Network Anomaly Detection

Anish Saha asaha19@stanford.edu
Vinay Sawal vsawal@stanford.edu
Vincent Ying vhying@stanford.edu

Repository: https://github.com/anish-saha/Network-Anomaly-Detection
Reproduction: https://worksheets.codalab.org/worksheets/0xb972380293b449d896f3c0ade600ee05

Abstract

In this project, we try to construct a robust prediction algorithm to predict whether or not a social network user is a Sybil (fake account) using artificial intelligence. Our training data is a network dataset of over 5.3 million Twitter users. [5] To accomplish this, we utilized SciKit-Learn, NetworkX, and Pandas to develop a two-step vertex classification pipeline involving common methodologies such as Random Forests, Gradient Boosting Machines, and Logistic Regression. [6] We also experimented with Graph Neural Networks (GNNs) [12] such as Graph Convolutional Networks (GCNs) [7] and Graph Attention Networks (GATs) [9] to optimize performance and improve accuracy. Our results were convincing enough in terms of accuracy, but unfortunately there is much work to be done in terms of improving the other performance metrics for our models, such as true positive rates. To improve our results, there are several optimizations to consider - collaborating with a large social network company to gather a reliably labelled social network graph dataset with additional non-graphical features, increasing the size of our dataset, and utilizing more powerful computational resources to leverage sparse models would all help improve the performance of our models.

1. Introduction

Alongside the meteoric rise of social networks has come the alarmingly prevalent problem of malicious users and bots that steal personal data and manipulate social media trends. Millions of fake accounts are created for malicious use and are responsible for a growing number of threats, including spamming, fake product reviews, and astroturfing for political or commercial purposes. There exist entire complex networks of fake users and bots that are created for the purposes of spreading misinformation, sensationalizing trends, and bolstering the popularity of social media entities. [4] As such, we aim to solve the problem of detecting fake accounts (Sybils) in social network graphs, as this could have a significant positive impact on society.

Using a network graph dataset representing the directed edges connecting the constituent 5,384,160 Twitter users, the goal of this project is to detect anomalous users as effectively as possible.

2. Related Work

Sybil detection methods can be separated into feature-based and graph-based models. In feature-based methods, user attributes and behavior are encoded as features for ingestion with machine learning classifiers. [10] Deep neural networks have also been utilized for this method in a variety of ways, e.g. the classification of tweet messages for bot detection by Kudugunta et al. [8] For graph-based methods, there are algorithms, such as *SybilRank* [2], that crawl the graph structure in the same manner as PageRank in order to ascertain the network topology based attributes for real and fake accounts.

A combination of feature and graph based methods has better performance and is more robust than either method by themselves. In Integro [1], a two phase method where victim accounts are classified by user-level activities and a modified random walk is made with edges incident to predicted victims having a much lower weight. Victim accounts are first classified because attackers have no control over those accounts and Sybils are directly connected to victim accounts.

Another combined, two-phase method is utilized by Kagan et al. to detect anomalous vertices, but they utilize the graph structure in the first phase to generate features for the second phase. First, they perform network topology-based link prediction on a vertex with a Random Forest classifier. Then, the link predictions from the classifier are utilized to identify anomalies in an unsupervised manner. [6]

Daya et al. also utilize a two-phased, graph-based system with supervised and unsupervised algorithms to perform botnet detection. Their first phase prunes benign hosts with the use of three unsupervised clustering algorithms. Then,

their second phase detects bots with four supervised learning algorithms. [3]

3. Dataset

The dataset we will use is a Twitter graph that is comprised of anonymized user ids, directed edges between two users, and real/fake labels obtained from Ben Gurion University in 2014. [5]

Network	Directed	Labeled	Vertices	Edges
Twitter	True	True	5,384,160	16,011,443

The output of this classification problem is a binary target variable representing whether or not each user is identified as a fake user. This is a supervised learning classification task - we have a definitive source of truth to test our models.

For example, let's consider that user `a` is friends with `b` and `c`, and `a` is a fake account with the others being real.
Input is represented as a Source and Destination list, [(`a`, `b`), (`a`, `c`)] and Fake list, [`a`].
Output is represented as a list of predicted labels, [(a, Fake), (b, Real), (c, Real)]

The problem we will attempt to tackle is Sybil detection in social network graphs. The input of our problem is a graph dataset of over 5.3 million Twitter users. After completing exploratory data analysis, we realized that our dataset was heavily skewed right and contained millions of users who had 0 followers or followed 0 users, meaning that they had little to no effect on the network. Any metrics derived from such users would be meaningless, and as such, we decided to omit these users.

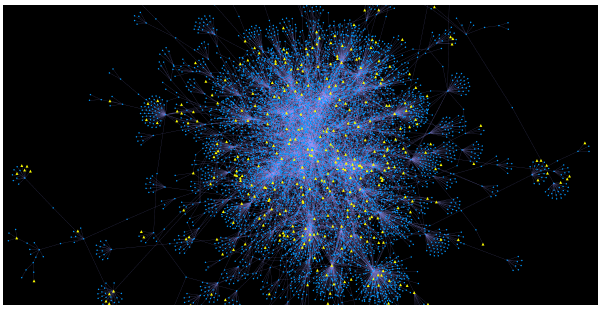

Figure 1: Twitter Social Network Graph Sample

Due to the nature of the dataset, we filtered the dataset down to only 75,624 users, removing all users who had 0 followers or followed 0 users, who subsequently would have little to no effect on the network. Figure 1 shows the network from the filtered dataset.

3.1. Dataset Splits

Such a huge dataset presented compute resource challenges during our experiments. Thus we adopted a different strategy for picking dataset based on the underlying model and compute resource available. Here are the details on dataset selection for each model:

- **Two-Stage Classifier:** We split the dataset using random sampling to extract 10,000 validation set users and 2,000 test set users.
- **Graph Convolution Network:** Since we used vertex embeddings and vertex features of the graph network as the input for Neural Network, we split the datset of 75,642 users in 80% training set, 10% validation set and 10% test set.
- **Graph Attention Network:** Graph Attention Networks require enormous amount of RAM since it utilizes masked self-attention layer. With the limited resources available, we had to further reduce the dataset by randomly sampling 5,000 nodes and splitting it in 80% training set, 10% validation set and 10% test set.

4. Methods

4.1. Two-stage Classifier

For the initial model in our project, we utilized a two-step architecture and vertex/edge features to classify whether a node is real or fake.

The first step of the model utilizes standard classifiers that leverage various features of the graph's edges to output the probabilities of those directed edges existing between two given vertices. Intuitively, a larger number of highly unlikely edges would indicate an anomalous user.

The second step of our approach utilizes these edge link probabilities as the input to accumulate and perform link prediction with a `threshold = 0.5`. An anomalous prediction is made for the vertices that have many unlikely edges. [6]

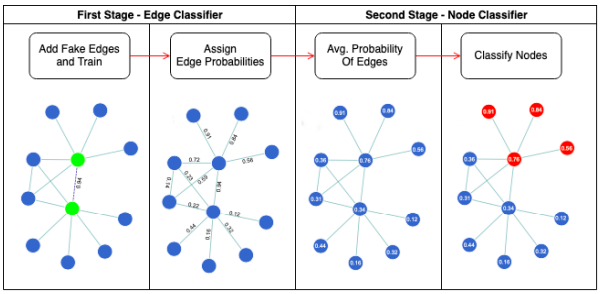

Figure 2: Link Prediction Classifier

4.1.1 First Stage Classifiers

Various standard classifiers were used in the first stage to improve the performance of the two-stage model:

- **Logistic Regression:** We optimized the Logistic Regression classifier model by exploring different feature combinations and their efficacy in generating edge link probabilities to improve the performance of the two-step procedure in predicting anomalous users within the test set. We experimented with different hyper-parameter configurations and found that the largest feature set yielded the best results.
- **Random Forest:** We optimized the Random Forest classifier model by exploring feature selection to predict whether or a given vertex is anomalous. We also experimented with different hyper-parameter configurations. Again, the largest feature set seemed to yield the best results for this particular model.
- **Gradient Boosting:** After these experiments, we decided to explore Gradient Boosting models as an alternative methodology for finding anomalous edges and generating edge probability predictions in the first step of our approach. In our gradient boosting classifier, we used exponential loss function with fractional subsample of `0.90` for fitting. This technique seemed to show a significant reduction in variance and improved the model's performance on both the validation set and the test set.
- **Bagging with Decision Trees:** We created an optimized tree-based approach using model leveraging the principle of bagging, which helps to address overfitting by reducing variance. This ensemble classifier model acts as a substitute methodology for finding anomalous edges and generating edge probability predictions during the the first part of our two-step classifier architecture.

4.2. Graph Neural Networks

Graph neural networks (GNNs) are connectionist models that capture the dependence of graphs via message passing between the nodes of graphs. Unlike standard neural networks, graph neural networks retain a state that can represent information from its neighborhood with arbitrary depth [12]. We experimented with two variants of GNNs as described in the following section and observed significant improvements in accuracy.

4.2.1 Graph Convolutional Network (GCN)

Using the vertex embeddings of the graph network based on local network neighborhoods as inputs for the neural network, we leveraged a Graph Convulational Network to predict anomalies within the graph dataset. Passing the inputs through an optimized 3-layer convolutional neural network, this model improved performance compared to standard machine learning methods on both the validation set and the test set. [7]

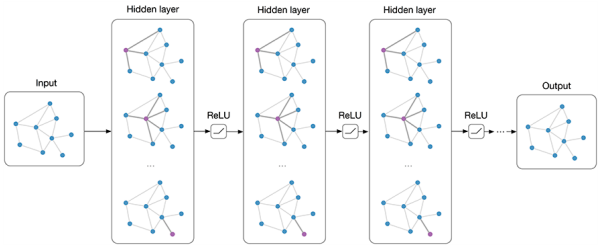

Figure 3: 3-layer GCN

Here, $H^{(l)}$ denotes the l^{th} layer in the network, σ is the non-linearity, and W is the weight matrix for this layer. D and A, as commonly seen, represent degree matrix and adjacency matrix, respectively. The $\sim$ is a re-normalization trick in which we add a self-connection to each node of the graph, and build the corresponding degree and adjacency matrix. The shape of the input $H^{(0)}$ is $N \times D$, where N denotes the number of nodes and D denotes the number of input features. We can chain up multiple layers as such to produce a node-level representation output with shape $N \times F$, where F represents the dimension of the output node feature vector.

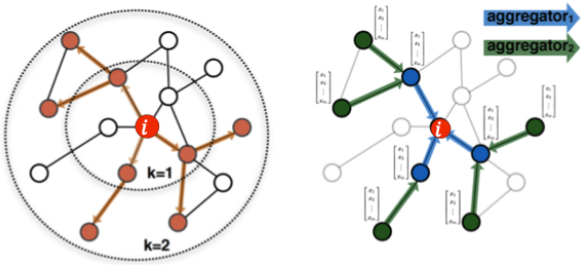

Figure 4: Vertex Embeddings based on local neighborhood

Mathematically, the GCN model follows the formula below.

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

4.2.2 Graph Attention Network (GAT)

We also explored the use of a Graph Attention Network, another specialized form of Graph Convolutional Networks. It utilizes masked self-attention layers that can bypass the limitations of methods involving graph convolutions and related approximations. [9]

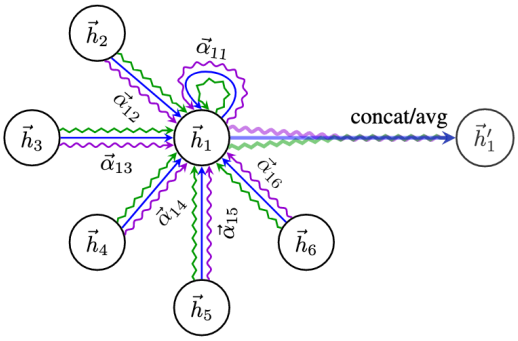

Figure 5: GAT layer with multi-head Attention [9]

As seen in figure 5, every neighbour i of node 1 sends its own vector of attentional coefficients, $\vec{\alpha}_{1i}$ one per each attention head α^k_{1i}. These are used to compute K separate linear combinations of neighbours' features $\vec{h}_i$, which are then aggregated (typically by concatenation or averaging) to obtain the next-level features of node 1, $\vec{h}'_1$.

$$\vec{h}'_i = \|_{k=1}^{K} \sigma \left(\sum_{j \in \mathcal{N}_i} \alpha^k_{ij} \mathbf{W}^k \vec{h}_j \right)$$

5. Optimizations

5.1. Feature Engineering

For this project, we extracted features from the graph dataset and created three feature sets from the filtered data, each with a different size.

Initially, only a few features were included in the small edge feature set for the baseline models. The baseline version only used the features `In-degree`, `Out-degree`, and `Jaccard's Coefficient`, we discovered that using larger feature sets consistently improved training set, validation set, and test set performance significantly.

In order to optimize the first step of our pipeline to more effectively predict link probabilities, we used the `NetworkX` Python library to extract all vertex features, corresponding to all users, as well as to extract various edge link features, e.g. `Common Friends`, `Adamic-Adar Index` (vertex relation strength), and `Preferential Attachment Score`. Our architecture leveraged the `NetworkX` library to compute a few different centrality measures (`In-degree centrality`, `Out-degree centrality`, `Degree Centrality`) as potential vertex features; the goal of this task to reduce variance on both the validation set and test set. We also experimented with other composite synthetic features (`Strongly-Connected Components (SCC)`, `Weakly-Connected Components (WCC)`, and `Average Neignbor degree`) to train our models, which helped to optimize validation set and test set performance.

After experimenting with a large array of different feature sets involving various combinations of synthetic features and analyzing feature importance metrics with our classification models, we decided to use the following features for our classification models:

- **Small -** The small feature set consists only of the similarity measure `Jaccard's Coefficient` and `In/Out Degree` of a node. This was utilized in the baseline model.
- **Medium -** In addition to the features in the small feature set, we include `Preferential Attachment Score`, `KNN weights`, `Common Friends`, and `Transitive Friends`.
- **Large -** In addition to the features in the small and medium feature sets, a `friend` measure is added. It measures the number of connections between the neighborhoods of two nodes. This feature set includes features for both edges and vertices in the graph. Running our models with the large feature set seemed to provide the best validation and test set performance results for all methodologies explored below.

All of the features were utilized in both steps of the classification pipeline approach; various classification methods leveraged these features to generate link probabilities, while the second stage of the classifier utilized that output to generate predictions on whether or not each vertex corresponds to a real user. Meanwhile, the neural network approaches only leveraged the vertex features to iteratively learn the mannerisms of fake and real users to generate predictions.

For a complete listing of all vertex and edge features utilized, see appendix 10.1.

5.2. Hyper-parameter Tuning

We utilized the `RandomizedSearchCV` library from `SciKit-Learn` to efficiently optimize hyper-parameter selection for training our classification models. Since the hyper-parameter combination is chosen using independent random sampling for each iteration.
As such, the formula for obtaining the probability that `RandomizedSearchCV` will retrieve at least one hyper-parameter configuration within 5% of the optimum for the given model in n iterations will be:

$$P(X) = 1 - (1 - 0.05)^n$$

Random variable X corresponds to the event that at least one of the configurations found in n iterations is within 5% of the optimum hyper-parameter configuration.

We elected to use 90 iterations (cross-validated twice) to achieve a 99% probability of selecting hyper-parameters within 5% of the optimal configuration. Each classification model uses different default parameters, so we delved into the specifics, choosing to experiment with various parameters such as `learning rate` (for AdaBoost / Gradient Boosting classification models) or `maximum tree depth` (for Random Forest classification models). While computationally intensive, this step of the process helped to improve the performance of all of our models significantly.

For the graph neural networks, several hyper-parameters were tried but there wasn't much significant effect on the accuracy for the test set. For a complete listing of GCN and GAT hyper-parameters refer to appendix 10.2.

6. Experiments

6.1. Baseline Results

To train our baseline models, we ran Logistic Regression and Random Forest classification on the edges descriptor dataset generated from the graph, with only the `Jaccard's Coefficient` feature. The results of the baseline models are in the below table. These results were generated on the filtered dataset.

Table 1: Baseline Results

Classifier	Accuracy
Logistic Regression	66.00%
Random Forest	77.36%

For the baseline, we have only calculated accuracy for the two SciKit-Learn prediction models. We will include other performance metrics to evaluate our improvements on the base models, such as F1 Score (combination of precision and recall) and AUC (Area under ROC curve). These metrics will be measured on the training and test sets.

6.2. Varying Feature Set Sizes

After filtering the dataset, we tried several standard models with varying feature set sizes, small, medium, and large. The results from the different features sizes on accuracy are listed in the tables below.

Table 2: Accuracy of Classifiers with Small feature set

Classifier	Validation set	Test set
Logistic Regression	76.94%	66.00%
Random Forest	83.09%	77.36%
Adaboost	**84.78%**	**82.09%**
Bagging w/ Decision Tree	81.80%	79.50%
Gradient Boosting	83.85%	80.09%

Table 3: Accuracy of Classifiers with Medium feature set

Classifier	Validation Set	Test Set
Logistic Regression	85.56%	69.72%
Random Forest	88.05%	84.00%
Adaboost	88.17%	85.95%
Bagging w/ Decision Tree	87.53%	86.23%
Gradient Boosting	**88.88%**	**86.27%**

Table 4: Accuracy of Classifiers with Large feature set

Classifier	Validation set	Test set
Logistic Regression	84.39%	68.95%
Random Forest	90.05%	87.70%
Adaboost	88.82%	88.22%
Bagging w/ Decision Tree	88.90%	88.36%
Gradient Boosting	**90.74%**	**87.77%**

As can be seen in figure 6, increasing the feature set improved accuracy for the two-stage model, no matter which classifier is used.

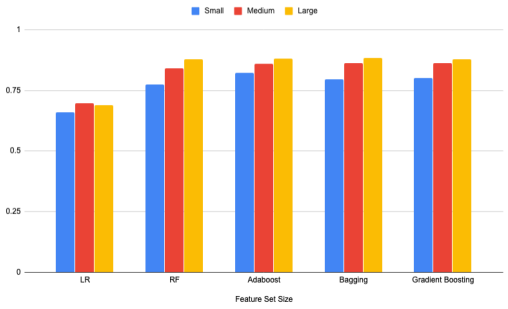

Figure 6: Two-stage model accuracies on small, medium, large feature sets

6.3. Final Test Results

As seen in results below, the test set accuracy increased as the number of features utilized increased. Hyperparameter optimization seemed to improve the results slightly. Among the two-stage classifier models, we see that Gradient Boosting performed the best on the three feature sets

while GNN based models performed significantly better than two-stage classifiers. The best performing model was the GAT, achieving an accuracy of 95.6%.

Table 5: Final accuracy results for Two-stage Classifiers

Classifier	Validation set	Test set
Logistic Regression	84.39%	88.15%
Random Forest	90.05%	87.97%
Adaboost	88.82%	88.60%
Bagging w/ Decision Tree	88.90%	88.54%
Gradient Boosting	**90.74%**	**88.79%**

Besides accuracy, the two-stage model did not perform as well for other calculated metrics (i.e., precision, recall, F1, and AUC). In the context of this problem, recall (percent of fake users classified out of the total number of fake users) should be more important than precision (percent of fake users correctly classified out of the total number of fake classifications). This is because fake users can be flagged for further inspection. AUC (area of false positive rate vs true positive rate) is another important metric. The table below shows results for other metrics. In terms of AUC, the Adaboost classifier performed the best.

Table 6: Other performance metrics for Two-Stage Classifiers

Classifier	Precision	Recall	F1	AUC
LR	0.112	0.06	0.0782	0.516
RF	0.114	0.055	0.0743	0.526
Adaboost	**0.105**	**0.045**	**0.0631**	**0.557**
Bagging w/ DT	0.0752	0.035	0.0477	0.495
Gradient Boosting	0.0841	0.045	0.0586	0.507

For the two-stage model, we also obtained precision at K with different first stage classifiers. After training with the large feature set and utilizing hyperparameter search for 90 iterations, the best classifiers for precision at K are two boosting classifiers, Adaboost and Gradient Boosting.

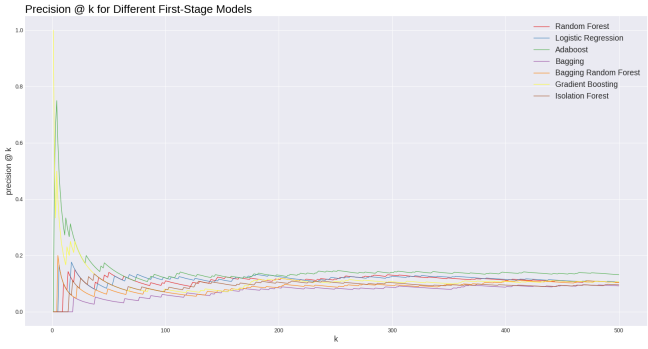

Figure 7: Precision at K for first stage Classifiers

Logistic regression and all of the other decision tree classifiers did not perform well, achieving 0.0 for the top 1 to top 10. The precision at K is given below for the two boosting models at $k = 1, 5, 10$.

Table 7: Precision @ K for best performing Classifiers

Classifier	K = 1	K = 5	K = 10
Adaboost	0.0	0.6	0.3
Gradient Boosting	**1.0**	**0.4**	**0.2**

Finally, we saw further improvements in the accuracy for both Graph Neural Networks as compared to two-stage Classifiers. Results are presented in following table.

Table 8: Final accuracy results for Graph Neural Networks

Classifier	Validation set	Test set
GCN	94.18%	94.76%
GAT	**93.61%**	**95.6%**

Table 9: Other performance metrics for Graph Neural Networks

Classifier	Precision	Recall	F1	AUC
GCN	0.947	1.000	0.973	0.501
GAT	**0.956**	**1.000**	**0.977**	**0.557**

7. Error Analysis

There are many potential sources of error in the models utilized above that may have caused sub-optimal performance metrics. One of the most important causes of error could be mislabelled data. We utilized the Twitter social network dataset provided in the baseline architecture. As stated in the repository containing the dataset from Ben Gurion University, this Twitter dataset was retrieved and labelled using a scraping tool that pulled millions of Twitter users, then labelled users as anomalous if they were removed when Twitter was scraped one year later in 2014. However, this is a source of error, since users who deleted their account would be mislabelled as fake, while fake users who were not detected and removed by Twitter would be mislabelled as real. Although we cleaned the data and removed all users who had 0 followers or 0 users following to mitigate this issue (since inactive accounts are not necessarily bots or fake users, they could likely also be experimental or deleted), this would not account for the possibility that many other users may have been mislabelled. To truly mitigate this source of error would likely require some level of collaboration with Twitter to retrieve more reliable data for the purposes of training our models. We could also further train the model to detect anomalous behavior by including more training data, either

from Twitter, or from another social network graph, such as Facebook, Instagram, or LinkedIn. While our models were able to identify the anomalies in a social network graph in many circumstances with high levels of accuracy, the other performance metrics were sub-optimal. Unfortunately, our models can only be good as the data they are based on.

Another major source of error lies in the nature of Sybil accounts. It turns out that large networks of fake users and bots are ubiquitous on Twitter, and many such networks closely resemble the complex structures of real Twitter users. [4] This causes our classification models to struggle at differentiating between genuine friend networks and fake user networks. Some estimates state that there may even be tens of millions of such fake users and bots. Thus, it would difficult to determine whether or not a user is fake or not solely based on the network graph data for its associated vertex. To circumvent this issue, a more detailed dataset would be necessary so that we could leverage a combination of graph-based features as well as other user-specific metrics, such as NLP-based features regarding the content of the user's tweets or the user's rate of follower growth, to more effectively detect Sybil accounts.

Moreover, our methodology for splitting the training, validation, and test sets may have been flawed from a graphical network perspective, and this may have been another source of error. In the process of splitting the data, through a process of randomly sampling vertices with instantiated real-to-fake proportions of users (50-50 for the training set, 90-10 for the validation and test sets), many networks may have been inadvertently disconnected. In many cases, there are isolated networks or networks missing vertices — some of the valuable information in these intricate network structures is inevitably lost in this process of splitting the data.

8. Conclusion

In conclusion, while our results were not optimal, we were able to classify between real users and Sybil accounts with consistently high levels of accuracy. As observed from our results, the Gradient Boosting classifier worked best in terms of the 2-step architecture approach to this anomaly detection problem, achieving an accuracy of 88.79% on the test set. However, in terms of precision and AUC, the Adaboost classifier performed best, achieving values of 0.105 and 0.557, respectively. The other models performed similarly. The best precision at K (for $k = 1, 5, 10$) was achieved by the two boosting methods, with the Gradient Boosting classifier performing better overall.

Meanwhile, in terms of the neural network approaches, both the GCN and GAT models performed effectively, attaining accuracy metrics of 94.76% and 95.6% on the test set, respectively. These models were a significant improvement over the 2-step classifiers. While there is room for optimization in terms decreasing variance to improve the performance of our models, a major roadblock towards applying our models lies in the data itself, as well as in the structural similarities between fake and real user networks. [4] Based on the numerous experiments and optimizations conducted involving various advanced methodologies, it seems that the task of detecting fake users by leveraging graphical data alone presents many challenges. Nevertheless, based on the performance metrics of our classification models, the optimizations we implemented allowed our models to marginally outperform the baseline models. A better dataset involving a larger sample of users, non-graphical features, and a reliable oracle (obtained using a better methodology of label classification to identify Sybils), would prove invaluable in the development process to build more robust and effective models for to detecting fake users.

While a graph-based approach alone did not solve the problem effectively, developing technology to detect fake users will have a profound positive impact on society. Such applications of artificial intelligence and machine learning will reduce the spread of misinformation, prevent the escalating phenomenon of astroturfing for political or commercial purposes, and curb the prevalence of fake product reviews and cyber-crime significantly.

8.1. Future Work

As with many approaches that utilize machine learning, our models also may have been prone to overfitting and high variance. Our models may have overfitted to certain dimensions based on the numerous synthetic features in our training set. Given more development time, we could have mitigated such sources of error using common techniques by implementing pipelines to perform regularization, early stopping, and autonomous feature selection (to reduce the number of features used). Given more powerful computational resources, we could also utilize sparse models and Residual Networks which would have been especially helpful in improving the performance metrics of our GCN and GAT models. Finally, to improve the performance of our graph neural network models, we could also investigate Graph Isomorphism Networks (GINs) that leverage the local neighborhood structures of each node. [11]

In addition to educating the users of social network, we believe there is great value in acquiring the computational and development resources to build effective models to tackle this classification problem. As outlined earlier, it is also imperative that future efforts leverage a larger, more descriptive, and reliably labelled datasets to build advanced au-

tonomous architectures that can accurately detect and eradicate Sybil accounts seamlessly and consistently with high levels of precision — without affecting the real users themselves. As such, we hope that the various models and improvements we have engineered have provided a more robust and effective model architecture for future developers and researchers to iterate upon.

9. Contributions & Acknowledgements

Throughout the course of this project, all three members of the team contributed equally in all aspects of the project including project planning & execution, code development, model training & debugging, result analysis, documentation, and paper writing. We would like to thank CS221 instructors and CA team for their help and valuable insights during the course of this project.

Our project benefited and built upon the following research.

1. Generic Anomalous Vertices Detection Utilizing a Link Prediction Algorithm
2. Graph Convolutional Networks
3. Graph Attention Networks

References

[1] Y. Boshmaf, D. Logothetis, G. Siganos, J. Lería, J. Lorenzo, M. Ripeanu, and K. Beznosov. Integro: Leveraging victim prediction for robust fake account detection in osns. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*, 2015.

[2] Q. Cao, M. Sirivianos, X. Yang, and T. Pregueiro. Aiding the detection of fake accounts in large scale social online services. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012*, pages 197–210, 2012.

[3] A. A. Daya, M. A. Salahuddin, N. Limam, and R. Boutaba. A graph-based machine learning approach for bot detection. *CoRR*, abs/1902.08538, 2019.

[4] A. A. A. A. H. A. K. I. U. D. M. G. Faiza Masood, Ghana Ammad and M. Zuair. Spammer detection and fake user identification on social networks. *Institute of Electrical and Electronics Engineers*, 2019.

[5] B. G. U. S. N. S. R. Group. Twitter dataset, 2012.

[6] D. Kagan, M. Fire, and Y. Elovici. Unsupervised anomalous vertices detection utilizing link prediction algorithms. *CoRR*, abs/1610.07525, 2016.

[7] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[8] S. Kudugunta and E. Ferrara. Deep neural networks for bot detection. *CoRR*, abs/1802.04289, 2018.

[9] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018. accepted as poster.

[10] G. Wang, T. Konolige, C. Wilson, X. Wang, H. Zheng, and B. Y. Zhao. You are how you click: Clickstream analysis for sybil detection. In *Proceedings of the 22th USENIX Security Symposium, Washington, DC, USA, August 14-16, 2013*, pages 241–256, 2013.

[11] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

[12] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun. Graph neural networks: A review of methods and applications. *CoRR*, abs/1812.08434, 2018.

10. Appendices

10.1. Graph Features

10.1.1 Link Features

- `Jaccard's Coefficient`
- `Preferential Attachment Score`
- `Adamic-Adar Index`
- `Directed KNN Weights (1-8)`
- `In-Common Friends`
- `Out-Common Friends`
- `Bi-Common Friends`
- `Total Friends`
- `Transitive Friends`
- `Opposite Direction Friends`

10.1.2 Vertex Features

- `In-degree, Out-degree`
- `Bi-degree`
- `Average Neighbor Degree`
- `In-degree density`
- `Out-degree density`
- `Bi-degree density`
- `Strongly-Connected Components(SCC)`
- `Weakly-Connected Components(WCC)`
- `In-degree centrality`
- `Out-degree centrality`
- `Degree Centrality`

10.2. GNN Hyper-parameters

- Number of hidden units: 4, 8, 16, 32.
  Chosen value: 16
- Non-linear function: ReLU, SeLU, Leaky-ReLU.
  Chosen value: Leaky-ReLU
- Learning rate: 1e-1, 1e-2, 5e-2, 1e-3.
  Chosen value: 1e-2

- Dropout Probability: 0.2, 0.5, 0.6.
  Chosen value: 0.5
- Weight Decay: 5e-3, 5e-4, 5e-5.
  Chosen value: 5e-4
- Alpha for Leaky-ReLU: 0.1, 0.2, 0.5.
  Chosen value: 0.2
- GAT Number of Attention heads: 4, 8, 10.
  Chosen value: 8