Detecting and Mitigating Rumors in Social Media

Mohammad Raihanul Islam

Dissertation submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Applications

Naren Ramakrishnan, Chair
B. Aditya Prakash
Chang-Tien Lu
Chandan Reddy
Huan Liu

May 18, 2020
Blacksburg, Virginia

Keywords: Rumor Veracity Detection, Rumor Mitigation, Social Network Analysis, Deep Learning
Copyright 2020, Mohammad Raihanul Islam
Detecting and Mitigating Rumors in Social Media

Mohammad Raihanul Islam

(ABSTRACT)

The penetration of social media today enables the rapid spread of breaking news and other developments to millions of people across the globe within hours. However, such pervasive use of social media by the general masses to receive and consume news is not without its attendant negative consequences as it also opens opportunities for nefarious elements to spread rumors or misinformation. A rumor generally refers to an interesting piece of information that is widely disseminated through a social network and whose credibility cannot be easily substantiated. A rumor can later turn out to be true or false or remain unverified. The spread of misinformation and fake news can lead to deleterious effects on users and society. The objective of the proposed research is to develop a range of machine learning methods that will effectively detect and characterize rumor veracity in social media. Since users are the primary protagonists on social media, analyzing the characteristics of information spread w.r.t. users can be effective for our purpose. For our first problem, we propose a method of computing user embeddings from underlying social networks. For our second problem, we propose a long short-term memory (LSTM) based model that can classify whether a story discussed in a thread can be categorized as a false, true, or unverified rumor. We demonstrate the utility of user features computed from the first problem to address the second problem. For our third problem, we propose a method that uses user profile information to detect rumor veracity. This method has the advantage of not requiring the underlying social network, which can be tedious to compute. For the last problem, we investigate a rumor mitigation technique that recommends fact-checking URLs to rumor debunkers, i.e., social network users who are very passionate about disseminating true news. Here, we incorporate the influence of other users on rumor debunkers in addition to their previous URL sharing history to recommend relevant fact-checking URLs.
Detecting and Mitigating Rumors in Social Media

Mohammad Raihanul Islam

(GENERAL AUDIENCE ABSTRACT)

A rumor is generally defined as an interesting piece of a story that cannot be authenticated easily. On social networks, a user can generally find an interesting piece of news or story and may share (retweet) it. A story that initially appears plausible can later turn out to be false or remain unverified. The propagation of false rumors on social networks has a deteriorating effect on user experience. Therefore, rumor veracity detection is important, and drawing interest in social network research. In this thesis, we develop various machine learning models that detect rumor veracity. For this purpose, we exploit different types of information regarding users, such as profile details and connectivity with other users etc. Moreover, we propose a rumor mitigation technique that recommends fact-checking URLs to social network users who are passionate about debunking rumors. Here, we leverage similar techniques used in e-commerce sites for recommending products to solve this problem.
## Contents

**List of Figures** viii

**List of Tables** xii

1 **Introduction** 1
   1.1 Background 1
   1.2 What Is a Rumor? 1
   1.3 Effect of False Rumors 2
   1.4 Motivation and Organization of the Thesis 3
   1.5 Problem 1: Generating User Features Suitable for Rumor Identification by Exploiting the Network Structure of Social Media 5
   1.6 Problem 2: Integrating Computed User Feature Into a Cascade Prediction Model to Identify the Veracity of Rumor Threads 5
   1.7 Problem 3: Leveraging User Profile Information Into the Cascade Prediction Model to Detect the Veracity of Rumors 6
   1.8 Problem 4: Leveraging Recommender System to Recommend Fact-checking URLs to Rumor Debunkers 6
   1.9 Related Work 7
      1.9.1 Network Embedding 7
      1.9.2 Variational Autoencoder 7
      1.9.3 Trust in Social Networks 7
      1.9.4 Sequential Recommendation 7
   1.10 Overall Structure of the Thesis 8

2 **Generating User Features Suitable for Rumor Identification by Exploiting the Network Structure of Social Media** 10
   2.1 Introduction 10
   2.2 Solution Approach 10
2.2.1 User Trust in Social Networks

2.2.2 Constructing the Trust Network

2.2.3 Feature Generation from Trust Networks

2.3 Problem Formulation

2.4 Distributed Representations of Signed Networks (SIGNet)

2.4.1 SIGNet for Undirected Networks

2.4.2 SIGNet for Directed Networks

2.4.3 Efficient Optimization through Targeted Node Sampling

2.4.4 Discussion

2.5 Experiments

2.5.1 Experimental Setup

2.5.2 Are Embeddings Interpretable?

2.5.3 Analyzing the Embedding Space

2.5.4 Edge Label Prediction

2.5.5 Node Label Prediction

2.5.6 Node Label Prediction with Partial Information

2.5.7 Scalability

2.5.8 Parameter Sensitivity

2.6 Discussion

3 Integrating Computed User Feature Into a Cascade Prediction Model to Identify the Veracity of Rumor Threads

3.1 Introduction

3.2 Problem Formulation

3.3 Introducing AdaLSTM

3.4 Experiments

3.4.1 Compared Methods

3.4.2 Experimental Settings

3.4.3 Does the TSM Algorithm Converge?
3.4.4 How Effective Is AdaLSTM in Classifying Rumors Versus State-of-the-Art Methods? ................................................................. 39
3.4.5 Case Study .................................................................................. 40
3.5 Discussion ..................................................................................... 42

4 Leveraging User Profile Information Into the Cascade Prediction Model to Detect the Veracity of Rumors 43
4.1 Introduction .................................................................................. 43
4.2 Proposed Model ............................................................................ 43
4.3 Experiments .................................................................................. 47
  4.3.1 Experimental Setup ................................................................. 47
  4.3.2 How Does RumorSleuth Perform Against Other Models in Detecting Rumor Veracity ................................................................. 48
  4.3.3 How Distinct Are the Latent User Vectors? ............................... 49
  4.3.4 How Early RumorSleuth Can Detect a Rumor? ........................... 49
4.4 Discussion ..................................................................................... 51

5 Leveraging Recommender System to Recommend Fact-checking URLs to Rumor Debunkers 52
5.1 Introduction .................................................................................. 52
5.2 Introducing NActSeer ................................................................. 52
  5.2.1 Modeling User History ............................................................. 53
  5.2.2 Aggregating Neighbor’s Preferences ......................................... 55
  5.2.3 Complete Architecture of NActSeer ......................................... 56
  5.2.4 Expediting User Embedding Computation ................................. 57
  5.2.5 Using Pre-trained Embedding .................................................. 58
5.3 Experiments .................................................................................. 59
  5.3.1 Experimental Setup ................................................................. 59
  5.3.2 How Effective Is NActSeer in Predicting the Next URL Share? .... 60
  5.3.3 How Useful Is the Expedited User Embedding Computation? ....... 61
List of Figures

1.1 Example of non-rumor, true, false and unverified rumor stories surrounding the Ottawa shooting incident. .................................................. 2

1.2 A rumor purporting that the president is injured led to a sharp dip of approx. $130 billion for the S&P 500.² ........................................ 3

1.3 Organization of the thesis .......................................................... 9

2.1 Comparison of avg. trustingness and trustworthiness of rumor supporters and rumor opposers. We can see rumor supporters have comparatively high trustingness and low trustworthiness indicating they are highly susceptible to rumors helping it to spread. Additionally low trustworthiness shows their shared content cannot be trusted readily. However we observe quite the opposite for true new supporters who posses a skeptic view on social media news (low trustingness) and are more reliable content sharer (high trustworthiness) 11

2.2 Given a signed network (a), a conventional network embedding (b) will not take signs into account and can result in faulty representations. (c) SIGNet learns embeddings respecting sign information between edges. ................................. 13

2.3 Signed triangles in an undirected graph. (a) and (b) are balanced but (c) and (d) are not. ................................................................. 14

2.4 (a) depicts a small network to illustrate why conventional negative sampling does not work. \(v_i\) and \(v_y\) might be considered too distant for their representations to be placed close to each other. Targeted node sampling solves this problem by constructing a cache of nodes which can be used as sampling. (b) shows how we resolve conflict. Although there are two ways to proceed from node \(v_i\) to \(v_l\) the shortest path is \(v_i, v_j, v_k, v_l\), which estimates a net positive relation between \(v_i\) and \(v_l\). As a result \(v_l\) will be added to \(\eta_i^+\). However for node \(v_m\) there are two shortest paths from \(v_i\), with the path \(v_i, v_p, v_o, v_n, v_m\) having more positive edges but with a net negative relation, so \(v_m\) will be added to \(\eta_i^-\) in case of a conflict. .................................................. 17

2.5 A comparative scenario depicting the optimization process inherent in both SiNE (a) and SIGNet (b). The shaded vertices represent the nodes both methods will consider while optimizing the edge \(e_{ij}\). We can see the SiNE only considers the immediate neighbors because it optimizes edges in 2-hop paths having opposite signs. On the other hand, SIGNet considers higher order neighbors \((v_a, v_b, v_c, v_x, v_y, v_z)\) for targeted node sampling. .......................... 19
2.6 Two dimensional embedding of alliances among sixteen tribes of New Guinea. Alliance between the tribes is shown in solid blue edges and a hostile relation is shown in dashed red edges. We can see that edges representing alliance are comparatively shorter than the edges represents hostility. 24

2.7 Predicting the polarity of adjectives in a subset of the ADJNet dataset. Here red labeled/boldface words are negative while the blue labeled/slanted words are positive. (Many adjectives have been removed to reduce clutter.) We use t-SNE to map the data into a 2D space. 28

2.8 Several conservatively and liberally disputed cases including Bipartisan Campaign Reform Act (BCRA) related cases that cite Citizens United vs. Federal Election Commission. Conservatively disputed cases are shown in red and liberally disputed cases are shown in blue. Our discussed cases are shown in filled circles while other cases are shown in unfilled circles. Dashed edges represents negatively connected and solid edges represents positively connected. 29

2.9 Performance evaluation on ADJNet and SCOTUS datasets varying the percent of nodes used for training. The $x$-axis shows the percent of nodes whose information is used to learn the embedding. $y$-axis shows the micro F1 score (top) and macro F1 score (bottom) for each dataset. SIGNet outperforms SIGNet-NS in all cases. 30

2.10 Scalability of SIGNet on Erdos-Renyi random graphs. (a) execution time of SIGNet varying the number of nodes; (b) execution time of SIGNet varying the number of threads. 31

2.11 Performance evaluation on ADJNet and SCOTUS dataset varying different parameters for node label prediction. Top two figures show macro F1 score varying total number of samples. Bottom figures show macro F1 score varying the dimension size of embedding $K$. 32

3.1 An overview of how rumor propagates in social media. An otherwise innocuous event or development (e.g., Russian President Putin not showing up in public for 10 days) causes people to speculate and initiate new lines of discussion. We exploit the underlying social structure and conversational thread to identify whether the story is true (non-rumor), a rumor, or unverified (best viewed in color). 34
3.2 A block architecture of AdaLSTM. First from the input thread sequence \(\langle u_1 \rangle, \ldots, \langle u_n \rangle\) a subsequence is extracted (here starting at position 6). Then the elements of the subsequence i.e. user and stance is converted into corresponding embedding. The embedding vectors along with the temporal features i.e. inter-event timing is fed to an LSTM model. The last hidden state vector of the LSTM is then given to the Location Definer Network (LDN) to get the next location of the subsequence to process. In this way a set of hidden state vectors are obtained which are passed to an attention layer to obtain a weighted state vector. This vector can be seen as the feature representation of the input cascade. Then a softmax layer is used to get a probability distribution of the rumor label (best viewed in color).

3.3 The summation of difference of trust scores for each node between two consecutive iterations. We can see that the difference is decreasing gradually demonstrating that the TSM algorithm is converging.

3.4 3D projection of user trust embedding discussing the event Charlie Hebdo Massacre. Good users are the ones who support a non-rumor story or contradict rumors whereas malicious users either deny a non-rumor story or support a rumor (best viewed in color).

4.1 (a) and (b) shows the CCDF of friends and status count of users who initiate true and false rumors. We can see that people who initiate a false rumor usually have lower status or friends count. (c) shows (%) of verified and non-verified accounts that start a rumor post. It is quite clear that unverified accounts are often responsible for starting a false rumor.

4.2 Model architecture of RumorSleuth. This model exactly resembles the AdaLSTM from problem 2 with one difference. Here we have a variational autoencoder to obtain the latent user representation of users and this representation is feed to the LSTM network.

4.3 2D projection of latent vector of the users who initiates a rumor. We can see that false and true rumor are quite separable while unverified rumors appear close to both types.

4.4 How early the models are able to classify a rumor in (a) PHEME and (b) RDTW dataset.

5.1 An example of rumor debunking post on Twitter.
5.2 The overall architecture of *NActSeer*. First, the timestamp of the entire share history of all the users is discretized into time spans. Then we apply log-bilinear model to obtain user embedding for each time span. The individual user action history along with the user embedding is then fed to the LSTM part of the *NActSeer* model, where user/debunker embedding is aggregated given the social network $G$ (shown in dashed line). At the end of the sequence the hidden state of the LSTM is passed through a softmax layer for next action prediction.

5.3 Performance comparison of *NActSeer* against other baseline models. *NActSeer* is able to outperform other model by a very good margin.

5.4 Comparison of execution time between expedited and normal user embedding computation for several generic datasets in the same problem setting. We can see that expedited user embedding computation significantly lowers the training time.

5.5 Normalized count of correct predictions vs number of neighbors/friends. We observe an increasing trend of performance as the number of friends increases showing that including neighbors actions improves performance. However, the trend has an diminishing return.
List of Tables

2.1 Statistics of the datasets used for performance evaluation. In social network datasets negative edges are underrepresented, however in ADJNet and SCOTUS they are well represented. ADJNet and SCOTUS also contain binary labels. ............................................................... 23

2.2 Average Euclidean distance between node representations connected by positive edges versus negative edges with std. deviation. We can see that the avg. distance between positive edge is significantly lower than negative edges indicating that SIGNet preserves the conditions of structural balance theory. ......................................................... 24

2.3 Comparison of edge label prediction in all four datasets. We show the micro F1 and AUC score for each feature scoring method. The best F1 score and AUC score across all the scoring method is shown in larger boldface. SIGNet outperforms node2vec and SiNE in every case. The results are statistically significant with $p$-value less than 0.01. .............................................................. 26

2.4 Comparison of methods for node label prediction on real world datasets. SIGNet outperforms SiNE and node2vec in all datasets. ................................................................. 27

2.5 Comparison of multiclass prediction on synthetic datasets using a one-vs-rest logistic regression classifier. SIGNet outperforms all the other methods in all datasets. ......................................................... 29

3.1 General information regarding the two datasets used for evaluation including length of the threads. Here F, T and U refers to False, True and Unverified rumor ............................................................... 37

3.2 Comparison of performance between AdaLSTM and other methods. Best performing method is shown in boldface while second best method is shown in italics. AdaLSTM outperforms other methods by a very good margin. . . 39

3.3 Two examples of subsequences picked by AdaLSTM. These subsequences were the most highly attended by AdaLSTM. We only show users whose support the thread. As we can see the trustworthiness score of rumor supporters are drastically lower than the non-rumor supporters. .................................................. 41

4.1 Comparison of performance between RumorSleuth, AdaLSTM and other methods. Proposed method is shown in boldface. ................................................................. 49
5.1 Important notations used in the paper

xiii
Chapter 1

Introduction

1.1 Background

Over the past decade, social media platforms have become an indispensable part of our lives. The widespread popularity of social media has turned the world into a true global village. Social media platforms like Twitter, Facebook, and Weibo allow users to rapidly disseminate information about breaking news events (e.g., mass protests, bombings, natural disaster, etc.) across the globe within hours. This ability to disseminate information rapidly also brings with it the capacity for misinformation and rumors to propagate through social media alongside true news events.

1.2 What Is a Rumor?

A rumor can be defined as a “widely circulated piece of information, which is easy to believe but hard to verify”. Rumors often create wide-spread confusion and mass panic. A rumor cascade emerges on a social media platform when a user makes a claim or asserts an opinion about a topic or a news-event. This claim or assertion may come in the form of writing or sharing a URL or a image \[69\]. Other users propagate this information by sharing/retweeting it. A rumor eventually can turn out to be true, false, or unverified. Any news regarding an event, which is not a rumor, can be considered as a non-rumor.

An example: We present an illustrative example of non-rumor, true rumor, false rumor, and unverified rumors in Fig. 1.1. All the stories in the examples revolve around the deadly shooting event at the Ottawa Parliament Hill in October 2014. Not all the posts regarding an event can be considered as rumor or fact-checking worthy. For example, if anyone writes a post like “Our thoughts and prayers are with the victims of Ottawa shooting incident”. This sentence is a non-factual post. As a result, this is not a rumor post. On the other hand, if anyone tweets like “Kevin Vickers is credited for taking down the shooter”, this sentence is fact-checking worthy and can be considered as a rumor. The veracity of this rumor is true. However, a tweet like “report of 3 separate shooting incidents” is also a rumor post and its veracity is false.
Figure 1.1: Example of non-rumor, true, false and unverified rumor stories surrounding the Ottawa shooting incident.

Terminology related to rumors: We present the definition of several rumor related terms.

Misinformation and Disinformation: Misinformation refers to the unintentional presentation of inaccurate or false information. On the other hand, disinformation means the spreading false news or reports to deliberately mislead people.

Hoax: Hoaxes contain false or inaccurate facts that are made to look like authentic news. The purpose of a hoax can be malicious or humorous. Hoaxes can spread in both social media and other internet mediums such as Wikipedia. For example, reports of false deaths of celebrities fall into this category.

Fake/False news: Usually fake/false news is published on the internet by some online news outlets. Then the news may be shared in social media thus starting rumor cascades. The intention of publishing fake news is to deliberately mislead people, whereas people may share them with benign intention (i.e. without knowing its veracity).

Urban Legends: Urban legends are a type of modern fictional story which is rooted in the local culture. Urban legends often have characters and elaborate plots with elements of mystery, humor, and horror.

1.3 Effect of False Rumors

False rumors have devastating consequences, especially in economic and social sectors. For example, on April 23, 2013, a tweet was posted from a hacked account claiming that
1.4 Motivation and Organization of the Thesis

The constant deluge of misinformation in social media has opened up a new research frontier, viz. rumor detection [18, 22, 32, 64, 69]. This is an especially challenging problem as it requires modeling, not just information content but also the temporal dynamics of how the information is propagated [42] and the users who act as the propagator [72].

In this thesis, our goal is to develop machine learning models that can leverage a range of information related to users to combat rumors. Specifically, we work on two facets of rumor research. The first is to analyze the propagators of content i.e. the users of social networks and their interaction with the content. Next, we will investigate the rumor mitigation techniques by fact-checking URL recommendations.

---

1. Obama Injured Rumor

Figure 1.2: A rumor purporting that the president is injured led to a sharp dip of approx. $130 billion for the S&P 500.2

explosion at the White House injured then-President Barack Obama1. This single fake tweet alone caused a devastating panic in worldwide financial markets, causing the S&P 500 index to (temporarily) lose about 130 billion dollars of market value within a small period (see Fig. 1.2). On a similar note, on December 5, 2015, an ISIS related photograph was posted on Facebook claiming that a pro-ISIS rally was held in Dearborn, MI, USA. The image was widely circulated across different social media platforms leading to counter-protests. Similar types of fake news and rumors circulate on social media at regular intervals.

---

1https://www.cnbc.com/id/100646197
Rumor Veracity Detection Perspective: One of the primary means of tackling rumor detection is by analyzing the users of a social network that act as the spreaders of contents. One way of achieving this is by analyzing the temporal diffusion pattern of content engagement in social media. Any content sharing can be viewed as a temporal diffusion, where we have the information of the user and possible other meta-information of the diffusion. Here, we need to capture the interaction between social media posts and the user himself. Therefore, we need to find a suitable user representation between the users. After computing the user representation, we can fuse the representation in a diffusion prediction model to identify the veracity of rumors. Now while computing the user representation we can leverage two types of information.

- **Representing users using structural information:** To represent users from the structural information we can transform the social network as a credibility network between users \[33\]. A credibility network is a signed network, where a positive relation between a pair of users signifies that they share similar viewpoints whereas, negative relation suggests conflicting viewpoints. To construct the credibility network we use two metrics called trustingness and trustworthiness \[62\]. We assign the trustingness and trustworthiness scores for each user and then compute the ‘believability’ score between them. Then we can normalize the believability score to convert the network into a signed network. Now we can compute the user features in an unsupervised manner using word2vec model.

- **Representing users by profile information:** The second way of representing the user is to use the profile information. Information like follower/friend count can indicate the status of the user in a network and quantify how he can be enticed by rumors and his propensity to share it. We can represent the profile information using an unsupervised model like Autoencoder.

Perspective from Mitigation Strategies: One of the most vital parts of rumor research is to analyze how it can be mitigated. Nowadays there exist several fact-checking websites that can provide investigative reports on possible rumor stories. There are a lot of ‘rumor debunkers’ active in social media platforms who avidly share fact-checking URLs to disseminate the veracity of rumors. Recommending them relevant URLs will expedite the debunking process. In this thesis, we propose a sequential URL recommendation for rumor debunkers, where we model the influence of rumor debunkers on one another.
1.5 Problem 1: Generating User Features Suitable for Rumor Identification by Exploiting the Network Structure of Social Media

Our first challenge is to develop a method to compute user features which should be pertinent to rumor veracity detection. Since users are the propagators of rumors, characterizing users can be fruitful in rumor veracity detection. For this purpose, we need a suitable user representation. However, generating user features from social networks is not trivial in our case. Since rumor veracity is highly correlated with user credibility, a trust-based network rather than the generic underlying social network would be more beneficial. As a result, we generate a suitable representation for users by taking user-user trust levels into account.

**Contribution:** For a trust-based network representation we convert the social network into a signed network. Then we propose a method called SIGNet that can compute node embeddings (i.e. user embedding) from the signed network. We demonstrate the efficacy of SIGNet on generic signed networks and published our work in [29].

1.6 Problem 2: Integrating Computed User Feature Into a Cascade Prediction Model to Identify the Veracity of Rumor Threads

Our second problem is to leverage the user embedding created by SIGNet to detect rumor veracity. Rumor propagation through a social network is modeled as a cascade phenomenon in this work. Therefore rumor veracity detection can be cast as a cascade sequence classification problem. We propose a cascade sequence classification approach using an LSTM model. We face two challenges while using an LSTM model in modeling cascade. First, the LSTM model does not provide a good mechanism to revisit an earlier portion of cascade if necessary and cascade sequence can be inherently long making the LSTM model computationally expensive.

**Contribution:** We propose an enhanced version of LSTM to tackle both shortcomings. Our proposed LSTM model *AdaLSTM* can analyze a small portion of the thread at a time thus enabling it to reduce computational overhead. Additionally, it can also revisit an earlier part of the thread if required providing more flexibility. We have experimented on generic cascades and published a paper [28]. We also have submitted an extended version to a journal, which is currently in review.
1.7 Problem 3: Leveraging User Profile Information Into the Cascade Prediction Model to Detect the Veracity of Rumors

In the third problem, we detect rumor veracity using the user profile information. We find that structural information in social media (e.g., Twitter) is hard to obtain. As a result, generating user features in this manner is very time-consuming. Therefore we leverage other types of user representation to fill this gap. We perform exploratory analysis on the user profile information and find that they are correlated with the rumor veracity. Therefore we exploit user profile information (e.g., follower count, friends count) to identify rumor veracity.

**Contribution:** We combine deep unsupervised feature learning methods, e.g., autoencoders along with an LSTM based model to develop a machine learning framework called RumorSleuth to classify rumor veracity. Specifically, we feed the learned user representation to the LSTM framework we develop in Problem 2. We compare the results with other rumor detection methods from problem 2 and also with the AdaLSTM model. We have published the extension of our work in [31].

1.8 Problem 4: Leveraging Recommender System to Recommend Fact-checking URLs to Rumor Debunkers

In the last problem, we try to improve one of the mitigation strategies of the rumor debunking process. One effective way to mitigate rumors is to spread of correct information to the social media space. Avid social network users are known as rumor debunkers regularly share fact-checking URLs to circulate the veracity of rumors. Our objective is to recommend relevant fact-checking URLs to the rumor-debunker community so that they can share them and take an effective part to combat rumors. For this purpose, we propose a sequential recommendation framework that can take the internal connections among the debunkers to recommend important URLs to them.

**Contribution:** We coupled the sequential modeling from an LSTM network and the aggregation techniques from the graph neural network to develop our model NActSeer. We represent the users as a function of URL representation so that we do not have to rely on user-specific features. This is very convenient, especially for new users/debunkers. We also leverage the headline of URL to generate pre-trained embedding for URLs. We compare the model with recently proposed sequential recommendation models and our proposed approach is able to outperform them. The work is published in [30].
1.9 Related Work

1.9.1 Network Embedding

Over the past few years, several deep learning based models have been proposed to generate embeddings in an unsupervised way. Inspired by algorithms for unsupervised learning in text [43, 52] several node embedding methods have been proposed in [20, 57, 66]. This approach has been extended to different variations of networks such as heterogeneous networks [14] and signed networks [71]. Recently deep autoencoder models have also been utilized to compute node embeddings [38, 70]. Besides learning representations for nodes, the embedding of other elements of graphs has also been attempted e.g., subgraph [55], edge [4] or node, and edge together [49]. Recently CNN has been generalized to represent graph structure to compute features for nodes [6, 13, 39]. This type of model is first introduced in [6], which is improved in [13] using fast localized convolution filters. The second line of work tries to represent the entire graph or a portion of it as a fixed-length vector [44, 56].

1.9.2 Variational Autoencoder

Autoencoders learn a compact representation of the data in an unsupervised manner. The learned representation can be used for any downstream prediction tasks such as text classification [73]. The variational autoencoder [37] is a recent addition to autoencoder models and has seen a flurry of interest in many applications, e.g., vision [19], NLP [5], and collaborative filtering [47]. It is also used with other deep learning models like CNN [74], RNN [10] to enhance their capabilities.

1.9.3 Trust in Social Networks

Nowadays trust has become an integral part of various social interactions. Although often interpreted as a subjective concept, researchers have defined various types of trust scores for nodes in a network to accomplish different tasks [34, 35, 53]. EigenTrust [35] assigns a unique trust score to each peer in a peer-to-peer network. In [34] a signed random walk with restart is proposed to rank users in a signed network. In [53] an iterative algorithm is used to compute the trust score by estimating the bias and prestige.

1.9.4 Sequential Recommendation

Sequential recommender systems seek to model item-item transitions to capture the sequential patterns between consecutive items [3, 36, 67]. FPMC [61] is a first-order markov chain
model with a matrix factorization term along with an item-item transition matrix. A higher-order chain can also be incorporated to capture a deeper user history, however, it is often not necessary as discussed in [24]. Recently deep learning based-models are also used to capture the sequential nature of user preferences [3, 26, 36, 67]. For instance, GRU4Rec [26] uses GRU for session-based recommendation whereas Caser [67], a CNN based model, treats the embedding matrix of previous items as an image and applies the convolutional operation to capture the sequence. Finally, SASRec [36] uses the self-attention mechanism to predict the next item from a user’s history.

1.10 Overall Structure of the Thesis

The overall organization of this document is shown in Fig. 1.3. First, in Chapter 2 we present a model to generate user features in social networks. We leverage the recently proposed word2vec method to create the backbone of our presented method SIGNet. Next, we apply the user embedding method in a rumor veracity detection model described in Chapter 3. We cast the problem as cascade classification and propose a novel long short-term memory (LSTM) based method called AdaLSTM to classify the veracity of rumors. Here we argue that it is not feasible to use social network information in all circumstances. Therefore we propose a method in Chapter 4 that uses user profile information in the same LSTM framework we develop earlier. We call our proposed method RumorSleuth. In our last problem, (Chapter 5) we investigate our rumor mitigation strategy. We propose a sequential URL recommendation model called NActSeer for rumor debunkers, where the model explicitly incorporates the influence of rumor debunkers on each other while making URL recommendations.
1.10. Overall Structure of the Thesis

Figure 1.3: Organization of the thesis
Chapter 2

Generating User Features Suitable for Rumor Identification by Exploiting the Network Structure of Social Media

2.1 Introduction

In this chapter, we develop a method to generate user features from a social network to detect rumor veracity. We leverage recent success in word and document embedding to explore similar representations for networks. As rumor propagation is highly correlated with the credibility of users, the method should be able to generate embeddings in a signed network, where positive relations can be considered as a trust relationship and negative relations can be interpreted as a distrust relation. Existing methods are largely focused on finding distributed representations for unsigned networks and are unable to discover embeddings that respect polarities inherent in edges. We propose SIGNet, a fast scalable embedding method suitable for signed networks. Our proposed objective function aims to carefully model the social structure implicit in signed networks by reinforcing the principles of social balance theory. Our method builds upon the traditional word2vec family of embedding approaches but we propose a new targeted node sampling strategy to maintain structural balance in higher-order neighborhoods. Before applying SIGNet in the rumor veracity detection scenario, we compare its performance on generic signed networks to evaluate its efficacy.

2.2 Solution Approach

2.2.1 User Trust in Social Networks

One of the more natural ways of detecting rumors is to identify users who can be susceptible to rumors when exposed to a dramatic event e.g., a shooting or mass protest. To measure this susceptibility we need to quantify the trust-level of each user. Trust plays an important part in any type of social interaction. Trust is an abstract concept and the level of trust
2.2. Solution Approach

Figure 2.1: Comparison of avg. trustingness and trustworthiness of rumor supporters and rumor opposers. We can see rumor supporters have comparatively high trustingness and low trustworthiness indicating they are highly susceptible to rumors helping it to spread. Additionally low trustworthiness shows their shared content cannot be trusted readily. However we observe quite the opposite for true new supporters who posses a skeptic view on social media news (low trustingness) and are more reliable content sharer (high trustworthiness) towards a user can be subjective and vary from user to user. Generally in trust research, a score is assigned to each user representing his/her level of trust. We, in our analysis, express the level of trust using two different scores, one manifesting how ‘trustworthy’ the user is and the other termed ‘trustingness’ demonstrating how susceptible (or inclined) the user is to trusting other users [62].

To show how trustingness and trustworthiness plays their role in rumor spreading we show the avg. trustingness and trustworthiness of users who support true stories and user who support rumor news in Fig 2.1 for PHEME [76] dataset, a dataset we use in our experiment. We can see on average users with high trustingness supports rumor stories, on the other hand, high trustworthy users support true news. We hypothesize users with high trustingness tend to trust other people easily and sometimes intentionally (not always) spread rumor in social media whereas people with high trustworthiness are more reliable content sharer. Given this observation we intend to compute a user representation which is able to preserve this type of intricate user characteristics. Then we explore different classification approach on the user representation to identify rumors.

To compute the trust level, we model interactions among users that can be leveraged to quantify trust. The type of interactions can be retweet or resharing, mentions or following. For example if a user has a lot of followers then it manifests that he/she is viewed as being trustworthy (of course this has exceptions but we anticipate that simultaneous resolution of constraints will address these cases). Similarly if a user retweets a lot of other users this shows that she is open to trusting other people easily, thus having a higher trustingness score. In our model we use the follower-follownee network between users to compute the trust
Chapter 2. Generating User Features Suitable for Rumor Identification by Exploiting the Network Structure of Social Media

2.2.2 Constructing the Trust Network

To calculate the trustingness and trustworthiness among users we apply the ‘trust scores in social media’ (TSM) algorithm \[62\] over the follower-followee network. TSM takes a directed graph as input and iteratively updates the trustingness and trustworthiness score for each node \(v \in V\) using the following equations:

\[
\text{trustingness}(s) = \sum_{t \in \text{out}(s)} \frac{w_{st}}{1 + \text{trustworthiness}(t)} \tag{2.1a}
\]

\[
\text{trustworthiness}(s) = \sum_{t \in \text{in}(s)} \frac{w_{ts}}{1 + \text{trustingness}(t)} \tag{2.1b}
\]

Here \(w_{st}\) represents the weight between node \(v_s\) and \(v_t\). \(\text{out}(s)\) and \(\text{in}(s)\) represent the outgoing and incoming edges of node \(v_s\) respectively. Both trustingness and trustworthiness are an aggregated measure of a user’s tendency to put trust on other users and others’ willingness to trust him/her. However, they do not gauge the trust relationship between two users. Quantifying the trust level between a pair of users (i.e. an edge) is important since this enables us to measure the relationship between users in a granular way. The follower-followee network does not characterize the level of trust between users. Therefore to quantify the trust level between two users (i.e. how user \(A\) trusts user \(B\)) we multiply the user \(A\)’s trustingness with user \(B\)’s trustworthiness. In \[58\] this term is coined as ‘believability’ \((B)\).

\[
B_{A \rightarrow B} = \text{trustingness}(A) \times \text{trustworthiness}(B) \tag{2.2}
\]

A high believability score for edge \(e_{AB}\) means user \(A\) trusts user \(B\) very much (although the converse may not be true). This concludes our description of constructing the trust network from user interaction network. Next we discuss how we learn user embeddings from trust networks.

2.2.3 Feature Generation from Trust Networks

To compute the user trust embedding from the trust network we can apply any node embedding strategy \[20, 57\]. Node embedding aims to find representations in a suitable multi-dimensional space that places similar users closer to each other. Similar to word embedding methods like word2vec \[52\], a sequence of nodes is generated using a random walk, and the similarity between walks is captured in the representation. However there is a shortcoming with this approach when applied to trust networks.

In generic node embedding methods the distance between two users indicates the proximity between them in terms of node connectivity and need not correlate with their trust level.
A weighted random walk based on trust scores may provide a partial solution; however it is still not foolproof.

Therefore to introduce the notion of trust (or distrust), we convert the trust network into a signed network. To accomplish this objective, we subtract the median believability value from each edge weight of the trust network. This way the trust network becomes a signed network, where negative values denote distrust. However, now generic node embedding methods cannot be applied as they are not geared toward signed networks.

### 2.3 Problem Formulation

**Definition 2.1. Signed Network:** A signed network can be defined as $G = (V, E)$, where $V$ is the set of vertices and $E$ is the set of edges between the vertices. Each element $v_i$ of $V$ represents an entity in the network and each edge $e_{ij} \in E$ is a tuple $(v_i, v_j)$ associated with a weight $w_{ij} \in \mathbb{Z}$. The absolute value of $w_{ij}$ represents the strength of the relationship between $v_i$ and $v_j$, whereas the sign represents the nature of relationship (e.g., friendship or antagonism). A signed network can be either directed or undirected. If $G$ is undirected then the order of vertices is not relevant (i.e. $(v_i, v_j) \equiv (v_j, v_i)$). On the other hand if $G$ is directed then order becomes relevant (i.e. $(v_i, v_j) \neq (v_j, v_i)$ and $w_{ij} \neq w_{ji}$).

![Figure 2.2](image.png)

Figure 2.2: Given a signed network (a), a conventional network embedding (b) will not take signs into account and can result in faulty representations. (c) SIGNet learns embeddings respecting sign information between edges.

Because the weights in a signed network carry a combined interpretation (sign denotes polarity and magnitude denotes strength), conventional proximity assumptions used in unsigned
network representations (e.g., in [20, 66]) cannot be applied for signed networks. Consider a network wherein the nodes $v_i$ and $v_j$ are positively connected and the nodes $v_k$ and $v_i$ are negatively connected (see Fig. 2.2(a)). Suppose the weights of the edges $e_{ij}$ and $e_{ik}$ are $+w_{ij}$ and $-w_{ik}$ respectively. Now if $|+w_{ij}| < |-w_{ik}|$, conventional embedding methods will place $v_i$ and $v_k$ closer than $v_i$ and $v_j$ owing to the stronger influence of the weight (Fig. 2.2(b)). (Ignoring weights does not solve this problem.) Ideally, we would like a representation wherein nodes $v_i$ and $v_j$ are closer than nodes $v_i$ and $v_k$, as shown in Fig. 2.2(c). This example shows that modeling the polarity of the relationship is as important as modeling the strength of the relationship.

Figure 2.3: Signed triangles in an undirected graph. (a) and (b) are balanced but (c) and (d) are not.

To accurately model the interplay between the vertices in signed networks we use the theory of structural balance proposed by Heider [25]. Structural balance theory posits that triangles with an odd number of positive edges are more plausible than an even positive edges (see Fig. 2.3). Cartwright and Harary [7] proposed a theorem about global network structure interpreted via balance theory. This theorem asserts that in an undirected network where each triangle is connected and obeys the structural balance theory, the vertices can be partitioned into two disjoint subsets where intra-subset edges are positive and inter-subset edges are negative:

**Theorem 2.2.** (Cartwright and Harary, 1956): A signed network is balanced if and only if the vertices can be divided into two mutually exclusive subsets so that each positive edge connects nodes within the subset and each negative edge connects nodes between the subsets.

Although structural balance theory has been adapted and refined significantly (e.g., by Davis [12]), here we primarily focus on the original notion of structural balance. Theorem 2.2 suggests that in a perfectly balanced network we can observe two groups, where members within groups are friendly to each other, and members across the groups are not. Inspired by this observation, we aim to obtain embeddings for signed networks that reflect such a dichotomy. It should be noted that although there are other theories (e.g., social status theory [21]) for signed networks, this paper focuses primarily on structural balance theory as a foundation to create the embedding space. The incorporation of other theories is a direction of future work.
Problem Statement: Distributed Representations of Signed Networks (SIGNet): Given a signed network $G$, compute a low-dimensional vector $d_i \in \mathbb{R}^K$, $\forall v_i \in V$, where positively related vertices reside in close proximity and negatively related vertices are distant.

2.4 Distributed Representations of Signed Networks (SIGNet)

2.4.1 SIGNet for Undirected Networks

Consider a weighted signed network defined as in Section 2.3. Now suppose each $v_i$ is represented by a vector $x_i \in \mathbb{R}^K$. Then a natural way to compute the proximity between $v_i$ and $v_j$ is by the following function (ignoring the sign for now):

$$p_u(v_i, v_j) = \sigma(x_j^T \cdot x_i) = \frac{1}{1 + \exp(-x_j^T \cdot x_i)} \quad (2.3)$$

where $\sigma(a) = \frac{1}{1 + \exp(-a)}$. Now let us breakdown the weight of edge $w_{ij}$ into two components: $r_{ij}$ and $s_{ij}$. $r_{ij} \in \mathbb{N}$ represents the absolute value of $w_{ij}$ (i.e. $r_{ij} = |w_{ij}|$) and $s_{ij} \in \{-1, 1\}$ represents the sign of $w_{ij}$. Given this breakdown of $w_{ij}$ and incorporating the weight information, the objective function for undirected signed network can be written as:

$$O_{un} = \sum_{e_{ij} \in E} r_{ij} \times \sigma(s_{ij}(x_j^T \cdot x_i)) \quad (2.4)$$

By maximizing Eqn. 2.4 we obtain a vector $x_i$ of dimension $K$ for each node $v_i \in V$ (we will also use $d_i$ to denote this embedding, for reasons that will become clear in the next section).

2.4.2 SIGNet for Directed Networks

Computing embeddings for directed networks is trickier due to the asymmetric nature of neighborhoods (and thus, contexts). For instance, if the edge $e_{ij}$ is positive, but $e_{ji}$ is negative, it is not clear if the respective representations for nodes $v_i$ and $v_j$ should be proximal or not. We solve this problem by treating each vertex as each vertex as itself plus a specific context; for instance, a positive edge $e_{ij}$ is interpreted to mean that given the context of node $v_j$, node $v_i$ wants to be closer. This enables us to treat all nodes consistently without worrying about reciprocity relationships. To this end, we introduce another vector $y_i, \forall v_i \in V$ in addition to $x_i$. For a directed edge $e_{ij}$ the probability of context $v_j$ given $v_i$ is:

$$p_d(v_j|v_i) = \frac{\exp(s_{ij}(y_j^T \cdot x_i))}{\sum_{k=1}^{\{|V|\}} \exp(s_{ik}(y_k^T \cdot x_i))} \quad (2.5)$$
Chapter 2. Generating User Features Suitable for Rumor Identification by Exploiting the Network Structure of Social Media

Treating the same entity as itself and as a specific context is very popular in the text representation literature [52]. The above equation defines a probability distribution over all context space w.r.t. node \( v_i \). Now our goal is to optimize the above objective function for all the edges in the network. However we also need to consider the weight of each edge in the optimization. Incorporating the absolute weight of each edge we obtain the objective function for a directed network as:

\[
O_{\text{dir}} = \sum_{e_{ij} \in E} r_{ij} p_d(v_j | v_i) \tag{2.6}
\]

By maximizing the above function we will obtain two vectors \( x_i \) and \( y_i \) for each \( v_i \in V \). The vector \( x_i \) models the outward connection of a node whereas \( y_i \) models the inward connection of the node. Therefore the concatenation of \( x_i \) and \( y_i \) represents the final embedding for each node. We denote the final embedding of node \( v_i \) as \( d_i \). It should be noted that for undirected network \( d_i = x_i \) whereas for a directed network \( d_i \) is the concatenation of \( x_i \) and \( y_i \). This means \(|x_i| = |y_i| = K^2\) in the case of directed graph (for the same representational length).

### 2.4.3 Efficient Optimization through Targeted Node Sampling

The denominator of Eqn. 2.5 is very hard to compute as we have to marginalize the conditional probability over the entire vertex set \( V \). We adopt the classical negative sampling approach [52] wherein negative examples are selected randomly from some distribution for each edge \( e_{ij} \). Incorporating the negative sampling we obtain the following objective function for each edge \( e_{ij} \):

\[
\log[\sigma(s_{ij}(y_j^T \cdot x_i))] + \sum_{c=1}^{N} E_{v_n \sim P_n(v)} \log[\sigma(-(y_n^T \cdot x_i))] \tag{2.7}
\]

Here \( N \) is the number of negative examples per edge. However, for signed network conventional negative sampling does not work. For example consider the network from Fig. 2.4(a). Viewing this example as an unsigned network, while optimizing for edge \( e_{ij} \), we will consider \( v_i \) and \( v_y \) as negative examples and thus they will be placed distantly from each other. However, in a signed network context, \( v_i \) and \( v_y \) have a friendlier relationship (than with, say, \( v_x \)) and thus should be placed closer to each other. We propose a new sampling approach, referred to as simply targeted node sampling wherein we first create a cache of nodes for each node with their estimated relationship according to structural balance theory and then sample nodes accordingly.

**Constructing the cache for each node**

We aim to construct a cache of positive and negative examples for each node \( v_i \) where the positive example cache \( \eta_i^+ \) contains nodes which should have a positive relationship with \( v_i \).
We denote this phenomena as $\sim$ if $s$ is not a neighbor of node $v$.\textcolor{red}{The base case for this formula is recursively. (In the rare instances where the sign (i.e. weight) between two nodes is zero, we consider them as negative).}\textcolor{red}{The one problem with this approach is that a node $v$ might be considered too distant for their representations to be placed close to each other. Targeted node sampling solves this problem by constructing a cache of nodes which can be used as sampling. (b) shows how we resolve conflict. Although there are two ways to proceed from node $v_i$ to $v_l$ the shortest path is $v_i, v_j, v_k, v_l$, which estimates a net positive relation between $v_i$ and $v_l$. As a result $v_l$ will be added to $\eta^+_i$. However for node $v_m$ there are two shortest paths from $v_i$, with the path $v_i, v_p, v_n, v_m$ having more positive edges but with a net negative relation, so $v_m$ will be added to $\eta^-_i$ in case of a conflict.}

and the negative example cache $\eta^-_i$ contains nodes which should have negative relationship with $v$, according to structural balance theory. To construct these caches for each node $v_i$, we apply random walks of length $l$ starting with $v_i$ to obtain a sequence of nodes. Suppose the sequence is $\Omega = <\alpha_i, v_{n_0}, \ldots, v_{n_{l-1}} >$. Now we add each node $v_{n_p}$ to either $\eta^+_i$ or $\eta^-_i$ by observing the estimated sign between $v_i$ and $v_{n_p}$. The estimated sign is computed using the following recursive formula:

$$\tilde{s}_{in_p} = \tilde{s}_{in_{p-1}} \times s_{n_{p-1}n_p}$$  \hspace{1cm} (2.8)

Here $\tilde{s}_{in_{p-1}}$ is the estimated sign between node $v_i$ and node $v_{n_{p-1}}$, which can be computed recursively (In the rare instances where the sign (i.e. weight) between two nodes is zero, we consider them as negative). The base case for this formula is $\tilde{s}_{in_1} = s_{in_0} \times s_{n_0n_1}$. If node $v_{n_p}$ is not a neighbor of node $v_i$ and $\tilde{s}_{in_p}$ is positive then we add $v_{n_p}$ to $\eta^+_i$. On the other hand if $\tilde{s}_{in_p}$ is negative and $v_{n_p}$ is not a neighbor of $v_i$ then we add it to $\eta^-_i$. For example for the graph shown in Fig. 2.4(a), suppose a random walk starting with node $v_i$ is $<v_i, v_j, v_k, v_z>$. Here node $v_k$ will be added to $\eta^+_i$ because $\tilde{s}_{ik} = s_{ij} \times s_{jk} > 0$ (base case) and $v_k$ is not a neighbor of $v_i$. On the other hand, $v_z$ will be added to node $\eta^-_i$ since $\tilde{s}_{iz} = \tilde{s}_{ik} \times s_{kz} <= 0$ and $v_z$ is not a neighbor of $v_i$.

The one problem with this approach is that a node $v_j$ may be added to both $\eta^+_i$ and $\eta^-_i$. We denote this phenomena as conflict and define the reason for this conflict in Theorem 2.3. We resolve this situation by computing the shortest path between $v_i$ and $v_j$ and compute $s_{ij}$ between them using the shortest path, then add to either $\eta^+_i$ or $\eta^-_i$ based on $s_{ij}$. To compute the shortest path we have to consider the network as unsigned since negative weight has a

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig2.4.png}
\caption{(a) depicts a small network to illustrate why conventional negative sampling does not work. $v_i$ and $v_y$ might be considered too distant for their representations to be placed close to each other. Targeted node sampling solves this problem by constructing a cache of nodes which can be used as sampling. (b) shows how we resolve conflict. Although there are two ways to proceed from node $v_i$ to $v_l$ the shortest path is $v_i, v_j, v_k, v_l$, which estimates a net positive relation between $v_i$ and $v_l$. As a result $v_l$ will be added to $\eta^+_i$. However for node $v_m$ there are two shortest paths from $v_i$, with the path $v_i, v_p, v_n, v_m$ having more positive edges but with a net negative relation, so $v_m$ will be added to $\eta^-_i$ in case of a conflict.}
\end{figure}
different interpretation for shortest path algorithms. In case of multiple shortest path we 
pick the one with highest number positive edges. A scenario is shown in Fig. 2.4(b).

**Theorem 2.3.** Node \( v_j \) will be added to both \( \eta_i^+ \) and \( \eta_i^- \) (conflict) if there are multiple paths from \( v_i \) to \( v_j \) and the union of these paths has at least one unbalanced cycle.

*Proof.* (By contradiction.) Suppose there is a conflict for node \( v_i \) where \( \eta_i^+ \) and \( \eta_i^- \) both contain node \( v_j \). Since there are at least two distinct \( v_i-v_j \) paths because of the conflict, the network contains a cycle \( c \) (ignoring the direction for directed networks). Now it is evident that the common edges of both paths are not responsible for the conflict since they occur in both paths. Now if cycle \( c \) is balanced there will be an even number of negative edges which will be distributed between the distinct \( v_x-v_y \) paths in \( c \). The distribution can occur in two ways: either both paths will have an odd number of negative edges or an even number of negative edges. In both cases the estimated sign between the \( v_x-v_y \) paths will be the same. However, this is a contradiction because the final estimated sign of two \( v_i-v_j \) paths are different and the signs between the common path are same, so the signs between the \( v_x-v_y \) paths must be different. Therefore, cycle \( c \) cannot be balanced and hence contains an odd number of negative edges. Thus we have identified at least one unbalanced cycle. \( \square \)

**Targeted edge sampling during optimization**

Now after constructing the cache \( \eta_i = \eta_i^+ \cup \eta_i^- \) for each node \( v_i \), we can apply the targeted sampling approach for each node. Here our goal is to extend the objective of negative sampling from classical word2vec approaches [52]. In traditional negative sampling, a random word-context pair is negatively sampled for each observed word-context pair. In a signed network both positive and negative edges are present, and thus we aim to conduct both types of sampling while sampling an edge observing its sign. Therefore when sampling a positive edge \( e_{ij} \), we aim to sample multiple negative nodes from \( \eta_i^- \) and while sampling a negative edge \( e_{ik} \) we aim to sample multiple positive nodes from \( \eta_i^+ \). Therefore the objective function for each edge becomes:

\[
O_{ij} = \log[\sigma(s_{ij}(y_j^T \cdot x_i))] + \sum_{c=1}^N E_{v_n \sim \tau(s_{ij})} \log[\sigma(s_{in}(y_n^T \cdot x_i))] \tag{2.9}
\]

Here \( \tau \) is a function which selects from \( \eta_i^+ \) or \( \eta_i^- \) based on the sign \( s_{ij} \). \( \tau \) selects from \( \eta_i^+ \) if \( s_{ij} \leq 0 \) or returns \( \eta_i^- \) if \( s_{ij} > 0 \).

The benefit of targeted node sampling in terms of global balance considerations across the entire network is shown in Fig. 2.5. Here we compare how our proposed approach SIGNet and SiNE [71] maintain structural balance. For simplicity suppose only edge \( e_{ij} \) has a negative sign. Now SiNE only optimizes w.r.t. pairs of edges in 2-hop paths each having different signs. Therefore optimizing the edge \( e_{ij} \) involves only the immediate neighbors of node \( v_i \) and
2.4. Distributed Representations of Signed Networks (SIGNet)

Figure 2.5: A comparative scenario depicting the optimization process inherent in both SiNE (a) and SIGNet (b). The shaded vertices represent the nodes both methods will consider while optimizing the edge $e_{ij}$. We can see the SiNE only considers the immediate neighbors because it optimizes edges in 2-hop paths having opposite signs. On the other hand, SIGNet considers higher order neighbors ($v_a, v_b, v_c, v_x, v_y, v_z$) for targeted node sampling.

$v_j$, i.e. $v_l, v_m, v_n, v_o$ (Fig. 2.5 (a)). However SIGNet skips the immediate neighbors while it uses higher order neighbors (i.e., $v_a, v_b, v_c, v_x, v_y, v_z$). Note that SIGNet actually uses immediate neighbors as separate examples (i.e. edge $e_{il}, e_{im}$ etc.). In this manner SIGNet covers more nodes to optimize the embedding space than SiNE.

2.4.4 Discussion

We now discuss several computational aspects of the SIGNet model.

**Optimization:** We adopt the asynchronous stochastic gradient method (ASGD) [60] to optimize the objective function $O_{ij}$ for each edge $e_{ij}$. The ASGD method randomly selects a mini batch of randomly selected edges and update emebeddings at each step. Now for each edge $e_{ij}$ the gradient of the objective function will have a constant coefficient $r_{ij}$ (i.e. $|w_{ij}|$). Now if the absolute weights of the edges have a high variance, it is hard to find a good learning rate. For example if we set the learning rate very small it would work well for large weighted edge but for small weighted edge the overall learning will be very inadequate resulting in poor performance. On the other hand, a large learning rate will work well for edges with smaller weights but for edges with large weight the gradient will be out of limits. To remedy this we adopt the *edge sampling* used in [66]. In edge sampling all the weighted edges treated as binary edges with non-negative weights (i.e. absolute value of edges $r_{ij}$). Now the edges are sampled during optimization according to the multinomial distribution constructed from the absolute value of the edge weights. For example suppose
Chapter 2. Generating User Features Suitable for Rumor Identification by Exploiting the Network Structure of Social Media

Algorithm 1 The SIGNet algorithm

Require: (Graph $G = (V,E)$, embedding size $K$, walks per node $r$, walk length $l$, total number of samples $s$, initial learning rate $\gamma$)

Ensure: $d_k \in \mathbb{R}^K, \forall v_k \in V$

1: for all $v_n \in V$ do
2:     for $i = 1$ to $r$ do
3:         $\omega_{ni} = \text{RandomWalk}(G, v_n, l)$
4:     for all $v_n \in V$ do
5:         for $i = 1$ to $r$ do
6:             for each $v_k \in \omega_{ni}$ do
7:                 Estimate relation between $v_k$ and $v_n$ using Eqn. 2.8
8:             Add $v_k$ to either $\eta^+_n$ or $\eta^-_n$ based on the relation
9:         resolve conflict for node $v_n$
10:    repeat
11:     for each mini-batch of edges do
12:         Sample an edge using edge sampling method
13:         Optimize the objective function in Eqn. 2.9.
14:     Update learning rate $\gamma$
15: until in total $s$ samples are processed

all the absolute values of the edges are stored in the set $R = \{r_1, r_2, \cdots, r_{|E|}\}$. Now during the optimization each edge is sampled according to the multinomial distribution constructed from $R$. However, each sampling from $R$ would take $O(E)$ time, which is computationally expensive for large network. To remedy this we use the alias table approach proposed in [46]. An alias table takes $O(1)$ time while continuously drawing samples from a constant discrete multinomial distribution.

Threshold value for $\eta_i$: Theoretically there should not be any bound on the size of $\eta^+_i$ and $\eta^-_i$. However empirical analysis shows limiting the size of $\eta^+_i$ and $\eta^-_i$ to very small values (i.e 5 – 7) actually gives better results.

$\eta_i$ for low degree nodes: Nodes with a low degree may not have an adequate number of samples for $\eta^+_i$ and $\eta^-_i$ from the random walks. This is why it is possible to exchange the nodes within $\eta^+_i$ and $\eta^-_i$. For example if node $v_x \in \eta^+_i$, one can add node $v_i$ to $\eta^+_i$. The same approach can be explored for $v_i$ and $v_y \in \eta^-_i$ in case there is an inadequate number of samples. However, we advise caution for taking this approach since it may violate structural balance for directed network.

Embedding for new vertices: SIGNet can learn embedding for newly arriving vertices. Since this is a network model, we can assume that advent of new vertices means we know its connection with existing nodes (i.e., neighbors). Suppose the new vertex is $v_n$ and its set of neighbors is $N_n$. We just have to construct $\eta_n$ and optimize the newly formed edges using the same optimization function stated in Eqn. 2.9 to obtain the embedding of node $n$. 

2.5. Experiments

Complexity: Constructing $\eta_i$ for node $v_i$ takes $O(rl)$ time where $l$ is the length of random walk and $r$ is the number of walk for each node. Since $rl \ll |V|$, the total cache construction actually takes very little time w.r.t. vertex size. Moreover conflict resolution only takes place for very rare instances where the length of the shortest path is at most $l$. This cost is thus negligible compared to random walk and cache construction time. Now, for optimizing each edge along with the node sampling take $O(K(N + 1))$, where $K$ is the size of embedding space and $N$ is the size of node sampling. The total complexity of optimization then become $O(K(N + 1)|E|)$, where $E$ is the set of edges. Therefore the overall complexity becomes $O(rl|V| + K(N + 1)|E|)$. A pseudocode of SIGNet is shown in Algorithm 1.

2.5 Experiments

In this section we present our empirical evaluation of SIGNet compared to other state-of-the-art methods, with a view toward answering the following questions:

1. Are the node embeddings learned by SIGNet interpretable? (Section 2.5.2)
2. Does the embedding space learned by SIGNet support structural balance theory? (Section 2.5.3)
3. Are representations learned by SIGNet effective at edge label prediction? (Section 2.5.4)
4. Are representations learned by SIGNet effective at node label prediction? (Section 2.5.5)
5. How much more effective is our sampling strategy in the presence of partial information? (Section 2.5.6)
6. How scalable is SIGNet for large networks? (Section 2.5.7)
7. Do parameter variations in SIGNet lead to overfitting? (Section 2.5.8)

2.5.1 Experimental Setup

We compare our algorithm against both the state-of-the-art method proposed for signed and unsigned network embedding. The description of the methods are below:

- node2vec [20]: This method, not specific to signed networks, computes embeddings by optimizing the neighborhood structure using informed random walks.
- SiNE [71]: This method uses a multi-layer neural network to learn the embedding by optimizing an objective function satisfying structural balance theory. SiNE only concentrates on the immediate neighborhood of vertices rather than on the global balance structure.
Chapter 2. Generating User Features Suitable for Rumor Identification by Exploiting the Network Structure of Social Media

- **SIGNet-NS**: This method is similar to our proposed method SIGNet except it uses conventional negative sampling instead of our proposed targeted node sampling.

- **SIGNet**: This is our proposed SIGNet method which uses random walks to construct a cache of positive and negative examples for targeted node sampling.

Although node2vec is only applicable to unsigned networks, we use this method to demonstrate the effects of unsigned network embedding methods when applied over signed networks. We skip hand crafted feature generation strategies for link prediction (e.g., [45]) because they cannot be applied to node label prediction and already demonstrate inferior performance compared to SiNE [71].

In the discussion below, we focus on five real world signed network datasets (see Table 2.1). Out of these five, two datasets are from social network platforms—Epinions and Slashdot—courtesy the Stanford Network Analysis Project (SNAP). The details on how the signed edges are defined are available at the project website\(^1\). The third dataset comprises voting records of Wikipedia adminship elections (Wiki), also from SNAP. The fourth dataset we study is an adjective network (ADJNet) constructed from the synonyms and antonyms collected from Wordnet database. Label information about whether the adjective is positive or negative comes from SentiWordNet \(^2\).

The last dataset is a citation network we constructed from written case opinions of the Supreme Court of the United States (SCOTUS). We expand the notion of SCOTUS citation network \(^17\) into a signed network. To understand this network, it is important to note that there are typically two main parts to a SCOTUS case opinion. The first part contains the majority and any optional concurring opinions where justices cite previously argued cases to defend their position. The second part (optional, does not exist in a unanimous decision) consists of dissenting opinions containing arguments opposing the decision of the majority opinion. In our modeling, nodes denote cases (not opinions). The citation of one case’s majority opinion to another case will form a positive relationship, and citations from dissenting opinions will form a negative relationship. We collected all written options from the inception of SCOTUS to construct the citation network. Moreover, we also collected the decision direction of supreme court cases from The Supreme Court Database\(^3\). This decision direction denotes whether the decision is conservative or liberal, information that we will use for validation. We also use 3 synthetic datasets in section 2.5.5, details are in the corresponding section.

Unless otherwise stated, we set the dimension of \(\mathbf{x}_i\) and \(\mathbf{y}_i\) to 20 for both SIGNet-NS and SIGNet. Therefore the final embedding for each node becomes 40. For a fair comparison, the embedding dimension for node2vec and SiNE is set to 40. We all set the total number of samples (examples) to 100 million and \(N = 5\) for SIGNet-NS and SIGNet. Moreover, we

---
\(^1\)http://snap.stanford.edu/
\(^2\)http://sentiwordnet.isti.cnr.it/
\(^3\)http://scdb.wustl.edu/
Table 2.1: Statistics of the datasets used for performance evaluation. In social network datasets negative edges are underrepresented, however in ADJNet and SCOTUS they are well represented. ADJNet and SCOTUS also contain binary labels.

<table>
<thead>
<tr>
<th>Statistics</th>
<th>Epinions</th>
<th>Slashdot</th>
<th>Wiki</th>
<th>ADJNet</th>
<th>SCOTUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>total nodes</td>
<td>131828</td>
<td>82144</td>
<td>7220</td>
<td>4579</td>
<td>28305</td>
</tr>
<tr>
<td>positive edges</td>
<td>717667</td>
<td>425072</td>
<td>83717</td>
<td>10708</td>
<td>43781</td>
</tr>
<tr>
<td>negative edges</td>
<td>123705</td>
<td>124130</td>
<td>28422</td>
<td>7044</td>
<td>42102</td>
</tr>
<tr>
<td>total edges</td>
<td>841372</td>
<td>549202</td>
<td>112139</td>
<td>17752</td>
<td>85883</td>
</tr>
<tr>
<td>% negative edges</td>
<td>14.703</td>
<td>22.602</td>
<td>25.345</td>
<td>39.680</td>
<td>49.023</td>
</tr>
</tbody>
</table>

set $l = 50$ and $r = 1$ for SIGNet. For all the other parameters for node2vec and SiNE we use the settings recommended in their respective papers.

### 2.5.2 Are Embeddings Interpretable?

For visual depiction of embeddings, we first utilize a small dataset denoting relations between sixteen tribes in Central Highlands of New Guinea [59]. This is a signed network showing the alliance and hostility between the tribes. We learned the embeddings in two dimensional space as an undirected network as shown in Fig. 2.6. We can see that in general solid blue edges (alliance) are shorter than the dashed red edges (hostility) confirming that allied tribes are closer than the hostile tribes. One notable point is tribe MASIL has no enemies and often works as a peace negotiator between the tribes. We can see that MASIL positions nicely between two groups of tribes \{OVE, GAHUk, ASARO, UKUDZ, ALiKA, GEHAM\} and \{UHETO, SEUVE, NAGAM, KOHIK, NOTOH\}. The tribes within these two groups are only allied to each other and MASIL but they are hostile to other tribes belonging to different groups. This actually justifies the position of MASIL. As reported in [23] there is another such group which consists of the tribes NAGAD, KOTUN, GAMA, GAVEV; notice that they position themselves in the lower left corner far away from other two groups. Therefore the embedding space learned by SIGNet clearly depicts alliances and relationships among the tribes.

### 2.5.3 Analyzing the Embedding Space

Here we present our analysis on whether the embedding space learned by SIGNet follows the principles of structural balance theory. We calculate the mean Euclidean distance between representations of nodes connected by positive versus negative edges, as well as their standard deviations (see Table 2.2). The lower value of positive edges suggests positively connected nodes stay closer together than the negatively connected nodes indicating that SIGNet has successfully learned the embedding using the principles of structural balance theory. Moreover, the ratio of average distance between the positive and negative edges is at
Chapter 2. Generating User Features Suitable for Rumor Identification by Exploiting the Network Structure of Social Media

Figure 2.6: Two dimensional embedding of alliances among sixteen tribes of New Guinea. Alliance between the tribes is shown in solid blue edges and a hostile relation is shown in dashed red edges. We can see that edges representing alliance are comparatively shorter than the edges represents hostility.

Table 2.2: Average Euclidean distance between node representations connected by positive edges versus negative edges with std. deviation. We can see that the avg. distance between positive edge is significantly lower than negative edges indicating that SIGNet preserves the conditions of structural balance theory.

<table>
<thead>
<tr>
<th>Type of edges</th>
<th>Epinions</th>
<th>Slashdot</th>
<th>Wiki</th>
<th>SCOTUS</th>
<th>ADJNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>positive</td>
<td>0.86 (0.37)</td>
<td>0.98 (0.31)</td>
<td>1.06 (0.27)</td>
<td>0.835 (0.25)</td>
<td>0.71 (0.16)</td>
</tr>
<tr>
<td>negative</td>
<td>1.64 (0.23)</td>
<td>1.60 (0.19)</td>
<td>1.56 (0.19)</td>
<td>1.655 (0.21)</td>
<td>1.77 (0.08)</td>
</tr>
<tr>
<td>ratio</td>
<td>0.524</td>
<td>0.613</td>
<td>0.679</td>
<td>0.505</td>
<td>0.401</td>
</tr>
</tbody>
</table>

most 67% over all the datasets suggesting that SIGNet grasps the principles very effectively.

2.5.4 Edge Label Prediction

We now explore the utility of SIGNet for edge label prediction. For all the datasets we sample 80% of the edges as a training set to learn the node embedding. Then we train a logistic regression classifier using the embedding as features and the sign of the edges as label. This classifier is used to predict the sign of the remaining 20% of the edges. Since
edges involve two nodes we explore several scores to compute the features for edges from the node embedding. They are described below:

1. **Concatenation**: \( f_{ij} = d_i \oplus d_j \)
2. **Average**: \( f_{ij} = \frac{d_i + d_j}{2} \)
3. **Hadamard**: \( f_{ij} = d_i \ast d_j \)
4. **\( L_1 \)**: \( f_{ij} = |d_i - d_j| \)
5. **\( L_2 \)**: \( f_{ij} = |d_i - d_j|^2 \)

Here \( f_{ij} \) is the feature vector of edge \( e_{ij} \) and \( d_i \) is the embedding of node \( v_i \). Except for the method of concatenation (which has a feature vector dimension of 80) other methods use 40-dimensional vectors. Since the datasets are typically imbalanced we use the micro-F1 and AUC scores to evaluate our method. We repeat this process five times and report the average results (see Table 2.3). Some key observations from this table are as follows:

1. **SIGNet**, not surprisingly, outperforms node2vec across all datasets. For datasets that contain relatively fewer negative edges (e.g., 14% for Epinions and 22% for Slashdot), the improvements are modest (around 7%). For Wiki the improvements are moderate (around 12%) where 25% of edges are negative. For ADJNet and SCOTUS where the sign distribution is less skewed, SIGNet outperforms node2vec by a huge margin (19% for ADJNet and 53% for SCOTUS).

2. **SIGNet** demonstrates a consistent advantage over SiNE, with improvements ranging from 7% (for the social network datasets) to 20 − 30% (for ADJNet and SCOTUS).

3. **SIGNet** also outperforms SIGNet-NS in almost all scenarios demonstrating the effectiveness of targeted node sampling over negative sampling.

4. Performance measures (across all scores and across all algorithms) are comparatively better for Epinions over other datasets because almost 83% of the nodes in Epinions satisfy the structural balance condition [15]. As a result edge label prediction is comparatively easier than in other datasets.

5. The feature scoring method has a noticeable impact w.r.t. different datasets. The Average and Concatenation methods subsidize differences whereas the Hadamard, \( L_1 \) and \( L_2 \) methods promote differences. To understand why this makes a difference, consider networks like ADJNet and SCOTUS where connected components denote strong polarities (e.g., denoting synonyms or justice leanings, respectively). In such networks, the Hadamard, \( L_1 \) and \( L_2 \) methods provide more discriminatory features. On the other hand, Epinions and Slashdot are relatively large datasets with diversified communities and so all these methods perform nearly comparably.
Table 2.3: Comparison of edge label prediction in all four datasets. We show the micro F1 and AUC score for each feature scoring method. The best F1 score and AUC score across all the scoring method is shown in larger boldface. SIGNet outperforms node2vec and SiNE in every case. The results are statistically significant with \( p \)-value less than 0.01.

<table>
<thead>
<tr>
<th>Eval.</th>
<th>Dataset Name</th>
<th>Epinions</th>
<th>Slashdot</th>
<th>Wiki</th>
<th>ADJNet</th>
<th>SCOTUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>concat</td>
<td>node2vec</td>
<td>0.831</td>
<td>0.776</td>
<td>0.749</td>
<td>0.594</td>
<td>0.513</td>
</tr>
<tr>
<td></td>
<td>SiNE</td>
<td>0.853</td>
<td>0.774</td>
<td>0.745</td>
<td>0.598</td>
<td>0.607</td>
</tr>
<tr>
<td></td>
<td>SIGNet-NS</td>
<td>0.911</td>
<td>0.793</td>
<td>0.816</td>
<td>0.599</td>
<td>0.560</td>
</tr>
<tr>
<td></td>
<td>SIGNet</td>
<td><strong>0.920</strong></td>
<td><strong>0.832</strong></td>
<td><strong>0.845</strong></td>
<td>0.573</td>
<td>0.557</td>
</tr>
<tr>
<td>avg</td>
<td>node2vec</td>
<td>0.853</td>
<td>0.775</td>
<td>0.747</td>
<td>0.603</td>
<td>0.516</td>
</tr>
<tr>
<td></td>
<td>SiNE</td>
<td>0.853</td>
<td>0.774</td>
<td>0.745</td>
<td>0.599</td>
<td>0.607</td>
</tr>
<tr>
<td></td>
<td>SIGNet-NS</td>
<td>0.837</td>
<td>0.774</td>
<td>0.769</td>
<td>0.620</td>
<td>0.509</td>
</tr>
<tr>
<td></td>
<td>SIGNet</td>
<td>0.879</td>
<td>0.809</td>
<td>0.801</td>
<td>0.574</td>
<td>0.512</td>
</tr>
<tr>
<td>had</td>
<td>node2vec</td>
<td>0.852</td>
<td>0.773</td>
<td>0.747</td>
<td>0.600</td>
<td>0.512</td>
</tr>
<tr>
<td></td>
<td>SiNE</td>
<td>0.853</td>
<td>0.774</td>
<td>0.745</td>
<td>0.598</td>
<td>0.607</td>
</tr>
<tr>
<td></td>
<td>SIGNet-NS</td>
<td>0.846</td>
<td>0.757</td>
<td>0.741</td>
<td>0.705</td>
<td>0.793</td>
</tr>
<tr>
<td></td>
<td>SIGNet</td>
<td>0.883</td>
<td>0.782</td>
<td>0.754</td>
<td>0.722</td>
<td>0.792</td>
</tr>
<tr>
<td>l1</td>
<td>node2vec</td>
<td>0.852</td>
<td>0.774</td>
<td>0.747</td>
<td>0.600</td>
<td>0.509</td>
</tr>
<tr>
<td></td>
<td>SiNE</td>
<td>0.853</td>
<td>0.774</td>
<td>0.745</td>
<td>0.598</td>
<td>0.607</td>
</tr>
<tr>
<td></td>
<td>SIGNet-NS</td>
<td>0.851</td>
<td>0.764</td>
<td>0.743</td>
<td>0.639</td>
<td>0.723</td>
</tr>
<tr>
<td></td>
<td>SIGNet</td>
<td>0.901</td>
<td>0.787</td>
<td>0.751</td>
<td>0.703</td>
<td>0.723</td>
</tr>
<tr>
<td>l2</td>
<td>node2vec</td>
<td>0.852</td>
<td>0.774</td>
<td>0.747</td>
<td>0.601</td>
<td>0.509</td>
</tr>
<tr>
<td></td>
<td>SiNE</td>
<td>0.787</td>
<td>0.774</td>
<td>0.745</td>
<td>0.598</td>
<td>0.607</td>
</tr>
<tr>
<td></td>
<td>SIGNet-NS</td>
<td>0.848</td>
<td>0.763</td>
<td>0.743</td>
<td>0.659</td>
<td>0.742</td>
</tr>
<tr>
<td></td>
<td>SIGNet</td>
<td>0.903</td>
<td>0.809</td>
<td>0.753</td>
<td>0.716</td>
<td>0.745</td>
</tr>
<tr>
<td>imp. over node2vec</td>
<td><strong>7.855</strong></td>
<td><strong>7.216</strong></td>
<td><strong>12.817</strong></td>
<td><strong>19.735</strong></td>
<td><strong>53.488</strong></td>
<td></td>
</tr>
<tr>
<td>imp. over SiNE</td>
<td><strong>7.855</strong></td>
<td><strong>7.494</strong></td>
<td><strong>13.423</strong></td>
<td><strong>20.534</strong></td>
<td><strong>30.478</strong></td>
<td></td>
</tr>
<tr>
<td>imp. over SIGNet-NS</td>
<td><strong>0.988</strong></td>
<td><strong>4.918</strong></td>
<td><strong>3.554</strong></td>
<td><strong>2.411</strong></td>
<td><strong>-0.126</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Eval.</th>
<th>Dataset Name</th>
<th>Epinions</th>
<th>Slashdot</th>
<th>Wiki</th>
<th>ADJNet</th>
<th>SCOTUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>concat</td>
<td>node2vec</td>
<td>0.773</td>
<td>0.719</td>
<td>0.654</td>
<td>0.589</td>
<td>0.516</td>
</tr>
<tr>
<td></td>
<td>SiNE</td>
<td>0.560</td>
<td>0.547</td>
<td>0.574</td>
<td>0.513</td>
<td>0.520</td>
</tr>
<tr>
<td></td>
<td>SIGNet-NS</td>
<td>0.851</td>
<td>0.780</td>
<td>0.830</td>
<td>0.606</td>
<td>0.555</td>
</tr>
<tr>
<td></td>
<td>SIGNet</td>
<td><strong>0.918</strong></td>
<td><strong>0.863</strong></td>
<td><strong>0.891</strong></td>
<td>0.561</td>
<td>0.555</td>
</tr>
<tr>
<td>avg</td>
<td>node2vec</td>
<td>0.745</td>
<td>0.715</td>
<td>0.606</td>
<td>0.607</td>
<td>0.517</td>
</tr>
<tr>
<td></td>
<td>SiNE</td>
<td>0.547</td>
<td>0.539</td>
<td>0.553</td>
<td>0.528</td>
<td>0.517</td>
</tr>
<tr>
<td></td>
<td>SIGNet-NS</td>
<td>0.733</td>
<td>0.742</td>
<td>0.757</td>
<td>0.675</td>
<td>0.537</td>
</tr>
<tr>
<td></td>
<td>SIGNet</td>
<td>0.876</td>
<td>0.823</td>
<td>0.815</td>
<td>0.571</td>
<td>0.540</td>
</tr>
<tr>
<td>had</td>
<td>node2vec</td>
<td>0.713</td>
<td>0.656</td>
<td>0.603</td>
<td>0.559</td>
<td>0.513</td>
</tr>
<tr>
<td></td>
<td>SiNE</td>
<td>0.540</td>
<td>0.536</td>
<td>0.548</td>
<td>0.526</td>
<td>0.515</td>
</tr>
<tr>
<td></td>
<td>SIGNet-NS</td>
<td>0.819</td>
<td>0.691</td>
<td>0.661</td>
<td>0.778</td>
<td>0.800</td>
</tr>
<tr>
<td></td>
<td>SIGNet</td>
<td>0.901</td>
<td>0.774</td>
<td>0.710</td>
<td>0.800</td>
<td>0.881</td>
</tr>
<tr>
<td>l1</td>
<td>node2vec</td>
<td>0.673</td>
<td>0.563</td>
<td>0.582</td>
<td>0.525</td>
<td>0.509</td>
</tr>
<tr>
<td></td>
<td>SiNE</td>
<td>0.592</td>
<td>0.516</td>
<td>0.522</td>
<td>0.525</td>
<td>0.507</td>
</tr>
<tr>
<td></td>
<td>SIGNet-NS</td>
<td>0.788</td>
<td>0.621</td>
<td>0.615</td>
<td>0.679</td>
<td>0.780</td>
</tr>
<tr>
<td></td>
<td>SIGNet</td>
<td>0.898</td>
<td>0.770</td>
<td>0.673</td>
<td>0.774</td>
<td>0.781</td>
</tr>
<tr>
<td>l2</td>
<td>node2vec</td>
<td>0.672</td>
<td>0.562</td>
<td>0.576</td>
<td>0.541</td>
<td>0.507</td>
</tr>
<tr>
<td></td>
<td>SiNE</td>
<td>0.560</td>
<td>0.519</td>
<td>0.528</td>
<td>0.526</td>
<td>0.509</td>
</tr>
<tr>
<td></td>
<td>SIGNet-NS</td>
<td>0.791</td>
<td>0.626</td>
<td>0.620</td>
<td>0.694</td>
<td>0.797</td>
</tr>
<tr>
<td></td>
<td>SIGNet</td>
<td>0.904</td>
<td>0.780</td>
<td>0.682</td>
<td>0.788</td>
<td>0.800</td>
</tr>
<tr>
<td>imp. over node2vec</td>
<td>18.758</td>
<td>20.028</td>
<td>36.239</td>
<td>31.796</td>
<td>70.406</td>
<td></td>
</tr>
<tr>
<td>imp. over SiNE</td>
<td>55.068</td>
<td>57.770</td>
<td>55.226</td>
<td>51.515</td>
<td>69.423</td>
<td></td>
</tr>
<tr>
<td>imp. over SIGNet-NS</td>
<td>7.873</td>
<td>10.641</td>
<td>7.349</td>
<td>2.828</td>
<td>0.114</td>
<td></td>
</tr>
</tbody>
</table>

### 2.5.5 Node Label Prediction

For datasets like SCOTUS and ADJNet (where nodes are annotated with labels), we learn a logistic regression classifier to map from node representations to corresponding labels.
Table 2.4: Comparison of methods for node label prediction on real world datasets. SIGNet outperforms SiNE and node2vec in all datasets.

<table>
<thead>
<tr>
<th>Performance measure</th>
<th>Algorithms</th>
<th>ADJNet</th>
<th>SCOTUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>micro f1</td>
<td>node2vec</td>
<td>0.5284</td>
<td>0.5392</td>
</tr>
<tr>
<td></td>
<td>SiNE</td>
<td>0.6257</td>
<td>0.6131</td>
</tr>
<tr>
<td></td>
<td>SIGNet-NS</td>
<td>0.7292</td>
<td>0.8004</td>
</tr>
<tr>
<td></td>
<td>SIGNet</td>
<td>0.8380</td>
<td>0.8419</td>
</tr>
<tr>
<td>imp (%) over node2vec</td>
<td>58.5920</td>
<td>56.1387</td>
<td></td>
</tr>
<tr>
<td>imp (%) over SiNE</td>
<td>33.9300</td>
<td>37.3185</td>
<td></td>
</tr>
<tr>
<td>imp (%) over SIGNet-NS</td>
<td>14.9205</td>
<td>5.1849</td>
<td></td>
</tr>
<tr>
<td>macro f1</td>
<td>node2vec</td>
<td>0.4605</td>
<td>0.4922</td>
</tr>
<tr>
<td></td>
<td>SiNE</td>
<td>0.5847</td>
<td>0.5696</td>
</tr>
<tr>
<td></td>
<td>SIGNet-NS</td>
<td>0.7261</td>
<td>0.7997</td>
</tr>
<tr>
<td></td>
<td>SIGNet</td>
<td>0.8374</td>
<td>0.8415</td>
</tr>
<tr>
<td>imp (%) over node2vec</td>
<td>81.8458</td>
<td>70.9671</td>
<td></td>
</tr>
<tr>
<td>imp (%) over SiNE</td>
<td>43.2187</td>
<td>47.7353</td>
<td></td>
</tr>
<tr>
<td>imp (%) over SIGNet-NS</td>
<td>15.3285</td>
<td>5.2270</td>
<td></td>
</tr>
</tbody>
</table>

(with a 50-50 training-test split). We also repeat this five times and report the average. See Table 2.4 for results. As can be seen, SIGNet consistently outperforms the SiNE and node2vec approaches. In particular, in the case of SCOTUS which is a citation network, some cases have a huge number of citations (i.e. landmark cases) in both ideologies. Targeted node sampling, by adding such cases to either $\eta^+_i$ or $\eta^-_i$, situates the embedding space close to the landmark cases if they are in $\eta^+_i$ or away from them if they are in $\eta^-_i$, thus supporting accurate node prediction.

The case of Citizens United vs. Federal Election Commission (FEC), one of the most controversial cases in recent times, is instructive. In this case, Citizens United seeks an injection against the FEC to prevent the application of the Bipartisan Campaign Reform Act (BCRA) so that a film on Hillary Clinton can be broadcasted. In a 5-4 vote, the court decides in favor of Citizens United. In Fig. 2.8, we depict the BCRA related cases that cite Citizens United vs. Federal Election Commission in Fig. 2.8 (in a 2D projection). The cases whose decisions support a conservative view are shown in red and the cases which support a liberal point of view are shown in blue. Another two cases disputing the application of BCRA cite this case (shown in filled circles), viz. Williams-Yulee vs The Florida Bar and McCutcheon vs FEC. In the first case the court supports the liberal point-of-view (shown in blue) and cites the case negatively (shown in dashed line). Therefore, its embedding resides far away from the Citizens United case. In McCutcheon vs FEC, the court supports a conservative point-of-view and decides in favor of McCutcheon. This case positively cites Citizens United...
Chapter 2. Generating User Features Suitable for Rumor Identification by Exploiting the Network Structure of Social Media

case and its embedding is therefore positioned closer to it.

One limitation of ADJNet and SCOTUS is that nodes are tagged with binary data. Although binary labeling seems plausible in a perfectly balanced signed network, it is possible to find the extension of this behavior in many social media analysis. For example, in an election campaign network, there could be multiple candidates, where supporters of one candidate are positively connected with each other but are negatively connected with supporters of other candidates.

Unfortunately, to the best of our knowledge there is no publicly available dataset for this evaluation. This is why we resort to synthetic datasets in performance evaluation. We generate the networks based on the method proposed in [9]. Given a total number of nodes \( N_V \), number of node labels \( N_G \) and sparsity score \( \alpha \), we first create \( N_G \) subgraphs from \( N_V \) nodes having only positive edges within the subgraphs. The nodes of \( i \)th subgraphs are labeled as class \( i \). Then we connect the subgraphs using exclusively negative edges. We also add random positive and negative edges as noise where \( \alpha \) controls the total number of edges. We create 3 synthetic datasets, each with \( N_V = 50000 \) nodes, and \( N_G \) set to 10 (Syn 10), 20 (Syn 20), and 50 (Syn 50).

We train a one-vs-rest logistic regression classifier for the prediction with a 50-50 training-test split. The result is shown in Table 2.5. We can see that SIGNet, not surprisingly,
2.5. Experiments 29

Figure 2.8: Several conservatively and liberally disputed cases including Bipartisan Campaign Reform Act (BCRA) related cases that cite Citizens United vs. Federal Election Commission. Conservatively disputed cases are shown in red and liberally disputed cases are shown in blue. Our discussed cases are shown in filled circles while other cases are shown in unfilled circles. Dashed edges represents negatively connected and solid edges represents positively connected.

Table 2.5: Comparison of multiclass prediction on synthetic datasets using a one-vs-rest logistic regression classifier. SIGNet outperforms all the other methods in all datasets.

<table>
<thead>
<tr>
<th>Performance measure</th>
<th>Algorithms</th>
<th>Syn 10</th>
<th>Syn 20</th>
<th>Syn 50</th>
</tr>
</thead>
<tbody>
<tr>
<td>micro f1</td>
<td>node2vec</td>
<td>0.1112</td>
<td>0.0527</td>
<td>0.0195</td>
</tr>
<tr>
<td></td>
<td>SiNE</td>
<td>0.1105</td>
<td>0.0545</td>
<td>0.0197</td>
</tr>
<tr>
<td></td>
<td>SIGNet-NS</td>
<td>0.1483</td>
<td>0.0848</td>
<td>0.0519</td>
</tr>
<tr>
<td></td>
<td>SIGNet</td>
<td><strong>0.1723</strong></td>
<td><strong>0.1104</strong></td>
<td><strong>0.0716</strong></td>
</tr>
<tr>
<td>gain (%) of SIGNet</td>
<td></td>
<td>16.1834</td>
<td>30.1887</td>
<td>37.9576</td>
</tr>
</tbody>
</table>

| macro f1            | node2vec   | 0.0967 | 0.0283 | 0.0032 |
|                     | SiNE       | 0.1083 | 0.0535 | 0.0187 |
|                     | SIGNet-NS  | 0.1344 | 0.0747 | 0.0486 |
|                     | SIGNet     | **0.1695** | **0.1084** | **0.0704** |
| gain (%) of SIGNet  |            | 26.1161 | 45.1138 | 44.8560 |
Chapter 2. Generating User Features Suitable for Rumor Identification by Exploiting the Network Structure of Social Media

Figure 2.9: Performance evaluation on ADJNet and SCOTUS datasets varying the percent of nodes used for training. The $x$-axis shows the percent of nodes whose information is used to learn the embedding. $y$-axis shows the micro F1 score (top) and macro F1 score (bottom) for each dataset. SIGNet outperforms SIGNet-NS in all cases.

outperforms other methods by a considerable margin.

2.5.6 Node Label Prediction with Partial Information

To evaluate the effectiveness of our targeted node sampling versus negative sampling, we remove all outgoing edges of a certain percent of randomly selected nodes (test nodes), learn an embedding, and then aim to predict the labels of the test nodes. We show the F-1 scores for SCOTUS and ADJNet in Fig. 2.9. As seen here, SIGNet consistently outperforms SIGNet-NS. Withholding the outgoing edges of test nodes implies that both methods will miss the same edge information in learning the embedding. However due to targeted node sampling many of these test nodes will be added to $\eta^+_i$ or $\eta^-_i$ in SIGNet (recall only the outgoing edges are removed, but not incoming edges). Because of this property, SIGNet will be able to make an informed choice while optimizing the embedding space.
2.5. Experiments

2.5.7 Scalability

To assess the scalability of SIGNet, we learn embeddings for an Erdos-Renyi random network for up to one million nodes. The average degree for each node is set to 10 and the total number of samples is set to 100 times the number of edges in the network. The size of the dimension is also set to 100 for this experiment. We make the network signed by randomly changing the sign of 20% edges to negative. The optimization time and the total execution time (targeted node sampling + optimization) is compared in Fig. 2.10 (a) for different vertex sizes. On a regular desktop, an unparallelized version of SIGNet requires less than 3 hours to learn the embedding space for over 1 million nodes. Moreover, the sampling times is negligible compared to the optimization time (less than 15 minutes for 1 million nodes). This actually shows SIGNet is very scalable for real world networks. Since SIGNet uses an asynchronous stochastic gradient approach, it is trivially parallelizable and as Fig. 2.10(b) shows, we can obtain a 3.5 fold improvement with just 5 threads, with diminishing returns beyond that point.

2.5.8 Parameter Sensitivity

Fig. 2.11 (a) and (b) show that while we see improvements in macro F1 score for node label prediction up to 100 million samples, further optimization beyond that point is not necessary as the performance gets saturated. A similar behavior is seen with size of the embedding dimension in Fig. 2.11 (c) and (d) (saturation after 1000). For other parameters like walk length \(l\) and size of node sampling \(N\) higher values lead to negligible increases in \(F1\) score.

Figure 2.10: Scalability of SIGNet on Erdos-Renyi random graphs. (a) execution time of SIGNet varying the number of nodes; (b) execution time of SIGNet varying the number of threads.
Chapter 2. Generating User Features Suitable for Rumor Identification by Exploiting the Network Structure of Social Media

Figure 2.11: Performance evaluation on ADJNet and SCOTUS dataset varying different parameters for node label prediction. Top two figures show macro F1 score varying total number of samples. Bottom figures show macro F1 score varying the dimension size of embedding $K$.

2.6 Discussion

We have presented a scalable feature learning framework suitable for signed networks. Using a targeted node sampling for random walks, and leveraging structural balance theory, we have shown how the embedding space learned by SIGNet yields interpretable as well as effective representations. In the next chapter, we will demonstrate how the user embedding computed from SIGNet can be used in a neural network framework to classify rumor veracity.
Chapter 3

Integrating Computed User Feature Into a Cascade Prediction Model to Identify the Veracity of Rumor Threads

3.1 Introduction

In this chapter we describe a way to incorporate the user trust embedding into rumor veracity detection methods. Leveraging only the underlying social network structure and users’ posting time and stance in discussing a particular story we aim to identify whether the story under discussion is true, false, or unverified. For this purpose, we use the feature representation from problem 1—referred to as a user trust embedding—to gauge the level of trust and distrust among users. An LSTM-based deep learning approach AdaLSTM is then developed to assess the veracity of the story. AdaLSTM can mitigate two critical issues of LSTM while analyzing a cascade. It can analyze a small portion of the thread at a time thus enabling it to reduce computational overhead. Additionally, it can also revisit an earlier part of the thread if required providing more flexibility. We present empirical results on two datasets containing public discussions of several breaking news events on Twitter with an underlying massive network of over 22 and 19 million users.

3.2 Problem Formulation

Consider a collection of conversational threads on a social media platform (e.g. Twitter) \( \mathcal{C} = \{C_1, C_2, \ldots, C_{|\mathcal{C}|}\} \). Each thread \( C_i \) begins with a source tweet mentioning or linking to a story and the rest of the thread features people discussing (e.g. commenting about) it. Each entry \( c_{ij} \) in \( C_i \) represents an interaction of a user \( u_{ij} \). Each thread is associated with a label \( y_i \), which represents whether the story being discussed is true, false or unverified. Besides, the underlying network structure of users \( \mathcal{G} \) is also given. Such a network \( \mathcal{G} \) can be constructed using a range interactions such as follower-followee, retweet or reshare, mentions or a combination of them. Suppose \( \mathcal{G} = (\mathcal{V}, \mathcal{E}) \) a directed network, where \( \mathcal{V} \) is the set
Chapter 3. Integrating Computed User Feature Into a Cascade Prediction Model to Identify the Veracity of Rumor Threads

of vertices and $\mathcal{E}$ is the set of edges. Our goal is to predict the labels of a set of test conversational threads $\mathcal{C}_{test}$.

Figure 3.1: An overview of how rumor propagates in social media. An otherwise innocuous event or development (e.g., Russian President Putin not showing up in public for 10 days\(^4\)) causes people to speculate and initiate new lines of discussion. We exploit the underlying social structure and conversational thread to identify whether the story is true (non-rumor), a rumor, or unverified (best viewed in color).

3.3 Introducing \textit{AdaLSTM}

Given a thread $C = [(u_1), \ldots, (u_n)]$ we can apply any recurrent neural network (RNN) model to classify the thread story since the input is sequential in nature and the thread length is variable. Especially we use a Long Short-Term Memory (LSTM) model [27] for this purpose.

However, the naïve LSTM model has two limitations in terms of modeling a conversational thread. For instance, when analyzing a long sequence of conversation, the model does not provide a good mechanism to revisit an earlier portion of the cascade if required. Another issue is that the thread sequences can be inherently longer than standard text sentences, making the LSTM computationally expensive. To deal with these shortcomings, we propose \textit{AdaLSTM} which can deal with these drawbacks.

\textbf{Model Description:} Our proposed \textit{AdaLSTM} has two major components. The first is the Thread Analyzer Module (TAM). TAM takes a conversational thread sequence and a valid index within the thread sequence. Then it extracts a subsequence at the given index and passes it to an LSTM model. The description of the processing of a subsequence is given below:

As long standing practice, we represent each node in the subsequence as a vector (not a one-hot vector), where the vector is the pre-trained embedding $\mathcal{D}$ computed from SIGNet. Now to extract the embedding of a specific node $v_q$ we can use a one-hot vector $q$, where $|q| = |\mathcal{V}|$ and multiply it with $\mathcal{D}$ i.e. $d_q = \mathcal{D} \cdot q$.

3.3. Introducing AdaLSTM

Now for the $j^{th}$ element of the input subsequence $\langle u_j \rangle$, suppose the embedding of the user $u_j$ is $d_{u_j}$. Then the LSTM equations are as follows:

\begin{align*}
i_j &= \sigma(W^i_u d_{u_j} + U_i h_{j-1} + b_i) \\
f_j &= \sigma(W^f_u d_{u_j} + U_f h_{j-1} + b_f) \\
\tilde{C}_j &= \tanh(W^c_u d_{u_j} + U_c h_{j-1} + b_c) \\
C_j &= \tilde{C}_j \odot i_j + f_j \odot C_{j-1} \\
o_j &= \sigma(W^o_u d_{u_j} + U_o h_{j-1} + b_o) \\
h_j &= o_j \odot \tanh(C_j)
\end{align*}

The hidden state vector $h_j$ captures the dynamics of conversational thread up to the current point. The last hidden vector of $h_n$ is the summary of the given subsequence of the conversational thread. We can consider $h_n$ as the feature vector of the current subsequence.

Next Subsequence Selection: The hidden state vector from the LSTM is given to a Location Definer Network (LDN) as input. LDN is a fully connected neural network which outputs a valid location w.r.t. the original sequence. This process is repeated $\xi$ number of times and we receive $\xi$ preliminary hidden state vectors.

Final Feature Computation: Instead of using the last hidden vector $h_n$ as feature vector, we apply an approach similar to attention mechanisms [2] to obtain a weighted representation of all the hidden states $h_n$.

\begin{align*}
\psi_k &= \tanh(W^T_\alpha h_j + b_\alpha) \\
\alpha^a_j &= \frac{\exp(\psi^T_j \psi_\alpha)}{\sum_\xi \exp(\psi^T_i \psi_\alpha)} \\
\tilde{h}_n &= \sum_\xi \alpha^a_j h_j
\end{align*}

Here $K$ is the state size, $K_\alpha$ is the attention size and $W_\alpha \in \mathbb{R}^{K \times K_\alpha}$ and $b_\alpha \in \mathbb{R}^{K_\alpha}$ are weight and bias of the attention layer and $\psi_\alpha \in \mathbb{R}^{K_\alpha}$ is the attention context vector. $h_j$ corresponds to the $j^{th}$ member of preliminary hidden state vectors.

Rumor Classification: The score for a conversation to fall into the rumor class can be computed by $w_l = V_l \tilde{h}_n + b_l$, where $V_l \in \mathbb{R}^{L \times K}$ and $b_u \in \mathbb{R}^L$. Here $L$ denotes the set of labels—false rumor, true rumor, unverified rumor. Now we can use a softmax layer to compute the probability of each label:

\[p_l|\tilde{h}_n = \frac{\exp(w_l)}{\sum_{k \in L} \exp(w_k)}\]

The benefit of using AdaLSTM is that it does not process the cascade as a whole, so the computational overhead is comparatively lower. Moreover, since it uses a separate neural
Figure 3.2: A block architecture of AdaLSTM. First from the input thread sequence \(<u_1>, ..., <u_n>\) a subsequence is extracted (here starting at position 6). Then the elements of the subsequence i.e. user and stance is converted into corresponding embedding. The embedding vectors along with the temporal features i.e. inter-event timing is fed to an LSTM model. The last hidden state vector of the LSTM is then given to the Location Definer Network (LDN) to get the next location of the subsequence to process. In this way a set of hidden state vectors are obtained which are passed to an attention layer to obtain a weighted state vector. This vector can be seen as the feature representation of the input cascade. Then a softmax layer is used to get a probability distribution of the rumor label (best viewed in color).

network to get a valid location within the original thread, it can reexamine a subsequence if needed to obtain a better representation.

**Space Complexity:** Let us assume the trust embedding size and the state size of the LSTM are \(F\) and \(K\) respectively. As a result, the size of the LSTM becomes \(4FK + K^2\). Moreover, the matrix size of the attention mechanism and the output layer are \(2K_\alpha + KK_\alpha\) and \(KL\) respectively, where \(L\) is the number of classes. So the total size of the trainable variable is \(4FK + K^2 + 2K_\alpha + KK_\alpha + KL\).
Table 3.1: General information regarding the two datasets used for evaluation including length of the threads. Here F, T and U refers to False, True and Unverified rumor

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Stats.</th>
<th>F</th>
<th>T</th>
<th>U</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># classes</td>
<td>62</td>
<td>137</td>
<td>98</td>
<td>297</td>
</tr>
<tr>
<td>PHEME</td>
<td># of users</td>
<td>613</td>
<td>1563</td>
<td>1260</td>
<td>3359</td>
</tr>
<tr>
<td></td>
<td>max len</td>
<td>46</td>
<td>111</td>
<td>104</td>
<td>111</td>
</tr>
<tr>
<td></td>
<td>avg. len</td>
<td>14.05</td>
<td>15.99</td>
<td>20.73</td>
<td>17.15</td>
</tr>
<tr>
<td>RDTW</td>
<td># classes</td>
<td>60</td>
<td>51</td>
<td>N.A.</td>
<td>111</td>
</tr>
<tr>
<td></td>
<td># of users</td>
<td>41863</td>
<td>67685</td>
<td>N.A.</td>
<td>104328</td>
</tr>
<tr>
<td></td>
<td>max len</td>
<td>23726</td>
<td>20914</td>
<td>N.A.</td>
<td>23726</td>
</tr>
<tr>
<td></td>
<td>avg. len</td>
<td>899.22</td>
<td>2294.98</td>
<td>N.A.</td>
<td>1540.514</td>
</tr>
</tbody>
</table>

3.4 Experiments

Datasets: For evaluation purposes we utilize two datasets. The first is the PHEME rumor scheme dataset [76] which contains 297 conversational threads (5093 posts) discussing several breaking news events. Each conversational thread is associated with one of the three labels: true, false or unverified. Moreover each comment in a thread is also annotated with one of the following user stances: support, deny, comment or question. A short description of the news events covered in the PHEME dataset are as follows:

Charlie Hebdo Shooting: Deadly shooting event that occurred in the office of the newspaper Charlie Hebdo by two men that led to the death of several people.

Sydney Hostage Crisis: Hostage taking of several people in Sydney, Australia by an armed man.


Ferguson Protest: Mass protest after a police officer killed an African-American teenager in Ferguson, MO.

Putin Missing: A rumor about the disappearance of Russian president Vladimir Putin.

Germanwings Crash: A (deliberate) crash of a passenger plane by its co-pilot resulting in the death of all its passengers.

Essien contracted Ebola: A rumor about footballer Essien being infected by the Ebola virus.

Prince Toronto: A rumor of singer Prince playing in Toronto, Canada in secret.

The second dataset we call RDTW is from [42]. This dataset contains 111 threads (8593 posts) on Twitter. General information regarding the two datasets are shown in Table 3.1.
3.4.1 Compared Methods

**Castillio et al. [8]:** This method uses a decision tree classifier over a set of handcrafted features (e.g. linguistic, user, propagation characteristics) extracted from the tweets.

**(RuRNN) [50]:** This model uses the tweet text and leverages a 2 layer GRU model to classify the veracity of rumor.

**Logistic Regression (LogReg):** We represent a conversational thread by the average of all its participating users’ trust embedding and then train a logistic regression classifier.

**node2vec:** We compute the embedding using node2vec algorithm from [20] and use the embedding to classify the rumor label.

**LSTM:** This model uses the user trust embedding which is fed into the LSTM model to classify the veracity of rumors.

**GRU:** This model uses a GRU cell instead of the LSTM.

**AdaLSTM:** This is the proposed model which uses an LSTM architecture and takes the user trust embedding as input to identify the veracity.

3.4.2 Experimental Settings

**Trust network computation:** We construct a follower-followee network from users and their followers in the conversational thread. We use the official Twitter API to construct the network. In PHEME dataset the total number of users in the network is 22,640,739 with 44,397,605 edges. In case of RDTW dataset the number of users is 19,399,067 with 197,020,759 edges. For trustworthiness and trustingness computation we set total number of iteration $N$ to 100, trust embedding size $K_u$ to 512 and involvement factor $\xi$ to 1.

**Parameter settings:** We conduct a 80–10–10 split of the datasets with 5 fold cross validation. The state and batch size is set to 128 and 5 respectively. We run the model for 30 epochs and took the best model based on the performance on the validation set. As

**Evaluation metric:** Since the class labels are imbalanced we use macro F1 in addition to micro F1 scores to evaluate the performance. Macro F1 score computes the F1 score for each class individually and reports the average, while micro F1 calculates performance across all the classes together.

3.4.3 Does the TSM Algorithm Converge?

We calculate the absolute difference of trustworthiness score for each node between two consecutive iterations ($\sum \Delta_{\text{trustworthiness}}$). We do the same for trustingness ($\sum \Delta_{\text{trustingness}}$) score and sum up the two differences and plot it in Fig. 3.3. We can see that the summation of the deviation between the respective scores are decreasing gradually. This illustrates that the TSM algorithm is converging.
Table 3.2: Comparison of performance between AdaLSTM and other methods. Best performing method is shown in boldface while second best method is shown in italics. AdaLSTM outperforms other methods by a very good margin.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Methods</th>
<th>Micro F1</th>
<th>Macro F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHEME</td>
<td>Castillo</td>
<td>46.091</td>
<td>37.538</td>
</tr>
<tr>
<td></td>
<td>LogReg</td>
<td>50.382</td>
<td>38.426</td>
</tr>
<tr>
<td></td>
<td>node2vec</td>
<td>47.111</td>
<td>36.478</td>
</tr>
<tr>
<td></td>
<td>LSTM</td>
<td>43.333</td>
<td>32.024</td>
</tr>
<tr>
<td></td>
<td>GRU</td>
<td>46.667</td>
<td>36.809</td>
</tr>
<tr>
<td></td>
<td>RuRNN</td>
<td>48.148</td>
<td>35.263</td>
</tr>
<tr>
<td></td>
<td><strong>AdaLSTM</strong></td>
<td><strong>52.667</strong></td>
<td><strong>42.849</strong></td>
</tr>
<tr>
<td></td>
<td>Imp. Over second best</td>
<td><strong>4.535</strong></td>
<td><strong>11.510</strong></td>
</tr>
<tr>
<td>RDTW</td>
<td>Castillo</td>
<td>60.671</td>
<td>52.487</td>
</tr>
<tr>
<td></td>
<td>LogReg</td>
<td>54.981</td>
<td>48.582</td>
</tr>
<tr>
<td></td>
<td>node2vec</td>
<td>46.707</td>
<td>52.173</td>
</tr>
<tr>
<td></td>
<td>LSTM</td>
<td>57.120</td>
<td>38.095</td>
</tr>
<tr>
<td></td>
<td>GRU</td>
<td>56.818</td>
<td>42.033</td>
</tr>
<tr>
<td></td>
<td>RuRNN</td>
<td>65.730</td>
<td>63.196</td>
</tr>
<tr>
<td></td>
<td><strong>AdaLSTM</strong></td>
<td><strong>68.619</strong></td>
<td><strong>63.229</strong></td>
</tr>
<tr>
<td></td>
<td>Imp. Over second best</td>
<td><strong>4.395</strong></td>
<td><strong>0.052</strong></td>
</tr>
</tbody>
</table>

3.4.4 How Effective Is AdaLSTM in Classifying Rumors Versus State-of-the-Art Methods?

We show the performance of all the methods in Table 3.2. The summary of our observations is as follows:

First, we can see AdaLSTM outperforms all the other methods by a very good margin. For PHEME dataset it’s nearest competitor is Logistic Regression (LogReg) against which AdaLSTM obtains 5% gain in case of micro F1 score. For macro F1 score the gain is even higher (11%). This shows AdaLSTM tackles the class imbalance problem better than other methods. On the other hand, in case of RDTW dataset AdaLSTM’s nearest competitor is RuRNN proposed by [50]. Here AdaLSTM obtains around 4% gain in case of micro F1 whereas performance is almost equal for macro F1 score.

Second, AdaLSTM outperforms the node2vec model (unsigned network embedding method) in both datasets (14% and 21% gain for PHEME and RDTW respectively for macro F1 score). This shows signed network embedding provides a better feature representation compared to node2vec in this case.
Chapter 3. Integrating Computed User Feature Into a Cascade Prediction Model to Identify the Veracity of Rumor Threads

Third, in both datasets, *AdaLSTM* performs better than LogReg although both are using the user trust embedding. Our hypothesis is *AdaLSTM* can learn a better feature representation than LogReg as the latter method loses some information because of taking an avg. representation of the user trust embedding.

Lastly, although both RNN models (i.e. LSTM and GRU) use the user trust embedding same as LogReg and *AdaLSTM*, they are outperformed by all the methods. Although RuRNN is a GRU based model it only models the textual part of the conversations so it performs quite well in both datasets.

### 3.4.5 Case Study

**Case Study 1:** First we show the effectiveness of user trust embedding by picking one of the stories as a case study. We show the user trust embedding of several people discussing the *Charlie Hebdo Massacre* in Fig. 3.4. We identify two types of users based on their stance: those who support non-rumor stories and reject rumor stories (i.e., Good users) and, second, those who support rumor stories and contradict non-rumor stories (Malicious users). As we can see from Fig. 3.4, both groups are far from each other. This depicts that user trust embedding is effective at separating the good from malicious users.

Figure 3.3: The summation of difference of trust scores for each node between two consecutive iterations. We can see that the difference is decreasing gradually demonstrating that the TSM algorithm is converging.
3.4. Experiments

Figure 3.4: 3D projection of user trust embedding discussing the event *Charlie Hebdo Mas- sacre*. Good users are the ones who support a non-rumor story or contradict rumors whereas malicious users either deny a non-rumor story or support a rumor (best viewed in color).

Table 3.3: Two examples of subsequences picked by *AdaLSTM*. These subsequences were the most highly attended by *AdaLSTM*. We only show users whose support the thread. As we can see the trustworthiness score of rumor supporters are drastically lower than the non-rumor supporters.

<table>
<thead>
<tr>
<th>false rumor</th>
<th>true rumor</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Text</strong>: #breaking: 1 dead, several injured in shootout north-east of paris ... ...</td>
<td><strong>Text</strong>: 11 confirmed dead, francois hollande to visit scene of attack ... ...</td>
</tr>
<tr>
<td>User 1</td>
<td><strong>Stance</strong>: Supporting</td>
</tr>
<tr>
<td><strong>trustworthiness score</strong>: ranks among bottom 1%</td>
<td><strong>trustworthiness score</strong>: ranks among top 0.1%</td>
</tr>
<tr>
<td>User 2</td>
<td><strong>Stance</strong>: Supporting</td>
</tr>
<tr>
<td><strong>trustworthiness score</strong>: ranks among bottom 1%</td>
<td><strong>trustworthiness score</strong>: ranks among top 51%</td>
</tr>
</tbody>
</table>

Case Study 2: Next we show several elements of subsequences, where *AdaLSTM* places the most attention (i.e. where the attention weights are the highest) in Table 3.3. Looking at the supporters in the subsequence we can see the rumor supporting users rank among the bottom 1% in terms of the trustworthiness score. On the other hand, the non-rumor supporters are ranked much higher. Therefore we see that *AdaLSTM* has identified important subsequences to classify rumors. (As an aside, based on this result, we also built a model that uses the average trustworthiness scores of all the participating users in a thread to classify rumors. However this approach does not result in good performance.)
3.5 Discussion

In this chapter, we propose a rumor veracity detection model in social media. This model uses user embedding generated from social networks in an unsupervised manner. The user embedding is then passed to a cascade prediction model to classify the veracity of the discussed rumor thread. One of the problems of this model is that one needs to precompute the user embedding and update it periodically to detect the rumor veracity, which can be cumbersome. To mitigate this our next problem leverages user profile information directly to detect rumor veracity.
Chapter 4

Leveraging User Profile Information Into the Cascade Prediction Model to Detect the Veracity of Rumors

4.1 Introduction

In this chapter, we present a machine learning method to detect rumor veracity user profile information. Our proposed method RumorSleuth can handle user information and learn a rich vocabulary of features to support detection or classification. We effectively integrate variational autoencoders (VAE) into the LSTM framework to model user features. The reason behind using user profile is that generic user embedding computation often requires the underlying social network which is difficult to obtain. We evaluate our model over two publicly available datasets which we used in Problem 2 and compare it against other baseline models used in the previous problem. Our model is able to outperform other models and perform on par with AdaLSTM proposed in Problem 2.

4.2 Proposed Model

Rumors constitute interesting news content or stories with questionable credibility. Due to their inherently appealing nature, rumors are widely circulated among users of social media platforms. A rumor can turn out to be True, False or Unverified. Any other circulating story can be considered as a non-rumor.

User Representation: In rumor detection research, usually only linguistic features of a post are used [41, 50, 51] with the notable exception of [42]. However, it has been shown that the characteristics of a user also contributes an impact on rumor propagation [8]. To demonstrate this we plot the CCDF of friends and status count of users who first shared the rumor posts in Fig 4.1 (a) and (b) for the PHEME dataset, which is one of the datasets used in our evaluation. It can be seen that false rumor discussions are mostly initiated by users who have comparatively lower friends and status counts. We also show the (%) of verified
Chapter 4. Leveraging User Profile Information Into the Cascade Prediction Model to Detect the Veracity of Rumors

Figure 4.1: (a) and (b) shows the CCDF of friends and status count of users who initiate true and false rumors. We can see that people who initiate a false rumor usually have lower status or friends count. (c) shows (%) of verified and non-verified accounts that start a rumor post. It is quite clear that unverified accounts are often responsible for starting a false rumor.

and non-verified accounts for both true and false rumors in Fig. 4.1(c) and observe that most of the time the account of false rumor starters are not verified. This analysis supports our hypothesis that rumor detection can benefit from leveraging user profile information.

One of the possible ways to incorporate user feature into our model is to apply unsupervised learning methods such as autoencoders to obtain a compact feature representation. Then the latent embedding of the autoencoder can be used in the task specific layers in conjunction with the hidden state of an LSTM model for supervised classification. For the choice of autoencoder we use a variational autoencoder (VAE) [37].

A VAE is a non-linear latent variable model based on variational principles. Given a set
4.2. Proposed Model

of user features $\mathcal{F}$, any latent variable model can be expressed as $p(f, z) = p(f|z)p(z)$, where $f \in \mathcal{F}$ and $z \in \mathbb{R}^K$. The prior of the latent variable is chosen to be a standard Gaussian distribution i.e $\mathcal{N}(0, 1)$. The conditional distribution $p(f|z)$ is implemented by a neural network which is also the decoder of the model. However, mapping from $f$ to $z$ is bit tricky as it requires us to compute $p(f) = \int p(f|z)p(z)dz$, which is intractable in general. A VAE uses a variational approximation $q(z|f)$ for this purpose where $q(z|f)$ is derived from a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$; here, $\mu$ and $\sigma$ is estimated through another neural network, i.e., the encoder. Now according to the variational Bayes formulation, the KL divergence between $p(z|f)$ and $q(z|f)$ should be minimized. It can be shown that minimizing this KL divergence is equivalent to maximizing the Evidence Lower Bound (ELBO)$=E[\log p(f|z)] - D_{KL}[q(z|f)||p(z)]$. Therefore the loss function of VAE can be cast as,

$$
O_v = -E[\log(p(f|z))] + D_{KL}[q(z|f)||p(z)]
$$ (4.1)

Here $-E[\log(p(f|z))]$ is the reconstruction loss or the negative log-likelihood. The second term represents the KL divergence between the encoders distribution $q(z|f)$ and the distribution of the latent variable $z$. It is necessary to for the model to allow for deterministic calculation of gradients w.r.t to the parameters of $q(z|f)$ i.e., $\mu$ and $\sigma$, for backpropagation. Thus it is common to re-parameterize the model to avoid stochastic sampling of $z$ by moving to a deterministic form — $z = \mu + \sigma \times \epsilon$ where $\epsilon \in \mathcal{N}(0, 1)$. Besides, for efficient information flow in the backpropagation, it is essential for all functions to be differentiable. The error cannot be easily backpropagated because of the stochastic sampling involved (sample from $q(z|f)$). Thus in VAE, it is common to reparameterize the sampling step to $z= u + sigma * random$ making it differentiable and allowing calculation of gradients w.r.t to $u$ and $sigma$.

We use the following user features in our model: number of friends, followers, statuses, public lists the user belong to and boolean indicators that represents whether the profile is verified, whether it has a URL and if the profile has a textual description. The model architecture is shown in Fig. 4.2.

Rumor Specific LSTM Layer: Our rumor detection layer is an LSTM network which takes a sequence of user features computed from VAE as input. We use the same LSTM framework developed in Problem 2. Recall that a conversational thread $T$ is a sequence of user interaction. Suppose $n$ refers to the number of users interacted in the thread. First we compute the user representation from VAE. Now the user embeddings are fed to the LSTM model.

Since the rumor veracity is detected for each conversational thread, we can treat it as a sequence classification problem. For the classification of rumor veracity we take the last hidden state of the LSTM layer $h_n^R \in \mathbb{R}^K$ as the feature. Then we apply a fully connected softmax layer to obtain the rumor class probabilities.
Chapter 4. Leveraging User Profile Information Into the Cascade Prediction Model to Detect the Veracity of Rumors

Figure 4.2: Model architecture of RumorSleuth. This model exactly resembles the AdaLSTM from problem 2 with one difference. Here we have a variational autoencoder to obtain the latent user representation of users and this representation is feed to the LSTM network.
where \( \odot \) is the concatenation operator, \( V_{rl} \in \mathbb{R}^{L \times K} \), \( b_{rl} \in \mathbb{R}^{L} \), \( L \) denotes the set of rumor labels—rumor, non-rumor, unverified. The reason behind using \( z_0 \) is from our observation in Fig. 4.1.

Given a set of training instances the objective for rumor veracity is

\[
\mathcal{O}_r = \min_{\theta_a, \theta_r} \sum_{i=1}^{T} \left[ -\log(p_{t=R_i|\tilde{h}_n^R}) \right] 
\]

(4.2)

Here \( \theta_a \) and \( \theta_r \) are the parameters from the shared layer and rumor specific layer receptively.

The final objective function is the summation of all three objective functions, i.e.,

\[
\mathcal{O}(\Theta) = \mathcal{O}_r + \mathcal{O}_v 
\]

(4.3)

where \( \Theta \) is the set of all the model parameters. We use adam optimizer to minimize the objective function. A pseudocode description for RumorSleuth is shown in Algorithm 1.

**Space Complexity:** Let us assume the latent user feature size and the original user feature size are \( F \) and \( K \) respectively. Then the size of the trainable variable of VAE is \( 3HK + 2FH \), where \( H \) is the size of the intermediate representation. Now assume the state size of the LSTM is also \( K \). As a result, the size of the LSTM becomes \( 4K^2 + K^2 = 5K^2 \). Finally, the matrix size of the attention mechanism and the output layer are \( 2K_{\alpha} + KK_{\alpha} \) and \( KL \) respectively, where \( L \) and \( K_{\alpha} \) are the number of the class labels and the attention size. So the total size of the trainable variable is \( 3HK + 2FH + 5K^2 + 2K_{\alpha} + KK_{\alpha} + KL \).

### 4.3 Experiments

#### 4.3.1 Experimental Setup

**Datasets and baselines:** A brief description of the dataset and the baseline models can be found in Section 3.4. We also create a model by combining user trust embedding and latent user representation and pass it to our LSTM framework. We name it AdaLSTM + RumorSleuth.
Chapter 4. Leveraging User Profile Information Into the Cascade Prediction Model to Detect the Veracity of Rumors

Algorithm 1: RumorSleuth

Input: A set of user interactions in conversational threads $\mathcal{T}$, user features $\mathcal{F}$, learning rate $\gamma$, max number of iterations $N$.

1. Initialize the model parameters $\Theta$ randomly

2. for $iter \leftarrow 1$ to $N$ do
3. \hspace{1em} optimize Eqn. 4.1 to get user embeddings;
4. end

5. for $iter \leftarrow 1$ to $N$ do
6. \hspace{1em} convert the data into batches
7. \hspace{2em} foreach batch in training data do
8. \hspace{3em} get the latent user vector $z$ from the VAE
9. \hspace{3em} feed the user embeddings to the LSTM model to obtain $\tilde{h}^R_j$
10. \hspace{3em} compute the gradient $\nabla(\Theta)$ using backpropagation
11. \hspace{3em} update the parameters using $\Theta \leftarrow \Theta - \gamma \nabla(\Theta)$
12. end

end

Parameter Settings: For RumorSleuth we set the latent size of the VAE to 8 and the encoder and decoder size to 16. We made a 80-10-10 split of the dataset for training, testing, and validation. We also set the maximum number of epochs to 50. We compare against the same set of methods mentioned in Problem 2.

4.3.2 How Does RumorSleuth Perform Against Other Models in Detecting Rumor Veracity

To answer this question we demonstrate the performance of each model in terms of micro and macro F1 score in Table 4.1. We can see that for both datasets AdaLSTM slightly outperforms the RumorSleuth in the PHEME dataset. However, RumorSleuth outperforms other methods in terms of micro-F1 score in the PHEME dataset. It also achieves decent performance in the RDTW dataset. RumorSleuth outperforms models like node2vec and logistic regression. Recall that logistic regression uses trust embedding. This shows that user generated features need to be exploited in a deep learning framework like AdaLSTM to get full benefit out of it. Finally, the combined version of trust embedding and user profile information AdaLSTM+RumorSleuth outperforms all the models as it exploits both types of user features.
Table 4.1: Comparison of performance between RumorSleuth, AdaLSTM and other methods. Proposed method is shown in boldface.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Methods</th>
<th>Micro F1</th>
<th>Macro F1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PHEME</td>
<td>Castillo</td>
<td>46.091</td>
<td>37.538</td>
</tr>
<tr>
<td></td>
<td><em>LogReg</em></td>
<td>50.382</td>
<td>38.426</td>
</tr>
<tr>
<td></td>
<td>node2vec</td>
<td>47.111</td>
<td>36.478</td>
</tr>
<tr>
<td></td>
<td>LSTM</td>
<td>43.333</td>
<td>32.024</td>
</tr>
<tr>
<td></td>
<td>GRU</td>
<td>46.667</td>
<td>36.809</td>
</tr>
<tr>
<td></td>
<td>RuRNN</td>
<td>48.148</td>
<td>35.263</td>
</tr>
<tr>
<td></td>
<td><em>AdaLSTM</em></td>
<td>52.667</td>
<td>42.849</td>
</tr>
<tr>
<td></td>
<td><em>RumorSleuth</em></td>
<td>51.428</td>
<td>32.505</td>
</tr>
<tr>
<td></td>
<td><em>AdaLSTM + RumorSleuth</em></td>
<td>53.330</td>
<td>46.560</td>
</tr>
<tr>
<td>RDTW</td>
<td>Castillo</td>
<td>60.671</td>
<td>52.487</td>
</tr>
<tr>
<td></td>
<td><em>LogReg</em></td>
<td>54.981</td>
<td>48.582</td>
</tr>
<tr>
<td></td>
<td>node2vec</td>
<td>46.707</td>
<td>52.173</td>
</tr>
<tr>
<td></td>
<td>LSTM</td>
<td>57.120</td>
<td>38.095</td>
</tr>
<tr>
<td></td>
<td>GRU</td>
<td>56.818</td>
<td>42.033</td>
</tr>
<tr>
<td></td>
<td>RuRNN</td>
<td>65.730</td>
<td>63.196</td>
</tr>
<tr>
<td></td>
<td><em>AdaLSTM</em></td>
<td>68.619</td>
<td>63.229</td>
</tr>
<tr>
<td></td>
<td><em>RumorSleuth</em></td>
<td>60.010</td>
<td>58.300</td>
</tr>
<tr>
<td></td>
<td><em>AdaLSTM + RumorSleuth</em></td>
<td>71.429</td>
<td>67.033</td>
</tr>
</tbody>
</table>

4.3.3 How Distinct Are the Latent User Vectors?

We show the 2D projection of the vector with the predicted rumor class in Fig. 4.3. We observe that even in a 2D projection the false and true classes are quite distinguishable. However, unverified rumors are spread around both the classes. Our hypothesis is that unverified rumors share characteristics of both false and true rumors w.r.t the latent user vectors.

4.3.4 How Early RumorSleuth Can Detect a Rumor?

We vary the percentage (%) thread observed and train the model on the observed threads. The micro F1 score is shown in Fig. 4.4 (a) and (b). We can see both AdaLSTM and RumorSleuth outshine the other methods by a good margin. We also observe the performance improvement for all methods as more portion of the thread is available.
Chapter 4. Leveraging User Profile Information Into the Cascade Prediction Model to Detect the Veracity of Rumors

Figure 4.3: 2D projection of latent vector of the users who initiates a rumor. We can see that false and true rumor are quite separable while unverified rumors appear close to both types.

Figure 4.4: How early the models are able to classify a rumor in (a) PHEME and (b) RDTW dataset.
4.4 Discussion

In the thesis, we evaluate two approaches of rumor veracity detection based on generating features from user interactions in social media. The first approach relies on structural information of users whereas the second one uses user profile information. In head to head comparison, we can see that the structural feature-based model is slightly better than the user profile based model. However, one of the pitfalls of this model is that computing user features from structural information is expensive. Moreover, structural information is difficult to obtain. On the other hand, features from the user profile are easy to obtain and can be fed to the classification model readily.
Chapter 5

Leveraging Recommender System to Recommend Fact-checking URLs to Rumor Debunkers

5.1 Introduction

Rumor mitigation is one of the most vital topics of rumor research. Effective rumor mitigation can make people aware of the true news and alleviate confusion throughout the social media space. However, it is not always easy to find the correct information and spread it to the people. Fortunately, there are a lot of fact-checking websites working tirelessly to debunk the rumor. Spreading the fact-checking URL will expedite the rumor mitigation process. There are rumor-debunkers in social media who follow the rumor stories and post the fact-checking information from these sites as soon as they are available. As a result, recommending relevant fact-checking URLs to the rumor debunkers will quicken the mitigation process. In this problem we propose a problem of fact-checking URL recommendation and propose a deep-learning model named NActSeer to solve it. We evaluate our model against several sequential recommendation models and our model outperforms the baseline model.

5.2 Introducing NActSeer

Consider an undirected social network $G = (V, E)$ where $V$ is the set of users/debunkers$^1$ and $E = V \times V$ represents the connections between them. Suppose, in the network a debunker $v \in V$ can share a fixed set of URLs $A^2$. Debunker can share the same URL many times. An URL sharing performed by a debunker $v \in V$ can be represented as a tuple $c_i^v = (a_i^v, s_i^v)$, where $a_i^v \in A$ and $s_i^v$ represents the timestamp. $l$ is index of this URL i.e. debunker $v$’s $l^{th}$ sharing. The timestamp represents the actual time when debunker performs the sharing. Let us assume the entire period of all the user history is $L$. Now for each debunker we can obtain an sharing history $p_v$ for each $v \in V$. Now suppose all the debunker’ history is stored in $H$.

---

$^1$We use terms user and rumor-debunkers interchangeably.

$^2$We use terms actions and URL sharing interchangeably.
5.2. Introducing \textit{NActSeer}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure5.1.png}
\caption{An example of rumor debunking post on Twitter}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure5.2.png}
\caption{The overall architecture of \textit{NActSeer}. First, the timestamp of the entire share history of all the users is discretized into time spans. Then we apply log-bilinear model to obtain user embedding for each time span. The individual user action history along with the user embedding is then fed to the LSTM part of the \textit{NActSeer} model, where user/debunker embedding is aggregated given the social network \( \mathcal{G} \) (shown in dashed line). At the end of the sequence the hidden state of the LSTM is passed through a softmax layer for next action prediction.}
\end{figure}

\textbf{Problem Statement:} Given a static social network \( \mathcal{G} \) and all the debunker' URL sharing history the goal of the problem is to predict the next URL sharing \( v \) is going to perform. Formally given \( p_v, v \in \mathcal{V} \), where \( p_v = \{c_1^v, c_2^v, \ldots, c_l^v\} \) our objective is to predict the next URL \( a_{l+1}^v \).

\subsection{5.2.1 Modeling User History}

The architecture of the model is shown in Fig. 5.2. Here we describe each part of the model.

\textbf{Representing URLs:} Inspired from the recent success in representation learning we represent each URL by a fixed length vector. Suppose the embedding of all the URLs \( \mathcal{A} \) are stored in a matrix \( \mathbf{X} \in \mathbb{R}^{|\mathcal{A}| \times D_a} \), where \( D_a \) is the size of embedding for URLs.

\textbf{Discretizing the Timestamp of URL sharing:} To integrate neighbor’s previous shar-
Chapter 5. Leveraging Recommender System to Recommend Fact-checking URLs to Rumor Debunkers

Table 5.1: Important notations used in the paper

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{G}$</td>
<td>The network between the users/debunkers</td>
</tr>
<tr>
<td>$\mathcal{V}$</td>
<td>The set of debunkers</td>
</tr>
<tr>
<td>$\mathcal{A}$</td>
<td>Set of all the sharing a debunker can perform</td>
</tr>
<tr>
<td>$\mathcal{H}$</td>
<td>The entire sharing history of all the debunkers</td>
</tr>
<tr>
<td>$p_v$</td>
<td>The original sharing history for debunker $v$</td>
</tr>
<tr>
<td>$q_v$</td>
<td>The time span tagged debunker history for debunker $v$</td>
</tr>
<tr>
<td>$X \in \mathbb{R}^{</td>
<td>A</td>
</tr>
<tr>
<td>$U \in \mathbb{R}^{</td>
<td>V</td>
</tr>
<tr>
<td>$U_t \in \mathbb{R}^{</td>
<td>V</td>
</tr>
<tr>
<td>$U_{v,t} \in \mathbb{R}^{</td>
<td>D_a</td>
</tr>
<tr>
<td>$\bar{U} \in \mathbb{R}^{</td>
<td>V</td>
</tr>
<tr>
<td>$\bar{U}_t \in \mathbb{R}^{</td>
<td>V</td>
</tr>
<tr>
<td>$D$</td>
<td>Diagonal degree matrix</td>
</tr>
<tr>
<td>$W$</td>
<td>Adjacency weight matrix</td>
</tr>
<tr>
<td>$L$</td>
<td>The entire period of debunker history</td>
</tr>
<tr>
<td>$T$</td>
<td>The number of time spans</td>
</tr>
<tr>
<td>$P$</td>
<td>The number of diffusion steps</td>
</tr>
<tr>
<td>$K$</td>
<td>Debunker history size for each time span</td>
</tr>
<tr>
<td>$D_a$, $D_m$</td>
<td>URL embedding size and state size</td>
</tr>
</tbody>
</table>

...ings into NActSeer we need a way to summarize the URL sharing for each user. To accomplish this for every sharing for a debunker, we have to compute the summary of her neighbors’ recent URL sharing. However, since social network usually contains huge number of users with lots of activities this would impose a huge computational overhead. To resolve this we divide the entire period of sharing into a fixed number of time spans. For example if the entire time period (i.e. our observation window) ($L$) ranges over multiple months then we can divide it by days. Suppose we divide the entire time period $L$ into $T$ fixed time spans. Recall that user’s entire sharing history is $p_v = \{c_{v1}, c_{v2}, \ldots, c_{vl}\}$, where $c_{vl} = \langle a_{vl}, s_{vl} \rangle$. Now if we divide the history by time span then it becomes $q_v = \{\langle a_{v1}^t, t_{v1}^t \rangle, \langle a_{v2}^t, t_{v2}^t \rangle, \ldots, \langle a_{vl}^t, t_{vl}^t \rangle\}$. Here $t_{vl}^t$ represents the discrete time span in which $s_{vl}$ falls i.e. $t_{vl-1}^t < s_{vl}^t \leq t_{vl}^t$, $t_{vl}^t \in \mathbb{N}$.

**Representing Users by Their Sharing:** After tagging each user’s sharing with a discrete time span we can compute the user embedding for each time span using the URL embedding matrix $X$. We propose in our case user embedding should be computed for each time span. The intuition behind this is rumor debunking is very dynamic in nature. Usually people shares the debunking about a particular topic (e.g. election, natural disaster) then move to the next. So the embedding of debunker keeps changing. Now suppose the user embedding
5.2. Introducing NActSeer

Matrix is $\mathbf{U} \in \mathbb{R}^{|\mathcal{V}| \times T \times |\mathcal{D}_a|}$. To compute the user embedding $\mathbf{U}_{v,t}$ for $v \in \mathcal{V}$ at time span $t$ we use the URLs she shares on time span $t$. However, within a time span a user can share a lot of URLs, this is especially true for active debunkers. Therefore we consider $K$ most recent sharings for a time span $t$ i.e. $\langle c_{t,n_t-K+1}^v, c_{t,n_t-K+2}^v, \ldots, c_{t,n_t}^v \rangle$, where $n_t$ is the number of sharings within the time span $t$. On the other hand, if the user performs less than $K$ sharings in a time span we can pad it with sharings from the previous time span. Here $K$ is a parameter to be set manually. Given this sequence of sharings we can employ any sequence model to compute the user embedding. However, a complex model would require more computational power. Therefore we use the log-bilinear model (LBL) [40, 54] to accomplish this. LBL is a feed forward neural network model with one hidden layer. Now given the $K$ most recent sharings of a user $v$ at time span $t$ the user embedding $\mathbf{U}_{v,t}$ can be computed using the following equation:

$$\mathbf{U}_{v,t} = \sum_{k=1}^{K} Y_k e_{t,k}$$

(5.1)

where $Y \in \mathbb{R}^{K \times |\mathcal{D}_a| \times |\mathcal{D}_a|}$ is a trainable parameter of the model and $e_{t,k}^v$ is the embedding of $k^{th}$ URL in time span $t$ for user $v$. In this way given the entire user history $\mathcal{H}$ of all the users we can compute the user embedding for every time span. For a given time span $t$, $\mathbf{U}_{v,t}$ summarizes user $v$’s activity during $t$.

Leveraging URL embedding to compute the user/debunker embedding has several benefits. Since the embedding of a user will be used in predicting her neighbors next sharing, computing user embedding this way is more beneficial as it directly comes from the URL embedding matrix $\mathbf{X}$. Static user features (e.g profile information) are less valuable here as they are hardly correlated with user’s next URL preference. Moreover this type of information may not be always available. However, if static feature for each user is also available they can also be incorporated here by concatenation. Lastly, in using such mechanism no retraining is required to obtain embedding for a new debunker.

5.2.2 Aggregating Neighbor’s Preferences

After computing the user embedding for each time span we can now aggregate it with the neighbor’s sharing preference. Recently several Convolutional Neural Network (CNN) models have been proposed which aggregate neighbor’s feature to generate representation for each node [6, 39, 48]. We can leverage similar architectures to compute an aggregated representation of the neighbor’s recent sharing. Suppose $A(\mathcal{G})$ captures the network structure of graph $\mathcal{G}$. Now given all users’ embedding $\mathbf{U}_t \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{D}_a|}$ on a time span $t$, the network aggregated user representation $\hat{\mathbf{U}}_t \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{D}_a|}$ for the $P^{th}$ order neighborhood can be computed
as follows:

\[
\tilde{U}_t^P = \text{NetAgg}(U_t) = \left[ \sum_{i=0}^{P-1} [A(G)]^i U_t \right]
\]

\[
\tilde{U}_t = [\tilde{U}_t^1; \ldots; \tilde{U}_t^P] W_n + b_w
\]

Here, \( W_n \in \mathbb{R}^{P \times D_a \times D_a} \) and \( b_w \in \mathbb{R}^{D_a} \) are the model parameters. Specifically \( \sum_{p=0}^{P-1} [A(G)]^p U_t \) results in a matrix of user embedding for the \( P \)th diffusion step. We repeat this process for each diffusion step to obtain \( P \) such user embeddings (i.e. each represents \( p^{th} \) order neighborhood where \( 0 \leq p < P \)). Then we concatenate them and pass it to a fully-connected network to obtain \( \tilde{U}_t \). This way we can aggregate not just immediate neighbors but higher-order neighborhoods as well. Now for the choice of \( A(G) \) we use normalized graph Laplacian matrix. Given \( D \) the diagonal degree matrix and \( W \) the adjacency weight matrix of \( G \) the normalized graph Laplacian matrix is

\[
A(G) = D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}
\]

### 5.2.3 Complete Architecture of \textit{NActSeer}

The input to \textit{NActSeer} is the time span tagged URL sharing history \( q_v = \{\langle a_v^1, t_v^1 \rangle, \ldots, \langle a_v^l, t_v^l \rangle\} \). This is passed to the embedding layer to convert it to a fixed length vector input.

**Embedding Layer:** Given \( q_v = \{c_v^1, c_v^2, \ldots, c_v^l\} \), where \( c_v^p = \langle a_v^p, t_v^p \rangle \) we represent each URL \( a_v^p \in A \) in the sequence as a vector. Recall that embedding of all the actions are stored in a matrix \( X \). To extract the embedding \( a_q \) of a specific action/URL sharing we can use the corresponding one-hot vector \( q \) of size \( |A| \). Now given \( q_v \) we can obtain the URL embedding sequence \( x_v = \{x_{v1}, x_{v2}, \ldots, x_{vl}\} \) where \( x_i \in \mathbb{R}^{D_a} \). Additionally we can gather the corresponding user embedding for each \( t_v^i, \forall i \in l \). As a result, we obtain the embedding sequence for a user, \( r_v = \{U_{t_1}, U_{t_2}, \ldots, U_{t_l}\} \) using Eqn. 5.1. It is to be noted that for each single user \( v \) the corresponding embedding sequence consists of the embedding of all other users i.e. \( \mathbb{R}^{|V| \times D_a} \) at each corresponding time span (as it is required for \textit{NetAgg} in Eqn 5.2). To avoid confusion between \( U_t \) and \( U_{v,t} \) (which only refers to a single user’s embedding i.e. \( \mathbb{R}^{D_a} \)) we use the notation \( U_{t_v^i} \) when discussing about the user sequence \( r_v \). So from here on \( r_v = \{U_{t_v^1}, U_{t_v^2}, \ldots, U_{t_v^l}\} \), where \( U_{t_v^i} \in \mathbb{R}^{|V| \times D_a} \). Finally, both the user sequence and the URL sequence are concatenated and given as input to the \textit{NActSeer} model.

**Layer Normalization:** We adopt the layer normalization [1] technique to normalize the inputs across features i.e. zero mean and unit variance to help improve stability and accelerate the training process. Suppose the input vector is \( x \) then layer normalization is defined as
5.2. Introducing NActSeer

LayerNorm ($\text{x}$) = $\alpha \odot \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta$

Here $\mu$ and the $\sigma$ are the mean and variance of $\text{x}$ and $\alpha$ and $\beta$ are learned parameters (i.e. scaling factors and bias). We apply the layer normalization on the concatenated input sequences.

**NActSeer Operation:** NActSeer takes layer-normed URL sharing embedding $\text{x}_v$ and user embedding $r_v$. The NActSeer equations are:

\[
\tilde{U} = NetAgg(U_{v_j})
\]

\[
\tilde{x}_{v_j} = x_{v_j} \oplus \tilde{U}_{v,t_j}
\]

\[
i_j = \sigma(W^i \tilde{x}_{v_j} + Q^i h_{j-1} + b_i)
\]

\[
f_j = \sigma(W^f \tilde{x}_{v_j} + Q^f h_{j-1} + b_f)
\]

\[
\tilde{C}_j = \tanh(W^c \tilde{x}_{v_j} + Q^c h_{j-1} + b_c)
\]

\[
C_j = \tilde{C}_j \odot i_j + f_j \odot C_{j-1}
\]

\[
o_j = \sigma(W^o \tilde{x}_{v_j} + Q^o h_{j-1} + b_o)
\]

\[
h_j = o_j \odot \tanh(C_j)
\]

Here $W^* \in \mathbb{R}^{D_m \times D_a}$ and $Q^* \in \mathbb{R}^{D_m \times D_m}$ are the parameters of different gates within the RNN cell of NActSeer and $\oplus$ represents element-wise addition.

**Next URL Prediction:** The score of an URL to be taken next can be computed by the following equation: $w_{a_{j+1}} = V_a h_j + b_a$. Here $V_a \in \mathbb{R}^{|A| \times D_m}$ and $b_a \in \mathbb{R}^{|A|}$. Now we can use a softmax layer to compute the probability of each URL $a$ being adopted.

\[
p_{\tilde{a}_{j+1} | h_j} = \frac{\exp(w_{a_{j+1}})}{\sum_{z \in A} \exp(w_{z_{j+1}})}
\]

The final objective function the becomes

\[
\mathcal{O} = \arg\min_{\theta} \left( -\sum_{v \in V} \sum_{j=1}^{l} \log \left( p_{\tilde{a}_{j+1} = a_{j+1} | h_j} \right) \right)
\]

where $\theta$ is the set of all the model parameters.

### 5.2.4 Expediting User Embedding Computation

From Eqn. 5.3 we see that at each step of our recurrent network we have to compute Eqn. 5.2. This imposes a huge computational burden on the overall training process. So, we modify
Chapter 5. Leveraging Recommender System to Recommend Fact-checking URLs to Rumor Debunkers

Algorithm 2: Computing User Embedding for a mini-batch

Input: a mini-batch of size \( B \) of time span tagged user activity \( q_v \) for \( v \in \mathcal{V} \)

Output: Network aggregated user embedding matrix \( \tilde{\mathbf{U}}_t \) for the mini-batch

for \( t \leftarrow 1 \) to \( T \) do
    foreach \( v \in \mathcal{V} \) do
        \( \mathbf{U}_{v,t} = \sum_{k=1}^{K} Y_{k} c_{t,k}^{v} \)
    end
end

for \( t \leftarrow 1 \) to \( T \) do
    calculate \( \tilde{\mathbf{U}}_t \) from Eqn. 5.2
end

\( \text{BatchUserEmbedding} \leftarrow \text{list()} \)

for \( b \leftarrow 1 \) to \( B \) do
    for \( j \leftarrow 1 \) to \( l \) do
        extract user embedding \( \tilde{\mathbf{U}}_{v,t_{j}} \) from \( \tilde{\mathbf{U}}_t \) given time span \( t_{j}^{v} \)
        add \( \tilde{\mathbf{U}}_{v,t_{j}} \) to \( \text{BatchUserEmbedding} \)
    end
end

return \( \text{BatchUserEmbedding} \in \mathbb{R}^{B \times l \times D_a} \)

the computational flow to pre-compute \( \tilde{\mathbf{U}} \). For a single mini-batch we compute the user embedding matrix \( \mathbf{U} \) only one time then compute Eqn. 5.2 for every time span \( t \) to obtain \( \tilde{\mathbf{U}} \). We then extract the network aggregated user embedding for corresponding time span as and when necessary. A pseudocode is shown in Algorithm 2. The output of Algorithm 2 is a matrix of network aggregated user embedding \( \text{BatchUserEmbedding} \in \mathbb{R}^{B \times l \times D_a} \) for a mini-batch of size \( B \) and sequence length \( l \). Now in the \( NActSeer \) cell we do not need to compute Eqn. 5.3(a). We can just add the embedding (i.e. Eqn. 5.3(b)) and then execute Eqn. 5.3(c–h). This removes the burden of computing Eqn. 5.2 too many times and expedite training process.

5.2.5 Using Pre-trained Embedding

We obtain the embedding of URLs by training Doc2Vec [43] on the headline of each fact checking claim. At the start of the training of \( NActSeer \) we initialize the feature vector of URLs by the embedding from Doc2Vec. We call this model \( NActSeerDV \).
5.3 Experiments

5.3.1 Experimental Setup

**Dataset:** We use the fact-checking sharing dataset published in [68]. For our evaluation purpose. This dataset contains the sharing history of 4834 URLs. The total number of debunkers/users is 12197.

**Baseline Methods:** We compare our model with state-of-the-art methods proposed for related problems. In order to have a thorough evaluation, we compare it against methods that can model sequential input and also methods that can make use of network structure.

- **LSTM** is a widely popular RNN model for sequence modeling. For this model we use the user’s previous action sequence to predict the next action.

- **GCN** [39] is a CNN based model to learn feature representation for nodes in graph. We use a binary vector of size $|A|$, wherein actions performed by a user in the past take the value of 1.

- **Caser** [67] is a recently proposed CNN-based method for sequential recommendation. It captures the sequential nature of actions by applying convolutional operations on the embedding matrix of the most recently accessed items.

- **SASRec** [36] is a state-of-the-art model for sequential recommendation. It uses positional embedding and multi-head attention mechanism to detect the most relevant items.

- **NActSeer** is our proposed model which combines the GCN method in a LSTM model to integrate other neighbor’s preferences.

- **NActSeerDV** is a version of our proposed model which uses pre-trained embedding from Doc2Vec.

**Parameter Settings:** Unless otherwise specified we set the state size of the model $D_m = 64$, user/action embedding size $D_a = 64$, context size $K = 1$, dropout prob. $\Delta = 0.3$. We run 100 iterations of each model and report the best result. We set the value of $T$ to 12 empirically.

**Evaluation Metric:** Given a user’s action history, acquiring the next action can be treated as a retrieval task since an arbitrarily large number of actions can be selected. Therefore an intuitive way for evaluation is to apply ranking metrics used in information retrieval. For this to work we rank all the actions by their probabilities and consider the relevant action to be the actual actions taken by the users. We use two widely popular ranking methods:

- **map@K**: This represents the Mean Average Precision used in information retrieval.
Chapter 5. Leveraging Recommender System to Recommend Fact-checking URLs to Rumor Debunkers

5.3.2 How Effective Is NActSeer in Predicting the Next URL Share?

We show the performance of different methods in Fig. 5.3. We make the following observations:

1. NActSeer outperforms all the methods by a very good margin. Performance gain ranges from around 25–31% in terms of hits@20 across different methods. On the other hand for map@20 it is around 17–51%. For lower values of $\kappa$ sometimes the gain is even higher. This demonstrates that NActSeer fruitfully combines the user’s previous action and her neighbors’ actions to predict effectively.

2. Though LSTM, SASRec and Caser can model the sequential nature of the input they all suffer from the network aggregation problem. Therefore their performance hurts.

Figure 5.3: Performance comparison of NActSeer against other baseline models. NActSeer is able to outperform other model by a very good margin.

- **hits@$\kappa$**: The rate of top $\kappa$ ranked actions containing the actual next action taken by the user.
- **ndcg@$\kappa$**: is a position-aware metric which assigns larger weights on higher positions.
5.3. Experiments

3. GCN is capable of aggregating user preferences, however it cannot model the sequential data. As a result, it cannot outperform NActSeer.

4. We observe that performance improves as $\kappa$ increases. This is expected since the target URL is more likely to be included if more candidate URLs are considered. Moreover, map@$\kappa$ scores are relatively lower as they also consider the position of the true action among the candidate actions.

5.3.3 How Useful Is the Expedited User Embedding Computation?

Recall that in Algorithm 2 we show a faster way of computing the user embedding for a mini-batch. Now we evaluate how does it expedite the training process. The comparison between the execution time of normal and expedited computation on several generic datasets is shown in Fig 5.4. We observe that the expedited computation method achieves up to 8.5% speed-up for each training iteration.
Chapter 5. Leveraging Recommender System to Recommend Fact-checking URLs to Rumor Debunkers

Figure 5.5: Normalized count of correct predictions vs number of neighbors/friends. We observe an increasing trend of performance as the number of friends increases showing that including neighbors actions improves performance. However, the trend has an diminishing return.

5.3.4 Does the Number of Friends Improve the Prediction?

To see whether the number of friends has an effect on overall performance we plot the number of correct predictions i.e. the number of users/debunkers for whom a correct prediction has been made vs number of neighbors. We normalize the correct prediction count by dividing the total number of users that has the corresponding number of neighbors. The result is shown in Fig. 5.5. We can see that having more neighbors improves the overall performance of the users. This shows NActSeer successfully leverage the neighbors activity to achieve better performance. However, this trend has a diminishing return in the end.

5.4 Discussion

In this paper, we present an intriguing problem recommending fact-checking URLs to mitigate rumors. To solve this problem we develop a network augmented RNN model NActSeer which can take debunkers’ neighborhood into account to predict his URL share. We compare our model against several state-of-the-art methods on a real-world dataset. NActSeer outperforms all the other models. One important future work can be to extend the model to include features from URLs. Another interesting concept is to extend the NActSeer model so that it can handle additional input signals pertinent to the rumor itself. Finally predicting how debunkers’ sharing leads to a change in the rumor propagation can be an interesting research direction.
Chapter 6

Conclusion and Future Work

Given the increasing importance of social media in many sectors, the truthfulness of rumors on social platforms should be verified. In this thesis, we worked on two facets of rumor research namely veracity detection and mitigation. For veracity detection we outlined several machine learning models that detect the truthfulness of a rumor. First, we found a suitable representation for users on social networks. Then, we leveraged the resulting representation to identify rumor veracity on social networks. We also proposed a method that can detect rumor veracity using user profile information. Our proposed approaches outperform several other methods of rumor veracity detection. Finally, we proposed a rumor mitigation technique that recommends relevant fact-checking URLs to rumor debunkers. Our model leveraged the internal social network among rumor debunkers to recommend pertinent URLs.

We believe that the synchronization of both types is necessary to combat rumors on social media. There are some practical considerations in applying the detection and mitigation models in real-world applications.

• **Class imbalance:** The number of possible false rumors is very small compared with the total number of posts published on social network platforms. Thus, we need to reduce the search space to narrow our focus. This can be performed in multiple ways. First, we can dynamically track the most prominent topics on social media to monitor the spread of rumor stories. Moreover, we can also leverage event detection models to identify breaking news stories in social spaces. Many social network platforms provide popular hashtags and topics to showcase current trends. Our proposed models can benefit from these features to narrow the search space.

• **Scalability:** One of the challenges of implementing the models is to compute and update the user embedding efficiently. This is especially true for the AdaLSTM model, which requires the trust scores and embedding of users beforehand. We believe a batch processing system needs to be applied on top of both methods to periodically update the scores. Our signed network embedding method is capable of periodic retraining to allow new nodes and connections into the network. As a result, we can keep the user representation up-to-date by running it periodically. Finally, we need to retrain the AdaLSTM model from time to time. Since fully retraining a deep learning model is a challenge, we can retrain only on discussion threads where the corresponding user embedding has been updated or new users have participated in the discussion.

• **Human intervention and interpretability:** Human intervention is a vital part of
dealing with societal problems such as fake news and rumors. An effective way to assessing the performance of rumor veracity detection models is to curate a list of discussion threads and send them to human experts for evaluation. This way, we can take advantage of the human expertise to decide whether to retrain the models or to look for more informative features. Another intuitive way is to develop a model that can use the user comments and reactions to detect rumor veracity [65, 75].

**Future Research:** There are possible research avenues that can be explored next. In this thesis, we work on Twitter; however, there are other social networks such as Reddit and Facebook, where our methods can be applied with appropriate modifications. Additionally, online news articles can be taken into account for rumor veracity detection.

- **Identifying bots and propaganda news:** Studies have found that bots can be used for propagating fake news and propaganda [16, 63]. There are some papers on identifying bots on Twitter [11]. Future rumor detection methods can leverage bot detection methods for improvement.

- **Meeting the challenges of URL recommendation:** Recommending fact-checking URLs can be a useful technique of mitigating rumors. However, there are several challenges here that must be addressed. There is a time lag between a rumor claim being posted on a social network and publishing the verdict on this claim on the fact-checking websites. Hence, detecting suspicious social media posts early is vital. We can exploit features such as trending topics published on the social network sites to obtain rumor claims early. Additionally, our proposed rumor veracity detection models can also be useful here. Another important issue is the development of better digital libraries that will enable the fact-checkers to take advantage of the right materials to examine claims.

Fake news and rumors seriously hamper the user experience in social networks. It also negatively impact our life politically and economically. Proper methods of rumor identification and mitigation are required to deal with such phenomena. In this thesis, we show that using user profiles and interactions can yield meaningful user representation for rumor veracity detection. Future work should concentrate on resolving the above mentioned issues and other aspects of social media to design efficient methods of combating rumors.
Bibliography


