
    pattern = re.compile('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+')
    tweetText = pattern.sub("", input_text)
    return tweetText

def remove_spchar(input_text):
    tweetText = re.sub('[^a-zA-Z#]+', '', input_text)
    tweetText1 = re.sub('[0-9]', '', tweetText)
    return tweetText1

def remove_emoji(input_text):
    emoji_pattern = re.compile("[
        u\"\\U0001F600-\\U0001F64F\" # emoticons
        u\"\\U0001F300-\\U0001F5FF\" # symbols & pictographs
        u\"\\U0001F680-\\U0001F6FF\" # transport & map symbols
        u\"\\U0001F1E0-\\U0001F1FF\" # flags (iOS)
        u\"\\U00002702-\\U000027B0\"
        u\"\\U000024C2-\\U0001F251\"
        "]+" , flags=re.UNICODE)
    return emoji_pattern.sub(r'', input_text)

credentials = {}

for event in Evenlist:
    try:
        fe = open(os.path.join("twitter1\\event", event+".json"), "r")
        EventJson = json.load(fe)
        TidList = EventJson["tid"]

```

```

os.mkdir(os.path.join("twitter1", event))
for TweetId in TidList:
    try:
        tweet1 = api.get_status(TweetId)
        text1 = remove_pattern(tweet1.text, "@[\w]*")
        text2 = remove_link(text1)
        text3 = remove_emoji(text2)
        text4 = remove_spchar(text3)
        credentials['text'] = text4
        Time = tweet1.created_at
        Time = int(Time.strftime("%Y%m%d%H%M%S"))
        credentials['time'] = Time

        print(os.path.join("twitter1", event, TweetId+".json"))
        with open(os.path.join("twitter1", event, TweetId+".json"), "w") as file:
            json.dump(credentials, file)
    except:
        continue
except:
    continue

```

5.2.2 Code to extract tweets from keyword and to store in a file

```

import tweepy
import numpy as np
import os

access_token = '1185203655705653249-
MI55LF0YDXJHU06sATMMQtHjfWrkKF' # access details of tweet

access_token_secret = '9PCjIq70gj0o8zHgrtkvr1K6vITo5H2P1TyQcfOqyh7B3'

consumer_key = 'kkEHLGQM2iRKAIRzh33eAQ2RN'

consumer_secret =
'pNa2PFNYUVWSnUPYuXaFPybExsl3ABY1oLHBkPTtY1AZIJpFd7'

auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)

api = tweepy.API(auth, wait_on_rate_limit=True,
wait_on_rate_limit_notify=True)

text1 = []
credentials = {}

for tweet1 in tweepy.Cursor(api.search, q=str(tweet), lang="en").items(count):
    credentials['text'] = tweet1.text
    text1.append(tweet1.text)
    Time = tweet1.created_at
    Time = int(Time.strftime("%Y%m%d%H%M%S"))
    credentials['time'] = Time
    with open(os.path.join("check\\event", str(var) + ".json"), "w") as file:
        json.dump(credentials, file)

```

5.2.3 Code to train the model

```

model = Sequential()
model.add(LSTM(CELL_SIZE, input_shape=(TIME_STEPS, INPUT_SIZE)))
model.summary()

model.add(Dense(OUTPUT_SIZE))
model.add(Activation('softmax'))
Adagrad = Adagrad(LR)
model.compile(optimizer=Adagrad, loss='mean_squared_error',
metrics=['accuracy'])
# train
print("Training-----")

history = model.fit(X_train, y_train, epochs=250, batch_size=BATCH_SIZE)
plt.plot(history.history['accuracy'])

plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.legend(['train','test'], loc='upper right')
plt.show()
plt.plot(history.history['loss'])
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['train','test'], loc='upper right')
plt.show()

print("\nTesting-----")
cost, accuracy = model.evaluate(X_test, y_test, batch_size=y_test.shape[0],
verbose=False)
print('test cost: ', cost)
print('test accuracy: ', accuracy)

```



```

y_pred= model.predict(X_test)
print(y_pred)
matrix = confusion_matrix(y_test.argmax(axis=1),y_pred.argmax(axis=1))
print(matrix)
plt.matshow(matrix)
plt.title('Confusion Matrix Plot')
plt.colorbar()
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
tn,fp,fn,tp =
confusion_matrix(y_test.argmax(axis=1),y_pred.argmax(axis=1)).ravel()

acc = (tn + tp)*100/(tp+tn+fp+fn)
print("Accuracy {:.2f}".format(acc))

recall = tp/(tp+fn)
print("Recall {:.2f}".format(recall))

f1 = (2*precision*recall)/(precision+recall)
print("F1 Score {:.2f}".format(f1))

filename='model.sav'
pickle.dump(model,open(filename,'wb'))

```

5.2.4 Code to check Rumour or non-rumour

```
import pickle
import re
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
import re
import json, os
import pandas as pd
from os import listdir
from matplotlib import pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
import numpy as np
from sklearn.metrics import confusion_matrix
from keras.layers import Flatten, TimeDistributed, MaxPooling1D
from keras.models import Sequential
from keras.layers import Dense, Activation, LSTM
from keras.utils import np_utils
from keras.optimizers import Adagrad
import gensim

model = gensim.models.KeyedVectors.load_word2vec_format('GoogleNews-
vectors-negative300.bin.gz', binary=True)
mypath = "check"
Evenlist = listdir(mypath)
TIME_STEPS = 6
InputWord = 15
INPUT_SIZE = 300 * InputWord
BATCH_SIZE = 30
BATCH_INDEX = 0
```

```

OUTPUT_SIZE = 2
CELL_SIZE = 175
LR = 0.001

def ContinuousInterval(intervalL): # Returns max continuous interval
    maxInt = []
    tempInt = [intervalL[0]]
    for q in range(1, len(intervalL)):
        if intervalL[q] - intervalL[q - 1] > 1:
            if len(tempInt) > len(maxInt):
                maxInt = tempInt
            tempInt = [intervalL[q]]
        else:
            tempInt.append(intervalL[q])
    if len(maxInt) == 0:
        maxInt = tempInt
    return maxInt

output_fin=[]
totalData = []
totalDataLabel = []
counter = 0
totalDoc = 0
totalpost = 0
tdlist1 = 0
Pos = 0
Neg = 0
maxpost = 0
minpost = 62827

def remove_link(input_text): # remove hyperlink
    pattern = re.compile('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[*\(\)\,]|(?:%[0-9a-fA-F][0-9a-fA-F]))+')

```

```

tweetText = pattern.sub(' ', input_text)
return tweetText

def remove_pattern(input_txt, pattern):          # remove handles
    r = re.findall(pattern, input_txt)
    for i in r:
        input_txt = re.sub(i, ' ', input_txt)
    return input_txt

def remove_spchar(input_text):                  # remove digits
    tweetText1 = re.sub('[0-9]', ' ', input_text)
    return tweetText1

def remove_emoji(input_text):                   # remove emojis
    emoji_pattern = re.compile("[
        u\"\\U0001F600-\\U0001F64F\"    # emoticons
        u\"\\U0001F300-\\U0001F5FF\"    # symbols & pictographs
        u\"\\U0001F680-\\U0001F6FF\"    # transport & map symbols
        u\"\\U0001F1E0-\\U0001F1FF\"    # flags (iOS)
        u\"\\U00002702-\\U000027B0\"
        u\"\\U000024C2-\\U0001F251\"
        "]+" , flags=re.UNICODE)
    return emoji_pattern.sub(r' ', input_text)

for event in Evenlist:
    totalDoc += 1
    print(event)
    try:
        fe = open(os.path.join("event", event + ".json"), "r")
    except:
        print(event + " not present in event folder")
        continue

```

```

EventJson = json.load(fe)
Label = EventJson["label"] # store label
path = os.path.join("check", event)
TidList = listdir(path)
TweetList = []

if len(TidList) == 1:                                # only one tweet id present
    tdlist1 += 1
    continue
if len(TidList) >= maxpost:
    maxpost = len(TidList)
if len(TidList) <= minpost:
    minpost = len(TidList)
for TweetId in TidList:
    event = re.sub('.json', "", event)
    ft = open(os.path.join("check", event, TweetId), "r")
    tweet1 = file1['text']
    text1 = remove_pattern(tweet1, "@[\w]*")
    text2 = remove_link(text1)
    text3 = remove_emoji(text2)
    text4 = remove_spcchar(text3)                    #preprocessed tweet

    totalpost += 1
    Time = file1['time']
    TweetList.append({"text": text4, "time": Time}) # time and text stored in
tweetlist

if Label == 0:
    Pos += 1
else:
    Neg += 1

```

```

TweetList = sorted(TweetList, key=lambda k: k['time'])          # Sort by time
TotalTimeLine = TweetList[-1]['time'] - TweetList[0]['time']
IntervalTime = TotalTimeLine / TIME_STEPS
k = 0
PreConInt = []

while True:
    k += 1
    tweetIndex = 0
    output = []
    if TotalTimeLine == 0:
        output.append(''.join(tweet["text"] for tweet in TweetList))
        break
    Start = TweetList[0]['time']
    Interval = int(TotalTimeLine / IntervalTime)

    Intset = []
    for inter in range(0, Interval):
        empty = 0
        interval = []
        for q in range(tweetIndex, len(TweetList)): # to store text in that interval
            if TweetList[q]['time'] >= Start and TweetList[q]['time'] < Start +
IntervalTime:
                empty += 1
                interval.append(TweetList[q]["text"])

            elif TweetList[q]['time'] >= Start + IntervalTime:
                tweetIndex = q - 1
                break

        if empty == 0:
            # empty interval

```

```

        output.append([])
    else:        # add the last tweet
        if TweetList[-1]['time'] == Start + IntervalTime:
            interval.append(TweetList[-1]["text"])
            Intset.append(inter)
            output.append(interval)

        Start = Start + IntervalTime # update start
        ConInt = ContinuousInterval(Intset)        # find max continuous interval
        if len(ConInt) < TIME_STEPS and len(ConInt) > len(PreConInt):
            IntervalTime = int(IntervalTime * 0.5)
            PreConInt = ConInt
            if IntervalTime == 0:
                output = output[ConInt[0]:ConInt[-1] + 1]
                break
        else:
            output = output[ConInt[0]:ConInt[-1] + 1]
            break
    counter += 1
    fe.close()

    for q in range(0, len(output)):
        output[q] = ".join(s for s in output[q])

    vectorizer = CountVectorizer(output,
        stop_words=["all", "in", "this", "and", "is", "as", "it", "so", "the", "we", "are",
        "via", "you", "your"])

    tf = vectorizer.fit_transform(output)
    transformer = TfidfTransformer()
    tfidf = transformer.fit_transform(tf)

    Allvocabulary = vectorizer.get_feature_names()

```

```

first_doc_vector = tfidf[0]                # get tfidf vec for first doc
df = pd.DataFrame(first_doc_vector.T.todense(), index=Allvocabulary,
columns=["tfidf"])
Input = []

for interval in tfidf.toarray():
    Wordvector = []
    WordvectorMatrix = []
    NonZeroCount = 0
    interval, Allvocabulary = zip(*sorted(zip(interval, Allvocabulary),
reverse=True))

    for word, value in zip(Allvocabulary, interval):
        if value != 0.0 and NonZeroCount < InputWord: # IMPUT_SIZE
            try:
                Wordvector = np.append(Wordvector, model[word])
                NonZeroCount += 1
            except:
                print("Word is not present")

    while NonZeroCount < InputWord:
        Wordvector = np.append(Wordvector, ([0.0] * 300))
        NonZeroCount += 1
    Input.append(Wordvector)

if len(Input) < TIME_STEPS:
    for q in range(0, TIME_STEPS - len(Input)):
        Input.insert(0, [0.0] * 300 * InputWord)
totalData.append(Input[:TIME_STEPS])
totalDataLabel.append(Label)

```



```

print(totalDataLabel)
print("totalDoc : " + str(totalDoc))
print("tdlist1 : " + str(tdlist1))
print("Pos : " + str(Pos))
print("Neg : " + str(Neg))
print("totalpost : " + str(totalpost))
print("maxpost : " + str(maxpost))
print("minpost : " + str(minpost))
print(len(totalData))
print(counter)

X_train = np.array(totalData[:int(counter/5*4)])
y_train = np.array(totalDataLabel[:int(counter/5*4)])
X_test = np.array(totalData[int(counter/5*4):])
y_test = np.array(totalDataLabel[int(counter/5*4):])

# LSTM
y_train = np_utils.to_categorical(y_train, num_classes=2)
y_test = np_utils.to_categorical(y_test, num_classes=2)

model=pickle.load(open('model.sav','rb'))

print("\nTesting-----")
cost, accuracy = model.evaluate(X_test, y_test, batch_size=y_test.shape[0],
verbose=False)
#print('test cost: ', cost)
print('test accuracy: ', accuracy)

y_pred1=model.predict_classes(X_test)
print("label:")
print(y_pred1)

```

```
if y_pred1==1:  
    result="Given Event is a Rumor..!! "  
else:  
    result = "Given Event is not a Rumor..!! "  
  
print(result)
```

6. TESTING

Software testing is one of the critical phase in the software development life cycle. It is used to check whether all the functionalities of the software are working as expected or not. It checks the quality of software and makes sure that it is defect free by identifying errors or gaps. Software testing is usually done by the tester or developer. Testing can be carried out using automated tools or manually. It is important because errors, bugs and gaps can be dangerous and expensive.

Usually, testing can be categorized in three types:

- Functional Testing
- Non-Functional Testing
- Maintenance

Unit Testing

Unit testing is a level of testing where individual components or units of the software are tested. The smallest testable portion of the software is called unit. The aim of unit testing is to validate that every individual unit are performing their task as anticipated. Unit testing is done by the developer in the development phase of the application. Some of the important reasons for performing unit testing are as follows:

- Identifies errors and bugs at an early stage of software development and saves costs.
- It helps to simplify integration.
- It provides project documentation.
- It helps use to re-use code in different application.
- It helps the developer to comprehend code more clearly.

Test no.	Test	Input	Expected Output	Actual Output
1	User Login Test	User Id and password	Authentic user should be loggedin.	Login Successful
2	Remove Links from tweets	Tweet	Tweets without links	Links removed
3	Remove emojis from tweets	Tweet	Tweets without emojis	Emojis removed
4	Remove tweet handle from tweets	Tweet	Tweets without tweet handle	Tweet handle removed
5	Remove special characters from tweets	Tweet	Tweets without special characters	Special characters removed
6	Display graph to check spread of rumour per day.	Time of tweets related to event.	Graph of number of tweets posted on that day is displayed.	Graph of number of tweets posted on that day is displayed
7	Check for Invalid Input	Invalid Input	User should enter valid input	Enter valid input
8	Keyword requirement restriction	No keyword	User should enter keyword before proceeding	Please enter keyword
9	Keyword requirement restriction to search tweets related to event.	Keyword	If keyword is entered, related tweets should be displayed.	Related tweets displayed
10	Number of tweets to be displayed should be entered.	Number of Tweets	System proceeds if user enters valid input.	System proceeds if user enters valid input.

Table 4: Unit Test Cases

Integration Testing -

Integration testing is a level of testing in which the individual modules or units are integrated and tested as a group. The software project includes many units and modules coded by various developers working in the project which on integrating may produce defects. Integration testing helps to identify various defects in the interaction of the units when they are integrated. Defects may occur due to different reasons like a developer has coded using a different programming logic than others, insufficient exception handling, problem in interfacing the database and modules, etc. Integration testing makes use of test stubs and drivers to carry out testing even when some unit is missing. Some of the important reasons for performing integration testing are as follows:

- Top-down and bottom-up approach for software testing leads to more code length coverage.
- It can be utilized in the early as well as later phases of testing.

Test no.	Test	Input	Expected Output	Actual Output
1	Tweets related to event are displayed when keyword is entered.	Keyword	Tweets are displayed	Tweets are displayed
2	When keyword is entered then graph to check spread of rumour per day is displayed.	Keyword related to event.	Graph of number of tweets posted on that day is displayed	Graph of number of tweets posted on that day is displayed
3	Retrieve tweets and then process for detection.	Keyword and number of tweets	Input query processed and check for rumour or not	Event rumour or non-rumour

Table 5: Integration test Cases

System Testing –

System testing is a level of testing which helps in validating the completely integrated system. It is carried out after the integration testing. System testing is a black box testing as the testing is carried out without the knowledge of code or design of software. It consists of both, functional and non-functional type of testing. Some of the important reasons for performing system testing are as follows:

- Helps check the system against the functional, non-functional and business requirements.
- Detects maximum bugs before acceptance testing.
- Tester needs no internal knowledge of the software.

Test no.	Test	Input	Expected Output	Actual Output
1	Complete System Test	Authorized user's credentials, input keyword and number of tweets	Utilization of the system should be successful	System works successful and gives correct output

Table 6: System Test Cases

Acceptance Testing –

Acceptance testing is a level of testing used to check if the software fulfills all the requirements of users and can be accepted for business use. It is carried out by the user to check if the software is easy to use and all requirements are taken care of. The feedback provided by the user helps to improve the software. Some of the important reasons for performing acceptance testing are as follows:

- Expose all the defects before delivery of software.
- Provide useful feedback from user.
- Increases the trust of the user.

Test no.	Test	Input
1	User Acceptance Test	All functional requirements are included in the system
2	Availability	System is easily accessible
3	Usability	Easy to use
4	Operational Acceptance Test	System performs efficiently and produces correct results
5	Confidentiality	Sensitive data is secured and only authorized user can login
6	Documentation	Proper documentation of system provided

Table 7: Acceptance Test Cases

7. RESULTS AND EVALUATION

Proposed model is implemented using python on Anaconda platform. TensorFlow and Keras are math libraries used for numeric calculation. Google Word2Vec is a tool is used for word embedding. Tweepy library is used to access Twitter API. Using the tweet text and the time of post of tweet, results are obtained.

Table 8 shows the summary of input provided by the model which shows total number of rumour and non-rumour events taken into consideration, it also shows total number of tweets posted and maximum and minimum number of tweets posted for an event.

Rumour	484
Non – Rumour	193
Total Post	349897
Maximum Post	15196
Minimum Post	6

Table 8: Summary of input

Table 9 shows the test accuracy for 677 events with 484 rumour events 193 non-rumour events. Along with accuracy; precision, recall and F1 score are some of the important model metrics used to calculate the goodness of the model. These metrics are calculated using True Positive (TP), False Positive (FP), True Negative (TN), False Negative (FN).

Formulae:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1\ Score = 2 \times \frac{Precision * Recall}{Precision + Recall}$$

Results

Accuracy	0.85
Precision	0.88
Recall	0.93
F1 Score	0.91

Table 9: Results of Model

Figure 16 shows plot of confusion matrix Where the true positive (TP), true negative (TN), false positive (FP) and false negative (FN) are displayed using different colours. These colours specify a range of values through which one can get an estimate of TP, TN, FP and FN values.

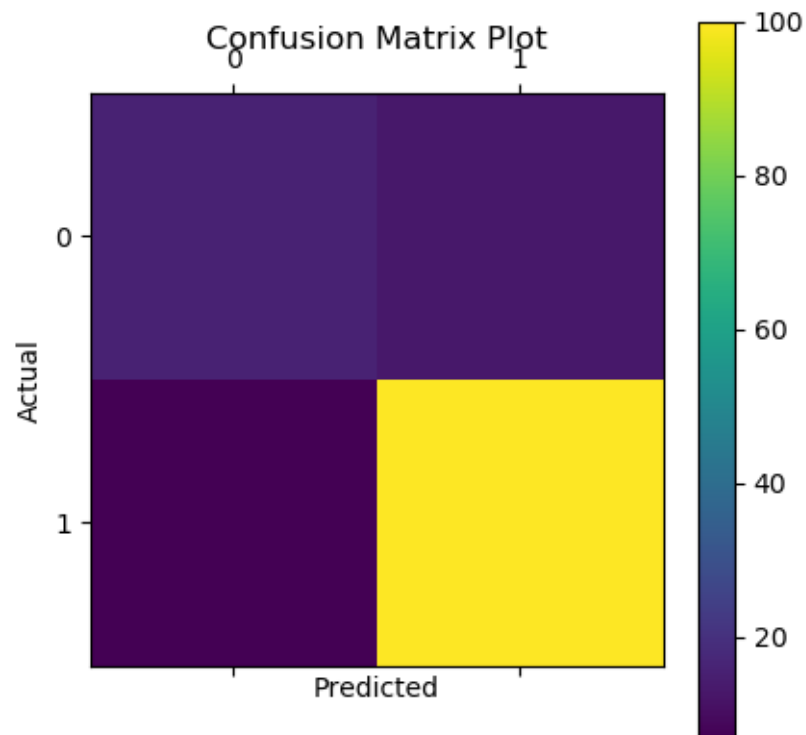


Figure 16: Confusion matrix plot

Figure 17 shows the graph of training accuracy versus epochs. This can be used to visualize the increase in accuracy of model during training in proportion to increase in epoch.

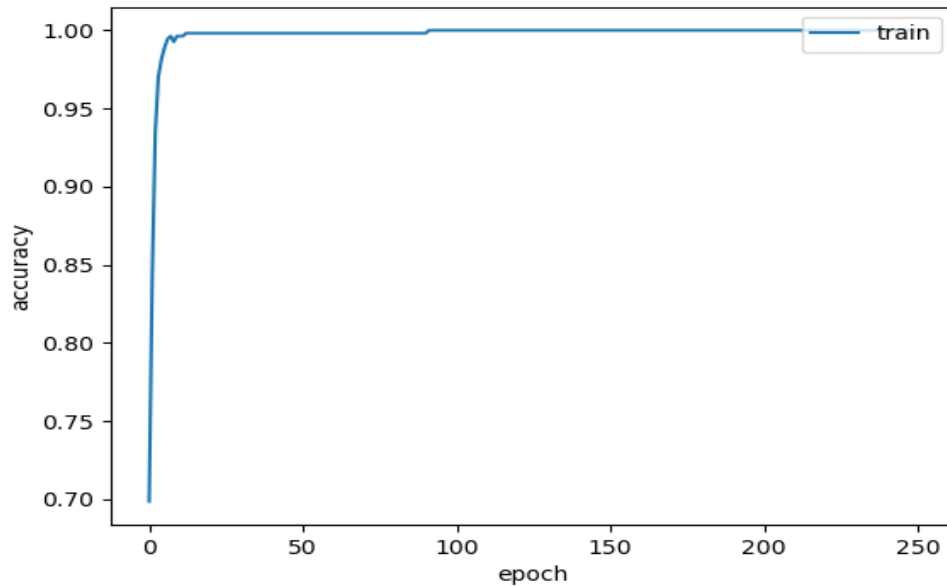


Figure 17: Graph of Train Accuracy vs. Epoch

Figure 18 shows graph of training loss versus epochs. This can be used to visualize the training loss of model gradually decreases as number of epochs are increased.

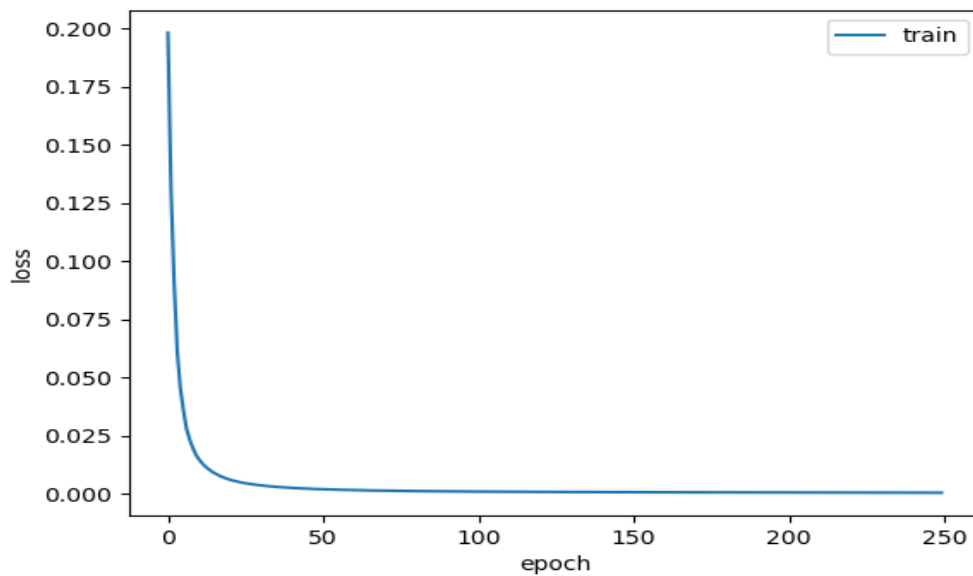


Figure 18: Graph of Train Loss vs. Epoch

Figure 19 gives comparison of different methods used for rumour detection on twitter like DT-Rank, attRNN, DTC, RFC, CallAtRumor, LSTM and HAS-BLSTM.

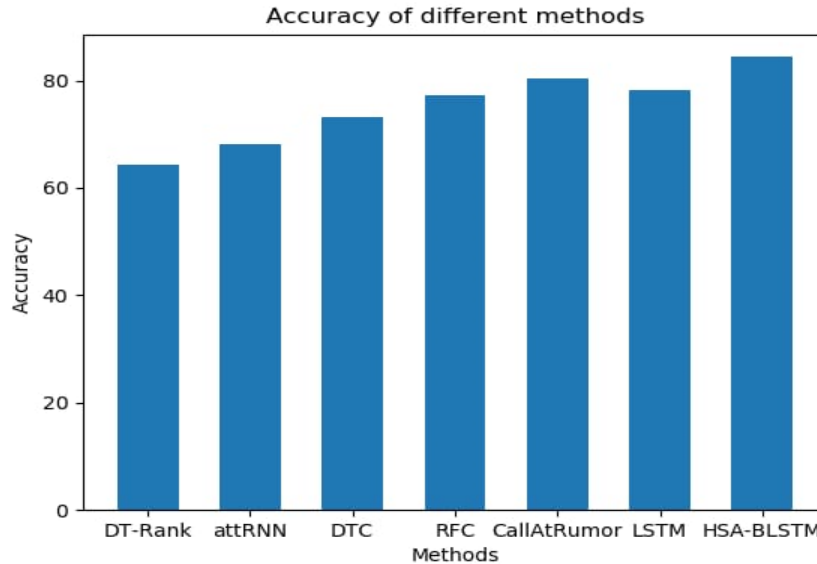


Figure 19: Comparison of Accuracy of Different Methods

DT-Rank: Zhao, Resnick, and Mei [14] proposed a model to identify trending rumors using decision tree-based ranking. They search for cluster disputed factual claims and enquiry phrases, and rank the clustered results based on statistical features.

attRNN: Zhiwei Jin, Yongdong Zhang, Juan Cao, Han Guo and Jiebo Luo [15] proposed a novel Recurrent Neural Network with an attention mechanism (att-RNN) to fuse multimodal features for effective rumor detection. In this end-to-end network, image features are incorporated into the joint features of text and social context, which are obtained with an LSTM network, to produce a reliable fused classification.

DTC: Carlos Castillo, Marcelo Meddoza and Barbara Poblete[16] proposed a Twitter information credibility model using Decision Tree Classifier.

RFC: Sejeong Kwon, Meeyoung Cha, Kyomin Jung, Wei Chen and Yajun Wang[17] proposed a Random Forest Classifier using three parameters to fit the temporal tweets volume curve.

CallAtRumor: Chen et al. [5] proposed a deep-attention model for early detection of rumours which is based on RNN. CallAtRumours(Call Attention to Rumours) is deep attention-based RNN for early identification of rumours.

LSTM : Proposed model in this project work which uses LSTM and pooling layer of CNN to detect rumours accurately. The model can easily learn the hidden features and local information very well.

HSA-BLSTM: Han Guo, Juan Cao, Yazhi Zhang, Junbo Guo and Jintao Li[18] proposed a hierarchical bidirectional long short memory model for representation learning and then, the social texts are incorporated into the network via attention mechanism, such that important semantic information is introduced to the framework for more robust detection.

8. GRAPHICS USER INTERFACE OF THE SYSTEM

Rumoured tweet – Spraying or introducing bleach or another disinfectant into your body will protect you against #corona

1. Login Screen -

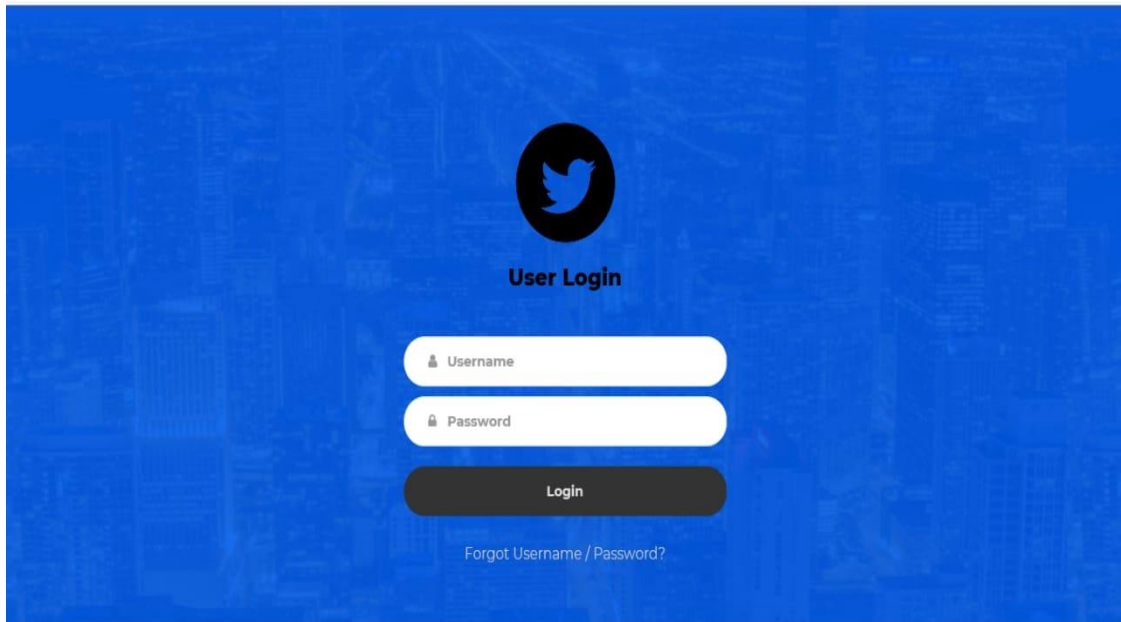


Figure 20: Login Interface of Rumour Detection Model

2. Password Verification –

The user has 3 attempts to enter credentials and get logged in.

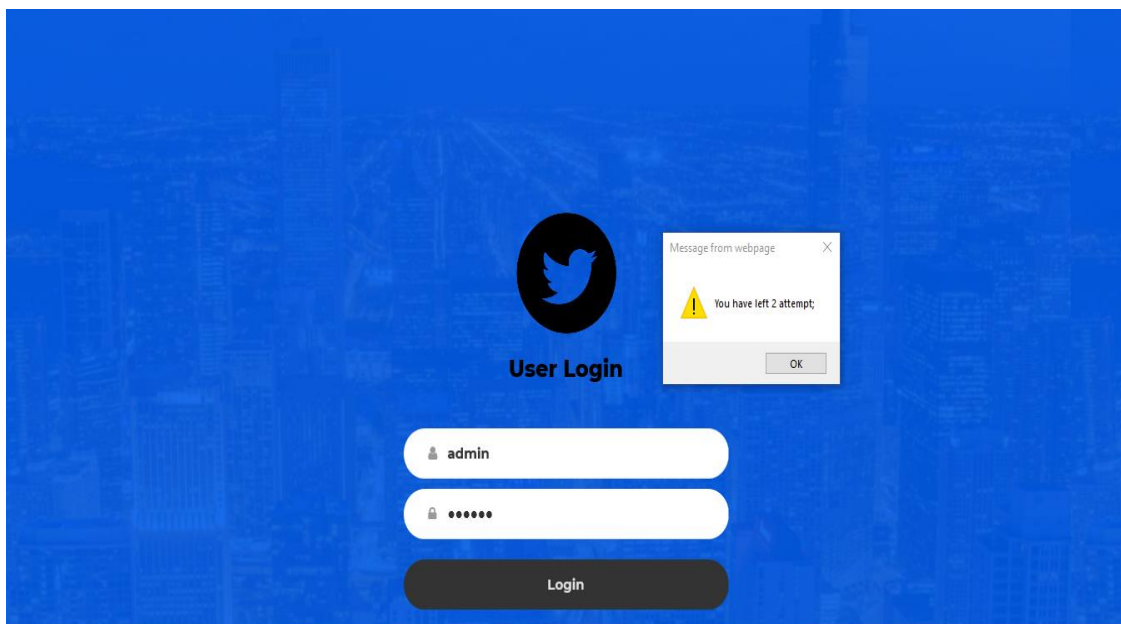


Figure 21: Alter user for number of attempts left to login successfully

3. Enter keyword –

The administrator enters the keyword related to event and number of tweets to be displayed.

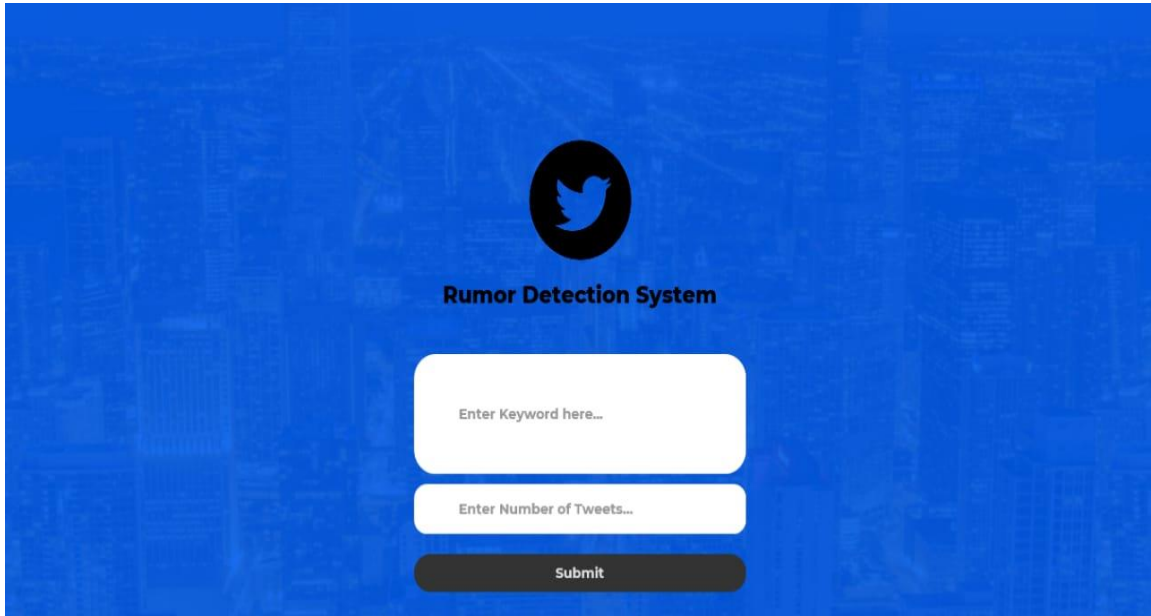


Figure 22: Interface to check rumour

4. Displaying tweets –

After keyword is entered by user the corresponding tweets are retrieved and displayed on screen.

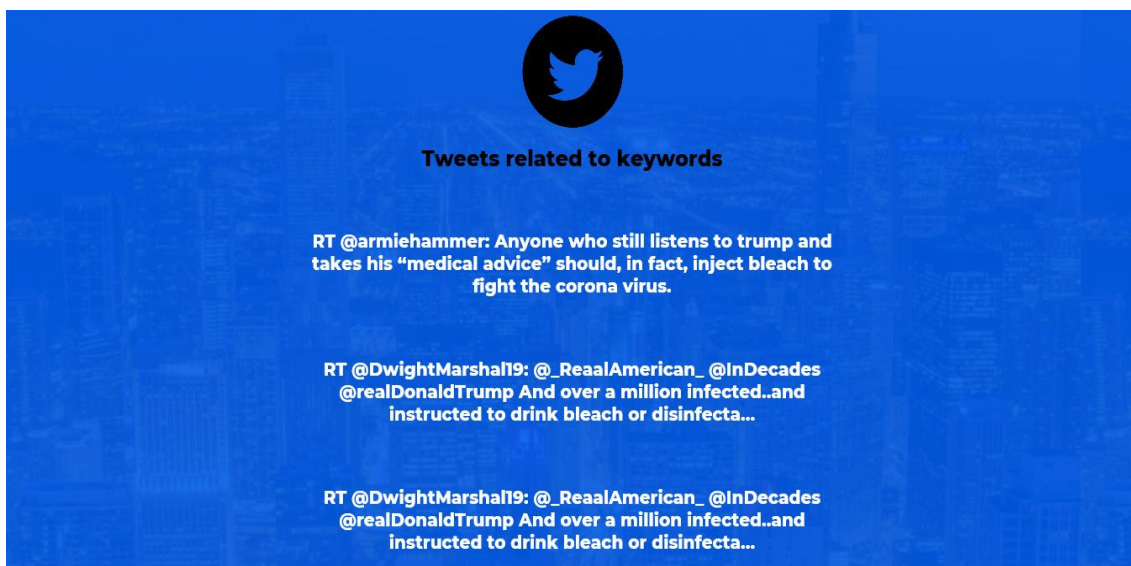


Figure 23: Tweets retrieved related to keywords

5. Analyze graph –

A graph of number of tweets posted per day is displayed to get an idea of propagation of rumour.

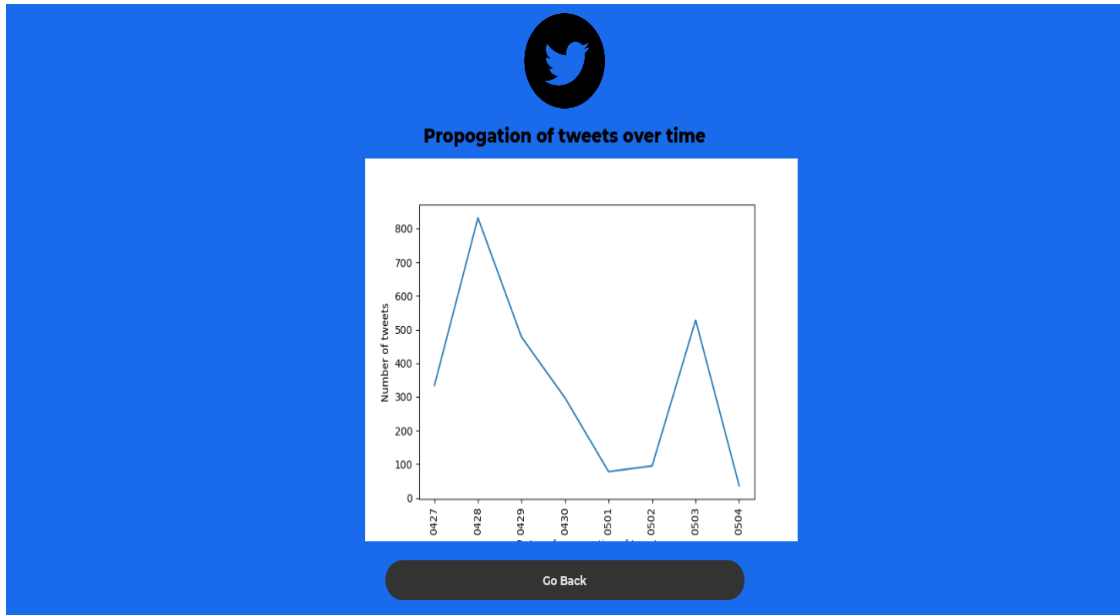


Figure 24: Interface to view graph

6. Rumour detection –

The model alerts the administrator whether the given event is rumour or not so that the administrator can take action when event is rumour.

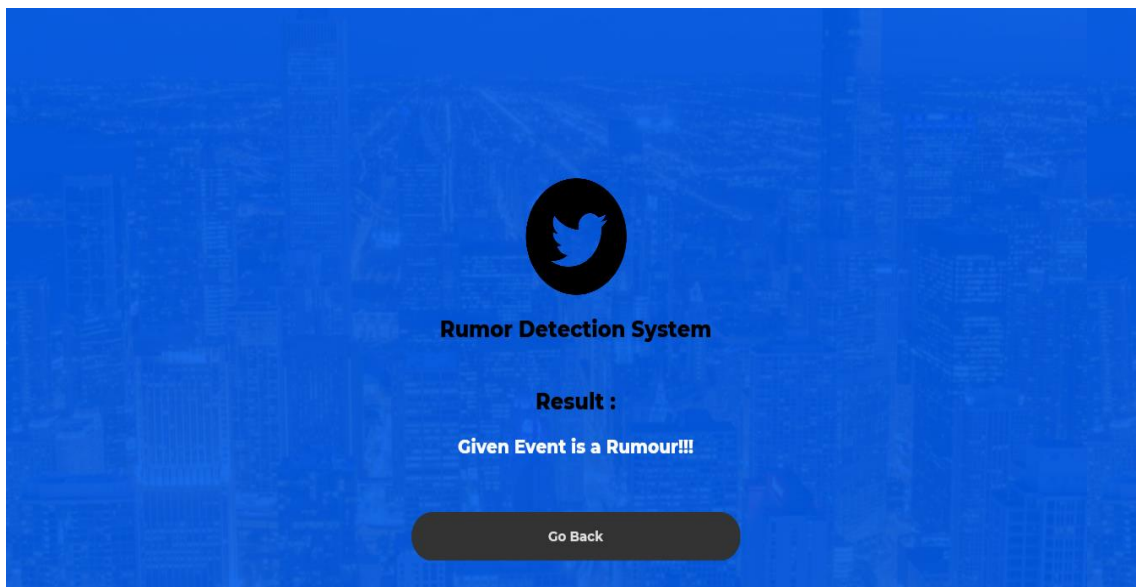


Figure 25: Result obtained for the tweet

9. FUTURE SCOPE AND CONCLUSION

9.1 FUTURE SCOPE

- The implementation can be further worked upon improving the accuracy by adding more influence and complex features for early rumor identification.
- The model can be worked upon increasing accuracy by detecting the spread of rumours through memes and edited images by considering the correlation between the pictures and the tweet.

9.2 CONCLUSION

Currently, some models use ML based techniques to detect rumours on social media sites. These models employ feature extraction which is time-consuming and biased. This system assists user to successfully detect rumoured tweets using deep learning based method. LSTM networks and pooling function of CNN are used in this model to accurately debunk the suspicious tweet as rumour or non-rumour. The system considers the forwarding contents of the user for detection purpose. Using the max pooling function, cost and dimensionality of the model is reduced and performance is improved. The accuracy obtained using this model is 85.29% for total of 677 events. The efficiency of the model can be further improved by adding numerous hidden layers to the LSTM.

10. PAPER PUBLICATION AND SPONSORSHIP DETAILS

Aparna Bindage, S. M. Jaybhaye, Sai Pande and Datta Dhebe, "Rumour Detection on Twitter Using Long Short-Term Memory", 2020, 2131 – 2138p.





Sponsorship Details –



GigaRaise Insights Research and Co.

F - 104, 'A' Wing, Sai Atharva Apartment,
Near Swaraj Garden, Pimple Saudagar, Pune.

To Whom It May Concern:

We looked over your application for project sponsorship from us. We would be moving forward with providing you the sponsorship for the project titled "Rumour Detection on Microblogging Sites Using LSTM" under the guidance of Prof. S. M. Jaybhaye, Dept. of IT, SCOE.

The team members include:

1. Aparna Bindage
2. Apeksha
3. Sai Pande
4. Datta Dhebe

Regards,
Dipali Kamble
Digitally signed
by Dipali Kamble
Date: 2019.11.27
08:51:08 +05'30'
Dipali Kamble

11. REFERENCES

- [1] G. Allport and L. Postman, “The psychology of rumour”, Rinehart & Winston, 1947.
- [2] N. DiFonzo, R. Rosnow and P. Bordia, “Reining in rumours”, *Organizational Dynamics*, 1994, 23(1):47–62p.
- [3] N. DiFonzo and P. Bordia, “Rumour psychology: Social and organizational approaches”, American Psychological Association, 2007.
- [4] J. Ma, B. J. Jansen, W. Gao, M. Cha, P. Mitra, K. Wong and S. Kwon, “Detecting rumors from microblogs with recurrent neural networks”, in *Proceedings of IJCAI*, 2016.
- [5] T. Chen, L. Wu, X. Li, Jun Zhang, Hongzhi Yin and Yang Wang, “Call attention to rumours: Deep attention based recurrent neural networks for early rumour detection”, 2018, 40-52p.
- [6] W. Chen, Y. Zhang, C. Yeo, C. Lau, B. Sung Lee. “Unsupervised rumour detection based on user’s behaviours using neural networks”, *Journal paper vol.105*, April 2018, 226-233p.
- [7] J. Ma, K. Wong and W. Gao “Rumor Detection on Twitter with Tree-structured Recursive Neural Network” in *Proc. 56th Annu. Meeting Assoc. Comput. Linguistics*, vol. 1, 2018.
- [8] O. Ajao, D. Bhowmik, and S. Zargari, “Fake news identification on Twitter with hybrid CNN and RNN models”, in *Proc. 9th Int. Conf. Social Media Soc.*, 2018, 226-230p.
- [9] N. Ruchansky, S. Seo, and Y. Liu, “CSI: A hybrid deep model for fake news detection” in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*.
- [10] T. Lan, C. Li, J. Li “Mining Semantic Variation in Time Series for Rumor Detection via Recurrent Neural Networks”, International Conference.
- [11] F. Yu, Q. Liu, S. Wu, L. Wang, T. Tan, “A Convolutional Approach for Misinformation Identification”, in *Proceedings of the 26th International Joint Conference on Artificial Intelligence*.
- [12] W. Y. Wang, “Liar, liar pants on fire : A new benchmark dataset for fake news detection”, 2017, arXiv: 1705.00648.

- [13] M. Al-Sarem, W. Bouilia, M. Al-Harbi, J. Qadir, A. Alsaeedi, “Deep learning – based rumor detection on microblogging platforms : A systematic review”.
- [14] Zhe Zhao, Paul Resnick, and Qiaozhu Mei. Enquiring minds: Early detection of rumors in social media from enquiry posts. In Proceedings of WWW, 2015.
- [15] Zhiwei Jin, Juan Cao, Han Guo, Yongdong Zhang and Jiebo Luo, “Multimodal Fusion with Recurrent Neural Networks for Rumour Detection on Microblogs”,2017.
- [16] C. Castillo, M. Meddoza and B. Poblete, “Information credibility on twitter”, *in preceedings of www*, 2011.
- [17] S. Kwon, M. Cha, K. Jung, W. Chen and Y. Wang, “Prominent features of rumor propagation in online social media”, *in Proceedings of ICDM*, 2013.
- [18] H. Guo, J. Cao, Y. Zhang, J. Guo and J. Li, “Rumour Detection with Hierarchical Social Attention Network”,2018.