Exploring Graph Heterogeneity for Semi-supervised Twitter Bot Detection

Kareem Alaa Abdellatif Hussein

Master of Science

Computer Science

School of Informatics

University of Edinburgh

2025

Abstract

Social bot activity on Twitter is a well-documented phenomenon with dangerous effects. Not only have they successfully manipulated users by disguising themselves as humans and influencing public discourse, they have also become increasingly effective at evading existing bot detection methods. To counter this, recently developed approaches utilise graph neural networks, attempting to capture the users' interactions with their communities alongside their individual features. However, despite the diverse nature of user interactions on Twitter, existing methods only utilise user-formed graphs with follow relationships, failing to encode more powerful heterogeneous semantics capturable through the inclusion of additional entities and relationships. Therefore, this project investigates the effect of these graph expansions on the classification performance of graph-based twitter bot detection systems, further evaluating the effects of including explicit node and edge type mappings. In addition, the relationships between these graph structures and the employed graph neural network architectures are also examined. Our results show that the observed performance effects are model-dependent; while one of our models benefits significantly from the addition of tweet nodes and relevant edges to the user-formed graphs, no such effect is observed for our second model. Furthermore, we find that the explicit encoding of the node and edge type mappings has no significant effect on the models' bot detection performance in isolation despite being a widely utilised step in existing work on heterogeneous graphs.

Research Ethics Approval

This project obtained approval from the Informatics Research Ethics committee.

Ethics application number: 619695
Date when approval was obtained: 2022-07-27

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Kareem Alaa Abdellatif Hussein*)

Acknowledgements

First and foremost, I would like to thank my parents for their constant love and support throughout university and during this degree, which without them would not have been possible.

I would also like to thank my supervisors at Amazon: John, Pawel and Tania. Their consistent guidance and advice shaped the directions of my work, and kept me on track in the many instances when I needed their input.

I would also like to thank my internal supervisor, Dr. Iain Murray, for facilitating this opportunity, offering guidance for my ethics application process and for granting access to the ECDF Eddie research cluster.

I would finally like to thank my friends for the constant support and all the memories we've made throughout the past year. It's been a wild ride and I couldn't be happier to have met you all. Special mention goes out to Jag, Sanjana, Ciara and Omar for always being there when I wanted to vent my frustrations.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

The prevalence of social media usage has exploded over the last decade; a recent report stated that the total number of users has surpassed 4.5 billion globally, with an average usage time of over 2.5 hours daily [1]. This has resulted in a widespread societal impact ranging from consumer behaviours [2] to political discourse [3] as well as adolescents' development and upbringing [4]. However, this has resulted in a variety of methods through which malicious actors could manipulate users, chief among which is the use of online bots [5].

The phenomenon of online bot activity has been studied for well over a decade - with particular research focus on the Twitter platform [6, 7, 8]. Not only have they been used for the dissemination of spam content, malware and phishing links, a more recent development has seen the use of more sophisticated "social bots", which - in a sense - have long surpassed the Turing Test [7]. These bots disguise themselves as human users, influencing political discourse on a variety of topics by leveraging their follower-base to spread misleading ideas through biased or otherwise low-quality news sources [6, 7]. This has been utilised by malicious actors to influence discourse around elections [9, 10, 11], peoples' responses to public health measures, and many other aspects of their lives [12, 13]. Furthermore, the sophistication of these bots, including their ability to evade detection, has continually increased over time, leaving the research community in an active arms race requiring increasingly effective detection methods [14, 15, 16].

Towards this end, a variety of approaches have been developed to distinguish between "human" and "bot" users. These have developed from methods utilising

classical Machine Learning (ML) [6, 17] models such as Support Vector Machines (SVMs) and random forests to more sophisticated approaches using deep learning [18, 19]. Among the latter, the most successful methods leveraged the patterns hidden in the users' interactions with their communities [20, 21], which have been found to be a differentiating pattern between human and bot users online [22, 23]. This property is effectively represented by social graphs, on which Graph Neural Networks (GNNs) have proven to be particularly effective in a variety of applications [24, 25]. These graphs represent the different entities on a social network and the relationships between them in the form of nodes and edges respectively [26].

In many social networks, these entities and relationships are not uniform [27, 28]; they may encode different semantic information depending on the type of entities or interactions between them [29]. Furthermore, the relationships signified by these different kind of interactions have also shown to hold different semantic significance [26]. In fact, the knowledge and encoding of these different types of entities and relationships into a graph, also known as *graph heterogeneity*, has been leveraged in many other graph-based tasks across literature to improve classification effectiveness [30, 31, 32], where a special class of *heterogeneous* GNNs exploit these enriched semantic representations in the process [33, 34, 35, 36].

Despite this, existing applications of heterogeneity for the twitter bot detection task are quite limited, having only been utilised on user-formed graphs in a few instances [21, 37]. While these approaches have successfully utilised these semantics to set state-of-the art performances, their approaches only utilise *edge heterogeneity*, wherein different edge types are represented but no such mapping exists for nodes. Furthermore, the attributability of the heterogeneity to the prediction performance of these models is also unclear; with many other aspects being different in these pipelines in comparison to other attempts.

1.2 Research Questions

Since Twitter social graphs have been shown to encode larger varieties of entities and interactions than those utilised in the bot detection task so far [27, 6] and since the incorporation of this graph heterogeneity has been shown to improve detection performance in other tasks, it is therefore beneficial to investigate the following questions:

1. How does expanding the set of represented entities and relationships in a social

graph impact graph-based twitter bot detection accuracy?

2. Similarly, what further effects are observed when directly encoding the representations for these different types into the graph using node and edge heterogeneity?

3. Do these effects change when evaluated using a model that is designed for these heterogeneous representations?

1.3 Contributions

To answer these questions, we create multiple different graph representations of the recent Twibot-20 [16] twitter bot detection dataset. We define graphs with two different sets of encoded entities: user-only graphs and graphs with users and tweets. For the latter, we also expand the set of encoded relationships to include user-tweet interaction in addition to the existing relationships between users. We create homogeneous, edge-heterogeneous and fully heterogeneous graph variations for each set of entities and utilise the resultant graphs. We then evaluate two different twitter bot detection pipelines on these graphs, including a modified version of a state-of-the-art pipeline [21], which we name TweetAugmentedRGCN, and a new pipeline employing a heterogeneous GNN model [35] - named MinibatchedAugHAN.

Through our comparison of the classification performances of these models across the different dataset variations, we find that the effects of expanding the sets of entities and relationships is model-dependent: while our modifications yielded no significant effect for TweetAugmentedRGCN, they resulted in a significant performance improvement for MinibatchedAugHAN. We argue that this is due to the model incorporating the presence of different types of entities and relationships in the graph into its design. On the other hand, we find that while encoding the node and edge type mappings into the graph does not cause any performance effects in isolation, it makes the implementation of pipelines utilising these heterogeneous graphs more straightforward.

1.4 Report Structure

The remainder of this report aims to outline our contributions in further detail. We first introduce the reader to key concepts and related work in chapter 2 that are necessary for gaining better context and understanding of our work. We then describe our methodology in chapter 3, describing our dataset and graph modifications as well as the employed

models and pipelines. We then outline our experimental setup, results and discussion in chapter 4, including our main observations and findings. We then conclude our report in chapter 5, providing a summary of our work as well as recommendations for future work which may build on top of our contributions.

Chapter 2

Background & Related Work

2.1 Background

2.1.1 Spectral Graph Theory & Notation

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a data structure encoding nodes $\mathcal{V} \in \mathbb{R}^N$ and edges $\mathcal{E} = \{(v_1, v_2) | v_1, v_2 \in \mathcal{V}\}$, where N is the number of nodes in the graph. Graphs can be directed or undirected, where undirected graphs have bi-directional edges that may be represented as $\{v_1, v_2\}$ [38, 39]. A node's neighbourhood $\mathcal{N}(v_i)$ is defined as the set of nodes connected to v_i, where neighbourhoods can be defined in undirected graphs as the set $\{v_j | \{v_i, v_j\} \in \mathcal{E}\}$. In directed graphs, a further distinction can be made between out-neighbours and in-neighbours, which are defined as $\{v_j | (v_i, v_j) \in \mathcal{E}\}$ and $\{v_j | (v_j, v_i) \in \mathcal{E}\}$ respectively. An edge (v_j, v_i) is also commonly represented as e_{ij} [38].

The structure of a graph can be represented by an adjacency matrix $\mathcal{A} \in \mathbb{R}^{N,N}$, where a 1 at $\mathcal{A}_{i,j}$ represents an edge (v_i, v_j). In undirected graphs, $\mathcal{A}$ is symmetrical [40]. A graph's degree matrix $\mathcal{D} \in \mathbb{R}^{N \times N}$ is a diagonal matrix of each node's degree, which is the number of edges the node is involved in. Entries can be mathematically represented as $\mathcal{D}_{i,i} = \sum_{j=0}^{N} \mathcal{A}_{i,j}$. A graph's laplacian matrix $L = \mathcal{D} - \mathcal{A}$ encodes various useful representations of a graph's structure through its eigenvectors, and has a normalised form $\mathcal{L} = \mathcal{D}^{-\frac{1}{2}} L \mathcal{D}^{-\frac{1}{2}}$ [38].

In a heterogeneous graph, both nodes and edges are mapped to discrete "types", where $\phi : \mathcal{V} \to \mathcal{T}_{\mathcal{V}}$ and $\psi : \mathcal{E} \to \mathcal{T}_{\mathcal{E}}$ denote the respective mapping functions. $\mathcal{T}_{\mathcal{V}}$ and $\mathcal{T}_{\mathcal{E}}$ are predefined sets of types for nodes and edges respectively. We denote homogeneous graphs as graphs where $|\mathcal{T}_{\mathcal{V}}| = 1$ and $|\mathcal{T}_{\mathcal{E}}| = 1$, edge-heterogeneous graphs as graphs

where $|\mathcal{T}_{\mathcal{V}}| = 1$ and $|\mathcal{T}_{\mathcal{E}}| > 1$, and fully heterogeneous graphs where $|\mathcal{T}_{\mathcal{V}}| > 1$ and $|\mathcal{T}_{\mathcal{E}}| > 1$ [35].

Meta-paths in heterogeneous graphs are composite relations linking two nodes indirectly through edges with a specific sequence of edge types[41, 35]. More formally, a meta-path Φ_p can be denoted as a tuple of edge types $(t_e^1, t_e^2, \ldots, t_e^M)$ where $t_e^m \in \mathcal{T}_{\mathcal{E}}, m \in \{1..M\}$. A sequence of edges between $v_i, v_j \in \mathcal{V}$ can be said to belong to Φ_p if:

$$\exists \mathcal{E}_{\Phi_p} = \{e_{ik}, ..., e_{lj}\} \subseteq \mathcal{E} s.t. \psi(e_{ik}) = t_e^1, \ldots, \psi(e_{lj}) = t_e^M, |\mathcal{E}_{\Phi_p}| = |\Phi_p| \tag{2.1}$$

For a meta-path Φ_p connecting v_j to v_i, v_j is considered to be in v_i's meta-path based neighbourhood for meta-path Φ_p, denoted as $\mathcal{N}^{\Phi_p}(v_i)$, which can be shortened as $\mathcal{N}_i^{\Phi_p}$ [41, 35].

2.1.2 Graph Neural Networks

Graph neural networks (GNNs) are a specialised class of neural networks which operate over graph-structured data, leveraging both node and topological features in a message-passing paradigm for a variety of tasks such as node classification, link prediction and graph classification [42, 43, 44]. While early forms were modifications of recurrent neural networks [45, 44], popular forms utilise graph convolutions [46, 47, 42] to propagate information over node neighbourhoods; these have been introduced in both spectral terms [47, 42] as well as directly over the nodes in the graph [46, 24].

Of the former, a notable milestone was achieved upon the introduction of the Graph Convolutional Network (GCN) architecture [42], which improved upon the scalability of previous attempts [46, 47] by defining graph convolutions over undirected graphs in terms of stackable individualised feature propagating layers - each being linear with respect to the graph's normalised Laplacian [42]. The resultant layer-wise propagation rule is shown below in equation 2.2 - is adapted from [42] - where $H^{(l)} \in \mathbb{R}^{N,F}$ denotes the entire graph's embeddings at layer l - F being the embedding size of each node. $\sigma(\cdot)$ indicates a non-linearity function, for which they used ReLU [48].

$$H^{(l+1)} = \sigma(\tilde{\mathcal{D}}^{-\frac{1}{2}} \tilde{\mathcal{A}} \tilde{\mathcal{D}}^{-\frac{1}{2}} H^{(l)} W^{(l)}) \tag{2.2}$$

In this equation, a renormalization trick is applied to control vanishing/exploding gradients, forming the term $\tilde{\mathcal{D}}^{-\frac{1}{2}} \tilde{\mathcal{A}} \tilde{\mathcal{D}}^{-\frac{1}{2}}$, which is a normalisation of the adjacency

matrix with added self-connections. In this formula, $\tilde{\mathcal{A}} = I_N + A$ and $\tilde{\mathcal{D}}$ is the corresponding degree matrix [42]. By "dividing" the adjacency matrix by the degree matrix, the result of multiplying this term with $H^{(l)}$ is a matrix where each node's embedding is the average of the previous layer's embeddings of itself and its neighbours.

However, by requiring knowledge of the graph's entire adjacency and degree matrices during the forward propagation step, the model cannot be utilised for mini-batched training, nor for inductive contexts, where test nodes are part of a different sub-graph from that of the training nodes in its proposed form. This was solved by Hamilton *et al.*upon the introduction of GraphSAGE [24], which defines convolutions directly over the graph by sampling a fixed-sized neighbourhood for each node for each layer of the model and learning a set of aggregator functions that can be applied over the neighbourhoods of any nodes encountered during inference. While they created a GCN-equivalent version of that model, named GraphSAGE-GCN, using a simple averaging function as the aggregator, they created two main variants: a pooling variant which passes each neighbouring node through a multi-layer perceptron (MLP) [49] and then takes the element-wise maximum, and a variant which uses an LSTM [50] over the matrix of the neighbouring nodes' embeddings [24].

2.1.3 Heterogeneity in Graph Neural Networks

However, the models discussed above only capture homogeneous graph semantics [33]. To leverage the richer semantics encoded in heterogeneous graphs, a heterogeneous class of neural networks was developed for a variety of tasks [33, 34, 35]. The first such attempt adapted GCNs for edge-heterogeneous graphs, resulting in the Relational-GCN (RGCN) model [33], which was created for knowledge base completion tasks. RGCN extends GCN by defining a different trainable weight matrix for each edge type, where they refer to edge types as "relations". They perform the weighted averaging first across individual neighbourhood types before summing the results from the different neighbourhoods together to produce the output embedding. The resultant layer-wise propagation rule is defined as follows in equation 2.3 (adapted from [33]), where $h_i^{(l)}$ describes the embedding for node i at layer l, $\mathcal{N}_i^{t_e}$ describes node i's neighbourhood under edge type t_e, and c_{i,t_e} is a normalisation constant such as $|\mathcal{N}_i^{t_e}|$. To prevent parameter explosion with increasing edge type counts, they also implement parameter sharing modifications to their model structure [33], making their model relatively efficient.

$$h^{(l+1)} = \sigma\Big(\sum_{t_e \in \mathcal{T}_e} \sum_{j \in \mathcal{N}_i^{t_e}} \frac{1}{c_{i,t_e}} W_{t_e}^{(l)} h_j^{(l)} + W_0^{(l)} h_i^{(l)}\Big) \tag{2.3}$$

While the model discussed above proved to be successful in a variety of edge-heterogeneous contexts [51, 21, 52, 53], it is not applicable to fully heterogeneous graphs. Our chosen model, the Heterogeneous Graph Attention Network (HAN) [35], is one such model which incorporates both node and edge heterogeneity.

HAN is based on the Graph Attention Network (GAT) model [43], which utilises masked self-attention [54] to implicitly learn different weights for a node's different neighbours. Since the attention mechanism can be applied to a variable number of input nodes, this also removed the necessity of the fixed-sized neighbourhood sampling step introduced for GraphSAGE [43]. HAN builds on this by using hierarchical attention over meta-path based neighbours through node-level and semantic-level attentions, which they define as attention over neighbours from the same meta-path and attention over the different meta-paths respectively [35]. A visualisation of this can be seen in figure 2.1.

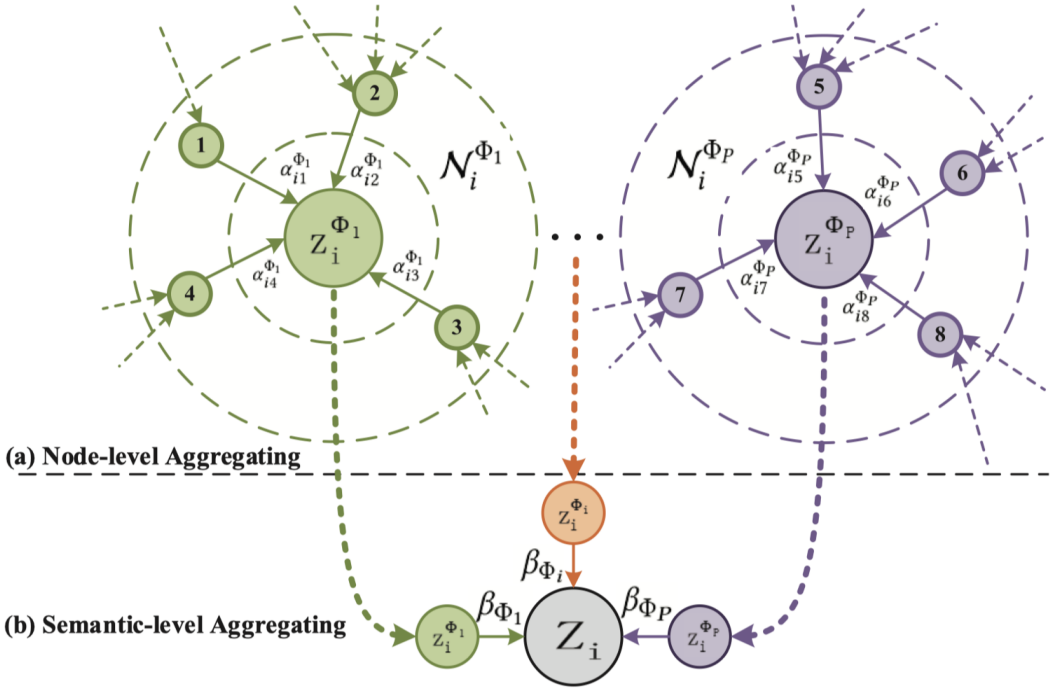


Figure 2.1: HAN hierarchical attention application to generate the output for node i, adapted from [35]

As shown in figure 2.1, to calculate the HAN layer output embedding for node i, denoted z_i, a semantic-specific embedding $z_i^{\Phi_p}$ is first calculated for node i for each meta-

path Φ_p in the graph's set of P meta-paths. These are in turn calculated by attending over the nodes in the respective meta-path based neighbourhood $\mathcal{N}_i^{\Phi_p}$, as shown by phase **(a)**. The model then aggregates these different semantic-level embeddings in phase **(b)** using a weighted sum, utilising meta-path attention coefficients β_{Φ_p} as the respective weights [35].

2.2 Related Work

To provide more context for our work, we discuss related work in the area, particularly focusing on twitter bot detection literature. We first outline some earlier work which went into detecting spam bots in the wild and classifying them using traditional ML methods [8, 55, 6]. We then discuss the phenomenon of social bots [7], outlining the differences between them and traditional spam bots and discussing the newer deep learning-based detection approaches developed to counter them [18, 19, 37]. Finally, we discuss the newer GNN-based approaches within these, including the state-of-the-art BotRGCN pipeline, which we adapt in our work [20, 21].

2.2.1 Early Twitter Spam Bot Detection

As introduced in section 1.1, detecting and studying twitter bots has received research attention for well over a decade. In earlier work, research was predominantly focused on spam bots; a notable contribution by Lee *et al.*[8] performed the first long term study of bot activity, setting up 60 "social honeypot" [55] accounts designed to attract spam bots, then tracking any accounts which engaged with these honeypots. By analysing and tracking over 36,000 accounts over 7 months, they found distinguishing behavioural patterns such as a large number of user mentions and links per tweet, as well as an unusually large number of accounts followed [8]. They then used their top 10 features to train a random forest [56] model to classify twitter accounts. In a similar vein, other early spam bot-oriented methods utilised off-the-shelf classical machine learning models such as support vector machines [57, 6], focusing their work on the features that were fed into those models [58].

2.2.2 Social Bot Detection

While these models proved to be successful at detecting spam bots with high precision, a paradigm shift was observed over the following years in the nature of the bots active

on twitter, where a new class of more sinister "social bots" rose in prominence [7, 15]. Rather than engaging in the spamming of malicious phishing links to users [5], malicious actors operating these bots attempt to fully disguise them as real humans, using stolen photos, fake bios and locations, engaging with real communities and "making friends" on the platform [7]. They then attempt to influence the public discourse around a variety of topics both within their follower base as well as through engagement with trending events and communities, doing so by generating traction for biased and low quality articles and arguments [7, 15]. These tactics have consequently been employed to influence elections [9, 11] and referendums [23, 10], manipulate the discussion around health measures such as vaccination mandates [12] and COVID-19 lockdowns [13].

Furthermore, it was found that this new class of social bots was adept at evading the suspicions of both the existing bot detection frameworks as well as human users, where a crowd-sourced experiment found that humans who were able to detect traditional spam bots at a 91% accuracy averaged a 24% accuracy on a set of these social bots[15]. The data used in this study was then released as the cresci-17 dataset [15]. Therefore, subsequent twitter bot detection pipelines made use of larger models as well as richer representations [18, 19, 20, 21], performing their evaluation on more up to date datasets [15, 59, 16].

Among these, Sayyadiharikandeh *et al.*employed an ensemble of specialised classifiers over a larger, more diverse feature set to improve upon the performances of previous approaches, deploying their model in an online tool named Botometer, which can be used to query the bot status of any public twitter account [60]. In different approaches, Kudugunta *et al.*and Wei *et al.*both utilised LSTM-based [50] deep neural network architectures on both aggregated tweet content and account metadata to achieve stronger performances - the latter pairing this with pretrained GloVe [61] word embeddings for the content representation [18, 19].

2.2.3 GNN-based Approaches

However, while these richer representations somewhat improved the performance of detection methods on social bots, more recent approaches have paired a selection of richer account metadata-based features [59, 20, 21] with more structural representations of the users' interactions, mainly through GNNs [20, 21, 37]. Of these, we particularly focus on BotRGCN, which sets state-of-the-art performance on the recent Twibot-20 dataset [16] - conducting their evaluation against many of the approaches mentioned

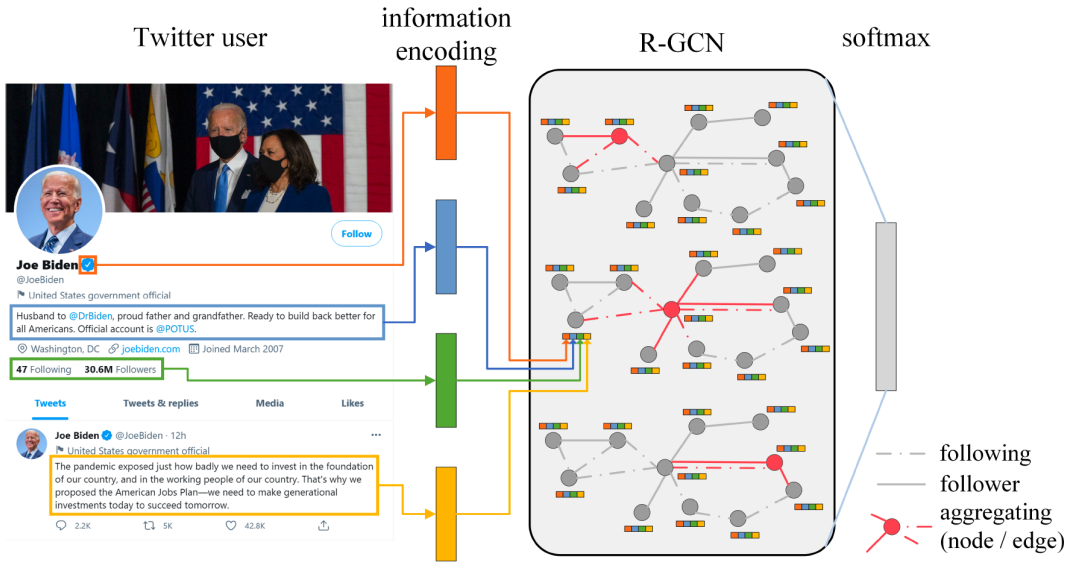


Figure 2.2: BotRGCN Embeddings & Graph: Categorical metadata, user description embedding, numerical metadata and averaged tweet embeddings are indicated by the orange, blue, green and yellow elements respectively. Adapted from [21].

above [21]. An in-depth description of Twibot-20 can be found below in section 3.1.

This approach utilises the RGCN model - introduced above in 2.1.3 - over an edge-heterogeneous graph formed entirely of user nodes with 2 types of unidirectional edges: "follower" and "following". They employ multi-modal node representations formed by concatenating learnt embeddings representing account metadata with pre-trained text content embeddings. For the latter, they employ a pre-trained RoBERTa language model [62], which has been successfully used to extract powerful text representations across a large variety of tasks in NLP [63, 64, 65].

The content embeddings are split into user description and tweet embeddings, where the latter is formed by averaging the individual tweets' RoBERTa embeddings. Similarly, the account metadata is split into two components corresponding to numerical and categorical features respectively, where the former features are mean-centered and scaled [66] in their preprocessing. Before concatenating the four components to form the user embedding, each of these is passed through a fully connected layer [67] with LeakyReLU [68] non-linearity to form the respective learnt embedding. This process is better visualised in figure 2.2, where the "information encoding" step represents the processing of the different components through the respective fully connected layers [21].

In their work, Feng *et al.*also performed an ablation study which showed that the

stronger performance was obtained as more of their components were added to the user embeddings. This includes the employed selection of the numerical and categorical features, which are listed in tables 2.1 and 2.2 respectively.

As stated above, we adapt this pipeline in our work, using it as the foundation for our TweetAugmentedRGCN pipeline, and sharing many components in our MiniBatchedAugHAN pipeline as well. Accordingly, a detailed description of our work and modifications can be found below in chapter 3.

Feature Name	Description
#followers	number of followers
#followings	number of followings
#favorites	number of likes
#statuses	number of statuses
active days	number of active days
screen name length	screen name character count

Table 2.1: Numerical features utilised by Feng *et al.*in the BotRGCN pipeline. Adapted from [21]

Feature Name	Description
protected	protected or not
geo enabled	enable geo-location or not
verified	verified or not
contributors enabled	enable contributors or not
is translator	translator or not
is translation enabled	translation or not
profile background tile	the background tile profile user
background image	have background image or not
has extended profile	has extended profile or not
default profile	the default profile
default profile image	the default profile image

Table 2.2: Categorical features utilised by Feng *et al.*in the BotRGCN pipeline. Adapted from [21]

Chapter 3

Methodology

Having discussed relevant areas in the GNN and Twitter Bot detection research spaces in chapter 2 - including the methods and models we utilise in our work - this chapter outlines the design and implementation of our graphs and pipelines. We begin by describing our dataset - Twibot-20 [16] - in depth, discussing the available data, the user metadata as well as how both our graph structures and node embeddings were created. We then describe our employed models and their application to our data. Finally, we list the technologies and resources employed for the implementation of our design.

3.1 Dataset

3.1.1 The Twibot-20 Dataset

As stated above in 2.2.3, we utilise the Twibot-20 dataset for our task, a semi-supervised bot detection dataset containing 229,580 users, of which 11,826 are labelled. Of the latter, 6,589 are labelled as bots; meanwhile 5237 are labelled as users. The labelling was conducted using a crowd-sourcing experiment, which might have introduced some noise and false positives in the labels [69, 70]. However, at the time of the development of this project, this was the largest and most up to date labelled twitter bot dataset available, although some more recent developments in the landscape are discussed in 4.3. Possible labelling biases aside, this results in a relatively balanced class distribution within the labelled set of users, with 55.72% bot and 44.28% human users.

The dataset was collected by Feng *et al.*using a breadth-first search (BFS) of follower graphs on twitter, starting from a set of 40 different popular seed users across different topics such as sports, entertainment and politics. This thus allows for the representation

of a subset of users participating in the discussion around different popular topics in the twittersphere, reducing any biases that may be specific to certain discussion topics. The web-scraping of this data was completed using Twitter API [71] calls over the collected user objects. The raw JSON [72] objects resultant from these API calls form the raw data in the dataset, which we obtained directly from the authors.

3.1.2 Extracted Graphs' Structural Properties

As introduced in 1.3, we conduct our experiments on multiple different graph variations extracted from Twibot-20. For the creation of the shared elements in these graphs, we adapt much of the work utilised by Feng *et al.*in BotRGCN [21] including aspects of the node embeddings as well as the chosen relationships represented by a subset of our edges. We reason that this allows for the presence of an empirically tested base structure from which our modifications are made.

However, some changes were made in order to arrive at our graphs due to various considerations. This included dropping the "support" set of unlabelled users, which is done both due to computational limitations and to accommodate the graph size expansion from introducing the additional nodes. For similar computational reasons, we utilise less resource-intensive content embeddings - described below in section 3.1.3.

The expansion mentioned above pertains to the introduction of the individualised tweet nodes alongside the existing users' nodes into some of our graphs; the embeddings for these are described below in section 3.1.3. We also extract a variety of different edge types as described below in 3.1.4 to link these tweet nodes to the user nodes and vice versa. For this modified graph, we introduce homogeneous, edge-heterogeneous and fully heterogeneous variations, as defined in section 2.1.1. In addition to these, we also maintain homogeneous and edge-heterogeneous versions of the user-only graphs. The list of resultant graphs, including their assigned names, are displayed in table 3.1.

Dataset Name	Tweet Nodes	Heterogeneous Edges	Heterogeneous Nodes
U-Homo	✗	✗	✗
U-EdgeHetero	✗	✓	✗
UT-Homo	✓	✗	✗
UT-EdgeHetero	✓	✓	✗
UT-FullHetero	✓	✓	✓

Table 3.1: The Names and Properties of our Employed Graphs

In the versions where the tweet nodes are extracted, the graph's size is increased to 2,011,614 nodes, consisting of the 11,826 labelled user nodes and 1,999,788 tweet nodes. While the resultant ratio of tweet to user nodes in this graph is relatively large, we reason that a heterogeneous model, such as HAN - or to an extent, RGCN - would be able to learn to place different significance on the different relationships so that useful representations are extracted.

Having described the general structural properties of our graphs, we describe the user and tweet node embeddings as well as the relationships represented by the different edges below.

3.1.3 Our Node Embeddings

As introduced above, we employ many aspects of the work undertaken in BotRGCN for the formation of our embeddings. In fact, for our user-only graphs, we utilise the same user node embeddings employed in their approach as described in section 2.2.3, with the exception of the employed method for generating the content embeddings from the raw text, where for computational reasons, we utilise a different approach described below.

However, we modify our user node embeddings in the user-tweet graph by removing the averaged tweet embedding component, instead keeping the individual tweet embeddings as the feature vectors for the tweet nodes. To extract the tweet nodes, we use the list of the latest 200 tweets included in the dataset per user, generating individual term frequency-inverse document frequency (TF-IDF) vectors [73], whose dimensionality is reduced using truncated Singular Value Decomposition (SVD) via scikit-learn [74, 75]. These embeddings were used in place of the original pipeline's RoBERTa [62] embeddings due to computational limitations.

The number of SVD components is kept consistent across the tweet and user description embeddings, and is left as a hyperparameter that we adjust during hyperparameter optimisation. The tweet embeddings fed into the GNN are the output of feeding the content embedding through a dense layer with an output dimensionality equivalent to the total embedding size used for the user nodes.

We keep these dimensions consistent in all experiments, including those where the fully heterogeneous graphs are used for the model. This is done to minimise the number of variables in the pipeline altered between the different dataset configurations - isolating the effect of introducing the graph structure modifications. This is also

necessary due to the model requiring uniform embedding sizes across all nodes in configurations where the model is not aware of the heterogeneity in the node types. To achieve this however, it was necessary to zero-pad the tweet node embeddings so that their embedding size matches up to that of the user nodes. The implications of this are outlined in section 4.3.

3.1.4 Edge Extraction

To extract our edges, we use the existing set of follower and following relationships encoded in the original Twibot-20 dataset well as 3 additional types of edges between users and tweets. We construct a user-tweet edge to represent a user writing a tweet, and different tweet-user edges representing mentions and retweets. While the extraction of the former edges is trivial due to the raw Twibot-20 JSON data embedding users' lists of most recent 200 tweets in their respective objects, some more work went into the extraction of the other edge types.

For these, we exploit the uniqueness of twitter usernames and the uniformity of retweet and mention representations in tweet text to extract the relevant edges using regular expressions. More specifically, we denote a string containing "RT @[username]" to indicate an edge from the tweet where that sub-string is found to the relevant user node - "[username]" being a placeholder for the relevant user's username. Meanwhile, mentions are denoted by "@[username]" sub-strings, where we utilise look-behinds in our regular expressions to ensure matches for retweets are not accidentally considered mentions and vice versa. The utilised regular expressions can be found in appendix A.

By introducing these different edges, both from users to their tweets then from those tweets to users who could have been mentioned or retweeted, we introduce meta-paths between users and the users they mention and retweet respectively. This would allow for a distinction between these different relationships and meta-paths, which an attention-based model - such as HAN - could learn to distinguish between and weigh accordingly. In addition to the directly encoded follower and following relationships for the users, this introduces additional logical relationships that would link different users together.

3.2 Models & Pipelines

As introduced earlier in section 1.3, we investigate the effects of our graph modifications on the twitter bot detection accuracy using two different models. The first of these

is based on the existing BotRGCN model utilised by Feng *et al.*[21]; meanwhile the second pipeline uses the HAN model, which is introduced in section 2.1.3. Outlined below are the respective processing steps applied to our datasets in each of the pipelines.

3.2.1 TweetAugmentedRGCN

We create this model as a closely matching counterpart to BotRGCN that can operate on our modified user-tweet graphs so that their results can be directly compared, with BotRGCN being utilised on our user-only graphs. We find the development of this model to be necessary since the existing BotRGCN model implementation undertakes the generation of the learnt embeddings and subsequent concatenation as a part of the model structure, requiring the presence of all 4 user embedding components. Accordingly, a forward pass through our model is described below:

The model first generates learnt embeddings for each of the remaining components: numerical properties, categorical properties and the user description. We set the output size for the description embedding to be half the size of the total node embedding, then we set the output size of each of the numerical property and categorical property layers to be a quarter of the size of the embedding.

For the tweet embedding, we also pass them through a dense layer with output size equal to the node embedding size, a value which is left as a hyperparameter to optimise. Matching BotRGCN, we also apply an extra dense layer over the entire input to form a part of the nodes' learnt embeddings, which are in turn passed into our RGCN block. The RGCN block consists of 2 RGCN feature propagation layers with Dropout and LeakyReLU non-linearity in between them, matching the architecture used in BotRGCN, as well as the optimal GCN layer count beyond which over-smoothing occurs - as discussed elsewhere in literature [76]. Finally, the RGCN block output is passed through two fully connected layers with output sizes equal to the embedding size and number of output classes respectively - the latter being 2 in this task.

While we are able to successfully employ this model over most of our user-tweet graphs, it does not capture fully heterogeneous semantics in graphs due to the employed RGCN model not incorporating information regarding node type mappings in its propagation rules. To include that version of the graph in our work, we develop the MinibatchedAugHAN pipeline described below.

3.2.2 MinibatchedAugHAN

In order to better exploit the additional semantic properties encoded in the modified graphs, this pipeline was also utilised for our experimentation. It is also used in order to verify the validity of the results obtained from the original pipeline, ensuring that performance changes aren't due to the properties of specific architectures. For this pipeline, the Heterogeneous Graph Attention Network (HAN) is utilised, which has been also introduced in section 2.1.3. We reason that due to the exploitation of meta-paths in the network - as well as the use of hierarchical attention across the different nodes and meta-paths -, this model should be powerful enough to learn the latent features that have been introduced in our graph modifications - particularly in the fully heterogeneous user-tweet graph.

One notable differentiating factor between this pipeline and the TweetAugmentedRGCN pipeline is the use of neighbourhood sampling for the implementation of mini-batching for the training of the graph - as some computational limitations were encountered upon attempting full graph training using this model. For our experiments, we sample a uniform number of neighbours of each node type per node at each depth, sampling a neighbourhood depth equal to the number of HAN layers used on our model - a hyperparameter that is optimised.

The minibatch tensors resultant from this process are then given an identical treatment to the TweetAugmentedRGCN pipeline to generate the learnt embeddings for the graph-based component of the pipeline in all node-homogeneous contexts. In that pipeline, the user and tweet tensors form separate inputs to the model, so we reason that it is helpful to use a separate fully connected layer for the tweets' learnt embeddings in place of the approach utilised above which shares the layer's weights across all nodes.

The learnt embeddings are then passed into a HAN-based block, also utilising LeakyReLU non-linearity and dropout to the outputs. The number of HAN layers utilised in this block is kept as a hyperparameter which we optimise as described in section 4.1.2. The user and tweet outputs are then passed through 2 separate respective linear layers with a LeakyReLU activation in between them. In both models, the raw values resultant from the last layer are used as the output as required by the PyTorch implementation of our loss function. To summarise the features of our pipelines, we list out a comparison in table 3.2.

Feature	BotRGCN [21]	TA-RGCN	MA-HAN
User-only graphs	✓	✗	✓
User-Tweet graphs	✗	✓	✓
Edge Heterogeneity	✓	✓	✓
Node Heterogeneity	✗	✗	✓
Neighbourhood Sampling	✗	✗	✓

Table 3.2: The employed pipelines' features in comparison to BotRGCN [21]. TA-RGCN and MA-HAN correspond to TweetAugmentedRGCN and MinibatchedAugHAN respectively

3.3 Technologies and Libraries

The above components were developed entirely using the python programming language [77], utilising the broad range of scientific computing, data processing and deep learning libraries available. To construct the graph dataset variations, we utilise the pandas [78], numpy [79] and scikit-learn [75] libraries. For our model training, construction and evaluation, we utilise the Pytorch-Geometric [80] GNN library, which in turn operates over the PyTorch [81] framework. We also directly utilise resources from the latter framework in our work, such as our loss and activation functions. Finally, we utilise the Eddie ECDF cluster for the computational resources required for training, tracking and logging our submitted experiments using Weights & Biases [82].

Chapter 4

Experiments, Results & Discussion

Having introduced the datasets and pipelines we employ in our approach, we outline the set of experiments we ran in order to investigate our research questions below in 4.1, discussing our experimental setup, metrics and hyperparameters. We then report our results and respective observations in 4.2, using them to inform our discussion 4.3, which in turn feed into our conclusions in chapter 5.

4.1 Experiments

We approach our research questions by measuring the performances of our pipelines on each of the datasets within the applicability constraints introduced in the previous chapter. To do this, we first set aside a test set of 10% of the users from the Twibot-20 labelled user set - utilising the same test set used by Feng *et al.*during their evaluation of BotRGCN's performance, as well the performance of various twitter bot detection baselines [21].

Within the first 90% of the labelled users, we then train our models using 5-fold cross validation [83], repeating each fold 3 times and reporting the means and standard deviations of our results over the different train and validation folds. We then evaluate our models on the remaining 10% of the users using a model trained on the entire 90% set, repeating our experiments 10 times and reporting the means and standard deviations across those runs.

4.1.1 Metrics

We optimise our model using stochastic gradient descent (SGD) [84] over the binary cross-entropy [85] of the output with respect to the labels - a commonly utilised loss function in classification pipelines in machine learning. The cross-entropy function takes the model's generated class probabilities as its input, which are usually generated using a softmax activation function over the output of the last layer of the model [86]. These probabilities are then used to calculate the class loss penalties as shown in equation 4.1 [86], where y_c^n is the binary label for class c for data point n, $\hat{y}_c^n$ is the respective predicted probability, N is the total number of data points, and C is the number of classes.

$$-\frac{1}{N} \cdot \sum_{n=0}^{N} (\sum_{c=0}^{C} y_c^n \cdot log(\hat{y}_c^n)) \tag{4.1}$$

Since label vector in multi-class classification is a binary vector where only the target class is set to 1, this calculates the mean negative log likelihood of the target class across the dataset [85]. However, as mentioned in section 3.2, the PyTorch implementation - which we utilise - takes the raw values as input, calculating the log of the softmax for the entries before applying the negative likelihood; while mathematically equivalent, this achieves faster performance and improves numerical stability [81].

To test our model's effectiveness at the classification task, we utilise the variety of binary classification metrics. To aid in their explanation, we first introduce the binary classification *confusion matrix*, which is a 2-by-2 matrix where the rows correspond to the labels and the columns correspond to the predictions, and predictions count as one of the four "classes": True Positives (TP), False Positives (FP), True Negatives (TN) and False Negatives (FN). This is visualised in figure 4.1.

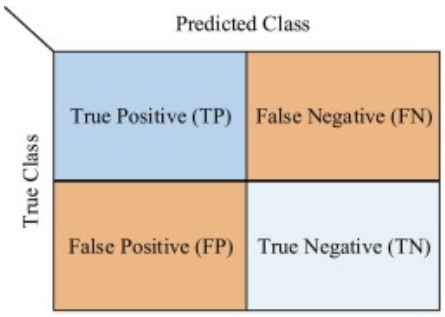


Figure 4.1: Binary Confusion Matrix. Taken from [87]

Accordingly, the classification accuracy is defined as proportion of correct predic-

tions, and can be calculated using $(TP+TN)/(TP+TN+FP+FN)$ [88]. While simple accuracy is a helpful measure of a model's effectiveness, more useful analysis can be conducted using more powerful metrics [88, 89, 90]. Precision and recall demonstrate a model's proportion of correct positive guesses to the positive predictions and positively labelled data-points respectively; these can alternatively be worded as $P = (TP/(TP+FP))$ and $R = (TP/(TP+FN))$ respectively [88]. This is useful since particularly within contexts with high class imbalance, a model could achieve a high accuracy by predicting every point to be a part of the dominant, which might not be particularly useful. Precision and recall therefore help in exposing similar model behaviour. The F1-score combines both precision and recall into a single score by taking their harmonic mean. This is calculated by $(2(P*R))/(P+R)$ [88].

For our final metric, we use the area under the Receiver Operating Characteristic (ROC) curve, or AUC-ROC score for short [89]. ROC curves plot a binary classification model's true positive rate (TPR) - also known as its recall - against the false positive rate (FPR) - calculated as $FPR = FP/(FP+TN)$ - at different classification confidence thresholds. The area under this curve is an approximation for a model's prediction "skill", where a more "skillful" model will predict a random positively labelled example to be positive at a higher probability, thus resulting in a prediction performance that is less affected by changing the prediction threshold. A completely random predictor would have an ROC "curve" equivalent to a line with a $y = x$ equation, and would thus have an area of 0.5; meanwhile a perfect predictor would have a "curve" with lines from (0,0) to (0,1) then from (0,1) to (1,1), achieving a score of 1.0 [89]. While other metrics such as the area under the precision-recall curve (AUC-PR) [91] were initially considered and the Mathews Correlation Coefficient [92], this was not deemed as necessary due to the reporting of the precision, recall and F1 scores as described above alongside the AUC-ROC, and due the relatively balanced nature of the class distribution in our dataset [90].

4.1.2 Hyperparameters

While predictions output by our model are directly influenced by its parameters, the optimisation of these parameters is directly influenced by the model's hyperparameters, which control different aspects of the model structure and the training process itself. While some hyperparameter settings were shared by both pipelines, others more related to the specific models' learning were given different values. We first introduce what our

Name	Description
lr	The base learning rate for training.
weight_decay	The weight decay coefficient.
epochs	Number of training epochs.
numHANLayers	Number of HAN layers in the MinibatchedAugHAN model.
neighboursPerNode	Neighbours sampled per node during mini-batching.
batch_size	The number of user nodes evaluated per batch.
svdComponents	Number of SVD components in the content embeddings.
embedding_size	Learnt embedding size & number of hidden units.
dropout	The dropout probability for dropout layers.
LeakyReLU alpha	LeakyReLU activation [68] negative value coefficient.

Table 4.1: Hyperparameter names and descriptions

hyperparameters are then discuss how their values were selected.

Since our models - particularly the RGCN-based variant - are based on a modification of the BotRGCN pipeline, we maintain many of the hyperparameters utilised in that approach. For our model training, we employ the AdamW optimiser [93], initialising our model's parameters using Kaiming weight initialisation [94] - both utilised in their pipeline as well. AdamW has the advantages of producing more generalisable models than the commonly used Adam optimiser [95] while maintaining significantly faster convergence than standard SGD [93]; meanwhile, Kaiming initialisation aids in avoiding vanishing and exploding gradients [96] during training [94].

Our remaining hyperparameters are shown below in table 4.1.

For the selection of our hyperparameter values, different approaches were taken for each of our two pipelines. For TweetAugmentedRGCN, we reason that since the model structure closely resembles that of BotRGCN - including the use of the same GNN model structure -, it is also reasonable to employ the hyperparameter values utilised in their work rather than running our own hyperparameter optimisation. While this may result in sub-optimal performance on our dataset, we reason that since achieving state of the art performance on this task is not a research objective, it is a worthwhile trade-off to make in order to allow more development time for the MinibatchedAugHAN pipeline. However, we perform a small grid search to test different values for the hyperparameters we introduce for this pipeline, such as the number of SVD components.

For the latter pipeline however, we employ Bayesian optimisation to optimise the

Hyperparameter Name	Value
lr	0.001
weight_decay	0.005
epochs	60
numLayers	2
neighboursPerNode	N/A
batch_size	N/A
svdComponents	100
embedding_size	96
dropout	0.3
LeakyReLU alpha	0.2
Training Loss	0.375
Validation Loss	0.429

(a) TweetAugmentedRGCN

Hyperparameter Name	Value
lr	0.0025
weight_decay	0.0088
epochs	41
numHANLayers	2
neighboursPerNode	207
batch_size	1024
svdComponents	50
embedding_size	240
dropout	0.510
LeakyReLU alpha	0.200
Training Loss	0.396
Validation Loss	0.432

(b) MinibatchedAugHAN

Table 4.2: Hyperparameter Values and Obtained Training and Validation Losses

hyperparameter values - setting the minimisation of the validation loss as our objective function [97]. Bayesian optimisation is a probabilistic black-box optimisation method which creates a surrogate function that attempts to model the model's output space with respect to its input variables using a joint probability distribution over the entire input space. It then iteratively updates the surrogate function by sampling different input values and measuring the objective function output, utilising a search strategy which balances between exploration and exploitation of minima in the search space [98]. We conduct our search using the UT-FullHetero dataset, reasoning that the maximally encoded information in this graph would result in a set of hyperparameters that would be transferable to versions of the graph with less encoded information.

We optimise our hyperparameters over 610 runs, periodically tracking the best obtained validation loss value until no substantial improvement is observed. The set of hyperparameter values, as well as the respective training and validation losses, obtained from the best performing run from this search are shown below in table 4.2. While these hyperparameters result in a validation loss improvement from 0.523 to 0.432, notable shortcomings in our approach are discussed below in section 4.2.3.

4.2 Results & Discussion

4.2.1 Main Results

As mentioned above in section 4.1, we first evaluated both the TweetAugmentedRGCN and MinibatchedAugHAN pipelines using 5-fold cross validation, repeating each fold thrice, and calculating our aggregate means and standard deviations across all 15 runs for our metrics. The training and validation results of these runs are shown in tables 4.3 and 4.5. We then evaluate these different configurations on the test set, reporting the results in tables 4.4 and 4.6.

While our hypothesis had been that the proposed dataset modifications - namely the additional representations and the encoding of the subsequent graph's heterogeneity - would contribute positively to the model's twitter bot detection accuracy, our results paint a mixed picture.

With regards to the effect of encoding the different node and edge types in the modified graph, our results suggest there is no significant effect on the detection performance. This is seen across both models, with minor ($< 1\%$ difference) variations in performance obtained throughout the different utilised metrics. However, this agreement is not observed when comparing results obtained from the user-only graph with those obtained from the user-tweet graphs, where the additional representations result in significantly stronger results for MinibatchedAugHAN, but no notable difference for TweetAugmentedRGCN.

We also note the fact that our models achieve consistently higher recall values in comparison to the respective precision, with precision and recall values ranging between 0.65-0.76 and 0.91-0.98 respectively. To visualise this further, we include ROC curves for our pipelines on sample U and UT graphs, comparing the performances of our models on user-only graphs with that on user-tweet graphs 4.2. We note that while models with more balanced precision and recall performances would have a "curve peak" with a TPR value nearer to (1-FPR), our models exhibit peaks significantly shifted to the top right, indicating a model performance that is more heavily skewed towards predicting that a user it is unsure about is a bot.

4.2.2 Neighbourhood Sampling & Heterogeneity Interactions

We also note the presence of larger performance instability in the results obtained from the MinibatchedAugHAN pipeline in comparison to the TweetAugmentedRGCN

Dataset Name	Train Loss	Train Acc	Val Loss	Val Acc
U-Homo	0.409 ± 0.015	80.6 ± 0.6	0.417 ± 0.037	80.4 ± 1.6
U-EdgeHetero	0.414 ± 0.025	80.5 ± 1.0	0.43 ± 0.038	80.3 ± 1.8
UT-Homo	0.405 ± 0.014	80.9 ± 0.7	0.425 ± 0.037	80.1 ± 1.5
UT-EdgeHetero	0.411 ± 0.017	80.6 ± 0.8	0.435 ± 0.038	79.7 ± 1.5

Dataset Name	Val Precision	Val Recall	Val F1	Val AUC-ROC
U-Homo	0.743 ± 0.044	0.977 ± 0.006	0.843 ± 0.029	0.771 ± 0.043
U-EdgeHetero	0.743 ± 0.043	0.971 ± 0.015	0.842 ± 0.032	0.768 ± 0.041
UT-Homo	0.742 ± 0.046	0.973 ± 0.008	0.841 ± 0.03	0.769 ± 0.041
UT-EdgeHetero	0.741 ± 0.043	0.963 ± 0.012	0.837 ± 0.027	0.765 ± 0.044

Table 4.3: 5-Fold Cross Validation Results: Means and Standard Deviations for Tweet-AugmentedRGCN

Dataset Name	Test Loss	Test Acc
U-Homo	0.395 ± 0.012	81.6 ± 0.7
U-EdgeHetero	0.409 ± 0.007	81.1 ± 0.4
UT-Homo	0.399 ± 0.008	81.1 ± 0.3
UT-EdgeHetero	0.403 ± 0.009	81.4 ± 0.4

Dataset Name	Test Precision	Test Recall	Test F1	Test AUC-ROC
U-Homo	0.756 ± 0.006	0.976 ± 0.015	0.852 ± 0.006	0.802 ± 0.006
U-EdgeHetero	0.753 ± 0.004	0.97 ± 0.004	0.848 ± 0.003	0.797 ± 0.004
UT-Homo	0.752 ± 0.005	0.971 ± 0.007	0.847 ± 0.002	0.797 ± 0.003
UT-EdgeHetero	0.754 ± 0.006	0.972 ± 0.01	0.849 ± 0.003	0.799 ± 0.005

Table 4.4: Test Set Results: Means and Standard Deviations for TweetAugmentedRGCN

Dataset Name	Train Loss	Train Acc	Val Loss	Val Acc
U-Homo	0.453 ± 0.01	74.5 ± 0.9	0.564 ± 0.047	70.9 ± 3.5
U-EdgeHetero	0.462 ± 0.008	74.2 ± 0.9	0.55 ± 0.035	71.1 ± 3.5
UT-Homo	0.304 ± 0.008	85.3 ± 0.4	0.606 ± 0.067	77.6 ± 1.4
UT-EdgeHetero	0.324 ± 0.008	84.4 ± 0.5	0.573 ± 0.056	77.2 ± 1.9
UT-FullHetero	0.314 ± 0.01	85.0 ± 0.5	0.609 ± 0.058	78.1 ± 2.0

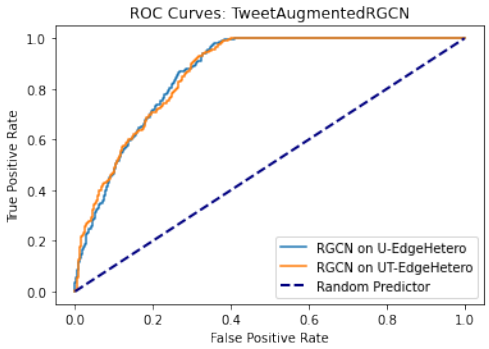
Dataset Name	Val Precision	Val Recall	Val F1	Val AUC-ROC
U-Homo	0.662 ± 0.068	0.954 ± 0.02	0.779 ± 0.048	0.667 ± 0.048
U-EdgeHetero	0.662 ± 0.069	0.959 ± 0.02	0.781 ± 0.051	0.668 ± 0.044
UT-Homo	0.733 ± 0.052	0.926 ± 0.018	0.817 ± 0.032	0.748 ± 0.037
UT-EdgeHetero	0.734 ± 0.052	0.91 ± 0.031	0.811 ± 0.038	0.744 ± 0.033
UT-FullHetero	0.735 ± 0.05	0.928 ± 0.042	0.819 ± 0.043	0.749 ± 0.032

Table 4.5: 5-Fold Cross Validation Results: Means and Standard Deviations for MinibatchedAugHAN

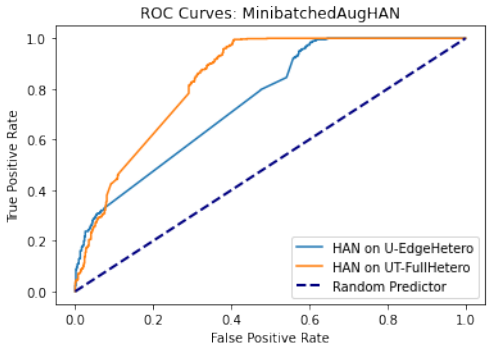
Dataset Name	Test Loss	Test Acc
U-Homo	0.542 ± 0.026	70.9 ± 0.8
U-EdgeHetero	0.523 ± 0.01	71.3 ± 0.9
UT-Homo	0.584 ± 0.04	78.6 ± 1.4
UT-EdgeHetero	0.519 ± 0.024	79.3 ± 1.4
UT-FullHetero	0.559 ± 0.027	79.2 ± 0.9

Dataset Name	Test Precision	Test Recall	Test F1	Test AUC-ROC
U-Homo	0.658 ± 0.007	0.963 ± 0.013	0.782 ± 0.006	0.687 ± 0.009
U-EdgeHetero	0.661 ± 0.007	0.964 ± 0.015	0.784 ± 0.006	0.691 ± 0.009
UT-Homo	0.747 ± 0.005	0.915 ± 0.038	0.822 ± 0.016	0.774 ± 0.012
UT-EdgeHetero	0.748 ± 0.004	0.93 ± 0.036	0.829 ± 0.015	0.781 ± 0.012
UT-FullHetero	0.747 ± 0.004	0.932 ± 0.021	0.829 ± 0.01	0.78 ± 0.009

Table 4.6: Test Set Results: Means and Standard Deviations for MinibatchedAugHAN



(a) ROC Curves for TweetAugmentedRGCN on U-EdgeHetero vs UT-EdgeHetero



(b) ROC Curves for MinibatchedAugHAN on U-EdgeHetero vs UT-FullHetero

Figure 4.2: ROC Curves for TweetAugmentedRGCN and MinibatchedAugHAN on sample U and UT graphs

pipeline, as shown by the standard deviations in our metrics across different runs. For example, the standard deviations in the reported test set accuracy are consistently higher in the MinibatchedAugHAN respective runs in comparison to the TweetAugmentedRGCN runs, with ranges of 0.4-0.7 and 0.8-1.4 respectively.

In fact, this is further supported by an interaction of our sampling strategy with the structure of the graph, wherein the difference in the standard deviation of the models' performances in the UT-Homo and UT-EdgeHetero datasets is particularly high. This is due to the neighbourhood sampling strategy employed by the model, wherein a fixed neighbourhood size is sampled per node type [24].

We believe that this causes high instability for these particular configurations since the graph in these datasets is represented by a uniform node type despite encoding multiple types of entities, and since the number of tweets is significantly larger than the number of users in the graph, the number of users in a user node's sampled neighbourhood is likely to vary greatly, affecting the feature propagation across the graph.

On the other hand, we argue that the use of sampling over the encoded node types rather than the entities being represented in the graph can be considered as a natural part of the dataset versions being configured as node-homogeneous, since the consideration of the different entity types in the sampling strategy would entail some representation of heterogeneity within these pipeline despite the model then receiving one set of nodes across the different entity types afterwards. Nevertheless, a modification of the sampling strategy for these versions of the dataset could be explored as an avenue of future work.

4.2.3 Hyperparameter Optimisation Shortcomings

As introduced in 4.1.2, the employed hyperparameter optimisation setup exhibits some shortcomings which were only realised afterwards. Not only are the model's hyperparameters optimised according to a specific dataset structure, the optimisation was also conducted using the same validation set across different runs, and each candidate set of hyperparameters was tested in a singular run, rather than averaging cross repeats - or indeed different folds - as was done in our final results.

While this was motivated by an attempt to achieve wider hyperparameter space coverage through a larger number of runs, it has multiple effects on our results. First and foremost, the use of the same validation set results in the model "overfitting", in a sense, to the set of users in the employed validation set, since the hyperparameters are tuned to extract the best performance on this specific set. This is reflected in the model's significantly worse cross-validation and test set losses when compared to the validation loss reported for the chosen set of hyperparameters during the optimisation process, with values of 0.609, 0.559 and 0.431 respectively.

In addition, it may be argued that the use of a specific dataset in the hyperparameter optimisation process biases the results towards that specific dataset, as well as similar datasets being evaluated. This might contribute to explaining the large difference between the MinibatchedAugHAN model performance on the UT graphs in comparison to the U graphs despite similar performances for TweetAugmentedRGCN.

4.2.4 Other Observations

We also note the slightly weaker results achieved for our MinibatchedAugHAN in comparison to TweetAugmentedRGCN - even at its best performing configuration. We mainly attribute this to a combination of the observed overfitting as well as the use of the enforced graph sparsity due to the fixed neighbourhood size in the sampling phase during graph mini-batching. Although a relatively large number of neighbours is utilised per node, the reduction in neighbourhood sizes likely causes some information loss in the graph for HAN - particularly in terms of nodes' meta-path based neighbourhoods.

Finally, we note the significant difference in run-time requirements for our models on the user-only graphs in comparison to the user-tweet graphs, with the latter runs taking anywhere between 8 and 40 times as long to run on the cluster. However, we caveat this with a limitation in our approach wherein CPU training was utilised instead of benefitting from GPU training speedups [99] due to configuration and memory issues

encountered on the latter.

4.3 Discussion

Based on the observations made in our results, we argue that the effects of the entity and relationship set expansions on the twitter bot detection accuracy is model-dependent. For a model such as RGCN, which sets a different custom weight matrix for each edge type but otherwise performs uniform graph convolutions [33, 42], the addition of the tweet nodes did not yield any performance benefits despite substantially increasing the computational resources needed to arrive at its predictions on the same set of users.

On the other hand, models such as HAN, which attend over different structural relationships encoded into the graph through the use of heterogeneous representations [35], benefit much more greatly from this entity and relationship addition. This is particularly due to the direct utilisation of attention over meta-paths in its operation, which in turn depend on the existence of different interacting entities in the graph [35].

However, we also find that the explicit encoding of the different node and edge types to form a heterogeneous graph does not result in any significant performance benefits in isolation. They also did not provide additional implementation or computational costs given the model being utilised. They do, however, make the implementation of these graphs more straightforward by removing any constraints on the uniformity of node and edge embedding sizes, thus avoiding feature engineering workarounds such as necessitating the use of learnt embeddings with uniform sizes, or the use of zero padding. It is therefore not difficult to understand why they are included by default in many applications of heterogeneous graphs [100, 101, 102, 30, 31, 32].

Furthermore, while our approach was built as a direct adaptation of the BotRGCN pipeline, we unfortunately achieve weaker results across the board in comparison to those reported by their work [21]. We suspect that this may be caused by the early modifications we did to reduce the computational footprint of our experiments - namely the dropping of the support set and the utilisation of the TF-IDF embeddings in place of the more powerful RoBERTa embeddings[73, 62].

However, the consistency in the observed performance benefits resultant from the graph expansion - regardless of whether the heterogeneous types were explicitly encoded - suggests that the graph structure has a direct effect on a model's twitter bot classification performance. This leaves room for an avenue of future work which leverages these modified graph structures for performance benefits in our task. This is

discussed further in chapter 5.

4.3.1 Limitations

We also note some limitations in our work, in addition to the shortcomings discussed in section 4.2.3. For instance, the expanded set of entities and relationships employed in our graphs is not exhaustive; in fact, they were limited during the development process by the data available in the Twibot-20 dataset [16]. With that stated, a recent development in the twitter bot detection landscape - being in the preprint phase at the time of writing - is the introduction of the Twibot-22 dataset [103], which addresses many of the shortcomings inherent to the dataset employed by our work in a new twitter bot detection benchmark.

In addition to expanding the labelled user set from 11K to 1 million users, their graphs also represent 4 different entities and 14 different relationships, the former being users, tweets, lists and hashtags. Their evaluation of existing twitter bot detection systems confirms the superior performance of graph-based approaches in comparison to others, where BotRGCN - as well as other edge-heterogeneous approaches [37] - are highlighted as the strongest performing pipelines.

Furthermore, the use of the more recent graph transformer-based models [36, 37, 104], which have been shown to perform well so far on this task in edge-heterogeneous contexts [37] as well as outperform GAT-based [43] architectures such as HAN on other tasks [36, 104]. Due to the demonstrated interactions of the different components of a graph based twitter bot detection system, such as the graph structure, graph heterogeneity and model structure, the introduction of Twibot-22 suggests that there is more to be done with regards to finding better performing combinations. Therefore, a more exhaustive version of our experimentation that leverages these additional representations remains an area of future work.

Chapter 5

Conclusions

5.1 Summary

This project aimed to measure the effect of a set of graph structure modifications on the accuracy of graph-based twitter bot detection systems, also investigating relationship between these effects and the GNN models employed for the feature propagation step in the operating pipeline. For these graph modifications, we mainly focused on the effects of expanding the set of entities and relationships represented in the graph, and the effect of then encoding the resultant heterogeneous node and edge types.

We thus conducted our investigation by evaluating the performances of two pipelines - each utilising a different GNN model in its feature propagation block - on the 5 resultant graphs from combining these different sets of modifications. However, while we hypothesised that the introduction of the additional nodes and edges would result in a performance improvement that would then be further compounded by the explicit encoding of those types, our results paint a different picture.

Although many approaches in literature operating over graphs containing multiple node and edge types encode these type mappings in their datasets [30, 31, 32], we find that the encoding of these mappings has no significant performance effect in isolation - regardless of the employed model. This implies that any performance improvements to be obtained from adding the node and edge representations are gained by the model regardless of whether the type mappings were specified.

Furthermore, we also conclude that the specified performance improvements are not universal - instead depending on the GNN architecture utilised for the feature propagation. This is demonstrated by a divergence in the results we obtained for our HAN-based [35] MinibatchedAugHAN pipeline in comparison to those obtained from

the RGCN-based [33] TweetAugmentedRGCN pipeline, where the addition of tweet nodes and a set of relevant interactions to our user-formed follower graphs resulted in a significant performance improvement for the former but no effect for the latter. We argue that this is directly attributable to the architectural differences between the GNN models employed in these pipelines.

However, we also caveat our conclusions with some limitations in our approach, which have been discussed above in sections 4.2.3 and 4.3.1. These mainly pertain to the overfitting observed in our results, which we identify to be a result of some shortcomings in our hyperparameter optimisation process, as well as limitations in the graph structure modifications on which our experiments were evaluated wherein only one possible user-tweet graph structure was empirically evaluated in our experiments. While other possible structural variations were mentioned in section 4.3, similar investigations on these remain an area of future work alongside the avenues mentioned below.

5.2 Future Work

As mentioned above, while our expanded set of entities and relationships captures more diverse user interactions in comparison to existing twitter bot detection methods, they still do not reflect the true diversity of the different interactions taking place on Twitter. As outlined in section 4.3.1, this was limited by the available information in the Twibot-20 dataset [16], which at the time of development was the most feature rich twitter bot detection dataset.

On the other hand, the recent introduction of the Twibot-22 dataset [103] opens the door for the development of graph-based twitter bot detection systems which can utilise wider sets of entities, relationships and graph structures. For instance, this includes allowing for the encoding of user interactions around popular hashtags - a powerful feature differentiating social botnet activity from genuine human activity on twitter [15, 23]. It also allows for the encoding of a much larger set of meta-paths, which perhaps would allow a meta-path based model such as HAN to translate its performance improvements over RGCN in other tasks [35] to Twitter Bot Detection.

In a wider scope, the observed direct effect of the interactions between graph structure and employed GNN model on bot detection accuracy suggests that there is room for the introduction of more powerful twitter bot detection pipelines exploiting these interactions. For example, we believe that a configuration utilising an expanded set of nodes and entities and more powerful embeddings while keeping the unlabelled

set of users in the graph may outperform the current state-of-the-art if paired with careful model selection. A contributing factor to this would likely be the presence of richer user feature propagation, which perhaps might prove to be particularly effective in combination with the other introduced information in the graph.

Bibliography

[1] Simon Kemp. Digital 2022: Global overview report - datareportal – global digital insights, May 2022.

[2] M. Nick Hajli. A study of the impact of social media on consumers. *International Journal of Market Research*, 56(3):387–404, 2014.

[3] Clay Shirky. The political power of social media: Technology, the public sphere, and political change. *Foreign Affairs*, 90(1):28–41, 2011.

[4] Gwenn Schurgin O'Keeffe, Kathleen Clarke-Pearson, Council on Communications, and Media. The impact of social media on children, adolescents, and families. *Pediatrics*, 127(4):800–804, 2011.

[5] Gabriel Magno, Tiago Rodrigues, and Virgílio A. F. Almeida. Detecting spammers on twitter. 2010.

[6] Jacob Ratkiewicz, Michael Conover, Mark Meiss, Bruno Goncalves, Alessandro Flammini, and Filippo Menczer. Detecting and tracking political abuse in social media. *Proceedings of the International AAAI Conference on Web and Social Media*, 5(1):297–304, Aug. 2021.

[7] Emilio Ferrara, Onur Varol, Clayton Davis, Filippo Menczer, and Alessandro Flammini. The rise of social bots. *Commun. ACM*, 59(7):96–104, jun 2016.

[8] Kyumin Lee, Brian Eoff, and James Caverlee. Seven months with the devils: A long-term study of content polluters on twitter. *Proceedings of the International AAAI Conference on Web and Social Media*, 5(1):185–192, Aug. 2021.

[9] Ashok Deb, Luca Luceri, Adam Badaway, and Emilio Ferrara. Perils and challenges of social media and election manipulation analysis: The 2018 us midterms. In *Companion Proceedings of The 2019 World Wide Web Conference*, WWW

'19, page 237–247, New York, NY, USA, 2019. Association for Computing Machinery.

[10] Philip N. Howard and Bence Kollanyi. Bots, #strongerin, and #brexit: Computational propaganda during the uk-eu referendum. *ArXiv*, abs/1606.06356, 2016.

[11] Alessandro Bessi and Emilio Ferrara. Social bots distort the 2016 us presidential election online discussion. *First monday*, 21(11-7), 2016.

[12] David A. Broniatowski, Amelia M. Jamison, SiHua Qi, Lulwah Alkulaib, Tao Chen, Adrian Benton, Sandra Crouse Quinn, and Mark Dredze. Weaponized health communication: Twitter bots and russian trolls amplify the vaccine debate. *American Journal of Public Health*, 108:1378–1384, 2018.

[13] Emily Chen, Kristina Lerman, and Emilio Ferrara. Tracking social media discourse about the covid-19 pandemic: Development of a public coronavirus twitter data set. *JMIR Public Health and Surveillance*, 6, 2020.

[14] Kai-Cheng Yang, Onur Varol, Clayton A Davis, Emilio Ferrara, Alessandro Flammini, and Filippo Menczer. Arming the public with artificial intelligence to counter social bots. *Human Behavior and Emerging Technologies*, 1(1):48–61, 2019.

[15] Stefano Cresci, Roberto Di Pietro, Marinella Petrocchi, Angelo Spognardi, and Maurizio Tesconi. The paradigm-shift of social spambots: Evidence, theories, and tools for the arms race. In *Proceedings of the 26th International Conference on World Wide Web Companion*, WWW '17 Companion, page 963–972, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee.

[16] Shangbin Feng, Herun Wan, Ningnan Wang, Jundong Li, and Minnan Luo. *TwiBot-20: A Comprehensive Twitter Bot Detection Benchmark*, page 4485–4494. Association for Computing Machinery, New York, NY, USA, 2021.

[17] Chao Yang, Robert Harkreader, and Guofei Gu. Empirical evaluation and new design for fighting evolving twitter spammers. *IEEE Transactions on Information Forensics and Security*, 8(8):1280–1293, 2013.

[18] Sneha Kudugunta and Emilio Ferrara. Deep neural networks for bot detection. *Information Sciences*, 467:312–322, 2018.

[19] Feng Wei and Uyen Trang Nguyen. Twitter bot detection using bidirectional long short-term memory neural networks and word embeddings. In *2019 First IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, pages 101–109, 2019.

[20] Seyed Ali Alhosseini, Raad Bin Tareaf, Pejman Najafi, and Christoph Meinel. Detect me if you can: Spam bot detection using inductive representation learning. In *Companion Proceedings of The 2019 World Wide Web Conference*, WWW '19, page 148–153, New York, NY, USA, 2019. Association for Computing Machinery.

[21] Shangbin Feng, Herun Wan, Ningnan Wang, and Minnan Luo. Botrgcn: Twitter bot detection with relational graph convolutional networks. In *Proceedings of the 2021 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, ASONAM '21, page 236–239, New York, NY, USA, 2021. Association for Computing Machinery.

[22] Matteo Bruno, Renaud Lambiotte, and Fabio Saracco. Brexit and bots: characterizing the behaviour of automated accounts on twitter during the uk election. *EPJ Data Science*, 11, 12 2022.

[23] Marco T. Bastos and Dan Mercea. The brexit botnet and user-generated hyperpartisan news. *Social Science Computer Review*, 37(1):38–54, 2019.

[24] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[25] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2019.

[26] Christo Wilson, Alessandra Sala, Krishna P. N. Puttaswamy, and Ben Y. Zhao. Beyond social graphs: User interactions in online social networks and their implications. *ACM Trans. Web*, 6(4), nov 2012.

[27] Stanley Wasserman and Katherine Faust. *Social Network Analysis: Methods and Applications*. Structural Analysis in the Social Sciences. Cambridge University Press, 1994.

[28] Evelien Otte and Ronald Rousseau. Social network analysis: a powerful strategy, also for the information sciences. *Journal of Information Science*, 28(6):441–453, 2002.

[29] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is Twitter, a social network or a news media? In *WWW '10: Proceedings of the 19th international conference on World wide web*, pages 591–600, New York, NY, USA, 2010. ACM.

[30] Danqing Wang, Pengfei Liu, Yining Zheng, Xipeng Qiu, and Xuanjing Huang. Heterogeneous graph neural networks for extractive document summarization. *arXiv preprint arXiv:2004.12393*, 2020.

[31] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 974–983, New York, NY, USA, 2018. Association for Computing Machinery.

[32] K Anoop, Manjary P Gangan, VL Lajish, et al. Leveraging heterogeneous data for fake news detection. In *Linking and Mining Heterogeneous and Multi-view Data*, pages 229–264. Springer, 2019.

[33] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In Aldo Gangemi, Roberto Navigli, Maria-Esther Vidal, Pascal Hitzler, Raphaël Troncy, Laura Hollink, Anna Tordai, and Mehwish Alam, editors, *The Semantic Web*, pages 593–607, Cham, 2018. Springer International Publishing.

[34] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V. Chawla. Heterogeneous graph neural network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 793–803, New York, NY, USA, 2019. Association for Computing Machinery.

[35] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *The World Wide Web Conference*, WWW '19, page 2022–2032, New York, NY, USA, 2019. Association for Computing Machinery.

[36] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. *Heterogeneous Graph Transformer*, page 2704–2710. Association for Computing Machinery, New York, NY, USA, 2020.

[37] Shangbin Feng, Zhaoxuan Tan, Rui Li, and Minnan Luo. Heterogeneity-aware twitter bot detection with relational graph transformers. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(4):3977–3985, Jun. 2022.

[38] Robin J. Wilson. *Definitions and Examples*. Prentice Hall, 2015.

[39] Paul M. Sant. "graph". in *Dictionary of Algorithms and Data Structures* [online], Paul E. Black, ed., December 2004. Available from: https://www.nist.gov/dads/HTML/graph.html (accessed 31 July 2022).

[40] Paul M. Sant. "adjacency-matrix representation". in *Dictionary of Algorithms and Data Structures* [online], Paul E. Black, ed., December 2004. Available from: https://www.nist.gov/dads/HTML/undirectedGraph.html (accessed 31 July 2022).

[41] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S Yu, and Tianyi Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment*, 4(11):992–1003, 2011.

[42] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[43] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[44] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[45] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734 vol. 2, 2005.

[46] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[47] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 3844–3852, Red Hook, NY, USA, 2016. Curran Associates Inc.

[48] Kunihiko Fukushima. Cognitron: A self-organizing multilayer neural network. *Biological Cybernetics*, 20:121–136, 1975.

[49] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[50] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[51] Vassilis N Ioannidis, Da Zheng, and George Karypis. Few-shot link prediction via graph neural networks for covid-19 drug-repurposing. *arXiv preprint arXiv:2007.10261*, 2020.

[52] Junfeng Jiang, An Wang, and Akiko Aizawa. Attention-based relational graph convolutional network for target-oriented opinion words extraction. In *EACL*, 2021.

[53] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

[54] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2015.

[55] Kyumin Lee, James Caverlee, and Steve Webb. Uncovering social spammers: social honeypots+ machine learning. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 435–442, 2010.

[56] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[57] Zachary Miller, Brian Dickinson, William Deitrick, Wei Hu, and Alex Hai Wang. Twitter spammer detection using data stream clustering. *Inf. Sci.*, 260:64–73, mar 2014.

[58] Clayton A. Davis, Onur Varol, Emilio Ferrara, Alessandro Flammini, and Filippo Menczer. Botornot: A system to evaluate social bots. *Proceedings of the 25th International Conference Companion on World Wide Web*, 2016.

[59] Kai-Cheng Yang, Onur Varol, Pik-Mai Hui, and Filippo Menczer. Scalable and generalizable social bot detection through data selection. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 1096–1103, 2020.

[60] Mohsen Sayyadiharikandeh, Onur Varol, Kai-Cheng Yang, Alessandro Flammini, and Filippo Menczer. Detection of novel social bots by ensembles of specialized classifiers. In *Proceedings of the 29th ACM international conference on information & knowledge management*, pages 2725–2732, 2020.

[61] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. volume 14, pages 1532–1543, 01 2014.

[62] Liu Zhuang, Lin Wayne, Shi Ya, and Zhao Jun. A robustly optimized BERT pre-training approach with post-training. In *Proceedings of the 20th Chinese National Conference on Computational Linguistics*, pages 1218–1227, Huhhot, China, August 2021. Chinese Information Processing Society of China.

[63] Usman Naseem, Imran Razzak, Shah Khalid Khan, and Mukesh Prasad. A comprehensive survey on word representation models: From classical to state-of-the-art word representation language models. *ACM Trans. Asian Low-Resour. Lang. Inf. Process.*, 20(5), jun 2021.

[64] Qian Li, Hao Peng, Jianxin Li, Congying Xia, Renyu Yang, Lichao Sun, Philip S. Yu, and Lifang He. A survey on text classification: From traditional to deep learning. *ACM Trans. Intell. Syst. Technol.*, 13(2), apr 2022.

[65] He Zhao, Dinh Q. Phung, Viet Huynh, Yuan Jin, Lan Du, and Wray L. Buntine. Topic modelling meets deep neural networks: A survey. *ArXiv*, abs/2103.00498, 2021.

[66] Paul Geladi and Bruce R. Kowalski. Partial least-squares regression: a tutorial. *Analytica Chimica Acta*, 185:1–17, 1986.

[67] Yoshua Bengio, Ian Goodfellow, and Aaron Courville. *Deep learning*, volume 1. MIT press Cambridge, MA, USA, 2017.

[68] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Citeseer, 2013.

[69] Eduardo Graells-Garrido and Ricardo Baeza-Yates. Bots don't vote, but they surely bother! a study of anomalous accounts in a national referendum. In *14th ACM Web Science Conference 2022*, WebSci '22, page 302–306, New York, NY, USA, 2022. Association for Computing Machinery.

[70] Adrian Rauchfleisch and Jonas Kaiser. The false positive problem of automatic bot detection in social science research. *SSRN Electronic Journal*, 01 2020.

[71] Twitter api documentation — docs — twitter developer platform.

[72] Felipe Pezoa, Juan L Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. Foundations of json schema. In *Proceedings of the 25th International Conference on World Wide Web*, pages 263–273. International World Wide Web Conferences Steering Committee, 2016.

[73] Karen Spärck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28:11–21, 1972.

[74] V. Klema and A. Laub. The singular value decomposition: Its computation and some applications. *IEEE Transactions on Automatic Control*, 25(2):164–176, 1980.

[75] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[76] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. *CoRR*, abs/1801.07606, 2018.

[77] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[78] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

[79] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[80] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[81] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[82] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com.

[83] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[84] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[85] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[86] John S Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing*, pages 227–236. Springer, 1990.

[87] Fatih Demir. 14 - deep autoencoder-based automated brain tumor detection from mri data. In Varun Bajaj and G.R. Sinha, editors, *Artificial Intelligence-Based Brain-Computer Interface*, pages 317–351. Academic Press, 2022.

[88] Alaa Tharwat. Classification assessment methods. *Applied Computing and Informatics*, 2020.

[89] Andrew P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognit.*, 30:1145–1159, 1997.

[90] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, page 233–240, New York, NY, USA, 2006. Association for Computing Machinery.

[91] Christopher Manning and Hinrich Schutze. *Foundations of statistical natural language processing*. MIT press, 1999.

[92] Brian W Matthews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 405(2):442–451, 1975.

[93] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[94] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[95] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[96] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[97] Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

[98] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, page 2951–2959, Red Hook, NY, USA, 2012. Curran Associates Inc.

[99] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[100] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, José Emilio Labra Gayo, S. Kirrane, Sebastian Neumaier, Axel Polleres, R. Navigli, Axel-Cyrille Ngonga Ngomo, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. Knowledge graphs. *Communications of the ACM*, 64:96 – 104, 2021.

[101] Shaohua Fan, Junxiong Zhu, Xiaotian Han, Chuan Shi, Linmei Hu, Biyu Ma, and Yongliang Li. Metapath-guided heterogeneous graph neural network for intent recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 2478–2486, New York, NY, USA, 2019. Association for Computing Machinery.

[102] Ziqi Liu, Chaochao Chen, Xinxing Yang, Jun Zhou, Xiaolong Li, and Le Song. Heterogeneous graph neural networks for malicious account detection. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 2077–2085, 2018.

[103] Shangbin Feng, Zhaoxuan Tan, Herun Wan, Ningnan Wang, Zilong Chen, Binchi Zhang, Qinghua Zheng, Wenqian Zhang, Zhenyu Lei, Shujie Yang, Xinshun Feng, Qingyue Zhang, Hongrui Wang, Yuhan Liu, Yuyang Bai, Heng Wang, Zijian Cai, Yanbo Wang, Lijing Zheng, Zihan Ma, Jundong Li, and Minnan Luo. Twibot-22: Towards graph-based twitter bot detection, 2022.

[104] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J. Kim. Graph transformer networks. In *NeurIPS*, 2019.

Appendix A

Retweet and Mention Edge Extraction RegEx

```
retweetUnameRegex = re.compile(r'(?<=RT @)(\w{1,15})')

mentionsUnameRegex = re.compile(r'(?<!RT @)(?<=@)(\w{1,15})')
```

Figure A.1: Regular Expressions Used for extracting retweet and mention edges from the raw text