Neighbours and Kinsmen: Hateful Users Detection with Graph Neural Network

Shu Li$^{1,2,3,4\text{✉}[000-0002-7445-7455]}$, Nayyar A. Zaidi$^{1[0000-0003-4024-2517]}$, Qingyun Liu^{2,3}, Gang Li$^{5[0000-0003-1583-641X]}$

[1] School of Information Technology, Deakin University, Geelong, VIC 3216, Australia
`{shul, nayyar.zaidi}@deakin.edu.au`
[2] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[3] National Engineering Laboratory of Information Security Technologies,Beijing, China
`liuqingyun@iie.ac.cn`
[4] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
[5] Centre for Cyber Security Research and Innovation, Deakin University, Geelong, VIC 3216, Australia
`gang.li@deakin.edu.au`

Abstract. With a massive rise of user-generated web content on social media, the amount of hate speech is also increasing. Countering online hate speech is a critical yet challenging task. Previous research has primarily focused on hateful content detection. In this study, we shift the attention from hateful content detection towards hateful users detection. Note, hateful users detection can benefit from users' tweets, profiles, social relationships, but the real benefit is that it can be aided by Graph Neural Networks (GNN). Typical Graph Neural Networks, such as `GraphSAGE`, only considers local neighbourhood information and samples the neighbourhood uniformly, thus they lack the ability to capture long-range relationships or to differentiate neighbours of a node. In this paper, we present `HateGNN` – a GNN-based method to address these two limitations. Our proposed method relies on the notion of latent neighbourhood, as well as systematic sampling of the neighbourhood nodes. The experimental results demonstrate that `HateGNN` outperforms state-of-the-art baselines in the task of detecting hateful users. We also provide a detailed analysis to demonstrate the efficacy of the proposed method.

Keywords: Hateful users; GNNs; Biased Sampling; Latent connections

1 Introduction

The proliferation of social media enables people to freely express their opinions online. However, it also becomes the breeding ground of hate speech that is described as abusive language, cyberbullying, discrimination, racism, sexism, threats, or toxicity [2]. The stark increase of hateful content on the internet has resulted in the emergence of conflict and hate [9]. Thus, hate speech classification has become a topic of growing interest for industry and academia. Over the past few years, a variety of models and methods on hate speech detection formulated it as a text classification task [13]. Current methods exploit the text representation as character n-grams or TF-IDF, and then resort to machine learning techniques, such as *Logistic Regression*, *SVM*, *Decision Trees*, and *Random Forests*. Recently, deep learning methods, such as *Recurrent Neural Networks* [14,10] and *Convolutional Neural Networks* [3], have been popularized in natural language processing to analyse online content.

Despite existing efforts in this area, hate speech detection remains a challenge. First, the state-of-the-art models oversimplify the problem, such as considering only tweets with hate-related words [1]. These methods rely entirely on textual (i.e., lexical and semantic) features [8], and are not aware of user and community information. Second, the state-of-the-art hate speech classifiers are vulnerable to extremely simple, model-agnostic attacks [7] [6]. These realistic attacks reduce the detection recall by nearly 50 % in some cases. Therefore, hate speech classifiers depending only on text detection are not robust against adversaries who deliberately mislead the classifiers.

Fortunately, the textual contents are not the only information that can be used to study hate speech in the social network. There is information that is often linked to a profile representing a person or an organization. Investigating such information presents plenty of opportunities to explore a richer feature space that can be helpful in identifying online hate speech. Moreover, it is more natural that users' profile is considered when detecting hate speech, rather than just considering isolated tweets. In addition, directly identifying (and controlling) hateful users who are intentionally propagating the hate speech is an important and effective measure for countering online hate speech. Therefore, in this study we shift the attention from hate speech toward hateful users.

Since hateful users detection can benefit from users' tweets, profiles and social relationships – it can be aided by *Graph Neural Networks* (GNNs). Specifically, users in the social network are nodes in the graph and the social relationship can be regarded as edges. Each node contains abundant property information such as user's profile, user's tweets. GNNs can be naturally applied to such node classification task by employing deep neural networks to aggregate feature information of neighbouring nodes. However, detecting hateful users based on GNNs has three main challenges.

- It is likely to be ineffective to aggregate neighbouring information for nodes that have no or too few relations with other nodes in the graph.
- A neighbourhood of a node is defined as the set of all neighbours which are one or more hops away. Typically, only features of the neighbourhood are aggregated. We conjecture that there might be nodes which could be very similar to the node in question, but are not in the neighbourhood of the node. The existing methods are not able to aggregate such high-similarity (non-neighbourhood) nodes.

[6] The model-agnostic attacks include diluting the hateful signal, obfuscating hateful tokens through character level perturbations, or injecting non-hate distractor.

– The current GNNs, especially the spatial-based methods like `GraphSAGE` [4], sample all neighbours equally when aggregating their information. It does not consider the fact that different neighbours may influence the node differently.

To address the limitations above, we propose a framework – `HateGNN`, for hateful user detection in the social network. `HateGNN` has following salient features:

– To address the first and second challenge described above, apart from the existing (explicit) social graph, we create a latent graph based on the node property information. This will help drastically for nodes which have no or too few neighbours. Moreover, the latent graph has the ability to capture the important features from distant but informative nodes.
– To address the third challenge, a bias strategy is applied to sample neighbours (not only immediate neighbours but latent neighbours) for differentiating the influences of the neighbours. We have proposed a sampling strategy to help choose the most informative features.
– Once the neighbours are selected, we aggregate the social and latent neighbourhoods to compute the final node embeddings.

We claim that the final embeddings obtained with `HateGNN` are much powerful than existing state-of-the-art methods. We back-up this claim by conducting experiments on two public datasets of hate speech. The results demonstrate the superior performance of `HateGNN` over state-of-the-art baselines. Moreover, we separately validate the efficacy of the salient features of `HateGNN`. We also provide detailed analyses on how various parameters (e.g., the size of sampling neighbours set, the similarity, the node properties) impact the model performance.

2 Related Work

Hate speech detection has been a popular research topic for decades. However, most of the existing literature focused on hate speech detection towards textual contents [13], and they attempted to adopt various typical classification algorithms [14,10,3]. Research on hateful users is still relatively under explored. The research of [11] characterized the hateful users on Twitter, and it showed that hateful users differ from normal ones in terms of their word usage, activity patterns and network structure. However, the work of [11] mainly focused on the analysis of hateful users rather than the detection model.

GNNs have been widely used to learn node embeddings. GNNs encode nodes into vectors by aggregating feature information from node's local neighbourhood via neural networks. To reduce the computational costs and improve performance, several recent studies have attempted to use different ways of neighbourhood aggregation. *Graph Convolutional Network* (GCN) uses a graph convolutional layer to encapsulate each node's hidden representation by summing "message" from all one-hop neighbours [6]. *Graph Attention Network* (GAT) [12] assigns different importance to different neighbours by utilising self-attention mechanism, and then combines their impacts to generate node embeddings. As a general inductive framework, `GraphSAGE` [4] is able to efficiently generate node embeddings for previously unseen data by sampling and aggregating features from a node's local neighbourhood. However, existing neighbourhood aggregation methods are not able to aggregate nodes that are similar but are far away from each other (i.e., not in immediate neighbourhood of each other). Moreover, these methods overlook the fact that different neighbours can influence a node in different ways. As discussed, we will address these issues with our proposed `HateGNN` method.

3 Problem Definition

In this section, we introduce the notions of social graph and latent graph, and then formally define the problem of *Hateful Users Detection with Graph Neural Network* (`HateGNN`). The social network, such as Twitter, is represented by a huge graph, in which every user is represented by a node and the follower/followee relationships between nodes are represented by an edge. Apart from the follower-followee network, the tweeting/retweeting network, representing the flow of information, can also be represented by a retweet graph, with users as the nodes and edges implying that one user has retweeted the post of another user. We define a social graph as:

Definition 1 (Social Graph). *A social graph is graph that can be created based on existing (explicit) connections between entities, and is parametrised as:* $\mathcal{G}^o = (\mathbf{V}, \mathbf{E}_{\mathcal{G}^o}, \mathbf{P})$*, where* $\mathbf{V}$ *are the nodes,* $\mathbf{E}_{\mathcal{G}^o}$ *correspond to the relationships of nodes, and* $\mathbf{P}$ *denotes the properties of nodes.*

The social graph defined above is fundamental to describe the relationship of the users, in which the node representation can be efficiently improved by aggregating features from its neighbours. However, it is unable to capture the long-distance dependencies among nodes with similar properties or topology structures (e.g. core, betweenness, and community bridges) when they are far away in the social graph. In order to solve the above problems, we propose a latent graph as:

Definition 2 (Latent Graph). *A latent graph is parametrised as:* $\mathcal{G}^l = (\mathbf{V}, \mathbf{E}_{\mathcal{G}^l}, \mathbf{P}, \lambda_{\mathcal{G}^l})$ *with nodes* $\mathbf{V}$ *and edges* $\mathbf{E}_{\mathcal{G}^l}$*, where the edges between two nodes* $u, v \in \mathbf{V}$ *denotes their similarity exceeds a certain threshold* $\lambda_{\mathcal{G}^l}$*.* $\mathbf{P}$ *represents the property of nodes as the same meaning in the* $\mathcal{G}^o$*.*

The problem of `HateGNN` is then formulated as follows. Given a set of users and their tweets in the social network, `HateGNN` aims to design a model $\mathcal{M}$ to learn the embedding of user, denoted as $\mathbf{x}_v$, and the function $\mathcal{F} : \mathbf{x}_v \rightarrow \Sigma$ that assigns the label information to users, so as to detect whether one user is hateful or not in the social network.

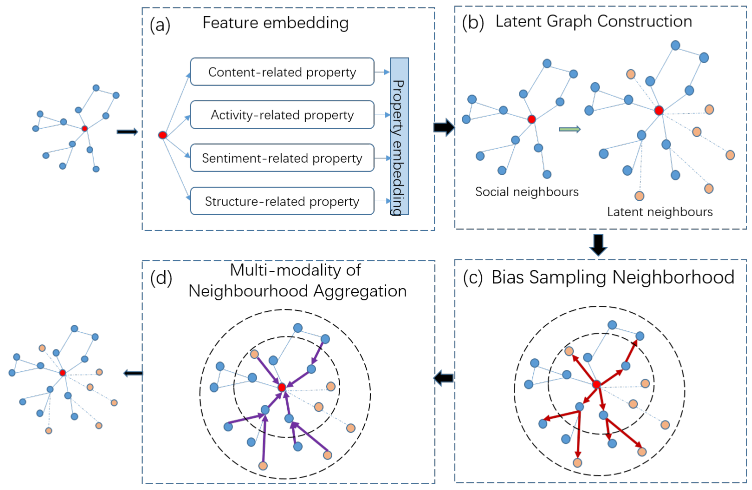
(a) Feature embedding
Content-related property
Activity-related property
Sentiment-related property
Structure-related property
Property embedding
(b) Latent Graph Construction
Social neighbours
Latent neighbours
(c) Bias Sampling Neighborhood
(d) Multi-modality of Neighbourhood Aggregation

Fig. 1: Pictorial illustration of the of `HateGNN` framework.

4 The `HateGNN` framework

In this section, we will describe our proposed `HateGNN` method, and illustrate it in Figure 1. First, the social graph $\mathcal{G}^o$ is obtained [7], where the nodes are the users, and each node has its property (e.g. users' tweets). Second, when the similarity of property vector between two nodes exceeds the threshold $\lambda_{\mathcal{G}^l}$, they are linked by a latent edge. These latent edges and the corresponding nodes form the latent graph $\mathcal{G}^l$. Third, a biased neighbourhood sampling strategy is implemented. Neighbours (both social and latent) that are more similar to the processed node, have higher priority to be sampled until a fixed-size set of neighbours $N^s_{(v)}$ is obtained. Finally, after choosing neighbours $N^s_{(v)}$, we aggregate their property information to obtain node embedding $\mathbf{x}_v$ by multiplying the weight matrices that are trained with neural networks.

4.1 Latent Graph Construction

Let us discuss the creation of latent graph. Note, for hateful user detection, the properties of node $v \in \mathbf{V}$, could be content-related, activity-related, or sentiment-related, etc. In addition, the topology structure for each node, could also be considered as a property [8]. The properties used in this work are detailed in Table 1. We obtain vector representation for each property (e.g. one-hot and labels encoders, `GloVe` for word embedding) and get the initial feature of node $v \in \mathbf{V}$, denoted as $\mathbf{z}_v$. For $u, v \in \mathbf{V}$, their similarity is defined based on as:

$$S(v,u) = \text{PearsonSimilarity}(\mathbf{z}_v, \mathbf{z}_u). \tag{1}$$

[7] The follower-followee or tweeting/retweeting relationship can be obtained conveniently by using Application Programming Interface (API) provided by the social network.
[8] The topology structure refers to centrality measurements for v in $\mathcal{G}^o$.

Table 1: List of various nodes' properties used in HateGNN.

Property Type	Property values
Content-related property	users' tweets.
Activity-related property	the number of tweets, retweet, follower, followees, favourites, hashtags, quote, URLs, mentions per tweet in average, and average and median time interval between tweets.
Sentiment-related property	sentiment of tweets, and bad-words usage.
Structure-related property	betweenness, eigenvector, indegree, outdegree, and the above property for the 1-neighbourhood of a user.

When the similarity between two nodes exceeds the threshold $\lambda_{\mathcal{G}^l}$, they are linked by a latent edge, which finally creates a latent graph $\mathcal{G}^l$.

4.2 Biased Sampling Neighbourhood

When the graph have high-degree nodes (i.e., the nodes have a large number of neighbours), considering all neighbours for aggregation is usually inefficient and unnecessary [5]. Given that a node's neighbours in graph have no natural ordering (e.g., sentences, images), GraphSAGE [4] proposed to uniformly sample a fixed-size set of neighbours, which outperforms strong GNNs models. Our proposed framework HateGNN improves GraphSAGE by deriving a set of sampled neighbours based on their similarity. The intuition is that similar neighbours (similar in any type of properties listed in Table 1) could consolidate and enhance the node embedding results.

Algorithm 1 describes the overall procedure of our sampling process. The operations from step 4 to 12 calculate the node similarity to sample neighbours. Then, for each node $v \in \mathbf{V}$, it aggregates the representations of its sampled neighbourhood, $\{\mathbf{h}_u^{k-1}, \forall u \in N^s_{(v)}\}$, and then concatenates the node's current representation, $\mathbf{h}_v^{k-1}$. This concatenated vector is fed through a fully connected layer with non-linear activation function σ. Finally, we get the final representations output at depth K, denoted as $\mathbf{x}_v = \mathbf{h}_v^k, \forall v \in \mathbf{V}$. Specifically, the similarity threshold $\lambda_{\mathcal{G}^l}$ in the latent graph could be set equal to the bias parameter η in the HateGNN model.

4.3 Multi-modality of Neighbourhood Aggregation

The neighbourhoods $N(v) = \Big\{N_o(v), N_l(v)\Big\}$ of node v includes its neighbourhood in both the social graph and the latent graph. The social-neighbourhood $N_o(v)$ consists of the set of v's adjacent nodes in the social graph $\mathcal{G}^o$, and the latent-neighbourhood $N_l(v)$ are those whose similarity to node v are higher than a parameter $\lambda_{\mathcal{G}^l}$. In aggregation process, we combine the social neighbourhood and the latent neighbourhood to generate the node embedding. The motivation is that different types of neighbours will make different contributions to the final node

Algorithm 1 `HateGNN` using Biased Sampling Neighbourhood

```
Input: Graph $\mathcal{G} = (\mathbf{V}, \mathbf{E}, \mathbf{P})$; input features $\{\mathbf{z_v}, \forall v \in \mathbf{V}\}$; depth $K$;
       Weight matrices $\mathbf{W}^k, \forall k \in \{1..., K\}$; non-linearity $\sigma$;
       Mean aggregator functions $\text{AGGREGATE}_k^{mean}$, $\forall k \in \{1..., K\}$ ;
       Neighbourhood $N(v)$;
       The sampled neighbourhood $N^s_{(v)}$, the size $|N^s_{(v)}|$;
       The size of neighbours to be sampled $\beta$; The bias parameter $\eta$;
Output: Vector representations $\mathbf{x}_v$ for all $v \in \mathbf{V}$;
 1: $\mathbf{h}_v^0 \leftarrow \mathbf{z_v}, \forall v \in \mathbf{V}$;
 2: for $\{k = 1...K\}$ do
 3:     for $v \in \mathbf{V}$ do
 4:         for $u \in N_{(v)}$ do
 5:             if $|N^s_{(v)}| \leq \beta$ then
 6:                 if $S(v,u) \geq \eta$ using Equation (1) then
 7:                     $N^s_{(v)} \leftarrow u$ ;
 8:                 end if
 9:             else
10:                 break;
11:             end if
12:         end for
13:         $\mathbf{h}^k_{N^s_{(v)}} \leftarrow \text{AGGREGATE}_k^{mean}\Big(\{\mathbf{h}_u^{k-1}, \forall u \in N^s_{(v)}\}\Big)$ ;
14:         $\mathbf{h}_v^k \leftarrow \sigma\Big(\mathbf{W}^k \cdot \text{ CONCAT } (\mathbf{h}_v^{k-1}, \mathbf{h}^k_{N^s_{(v)}})\Big)$;
15:     end for
16:     $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / ||\mathbf{h}_v^k||_2, \forall v \in \mathbf{V}$ ;
17: end for
18: $\mathbf{x}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathbf{V}$;
```

representation. For the social-neighbourhood, it denotes the effect of user' social nature. In comparison to this explicit relationship, the latent-neighbourhood indicates the long-range dependencies with the node, which is invisible and cannot be captured directly. Thus the step 13 in `HateGNN` could be updated with Equation (2), in which $N^s_{o_{(v)}}$ and $N^s_{l_{(v)}}$ are the sampled neighbours from the social graph and the latent graph, respectively, that is:

$$\mathbf{h}^k_{N^s_{(v)}} \leftarrow \text{AGGREGATE}_k^{mean}\left(\left\{\mathbf{h}_u^{k-1}, \forall u \in \{N^s_{o_{(v)}} \cup N^s_{l_{(v)}}\}\right\}\right). \tag{2}$$

4.4 Model Training

`HateGNN` is not attempting to learn the embedding results for all nodes in a graph, but to learn a mapping that generates embedding for each node. Depending on the dataset with node labels for hateful user detection, we train the model in a semi-supervised learning paradigm. With the labelled nodes, we train `HateGNN` by minimizing the cross entropy via back-propagation and gradient descent. thus, the loss function is calculated as:

$$\mathcal{L} = \sum_{v \in \mathbf{V}} \Big(y_v \log p_v + (1 - y_v)\log(1 - p_v)\Big), \quad \text{where} \quad p_v = \sigma(\mathbf{w}^T \mathbf{x}_v + b).$$

Table 2: Datasets used in the Experiments

Data	Users	Edges
`HateUser5K`	100,386	retweet edges: 2,286,592
`Tweet9K`	1,448	follower-followee edges: 3,471

5 Experiments

In this section, we conduct an empirical evaluation of our proposed method `HateGNN`, with the aim of answering the following research questions:

RQ1: How does `HateGNN` perform vs. the baselines for hateful users detection?
RQ2: How do the components of `HateGNN` (latent neighbourhood, bias-sampling, multi-modality of neighbourhood aggregation) affect the model performance?
RQ3: How do various parameters, e.g., the size of sampling neighbours set, the similarity, the node properties, impact performance of the model?

We compared `HateGNN` with five baselines, including `AdaBoost`, `GradBoost`, `GCN` [6], `GAT` [12] and `GraphSAGE` [4], and we evaluated their performances using two widely-used datasets in the hate speech domain, for which the statistics are summarized in Table 2.

`HateUser5K` contains a network of $100k$ users, out of which about $5k$ were annotated to be either hateful or not. Hateful users are those who endorse any type of hate speech (e.g., abusive language, discrimination, racism). Each user has several activity-related, content-related, and structure-related properties, as shown in Table 1. If one user has retweeted another users, such retweet connection is represented as the social graph in this dataset.

`Tweet9K` is the dataset of online tweets, which contains $16,907$ tweet IDs and their labels. It was collected from *Twitter* by [13], and was annotated as sexism, racism, both or neither by recruited experts. By using the *Tweepy* library, we retrieved the tweets and also collected the follower-followee information for the users as the edges in the social graph. Since some users have now been suspended, only $9,755$ tweets of $1,448$ users were acquired.

5.1 Comparisons with Baselines (RQ1)

We compare our model with other baselines (`AdaBoost, GradBoost, GCN, GAT, GraphSAGE`) in Table 3, and used `Accuracy`, `F1-score`, `Area under the ROC Curve (AUC)`, to evaluate the performance of classification. It is noteworthy that `HateGNN` outperforms all baselines on both datasets. In terms of accuracy, our model leads to a performance improvement of over 5 % on `HateUser5K`, and over 2 % on `Tweet9K`, which is very encouraging.

5.2 Performance Analysis (RQ2)

To answer **RQ2**, we design experiments to evaluate the efficacy of each component of `HateGNN`. The performance is reported in Table 4, where the best results are highlighted in bold.

Table 3: Comparison of `HateGNN` with baselines and state-of-the art `GraphSage`.

Dataset	Methods	Accuracy	F1-score	AUC
HateUser5K	AdaBoost	0.6894 ± 0.0132	0.3724 ± 0.0137	0.8499 ± 0.0182
	GradBoost	0.8389 ± 0.0104	0.5043 ± 0.0217	0.8768 ± 0.0086
	GCN	0.8543 ± 0.0254	0.5397 ± 0.0127	0.8716 ± 0.0376
	GAT	0.8643 ± 0.0302	0.4578 ± 0.0212	0.7900 ± 0.0197
	GraphSAGE	0.8904 ± 0.0372	0.6355 ± 0.0880	0.9392 ± 0.0334
	HateGNN	**0.9509 ± 0.0354**	**0.7987 ± 0.1385**	**0.9649 ± 0.0442**
Tweet9K	AdaBoost	0.4475 ± 0.0168	0.4891 ± 0.0112	0.7567 ± 0.0290
	GradBoost	0.7342 ± 0.0280	0.5919 ± 0.0404	0.7813 ± 0.0376
	GCN	0.6897 ± 0.1880	0.5048 ± 0.0314	0.7003 ± 0.0323
	GAT	0.7023 ± 0.0820	0.4991 ± 0.0124	0.6813 ± 0.0536
	GraphSAGE	0.8598 ± 0.0974	0.7889 ± 0.1302	0.9243 ± 0.0694
	HateGNN	**0.8715 ± 0.1052**	**0.8062 ± 0.1434**	**0.9244 ± 0.0843**

Table 4: The effects of each component of `HateGNN`.

Dataset	Graph	Methods	Accuracy	F1-score	AUC
HateUser5K	Social Graph	GraphSAGE	0.8922 ± 0.0339	0.6399 ± 0.0789	0.9396 ± 0.0346
		HateGNN	**0.9276 ± 0.0364**	**0.7264 ± 0.1196**	**0.9509 ± 0.0437**
	Latent Graph	GraphSAGE	0.8892 ± 0.0341	0.6337 ± 0.0787	0.9398 ± 0.0326
		HateGNN	**0.9237 ± 0.0408**	**0.7302 ± 0.1252**	**0.9570 ± 0.0336**
	Social + Latent	GraphSAGE	0.9008 ± 0.0356	0.6652 ± 0.0886	0.9458 ± 0.0339
		HateGNN	**0.9260 ± 0.0322**	**0.7204 ± 0.1031**	**0.9548 ± 0.0359**
Tweet9K	Social Graph	GraphSAGE	0.8345 ± 0.1128	0.7812 ± 0.1362	**0.9005 ± 0.0959**
		HateGNN	**0.8429 ± 0.1118**	**0.7856 ± 0.1439**	0.8938 ± 0.1044
	Latent Graph	GraphSAGE	0.8598 ± 0.0974	0.7889 ± 0.1302	**0.9243 ± 0.0694**
		HateGNN	**0.8626 ± 0.0974**	**0.7926 ± 0.1328**	0.9205 ± 0.0825
	Social + Latent	GraphSAGE	0.8660 ± 0.0970	0.7974 ± 0.1322	**0.9273 ± 0.0734**
		HateGNN	**0.8715 ± 0.1052**	**0.8062 ± 0.1434**	0.9244 ± 0.0843

The Efficacy of the Latent Neighbourhood. It can be seen from Table 4 by focusing on results reported individually on social graph and latent graph – that, the performance on latent graph is comparable with that on social graph. It is encouraging to see that the performance on latent graph is even better than that on the social graph of dataset `Tweet9K`, thus, demonstrating the efficacy of the latent connections for this problem. Please note that the individual reported results on social graph and latent graph, do not include biased sampling of the neighbours.

The Efficacy of the Biased Sampling Strategy. We conducted the experiments on social graph, latent graph and the (social+latent) graph, respectively. The results in Table 4 show that `HateGNN` generally outperforms `GraphSAGE`. This confirms that the biased sampling strategy helps learn the node embedding from the neighbours.

The Effect of Multi-modality of Neighbourhood Aggregation. According to Table 4, for `HateUser5K` dataset, the performances of `HateGNN` on the (social+latent) graph is slightly superior than the results on latent graph in `Accuracy` and on social graph in `AUC`. In terms of `Tweet9K`, `HateGNN` performs better on the (social+latent) graph than on individually latent graph or social graph. Moreover, `GraphSAGE` trained on the combination of the social and latent graph has a better performance than `GraphSAGE` on either on `HateUser5K` or `Tweet9K`, demonstrating the efficacy of multi-modality neighbourhood.

Table 5: The comparative analysis of the sampling size.

Methods	Sampling size	Accuracy	F1-score	AUC
GraphSAGE	S_1=5; S_2=1	0.8904 ± 0.0372	0.6355 ± 0.0880	0.9392 ± 0.0334
	S_1=10; S_2=5	0.8944 ± 0.0363	0.6448 ± 0.0875	0.9418 ± 0.035
	S_1=25; S_2=10	0.8936 ± 0.0365	0.6423 ± 0.0890	0.9427 ± 0.0340
	S_1=100; S_2=50	0.8968 ± 0.0357	**0.6516 ± 0.0879**	0.9440 ± 0.0349
	S_1=500; S_2=100	**0.8972 ± 0.03466**	0.6493 ± 0.0876	**0.9447 ± 0.0344**
HateGNN	S_1=5; S_2=1	**0.9260 ± 0.0322**	**0.7204 ± 0.1031**	**0.9548 ± 0.0359**
	S_1=10; S_2=5	0.9155 ± 0.0348	0.6983 ± 0.0970	0.9500 ± 0.0355
	S_1=25; S_2=10	0.9103 ± 0.0355	0.6844 ± 0.0980	0.9488 ± 0.0359
	S_1=100; S_2=50	0.9063 ± 0.0296	0.6729 ± 0.0768	0.9496 ± 0.0330
	S_1=500; S_2=100	0.9016 ± 0.0344	0.6599 ± 0.0889	0.9474 ± 0.0334

5.3 Parameter Analysis (RQ3)

How do various parameters, e.g., the size of sampling neighbours set, the similarity measure and the node properties impact the model performance? To answer **RQ3**, we discuss these questions in this section. Due to space constraints, we only present results on `HateUser5K` dataset, however, a similar pattern of results was observed on `Tweet9K`.

The Setting of the Sampling Size. In this section, we probe the influence of the size of sampling neighbours set on the model performance. [4] found that setting the depth of neighbourhood $K = 2$ provided a consistent boost in accuracy. Thus, we set the default value for K. Because of the memory limit and the run-time requirements, we only adjust the neighbourhood sample sizes S_1 and S_2 from $\{5, 1\}$ to $\{500, 100\}$. The results are presented in Table 5. For random sampling of `GraphSAGE`, increasing the neighbourhood sample size basically obtained no more than 1 % performance improvement. For biased sampling strategy of `HateGNN`, a small sampling size achieved the best performance, showing that learning node embedding from a small number of sampling neighbours is able to maintain promising results. It is encouraging to see that a small sample size for `HateGNN` leads to much better accuracy that was achieved with `GraphSage` with much larger sample.

The Similarity Measure. We have to use some forms of similarity measure to calculate the similarity among nodes, and then conduct latent graph construction and biased sampling. In this experiment, we compare two similarity measures, namely *Spearman* and *Pearson*, as presented in Table 6. Compared with random sampling, `HateGNN` model trained with *Spearman* or *Pearson* similarity measures, has much better performance. Secondly, *Pearson*-based similarity measure leads to better performance than *Spearman*. This is the reason, we present *Pearson* as the default option, and all the results presented in this work are based on *Pearson* measure.

The Effect of Similarity and Dissimilarity. In Section 5.2, we demonstrated the effectiveness of biased sampling. Here we discuss why we can not do biased

Table 6: The impact of various similarity approaches.

Neighbourhood	**Similarity**	**Accuracy**	**F1-score**	**AUC**
Social Neighbourhood	Random	0.8922 ± 0.0339	0.6399 ± 0.0789	0.9396 ± 0.0346
	Spearman	0.9163 ± 0.0335	0.7009 ± 0.0961	0.9505 ± 0.0361
	Pearson	0.9276 ± 0.0364	0.7264 ± 0.1196	0.9509 ± 0.0437
Latent Neighbourhood	Random	0.8892 ± 0.0341	0.6337 ± 0.0787	0.9398 ± 0.0326
	Spearman	0.9139 ± 0.0360	0.6965 ± 0.1063	0.9512 ± 0.0373
	Pearson	0.9237 ± 0.0408	0.7302 ± 0.1252	0.9570 ± 0.0336
Social + Latent	Random	0.8904 ± 0.0372	0.6355 ± 0.0880	0.9392 ± 0.0334
	Spearman	0.9177 ± 0.0349	0.6994 ± 0.1059	0.9509 ± 0.0377
	Pearson	0.9260 ± 0.0322	0.7204 ± 0.1031	0.9548 ± 0.0359

Table 7: The effect of the use of similarity or dissimilarity measure.

Bias	**Accuracy**	**F1-score**	**AUC**
Dissimilarity	0.8853 ± 0.0412	0.6313 ± 0.0951	0.9333 ± 0.0327
Random	0.8904 ± 0.0372	0.6355 ± 0.0880	0.9392 ± 0.0334
Similarity	**0.9260 ± 0.0322**	**0.7204 ± 0.1031**	**0.9548 ± 0.0359**

sampling with dissimilarity measures? In this experiment, we compared HateGNN model performance by biased sampling neighbours according to similarity and dissimilarity (Table 7). Dissimilarity-based sampling strategy is unable to achieve further performance improvement, even worse than GraphSAGE with random sampling. It is demonstrated that nodes embedding cannot benefit from the dissimilar neighbours in this task.

The Impacts of the Nodes' Properties. As collecting content-related and activity-related properties are relatively easy (i.e. only the user itself is involved), we have only utilized these two properties in this study. Here, we will explore the impact of more nodes' properties on HateGNN model. It can be seen from Table 8, that by adding activity-related property, the relative improvements are no more than 1 % in three evaluation metrics. But when using all properties, HateGNN model achieves further performance improvement, with the relative improvements are about 3 %, 9 %, 2 % in Accuracy, F1-score, AUC, respectively. This shows the benefit of utilizing more nodes' properties in the model for this task.

6 Conclusions

In this paper, we develop a sophisticated framework for hateful users detection in the social network – HateGNN, which not only exploits the explicit social graph, but also builds a latent graph. In addition, it has an effective neighbour sampling technique that can choose the most informative features from neighbours. On two standard hate-speech detection datasets, the proposed model leads to better performance than existing state of the art methods such as GraphSAGE, etc. In the future, we will investigate the application of HateGNN on even larger datasets, together with biased sampling weight learning based on multi-view node properties. It is important to note that formulation of HateGNN is general,

Table 8: The impact of nodes' properties

Property	Accuracy	F1-score	AUC
Content (300d)	0.9195 ± 0.0389	0.7084 ± 0.1204	0.9480 ± 0.0438
Content+Activity(320d)	0.9260 ± 0.0322	0.7204 ± 0.1031	0.9548 ± 0.0359
Content+Activity+ Sentiment+Structure(1028d)	**0.9509 ± 0.0354**	**0.7987 ± 0.1385**	**0.9649 ± 0.0442**

and though we have constrained ourselves to hate-speech detection problem in this work, the application of `HateGNN` to general graphs is straight-forward and is currently under-progress.

Acknowledgment

This work is supported by Scientific Research Guiding Project (Grant No. Y9W0013401), Key Technical Talents Project of CAS (Grant No. Y8YY041101) and National Natural Science Fund of China (Project No. 71871090).

References

1. Arango, A., Pérez, J., Poblete, B.: Hate Speech Detection is Not as Easy as You May Think: A Closer Look at Model Validation. In: COLING (2019)
2. Fortuna, P., Nunes, S.: A Survey on Automatic Detection of Hate Speech in Text. ACM Computing Surveys **51**(4), 1–30 (2018)
3. Georgakopoulos, S.V., Tasoulis, S.K., Vrahatis, A.G., Plagianakos, V.P.: Convolutional neural networks for toxic comment classification. In: Proceedings of the 10th Hellenic Conference on Artificial Intelligence, SETN (2018)
4. Hamilton, W.L., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: NIPS. pp. 1024–1034 (2017)
5. Hou, Y., Chen, H., Li, C., Cheng, J., Yang, M.C.: A Representation Learning Framework for Property Graphs. In: SIGKDD. pp. 65–73. ACM (Jul 2019)
6. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: ICLR (2016)
7. Kurita, K., Belova, A., Anastasopoulos, A.: Towards robust toxic content classification. CoRR (2019)
8. Mishra, P., Del Tredici, M., Yannakoudakis, H., Shutova, E.: Author Profiling for Abuse Detection. In: COLING (2018)
9. Petulla, S., Kupperman, T., Schneider, J.: Hate Crimes Spurned By Group-Based Hatred (2018)
10. Pitsilis, G.K., andHelge Langseth, H.R.: Effective hate-speech detection in twitter data using recurrent neural networks. Appl. Intell. **48**(12), 4730–4742 (2018)
11. Ribeiro, M.H., Calais, P.H., Santos, Y.A., Almeida, V.A., Meira Jr, W.: Characterizing and detecting hateful users on twitter. In: ICWSM (2018)
12. Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. In: ICLR (2018)
13. Waseem, Z., Hovy, D.: Hateful symbols or hateful people? predictive features for hate speech detection on twitter. In: SRW@HLT-NAACL. pp. 88–93 (2016)
14. Zhou, P., Qi, Z., Zheng, S., Xu, J., Bao, H., Xu, B.: Text classification improved by integrating bidirectional LSTM with two-dimensional max pooling. In: COLING (2016)