TIMME: Twitter Ideology-detection via Multi-task Multi-relational Embedding

Zhiping Xiao
CS Department, UCLA
Los Angeles, CA, United States
patricia.xiao@cs.ucla.edu

Weiping Song
CS Department, School of EECS, Peking University
Beijing, China
weiping.song@pku.edu.cn

Haoyan Xu
College of Control Science and Engineering, Zhejiang University
Hangzhou, China
3160104027@zju.edu.cn

Zhicheng Ren
CS Department, UCLA
Los Angeles, CA, United States
franklinnwren@g.ucla.edu

Yizhou Sun
CS Department, UCLA
Los Angeles, CA, United States
yzsun@cs.ucla.edu

ABSTRACT

We aim at solving the problem of predicting people's ideology, or political tendency. We estimate it by using Twitter data, and formalize it as a classification problem. Ideology-detection has long been a challenging yet important problem. Certain groups, such as the policy makers, rely on it to make wise decisions. Back in the old days when labor-intensive survey-studies were needed to collect public opinions, analyzing ordinary citizens' political tendencies was uneasy. The rise of social medias, such as Twitter, has enabled us to gather ordinary citizen's data easily. However, the incompleteness of the labels and the features in social network datasets is tricky, not to mention the enormous data size and the heterogeneousity. The data differ dramatically from many commonly-used datasets, thus brings unique challenges. In our work, first we built our own datasets from Twitter. Next, we proposed **TIMME**, a multi-task multi-relational embedding model, that works efficiently on sparsely-labeled heterogeneous real-world dataset. It could also handle the incompleteness of the input features. Experimental results showed that **TIMME** is overall better than the state-of-the-art models for ideology detection on Twitter. Our findings include: links can lead to good classification outcomes without text; conservative voice is under-represented on Twitter; *follow* is the most important relation to predict ideology; *retweet* and *mention* enhance a higher chance of *like*, etc. Last but not least, **TIMME** could be extended to other datasets and tasks in theory.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**.

KEYWORDS

multi-task learning, ideology detection, heterogeneous information network, social network analysis, graph convolutional networks

ACM Reference Format:
Zhiping Xiao, Weiping Song, Haoyan Xu, Zhicheng Ren, and Yizhou Sun. 2020. TIMME: Twitter Ideology-detection via Multi-task Multi-relational Embedding. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Studies on ideology never fails to attract people's interests. Ideology here refers to the political stance or tendency of people, often reflected as left- or right-leaning. Measuring the politicians' ideology helps predict some important decisions' final outcomes, but it doesn't provide more insights into ordinary citizens' views, which are also of decisive significance. Decades ago, social scientists have already started using probabilistic models to study the voting behaviors of the politicians. But seldom did they study the mass population's opinions, for the survey-based study is extremely labor-intensive and hard-to-scale [1, 26]. The booming development of social networks in the recent years shed light on detecting ordinary people's ideology. In social networks, people are more relaxed than in an offline interview, and behave naturally. Social networks, in return, has shaped people's habits, giving rise to opinion leaders, encouraging youngsters' political involvement [24].

Most existing approaches of ideology detection on social networks focus on text [5, 8, 14–16]. Most of their methodologies based on probabilistic models, following the long-lasting tradition started by social scientists. Some others [2, 12, 16, 28] noticed the advantages of neural networks, but seldom do they focus on links. We will show that the social-network links' contribution to ideology detection has been under-estimated.

An intuitive explanation of how links could be telling is illustrated in Figure 1. Different types of links come into being for different reasons. We have five relation types among users on Twitter today: *follow, retweet, reply, mention, like*, and the relations affect each other. For instance, after Rosa *retweet* from Derica and *mention* her, Derica *reply* to her; when Isabel *mention* some politicians in her posts, the politician's *follow*ers might come to interact with her. One might *mention* or *reply* to debate, but *like* always stands

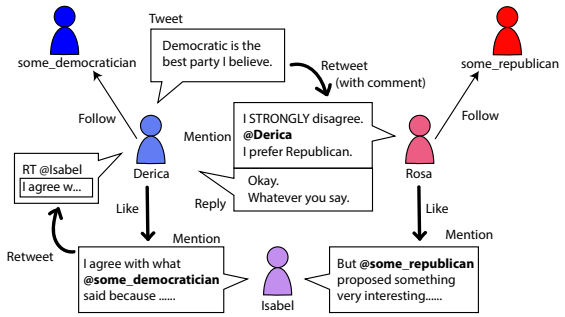


Figure 1: An example of different relation types on Twitter. Derica is on liberal (left) side while Rosa is on the conservative (right) side. Isabel does not have significant tendency.

for agreement. The relations could reflect some opinions that a user would never tell you verbally. Words could be easily disguised, and there is always a problem called "the silent majority", for most people are unwilling to express.

Yet there are some uniqueness of Twitter dataset, bringing about many challenges. It is especially the case when existing approaches are mostly dealing with smaller datasets with much sparser links than ours, such as academic graphs, text-word graphs, and knowledge-graphs. First, our Twitter dataset is large and the links are relatively dense (Section 4). Some models such as GraphSAGE [13] will be super slow sampling our graph. Second, labels are extremely sparse, less than 1%. Most approaches will suffer from severe over-fitting, and the lack of reliable evaluation. Third, features are always incomplete, for in real-life datasets like Twitter, many accounts are removed or blocked. Fourth, modeling the heterogeneousity is non-trivial. Many existing methods emerged from homogeneous networks tend to ignore the information brought by the types of links.

Existing works can not address the above challenges well. Even though some realized the importance of links [9, 12], they failed to provide an embedding. Most people learn an embedding by separating the heterogeneous graph into different homogeneous views entirely, and combine them in the very end.

We propose to solve the above-listed problems by **TIMME** (Twitter Ideology-detection via Multi-task Multi-relational Embedding), a model good at handling sparsely-labeled large graph, utilizing multiple relation types, and optionally dealing with missing features. The code with data is released on Github at https://github.com/PatriciaXiao/TIMME. Our major contributions are:

- We propose **TIMME** for ideology detection on Twitter, whose encoder captures the interactions between different relations, and decoder treats different relations separately while measuring the importance of each relation to ideology detection.
- The experimental results have proved that **TIMME** outperforms the state-of-the-art models. Case studies showed that conservative voice is typically under-represented on Twitter. There are also many findings on the relations' interactions.
- The large-scale dataset we crawled, cleaned, and labeled (Appendix A) is a good measurement of how well a model can handle the real-life problems. It will be very valuable to the study of heterogeneous neural networks.

In this paper, we will walk through the related work in Section 2, introduce the preliminaries and the definition of the problem we are working on in Section 3, followed by the details of the model we propose in Section 4, experimental results and discussions in Section 5, and Section 6 for conclusion.

2 RELATED WORK

2.1 Ideology Detection

Ideology detection in general could be naturally divided into two directions, based on the targets to predict: of the politicians [7, 23, 27], and of the ordinary citizens [1, 2, 5, 8, 12, 14–16, 19, 22, 28]. The work conducted on ordinary citizens could also be categorized into two types according to the source of data being used: intentionally collected via strategies like survey [1, 19], and directly collected such as from news articles [2] or from social networks [12, 14, 16]. Some studies take advantages from both sides, asking self-reported responses from a group of users selected from social networks [28], and some researchers admitted the limitations of survey experiments [22]. Emerging from social science, probabilistic models have been widely used for such kinds of analysis since no latter than the early 1980s [2, 12, 27]. On the other hand, on social network datasets, it is quite intuitive trying to extract information from text data to do ideology-detection [5, 8, 14–16], only a few paid attention to links [9, 12]. Our work differs from them all, since: (1) unlike probabilistic models, we use GNN approaches to solve this problem, so that we take advantage of the high-efficient computational resources, and we have the embeddings for further analysis; (2) we focus on **relations** among users, and proved how telling those relations could be.

2.2 Graph Neural Networks (GNN)

2.2.1 Graph Convolutional Networks (GCN). Inspired by the great success of convolutional neural networks (CNN), researchers have been seeking for its extension onto information networks [11, 18] to learn the entities' embeddings. The Graph Convolutional Networks (GCN) [18] could be regarded as an approximation of spectral-domain convolution of the graph signals. A deeper insight [20] shows that the key reason why GCN works so well on classification tasks is that its operation is a form of Laplacian smoothing, and concludes the potential over-smoothing problem, as well as emphasizes the harm of the lack of labels.

GCN convolutional operation could also be viewed as sampling and aggregating of the neighborhood information, such as GraphSAGE [13] and FastGCN [4], enabling training in batches while sacrificing some time-efficiency. To improve GraphSAGE's expressiveness, GIN [39] is developed, enabling more complex forms of aggregation. In practice, due to the sampling time cost brought by our links' high density, GIN, GraphSAGE and its extension onto heterogeneous information network such as HetGNN [42] and GATNE [3] are not very suitable on our datasets.

The relational-GCN (r-GCN) [31] extends GCN onto heterogeneous information networks. A very large number of relation-types R ends up in overwhelming parameters, thus they put some constraints on the weight matrices, referred to as weight-matrix decomposition. GEM [21] is almost a special case of r-GCN. Unfortunately, their code is kept confidential. According to the descriptions in their

paper, they have a component of similar use as α in our encoder, but it is treated as a free parameter.

Another way of dealing with multiple link types is well-represented by SHINE [37], who treats the heterogeneous types of links as separated homogeneous links, and combines embeddings from all relations in the end. SHINE's didn't make good use of the multiple relations to its full potential, modeling the relations without allowing complex interactions among them. GTN [41] is similar with SHINE in having different views to train separately and simply combining the output embeddings at the end of the encoder. Meta-path is used in GTN, similar with the treatment of multiple-relations of HAN [38], being potentially more expressive than SHINE, but would rely heavily on the amount of meta-paths being used.

2.2.2 Graph Attention Networks. Graph Attention Networks (GAT) [35] is another nontrivial direction to go under the topic of graph neural networks. It incorporates attention into propagation, attending over the neighbors via self-attention. Multi-head mechanism is used to ensure the stability.

An extension of GAT on heterogeneous information networks is proposed in HAN [38]. Beside inheriting the node-level attention from GAT, it considers different relation types by sampling its neighbors from different meta-paths. It first conducts type-specific transformation and compute the importance of neighbors of each node. After that, it aggregates the coefficients of all neighbor nodes to update the current node's representation. In addition, to obtain more comprehensive information, it conducts semantic-level attention, which takes the result of node-level attention as input and computes the importance of each meta-path. We use HAN as an important base line in our experiments.

2.3 Multi-Task Learning (MTL)

In multi-task learning (MTL) settings, there are multiple tasks sharing the same inductive bias jointly trained. Ideally, the performance of every task should benefit from leveraging auxiliary knowledge from each other. As is concluded in an overview [30], MTL could be applied with or without neural network structure. On neural network structure, the most common approach is to do hard parameter-sharing, where the tasks share some hidden layers. The most common way of optimizing a MTL problem is to solve it by joint-training fashion, with joint loss computed as a weighted combination of losses from different tasks [17]. It has a very wide range of application, such as the DMT-Demographic Models [36] where multiple aspects of Twitter data (e.g. text, images) are fed into different tasks and trained jointly. Aron and Nirmal et al. [10] also apply MTL on Twitter, separating the tasks by user categories. Our multi-task design differs from theirs, as we treat different relations' predictions as different tasks, together with a node-classification task, whose result serves as the ideology predicted.

3 PROBLEM DEFINITION

Our goal is to predict Twitter users' ideologies. We achieve this goal by learning an ideology embedding of each user in the political-centered social network. To start with, we define the political-centered social network.

Definition 3.1. (Heterogeneous Information Network) Following previous work [33], we say that an information network $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where number of vertices is $|\mathcal{V}| = N$, is a **heterogeneous information network**, when there's $|\mathcal{T}| = T$ types of vertices, $|\mathcal{R}| = R$ types of edges, and $\max(T, R) > 1$. $\mathcal{G}$ could be represented as $\mathcal{G} = \{\{\mathcal{V}_1, \mathcal{V}_2, \ldots \mathcal{V}_T\}, \{\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_R\}\}$

Each possible edge from the i^{th} node to the j^{th}, represented as $e_{ij} \in \mathcal{E}$ has a weight value $w_{ij} > 0$ associated to it, where $w_{ij} = 0$ representing $e_{ij} \notin \mathcal{E}$. In our case, $\mathcal{G}$ is a directed graph. In general, we have $\langle v_i, v_j \rangle \not\equiv \langle v_j, v_i \rangle$ and $w_{ij} \not\equiv w_{ji}$.

Twitter data $\mathcal{G}_{Twitter}$ contains $T = 1$ type of entities (users), and $R = 5$ different types of edges (relations) among the entities, namely *follow*, *retweet*, *like*, *mention*, *reply*.

$$\mathcal{G}_{Twitter} = \{\mathcal{V}, \{\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3, \mathcal{E}_4, \mathcal{E}_5\}\}$$

Detailed description about Twitter data is included in Appendix A, and we call the subgraph we selected from Twitter-network a *political-centered social network*, which is defined as follows:

Definition 3.2. (Political-Centered Social Network) The political-centered social network is a special case of directed heterogeneous information network. With a pre-defined politicians set $\mathcal{P}$, in our selected heterogeneous network $\mathcal{G}_{Twitter}$, $\forall e = \langle v_i, v_j \rangle \in \mathcal{E}_r$ where $r \in \{1, 2, \ldots, R\}$, there has to be either $v_i \in \mathcal{P}$ or $v_j \in \mathcal{P}$. All the politicians in this dataset have ground-truth labels indicating their political stance. The political-centered social networks are represented as $\mathcal{G}_p$.

We would like to leverage the information we have to learn the representation of the users, which could help us reveal their ideologies. Due to the lack of Independent representatives (only two in total), we consider the binary-set labels only: { *liberal*, *conservative* }. *Democratic* on liberal side, *Republican* for conservative.

Definition 3.3. (Multi-task Multi-relational Network Embedding) Given a network $\mathcal{G}_p = \{\mathcal{V}, \{\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3, \mathcal{E}_4, \mathcal{E}_5\}\}$ where the number of nodes is $|\mathcal{V}| = N$, the goal of TIMME is to learn such a representation $h_i \in \mathbb{R}^d$ where $d \ll N$ for $\forall v_i \in \mathcal{V}$, that captures the categorical information of nodes, such as their ideology tendencies. As a measurement, we want the representation $H \in \mathbb{R}^{N \times d}$, to success on both node-classification and link-prediction.

4 METHODOLOGY

The general architecture of our proposed model is illustrated in Figure 2. It contains two components: encoder and decoder. The encoder contains two multi-relational convolutional layers. The output of the encoder is passed on to the decoder, who handles the downstream tasks.

4.1 Multi-Relation Encoder

As mentioned before in Section 1, the challenges faced by the encoder part are the large data scale, the heterogeneous link types, and the missing features.

GCN is very effective in learning the nodes' embeddings, especially good at classification tasks. Meanwhile, it is also naturally efficient, in terms of handling the large amount of vertices N.

Random-walk-based approaches such as node2vec could suffer from a cover time of $O(N^3)$ in the worst case, in order to

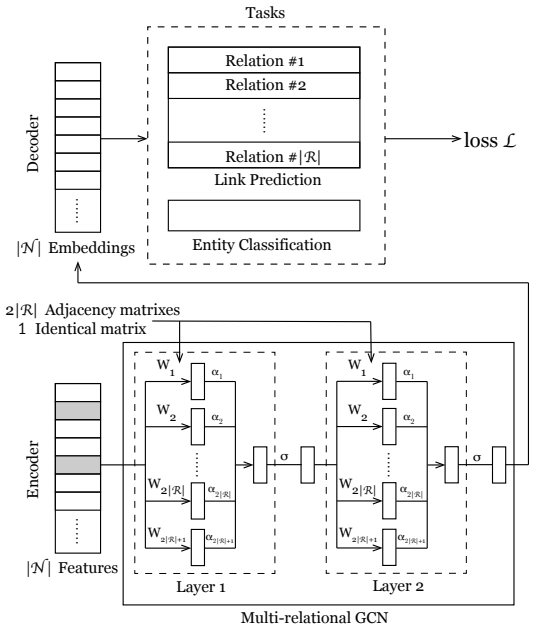


Figure 2: The general architecture of our model, with the encoder shown in details. Grey blocks represent missing features. Our model can either handle them by treating them as learnable parameters, or use one-hot features.

get adequate samples from the graph. On the other hand, GCN-based approaches are naturally efficient here. Like is analyzed in Cluster-GCN [6], the time complexity of the standard GCN model is $O(L\|A\|_0 F + LNF^2)$, where L is the number of layers, $\|A\|_0$ the number of non-zeros in the adjacency matrix, F the number of features. Realizing that the time complexity increase linearly when N increases. A GCN model's layer-wise propagation could be written as:

$$H^{(l+1)} = \sigma\Big(\hat{A}H^{(l)}W^{(l)}\Big)$$

$\hat{A} = \tilde{D}^{\frac{1}{2}}(A + I_N)\tilde{D}^{\frac{1}{2}}$, where $\tilde{D}$ is defined as the diagonal matrix and A the adjacency matrix. D_{ii}, the diagonal element d_i, is equal to the sum of all the edges attached to v_i; $H^{(l)} \in \mathbb{R}^{N\times d^{(l)}}$ is the $d^{(l)}$-dimensional representation of the N nodes at the l^{th} layer; $W^{(l)} \in \mathbb{R}^{d^{(l)}\times d^{(l+1)}}$ is the weight parameters at layer l which is similar with that of an ordinary MLP model [1]. In a certain way, $\hat{A}$ could be viewed as A after being normalized.

We propose to model the heterogeneous types of links and their interactions in the encoder. Otherwise, if we split the views like many others did, the model will never be expressive enough to capture the interactions among relations. For any given political-centered graph $\mathcal{G}_P$, let's denote the total number of nodes $|\mathcal{V}| = N$, the number of relations $|\mathcal{R}| = R$, the set of nodes $\mathcal{V}$, the set of relations $\mathcal{R}$, and $\mathcal{E}_r$ being the set of links under relation $r \in \mathcal{R}$. Representation being learned after layer l ($l \in \{1, 2\}$) is represented as $H^{(l)} \in \mathbb{R}^{N\times d^{(l)}}$, and the input features form the matrix $H^{(0)} \in \mathbb{R}^{N\times d^{(0)}}$. $\hat{\mathcal{R}}$ where $|\hat{\mathcal{R}}| = 2R+1$ represents all relations in the original direction (R), the relations in reversed direction (R), and an identical-matrix relation (1). Our dataset has $|\mathcal{R}| = R = 5$, so it should be fine not to conduct a weight-matrix decomposition like r-GCN [31]. We model the layer-wise propagation at Layer $l+1$ as:

$$H^{(l+1)} = \sigma\Big(\sum_{r\in\hat{\mathcal{R}}} \alpha_r \hat{A}_r H^{(l)} W_r^{(l)}\Big) = \sigma\Big(\sum_{r\in\hat{\mathcal{R}}} H_r^{(l+1)}\Big)$$

$H^{(l)} \in \mathbb{R}^{N\times d^{(l)}}$ is used to denote the representation of the nodes after the l^{th} encoder layer, $H_r^{(l+1)}$ is the intermediate embeddings, and the initial input feature is $H^{(0)}$. The activation function σ we use is ReLU. $\alpha \in \mathbb{R}^{2R+1}$ is calculated by scaled dot-product self-attention over the outputs of $\hat{A}_r H^{(l)} W_r^{(l)}$. By default, we apply a scaled dot-product self-attention in each layer, computed by:

$$A = Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d}})V \in \mathbb{R}^{(2R+1)\times d}$$

where $Q = K = V \in \mathbb{R}^{(2R+1)\times d}$ comes from the output of an encoder layer, calculated as the out embedding $O \in \mathbb{R}^{(2R+1)\times N\times d}$ taking an average over the N entities. We calculate an attention over the $2R+1$ outputs as:

$$\alpha = softmax\ \mathrm{sum}_{\mathrm{col}}\Big(\frac{QK^T}{\sqrt{d}}\Big) \in \mathbb{R}^{(2R+1)}$$

where $\mathrm{sum}_{\mathrm{col}}(X)$ takes the sum of each column in $X \in \mathbb{R}^{d_1\times d_2}$ and ends up in a vector $\in \mathbb{R}^{d_2}$.

The last problem to solve is that the initial features $H^{(0)}$ is often incomplete in real life. In most cases, people would go by one-hot features or randomized features. But we want to enable our model to use the real features, even if the real-features are incomplete. Inspired by graph representation learning strategies such as LINE [34], we proposed to treat the unknown features as trainable parameters. That is, for a graph $\mathcal{G}_p$ whose vertice set is $\mathcal{V}$, $\mathcal{V}_{featured} \bigcap \mathcal{V}_{featureless} = \varnothing$ and $\mathcal{V}_{featured} \bigcup \mathcal{V}_{featureless} = \mathcal{V}$, for any node with valid feature $\forall v_i \in \mathcal{V}_{featured}$, the node's feature vector $H_i^{(0)}$ is known and fixed. For $\forall v_j \in \mathcal{V}_{featureless}$, the corresponding row vector $H_j^{(0)}$ is unknown and treated as a trainable parameter. The generation of the features will be discussed in the Appendix A. In brief, TIMME can handle any missing-feature.

4.2 Multi-Task Decoder

We propose **TIMME** as a multi-task learning model such that the sparsity of the labels could be overcome with the help of the link information. As is shown in Figure 3, we propose two architectures of the multi-task decoder. When we test it on a single-task i, we simply disable the remaining losses but a single $\mathcal{L}_i$, and name our model in single-task mode **TIMME-single**.

$\mathcal{L}_0$ is defined the same way as was proposed in [18], in our case a binary cross-entropy loss:

$$\mathcal{L}_0 = -\sum_{y\in Y_{train}} \big(y\log(y) + (1-y)\log(1-y)\big)$$

where Y_{train} contains the labels in the training set we have.

$\mathcal{L}_1, \ldots \mathcal{L}_R$ are link-prediction losses, calculated by binary cross-entropy loss between link-labels and the predicted link scores' logits. To keep the links asymmetric, we used Neural Tensor Network

[1] MLP here refers to Multi-layer Perceptron.

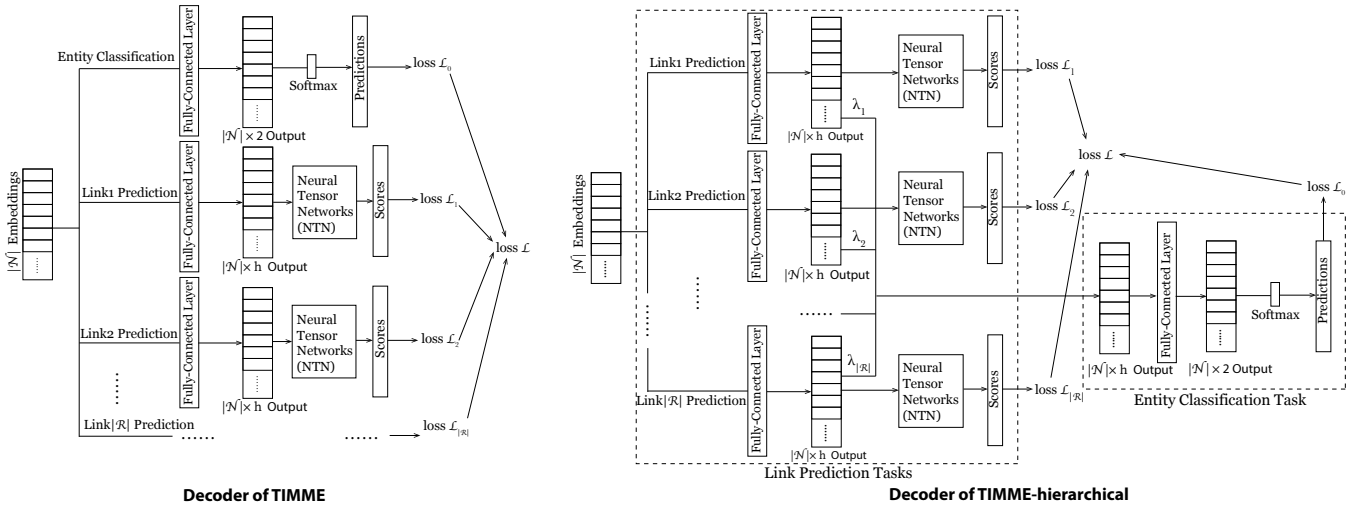


Figure 3: The two types of decoder in our multi-task framework, referred to as *TIMME* and *TIMME-hierarchical*.

(NTN) structure [32], with simplification inspired by DistMult [40]. We set the number of slices be $k = 1$ for $W_r \in \mathbb{R}^{d\times d\times k}$, omitting the linear transformer U, and restricting the weight matrices W_r each being a diagonal matrix. For convenience, we refer to this link-prediction cell as **TIMME-NTN**. Consider triplet (v_i, r, v_j), and denote the encoder output of $v_i, v_j \in \mathcal{V}$ as $h_i, h_j \in \mathbb{R}^d$, the score function of the link is calculated as:

$$s(i, r, j) = h_i W_r h_j + V \begin{bmatrix} h_i \\ h_j \end{bmatrix} + b$$

where $W_r \in \mathbb{R}^{d\times d}$ is a diagonal matrix for any $\forall r \in \mathcal{R}$. $W_r, V \in \mathbb{R}^{2d}$ and $b \in \mathbb{R}$ are all parameters to be learned. Group-truth label of a positive (existing) link is 1, otherwise 0.

The first decoder-architecture **TIMME** sums all $R + 1$ losses as $\mathcal{L} = \sum_{i=0}^{R} \mathcal{L}_i$. Without average, each task's loss is directly proportional to the amount of data points sampled at the current batch. Low-resource tasks will take a smaller portion. This is the most straightforward design of a MTL decoder.

The second, **TIMME-hierarchical**, has $\lambda = [\lambda_1, \ldots, \lambda_{|\mathcal{R}|}]^T$ being computed via self-attention on the average embedding over the R link-prediction task-specific embeddings. Here, $\mathcal{L} = \sum_{i=0}^{R} \mathcal{L}_i$ is the same with **TIMME**. **TIMME-hierarchical** essentially derives the node-label information from the link relations, thus provides some insights on each relation's importance to ideology prediction. **TIMME**, **TIMME-hierarchical**, **TIMME-single** models share *exactly the same* encoder architecture.

5 EXPERIMENTS

In this section, we'll introduce the dataset we crawled, cleaned and labeled, together with our experimental results and analysis.

5.1 Data Preparation

5.1.1 Data Crawling. The statics of the political-centered social network datasets we have are listed in Table 1. Data prepared is described in Appendix A, ready by April, 2019. In brief, we did:

	PureP	P50	P20~50	P+all
# User	583	5,435	12,103	20,811
# Link	122,347	1,593,721	1,976,985	6,496,107
# Labeled User	581	759	961	1,206
# Featured User	579	5,149	11,725	19,418
# Follow-Link	59,073	529,448	158,746	915,438
# Reply-Link	1,451	96,757	121,133	530,598
# Retweet-Link	19,760	311,359	595,030	1,684,023
# Like-Link	14,381	302,571	562,496	1,794,111
# Mention-Link	27,682	353,586	539,580	1,571,937

Table 1: Descriptive statistics of the three selected subsets of our dataset.

(1) Collecting some Twitter accounts of the politicians $\mathcal{P}$;
(2) For every politician $\forall p \in \mathcal{P}$, crawl her/his most-recent s followers and s followees, putting them in a candidate set C.
(3) For every candidate $c \in C$, we also crawl their most-recent s followers to make the *follow* relation more complete.
(4) For every user $u \in \mathcal{P} \cup C$, crawl their tweets as much as possible, until we hit the limit ($\approx 3,200$) set by Twitter API.
(5) From the followers & followees we collect *follow* relation, from the tweets we extract: *retweet*, *mention*, *reply*, *like*.
(6) Select different groups of users from C, based on how many connections they have with members in $\mathcal{P}$, and making those groups into the 4 subsets, as is shown in Table 1.
(7) We filter the relations within any selected group so that if a relation $e = \langle v_i, v_j \rangle \in \mathcal{G}_p$, there must be $v_i \in \mathcal{G}_p$ and $v_j \in \mathcal{G}_p$.

Our four datasets represent different user groups. **PureP** contains only the politicians. **P50** contains politicians and users keen on political affairs. **P20~50** is politicians with the group of users who are of moderate interests on politics. **P+all** is a union set of the three, plus some randomly-selected outliers of politics. **P+all** is the most challenging subset to all models. More details on the dataset,

Model	PureP	P50	P20~50	P+all
GCN	**1.0000**/**1.0000**	0.9600/0.9600	0.9895/0.9895	0.9076/0.9083
r-GCN	**1.0000**/**1.0000**	0.9733/0.9733	0.9895/0.9895	0.9327/0.9333
HAN	0.9825/0.9824	0.9466/0.9467	0.9789/0.9789	0.9238/0.9250
TIMME-single	**1.0000**/**1.0000**	0.9733/0.9733	0.9895/0.9895	0.9333/0.9324
TIMME	0.9825/0.9824	**0.9867**/**0.9867**	**1.0000**/**1.0000**	0.9495/0.9500
TIMME-hierarchical	**1.0000**/**1.0000**	0.9733/0.9780	0.9895/0.9895	**0.9580**/**0.9583**

Table 2: Node classification measured by F1-score/accuracy.

Model	PureP	P50	P20~50	P+all
Follow Relation				
GCN+	0.8696/0.6167	0.9593/0.8308	0.9870/0.9576	0.9855/0.9329
r-GCN	0.8596/0.6091	0.9488/0.8023	0.9872/0.9537	0.9685/0.9201
HAN+	**0.8891**/**0.7267**	0.9598/0.8642	0.9620/0.8850	0.9723/0.9256
TIMME-single	0.8809/0.6325	0.9717/0.8792	0.9920/0.9709	0.9936/0.9696
TIMME	0.8763/0.6324	**0.9811**/**0.9154**	0.9945/0.9799	0.9943/0.9736
TIMME-hierarchical	0.8812/0.6409	0.9809/0.9145	**0.9984**/**0.9813**	**0.9944**/**0.9739**
Reply Relation				
GCN+	0.8602/0.7306	0.9625/0.9022	0.9381/0.8665	0.9705/0.9154
r-GCN	0.7962/0.6279	0.9421/0.8714	0.8868/0.7815	0.9640/0.9085
HAN+	0.8445/0.6359	0.9598/0.8616	0.9495/0.8664	0.9757/0.9210
TIMME-single	0.8685/0.7018	0.9695/0.9307	0.9593/0.9070	0.9775/0.9508
TIMME	0.9077/0.8004	**0.9781**/**0.9417**	**0.9747**/**0.9347**	0.9849/0.9612
TIMME-hierarchical	**0.9224**/**0.8152**	0.9766/0.9409	0.9737/0.9341	**0.9854**/**0.9629**
Retweet Relation				
GCN+	0.8955/0.7145	0.9574/0.8493	0.9351/0.8408	0.9724/0.9303
r-GCN	0.8865/0.6895	0.9411/0.8084	0.9063/0.7728	0.9735/0.9326
HAN+	0.7646/0.6139	0.9658/0.9213	0.9478/0.8962	0.9750/0.9424
TIMME-single	0.9015/ 0.7202	0.9754/0.9127	0.9673/0.9073	0.9824/0.9424
TIMME	0.9094/0.7285	0.9779/0.9181	**0.9772**/**0.9291**	0.9858/0.9511
TIMME-hierarchical	**0.9105**/**0.7344**	**0.9780**/**0.9190**	0.9766/0.9275	**0.9869**/**0.9543**
Like Relation				
GCN+	0.9007/0.7259	0.9527/0.8499	0.9349/0.8400	0.9690/0.9032
r-GCN	0.8924/0.7161	0.9343/0.7966	0.9038/0.7681	0.9510/0.8945
HAN+	0.8606/0.6176	0.9733/0.8851	0.9611/0.9062	**0.9894**/0.9481
TIMME-single	0.9113/0.7654	0.9725/0.9119	0.9655/0.9069	0.9796/0.9374
TIMME	0.9249/0.7926	**0.9753**/0.9171	**0.9759**/**0.9292**	0.9846/0.9504
TIMME-hierarchical	**0.9278**/**0.7945**	0.9752/**0.9175**	0.9752/0.9271	0.9851/**0.9518**
Mention Relation				
GCN+	0.8480/0.6233	0.9602/0.8617	0.9261/0.8170	0.9665/0.8910
r-GCN	0.8312/0.6023	0.9382/0.7963	0.8938/0.7563	0.9640/0.8902
HAN+	**0.9000**/**0.7206**	0.9573/0.8616	0.9574/0.8891	0.9724/0.9119
TIMME-single	0.8587/0.6502	0.9713/0.8981	0.9614/0.8923	0.9725/0.9096
TIMME	0.8684/0.6689	0.9730/0.9035	**0.9730**/**0.9185**	0.9839/0.9446
TIMME-hierarchical	0.8643/0.6597	**0.9732**/**0.9046**	0.9723/0.9166	**0.9846**/**0.9463**

Table 3: Link-prediction measured by ROC-AUC/PR-AUC.

including how we generated features and how we tried to get more labels, are all described in details in Appendix A.

5.2 Performance Evaluation

We split the train, validation, and test set of node labels by 8:1:1, keep it the same across all datasets & all models, every time measuring by F1-score and accuracy. For link-prediction tasks, we split all positive links into training, validation, and testing sets by 85:5:10, keeping same portion across all datasets & all models, measuring by ROC-AUC and PR-AUC. [2]

5.2.1 Baseline Methods. We have explored a lot of possible baseline models. Some methods we mentioned in section 2, HetGNN [42], GATNE [3] and GTN [41] generally converge $\approx 10 \sim 100$ times slower than our model on any task. GraphSAGE [13] is not very suitable on our dataset. Moreover, other well-designed models such as GIN [39] are way too different from our approach at a very fundamental level, and we didn't consider them as baselines. Some other methods such as GEM [21] and SHINE [37] should be capable of handling the dataset at this scale, but they are not releasing their code to the public, and we can't easily guarantee reproduction.

We decided to use the three baselines: GCN, r-GCN and HAN. They are closely-related to our model, open-sourced, and efficient. We understand that none of them were specifically designed for social-networks dataset, and early explorations without tuning them resulted in terrible outcomes. To make the comparisons fair, we did a lot of work in hyper-parameter optimization, so that their performances are significantly improved. Our GCN baseline treats all links as the same type and takes in only one adjacency matrix. The models are adapted to new tasks that weren't mentioned in their original papers. We refer to GCN+ and HAN+ as the GCN-base-model or HAN-base-model with **TIMME-NTN** attached to it. By comparing with GCN/GCN+, we show that treating relations as heterogeneous is beneficial. Comparing with r-GCN we prove in practice that their design is not suitable for social networks like ours. With HAN/HAN+ we show that, although their model is potentially more expressive, our model still outperforms theirs in most cases, even after we carefully improved it to its highest potential (Appendix C). We didn't have to tune the hyper-parameters of **TIMME** models closely as hard, thanks to its robustness.

HAN+ significantly perform better with more sample points. Its expressive and flexible complex structure helps it achieve high in some tasks. The downsides of HAN/HAN+ are also obvious: it easily gets over-fitting, extremely sensitive to dataset statistics, large memory consumption that takes more than *32G* memory to run tasks on **P+all**, where TIMME models takes less than *4G* space with the same hidden size and embedding dimensions.

5.2.2 TIMME. To stabilize the training, we'd have to use the step-decay learning rate scheduler, the same with that for ResNet. The optimizer we use is Adam, kept consistent with GCN and r-GCN. By default, our encoder utilizes one-hot embedding. One of the many advantages of **TIMME** is how robust it is to the hyper-parameters and all other settings, reflected by that the same default parameter settings serve all experiments well. Like many others have done before, to avoid information leakage, whenever we run tasks involving link-prediction, we will remove all link-prediction test-set links from our adjacency matrices.

It is shown in Table 2 and 3 that multi-task models **TIMME** and **TIMME-hierarchical** are generally better than **TIMME-single** on most tasks. Even **TIMME-single** is superior to the baseline models most of the times. TIMME models are stable and scalable. The classification task, despite the many labels we manually added, easily over-estimating the models. Models trained on single node-classification task will easily get over-fitted. If we force them to keep training after convergence, only **TIMME** and **B** keep stable, the baselines and **TIMME-single** will suffer from dramatic performance-drop, especially HAN/HAN+.

5.3 Case Studies

5.3.1 The Influence of Text Feature. To justify the reason why we gave up using text features, we show the node-classification

[2] AUC refers to Area Under Curve, PR for precision-recall curve, ROC for receiver operating characteristic curve.

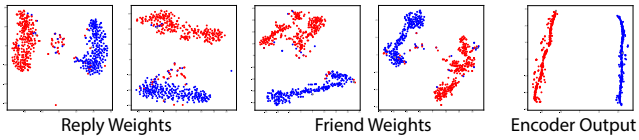
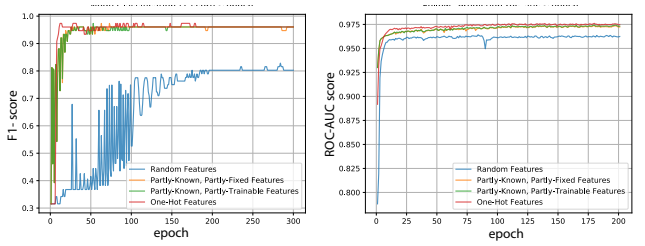


Figure 4: TSNE results on the reply (and reversed), friend (and reversed) weight matrices of the first convolutional layer ($W^{(0)}$), and the encoder output embeddings ($H_r^{(2)}$). Red for ground-truth republican nodes, blue for democratic nodes.



(a) Features' Impact on Node-Classification Task (b) Features' Impact on "Follow" Link-Prediction Task

Figure 5: Illustration of impact of features. Random features in blue, partly-know partly randomized (and fixed) in yellow, partly-known partly-trainable in green, one-hot in red.

training-curves of **TIMME-single** with one-hot features, randomized features, partly-known-partly-randomized features, and with partly-known-partly-trainable features. The results are collected from **P50** dataset. To make it easier to compare, we have fixed training epochs 300 for node-classification, and 200 for follow-relation link-prediction. It is shown that text feature is significantly better than randomized feature, and treating the missing part of the text-generated feature as trainable is better than treat it as fixed randomized feature. However, one-hot feature always outperforms them all, essentially means that relations are more reliable and less noisy than text information in training our network embedding. We have proved in Appendix B that the $2R+1$ weight matrices at the first convolutional layer captures the nodes' learned features when using one-hot features. Experimental evidence is shown in Figure 4, showing the two weight matrices corresponding to the two (from true direction + reversed) $\hat{A}$ of some relations. Shown on the right is the TSNE embedding of the encoder output at the same time. It shows that although the first embedding layer also captured the features of nodes, the encoder output is always the one with the highest quality. All these come from the embedding at epoch 300 train by node-classification task on **PureP**.

5.3.2 Performance Measurement on News Agency. A good measurement of our prediction's quality would be on some users with ground-truth tendency, but regarded unlabeled in our dataset. News agents' accounts are typically such users, as is shown in Figure 8. Among them we select some of the agencies believed to have the most extreme tendencies. [3] The continuous scores we have for prediction come from the softmax of the last-layer output of our node-classification task, which is in the format of $(prob_{left}, prob_{right})$. Right in the middle represents $(prob_{left}, prob_{right}) = (0.5, 0.5)$, left-most being $(1.0, 0.0)$, right-most $(0.0, 1.0)$. For most cases, our model's predictions agree with people's common belief. But CNN News is an interesting case. It is believed to be extremely left, but predicted as slightly-left-leaning centrist. Some others have findings supporting our results: CNN is actually only a little bit left-leaning. [4] Although the public tends to believe that CNN is extremely liberal, it is more reasonable to consider it as centrist biased towards left-side. People's opinion on news agencies' tendencies might be polarized. Besides, although there are significantly more famous news agencies on the liberal side, those right-leaning ones tend to support their side more firmly.

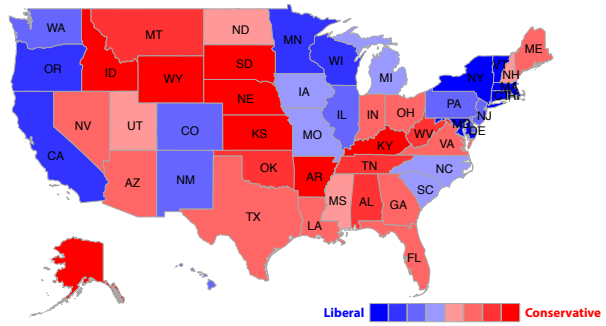


Figure 6: Overall ideology on Twitter in each state.

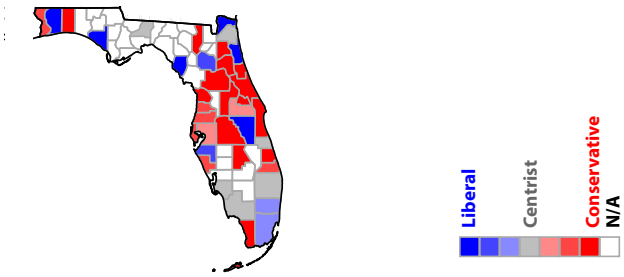


Figure 7: Overall ideology on Twitter, Florida (FL).

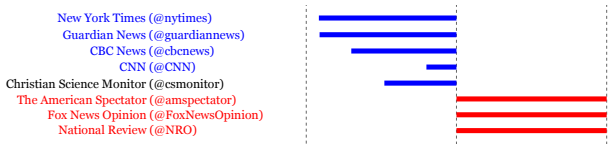


Figure 8: The News Agencies' Ideologies. Text colors come from the public's voting online, blue for left and red for right, black for middle (centrist). Length reflects the extent.

[3] We fetch most of the ground-truth labels of the news agents from the public voting results on https://www.allsides.com/media-bias/media-bias-ratings, got them **after** the prediction results are ready.

[4] https://libguides.com.edu/c.php?g=649909&p=4556556

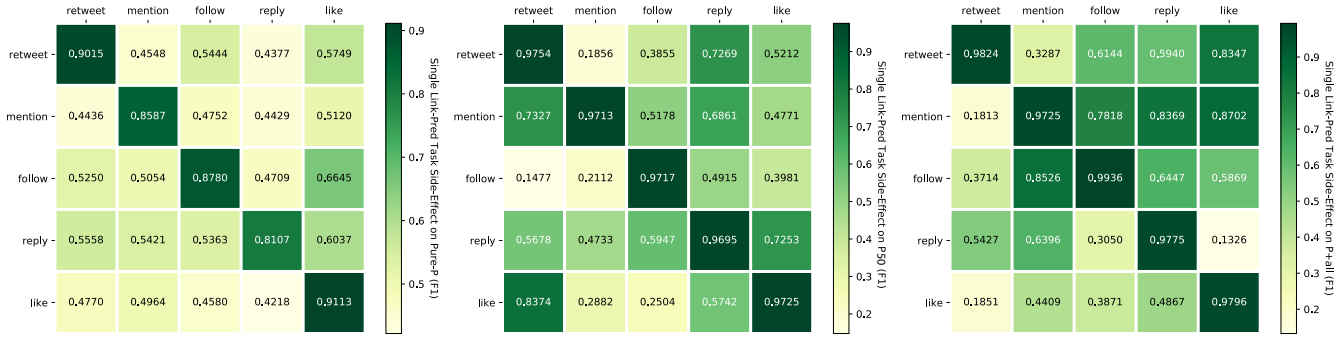


Figure 9: The impact of training on single-link-prediction tasks, on Pure-P (left), P50 (middle), P+all (right) dataset respectively.

5.3.3 Geography Distribution. Consider results from the largest dataset (**P+all**), and with predictions coming out from **TIMME-hierarchical**. We predict each Twitter user's ideology as either liberal or conservative. Then we calculate the percentage of the users on both sides, and depict it in Figure 6. Darkest red represents $p \in [0, \frac{1}{8}]$ of users in that area are liberal, remaining $[\frac{7}{8}, 1]$ are conservative; darkest blue areas have $[\frac{7}{8}, 1]$ users being liberal, $[0, \frac{1}{8}]$ conservative. The intermediate colors represent the evenly-divided ranges in between. The users' locations are collected from the public information in their account profile. From our observation, conservative people are typically under-represented. [5][6] For instance, as a well-known firmly-conservative state, Utah (UT) is only shown as slightly-right on our map.

This is intuitively reasonable, since Twitter users are also biased. Typically biased towards youngsters and urban citizens. Although we are able to solve the problem of silent-majority by utilizing their link relations instead of text expressions, we know nothing about offline ideology. We suppose that some areas are silent on Twitter, and this guess is supported by the county-level results at Florida, shown in Figure 7. This time the color-code represents evenly-divided seven ranges from $[0, \frac{1}{7}]$ to $[\frac{6}{7}, 1]$, because of the necessity of reserving one color for representing silent areas (denoted as white for *N/A*). The silent counties, typically some rural areas, have no user in our dataset, inferring that people living there don't use Twitter very often. The remaining parts of the graph makes complete sense, demonstrating a typical swing state. [7]

5.3.4 Correlated Relations. When we train **TIMME-single** with only one relation type, some other relations' predictions benefit from it, and are becoming more and more accurate. We assume that, if by training on relation r_i we achieve a good performance on relation r_j, then we say relation r_i probably leads to r_j. As is shown in Figure 9, relations among politicians are relatively independent except that all other relations might stimulate *like*. In more ordinary user groups, *reply* is the one that significantly benefit from all other relations. It is also interesting to observe that the highly-political

[5]National General Election Polls data partly available at https://www.realclearpolitics.com/epolls/2020/president/National.html.
[6]Compare with the visualization of previous election at https://en.wikipedia.org/wiki/Political_party_strength_in_U.S._states.
[7]The ground-truth election outcome in Florida at 2016 is at https://en.wikipedia.org/wiki/2016_United_States_presidential_election_in_Florida.

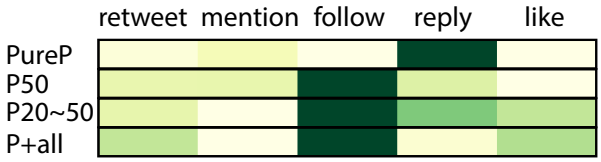


Figure 10: Illustration of λ value in decoder on each dataset.

P50 shows that *like* leads to *retweet*, while from more ordinary users' perspective once they *liked* they are less likely to *retweet*. The relations among the relations are asymmetric.

5.3.5 Relation's Contributions to Ideology Detection. The importance of each relation to ideology prediction could be measured by the value of the corresponding λ_r values in the decoder of **TIMME-hierarchical**. All the values are close to 0.2 in practice, in $[0.99, 2.01]$, but still has some common trends, as is shown in Figure 10. Despite that *reply* pops out rather than *follow* on **PureP**, we still insist that ***follow*** *is the most important relation.* That is because we only crawled the most recent about 5000 followers / followees. If a follow happened long time ago, we wouldn't capture it. The *follow* relation is especially incomplete on **PureP**.

6 CONCLUSION

TIMME, the model we proposed, is good at handling multiple relations, with a multi-relational encoder, and multi-task decoder. We step aside the silent-majority problem by relying mostly on the relations, instead of the text information. Optionally, we accept incomplete input features, but we showed that links are able to do well on generating the ideology embedding without additional text information. From our observation, links help more than text in ideology-detection problem, and *follow* is the most important relation to ideology detection. We also concluded from visualizing the Twitter average state-level ideology map that conservative voices tend to be under-represented on Twitter. We also conclude that public opinions on news agencies' ideology could be polarized. Our model could be easily extended to any other social network embedding problem, such as on any other dataset like Facebook as long as the dataset is legally available, and of course it works on predicting other tendencies like preferring Superman or Batman. Our dataset would be beneficial to the community once released.

7 ACKNOWLEDGEMENT

At the early stage of this work, Haoran Wang [8] contributing a lot to the initial implementation of the model, everyone from the team benefits a lot from his well-organized structure and clear coding-logic. Meanwhile, Zhiwen Hu [9] compared many related methods' efficiencies, and his works shed light on the right path for us to go.

Our team also received some external help from Yupeng Gu. He offered us his crawler code and his old dataset as references.

REFERENCES

[1] Christopher H Achen. 1975. Mass political attitudes and the survey response. *American Political Science Review* 69, 4 (1975), 1218–1231.
[2] Ramy Baly, Georgi Karadzhov, Abdelrhman Saleh, James Glass, and Preslav Nakov. 2019. Multi-task ordinal regression for jointly predicting the trustworthiness and the leading political ideology of news media. *arXiv preprint arXiv:1904.00542* (2019).
[3] Yukuo Cen, Xu Zou, Jianwei Zhang, Hongxia Yang, Jingren Zhou, and Jie Tang. 2019. Representation learning for attributed multiplex heterogeneous network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1358–1368.
[4] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247* (2018).
[5] Wei Chen, Xiao Zhang, Tengjiao Wang, Bishan Yang, and Yi Li. 2017. Opinion-aware Knowledge Graph for Political Ideology Detection.. In *IJCAI*. 3647–3653.
[6] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 257–266.
[7] Joshua Clinton, Simon Jackman, and Douglas Rivers. 2004. The statistical analysis of roll call data. *American Political Science Review* 98, 2 (2004), 355–370.
[8] Michael D Conover, Bruno Gonçalves, Jacob Ratkiewicz, Alessandro Flammini, and Filippo Menczer. 2011. Predicting the political alignment of twitter users. In *2011 IEEE third international conference on privacy, security, risk and trust and 2011 IEEE third international conference on social computing*. IEEE, 192–199.
[9] Michael D Conover, Jacob Ratkiewicz, Matthew Francisco, Bruno Gonçalves, Filippo Menczer, and Alessandro Flammini. 2011. Political polarization on twitter. In *Fifth international AAAI conference on weblogs and social media*.
[10] Aron Culotta, Nirmal Ravi Kumar, and Jennifer Cutler. 2015. Predicting the Demographics of Twitter Users from Website Traffic Data.. In *AAAI*, Vol. 15. Austin, TX, 72–8.
[11] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*. 3844–3852.
[12] Yupeng Gu, Ting Chen, Yizhou Sun, and Bingyu Wang. 2016. Ideology detection for twitter users with heterogeneous types of links. *arXiv preprint arXiv:1612.08207* (2016).
[13] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in neural information processing systems*. 1024–1034.
[14] Mohit Iyyer, Peter Enns, Jordan Boyd-Graber, and Philip Resnik. 2014. Political ideology detection using recursive neural networks. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 1113–1122.
[15] Kristen Johnson and Dan Goldwasser. 2016. Identifying stance by analyzing political discourse on twitter. In *Proceedings of the First Workshop on NLP and Computational Social Science*. 66–75.
[16] Sandeepa Kannangara. 2018. Mining twitter for fine-grained political opinion polarity classification, ideology detection and sarcasm detection. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. 751–752.
[17] Alex Kendall, Yarin Gal, and Roberto Cipolla. 2018. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 7482–7491.
[18] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).
[19] Theresa Kuhn and Aaron Kamm. 2019. The national boundaries of solidarity: a survey experiment on solidarity with unemployed people in the European Union. *European Political Science Review* 11, 2 (2019), 179–195.
[20] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.
[21] Ziqi Liu, Chaochao Chen, Xinxing Yang, Jun Zhou, Xiaolong Li, and Le Song. 2018. Heterogeneous graph neural networks for malicious account detection. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 2077–2085.
[22] Sergio Martini and Mariano Torcal. 2019. Trust across political conflicts: Evidence from a survey experiment in divided societies. *Party Politics* 25, 2 (2019), 126–139.
[23] Viet-An Nguyen, Jordan Boyd-Graber, Philip Resnik, and Kristina Miler. 2015. Tea party in the house: A hierarchical ideal point topic model and its application to republican legislators in the 112th congress. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 1438–1448.
[24] Chang Sup Park. 2013. Does Twitter motivate involvement in politics? Tweeting, opinion leadership, and political engagement. *Computers in Human Behavior* 29, 4 (2013), 1641–1648.
[25] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.
[26] Gary Pollock, Tom Brock, and Mark Ellison. 2015. Populism, ideology and contradiction: mapping young people's political views. *The Sociological Review* 63 (2015), 141–166.
[27] Keith T Poole and Howard Rosenthal. 1985. A spatial model for legislative roll call analysis. *American Journal of Political Science* (1985), 357–384.
[28] Daniel Preoţiuc-Pietro, Ye Liu, Daniel Hopkins, and Lyle Ungar. 2017. Beyond binary labels: political ideology prediction of twitter users. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 729–740.
[29] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084* (2019).
[30] Sebastian Ruder. 2017. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098* (2017).
[31] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*. Springer, 593–607.
[32] Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. 2013. Reasoning with neural tensor networks for knowledge base completion. In *Advances in neural information processing systems*. 926–934.
[33] Yizhou Sun and Jiawei Han. 2012. Mining heterogeneous information networks: principles and methodologies. *Synthesis Lectures on Data Mining and Knowledge Discovery* 3, 2 (2012), 1–159.
[34] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*. 1067–1077.
[35] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).
[36] Prashanth Vijayaraghavan, Soroush Vosoughi, and Deb Roy. 2017. Twitter demographic classification using deep multi-modal multi-task learning. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. 478–483.
[37] Hongwei Wang, Fuzheng Zhang, Min Hou, Xing Xie, Minyi Guo, and Qi Liu. 2018. Shine: Signed heterogeneous information network embedding for sentiment link prediction. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. 592–600.
[38] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. 2019. Heterogeneous graph attention network. In *The World Wide Web Conference*. 2022–2032.
[39] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).
[40] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2014. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575* (2014).
[41] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. 2019. Graph Transformer Networks. In *Advances in Neural Information Processing Systems*. 11960–11970.
[42] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V Chawla. 2019. Heterogeneous graph neural network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 793–803.

[8] *wanghran@gmail.com*, currently working at Snap Inc.
[9] *huzhiwen@g.ucla.edu*, currently working at PayPal.

A DATA PREPARATION

The dataset we want is a political-centered social network (Section 3), a selected subset from the giant Twitter network we take to analyze. It is a challenging task to learn good embedding on this dataset. For example, for GraphSAGE, neighborhood-sampling can not be easily done both effectively and efficiently. The tools we used to crawl politicians' name lists from the government website, and their potential Twitter accounts from Google, is Scrapy. [10] To legally and reliably crawl from Twitter data, we first applied for Developer API from Twitter [11], and then used Tweepy [12] for crawling. We set very strict rate limits for our crawlers so as not to harm any server. Our dataset is released at https://github.com/PatriciaXiao/TIMME. Our dataset reaches some blind spots of many existing models. Raw data was collected by April, 2019.

A.1 Twitter IDs Preparation

Let's take the same notation as in Section 3. We describe the process as: to construct $\mathcal{G}_p = \{\mathcal{V}, \{\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3, \mathcal{E}_4, \mathcal{E}_5\}\}$, we first select the users to be included $\mathcal{V}$, then we include the links among vertices in $\mathcal{V}$ under each relation $r \in \mathcal{R} = \{1, 2, 3, 4, 5\}$ into $\mathcal{E}_r$ accordingly.

A.1.1 Politicians Twitter IDs. As is described briefly in Section 5.1, we need to start from a set of politicians $\mathcal{P}$, which we treat as seeds for further crawling.

To start with, we first get the name list of the recently-active politicians, consists of:

- The union-set of 115^{th} and 116^{th} US congress members, where we observe a lot of overlap between the two groups; [13]
- Recent-years' presidents and their cabinets; [14]
- Additional politicians must be included: Hilary Clinton, who was running for the president of the United States not long ago; Michelle Obama, who was the former First Lady.

Next, with the help of Google, we crawled the most-likely Twitter names and IDs of the politicians. We do so automatically, by providing Google a politician's name and the keyword "twitter", and parsing the first response. Then after manual filtering, we have 583 politicians' Twitter accounts available, who make up our politicians set $\mathcal{P}$. Anyone else to be included in our dataset must be in the 1-hop neighborhood of a politician (Section 3).

A.1.2 Candidate Non-Politicians Twitter IDs. With the help of Twitter Developer API, we are able to get the **full** followers and followees list of any Twitter user.

However, it is not affordable to include all followers and followees of the politicians, thus we set a limit on *window size s* when crawling the candidate non-politicians list, only accepting the most-recent $s = 5,000$ followers or followees of any politician. These followers and followees we collected form a raw candidate set C_{raw}. Then we remove the politicians from this set, resulting in the final candidates set $C = C_{raw} - \mathcal{P}$. $\forall v_i \in C$, we apply the same window size $s = 5,000$ and crawled their most recent s followers, s followees. All follower-followee pairs are stored into a database for the convenience of the following steps.

A.1.3 Selecting Subgroups from Candidates. C is still too large a user set, and chaotic, as we don't know anything about its components. To conduct meaningful analysis, we need to select some meaningful subgroups from it, such as a very-political subgroup, and a political-outliers subgroup, etc.

The criteria we used to select the desired subgroups of users is some thresholds. We define a political-measurement t_i for each user $v_i \in C$, who is followed by $t_{i,1}$ politicians $p \in \mathcal{P}$, and meanwhile following $t_{i,2}$ politicians, thus t_i is computed by $t_i = \max(t_{i,1}, t_{i,2})$.

Then we set a threshold range t, set upon each t_i, used for filtering the groups of users. Considering we set t as threshold range for graph $\mathcal{G}_p$, $\forall v_i \in \mathcal{V}$, if $t_i \in t$, then $v_i \in \mathcal{G}_p$, otherwise $v_i \notin \mathcal{G}_p$. By having $t = \{\infty\}$, we select a minimum subgraph containing purely politicians, resulting in our **PureP** dataset. $t \in [50, \infty)$ allows us to select a small group of users who are keen on political topics, together with the politicians, being our **P50** dataset. $t \in [20, 50)$ for less-political users, plus the politicians, being our **P20~50** dataset. $t \in [20, \infty)$ includes all nodes v_i whose $t_i \geq 20$. We want to have a dataset representing more general users, containing some users from each group. Therefore, we include another $3,000$ users randomly selected from the group $t \in [1, 5)$. Adding these random political-outlier users will make the dataset resembles the real network even more. Putting together the politicians, $t \in [20, \infty)$ group, and the $3,000$ random outliers from $t \in [0, 5)$ group, we form the dataset **P+all**. Ideally, **P+all** has representatives of all groups of users on Twitter. The statistics are concluded in Table 1.

A.2 Relation Preparation

Only the *follow* relation is directly observed and already well-prepared at this stage (stored in a database, as we mentioned before). Other Twitter relations: *retweet*, *mention*, *like*, *reply*, must be concluded from tweets. We distinguish the different relation types from the tweets by the tweets' fields in responded JSON from API. For example, there are some fields indicating if an **""** mark is a *mention*, a *retweet*, or it links to nothing. According to our observation, the fields in the Json file responded from Twitter API might change across time. We don't know when will it be the next update, so there's no ground-truth solution for this part. We suggest whoever want to do so test the crawler first on her/his own account, trying all behaviors to conclude some patterns. Note: rate limit applies. [15]

Due to the Twitter official API limits, the maximum amount of tweets we could crawl for each user along the timeline is around $3,200$. Therefore, all relations are incomplete. All links we have only reflect some recent interactions among the users.

A.3 Feature Preparation

We get feature from text, using a user's tweets posted to generate her/his feature. Although there has been some recent advances in NLP with transformer-based structures, such as BERT and XLNet,

[10] https://scrapy.org/
[11] https://developer.twitter.com/
[12] https://www.tweepy.org/
[13] Congress members' name list with party information is publicly available at https://www.congress.gov/members .
[14] Obama and Trump's cabinet is publicly available at https://obamawhitehouse.archives.gov/administration/cabinet and https://www.whitehouse.gov/the-trump-administration/the-cabinet/ respectively
[15] https://developer.twitter.com/en/docs/basics/rate-limiting

Sentence-BERT [29] found that BERT / XLNet embeddings are generally performing worse than GloVe [25] average on sentence-level tasks. Not to mention the computational cost of transformers. We therefore use GloVe-average of the words as features, Wikipedia 2014 + Gigaword 5 (300d) pre-trained version. When we apply the average-GloVe embedding on tweet-level, and want to tell the ideology behind the tweets, we could easily achieve $\approx 72.84\%$ accuracy, using a 2-layers MLP, after only 200 epochs of training.

A.4 Label Preparation

If we are to use only the 583 labels from the politicians, the evaluation will always be untrustworthy. To overcome this issue, we manually expand the labels. We first crawled the users' profiles of $\forall v_i \in \mathcal{P} \cup C$, getting their information such as location and account description. Next, using the descriptions, searching for the words *democratic*, *republican*, *conservative*, *liberal*, their correct spell and variations, we have a large group of candidates. Then we do manual filtering to get rid of the uncertain users, reading their descriptions and recent tweets. We successfully included 2, 976 high-quality new labels in the end. Those labels make the node-classification task significantly more stable and reliable. The location information also plays an important role in plotting the maps.

B PROOF OF WEIGHT BEING FEATURE

Starting from our layer-wise propagation formula, we have that, at the first convolutional layer (notations in Section 4):

$$H^{(1)} = \sigma\Big(\sum_{r \in \hat{\mathcal{R}}} \alpha_r \hat{A}_r H^{(0)} W_r^{(0)} \Big)$$

where $H^{(0)} \in N \times d^{(0)}$ is the input feature-matrix. When using one-hot embedding of features, $H^{(0)} = I$ and $d^{(0)} = N$, thus the right-hand-side is equivalent with $\sigma\Big(\sum_{r \in \hat{\mathcal{R}}} \alpha_r \hat{A}_r W_r^{(0)} \Big)$. Now, $W_r^{(0)}$ on its own plays the role of $H^{(0)} W_r^{(0)}$ when $H^{(0)} \neq I$. Previously, relation r's propagation could be viewed as aggregation of a linear transformation ($W_r^{(0)}$) done on $H^{(0)}$, from the neighborhood ($\hat{A}_r$) of each node under relation r. Now, it could simply be viewed as the propagation of $W_r^{(0)} \in \mathbb{R}^{N \times d^{(1)}}$. From another point of view, it is equivalent as having input features being $\tilde{H}^{(0)} = W_r^{(0)} \in \mathbb{R}^{N \times d^{(1)}}$, and set $\tilde{W}_r^{(0)} \in \mathbb{R}^{d^{(1)} \times d^{(1)}} = I_{d^{(1)}}$ being fixed identical matrix not to be updated. That's the reason why we believe that $W_r^{(0)}$ captures the nodes' learned features under relation r.

C BASELINE HYPER-PARAMETER AND ARCHITECTURAL OPTIMIZATIONS

C.1 Applying GCN model Directly

As is discussed in Section 2, due to the uniqueness of the *political-centered social network dataset*, most of the existing models won't work well for our problem settings. We want to examine how well could GCN do when treating all relations as the same, ignoring the heterogeneous types. Very interestingly, without much work on hyper-parameter optimization, we only increased the hidden size and added the learning rate scheduler, it works pretty well. This phenomenon could potentially be an indirect evidence that relations are correlated, in addition to the discussions in Section 5.

C.2 Missing-Task Completion

We compare our model's performance on each task with the baselines. Ideally, we want models working on heterogeneous information networks with both node-classification task and link-prediction task as our baselines, so that we could compare with them directly. However, the situation we faced was not as easy as such. For instance, GCN and HAN never considered applying themselves directly on link-prediction tasks. But we all know that once we have the embeddings of the nodes, link prediction is doable.

Therefore, we decided that whenever a baseline originally couldn't handle a task, we give it our decoder's task-specific cells. This decision brings about some significant improvements on the link prediction performances of NTN+ and GCN+, since **TIMME-NTN** is powerful and efficient for link-prediction. Just in case, we also decide that when a node-classification task is missing, we should add a linear transformation layer with output units 2, the same as what we did, and apply a simple cross-entropy loss. From this perspective, it is no longer fair to compare them with r-GCN directly. To distinguish them from others' standard models, we add a plus sign **"+"** to the names, indicating that "we lend it our cells".

C.3 Optimizing r-GCN

The most important contribution of r-GCN is the weight-matrix decomposition methods. This mechanism would be very helpful in reducing the parameters, especially when the number of relations R is super high. However, in our case where R is small, the weight-decomposition operation is counter-effective. The first option, basis decomposition, the number of basis b is easily being larger than R. In the second option, block-diagonal decomposition, reduces the parameter size too dramatically, and harms the model's performance. Reviewing the experiments reported in the r-GCN paper, seeing how they chose these hyper-parameters across datasets, we found that when R is small, they often chose basis-decomposition with $b = 0$. We go by the same option, which works well in practice.

C.4 Optimizing HAN

When implementing node classification task on all datasets and link prediction task on smaller dataset, HAN/HAN+ gets easily over-fitting. What makes things worse, its training curve is never stable, and our early tryouts on using validation set to automatically stop it at an optimal point didn't work well. We had do it manually, by verifying when its best result appears on the validation set and when over-fitting starts, then select a good time to quit training. By default, we set learning rate to 0.005, regularization parameter 0.001, the semantic-level attention-vector dimension 128, multi-head-attention cell's number of heads $K = 8$. We set the hyper-parameters in the **TIMME-NTN** component of HAN+ the same with ours. Optimizing HAN was a tough work to do, for it requires re-adapting every choices we made on every dataset for every task. Adding more meta-path would potentially boosting its performance, but the computational cost will be overwhelming. Another observation is that, **TIMME models** are significantly better than HAN/HAN+ in handling imperfect features. When using GloVe-average features, **TIMME models** typically perform about 1% worse than using one-hot features, while HAN/HAN+ experience performance-drop up to around 10%.