Using Graph Neural Network to Detect Twitter Malicious Accounts

Chase Cunningham, Shen-Han Chiu, and Eric Wu

Abstract—With the widespread use of social media platforms, phishing and social scam bots have become a major concern for regular users. Advances in GPU power and data availability have led many researchers to adopt neural networks for bot detection. Graph Neural Networks (GNNs) have proven particularly effective for analyzing social media data due to their ability to leverage the inherent structure of social graphs. Many GNN studies incorporate tweets and supplemental sentiment analysis, which can be costly and computationally intensive. This study demonstrates that GNNs can classify user nodes effectively using only graph inputs, outperforming baseline machine learning models by 6% in F1 score. This demonstrated that GNN is better at addressing data imbalance issues better than other models. Notably, our results show that GNNs achieve strong performance without relying on tweets, avoiding the additional costs and biases associated with sentiment analysis. Future work could include incorporating weighted penalties for training labels to further address data imbalance or expanding the network architecture for improved performance.

***Index Terms*—Node classification, node embeddings, machine learning, graph neural network.**

I. Introduction

SINCE the early 2000s, a deep neural network architecture, the so-called Graph Neural Networks (GNNs) has achieved a reputation by combining both message passing mechanisms and neural networks into graph level tasks. A fascination with that approach allows the deep learning community to apply GNN in real-world applications such as node classification, community detection, link prediction, computer vision, and natural language processing. In this paper, the primary employment of the GNN, along with other semi-supervised learning techniques, focused on malicious account detection in social media. The goal is to train a classifier on the supervised nodes and use it to predict labels of other nodes during testing. Many researchers have used the GNN models extensively for real-world graphs and large-scale datasets including millions of nodes and edges. For example, past research has shown that GCN combines one or more layers more efficiently by applying first-order approximation in the spectral domain; while GraphSage is better at predicting embedding of a node at the one-hop neighborhood information. As a result, current research predominantly deals with the semi-supervised machine learning – almost every GNN related research dismisses the rich diversity of integrating supervised machine learning and increasing the opportunity for classification, prediction, and comparison.

Most malicious account detection problems started in semi-supervised node classification, where nodes need to be classified as bots or non-bots; i.e., a trained GNN model can predict on a test node based on the node features of the labeled nodes (V_L). The graph is composed of $\{A, X\}$, where a small set of nodes V_L are labeled with the label set as Y_L – we are trying to predict the label of unlabeled nodes $V_u = V \setminus V_L$ or V_i's label distribution. The node classification itself to detect bots makes it promising for malicious account detection, especially when bots are generally fake accounts that are different from humans in terms of behaviors with obvious patterns. For example, the majority of users in the social network tend to have more followers than bots, and there are more edges between users to denote the relationship $(v_1, v_2), (v_3, v_1), (v_2, v_4), (v_3, v_5)$.

As shown in Figure 1, each blue node indicates a regular user, and each red node refers to a fake user or malicious account. The task is to predict whether or not the dashed line is real. Is there a link between node v_i and v_j? There are numerous ways to classify either node pair (v_i, v_j) as 1 or 0 to predict the missing link for node classification. It is stipulated that a node's variable importance or information can determine whether the neighboring nodes are malicious accounts or not. The node features that are most influential were able to predict more accurately about the unlabeled nodes in the social network. Therefore, we seek to develop GNN models for node classification.

In traditional machine learning algorithms, it is often faster to train because of smaller datasets. Existing techniques such as clustering, logistic regression, K-nearest neighbors, and Random Forest have extensively improved the performance of the model. It allows the model to make more accurate predictions on the new training set and adjust its parameters to avoid challenges (e.g, overfitting, sampling bias, and complex environments). For graph-based machine learning, two methods can be used to represent graphs: **adjacency matrices** and **node embeddings**. An adjacency matrix seeks to provide a result of the presence or absence of edges between nodes. It can be useful to learn graph structure and edge prediction as a binary classification task. The node embeddings, representing individual nodes too, are preserved in more structural and feature information because this process had aggregation to combine information from other nodes. Overall, both methods have shown to effectively perform in node prediction and classification.

In this paper, we propose to compare existing GNN approaches like GCN to supervised machine learning baseline

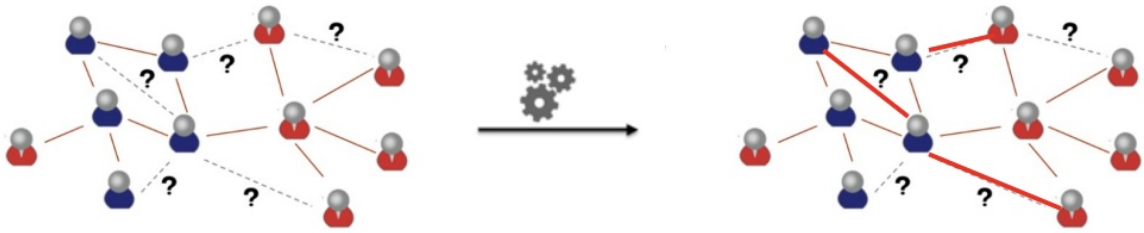

Fig. 1: Illustrated Example of Link Prediction in Graph Neural Networks. Blue nodes are real users, red nodes are bot, and dashed line are the missing link between users in a social network.

Item	Value	Item	Value	Item	Value
entity type	4	post	88,217,457	following	2,626,979
relation type	14	pin	347,131	follower	1,116,655
user	1,000,000	like	595,794	contain	1,998,788
hashtag	5,146,289	mention	4,759,388	discuss	66,000,633
list	21,870	retweet	1,580,643	bot	139,943
tweet	8,676,416	quote	289,476	human	860,057
user metadata	17	reply	1,114,980	entity	14,844,575
hashtag metadata	2	own	21,870	relation	170,185,937
list metadata	8	member	1,022,587	max degree	270,344
tweet metadata	20	follow	493,556	verified user	95,398

Fig. 2: Detailed Statistics for Twibot-22.

models with deep learning frameworks. First of all, GCN can solve the problem of classifying nodes in a graph. The idea is that you want to extract features from the nodes and their neighboring nodes in the network. Here is an example: we can use the stellarGraph library to support state-of-the-art machine learning algorithms on graphs. (i). Load the Twibot-22 dataset using Pandas (ii). Split the data for validation and testing (iii). Convert to numeric arrays (iv). Create GCN layers (v). Train the model (vi). Make predictions with the model. For specific steps to perform GCN model training and performance, please refer to our Github repository. Second, machine learning algorithms were beneficial for testing the accuracy and F1 score. For example, Random Forest (RFC) in classification increases the feature importance and predictive power, while Adaptive Boosting (ADA) assigns higher weights to wrong classifiers and improves the efficiency of binary classifiers. And Support Vector Machine (SVM) can find the different classes within the high-dimensional space as part of the classification task. K-Nearest Neighbor (KNN) predicts the average weight of the k-nearest neighbors and makes it suitable for the quick prediction phase.

Our dataset, **TWIBOT-22**[1], includes more than 1M records and eight graph-based structure files (please refer to Figure 2.). It contains node.json, tweet.json, user.json, list.json, hashtag.json, label.csv, split.csv, and edge.csv. The file node.json introduces twitter user information and entities including users, tweet, list, and hashtag. The split.csv has data split information, where the first column is the user id and the second column is the corresponding split (train, valid or test). Label.csv includes ground truth labels, where the first column is the user id and the second column is the corresponding label (human or bot). Edge.csv refers to the source_id, target_id and other relation types. Beyond this paper, we hope to utilize Twibot-22 for future GNN research aiming to build a solid foundation in social media detection models.

To address such a large dataset and challenges, 1) large memory requirement, and 2) long computation time for training, whether it is finding variable importance or performance matrix (e.g., accuracy, F1 score, and AUC). Our main contributions can be summarized as follows.

- We discover the novelty of leveraging semi-supervised GNN and supervised machine learning algorithms by generating prediction classifiers for graph-based node classification.
- We design a new interactive Tableau dashboard which explores the ML results and model performance trends by training GNN and ML models in the PyG library.
- We further extend GNN to augment the training data with one-hot encoding to predict the probability of labels using the cross-entropy loss function.
- Experiment results show that GCN outperforms all baseline models with a small gap. The accuracy and F1 score of GCN is among the highest compared to the machine learning algorithms (i.e, RFC, ADA, SVM, and KNN). Further analysis of hyperparameter tuning, increasing hidden layers, and other boosting methods can be recommended to improve the performance of the models, as well as comparing other results in future

[1]Dataset is avaliable at https://github.com/LuoUndergradXJTU/TwiBot-22

research.

II. Related Work

A. Supervised Machine Learning for Bot Detection

For years now, Social media platform, Twitter has attracted attention for malicious bots that are used for spreading misinformation, advertising, and other fake news activities. There are many machine learning algorithms that can be applied to solving the problem. In response to existing research papers or related work about detecting bot or malicious accounts, we propose a combination of GNN and supervised machine learning for bot detection.

Swe and Myo's contributions to early bot detection research have been successful to identify tweet-based features (the content of the tweet), relying on feature extraction from user data, tweet content, and network activity to classify accounts as either bots or legitimate users [5]. Despite the success of tweet-based features, it is important to include other features such as user profile-based features (the content of the tweet), user profile-based features(account age, number of followers), and social graph-based features (follower network structure) for node classification. It is efficient to incorporate machine learning algorithms such as K-Nearest Neighbors(K-NN), Decision Trees, Gaussian Naive Bayes, Support Vector Machines etc.

The idea that features vectors - the global outlier standard score and local outlier standard score - is uniquely employed in that it both identifies malicious accounts on a global and local scale. It is especially useful for detecting outliers that will be permissible when it comes to the presence of bot activity. For bot detection systems, bot-or-not's classification system applies supervised learning techniques – Deep Neural Networks (DNNs) and Convolutional Neural Networks (CNNs) – for detecting spam on Twitter. This is primarily relying on text-based classifiers. The system is measured by F1 score, which is a matrix of achieving high classification accuracy.

B. Graph-Based Machine Learning for Bot Detection

While machine learning was used more abstractly in data science, it often fails to consider the kind of relationships between users in a social network. Singh and Bansal's paper argue that graph-based models are more suitable in detecting bots in a network. First of all, Graph Convolutional Networks (GCNs) considered network structure and connections between users. Second, Graph Attention Networks (GATs) incorporate attention mechanisms to focus on the neighbors in the graph. Third, GraphSAGE will be able to generate embeddings by sampling and aggregating features from a node's nearest neighbors. Lastly, Graph Isomorphic Networks (GINs) is dedicated to capture more complex patterns in graphs and identified connection patterns for identifying bots. Utilizing these models can be an effective strategy to tackle the problem of node classification.[4]

C. Feature-Based and Propagation-Based Bot Detection

Zhao and Xin's research is focused on feature-based and propagation-based methods to characterize user features. What the approach meant was to find out how influential, in a social network, as the user has been detected as spam bot, which often relates to spreading advertisements and fake content. The method of using Markov Random Fields on similarity graphs is essential to detect spambots, as the edges represent the relationship between users and labels (bot or non-bot) are based on the similarity of the nodes. [8]

All of these methods were involved in detecting bots on malicious activities, such as spreading advertisements, malicious links, and fake news. By focusing on the propagation of harmful content and the relationships between bots and legitimate users, these models can not only identify the presence of bots but also whether they are engaged in malicious activities.

III. Problem Statement

Suppose that there is an attributed graph, where the node attributes are listed beside each node. Node v_1 is labeled as class 1 (Blue/regular user), Node v_5 is labeled as class 2 (Red/malicious account), and Node v_3 is labeled as class 3 (Purple/undetermined). We plan to train a one-layer GNN followed by a softmax layer for semi-supervised node classification. The message-passing rule of the one-layer GNN can be written as:

$$\boldsymbol{h}_i = \text{ReLU}\left(\alpha \cdot \boldsymbol{x}_i \boldsymbol{W} + (1-\alpha) \cdot \frac{1}{|N(v_i)|} \sum_{v_j \in N(v_i)} \boldsymbol{x}_j \boldsymbol{W}\right)$$

where $\boldsymbol{x}_i$ is a row vector denoting the node attributes of v_i, and $N(v_i)$ is the set of v_i's neighbors. $\boldsymbol{W}$ is the model parameter of the one-layer GNN. The softmax layer with v_i's representation $\boldsymbol{h}_i$ is used to predict the probability of node distribution v_i belonging to class k as:

$$P(y_i = k) = \frac{\exp(\boldsymbol{h}_i \cdot \boldsymbol{w}_k)}{\sum_{c=1}^{3} \exp(\boldsymbol{h}_i \cdot \boldsymbol{w}_c)}$$

where $\boldsymbol{w}_c$ is the model parameter of the softmax for class c. Note that for simplicity, we did not add the bias term in the softmax function. For node classification, we use cross-entropy as the loss function to predict as close to the ground truth labels as possible:

$$\mathcal{L} = \frac{1}{|V_L|} \sum_{v_i \in V_L} \sum_{c=1}^{3} -y_{ik} \log P(y_i = k)$$

where y_{ik} is the k-th element of the one-hot encoding of y_i.

IV. Methods

A. Baseline Model Training

Baseline model training is a critical initial step in a data science project. Baseline models serve as a reference point to analyze more complex models by providing comparisons to analyze it against and serving as a minimum standard for its performance. Beyond setting expectations for the performance of more complex models, these baselines also offer valuable insights into the dataset's predictive power and assessing

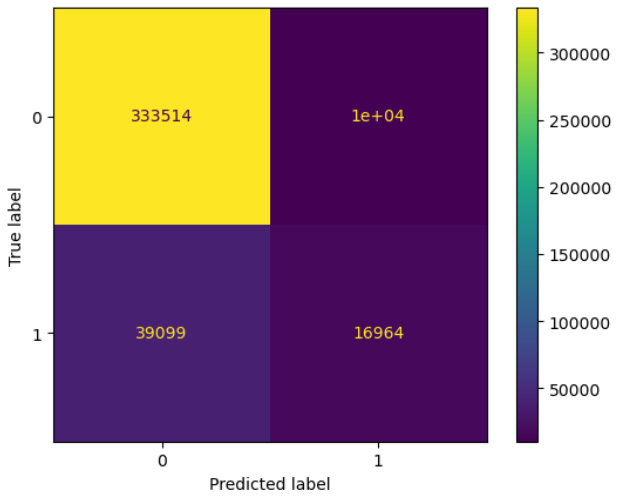

Fig. 3: Random Forest Classifier Confusion Matrix

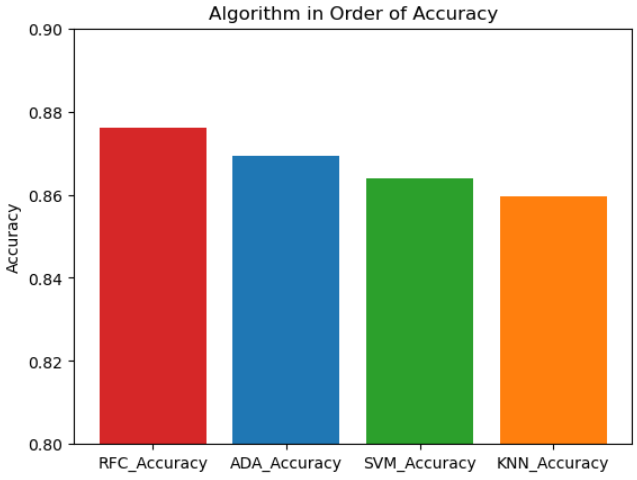

Fig. 4: Accuracy of Baseline Models

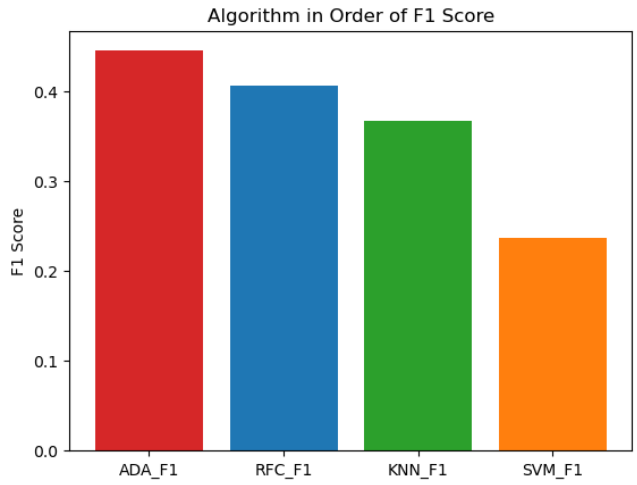

Fig. 5: F1 Scores of Baseline Models

whether the data adequately supports accurate predictions. Additionally, baseline models facilitate early error diagnosis, revealing potential issues such as weak features that may require further feature engineering or adjustment. Given the simplicity of our baseline models, they were generally quick to implement, enabling the testing of the data engineering pipeline and ensure data quality without significant time investment. Most importantly, as more complex models are developed, their performance gains can be measured directly against this baseline, validating that any added complexity results in substantial improvement. Therefore, baseline model training is essential for guiding project development and ensuring that each modeling iteration contributes tangible value.

The baseline models used were from SKLearn and consisted of predetermined algorithms. The four models included the Random Forest Classifier, Adaptive Boost Classifier, K-Nearest Neighbors Classifier, and Support Vector Machine, and a confusion matrix was created for all models. Our workflow was as follows: 1) scaling the data for certain baseline models, 2) splitting the data into training and testing sets, 3) importing models from the SKLearn library, 4) performing hyper-parameter tuning using randomized search and grid search, 5) fitting the models and predicting results, and 6) calculating confusion matrices and other important metrics.

Randomized search and grid search have distinct advantages. A grid search algorithm explores the entire hyper-parameter space and fits each set of hyper-parameters to the model, while a randomized search uses a random seed to calculate random combinations of hyper-parameters, stopping when it reaches a local or global maximum. The advantages of these two algorithms are clear: grid search can find a global maximum but is computationally expensive, whereas randomized search is cost-efficient but might not yield an optimal result.

The confusion matrix, which displays the number of actual values compared to the model's predicted values, was used to determine accuracy and F1 scores. Figure 3 shows one of the four confusion matrices created for each of the models, with this example coming from the Random Forest Classifier. When both the true and predicted labels match, they are classified as either true positives or true negatives. When the predicted value does not match the true value, they are classified as false positives or false negatives. The goal of any model is to maximize the number of true positives and true negatives while minimizing the number of false positives and false negatives.

In figure 4, we see the accuracy of the baseline models from SKLearn compared to one another. Accuracy is calculated by take the true positives and negatives divided by all values which produces the percentage of correctly labeled values. All four have an accuracy between 0.86 and 0.88. These are relatively good accuracies and provide us with the idea that our Graph Convolution Neural Network needs to have an accuracy greater than 0.88 to have a more positive impact than the SKLearn algorithms that are much less complex.

In figure 5, the F1 scores of the baseline algorithms from SKLearn are displayed. F1 scores are used to determine accuracy when the dataset is imbalanced. Together with ac-

Feature	Importance
follower_count	0.25
tweet_count	0.17
description_False	0.14
description_True	0.13
following_count	0.13
listed_count	0.07
year_created	0.05
entities_True	0.01
entities_False	0.01
location_True	0.01
location_False	0.01
protected_True	0.01
protected_False	0.01
verified_True	0.01
verified_False	0.01
url_False	0.01
url_True	0.00
withheld_True	0.00
withheld_False	0.00

Fig. 6: Variable Importance Table

curacy, these scores can give insight into the performance of a model. From figures 4 and 5, we can determine that the two best baseline models were Adaptive Boost Classifier and Random Forest Classifier. Our goal will be to develop a Graph Convolutional Neural Network that has greater accuracy and F1 scores than those two baseline algorithms.

Aside from the metric, the Random Forest Classifier from SKLearn also comes with a variable importance attribute that can be used to generate an importance table. In figure 6, it's easy to tell that variables like follower_count, following_count, tweet_count, description_True, and description_False are all importance variables that contributes to the model the most.

B. Graph Convolutional Neural Network

For our deep learning framework, we decided to utilize the PyTorch Geometric (PyG) library, which is built on PyTorch and specializes in training Graph Convolutional Networks (GCNs) and other Graph Neural Networks (GNNs). The PyG data object provides a user input and an edge input, eliminating the need to manually combine relationship and user data. To start, we encoded the user IDs into unique integers and selected all edges with existing user IDs using mapping functions. We created appropriate tensors with numerical variable features,

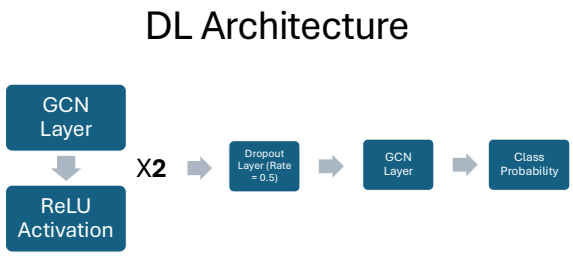

Fig. 7: GCN architecture

transformed the edge list, and normalized the final data object to ensure proper scaling.

Our GCN architecture consists of two replicated GCN layers, each followed by a ReLU activation function to introduce non-linearity. These layers are then fed into a dropout layer with a 50% dropout rate to prevent over-fitting (see Fig. 7). The architecture outputs class probabilities (logit), which are transformed into class labels using a cross-entropy loss function for optimization.

For the training loop, we set the number of epochs to 50,000 and ran the deep learning script on the Bridge2 cluster, as Pittsburgh's Super-computing Center was essential for handling the computational demands. The training process required approximately 36 hours to complete, reflecting the complexity of our dataset and model.

V. Result

A. Tableau

One of the novel aspects of our work is the presentation of results using a Tableau dashboard. The dashboard (Fig. 8) highlights key insights from both our exploratory data analysis (EDA) and deep learning results.

The first important insight is presented in the "Bot Distribution Comparison" visualization, which emphasizes the clear differences between bot and human users based on their follower counts. The second insight from our EDA is the proportion of key variables, such as the protected user ratio, the verified user ratio, and the bot distribution, all integrated with the variable importance table. These ratios provide valuable context, although their contribution is somewhat limited due to the small number of Twitter users categorized as protected or verified.

On the other hand, the performance comparison demonstrated that the GCN model is rank third in terms of accuracy but first in terms of F1 score. The result doesn't go as planned, but having the highest F1 score means that the GCN model performs the best in handling imbalanced data. In addition to the comparison, the loss trend stated that the best performance model occurred at around 10,000 epoches, but the loss is very stable from 10,000 to 50,000 epoches.

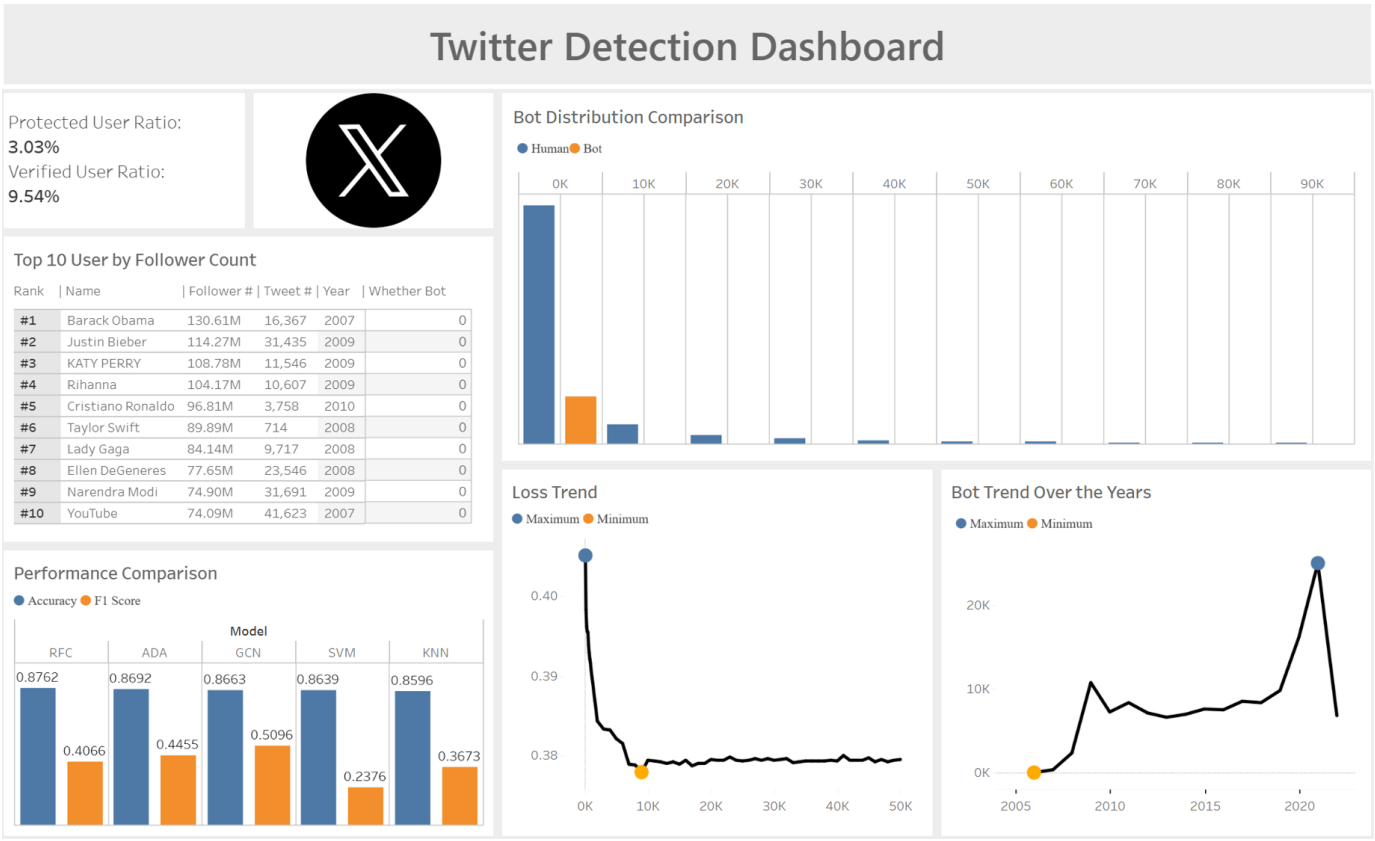

Fig. 8: Tableau Dashboard

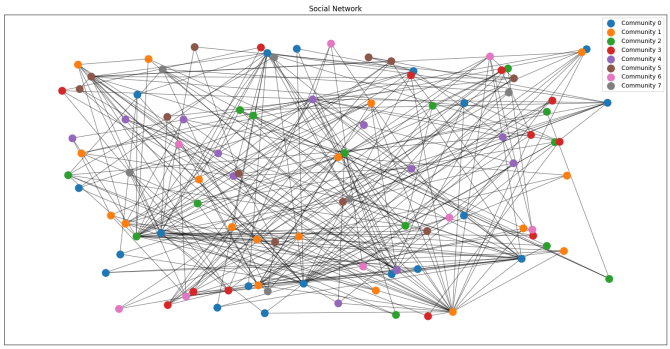

Fig. 9: NetworkX

B. NetworkX and AUC Analysis

In this subsection, we analyze the visualization of the GCN model using NetworkX, a python package dedicated for creating, analyzing, and studying complex graph networks. To conduct NetworkX analyzes, we use torch_geometric.utils and torch_geometric.nn libraries. First of all, we examine the sample and sample_edge dataset. Then we successfully created a graph with G= nx.Graph() by utilizing the nodes and edges from the dataset. Second, we performed BFS and DFS on the edges. Next, we completed graph community detection on the nodes that are most similar to each other. The NetworkX visualized five different communities on the subset graph of 1000 nodes. Finally, we visualized the GCN layers by using nx.barabasi_albert_graph function. From the Figure 9, we can observe that:

- GCN adapts well to NetworkX visualizations. The complexity of layers are outlined in the social network where each edge denotes a follower or following relationship between each user.
- Compared with machine learning baseline models, the major difference is that the accuracy is relatively higher because it captures a complex relationship in node representation. NetworkX is more flexible and can model more complex relation information, and therefore is more computationally expensive to train. The embedding space needs to be normalized before training the graph. In this example, we trained a subgraph of the dataset which takes less time and results in a higher accuracy score.
- It is unknown to us which cluster or community is more likely to be malicious accounts. Given the seven communities which denotes in seperate colors, we can use the most influential node and less influential node to predict which node is a malicious account. For example, blue nodes tend to fall on the edge of social networks. It may not include the same amount of followers as the regular users to reach the standard activity. Please refer to our Github[2] repository for the detailed code.

Lastly, the AUC score of the GCN model is comparably higher than all other baseline models, which supports the conclusion of the Tableau Dashboard. GCN, indeed, is the

[2]Find the code at https://github.com/Eric7895/DS_340W_Course_Project

most effective model to perform malicious account detection. Unfortunately, the main author of the paper did not save the AUC curve after viewing it the first time. The AUC curve of GCN, RFC, ADA, SVM, and KNN looked close to each other and GCN is slightly above the other models. It is believed that after one week of computationally expensive process, the AUC curve did not survive the one million case of GCN training. Therefore, GCN can handle the Two-bot22 dataset or our assumption of any detection tasks well.

VI. Discussion

The results of our project highlight key strengths and limitations of Graph Convolutional Networks (GCNs) compared to traditional supervised machine learning methods for malicious account detection on social media platforms. GCNs demonstrated superior F1 scores, emphasizing their capacity to handle imbalanced datasets by effectively integrating graph structure and node-level features. However, their accuracy lagged behind Adaptive Boosting (ADA) and Random Forest (RF), suggesting that while GCNs excel at identifying the minority class, their performance on the majority class may require optimization.

The strong performance of ADA in terms of accuracy reflects its ability to adaptively focus on hard-to-classify samples, making it a possible alternative when high overall accuracy is prioritized. However, its lower F1 score highlights challenges in effectively detecting malicious accounts within the imbalanced dataset. Due to the nature of our problem, most datasets that reflect the true ratio of human accounts and malicious accounts will be imbalanced, with more human accounts compared to malicious accounts. This trade-off between accuracy and the F1 score is crucial in applications where minimizing false negatives is a priority, as missed malicious accounts can pose significant security risks and decreased user experience. In this scenario, it is also crucial to minimize false positives, as misidentified accounts thought to be malicious could be removed when, in reality, they are a human account, upsetting users of the platform.

GCNs also introduce computational complexity that may limit scalability for larger datasets. While they leverage graph topology, this comes with increased resource requirements, particularly for datasets with high connectivity. Traditional methods such as RF and ADA, by contrast, offer computational simplicity, making them practical in scenarios where real-time detection is essential or computational resources are constrained.

Feature importance analysis revealed that node-level features played a vital role across all models, underscoring their value in distinguishing malicious accounts. However, GCNs benefited uniquely from incorporating edge relationships, suggesting potential for further research in optimizing graph-based feature engineering. Future work could explore hybrid approaches combining the interpretability and efficiency of traditional models with the contextual depth of graph-based methods.

These findings emphasize the importance of model selection based on specific application priorities, such as prioritizing recall for security-critical tasks or accuracy for broader classifications. Further research could to test different class weight and penalized models for guess human users to some extent, used the area under Receiver operating characteristic curve (AUC) as it's an important metrics in testing model's ability to handle imbalanced data, and address the scalability of GCNs and develop strategies to enhance their performance in detecting majority-class samples, ensuring their applicability in large-scale, real-world scenarios.

VII. Acknowledgments

The authors thank the course faculty, Dr. James Wang and our TA Sree Bhattacharyya, for their continued support throughout the semester. Their feedback and recommendations as we progressed through our project were crucial. The authors would also like to thank Pittsburgh's super-computing center for allowing us to use their quantum computing in providing the computational power that was necessary to run our deep learning model. The authors also thank the TwitBot-22 research team that were kind enough to share their data with us. This project would not have been possible with their team sharing the TwitBot-22 datasets with us.

With respect to individual writing contributions from the authors of this paper:

- Chase Cunningham is responsible for the discussion and conclusion section.
- Shen-Han Chiu is responsible for the introduction, related works, problem statement section, as well as NetworkX and AUC Analysis subsection.
- Eric Wu is responsible for the methods and results section.

All authors of the paper contributed to all aspects of this project.

VIII. Conclusion

In this pr0ject, we demonstrated that Graph Convolutional Networks (GCNs) provide a promising approach for malicious account detection by leveraging graph structure and node features, particularly excelling in F1 performance. While traditional models like Adaptive Boosting and Random Forest offer competitive accuracy with lower computational demands, it is shown that GCNs approach to dealing with large dataset can be essential for nuanced detection tasks.

These results further investigate the necessity of aligning model choice with the specific requirements of social media real-world applications, whether that's prioritizing detection accuracy, computational efficiency, or the ability to handle imbalanced datasets. Future research should focus on improving the scalability of graph-based models and exploring innovative solutions to bridge the gaps of both GNN and machine learning paradigm. There are many real-world methods which can be treated as node classification problems to detect malicious activity in increasingly complex social media environments.

REFERENCES

[1] Feng, S., Tan, Z., Wan, H., Wang, N., Chen, Z., Zhang, B., Zheng, Q., Zhang, W., Lei, Z., Yang, S., Feng, X., Zhang, Q., Wang, H., Liu, Y., Bai, Y., Wang, H., Cai, Z., Wang, Y., Zheng, L., . . . Luo, M. (2022, June 9). TwiBot-22: Towards Graph-Based Twitter Bot Detection. arXiv.org. https://arxiv.org/abs/2206.04564

[2] T. Zhao, X. Zhang and S. Wang, "Imbalanced Node Classification With Synthetic Over-Sampling," in IEEE Transactions on Knowledge and Data Engineering, vol. 36, no. 12, pp. 8515-8528, Dec. 2024, doi: 10.1109/TKDE.2024.3443160. keywords: Task analysis;Training;Interpolation;Chatbots;Classification algorithms;Topology;Training data;Node classification;imbalanced learning;graph;data augmentation;graph neural network

[3] Ilias, L., Roussaki, I. (2021). Detecting malicious activity in Twitter using Deep Learning Techniques. Applied Soft Computing, 107, 107360. https://doi.org/10.1016/j.asoc.2021.107360

[4] L. Caruccio, G. Cimino, S. Cirillo, D. Desiato, G. Polese, and G. Tortora, "Malicious account identification in social network platforms," ACM Transactions on Internet Technology, vol. 23, no. 4, pp. 1–25, Nov. 2023, doi: 10.1145/3625097.

[5] Monika Singh, Divya Bansal, and Sanjeev Sofat. 2014. Detecting Malicious Users in Twitter using Classifiers. In Proceedings of the 7th International Conference on Security of Information and Networks (SIN '14). Association for Computing Machinery, New York, NY, USA, 247–253. https://doi.org/10.1145/2659651.2659736

[6] M. Swe and N. N. Myo, "Detecting Malicious Users on Twitter Using Topic Modeling," 2023 IEEE Conference on Computer Applications (ICCA), Yangon, Myanmar, 2023, pp. 214-219, doi: 10.1109/ICCA51723.2023.10181604. keywords: Accesslists;Social networking (online);Blogs;Knowledge discovery;User experience;Blocklists;Grippers;whitelist;blacklist;legitimate;spammer,

[7] Pakaya, Farhan Ibrohim, Muhammad Budi, Indra. (2019). Malicious Account Detection on Twitter Based on Tweet Account Features using Machine Learning. 1-5. 10.1109/ICIC47613.2019.8985840.

[8] X. Wang, Y. Sui, Y. Tao, Q. Zhang, and J. Wei, "Detecting Abnormal Social Network Accounts with Hurst of Interest Distribution," Security and Communication Networks, vol. 2021, pp. 1–14, Jun. 2021, doi: 10.1155/2021/6653430.

[9] Zhao C, Xin Y, Li X, Zhu H, Yang Y, Chen Y. An Attention-Based Graph Neural Network for Spam Bot Detection in Social Networks. Applied Sciences. 2020; 10(22):8160. https://doi.org/10.3390/app10228160

[10] Vatter, J., Mayer, R., & Jacobsen, H. (2023). The evolution of distributed systems for graph neural networks and their origin in graph processing and deep Learning: a survey. ACM Computing Surveys, 56(1), 1–37. https://doi.org/10.1145/3597428

[11] T. Le, S. Wang and D. Lee, "MALCOM: Generating Malicious Comments to Attack Neural Fake News Detection Models," 2020 IEEE International Conference on Data Mining (ICDM), Sorrento, Italy, 2020, pp. 282-291, doi: 10.1109/ICDM50108.2020.00037. keywords: Social networking (online);Biological system modeling;Ecosystems;Detectors;Machine learning;Data models;Robustness;Fake News;Adversarial;Attack;Malicious Comments;MALCOM;Misinformation;social media,

[12] W. Fan et al., "Jointly Attacking Graph Neural Network and its Explanations," 2023 IEEE 39th International Conference on Data Engineering (ICDE), Anaheim, CA, USA, 2023, pp. 654-667, doi: 10.1109/ICDE55515.2023.00056. keywords: Perturbation methods;Image edge detection;Self-supervised learning;Inspection;Predictive models;Feature extraction;Graph neural networks;Graph Neural Networks;Adversarial Attacks;Explanations;GNNEXPLAINER;Trustworthy GNNs,