Network Anomaly Detection

Anish Saha, Vinay Sawal, Vincent Ying
CS 221: Artificial Intelligence - Principles and Techniques

Introduction

We worked on the problem of detecting fake accounts (Sybils) in social network graphs. Fake accounts are created for malicious use and are responsible for a growing number of threats, including fake product reviews, spam on social networks, and astroturfing for political or commercial purposes.

Using a network graph dataset representing the connections of over 5.3 million Twitter users, the goal of this project is to detect anomalous users (spam accounts) as effectively as possible.

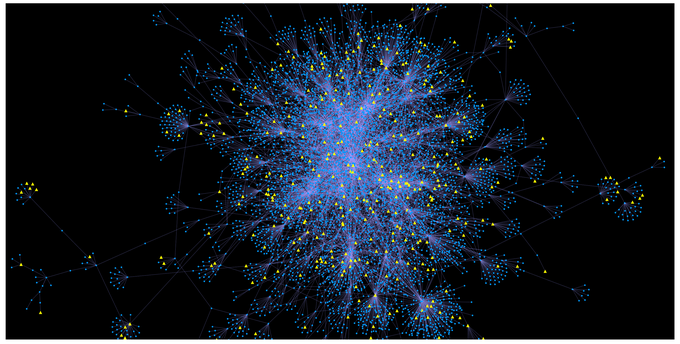
Dataset

Our dataset is a Twitter graph that is comprised of anonymized user ids, directed edges between two users, and real/fake labels.

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

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. We split the dataset using random sampling to extract 10,000 validation set users and 2,000 test set users.

Twitter Social Network Graph



Legend: Blue dots = Real, Yellow triangles = Fake accounts

Feature Engineering

We used the following features for our classification models:

Link Features

- `Jaccard's Coefficient, Preferential Attachment Score, Adamic-Adar index`
- `Directed KNN Weights, Total Friends, Transitive Friends`
- `In-Common Friends, Out-Common Friends`
- `Bi-Common Friends, Opposite Direction Friends`

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`

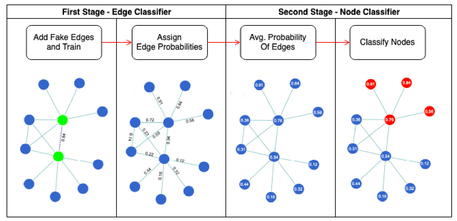
While the baseline versions of our models only used the features `In-degree, Out-degree,` and `Jaccard's Coefficient`, we discovered that using larger feature sets consistently improved training set, validation set, and test set performance metrics significantly. We used the `NetworkX` Python library to both generate the social network graph and to extract all vertex features, corresponding to all users, as well as to extract all link features, corresponding to the edges representing unidirectional connections between users. All of features were used in both the two-step 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 varying mannerisms of fake and real users to generate predictions.

Architectures

Link Prediction Classifier

First Stage Classifiers:

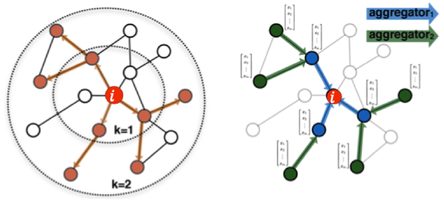
- Logistic Regression
- Random Forest
- Adaboost
- Decision Tree Bagging
- Random Forest Bagging
- Gradient Boosting



Graph Convolutional Network (GCN)

Simple neighborhood aggregation:

$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}\right)$$



Graph Attention Network (GAT)

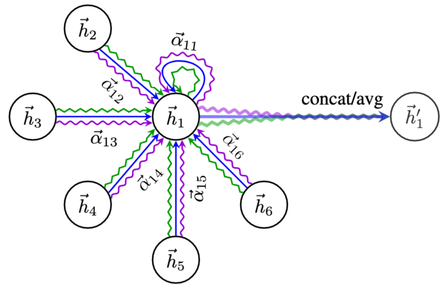
Attention Coefficients:

$$e_{vu} = a(\boldsymbol{W}_k \boldsymbol{h}_u^{k-1}, \boldsymbol{W}_k \boldsymbol{h}_v^{k-1})$$

Normalized Attention Coefficients:

$$\alpha_{vu} = \frac{\exp(e_{vu})}{\sum_{k \in N(v)} \exp(e_{vk})}$$

$$\boldsymbol{h}_v^k = \sigma\left(\sum_{u \in N(v)} \alpha_{vu} \boldsymbol{W}_k \boldsymbol{h}_u^{k-1}\right)$$



Hyperparameter Optimization

We utilized the `RandomizedSearchCV` library from `SciKit-Learn` to efficiently optimize hyperparameter selection for training our classification models. Since the hyperparameter 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 hyperparameter configuration within 5% of the optimum in n iterations for a given model will be:

$$P(X = \text{true}) = 1 - (1 - 0.05)^n$$

where X corresponds to the event that our search has retrieved a config. within 5% optima

We elected to use 90 iterations (cross-validated twice) to achieve a 99% probability of selecting hyperparameters 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 improved performance metrics for all of our models significantly.

Results

The results from all three models on the test set with all link and vertex features included.

Architecture	Classifier	Baseline Accuracy	Tuned Accuracy
2-stage	Logistic Regression	68.95%	88.00%
	Random Forest	87.70%	87.32%
	Adaboost	86.95%	87.81%
	DT Bagging	89.39%	87.41%
	RF Bagging	86.82%	86.14%
	Gradient Boosting	85.27%	87.72%
	GCN	94.38%	TBD
	GAT	94.56%	TBD

Below is a chart representing the resultant test set accuracy metrics from using our different classification models with respect to three different initial feature sets for the two-stage classifier:

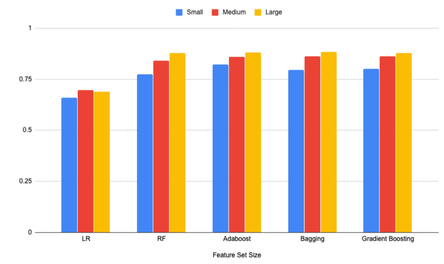


Figure 1: Test Set Accuracy of Preliminary Models

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 could be mislabelled data. We utilized the Twitter social network dataset provided in the baseline architecture. As stated in the research paper from the University of Washington, the Twitter dataset was retrieved and labelled using a scraper that pulled millions of Twitter users and labelled users as anomalous if they were removed when Twitter was scraped 1 year later. However, this is a source of error as users who deleted their account would be mislabelled as fake, while fake users who were not detected 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 such sources of error would likely require some level of collaboration with Twitter to retrieve more reliable data for the purposes of training our models.

Conclusion

In conclusion, while our results were not optimal, we were able to classify between real users and Sybil accounts with consistently high accuracy metrics. As can be observed from our results, the Decision Tree Bagging classifier worked best for the purposes of the 2-step architecture approach to this anomaly detection problem, achieving an accuracy of 89.39% on the test set. Meanwhile, in terms of the neural network approaches, both the GCN and GAT models performed effectively, attaining accuracy metrics of 94.38% and 94.56% on the test set, respectively. While there is room for optimization in terms decreasing variance to improve the true positive rates of our models, a major roadblock towards applying our models lies in the data itself. Using reliable social network data would prove invaluable in the development process to build a more robust model architecture to detect Sybil accounts most effectively.

Future Works

Developing the technology to detect fake users effectively will have a profound positive impact on society. Such applications of artificial intelligence and machine learning will help reduce cyber-crime significantly, preventing the likes of phishing scams from victimizing the younger users or the less tech-savvy users of social networks. In addition to educating the users of social network, we believe there is great value in acquiring the computational and development resources, as well as the reliable data necessary to build advanced autonomous 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 to iterate upon.