
SOCIAL MEDIA BOT DETECTION USING GRAPH FEATURES AND MACHINE LEARNING

VEDANT SAWAL
CS-171 FINAL PROJECT
DEPARTMENT OF COMPUTER SCIENCE
SAN JOSE STATE UNIVERSITY
MAY 19, 2025

Table of Contents

Introduction.....	3
Dataset.....	3
Mathematical Representation.....	4
Experiments.....	4
Logistic Regression.....	4
Random Forest.....	5
Gradient Boosting.....	5
Bagging with Decision Tree Classifier.....	6
Feature Engineering.....	6
Hyperparameter Tuning.....	7
Results.....	8
Baseline Result.....	8
Final Test Results.....	8
Performance with Small Feature Set:.....	9
Performance with Large Feature Set:.....	9
Limitations and Sources of Error.....	10
Conclusion.....	11
References.....	12

Introduction

The current widespread popularity and usage of social media platforms inadvertently bring an increase in malicious activity from fake users, commonly called “bots”. These bot accounts often engage in harmful online behaviors such as spreading spam, spreading misinformation, inflating follower counts, manipulating public opinions, and artificially boosting the visibility of certain types of content. The detection of these bot accounts is a critical challenge in modern online social networks. Since X (formerly known as Twitter) and other social media platforms are the main channels for news, commerce, and public interactions, identifying such malicious users has become increasingly important to ensure the authenticity and safety of online communication.

In this project, I aim to develop a Machine Learning model that can automatically detect fake X users based on the structure of the social network. Unlike traditional classification problems that rely on textual or behavioral data, this project focuses solely on the graph-based structure of the network. It mainly focuses on the connections between users. I implemented a Two-Stage Classifier approach that first evaluates the probability of edges being real or fake, and then uses these predictions to classify users as bots or real accounts based on the number of suspicious relationships.

This report presents the methodology, data preprocessing steps, model implementation, feature engineering, and evaluation metrics used to assess the performance of the machine learning model. The findings suggest that with appropriate feature selection and model tuning, it is possible to effectively detect fake X bot accounts.

Dataset

The dataset that I used is a Twitter graph that is comprised of anonymized user ID's, directed edges between two users, and real/fake labels obtained from Data4GoodLab [2]. This is a labeled dataset with true and fake labels for every account. It is a directed graph with 5,384,160 vertices and 16,011,443 edges.

After completing exploratory data analysis, I found out that the dataset was heavily skewed and contained a few million 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 for this reason, I decided to omit these users. Thus, the final user count was reduced to approximately 75,000 users.

Even with the filtered-down list, such a large dataset presented compute resource challenges during my experiments. Thus, based on the underlying model and the compute resource availability, I split the dataset using random sampling to extract 10,000 validation set users and 2,000 test set users.

Mathematical Representation

A real-world social network can be modeled as a graph $G = (V, E)$ where the set of vertices V represents all of the users and the set of edges E represents the following/follower relationships between the users. The output of this classification task is a binary target variable representing whether or not each user is identified as a fake user. This is a supervised learning classification task, meaning that there exists a definitive source of truth to test our models. For example, let us 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 adjacency list, [(a, b), (a, c)] and Fake list, [a]. Output is represented as a list of predicted labels, [(a, fake), (b, real), (c, real)].

Experiments

For the main model of my project, I utilized a two-stage architecture and vertex/edge features to classify whether a node is real or fake. The first stage 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 the two given vertices. Intuitively, a larger number of highly unlikely edges would indicate an anomalous user or bot account. The second stage uses 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. I used the following classifiers to experiment and improve the performance of the two-stage model.

Logistic Regression

I 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 model in predicting anomalous users. I experimented with different hyperparameter configurations and found that the largest feature set yielded the best results. A list of the hyperparameters that I experimented with is listed below:

Parameter Name and Description	Experimented Values
penalty: Norm of the penalty	['l1', 'l2']
dual: Dual (constrained) or primal (regularized) formulation	[True, False]
max_iter: Maximum number of iterations taken for the solvers to converge	[75, 100, 125, 150]
C: Inverse of regularization strength. Smaller values mean stronger regularization.	[0.5, 1.0, 1.5, 2.0, 2.5]
tol: Tolerance for stopping criteria	[0.00001, 0.00005, 0.0001, 0.0005, 0.001]

Random Forest

I optimized the Random Forest classifier model by exploring feature selection to predict whether or not a given vertex is anomalous. I also experimented with different hyperparameter configurations. Again, the largest feature set seemed to yield the best results for this particular model. A list of the hyperparameters that I experimented with is listed below:

Parameter Name and Description	Experimented Values
bootstrap : Whether bootstrap samples are used when building trees. If <code>False</code> , the whole dataset is used to build each tree	[<code>True</code> , <code>False</code>]
max_depth : Maximum depth of the tree	[10, 20, 40, 60, 80, 100, <code>None</code>]
min_samples_leaf : Minimum number of samples required to be at a leaf node	[1, 2, 4]
max_features : Number of features to consider when looking for the best split	[‘sqrt’, ‘log2’]
min_samples_split : Minimum number of samples required to split an internal node	[2, 5, 10]
n_estimators : Number of trees in the forest	[100, 200, 400, 600, 800, 1000]

Gradient Boosting

I also decided to explore Gradient Boosting models as an alternative methodology for finding anomalous edges and generating edge probability predictions. In my gradient boosting classifier, I used the exponential loss function with a 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. A list of the hyperparameters that I experimented with is listed below:

Parameter Name and Description	Experimented values
subsamples : The fraction of samples to be used for fitting the individual base learners	[0.7, 0.8, 0.9, 1]
max_depth : Maximum depth of the tree	[10, 20, 40, 60, 80, 100, <code>None</code>]
min_samples_leaf : Minimum number of samples required to be at a leaf node	[1, 2, 4]
max_features : Number of features to consider when looking for the best split	[‘sqrt’, ‘auto ’]

<code>min_samples_split</code> : Minimum number of samples required to split an internal node	[2, 5, 10]
<code>n_estimators</code> : Number of trees in the forest	[50, 100, 200, 400, 600, 800, 1000]

Bagging with Decision Tree Classifier

I created an optimized tree-based approach using a 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. A list of the hyperparameters that I experimented with is listed below:

Parameter Name and Description	Experimented values
<code>estimator</code> : The base estimator to fit on random subsets of the dataset	<code>DecisionTreeClassifier</code>
<code>max_depth</code> : Whether bootstrap samples are used when building trees. If <code>False</code> , the whole dataset is used to build each tree	[10, 20, 40, 60, 80, 100, <code>None</code>]
<code>n_estimators</code> : The number of base estimators in the ensemble	[50, 100, 200, 400, 600, 800, 1000]
<code>max_samples</code> : The number of samples to draw from X to train each base estimator	[0.01, 0.05, 0.1, 0.2, 0.4, 0.6, 0.8, 1]
<code>bootstrap</code> : Whether samples are drawn with replacement. If <code>False</code> , sampling without replacement is performed.	[<code>True</code> , <code>False</code>]
<code>oob_score</code> : Whether to use out-of-bag samples to estimate the generalization error	[<code>True</code> , <code>False</code>]

Feature Engineering

For this project, I extracted graph-based features to support the Two-Stage Classifier model. These features were divided into two main categories: edge features (which describe the relationships between pairs of users) and vertex features (which describe individual users).

Edge 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

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

To evaluate the impact of feature complexity, I created two feature sets:

- Small Feature Set: Includes only the Jaccard's Coefficient and the In/Out degree. This served as the baseline configuration.
- Large Feature Set: Builds upon the small set by incorporating all other edge and vertex features listed above. It captures richer information about the user and their network neighborhood.

Feature extraction was performed using the NetworkX Python library, which allowed me to compute both standard network metrics and more advanced structural properties. This enhanced the model's ability to detect anomalies by capturing how users are embedded in the graph.

Through experimentation, I found that the large feature set significantly outperformed the small one in terms of both validation and test accuracy. These features were used in both stages of the classifier: the first stage to predict the probability of an edge being valid, and the second stage to classify users based on the number of unlikely connections.

Hyperparameter Tuning

I used the **RandomizedSearchCV** library from SciKit-Learn to efficiently optimize hyperparameter selection for training the classification models. 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 for the given model in n iterations is:

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

Where the random variable X corresponds to the event that at least one of the configurations found in n iterations is within 5% of the optimum hyperparameter configuration. I elected to use 90 iterations (cross-validated twice) to achieve a 99% probability of selecting hyperparameters within 5% of the optimal configuration.

Since each classification model uses different default parameters, I 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.

Results

This section presents the results obtained from training and evaluating various classifiers within the Two-Stage Classifier framework. The experiments were conducted using two different feature sets (small and large) to evaluate the impact of feature complexity on model performance. The performance of each model was assessed on both the validation and test sets using accuracy as the primary metric.

Baseline Result

To establish a performance baseline, I initially trained two models, Logistic Regression and Random Forest, using only the Jaccard's Coefficient as the sole feature. This minimal feature set served as a benchmark to measure the effectiveness of more complex feature sets later in the project.

These models were trained on a filtered version of the dataset, which excluded users with no meaningful graph connections (e.g., no followers or followings). The results are shown in the table below:

Classifier	Accuracy
Logistic Regression	66.00%
Random Forest	77.36%

Figure 1: Baseline Results

The Random Forest classifier significantly outperformed Logistic Regression in this setting, demonstrating the advantage of using an ensemble method even with limited input features.

Final Test Results

After establishing the baseline, I trained additional models using two progressively richer feature sets: a small set and a large set. The small feature set included basic structural features such

as in-degree, out-degree, and Jaccard's Coefficient. The large feature set added a wide range of edge and vertex features, as described in the Feature Engineering section.

The following tables summarize model performance for both feature sets:

Performance 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	81.80%	79.50%
Gradient Boosting	83.85%	80.09%

Figure 2: Accuracy of different models with a small feature set size.

Performance 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	88.90%	88.36%
Gradient Boosting	90.74%	87.77%

Figure 3: Accuracy of different models with a large feature set size.

The results indicate that model accuracy improves as more features are incorporated. All classifiers saw notable performance gains when moving from the small to the large feature set, with the ensemble methods (Random Forest, Bagging, Adaboost, and Gradient Boosting) benefiting the most.

In particular, the Gradient Boosting classifier consistently performed among the best, achieving the highest accuracy on both the validation (90.74%) and test sets (87.77%) with the large

feature set. While other ensemble methods like Adaboost and Bagging performed comparably, Gradient Boosting offered a slightly better balance of generalization and performance.

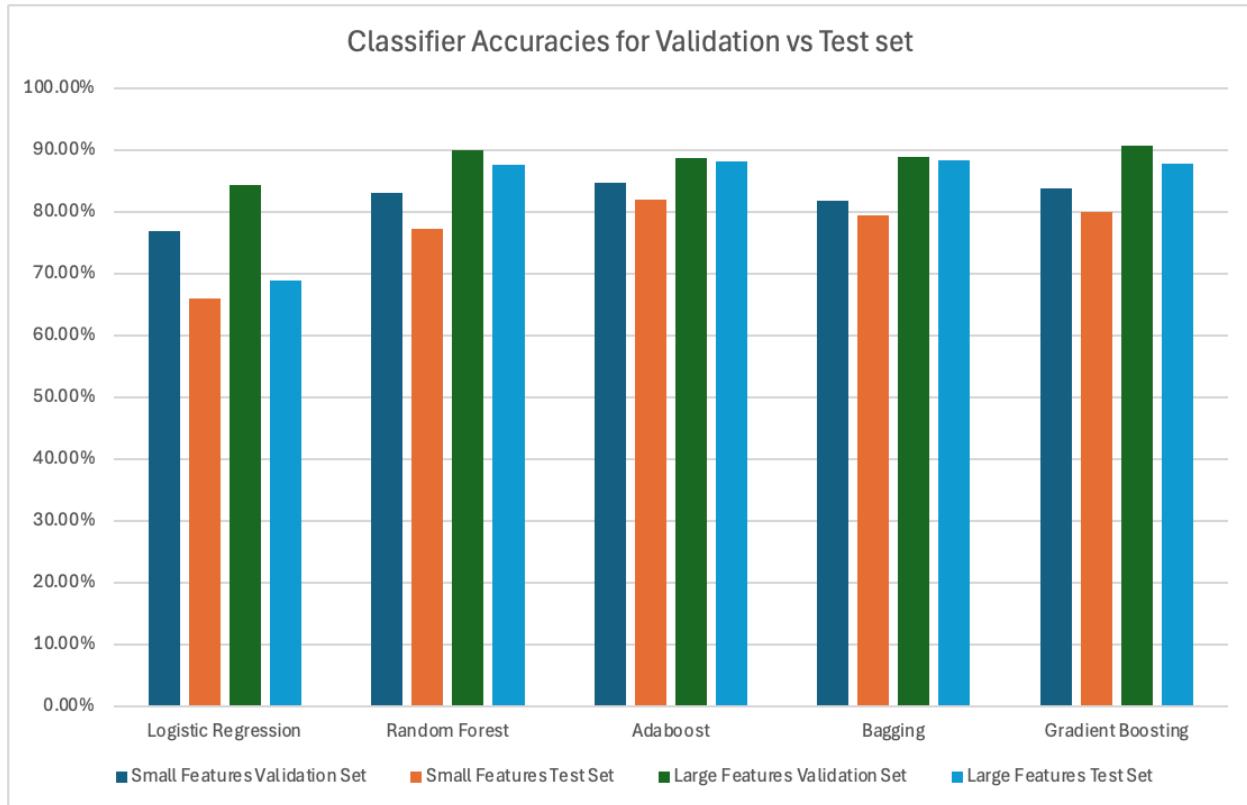


Figure 4: Final accuracy results for Two-stage Classifier

Additionally, although hyperparameter tuning only slightly improved accuracy compared to default configurations, it played an important role in stabilizing model performance across both validation and test sets.

These findings reinforce the importance of both robust feature engineering and model selection when tackling network-based classification problems such as bot detection.

Limitations and Sources of Error

While the models trained in this project demonstrated strong accuracy, they can be further improved by overcoming the limitations stated below. These limitations primarily stem from issues related to data labeling and the inherent complexity of distinguishing real users from bots in large social networks.

1. **Labeling Inaccuracies in the Dataset:** A significant source of error is the potential mislabeling in the dataset. The Twitter social network data used in this project was

provided by Data4GoodLab [2] and labeled based on a heuristic approach where users were considered fake (anomalous) if their accounts were removed from Twitter in the year following initial data collection (2014). However, this labeling strategy introduces ambiguity, such as, legitimate users who voluntarily deleted their accounts would be misclassified as bots; Fake accounts that remained undetected by Twitter would be incorrectly labeled as real. Although the dataset was preprocessed to remove users with no connections (i.e., zero followers or followings), which likely filtered out many inactive or irrelevant accounts, this step alone cannot resolve the deeper issue of unreliable ground truth labels. Addressing this limitation would ideally require access to a more rigorously validated dataset, potentially obtained through collaboration with Twitter or other social media platforms.

2. Structural Similarity Between Real and Fake Users: Another challenge lies in the structure of the social network itself. Many bot accounts are part of sophisticated botnets that are intentionally designed to mimic the patterns of real user interactions. These fake networks can exhibit similar connection densities, follower distributions, and clustering behaviors as genuine user networks. As a result, even models trained on rich graph-based features may struggle to distinguish between authentic and malicious users.

This structural overlap limits the effectiveness of models that rely exclusively on graph features. To improve detection, future work should consider incorporating additional user-level information, such as:

- Natural Language Processing (NLP) features from user-generated content (e.g., tweet text, bios).
- Temporal patterns, such as posting frequency or follower growth rates.
- Behavioral metrics, including retweet ratios or mentions per post.

Combining these non-graph-based attributes with the existing network-based approach could provide a more robust signal for identifying bot activity.

Conclusion

This project explored the task of detecting fake users on Twitter using a Two-Stage Classifier model based on structural features derived from the social network graph. The model demonstrated strong classification performance, with the Gradient Boosting classifier achieving the best results with an accuracy of 88.79% on the test set. These findings validate the effectiveness of ensemble methods in the context of graph-based anomaly detection.

Despite these promising results, several challenges remain. One of the most significant limitations lies in the quality of the dataset, particularly the reliability of the ground truth labels. The heuristic labeling method used (classifying accounts as fake if they disappeared from the platform after one year) likely introduced noise and misclassifications into the dataset. Additionally, the structural similarity between real and fake user networks further complicates the classification task, as bot networks can often mimic genuine interaction patterns.

The experiments conducted throughout this project highlight both the potential and limitations of using graph-only features for bot detection. While the model effectively leveraged network structure to identify anomalies, it also revealed that structural data alone may not be sufficient for achieving extremely high accuracy in more complex, real-world settings.

Future progress in this domain will likely depend on access to higher-quality datasets, enriched with both graph-based and non-graph features, such as content, behavior, and temporal dynamics. Integrating such data would enable the development of more accurate and generalizable models.

Ultimately, the ability to detect and remove fake accounts at scale is crucial for preserving the integrity of online platforms. Advances in machine learning and artificial intelligence, when supported by high-quality data, can play a critical role in combating misinformation, reducing digital fraud, and ensuring healthier online ecosystems.

References

- [1] [Generic Anomalous Vertices Detection Utilizing a Link Prediction Algorithm \(GitHub link\)](#)
- [2] [X \(formerly known as Twitter\) Dataset](#)