Dissertation on

"RUMOUR DETECTION AND VERACITY VERIFICATION USING SOCIO-LINGUISTIC DATA AND SOCIAL GRAPHS"

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

Bachelor of Technology
in
Computer Science & Engineering

UE17CS490A – Capstone Project Phase - 2
Submitted by:

Sukanya Harshvardhan	**PES1201700214**
Sirisha Lanka	**PES1201700294**
Prajna Girish	**PES1201701261**

Under the guidance of

Prof. Bhaskarjyoti Das
Visiting Professor
PES University

January - May 2021

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

'Rumour Detection and Veracity Verification using socio-linguistic data and social graphs'

is a bonafide work carried out by

Sukanya Harshvardhan	**PES1201700214**
Sirisha Lanka	**PES1201700294**
Prajna Girish	**PES1201701261**

in partial fulfilment for the completion of the eighth semester Capstone Project Phase - 2 (UE17CS490A) in the Program of Study - Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period January 2021 - May 2021. 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. Bhaskarjyoti Das	**Dr. Shylaja S S**	**Dr. B K Keshavan**
Visiting Professor	Chairperson	Dean of Faculty

External Viva

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

DECLARATION

We hereby declare that the Capstone Project Phase - 2 entitled **“Rumour Detection and Veracity Verification using socio-linguistic data and social graphs''** has been carried out by us under the guidance of Prof.Bhaskarjyothi Das, Visiting Professor, and submitted in partial fulfilment of the course requirements for the award of degree of **Bachelor of Technology** in **Computer Science and Engineering** of **PES University, Bengaluru** during the academic semester January - May 2021. The matter embodied in this report has not been submitted to any other university or institution for the award of any degree.

PES1201700214	**Sukanya Harshvardhan**	[illegible]
PES1201700294	**Sirisha Lanka**	[illegible]
PES1201701261	**Prajna Girish**	Prajna

ACKNOWLEDGEMENT

We would like to express our gratitude to Prof. Bhaskarjyoti Das, Department of Computer Science and Engineering, PES University, for his continuous guidance, assistance, and encouragement throughout the development of this UE17CS490A - Capstone Project Phase – 2.

We are grateful to the project coordinators, Prof. Silviya Nancy J, for organizing, managing, and helping with the entire process.

We 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. Suryaprasad J, Vice-Chancellor, PES University for providing to us 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 family and friends.

ABSTRACT

Sharing content on social media platforms such as Twitter is one of the most effective ways to target a large audience and spread awareness. However, this privilege can be misused when the shared content includes unverified and mistrusted information in the form of rumours.

Hence, early detection of these rumours and assessing their veracity is a key problem to be addressed on such platforms. In this project, we have proposed a learning mechanism which incorporates analysis of socio-linguistic data and social graph perspectives for rumour detection and veracity verification.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER- 1
INTRODUCTION

Society has moved online. Although this has been majorly a boon owing to its accessibility and infinite resource power, there is a major downside which needs to be addressed. People nowadays acquire all their information from social media platforms, rather than conventional news channels, and usually believe most of what they see. With all social media becoming increasingly popular, it is possible that information posted by any user on the network, grasps the attention of almost a billion users, within a few seconds.

Since there are no official restrictions on the content of what is being posted on these social media platforms, there tends to be a lot of unverified information online, also known as rumours. The low cost of information exchange on such social media sites help data and information to spread extremely rapidly, and the more a rumour spreads the more people tend to believe it. Hence, their early detection is of utmost importance.

Our focus for this research problem mainly is on Twitter data, considering that it is the primary social media platform used to post breaking news and other current affairs.

Further, usage of a social graph would be perfect to understand the data flow on twitter considering that there are fixed data propagation algorithms - retweets, shares, replies etc. The user's behaviour and credibility can be judged based on his/her connections in the social graph, which would further help us determine if the user's post contains unverified information or not.

Finally, the language spoken on social media platforms such as Twitter does not abide to the norms of the conventional English language- there is a limit on the number of characters, people use abbreviations and slang, and there might also be spelling errors. This type of language comes under the umbrella of socio-linguistic data, which is what we would be using in this project. Social information propagation and socialization behaviours are not only related to the content of the post and user interests but also to social relationships of the user. Hence, there is a need to integrate all these factors to construct a better learning model for rumours. The problem at hand can be split into various auxiliary tasks. First, is the rumour detection aspect, where we determine if a tweet's content contains to be verified rather than it just containing an expression. Second, is the stance taken by a user and to determine his/her feelings towards this particular tweet. Finally, is the veracity verification where we determine if the rumour at hand is true, false or unverified.

CHAPTER – 2
PROBLEM STATEMENT

Detecting rumours on online platforms such as Twitter and determining their veracity is a crucial task in preventing the unverified information from propagating across the network.

Over the past few years, various methodologies have been implemented to solve this problem.
Multiple ML and Deep Learning approaches have been researched, which analyse the context of a post to classify it as a rumour or non-rumour have been the most common approaches.

Through our research, we propose to build a joint learning model that integrates the socio linguistic data features present in the post along with a social graph perspective. This joint learning model will help include various features such as the content of the post, the propagation pattern as well as features extracted from the social network analysis of the network the post is a present in.

A joint learning model with the inclusion of many more features as compared to a single model will help us detect rumours and determine their veracity with more accurate results.

CHAPTER- 3

LITERATURE SURVEY

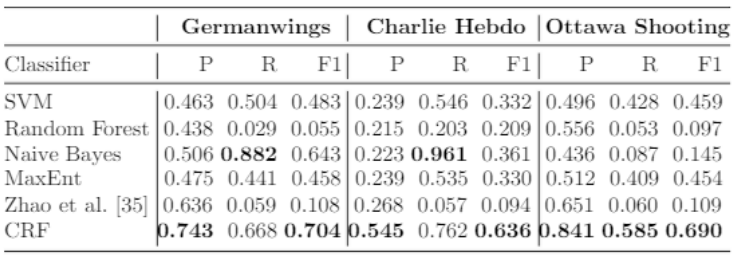
3.1 Content Based Models

This section consists of all the papers that contain content-based methodologies for rumour detection and veracity verification.

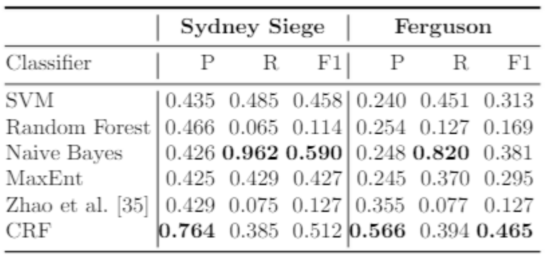
3.1.1 Exploiting Context for Rumour Detection in Social Media (2017)

The paper aims to leverage the tweet thread a current tweet is present in, by aggregating and classifying the tweets that precede a post as rumour and non-rumour thereby improving the performance of the rumour detection system.

A linear chain Conditional Random Fields (CRF) model is used to learn from the sequential dynamics of the social media posts.



	Germanwings			Charlie Hebdo			Ottawa Shooting		
Classifier	P	R	F1	P	R	F1	P	R	F1
SVM	0.463	0.504	0.483	0.239	0.546	0.332	0.496	0.428	0.459
Random Forest	0.438	0.029	0.055	0.215	0.203	0.209	0.556	0.053	0.097
Naive Bayes	0.506	**0.882**	0.643	0.223	**0.961**	0.361	0.436	0.087	0.145
MaxEnt	0.475	0.441	0.458	0.239	0.535	0.330	0.512	0.409	0.454
Zhao et al. [35]	0.636	0.059	0.108	0.268	0.057	0.094	0.651	0.060	0.109
CRF	**0.743**	0.668	**0.704**	**0.545**	0.762	**0.636**	**0.841**	**0.585**	**0.690**



	Sydney Siege			Ferguson		
Classifier	P	R	F1	P	R	F1
SVM	0.435	0.485	0.458	0.240	0.451	0.313
Random Forest	0.466	0.065	0.114	0.254	0.127	0.169
Naive Bayes	0.426	**0.962**	**0.590**	0.248	**0.820**	0.381
MaxEnt	0.425	0.429	0.427	0.245	0.370	0.295
Zhao et al. [35]	0.429	0.075	0.127	0.355	0.077	0.127
CRF	**0.764**	0.385	0.512	**0.566**	0.394	**0.465**

Table 1: Classification Results for various classifiers event wise

3.1.2 Detecting Rumours from Microblogs with Recurrent Neural Networks (2016)

Textual content of a post has been one of the primary attributes utilized for rumour detection. It includes the content in the source post and all user replies following it. In this paper, the model proposed was an recurrent neural networks (RNN) architecture. It aimed to learn the hidden representations that capture the variation of contextual information of relevant posts over time."

Dataset Used: 1. Twitter data collected using Twitter APIs

2. Weibo dataset

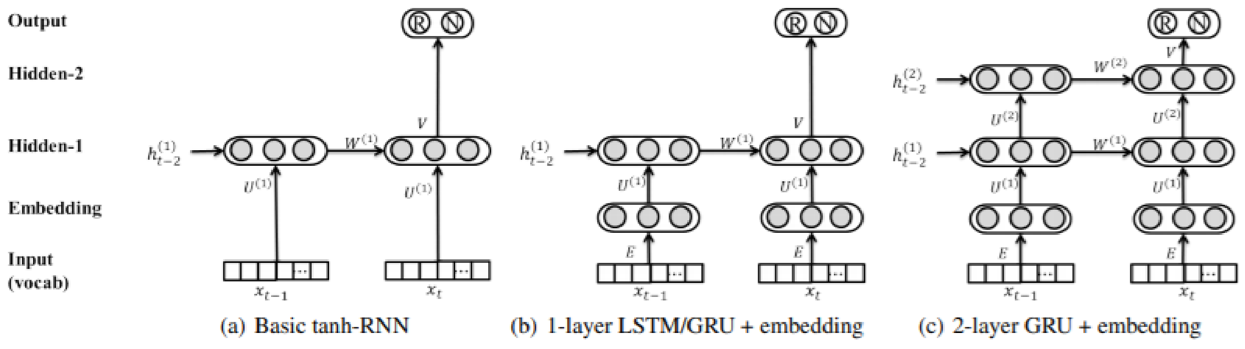
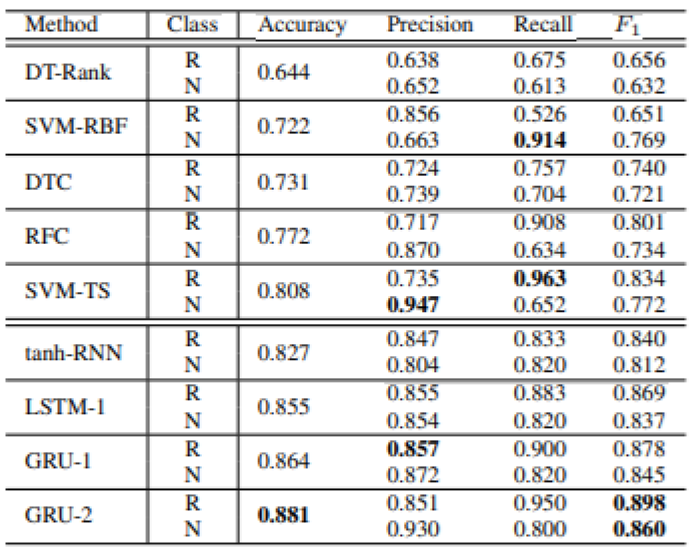


Figure 1. RNN based model used for rumour detection



Method	Class	Accuracy	Precision	Recall	F_1
DT-Rank	R	0.644	0.638	0.675	0.656
	N		0.652	0.613	0.632
SVM-RBF	R	0.722	0.856	0.526	0.651
	N		0.663	**0.914**	0.769
DTC	R	0.731	0.724	0.757	0.740
	N		0.739	0.704	0.721
RFC	R	0.772	0.717	0.908	0.801
	N		0.870	0.634	0.734
SVM-TS	R	0.808	0.735	**0.963**	0.834
	N		**0.947**	0.652	0.772
tanh-RNN	R	0.827	0.847	0.833	0.840
	N		0.804	0.820	0.812
LSTM-1	R	0.855	0.855	0.883	0.869
	N		0.854	0.820	0.837
GRU-1	R	0.864	**0.857**	0.900	0.878
	N		0.872	0.820	0.845
GRU-2	R	**0.881**	0.851	0.950	**0.898**
	N		0.930	0.800	**0.860**

Table 2. Results of RNN model on Twitter Dataset

3.1.3 Predicting Stances in Twitter Conversations for Detecting Veracity of Rumors: A Neural Approach (2018)

This research paper has considered features such as the content/text that is present in the tweet, the timestamp associated with it, and the sequence of the conversational structures leading up to the main tweet under consideration. Veracity verification has been carried out by detecting the stance of each individual tweet. The veracity of the original rumour is finally determined by the final prediction of the stances of all the tweets in the conversation tree.

The dataset referred to in this paper is the SemEval2017 rumor detection dataset.

3.1.4 Early Detection of Rumours on Twitter via Stance Transfer Learning (2020)

This paper leverages models with transfer learning of stance representation against its counterparts without transfer learning and popular models such as SVM and ML-TD.

These models perform supervised learning, breaking the task into two sub-problems: the first being, stance classification from comments and second, rumour detection from source tweet text.

This approach is implemented by building a CNN and fine-tuned BERT-based model which learns the stance representation using transfer learning along with a CNN-BiLSTM and BERT-based model to integrate the stance representation into tweet representation.

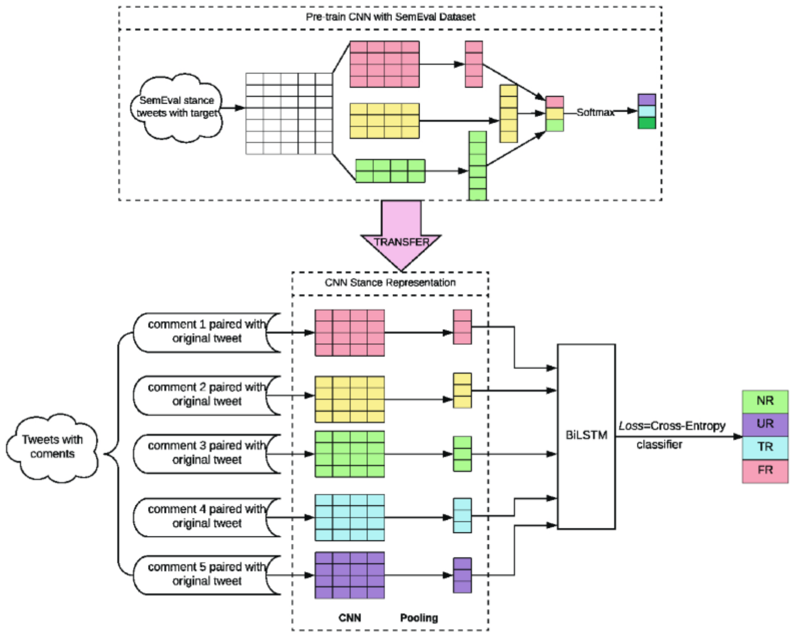

Pre-train CNN with SemEval Dataset
SemEval stance tweets with target
Softmax
TRANSFER
CNN Stance Representation
Tweets with coments
comment 1 paired with original tweet
comment 2 paired with original tweet
comment 3 paired with original tweet
comment 4 paired with original tweet
comment 5 paired with original tweet
CNN
Pooling
BiLSTM
Loss=Cross-Entropy classifier
NR
UR
TR
FR

Figure 2. Stance-CNN+BiLSTM model

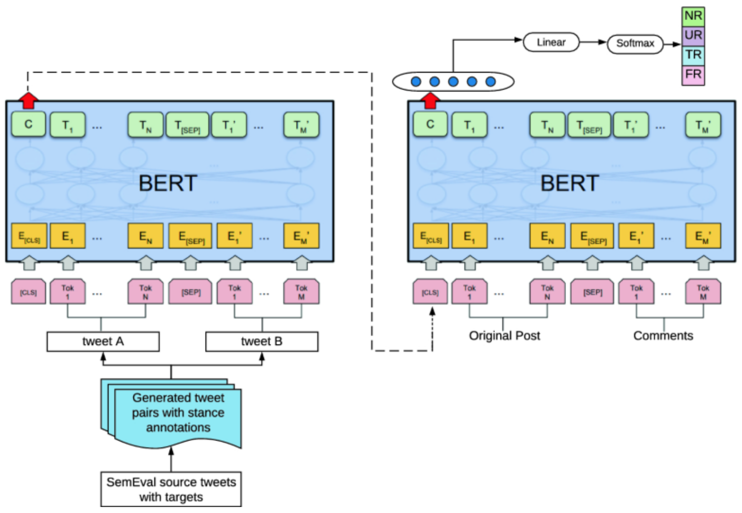

NR
UR
TR
FR
Linear
Softmax
BERT
BERT
tweet A
tweet B
Generated tweet pairs with stance annotations
SemEval source tweets with targets
Original Post
Comments

Figure 3. Stance-BERT model

3.1.5 Detecting Rumours on Social Media Based on a CNN Deep Learning Technique (2020)

This paper implements CNN deep learning technique to detect rumours spreading on Twitter. The author’s claim that it is extremely crucial to find the optimal hyperparameter settings to achieve the best performance result of the model.

The recall and precision balance is the best when a CNN model is implemented.

3.1.6 Attention Based LSTM for Stress Detection

The defining factor of a rumour is the fact that it creates a sense of anxiety and chaos among readers. This paper uses a long Short-Term Memory (LSTM) model fed with word embeddings for detecting stress.

The model makes use of an Attention mechanism which weighs the importance of every word and chooses what to retrieve.

It also learns which word combinations and sequences are expressed more often when someone is stressed.

3.2 Graph Based Models

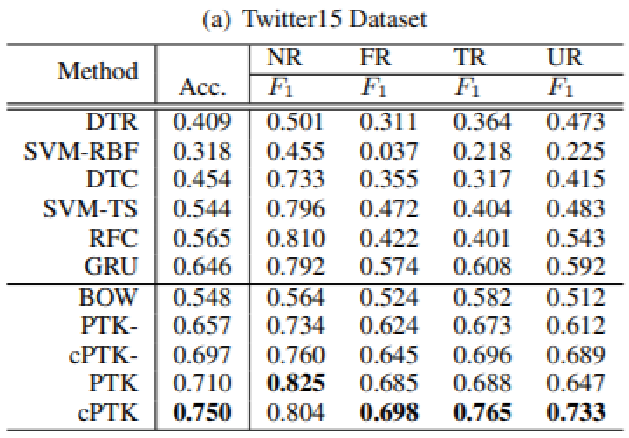
This section consists of all the papers that contain graph-based methodologies for rumour detection and veracity verification.

3.2.1 Detect rumours in microblogger's content with kernel learning (2017)

The propagation structures of a rumour post vary from that of a non-rumour post present in any network. This paper suggests that the analysis of these structures will help detect rumours on any social media site.

To carry out this implementation a Propagation structure via kernel learning is used.

This method captures high-order patterns that differentiate the different types of rumours by noting how similar the trees made from the propagation structures are.



(a) Twitter15 Dataset

Method		NR	FR	TR	UR
	Acc.	F_1	F_1	F_1	F_1
DTR	0.409	0.501	0.311	0.364	0.473
SVM-RBF	0.318	0.455	0.037	0.218	0.225
DTC	0.454	0.733	0.355	0.317	0.415
SVM-TS	0.544	0.796	0.472	0.404	0.483
RFC	0.565	0.810	0.422	0.401	0.543
GRU	0.646	0.792	0.574	0.608	0.592
BOW	0.548	0.564	0.524	0.582	0.512
PTK-	0.657	0.734	0.624	0.673	0.612
cPTK-	0.697	0.760	0.645	0.696	0.689
PTK	0.710	**0.825**	0.685	0.688	0.647
cPTK	**0.750**	0.804	**0.698**	**0.765**	**0.733**

Table 3. Rumour Detection Results using the Propagation Tree Kernel method

3.2.2 Rumour Detection in Twitter with Social Graph Structures (2019)

Social graphs can be constructed from existing social networks and information networks. The construction of these graphs is usually based on the user's likes, follows and follow backs.

The social graph constructed using the user's followers could indicate the user's intentions which could in turn indicate their credibility.

In this paper, features extracted from these constructed social graphs are used for learning by machine learning models to determine whether a post is a rumour or not.

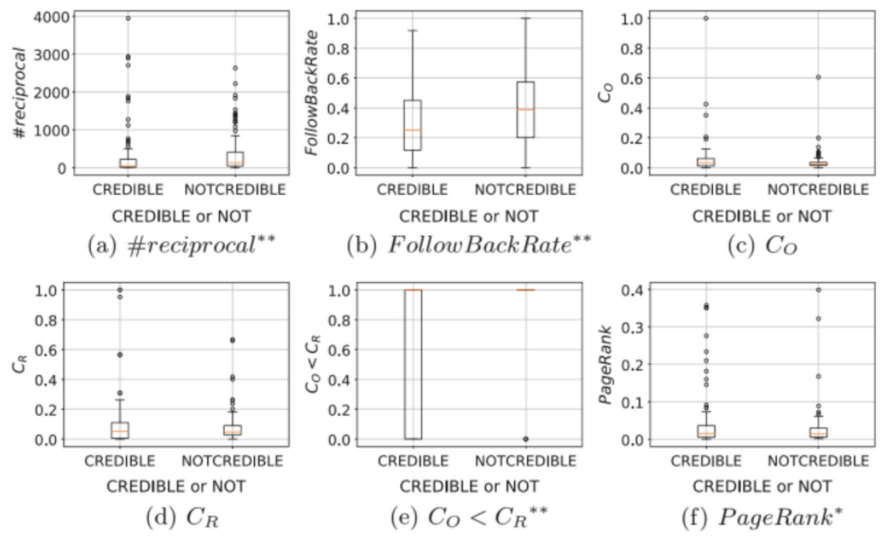


Figure 5. Feature Distribution

3.2.3 Detecting Rumour Patterns in Streaming Social Media (2015)

This paper aims to identify rumours through their structural and behavioural aspects of their patterns. The graph structure patterns of propagation are constructed to detect non-credible news. All the important cascades can be decomposed into a set of the most popular cascade patterns. Semantic analysis of these streaming posts such as user attitude (deny, support, question) and information propagated relationship (retweet, mention, reply) can help in the detection of rumours. Additional features such as relational-index, TF-IDF are also used.

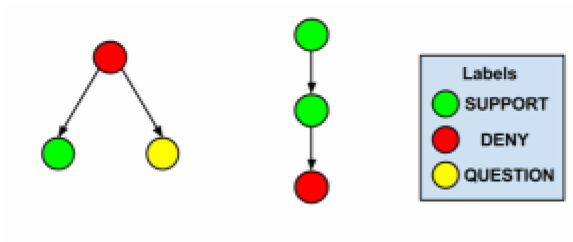


Figure 6. Rumour Pattern

3.2.4 Rumour Detection on Social Media with Bi-Directional GCN

In this paper, a Bi-GCN, also known as a Bi-Directional Graph Convolutional Network, runs different propagation strategies on the given tweet data. The structures of graphs built on rumours and non-rumours vary. This graph convolutional network learns the graph structures for the spread of rumours and understands the patterns of the rumour and how it spreads. Data embedded in the main tweet is used in every layer of the model, to have high emphasis on the more important tweet at hand, rather than the replies/retweets to it. This model shows better results than the current approaches.

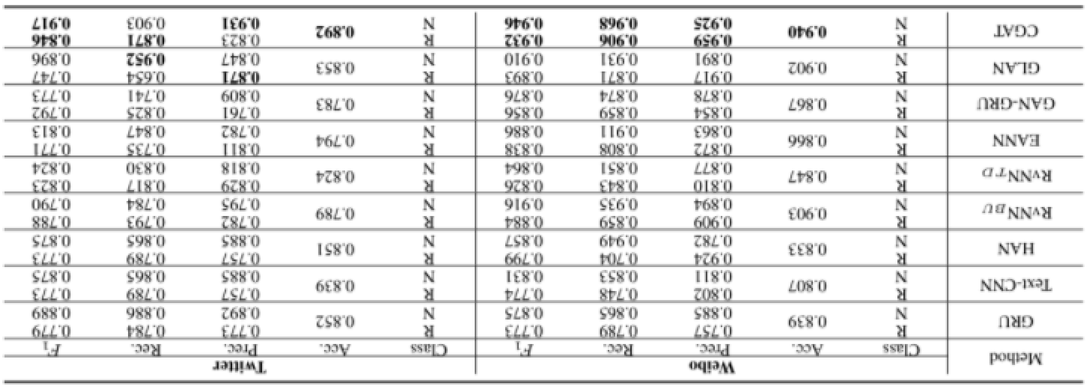
3.2.5 Rumour spreading in social networks

This paper studies the performance of the classic preferential model for rumour spreading in social networks. This model was built by analysing properties of preferential attachment graphs.
The randomized gossip protocol is studied as a part of this methodology in order to understand the mechanism of how information is disseminated in various networks.
This paper concludes that the PUSH—PULL strategy helps in disseminating information quickly across nodes of a PA graph and the PUSH, PULL strategies themselves obtain the same result but slowly.

3.2.6 Rumour Detection on Social Media with Graph Structured Adversarial Learning

The aim of this paper is to develop a graph based detector which identifies rumours in a network from an adversarial perspective.
A data network with information is built to extract essential features from the tweets, the source user and comments by other users for detection. A graph adversarial learning framework is then proposed where a malicious user tries to inject bugs/raise concerns in a dynamic behaviour on the structure of the graph to confuse the detector, during which the detector would learn more in depth and distinct features of the structures to stop or resist such changes.

Datasets used: Twitter Data and Weibo Data



Method	Weibo					Twitter				
	Class	Acc.	Prec.	Rec.	F_1	Class	Acc.	Prec.	Rec.	F_1
GRU	R	0.839	0.757	0.789	0.773	R	0.852	0.773	0.784	0.779
	N		0.885	0.865	0.875	N		0.892	0.886	0.889
Text-CNN	R	0.807	0.802	0.748	0.774	R	0.839	0.757	0.789	0.773
	N		0.811	0.853	0.831	N		0.885	0.865	0.875
HAN	R	0.833	0.924	0.704	0.799	R	0.851	0.757	0.789	0.773
	N		0.782	0.949	0.857	N		0.885	0.865	0.875
$RvNN_{BU}$	R	0.903	0.909	0.859	0.884	R	0.789	0.782	0.793	0.788
	N		0.894	0.935	0.916	N		0.795	0.784	0.790
$RvNN_{TD}$	R	0.847	0.810	0.843	0.826	R	0.824	0.829	0.817	0.823
	N		0.877	0.851	0.864	N		0.818	0.830	0.824
EANN	R	0.866	0.872	0.808	0.838	R	0.794	0.811	0.735	0.771
	N		0.863	0.911	0.886	N		0.782	0.847	0.813
GAN-GRU	R	0.867	0.854	0.859	0.856	R	0.783	0.761	0.825	0.792
	N		0.878	0.874	0.876	N		0.809	0.741	0.773
GLAN	R	0.902	0.917	0.871	0.893	R	0.853	**0.871**	0.654	0.747
	N		0.891	0.931	0.910	N		0.847	**0.952**	0.896
CGAT	R	**0.940**	**0.959**	**0.906**	**0.932**	R	**0.892**	0.823	**0.871**	**0.846**
	N		**0.925**	**0.968**	**0.946**	N		**0.931**	0.903	**0.917**

Table 4. Comparison Results on the two datasets

3.2.7 Birds of a Feather Check Together: Leveraging Homophily for Sequential Rumour Detection

Homophily is used as a feature for detecting rumours, and the paper concluded that a user is more likely to post rumours if he/she follows other rumour mongers.

It learns from sequential dynamics of reporting, and finally concludes whether a post is a rumour or not on the basis of a user's social presence and influence in the network.

3.2.8 Graph-based Rumour Detection for Social Medi

From this paper, we understood that there are several features that need to be taken into consideration for rumour detection.

Firstly, the User features (such as the follower to following ratio, whether he/she is verified on twitter, etc.) play an important role in determining user credibility.

Second, the number of retweets a particular tweet receives gives us some insight, because one of the defining features of a rumour is the fact that it creates anxiety amongst the public and is viral.

This paper also does some analysis on the anomalies with regard to historical data as well.

3.2.9 Graph-based Text Classification: Learn from Your Neighbour

This paper mainly focuses on the strongest dependencies among immediate neighbours of a particular user (represented in the form of a node in the Twitter social graph).

It uses a relaxation labelling algorithm, in this case a Naive Bayes text classifier to assign class probabilities to each node.

Considers each page in turn and re-evaluates its class probabilities according to the class probabilities of its neighbours.

3.2.10 Text Classification: Exploiting Social Networks

This paper works on the graph analysis for text classification. This is done by extracting graph features from multiple graphs.

Following this, the features are embedded using GraphSAGE.

This paper provided further insight by comparing classification using

a) SVM

b) Extra-Trees

c) DNN

and further, suggested which model would be most appropriate for this purpose of graph based rumour detection.

3.2.11 A Graph-Theoretic Embedding-Based Approach for Rumor Detection in Twitter

POS Tagging involves labelling every word in a sentence with its appropriate part of speech. According to the english language, the current parts of speech include adjectives, nouns, verbs, pronouns and adverbs. The work done in this paper performs the task of POS tagging on the content present in a tweet.

On the basis of these different lexical categories it obtains post the task of tagging the sentence, it then classifies words as anxiety-causing or rumours-causing.

Further, this model uses the concepts of degree and closeness centrality to find all the prominent words.

3.2.12 Deep Structure Learning for Rumor Detection on Twitter

The model consists of three parts- first being the user encoder. This models the users attributes and behaviors based on graph convolutional networks to obtain user representation. Second, the propagation tree encoder which encodes the structure of the rumor propagation tree as a vector. Finally there is an integrator, which integrates the output of the above modules to identify rumours.

3.3 Stance Classification Models

The following set of papers deal with classifying the rumour into four categories:

a) Support: When a particular tweet agrees with the content in the source tweet

b) Deny: When a particular tweet disagrees with the content in the source tweet

c) Query: Asking a question, on the basis of the content in a tweet.

d) Comment : Stating a fact that does not particularly relate to the main intention of the tweet.

This type of classification method is known as SDQC.

3.3.1 Using Lists to Measure Homophily on Twitter

The work done in this paper focuses on extracting data from Twitter, and converting them into a list form as a source of information.

It associates a particular tag to each list, and on the basis of these tags, a direct/indirect following relationship between subscribers of the list and nature of the list is established.

This method is beneficial for extracting topics from the list and measuring similarity between all the listed users on the basis of the topics that the model has previously learnt on.

3.3.2 Modeling Conversation Structure and Temporal Dynamics for Jointly Predicting Rumor Stance and Veracity

This uses a Hierarchical framework to tackle rumor stance classification and veracity prediction jointly. This is achieved by exploiting both structural characteristic and temporal dynamics of the content in a particular tweet.

Further, this model encodes conversation structures for learning stance features.

3.4 Survey Papers on Existing Techniques

This section consists of all the papers that contain comparative performance analysis of various techniques implemented

3.4.1 Detection and Verification of Rumour in Social Media: A Survey (2017)

The authors approach the problem with a multi task learning model where each task's output is considered as an input to the next.

The problem is split as 4 tasks where the first 3 are auxiliary tasks and the last one is the primary task:

1. Rumour Detection: Determining if a tweet is a rumour or not
2. Rumour Propagation: To understand how the rumour flows across Twitter by collecting related information relevant to the rumour at hand.
3. Stance Classification: Determining a user's feelings towards the particular tweet
4. Veracity Verification: To determine if the rumour at hand is true, false or unverified.

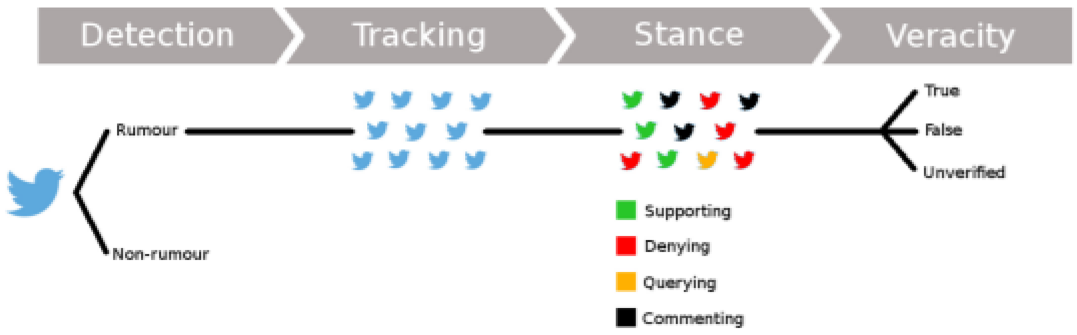


Figure 7. Multi task learning architecture for rumour detection and veracity verification

3.4.2 Rumour Detection, verification and controlling mechanisms in online social networks: A Survey(2019)

In this paper a binary classification system is used to classify a stream of posts into rumours and non rumours. Regular expressions are used to detect whether a post contains unverified information after which logistic regression along with NLP techniques are implemented to inspect the behaviour of users posting rumour information.

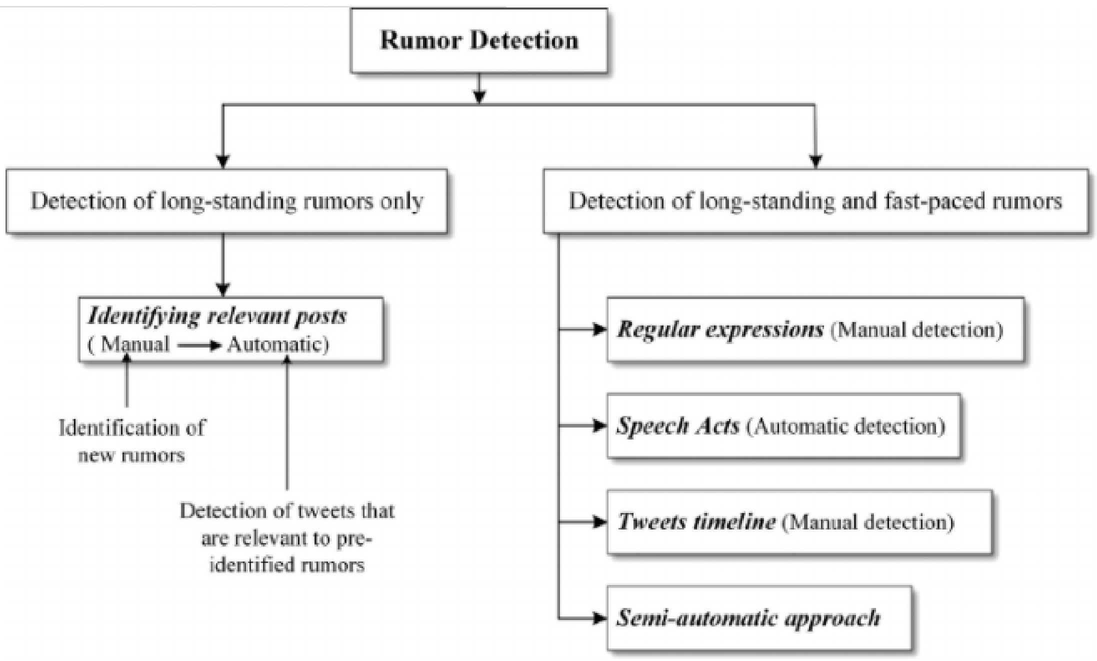
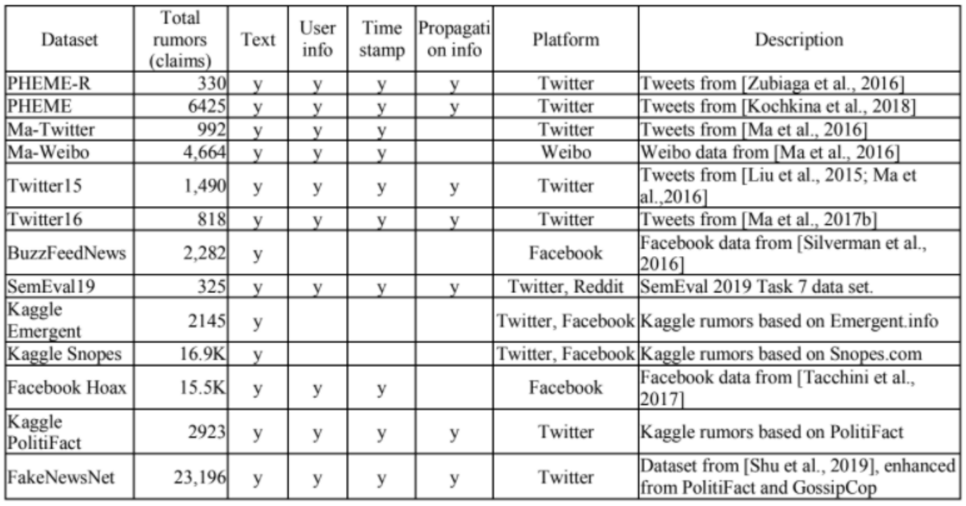

Rumor Detection
Detection of long-standing rumors only
Detection of long-standing and fast-paced rumors
Identifying relevant posts (Manual → Automatic)
Identification of new rumors
Detection of tweets that are relevant to pre-identified rumors
Regular expressions (Manual detection)
Speech Acts (Automatic detection)
Tweets timeline (Manual detection)
Semi-automatic approach

Figure 8. Methods to detect rumours which are long standing and fast paced

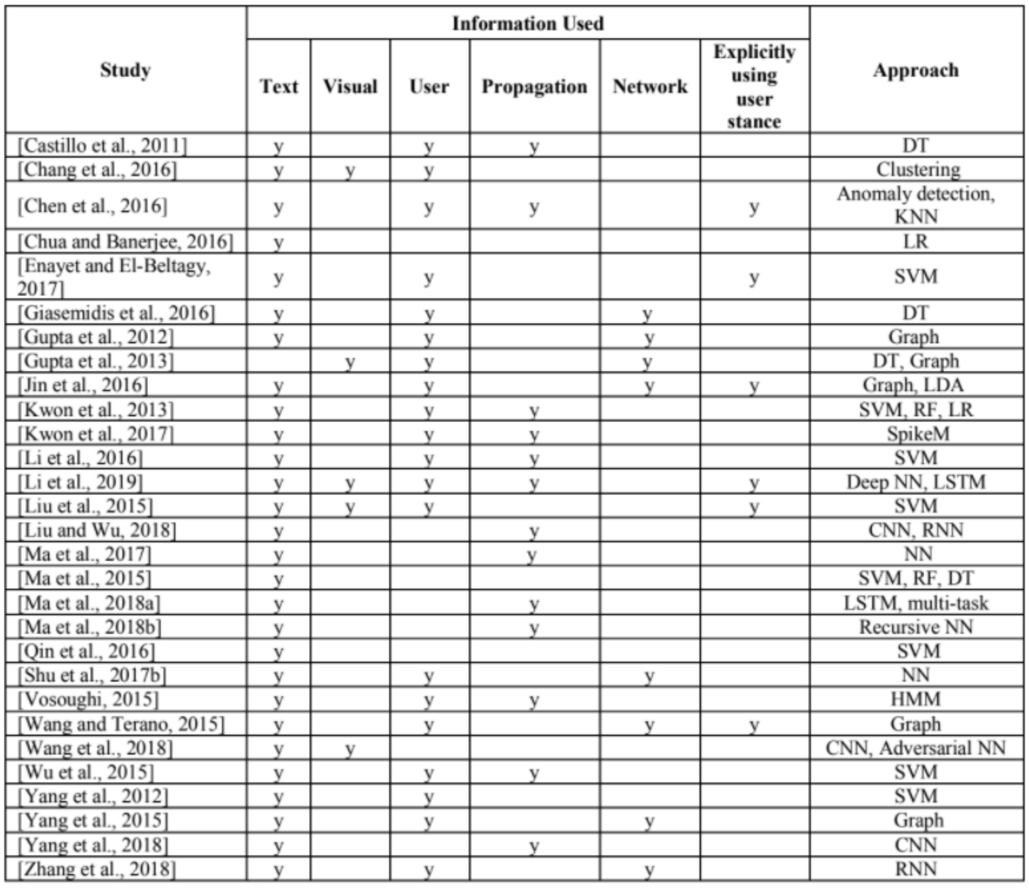
3.3.3 Rumour Detection on Social Media: Datasets, Methods and Opportunities

This paper provides an overall outline of the recent research done in the field of rumour detection. A detailed list of datasets and comparative results are elaborated in this paper.



Dataset	Total rumors (claims)	Text	User info	Time stamp	Propagation info	Platform	Description
PHEME-R	330	y	y	y	y	Twitter	Tweets from [Zubiaga et al., 2016]
PHEME	6425	y	y	y	y	Twitter	Tweets from [Kochkina et al., 2018]
Ma-Twitter	992	y	y	y		Twitter	Tweets from [Ma et al., 2016]
Ma-Weibo	4,664	y	y	y		Weibo	Weibo data from [Ma et al., 2016]
Twitter15	1,490	y	y	y	y	Twitter	Tweets from [Liu et al., 2015; Ma et al.,2016]
Twitter16	818	y	y	y	y	Twitter	Tweets from [Ma et al., 2017b]
BuzzFeedNews	2,282	y				Facebook	Facebook data from [Silverman et al., 2016]
SemEval19	325	y	y	y	y	Twitter, Reddit	SemEval 2019 Task 7 data set.
Kaggle Emergent	2145	y				Twitter, Facebook	Kaggle rumors based on Emergent.info
Kaggle Snopes	16.9K	y				Twitter, Facebook	Kaggle rumors based on Snopes.com
Facebook Hoax	15.5K	y	y	y		Facebook	Facebook data from [Tacchini et al., 2017]
Kaggle PolitiFact	2923	y	y	y	y	Twitter	Kaggle rumors based on PolitiFact
FakeNewsNet	23,196	y	y	y	y	Twitter	Dataset from [Shu et al., 2019], enhanced from PolitiFact and GossipCop

Table 5. List of datasets available for rumour veracity verification



Study	Information Used						Approach
	Text	Visual	User	Propagation	Network	Explicitly using user stance	
[Castillo et al., 2011]	y		y	y			DT
[Chang et al., 2016]	y	y	y				Clustering
[Chen et al., 2016]	y		y	y		y	Anomaly detection, KNN
[Chua and Banerjee, 2016]	y						LR
[Enayet and El-Beltagy, 2017]	y		y			y	SVM
[Giasemidis et al., 2016]	y		y		y		DT
[Gupta et al., 2012]	y		y		y		Graph
[Gupta et al., 2013]		y	y		y		DT, Graph
[Jin et al., 2016]	y		y		y	y	Graph, LDA
[Kwon et al., 2013]	y		y	y			SVM, RF, LR
[Kwon et al., 2017]	y		y	y			SpikeM
[Li et al., 2016]	y		y	y			SVM
[Li et al., 2019]	y	y	y	y		y	Deep NN, LSTM
[Liu et al., 2015]	y	y	y			y	SVM
[Liu and Wu, 2018]	y			y			CNN, RNN
[Ma et al., 2017]	y			y			NN
[Ma et al., 2015]	y						SVM, RF, DT
[Ma et al., 2018a]	y			y			LSTM, multi-task
[Ma et al., 2018b]	y			y			Recursive NN
[Qin et al., 2016]	y						SVM
[Shu et al., 2017b]	y		y		y		NN
[Vosoughi, 2015]	y		y	y			HMM
[Wang and Terano, 2015]	y		y		y	y	Graph
[Wang et al., 2018]	y	y					CNN, Adversarial NN
[Wu et al., 2015]	y		y	y			SVM
[Yang et al., 2012]	y		y				SVM
[Yang et al., 2015]	y		y		y		Graph
[Yang et al., 2018]	y			y			CNN
[Zhang et al., 2018]	y		y		y		RNN

Table 6. Information on previous studies conducted

CHAPTER- 4

PROJECT REQUIREMENTS SPECIFICATION

4.1 Dataset

Some of the initial studies done on rumor detection can be traced back to 1998, as a patent filed by Microsoft Corporation. This study focused on identifying influential rumormongers through resource usage data using a greedy graph covering algorithm. Since then, detection and evaluation of rumors has come a long way with the help of Natural Language Processing and deep-learning.

We have used different datasets for different subtasks. Although the structures of all are the same, they vary slightly in terms of the features and qualities that they capture and represent.

The data is organised as undermentioned: The dataset consists of 9 events, each of which is divided into two main subfolders: the rumours and the non-rumours. These folders in turn consist of tweet-id folders. These tweet ids correlate to the tweet id of the associated source tweet. Each tweet id folder has four main components: the source tweet folder, the replies folder, the structure or flow of conversation in json (structure.json) and the labels associated with that particular conversation (such as veracity) in json (annotations.json).

As long as a dataset with a similar structure and organisation is used, the model should perform in the same manner.

We have decided to use one of the most extensive datasets, the Pheme dataset for the purpose of rumour detection ie.determining whether a tweet contains information that is verified or not. Many variations of this dataset exist, but we use the Pheme dataset for Rumor detection and Veracity Verification. It is a popular research intended compilation of Twitter rumours and non-rumours posted during breaking news.

However, for the purpose of the rest of the subtasks, namely stance classification and rumour veracity verification, we use the SemEval2019 Dataset. This consists of Twitter as well as Reddit data. It shares the same structure as Pheme, but captures more features about the user - such as the number of followers and the number of people he/she is following, user verification, number of retweets etc. This helps determine the user credibility, which is an important determining factor for whether a tweet is a rumour or not.

4.2 Software Requirements

- Python Interpreter: To run the model as a python file.
- Python modules:
  - os module (*inbuilt*): provides a portable way of using system functionality. It is mostly used to parse and process the dataset.
  - json module (*ver 2.0.9*): used to process data in json format.
  - scikit-learn module (*ver 0.23.2*): machine learning library that supports supervised and unsupervised learning.
  - nltk (*ver 3.5*): platform to build python programs to work with human language data.
  - transformers (*ver 3.5.1*): library with NLP-oriented architectures of pre-trained models
  - pytorch (*ver 1.7.0*): open-source library that is based on the Torch machine learning library.
  - numpy (*ver 1.19.0*): module that will assist in working with large numerical data.
  - re (*ver 2.2.1*): helps evaluate regular expressions in textual data.
  - word2vec module to capture the content of a tweet
  - torchmoji to capture the emotions associated with a particular tweet

4.3 Functional Requirements

We clean and pre-process the rumours and non-rumours dataset before we train our model using this data. From all the available fields, we extract the tweet id, the contents of the tweet and it's corresponding label (true, false or unverifiable). We also remove all emoticons from this content. To be precise, first the hashtags are removed, followed by the user mentions and non-alphabetic characters. Then, the entire text is converted to lowercase, tokenized are finally converted to vectors. The vectors are then padded to make them all of equal length. Finally, the labels are converted to one hot vectors.

- Padding vectors to make them equal length
- Convert labels to one hot vectors

The following features are taken into consideration to capture the stance of the tweet :

- Content of the tweet
- Count of negative words
- Presence of punctuations
- Relation to source tweet and previous tweet
- Emotions associated with the tweet

Similarities and differences between tweets are captured using the cosine similarity concept.

After constantly fine tuning and optimizing for a certain number of epochs, the model is finally validated using a subset of the dataset and its precision metrics are calculated.

4.4 Non-Functional Requirements

4.4.1 Robustness

Twitter data can have a lot of noise in terms of spelling errors, emoticons, elongation of words, slangs and abbreviations. The dataset should be appropriately cleaned in different situations. It may later on help to extend this application over other social media applications keeping data format in mind.

4.4.2 Accuracy metrics

The application should yield a higher recall as a false negative might have a more serious and costlier impact than a false positive.

4.4.3 Speed/ Response Time

Considering the aim is to ultimately use such a program commercially, it would perhaps help to be able to use this in real time. It would need to be fast and efficient. Upgrading hardware or using cloud services might help. The earlier it detects a rumour, the lesser costs are involved in mitigating its effects.

CHAPTER- 5

SYSTEM REQUIREMENTS SPECIFICATION

5.1 Operating System

For this implementation, we have developed the application over a Windows 10 OS with the intel core i5 8th Gen processor. Using a multi-threaded paradigm or a cloud-based service can help improve non-functional requirements of the application.

5.2 General Constraints, Assumptions and Dependencies

5.2.1 Constraints

- **Legal**: All data used in this application is done so in compliance with the Twitter Developer Agreement and Policy. Any future research/commercial/personal use of the application must be done in conforming with the Twitter Developer Agreement and Policy, keeping data format in mind.
- **Usage Limitation**: This application can only work on linguistic content, not images attached to posts.

5.2.2 Assumptions

- **Normalization**: Language used on Social media tends to include slangs that are not grammatically or syntactically accurate (such as “I loooooovvvvvveeeeee Putin”). We assume our data doesn’t have any such terms.
- **Label Accuracy**: We have assumed that the labels attached to the dataset is accurate as they have manually been labelled.

CHAPTER - 6
HIGH LEVEL DESIGN

6.1 Novelty

Rumours in itself mean that it is unverified and hence may or may not come from a credible source. During some unexpected sudden events such as a terrorist attack or a natural calamity, people tend to refer to social media platforms before conventional media such as news channels because the latter take much more time to publish news owing to the fact that data needs to be first collected and then verified. This delay is usually not desirable especially when people are in a state of panic, and hence their first instinct is to collect whatever information that they can gather from sites such as Twitter. This entire process has been taken advantage of, because the users’ primary focus shifts from retaining the veracity of the news to adding extra information to make it more appealing to the public.

There have been various attempts in the past to detect and verify streaming rumours on social media. However, this is a slightly complex task considering that we are dealing with socio-linguistic4 data. The previous attempts show average results, which can be greatly improved. The novelty in our research project is that we are combining a content based learning model as well as a social graph aspect into finding the solution to our research problem. We aim to build a joint learning model for socio-linguistic rumour classification which predicts its veracity based on the content and the user’s stance in a social graph.

6.2 Innovativeness

When we build a social graph using Twitter data, the nodes are the Twitter users, and the links are the explicit connections in between users, which could either be a *follow, retweet* or *reply.* These social graphs are beneficial in such a situation because we can understand how it propagates through users, and understand the user's stance on the tweets as well. When using social graphs, users leave their digital footprints on social media. Further it uses the Principle of Homophily , which can help us understand a user's behaviour and credibility from his/her connections. Further, social information flow (understanding how data flows and by whom) and socialization behaviours help us draw implicit social graphs from the data (for example - retweet, reply, forward on twitter). We plan to incorporate a social graph based learning into our content based model and construct a joint learning from the two. We innovate by combining two learning methods into our model and improve the overall effectiveness and accuracy.

6.3 Performance

The entire problem can be broken into four auxiliary steps, which are rumour detection, rumour tracking, user stance identification and veracity verification. The current systems that deal with the aforementioned problem using content based, propagation based, and sequence based models have done extensive research on only the last 2 auxiliary problems and not much on the first two. They have obtained average results in the user stance and veracity verification sections of the problem, however this can be improved upon greatly. Alongside this, the domains of rumour detection and tracking need to undergo high amounts of enhancement as well. Through our combination of using a graph based and content based model, we plan to achieve this refinement and obtain an extremely high performing model, with good results.

6.4 Reliability

Our model relies heavily on data from Twitter, because it makes veracity decisions on the bases of the Tweets obtained. Every time there is some type of breaking news, Twitter is the fastest means to transmit the intricacies about this event because conventional media takes time as it is required to verify information before broadcasting it. Hence, in such a high pressure situation, people usually rely on twitter for a first hand and quick recount of events and experiences. News will always, irrespective of the place or date or time, will be displayed on Twitter seconds after an important event, and hence our model will never have a shortage of data about any event. As soon as tweets are received, it will be fed into our model, which can predict whether it should be believed or not. Hence, our model shows high levels of reliability.

6.5 Maintainability

Our model mostly relies on Twitter data. The maintenance of this data is relatively easy because this data has a fixed format and character limit. Hence, the algorithm used currently to preprocess and clean the data will not change. This makes our model extremely easy to maintain and would not require a cleaning or a check up often.

6.6 Reusability

Our current project is applicable only towards Twitter data. Other social media sites such as Reddit or Instagram, have not been considered. So, our model cannot be reused for data posted on other applications, but can be reused whenever Twitter data is being studied.

CHAPTER - 7

LOW LEVEL DESIGN

Algorithm Used

```
Input: $conversationthread$
Output: $Non-Rumour/TrueRumour/FalseRumour/UnverifiedRumour$
[1] $n \leftarrow$ no of replies
$graphstruct \leftarrow$graph struct
$setofreplies \leftarrow [r_1, r_2, ....., r_n]$
full text $\leftarrow$ source text
$i \leftarrow 1$ to $N$

    $replytext \leftarrow$set of replies$_i$

    $fulltext = fulltext.concatenate($reply text$)$

    $i \leftarrow i + 1$

$detectiontext = BERTModel(fulltext)$
$detectiongraph = RFClassifier(graphstruct)$
$finaldetection = JointModel(detectiontext, detectiongraph)$

IF$finaldetecion == NonRumour$ then
EXIT

$stanceemoji = []$
i $\leftarrow 1$ to $N$

    $stance = Torchmoji($set of replies$_i)$

    $stanceemoji$ =stance emoji.append($stance$)

stance bilstm= []
$i \leftarrow 1$ to $N$

    $stance = Bilstm($set of replies$_i)$

    $stancebilstm$ =stance bilstm.append($stance$)

$setofstances$ =joint stance($stanceemoji$,stance bilstm)
veracity= $veracitybilstm($set of stances$)$
```

7.1 Text Based Rumour Detection

For the purpose of rumour detection, we have used a BERT transformer. Our model is given all the related tweets and then performs various forms of data cleaning and data preprocessing. First off, all the Emoji characters are removed, making the text more easy to comprehend and understand. Next, the words in the Tweet are embedded using BERT which performs deep bidirectional embedding, trying to capture all token and position related information to produce a real-valued vector[5]. The elements of these vectors serve as parameters to our model.

For the tokenizer, we use the "bert-base-uncased" version of BertTokenizer. Using TorchText, we first create the Text Field and the Label Field. The Text Field will be used for containing the news articles

and the Label is the true target. We limit each article to the first 128 tokens for BERT input. Then, we create a TabularDataset from our dataset csv files using the two Fields to produce the train, validation, and test sets. Then we create Iterators to prepare them in batches. We used the “bert-base-uncased” version of BERT, which is the smaller model trained on lower-cased English text (with 12-layer, 768-hidden, 12-heads, 110M parameters).
Finally, the embedded vectors in the Tweet are stored in a database, for easy data retrieval and access. The tweets are then fed into our transformer model, as part of the training data and our model learns the optimal weights for different parameters through backpropagation. These weights are fine-tuned further using Adam optimization.

We used a suitable learning rate to tune BERT for 5 epochs. We use BinaryCrossEntropy as the loss function. The output is passed through Sigmoid before calculating the loss between the target and itself. During training, we evaluate our model parameters against the validation set. We save the model each time the validation loss decreases so that we end up with the model with the lowest validation loss, which can be considered as the best model.

The next stage of our project is where the user and his/her input comes into play. A user wants to verify if a given tweet with a particular tweet ID contains information that is true or not. The user inputs the tweet into our model. The same procedure of Emoji character removal and word vectorisation is followed, and done as part of the data preprocessing phase. Following this, our model predicts whether the content of the tweet is agreeing/ denying/questioning a topic, rather than stating a topic.

7.2 Graph Based Rumour Detection

An important feature that needs to be captured is the propagation structure of the tweet- how exactly it travels through the social network of Twitter and how it moves from one user to another. This can be captured by the Tweet’s *favourite* and *retweet* count. For every tweet, we create a dictionary for all 5 of these features, with the key as the feature name and the corresponding value of that feature. The conversation thread is then represented as a dictionary of feature vectors in a tree structure. This dictionary and all associated features and then vectoried using the sklearn DictVectorizer. Following this, the obtained vector is being fed into a random forest classifier, which does the task of classifying it as a rumour or not.

7.3 Stance Classification

Using Torchmoji and Bi-LSTM

Here, we classify whether a given tweet is supporting, denying, querying or commenting the given source tweet. The stance of a tweet is being classified with the help of layers of LSTM units and uses factors such as cosine similarity and the number of negative words/ punctuations to determine how similar or different the reply tweet is from the source.

The input at each time step of the LSTM layers is a reply from the conversation thread and the output at each time step is the stance of the reply. The layers of LSTM units help understand the sequential structure of the conversation tree which includes the source and reply tweets.

We have used a neural network architecture that uses layers of LSTM units to process the whole branch of tweets, thus incorporating structural information of the conversationThe input at each time step i of the LSTM layer is the representation of the tweet as a vector. We record the output of each time step so as to attach a label to each tweet in a branch5 . This output is fed through several dense ReLU layers, a 50% dropout layer, and then through a softmax layer to obtain class probabilities. We use zero-padding and masks to account for the varying lengths of tweet branches. The model is trained using the categorical cross entropy loss function. Since there is overlap between branches originating from the same source tweet, we exclude the repeating tweets from the loss function using a mask at the training stage. Further, we are also using the Torchmoji module with which emoticons are assigned to a particular tweet, based on the emotion present in the content of the tweet.

7.4 Veracity Verification

To determine whether the Tweet is True, False or Unverified, we use a combination of the outputs obtained from the Torchmoji model and the Bi-LSTM in the stance classification phase, as well as another Bi-LSTM which is solely focussed on predicting the veracity of the tweet. We have assigned a particular weightage to the outputs obtained from the different models, which helps us obtain one outcome. We have chosen a Bi-LSTM for this purpose because here we feed the learning algorithm with the original data once from beginning to the end to learn past information, and once from end to beginning, to learn future information. Also,it usually learns faster than a one-directional approach.

CHAPTER - 8
SYSTEM DESIGN

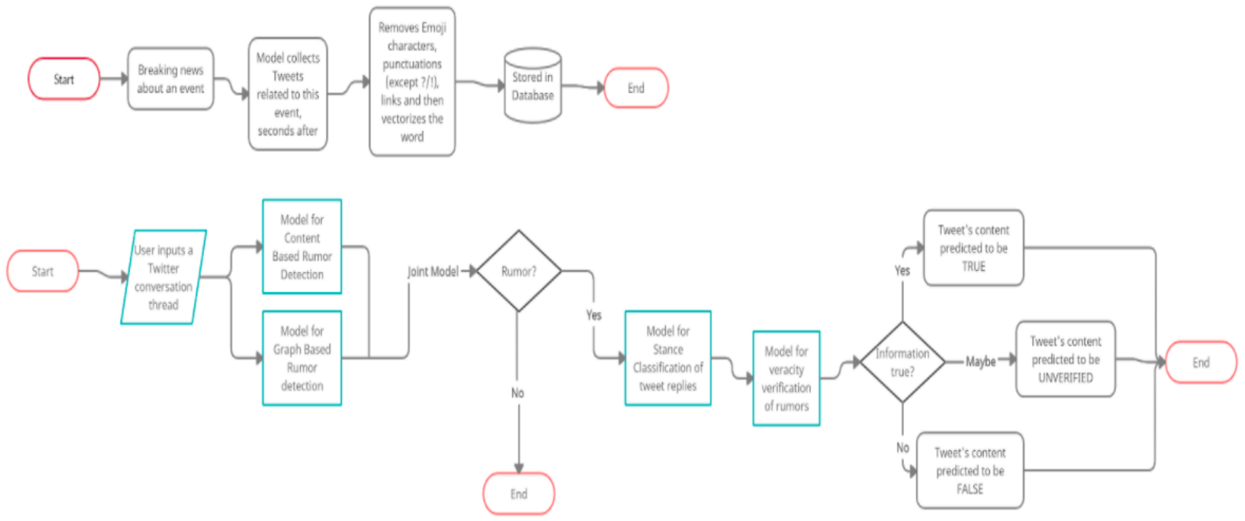

Start
Breaking news about an event
Model collects Tweets related to this event, seconds after
Removes Emoji characters, punctuations (except ?/!), links and then vectorizes the word
Stored in Database
End
Start
User inputs a Twitter conversation thread
Model for Content Based Rumor Detection
Model for Graph Based Rumor detection
Joint Model
Rumor?
Yes
No
End
Model for Stance Classification of tweet replies
Model for veracity verification of rumors
Information true?
Yes
Tweet's content predicted to be TRUE
Maybe
Tweet's content predicted to be UNVERIFIED
No
Tweet's content predicted to be FALSE
End

Figure 9. Proposed Methodology

Social Media sites such as Twitter and Reddit are usually the first ones to get first hand accounts of breaking news, because conventional media sources take time to publish information because it needs to be verified before broadcasting it. Hence, most of the population relies heavily on social media sites, mainy Twitter, for an instant recount of drastic events that have taken place, seconds after they have occured.

Initially, right after an event occurs, there are numerous tweets about this particular event, which can all be tagged because they will all use a common hashtag, something that describes the event and its happenings.

For the purpose of rumour detection, we have used a BERT transformer. Our model is given all the related tweets and then performs various forms of data cleaning and data preprocessing. First off, all the Emoji characters are removed, making the text more easy to comprehend and understand. Next, the words in the Tweet are embedded using BERT which performs deep bidirectional embedding, trying to capture all token and position related information to produce a real-valued vector. The elements of these vectors serve as parameters to our model.

Finally, the embedded vectors in the Tweet are stored in a database, for easy data retrieval and access. The tweets are then fed into our transformer model, as part of the training data and our model learns the optimal weights for different parameters through backpropagation. These weights are fine-tuned further using Adam optimization.

The next stage of our project is where the user and his/her input comes into play. A user wants to verify if a given tweet with a particular tweet ID contains information that is true or not. The user inputs the tweet into our model. The same procedure of Emoji character removal and word vectorisation is followed, and done as part of the data preprocessing phase. Following this, our model predicts whether the content of the tweet is agreeing/ denying/questioning a topic, rather than stating a topic.

Next, we have the graph based model for rumour detection and veracity verification. For every tweet- we extract 5 features from it, namely:

- Number of followers of the user
- Number of accounts the user follows
- Whether the user is verified on Twitter or not
- Tweet's favourite count
- Tweet's retweet count

An important feature that needs to be captured is the propagation structure of the tweet- how exactly it travels through the social network of Twitter and how it moves from one user to another. This can be captured by the Tweet's *favourite* and *retweet* count. For every tweet, we create a dictionary for all 5 of these features, with the key as the feature name and the corresponding value of that feature. This dictionary and all associated features and then vectoried using python modules. Following this, the obtained vector is being fed into a random forest classifier, which does the task of classifying it as a rumour or not. The obtained model gives us an accuracy of 73%.

Finally, we combine both the rumour detection models by assigning a weightage to each model on the basis of its accuracy. We also look into the confidence scores offered by both of the models and then make an informed decision on what has a higher weightage. After combining both models, we have successfully obtained an accuracy of 93%.

On the basis of this, it decides whether the tweet is a rumour or not. If not, the tweet is dropped. Based on content on individual tweets and their graph perspectives, we classify them as rumors or not. With this, we are able to detect rumors.

The next phase of our project deals with the stance classification, which classifies whether a given tweet is supporting, denying, querying or commenting the given source tweet. The stance of a tweet is being classified with the help of a Bi-LSTM and uses factors such as cosine similarity and the number of negative words/ punctuations to determine how similar or different the reply tweet is from the source.

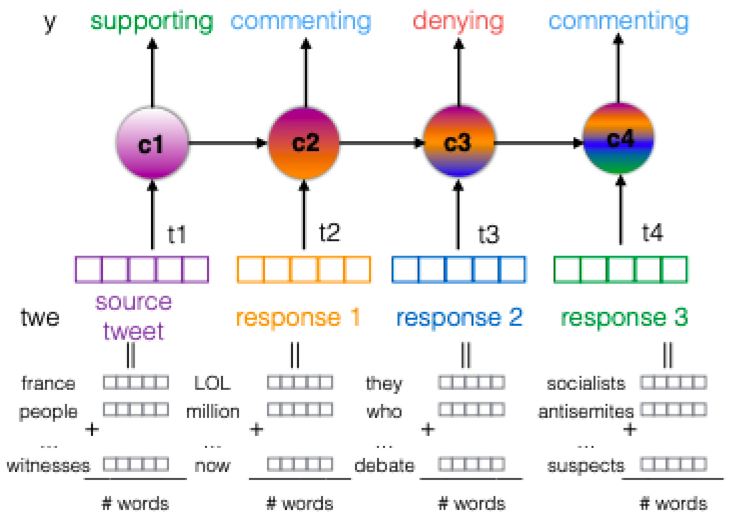


Figure 10: Illustration of the input/output structure of the LSTM-branch model for Stance Classification

In addition to this, we are using the torchmoji vector in python for better results. With the help of this module, emoticons are assigned to a particular tweet, based on the emotion present in the content of hte tweet. The emotion associated with the majority of the emoticons in the tweet is the emotion assigned to the tweet in itself. A combination of this module and the Bi-LSTM gives us an accuracy of 78%. Finally, is the veracity verification aspect where we finally produce the outcome of whether a Tweet is true, false or unverified. On the basis of the stance taken by all of the source tweet's replies, we use a Bi-LSTM to finally predict the outcome. This final model provides us with an accuracy of 92%. The model has three outcomes - True, False or unverified.

CHAPTER – 9

IMPLEMENTATION AND PSEUDOCODE

9.1 Graph Based Model

```
import os
import json
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction import DictVectorizer
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
from keras.models import Sequential
from keras.layers import Dense
from keras import optimizers
from sklearn.ensemble import RandomForestClassifier
import pickle

def get_features(file):
    data=json.load(file)
    struct={}
    struct["verified"]=data["user"]["verified"]
    struct["following"]=data["user"]["friends_count"]
    struct["followers"]=data["user"]["followers_count"]
    struct["retweets"]=data["retweet_count"]
    struct["favourites"]=data["favorite_count"]

    return(struct)

def create_tree(old_dict,rum,key,event,Bool_rum):
    new_dict = { }
    for key in old_dict.keys():
        # print(key)
        try:
            if(Bool_rum==True):
                file=open("D:/Courses/VII
Sem/Capstone/PHEME_veracity/PHEME_veracity/all-rnr-annotated-threads/"+event+"/rumours/"+str(ru
m)+"/reactions/"+str(key)+".json")
            else:
                file=open("D:/Courses/VII
Sem/Capstone/PHEME_veracity/PHEME_veracity/all-rnr-annotated-threads/"+event+"/non-rumours/"+st
r(rum)+"/reactions/"+str(key)+".json")
        except:
            if(Bool_rum==True):
                file=open("D:/Courses/VII
Sem/Capstone/PHEME_veracity/PHEME_veracity/all-rnr-annotated-threads/"+event+"/rumours/"+str(ru
m)+"/source-tweets/"+str(rum)+".json")
            else:
                file=open("D:/Courses/VII
Sem/Capstone/PHEME_veracity/PHEME_veracity/all-rnr-annotated-threads/"+event+"/non-rumours/"+st
r(rum)+"/source-tweets/"+str(rum)+".json"
        new_key = get_features(file)
        if isinstance(old_dict[key], dict):
            new_dict[new_key] = create_tree(old_dict[key],rum,key,event,Bool_rum)
        else:
            new_dict[new_key] = old_dict[key]
    return new_dict

def get_dataset():
    path="D:/Courses/VII Sem/Capstone/PHEME_veracity/PHEME_veracity/all-rnr-annotated-threads"
    raw_data={"id":[],"struct":[],"label":[]}
    files=os.listdir(path)
    events=[]
    for file in files:
        if os.path.isdir(os.path.join(os.path.abspath(path), file)):
            events.append(file)
    #print(events)
    for event in events:
        event_path_rumours="D:/Courses/VII
Sem/Capstone/PHEME_veracity/PHEME_veracity/all-rnr-annotated-threads/"+event+"/rumours"
        rumour_files=os.listdir(event_path_rumours)
        rumour_dirs=[]
        for f in rumour_files:
            if os.path.isdir(os.path.join(os.path.abspath(event_path_rumours), f)):
                rumour_dirs.append(f)
        for rum in rumour_dirs:
            file=open("D:/Courses/VII
Sem/Capstone/PHEME_veracity/PHEME_veracity/all-rnr-annotated-threads/"+event+"/rumours/"+rum+
"/source-tweets/"+rum+".json")
            struct=get_features(file)
            raw_data["id"].append(rum)
            raw_data["struct"].append(struct)
            raw_data["label"].append(1)
        event_path_nonrumours="D:/Courses/VII
Sem/Capstone/PHEME_veracity/PHEME_veracity/all-rnr-annotated-threads/"+event+"/non-rumours"
        non_rumour_files=os.listdir(event_path_nonrumours)
        non_rumour_dirs=[]
        for f in non_rumour_files:
            if os.path.isdir(os.path.join(os.path.abspath(event_path_nonrumours), f)):
                non_rumour_dirs.append(f)
        for nrum in non_rumour_dirs:
            file=open("D:/Courses/VII
Sem/Capstone/PHEME_veracity/PHEME_veracity/all-rnr-annotated-threads/"+event+"/non-rumours/"+n
rum+"/source-tweets/"+nrum+".json")
            struct=get_features(file)
            raw_data["id"].append(nrum)
            raw_data["struct"].append(struct)
            raw_data["label"].append(0)


    X_train, X_test, y_train, y_test = train_test_split(raw_data['struct'], raw_data['label'], test_size=0.3,
random_state=2018,stratify=raw_data['label'])

    return X_train, X_test, y_train, y_test

train_text,temp_text, train_labels, temp_labels = get_dataset()
print(temp_text,temp_labels)
v = DictVectorizer(sparse=False)
train_x=v.fit_transform(train_text)
test_x=v.fit_transform(temp_text)
print(train_x)
clf=RandomForestClassifier(n_estimators=100)
clf=clf.fit(train_x,train_labels)
y_pred=clf.predict(test_x)
```

```
filename = 'rf_tree.pkl'
pickle.dump(clf, open(filename, 'wb'))

print(confusion_matrix(temp_labels,y_pred))
print(classification_report(temp_labels,y_pred))

loaded_model = pickle.load(open(filename, 'rb'))
result = loaded_model.score(test_x, temp_labels)
print(result)
```

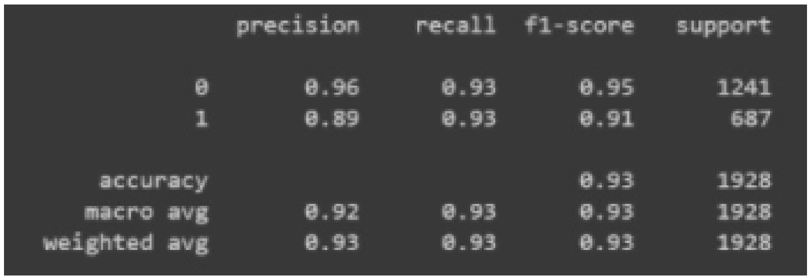
Results:

9.2 Text Based Rumour Detection

Using the BERT classifier

```
device = "cuda" if torch.cuda.is_available() else "cpu"

bert = AutoModel.from_pretrained('bert-base-uncased')
tokenizer = BertTokenizerFast.from_pretrained('bert-base-uncased')
def drop_emoji(x):
  x= re.sub(r"http\S+", "", x)
  x=re.sub(r"[^\w\d\!\?\s]+","",x)
  drop_list = []
  for i,_x in enumerate(x):
    if ord(_x) >= 128:
      drop_list.append(i)
  new_x = []
  for i,_x in enumerate(x):
    if i not in drop_list:
      new_x.append(_x)
  return ''.join(new_x)



# print(tweets)
tweets=list(map(drop_emoji,tweets))
#print(tweets)
tokens_test = tokenizer.batch_encode_plus(
  tweets,
    max_length = 100,
    pad_to_max_length=True,
    truncation=False,
    padding=True
)

test_seq = torch.tensor(tokens_test['input_ids'])
test_mask = torch.tensor(tokens_test['attention_mask'])

for param in bert.parameters():
    param.requires_grad = False

classes=["Non-Rumour","Rumour"]
model = BERT_Arch(bert)
path = '/content/drive/MyDrive/CAPSTONE PROJECT - PRAJNA SIRISHA
SUKANYA/code/saved_weights.pt'
model.load_state_dict(torch.load(path),strict=False)

with torch.no_grad():

  pred_logit=model(test_seq.to(device), test_mask.to(device))
  results = torch.softmax(pred_logit, dim=1).tolist()[0]
  print(pred_logit)
  preds = pred_logit.detach().cpu().numpy()
  print(preds)

for i in range(len(classes)):
    print(f"{classes[i]}: {round(results[i] * 100)}%")
prob_nr=results[0]
prob_r=results[1]
pred_text=""
preds =list(np.argmax(preds, axis = 1))
print(preds)
for i in range(len(preds)):
    if(preds[i]==0):
        preds[i]="Non-Rumour"
        pred_text=preds[i]
    else:
      preds[i]="Rumour"
      pred_text=preds[i]
for i in range(len(tweets)):
    print(i+1,tweets[i],"\n",preds[i])

diff1=abs(prob_nr-prob_r)
prob_nr_text=prob_nr
prob_r_text=prob_r

filename = '/content/drive/MyDrive/CAPSTONE PROJECT - PRAJNA SIRISHA
SUKANYA/code/rf_tree.pkl'
loaded_model = pickle.load(open(filename, 'rb'))

preds=loaded_model.predict(tweets_x)
y_pred=loaded_model.predict_proba(tweets_x)
print(y_pred)
prob_nr=y_pred[0][0]
prob_r=y_pred[0][1]
pred_graph=""
for pred in preds:
  if(pred==0):
    print("Non-Rumour")
    pred_graph="Non-Rumour"
  else:
    print("Rumour")
    pred_graph="Rumour"
diff2=abs(prob_nr-prob_r)
prob_nr_graph=prob_nr
prob_r_graph=prob_r

sum1=(-1*(prob_nr_text)+prob_r_text)*(0.73)
sum2=(-1*(prob_nr_graph)+prob_r_graph)*(0.73)
final_sum=sum1+sum2
if(final_sum<0):
  detection="Non-Rumour"
  print("Final Detection: Non-Rumour")
else:
  detection="Rumour"
  print("Final Detection: Rumour")
if(detection=="Non-Rumour"):
  exit(0)
```

Results:



```
              precision    recall  f1-score   support

           0       0.96      0.93      0.95      1241
           1       0.89      0.93      0.91       687

    accuracy                           0.93      1928
   macro avg       0.92      0.93      0.93      1928
weighted avg       0.93      0.93      0.93      1928
```

9.3 Stance Classification

-*Using Bi-LSTM*

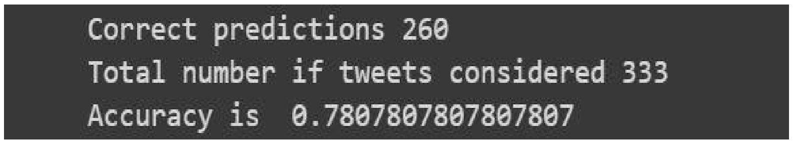
Code for preprocessing the data

```
import numpy as np
import nltk
from nltk.corpus import stopwords
import gensim
from gensim.models import Word2Vec
import re
from copy import deepcopy

def word_to_vec_sum(Tweet_, avg=True):
    global model_GoogleNews
    model = model_GoogleNews
    num = 300
    temp = np.zeros(num)
    word_list = get_word_list(Tweet_['text'], Tweet_, remove_stopwords=False)
    for w in range(len(word_list)):
        if word_list[w] in model:
            temp += model[word_list[w]]
    if avg and len(word_list) != 0:
        sum_w2v = temp/len(word_list)
    else:
        sum_w2v = temp
    return sum_w2v

def loadW2vModel():
    global model_GoogleNews
    model_GoogleNews =
gensim.models.KeyedVectors.load_word2vec_format('C:/Users/Siri/Downloads/GoogleNews-vectors-negativ
e300 (1).bin/GoogleNews-vectors-negative300.bin', binary=True)

def cosinesimiliarity(words, wordssrc):
    global model_GoogleNews
    model = model_GoogleNews
    words_2 = []
    for w in words:
        if w in model.wv.vocab:
            words_2.append(w)
    word_src = []
    for w in wordssrc:
        if w in model.wv.vocab:
            word_src.append(w)
    if len(words_2) > 0 and len(word_src) > 0:
        return model.n_similarity(words_2, word_src)
    return 0
def get_word_list(Tweet_text, Tweet_, remove_stopwords=False):
    str_text = re.sub("[^a-zA-Z]", " ", Tweet_text)
    words = nltk.word_tokenize(str_text.lower())
    if remove_stopwords:
        stops = set(stopwords.words("english"))
        words = [w for w in words if w not in stops]
    return(words)
```

Code for the models

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Dropout, LSTM
from keras.layers import TimeDistributed, Masking
from keras import optimizers
from keras import regularizers
#from keras.models import load_model
import json
#%%

def LSTM_model_veracity(x_train, y_train, x_test, params,eval=False ):
    no_units = int(params['no_units'])
    no_layers = int(params['no_layers'])
    no_dunits = int(params['no_dunits'])
    no_dlayers = int(params['no_dlayers'])
    epochs = params['epochs']
    lr = params['lr']
    size = params['size']
    l2reg = params['l2reg']
    model_veracity = Sequential()
    features_n = x_train.shape[2]
    model_veracity.add(Masking(mask_value=0., input_shape=(None, features_n)))
    for n in range(no_layers-1):
        model_veracity.add(LSTM(no_units, dropout=0.2, recurrent_dropout=0.2,
                      return_sequences=True))
    model_veracity.add(LSTM(no_units, dropout=0.2, recurrent_dropout=0.2,
                  return_sequences=False))
    for n in range(no_dunits):
        model_veracity.add(Dense(no_dlayers, activation='relu'))
    model_veracity.add(Dropout(0.5))
    model_veracity.add(Dense(3, activation='softmax',activity_regularizer=regularizers.l2(l2reg)))
    adam_opt = optimizers.Adam(lr=lr, beta_1=0.9, beta_2=0.999,epsilon=1e-08, decay=0.0)
    model_veracity.compile(optimizer=adam_opt, loss='categorical_crossentropy', metrics=['accuracy'])
    model_veracity.fit(x_train, y_train,batch_size=size,epochs=epochs, shuffle=True, class_weight=None,
verbose=0)

    if eval==True:
        model_veracity.save('output/my_model_veracity.h5')
        json_string = model.to_json()
        with open('output/model_architecture_veracity.json','w') as fout:
            json.dump(json_string,fout)
        model_veracity.save_weights('output/my_model_veracity_weights.h5')


    predprob = model_veracity.predict(x_test, batch_size=size, verbose=0)
    conf = np.max(predprob, axis=1)
    pred = model_veracity.predict_classes(x_test, batch_size=size)
    return pred, conf
def LSTM_model_stance(x_train, y_train, x_test, params,eval=False ):
    no_units = int(params['no_units'])
    no_layers = int(params['no_layers'])
    no_dunits = int(params['no_dunits'])
    no_dlayers = int(params['no_dlayers'])
    epochs = params['epochs']
    lr = params['lr']
    size = params['size']
    l2reg = params['l2reg']
    stance_model = Sequential()
    features_n = x_train.shape[2]
    stance_model.add(Masking(mask_value=0., input_shape=(None, features_n)))
    for n in range(no_layers-1):
        stance_model.add(LSTM(no_units, kernel_initializer='glorot_normal',dropout=0.2,
recurrent_dropout=0.2,return_sequences=True))
    stance_model.add(LSTM(no_units, kernel_initializer='glorot_normal',
                  dropout=0.2, recurrent_dropout=0.2, return_sequences=True))
    stance_model.add(TimeDistributed(Dense(no_dlayers, activation='relu')))
    for n in range(no_dunits-1):
        stance_model.add(TimeDistributed(Dense(no_dlayers, activation='relu')))
    stance_model.add(Dropout(0.5))
    stance_model.add(TimeDistributed(Dense(4, activation='softmax',
                     activity_regularizer=regularizers.l2(l2reg))))
    adam_opt = optimizers.Adam(lr=lr, beta_1=0.9, beta_2=0.999,epsilon=1e-08, decay=0.0)
    stance_model.compile(optimizer=adam_opt, loss='categorical_crossentropy',metrics=['accuracy'])
    stance_model.fit(x_train, y_train,batch_size=size,epochs=epochs, shuffle=True, class_weight=None,
verbose=0)
    if eval==True:
        stance_model.save('output/my_model_stance.h5')
        json_string = stance_model.to_json()
        with open('output/model_architecture_stance.json','w') as fout:
            json.dump(json_string,fout)
        stance_model.save_weights('output/my_model_stance_weights.h5')

    predprob = stance_model.predict(x_test, batch_size=size, verbose=0)
    conf = np.max(predprob, axis=2)
    pred = stance_model.predict_classes(x_test, batch_size=size)
    return pred, conf
```

Results:



```
Correct predictions 260
Total number if tweets considered 333
Accuracy is  0.7807807807807807
```

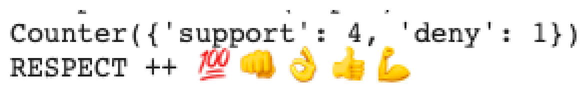
- ***Using Torchmoji***

```
!pip3 install torch==1.0.1 -f https://download.pytorch.org/whl/cpu/stable
!git clone https://github.com/huggingface/torchMoji
import os
os.chdir('torchMoji')
!pip3 install -e .
!python3 scripts/download_weights.py


import numpy as np
import emoji, json
from torchmoji.global_variables import PRETRAINED_PATH, VOCAB_PATH
from torchmoji.sentence_tokenizer import SentenceTokenizer
from torchmoji.model_def import torchmoji_emojis
import statistics
from statistics import mode
from collections import Counter
 EMOJIS = ":joy: :unamused: :weary: :sob: :heart_eyes: :pensive: :ok_hand: :blush: :heart: :smirk:
:grin: :notes: :flushed: :100: :sleeping: :relieved: :relaxed: :raised_hands: :two_hearts: :expressionless:
:sweat_smile: :pray: :confused: :kissing_heart: :heartbeat: :neutral_face: :information_desk_person:
:disappointed: :see_no_evil: :tired_face: :v: :sunglasses: :rage: :thumbsup: :cry: :sleepy: :yum:
:triumph: :hand: :mask: :clap: :eyes: :gun: :persevere: :smiling_imp: :sweat: :broken_heart:
:yellow_heart: :musical_note: :speak_no_evil: :wink: :skull: :confounded: :smile:
```

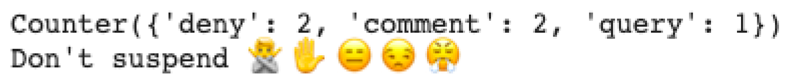
```
:stuck_out_tongue_winking_eye: :angry: :no_good: :muscle: :facepunch: :purple_heart:
:sparkling_heart: :blue_heart: :grimacing: :sparkles:".split(' ')


stance={"support":["joy","yum","clap","sparkles","heart_eyes","ok_hand","blush","heart","flush
ed","sparkling_heart","100","relieved","muscle","raised_hands","purple_heart","two_hearts","kis
sing_heart","blue_heart","v","heartbeat","sunglasses","thumbsup","smile","yellow_heart"],"deny"
:["sob","triumph","confounded","grin","cry","disappointed","tired_face","rage","sleepy","smiling
_imp","sweat","broken_heart","skull","angry","no_good","facepunch","grimacing"],"query":["pe
nsive","confused","neutral_face","information_desk_person","hand"],"comment":["unamused","st
uck_out_tongue_winking_eye","weary","speak_no_evil","smirk","expressionless","sweat_smile","p
ray","neutral_face","see_no_evil","eyes","gun","confounded","wink"]}
model = torchmoji_emojis('model/pytorch_model.bin')
with open(VOCAB_PATH, 'r') as f:
 vocabulary = json.load(f)
st = SentenceTokenizer(vocabulary, 30)
def deepmojify(sentence,top_n =5):
 def top_elements(array, k):
   ind = np.argpartition(array, -k)[-k:]
   return ind[np.argsort(array[ind])][::-1]
 tokenized, _, _ = st.tokenize_sentences([sentence])
 prob = model(tokenized)[0]
 emoji_ids = top_elements(prob, top_n)
 emojis = list(map(lambda x: EMOJIS[x], emoji_ids))
 stances=[]
 for e in emojis:
   em=e[1:len(e)-1]
   # print(em)
   for s in stance:
     if em in stance[s]:
       stances.append(s)
 # print(stances)
 print(Counter(stances))
 # print(mode(stances))
 return emoji.emojize(f"{sentence} {' '.join(emojis)}", use_aliases=True)
text = ['Get back to work.']
```

```
for _ in text:
 print(deepmojify(_, top_n = 5))
```

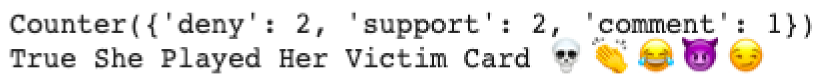
Results obtained from Torchmoji



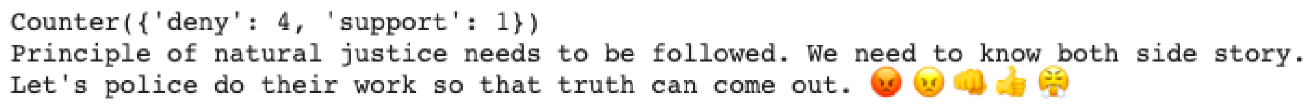
```
Counter({'support': 4, 'deny': 1})
RESPECT ++ 💯👊👌👍💪
```



```
Counter({'deny': 2, 'comment': 2, 'query': 1})
Don't suspend 🙅✋😑😒😤
```



```
Counter({'deny': 2, 'support': 2, 'comment': 1})
True She Played Her Victim Card 💀👏😂😈😏
```



```
Counter({'deny': 4, 'support': 1})
Principle of natural justice needs to be followed. We need to know both side story.
Let's police do their work so that truth can come out. 😡😠👊👍😤
```

CHAPTER- 10

CONCLUSION OF CAPSTONE PROJECT PHASE - 2

Through this project, we attempt to analyze rumors through, what we believe to be, two of its major features - its content, and the user's credibility in a social network, represented in the form of a graph.

One of our main concerns was the bias that could be introduced in our model weights due to our dataset, considering this dataset doesn't capture all categories of rumours (such as topic of the rumour subject, the user communication style,etc.) By using a model that has been pre-trained on the extensive Image. Net dataset (with 1.2B data points) and over 1000 classes, we predominantly take dataset bias into consideration.

Through the second phase of our Capstone project , we aim to analyse rumours on the basis of the other most important feature: Social Graphs.

- We selected extracted features from the dataset that help gauge the presence of an individual in a social network.
- We developed a model that could learn rumours over social graph related information like user relationships, rumour propagation patterns and more.
- Combined the loss functions of the linguistic based model and graph based model to generate a final model. This newly constructed model can classify a rumour using its two facets: sentiment relayed through text and behaviour identified in social graphs.
- Once combined, we performed a final analysis on this model with the firm belief that it shall surpass other existent models.

As observable from the decreasing training and validation loss, as well as the improving confusion matrix values through all the epochs, we can confidently claim that this model works efficiently.

It is very important to detect and verify rumours as soon as possible so as to minimize the potential damage that it can cause. Our current model does precisely that and we hope to further fine-tune it in the future to achieve higher efficiencies and also extend it to different social media platforms, such as Reddit, Facebook and Instagram as well.

CHAPTER- 11

REFERENCES/BIBLIOGRAPHY

[1] Zubiaga, Arkaitz & Liakata, Maria & Procter, Rob. (2017). Exploiting Context for Rumour Detection in Social Media. 10.1007/978-3-319-67217-5_8.

[2] Ma, Jing & Gao, Wei & Mitra, Prasenjit & Kwon, Sejeong & Jansen, Jim & Wong, Kam-Fai & Cha, Meeyoung. (2016). Detecting Rumors from Microblogs with Recurrent Neural Networks.

[3] Poddar, Lahari & Hsu, Wynne & Lee, Mong & Subramaniyam, Shruti. (2018). Predicting Stances in Twitter Conversations for Detecting Veracity of Rumors: A Neural Approach. 65-72. 10.1109/ICTAI.2018.00021.

[4] Tian, Lin & Zhang, Xiuzhen & Wang, Yan & Liu, Huan. (2020). Early Detection of Rumours on Twitter via Stance Transfer Learning. 10.1007/978-3-030-45439-5_38.

[5] Alsaeedi, Abdullah & Al-Sarem, Mohammed. (2020). Detecting Rumors on Social Media Based on a CNN Deep Learning Technique. Arabian Journal for Science and Engineering. 10.1007/s13369-020-04839-2.

[6] Ma, Jing & Gao, Wei & Wong, Kam-Fai. (2017). Detect Rumors in Microblog Posts Using Propagation Structure via Kernel Learning. 10.18653/v1/P17-1066.

[7] Yoshida, Zen & Aritsugi, Masayoshi. (2019). Rumor Detection in Twitter with Social Graph Structures: ICICT 2018, London. 10.1007/978-981-13-1165-9_54.

[8] Wang, Shihan & Terano, Takao. (2015). Detecting rumor patterns in streaming social media. 2709-2715. 10.1109/BigData.2015.7364071.

[9] Bian, Tian & Xiao, Xi & Xu, Tingyang & Zhao, Peilin & Huang, Wenbing & Rong, Yu & Huang, Junzhou. (2020). Rumor Detection on Social Media with Bi-Directional Graph Convolutional Networks.

[10] Chierichetti, Flavio & Lattanzi, Silvio & Panconesi, Alessandro. (2011). Rumor Spreading in Social Networks. Theor. Comput. Sci.. 412. 2602-2610. 10.1007/978-3-642-02930-1_31.

[11] Yang, X., Lyu, Y., Tian, T., Liu, Y., Liu, Y., & Zhang, X. (2020). Rumor Detection on Social Media with Graph Structured Adversarial Learning. *IJCAI*.

[12] Zubiaga, Arkaitz & Aker, Ahmet & Bontcheva, Kalina & Liakata, Maria & Procter, Rob. (2018). Detection and Resolution of Rumours in Social Media: A Survey. ACM Computing Surveys. 51. 10.1145/3161603.

[13] Ahsan, Mohammad & Kumari, Madhu & Sharma, T. (2019). Rumors detection, verification and controlling mechanisms in online social networks: A survey.

[14] Li, Quanzhi & Zhang, Qiong & Si, Luo & Liu, Yingchi. (2019). Rumor Detection on Social Media: Datasets, Methods and Opportunities.

[15] Altschuler, S. J., Wu, L.F. and Ingerman, D. (1998) Methods and apparatus for determining or inferring influential rumormongers from resource usage data

[16] https://paperswithcode.com/task/rumour-detection

[17] https://www.sciencedirect.com/science/article/pii/S187705092030747X

[18] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7274137/

APPENDIX A

DEFINITIONS, ACRONYMS AND ABBREVIATIONS

1. **Rumour:**
   Piece of unverified information. Has some element of ambiguity attached with it.

2. **Social Graph:**
   This consists of people/organisations as nodes, and the relationships between them as links between the nodes. This can be used to understand the behaviour of the individual and the entire network in question as well.

3. **Socio-linguistic:**
   The language spoken on social media platforms such as Twitter does not abide to the norms of the conventional English language- there is a limit on the number of characters, people use abbreviations and slang, and there might also be spelling errors. This type of language comes under the umbrella of socio-linguistic data

4. **Propagation Tree Kernel*:***
   Finds some pattern in the spreading/diffusion structures of rumours by noting how similar the trees made from the propagation structures are.

5. **RNN**: Recurrent Neural Network

6. **CNN**: Convolutional Neural Network

7. **LSTM**: Long Short-Term Memory

8. **CRF**: Conditional Random Fields

9. **SVM**: Support Vector Machine

10. **BERT**: Bidirectional Encoder Representations from Transformers

11. **TF-IDF**: Term Frequency- Inverse Document Frequency

12. **NLP**: Natural Language Processing