Frontiers in Computing and Intelligent Systems
ISSN: 2832-6024 | Vol. 8, No. 1, 2024

Ethereum Eclipse Attack Detection based on BG-XGBoost

Chao Huang *, Weiping Yang

School of Cybersecurity, Chengdu University of Information Technology, Chengdu, Sichuan, 610200, China
* **Corresponding author**: Chao Huang (Email: huangchao251@gmail.com)

Abstract: Abstract: Eclipse attacks pose a serious threat to blockchain networks. Research has proven that Ethereum is more vulnerable to the Eclipse attack than the Bitcoin peer-to-peer network. Therefore, related research on Eclipse attacks on Ethereum is of great value. This paper proposes an improved XGBoost algorithm based on Bagging. It simulates a variety of random situations through the Bagging method, introduces randomness, reduces the risk of high errors, reduces the variance of the XGBoost model output, and improves the generalization ability of the model. It further enhances the model performance on binary classification problems and achieves efficient identification of Eclipse attack traffic and normal traffic.

Keywords: Blockchain; Network Security; Eclipse Attack; Ethereum; Traffic Detection.

1. Introduction

With the rapid development of blockchain technology, Ethereum, as one of the most representative public blockchains, has demonstrated strong application potential in areas such as digital asset transactions and smart contracts. Ethereum is a decentralized open source public blockchain platform with smart contract functions and one of the largest open source blockchain platforms [1]. Similar to traditional blockchain platforms, Ethereum utilizes a decentralized network of nodes to verify and record transactions. Ethereum has several important characteristics, including openness, immutability, transparency, decentralization, integrity, and programmability.

With the rapid development of the blockchain industry, the security issues of blockchain have attracted more and more attention. At present, the main research directions on blockchain security are mostly focused on the technical aspects of blockchain, such as consensus algorithm security, smart contract security, privacy protection, side chain and cross-chain security, decentralized application (DApps) security direction, etc. [2]. As one of the most valuable blockchain platforms, Ethereum still has huge room for development in the future because of its openness and innovation. Therefore, research on its technology, security, performance, economic model and application has extensive value. Despite growing research into the security of the Ethereum consensus algorithm and scripting language, the properties and security of the Ethereum peer-to-peer network remain largely unexplored. And research has shown that the security properties of a proof-of-work blockchain depend on the security of its underlying peer-to-peer network. Although it is a relatively mature blockchain technology, Ethereum developers have proposed a variety of security mechanisms to deal with the increasingly severe blockchain network security environment. However, relevant research has proven that compared to other traditional blockchain systems such as Bitcoin, Ethereum is still subject to a variety of security threats targeting different architectural levels of the system.

Eclipse attacks are one of the most harmful blockchain network security threats. Heilman et al. highlighted this risk by demonstrating the first Eclipse attack on the Bitcoin peer-to-peer network [3]. Research has proven that Ethereum is more vulnerable to Eclipse attacks than Bitcoin peer-to-peer networks [4]. Therefore, as one of the most promising public blockchain projects, related research on Ethereum eclipse attacks has more important value. This article systematically analyzes the operating mechanism and potential security risks of the Ethereum network through in-depth study of the principles and security technologies of the Ethereum peer-to-peer network. This article focuses on the Eclipse attack against Ethereum. By exploring new detection algorithms and defense mechanisms, it can provide new ideas and methods for research in the field of blockchain network security. At the same time, with the help of Ethereum as a research environment, the characteristics of the blockchain system and its existing security vulnerabilities are analyzed. Researching and implementing the detection of Eclipse attacks on Ethereum has important practical significance and research value.

2. Eclipse Attack on Blockchain

2.1. Attack Principle

The eclipse attack is a typical blockchain network layer attack that adds a large number of malicious nodes to the neighbor node set of normal nodes by invading the routing table of blockchain network nodes. Afterwards, before the attacker restarts the victim node in the blockchain, they create malicious request links and fill up the victim node's routing table. This forces the victim node to establish a routing connection with the attacker after restarting. Furthermore, the attacker continues to initiate inbound connection requests to the victim node, with the purpose of occupying the victim node's communication channel and controlling the flow of information. As a result, affected miner nodes have to process and pass on erroneous or malicious blockchain ledger data sent by the attacker. Through this step-by-step strategy, eclipse attacks can gradually control the blockchain network layer channels and data flows of more nodes. Such attacks ultimately destroy the routing structure of the attacked subnet, resulting in a waste of computing resources and allowing the attacker to gain an advantage in computing power. When a node is attacked by Eclipse, most of the node's external

connections will be controlled by malicious nodes, allowing the attacker to carry out further attacks [5]. The principle of Ethereum Eclipse attack is shown in the figure 1.

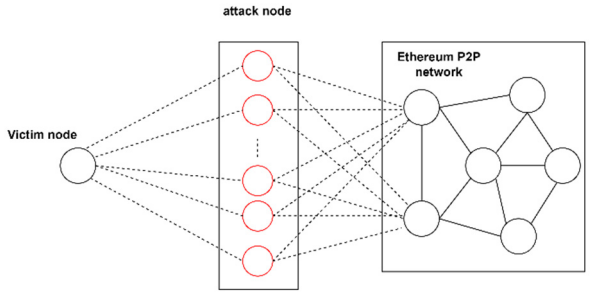

Figure 1. Ethereum Eclipse attack principle

2.2. Safety Hazards

Heilman et al. highlighted this potential risk by defining and demonstrating the first eclipse attack on the Bitcoin peer-to-peer network [3]. In an attack, the attacker has complete control over their target's access to other network nodes, allowing the attacker to prevent the target from viewing the ledger and even further exploit the target's computing resources to conduct more complex attacks. The Eclipse attack can serve as a precursor to other attacks on blockchain platforms. This work demonstrates that eclipse attacks can be used to impact the mining capabilities of an attack target, thereby attacking the underlying blockchain consensus algorithm. Atzei et al. [6] further pointed out that Ethereum smart contracts may be attacked in cases where nodes receive inconsistent ledger information. At the same time, Eclipse attacks can be exploited as part of a countermeasure strategy to inject inconsistencies. Later, Anderson et al. [7] demonstrated that the same attack also affects the Ethereum peer-to-peer network, destabilizing the platform..

2.3. Defense Strategy

Heilman et al. [3] first proposed the Eclipse attack method in the Bitcoin P2P network and highlighted the potential risks associated with this attack. Attackers can exploit vulnerabilities in the blockchain consensus mechanism to control the routing information of attacked nodes, restrict the communication between these nodes and other nodes, and prevent the attacked nodes from obtaining correct ledger data. Furthermore, sophisticated attacks can be launched against the computing resources of the compromised node, which affects the victim's mining capabilities and the blockchain's uplink results, thus weakening the stability of the computing power distribution. According to the report, when the ledger information received by Ethereum nodes is inconsistent, attackers can also exploit vulnerabilities in Ethereum smart contracts to launch Eclipse attacks. In addition, Yuval Marcus et al. [8] experimentally proved that Eclipse attacks can directly affect the security of each Ethereum node and destroy the stability of the blockchain system. The Eclipse attack detection method in blockchain has been studied in previous research; however, this method has some disadvantages. Currently, there are two types of Eclipse attack detection methods. The first is Eclipse detection based on routing topology awareness. Eclipse attackers can send continuous connection requests to the attacked target to occupy the node's routing table, because redundant storage exists in the node flow table of wireless communication and mobile computing networks, Chord and Kademlia structures [9][10]. In this case, the detector can analyze and model the routing structure according to various parameters, such as the topology of the blockchain network layer and the routing table forwarding status of key nodes, so that detecting changes in routing status can help identify Eclipse occurrence of attacks [11]. Generally, such detection methods are highly reliable in Eclipse attack detection results and have high reference value for mining structural vulnerabilities in the blockchain network layer [12]. However, its detection mode suffers from a complex problem, resulting in weak generalization and expansion capabilities of the model. Therefore, gaps can be discovered in real-time detection of the model. Furthermore, it cannot adapt to the dynamically changing blockchain network layer traffic environment [13] because of its weak awareness in dynamic non-fixed Eclipse attack paths. The second type of Eclipse attack detection method is based on link traffic status analysis. Eclipse attackers must send large amounts of malicious routing traffic to target nodes and subnets to achieve route masking and overrides because Eclipse attacks are designed to disrupt the routing structure. Attack traffic behavior has specific distribution rules [14]. In this case, the proposed method can mine the core characteristic indicators of Eclipse attacks by capturing the traffic in the blockchain network layer and analyzing the real-time traffic status. Furthermore, the proposed method can build statistical or machine learning models to determine the occurrence of Eclipse attacks [4]. Methods with strong real-time detection and good generalizability can monitor and identify Eclipse attack traffic in real time based on dynamic changes in traffic characteristics. However, when an attacker masks its route building requests by disguising traffic routed into the blockchain subnet, it is difficult for existing detection techniques to detect such dynamic multi-path Eclipse attacks because the attacker may not directly attack the area. The sub-network structure of the blockchain. Therefore, these detection methods are still insufficient in dynamically sensing attack traffic characteristics. In addition, since the behavior of Eclipse attack traffic is very similar to normal traffic, and the differences in core characteristics are minimal, these methods still have a lot of room for improvement in the accuracy of attack detection. At the same time, there is a significant gap between existing detection capabilities and the security requirements of the blockchain network environment. Xu et al. [4] proposed a detection model based on random forest classification for blockchain traffic. The detection accuracy and recall rate were 72% and 93% respectively, but they did not propose an actual defense system and countermeasures. Dai et al. [15] proposed a model that combines CNN and Bi-LSTM technology with a multi-head attention mechanism to improve classification and detection performance by perceiving different levels of eclipse attack traffic characteristics, but the computational complexity of the model is too high. , it is difficult to cope with real-time attack traffic detection. At present, there are few studies on the detection of blockchain Eclipse attack traffic. In the future, research can be conducted on detection algorithm optimization and mechanism innovation for a variety of different blockchain applications.

3. Attack Traffic Detection Method Based on BG-XGBoost

3.1. Framework and Algorithm

Extreme Gradient Boosting Algorithm (XGBoost) is an efficient and powerful machine learning method. It is an

optimized implementation of the Boosting algorithm in ensemble learning, especially when dealing with classification problems. The core idea is to combine several models with weak performance (such as decision trees) and generate a powerful integrated model through iterative optimization.

XGBoost introduces regularization terms during the training process, which not only controls the complexity of the model, but also prevents overfitting, thus improving the generalization ability while ensuring the accuracy of the model. In addition, XGBoost supports a variety of objective optimization functions and evaluation criteria, making it widely applicable to various regression, classification and ranking problems.

When faced with large-scale data sets, XGBoost demonstrates efficient computing speed and good scalability, thanks to its optimized data structure and computing algorithm. XGBoost can automatically utilize multi-threads for parallel computing and supports distributed computing frameworks such as Hadoop and Spark.

When processing sparse data, XGBoost is optimized through sparsity-aware algorithms, allowing the model to automatically handle missing data and use the sparse characteristics of data to reduce computational complexity. These characteristics make XGBoost widely used in financial risk control, network security, bioinformatics and other fields.

XGBoost (Extreme Gradient Boosting) is an ensemble learning algorithm based on decision trees, optimized under the framework of gradient boosting. It not only has significant advantages in forecast accuracy, but also excels in efficiency and flexibility.

3.1.1. Bagging-XGBoost Algorithm

In general, XGBoost is an algorithm with strong learning ability and high accuracy, and has certain generalization ability and anti-overfitting ability. However, the XGBoost algorithm improves accuracy through parameter constraints and serial iteration, making the structure of the entire model deeper, and there is still a risk of overfitting, and there is a risk of higher errors due to the lack of randomness when changing data sources.

Therefore, the Bagging (Bootstrap aggregating) algorithm can be introduced, which is also an integrated learning idea, to improve the XGBoost algorithm to further enhance its generalization ability and reduce the output variance. The essence of bagging is to randomly sample from the initial sample data to obtain a sample subset. Then each base model is trained based on a subset of samples, and finally the average of each model's results is taken as the output. Bagging can effectively reduce variance and overfitting. It simulates various random situations, introducing randomness and reducing the risk of high errors.

Therefore, this article proposes an improved XGBoost algorithm based on Bagging, named Bagging-XGBoost (BG-XGBoost). By introducing regularization and randomness, it limits the model complexity of BG-XG and reduces the variance of the output results. Through serial iteration and parallel sampling, the learning ability of BG-XG is improved, the deviation of the output results is reduced, and the generalization ability is enhanced. The structure of the Bagging-XGBoost algorithm is shown in Figure 2.

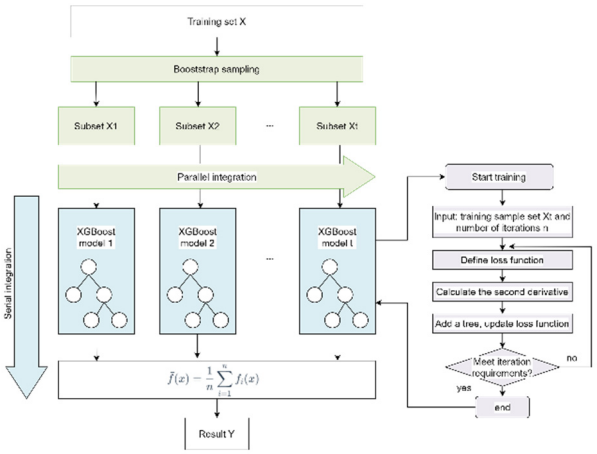

Figure 2. Bagging-XGBoost algorithm structure

3.2. Attack Traffic Detection Model

This paper proposes the BG-XGBoost model, which is a hybrid ensemble learning model that combines bagging and XGBoost algorithms. Based on the feature parameters extracted in Chapter 3, an Eclipse attack traffic detection data set is constructed. 10k cross-validation, Bayesian hyperparameter optimization and parallel processing are used to improve the accuracy of network intrusion detection, thereby improving the performance of the network security system. Use python language to build and execute the model. In practice, combining bagging and XGBoost means training multiple XGBoost models on different self-sampled data and then aggregating the predictions of all models. By combining the sample randomness of the Bagging method with the efficient learning ability of the XGBoost algorithm, this model can not only effectively reduce the bias and variance of the model, but also provide accurate predictions for the complex Ethereum Eclipse attack mode. In addition, Bayesian optimization is used to intelligently adjust model hyperparameters, further enhancing the model's prediction accuracy and computational efficiency. With the help of parallel processing technology, the training and prediction process of the model can be accelerated, meeting the needs of large-scale data processing.

3.2.1. Data Collection

Based on the research of Xu et al., this paper designs an Ethereum Eclipse attack experiment [4]. Dai [15] et al. The Wireshark application captures samples of real network traffic such as UDP, TCP, and ICMP. Additionally, Wireshark's Ethereum devp2p protocol parser plugin was installed to inspect Ethereum traffic packets. Ethereum 2.0 and Hyperledger Fabric 1.4 are implemented to emulate the integrated network traffic of the Ethereum network layer. Ubuntu 22.04 LTS with Docker Networking for generating IP segments is provided to simulate and run the Ethereum Virtual Machine, Geth program and Hyperledger Fabric1.4. Eclipse attacks are launched via two methods. First, Yersinia is a secure network tool that allows practitioners to launch layer 2 attacks by crawling neighbor nodes in the network. Then, the attack script generated by the Scapy library in Python 3.9 is executed, and eclipse attack packets such as Ping, Pong, Findnode, and Neighbors are repeatedly broadcast, forcing the target node to create an incorrect routing table.

Data are collected under three conditions depending on the attack status. The first stage is the normal operating state, which means that the node has been connected to the network for at least twenty-four hours. Suppose that node's database

and END table might store some information about its neighbor nodes. Data collected from this state is considered "normal network traffic" or "majority samples". The second stage is the offensive state. At this stage, the eclipse attack was launched while the Ethereum system was running. Data collected from this state is considered "Eclipse Attack Network Traffic" or "Minority Samples". Finally, reconnect the state. After an Eclipse attack is launched, each target node will reboot and reconnect to the Ethereum network. Technically, the END table is empty when a node is in this state. Therefore, an attacker can easily control a node by launching inbound packets to the target node. Data collected from this state is also considered "Eclipse Attack Network Traffic" and "Minority Samples".

3.2.2. Model Building

This article follows the following steps to build the model:

(1) Data preprocessing and cleaning: First, the data set is preprocessed and cleaned to correct missing values, outliers, or inconsistencies, which may negatively affect the quality of the model.

(2) Data set preparation for Bayesian hyperparameter optimization: A smaller data set was constructed from the complete data set for Bayesian optimization of the hyperparameters of the XGBoost base model. This step is to obtain optimal hyperparameters when training and testing the model on the full data set.

(3) Application of parallel processing: Ensure the use of parallel processing during Bayesian hyperparameter optimization and model training and testing to improve computational efficiency.

(4) Definition of fold number of cross-validation: Define the fold number of cross-validation. In the case of this article, 10-fold cross-validation is used, that is, k=10.

(5) Model initialization: Use XGBoost as the base algorithm and Bagging as the wrapper to initialize the model.

(6) Cross-validation loop: Loop through each fold of the cross-validation and perform the following steps:

·Divide the training data into k folds;

·For each fold, use Bagging to create multiple instances of the XGBoost model, each using

·Use different subsets of training data;

·Corresponding to each Bagged XGBoost model, use the corresponding training data subset for training;

·Use the trained XGBoost model to predict the test data.

(7) Aggregation of prediction results: Aggregate the prediction results of all models through the average method to obtain the final prediction for each instance in the test data.

(8) Performance evaluation: Calculate evaluation indicators such as Cohen's Kappa, F1 score, precision, recall rate, and ROC AUC to evaluate the performance of the model.

3.2.3. Bayesian Optimization

In machine learning, hyperparameters are model parameters that are set before the training process rather than learned from the training data, such as learning rate, number of hidden layers, and regularization strength. Bayesian optimization is a technique used for hyperparameter tuning in machine learning, which aims to find an optimal set of hyperparameter combinations for the model. Bayesian optimization models the relationship between hyperparameters and model performance, often using a Gaussian process to guide the search process to find optimal hyperparameters that maximize performance metrics (such as accuracy, F1 score, or AUC) while minimizing The number of model evaluations.

In order to speed up the hyperparameter tuning process, this paper selects a smaller data subset from the main data set for Bayesian optimization. Hyperparameter optimization using methods such as grid search or random search can be computationally expensive, especially when the dataset is large or the number of hyperparameters that need to be tuned is high. Using a smaller data set for Bayesian optimization can significantly reduce the time required to find optimal hyperparameters. By reducing the size of the data set, the time required for model training and evaluation is shortened, allowing optimization algorithms to explore the hyperparameter space faster. Once the optimal hyperparameters are found using a smaller dataset, these hyperparameters can be used to train the model on the full dataset.

In the model of this article, optimization is implemented through the BayesianOptimization function in the bayes_opt package. This function accepts the objective function xgb_optimization as a parameter, which represents the evaluation of the XGBoost model under different hyperparameters. The xgb_optimization function accepts the hyperparameters to be optimized and returns the average F1 score across all folds. Bayesian optimization is performed within the range of hyperparameters defined in the bounds dictionary. The BayesianOptimization function finds the optimal hyperparameters of the XGBoost model through the process of iterative exploration and utilization. In the first step, several initial points are randomly selected from the hyperparameter space to explore different regions. The function then uses a Bayesian model to estimate the probability that different hyperparameters lead to the best outcome, based on previous results. The function selects the hyperparameters that are expected to improve the most based on the Bayesian model and uses these hyperparameters to evaluate the objective function. This process of sampling, modeling, and evaluation loops until the maximum number of iterations is reached.

3.2.4. Parallel Processing

During the model construction process, this article realizes parallel processing of Bayesian optimization, XGBoost and Bagging models by setting the n_jobs parameter. Setting n_jobs to -1 enables parallel processing by allowing the model to utilize all available CPU cores. When defining the XGBoost model, by n_jobs=-1, the model can use all available CPU cores to accelerate the training process. Similarly, when defining the Bagging model, n_jobs=-1 is also used to implement parallel processing during model training. In addition, by using the multiprocessing module, the number of parallel jobs is set to the number of available CPU cores, ensuring that the optimization process can be carried out efficiently and quickly using available computing resources.

Parallel processing is also used in Bayesian optimization. The BayesianOptimization function accepts an optional verbose parameter, with a default value of 1. Setting this parameter to 0 enables parallel processing. By default, BayesianOptimization runs in single-threaded mode, but it can be easily parallelized by setting the n_jobs parameter to the desired number of parallel jobs.

Through the above method, this paper not only achieves parallel processing in the training phase of the model, but also effectively utilizes parallel computing resources in the hyperparameter optimization phase, greatly improving the

efficiency of model training and parameter optimization. This parallel processing strategy ensures that When processing large-scale data sets, the computational load of the model building process can be distributed to multiple processors, thereby shortening the overall time of model training and optimization and improving the efficiency of model development.

3.2.5. Combination of Bagging and XGBoost

In the research of this article, Bagging and XGBoost are effectively combined by using the BaggingClassifier class of the scikit-learn library. BaggingClassifier is a meta-estimator that separately fits base classifiers on random subsets of the original data set and aggregates their respective prediction results. In this scenario, the basic classifier is XGBClassifier, which is an implementation of the XGBoost algorithm.

BaggingClassifier requires an estimator as a parameter. In this article, the parameter is the XGBClassifier that obtains the optimal hyperparameters through Bayesian optimization. The n_estimators parameter is used to specify the number of base classifiers used, while the n_jobs parameter allows for parallel processing.

Therefore, by using BaggingClassifier to fit a series of XGBClassifier models with optimal hyperparameters and using parallel training, this code realizes the combination of Bagging and XGBoost, thus making full use of available computing resources. This combination not only improves the robustness of the model, but also helps reduce the possible overfitting risk of the XGBoost model by introducing Bagging's random sampling strategy, thereby improving the model's generalization ability on complex data sets. In addition, through the application of parallel processing, the efficiency of model training is significantly improved, making it possible to quickly process large-scale data sets.

3.3. Model Performance Evaluation

3.3.1. Model Evaluation Index

When evaluating the performance of a machine learning model, it is crucial to choose appropriate evaluation metrics. These metrics not only reflect the performance of the model on a specific task, but also help us understand the strengths and weaknesses of the model. For classification problems, especially when dealing with imbalanced data sets or performing refined class distinctions, a single accuracy metric often cannot comprehensively assess model performance. Therefore, this section will introduce several key evaluation indicators in detail, including F1 score, precision rate, recall rate, accuracy rate and ROC AUC. These indicators measure the performance of the model from different perspectives and help us comprehensively understand the model's performance in prediction tasks. By comprehensively considering these indicators, this article can more accurately evaluate and optimize the classification model to achieve better prediction results.

For a two-class classification problem, the data set is divided into two categories: true and false. Therefore, there are a total of the following four classification results in the two-class classification problem:

FN: False Negative, classified as false, actually true

FP: False Positive, classified as true, actually false

TN: True Negative, classified as false, actually false

TP: True Positive, classified as true, actually true

F1 score is the weighted average of precision and recall, with the best value being 1 and the worst value being 0. It achieves a balance between precision and recall by calculating the harmonic mean of the two. Mathematically, the F1 score can be expressed as:

$$F1 \text{ score} = \frac{2 \text{ x Precision x Recall}}{Precision + Recall} \quad (1)$$

Precision is the ratio of true examples (i.e., correct positive predictions) to the total number of all positive predictions. It measures the accuracy of positive predictions, with a high accuracy score meaning fewer false positive predictions. Mathematically, the accuracy can be expressed as:

$$Precision = \frac{TP}{TP + FP} \quad (2)$$

Recall is the ratio of true examples to the total number of actual positive examples. It measures the completeness of positive predictions, with a high recall score meaning that the vast majority of actual positive examples are captured by positive predictions. Mathematically, recall can be expressed as:

$$Recall = \frac{TP}{TP + FN} \quad (3)$$

Accuracy is the proportion of correctly predicted instances (including true examples and true counterexamples) to the total number of all instances. It measures the overall level of accuracy of the model's predictions, with the best value being 1 and the worst value being 0. A high accuracy score means the model got a higher proportion of all predictions correct. Mathematically, the accuracy can be expressed as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (4)$$

ROC AUC is a performance measure for binary classification problems. It is the area under the ROC curve, which is a graphical representation of model performance that plots the true rate (recall) versus the false positive rate. AUC measures the separability of two classes, with a value of 1 representing perfect separation and a value of 0.5 representing random guessing.

3.3.2. Model Comparison

The ten-fold cross-validation method is used to compare the performance of algorithms such as BG-XGboost, random forest, naive Bayes, SVM, and KNN. The data set used is the sample data set constructed in Section 3.3. The algorithm model evaluation results are shown in the table. The results show that the BG-XGBoost algorithm model is the model with the best classification effect under the evaluation of F1 score, Precision, Recall, Accuracy, ROC AUC and other indicators.

Table 1. Model comparison

algorithm	F1 score	Precision	Recall	Accuracy
Naive Bayes	0.844	0.907	0.782	0.786
SVM	0.875	0.878	0.872	0.871
KNN	0.912	0.923	0.901	0.904
Random Forest	0.927	0.902	0.951	0.946
BG-XGBoost	0.981	0.974	0.989	0.983

Naive Bayes shows relatively high precision (0.907), but its recall (0.782) is low, which results in an F1 score of 0.844. Although the accuracy is 0.786, which is slightly insufficient, its ROC AUC score reaches 0.978, indicating that it has excellent positive and negative class discrimination capabilities. SVM performed stably on all evaluation metrics,

with an F1 score of 0.875, precision and recall of 0.878 and 0.872 respectively, and an accuracy of 0.871. The ROC AUC score is 0.931, showing that SVM has strong classification ability and good discrimination. The KNN algorithm highlights its high efficiency with an F1 score of 0.912, a precision of 0.923, a recall rate of 0.901, and an accuracy rate of 0.904. The ROC AUC score as high as 0.971, indicating that KNN has high accuracy and significant class discrimination ability on this data set. Random Forest stands out for its high recall (0.951), with an F1 score of 0.927, slightly lower precision of 0.902, and accuracy of 0.946. However, its ROC AUC score is 0.925, which is lower than other models, which may mean that its ability to distinguish positive and negative classes is slightly insufficient in some scenarios. BG-XGBoost showed the best performance in all indicators, with an F1 score of 0.981, a precision of 0.974, a recall rate of 0.989, an accuracy rate of 0.983, and a ROC AUC score of 0.982. This shows that the BG-XGBoost model has reached extremely high standards in terms of ensuring accuracy, improving recall rate, and distinguishing positive and negative categories.

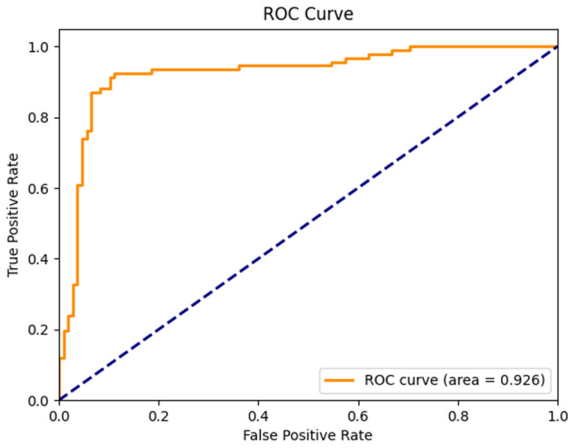

Figure 3. ROC curve

In this study, the BG-XGBoost model demonstrated a high accuracy of 0.983. In order to test whether the model is overfitting, we made a learning curve for analysis. As shown in Figure 4, the model performs very well on both the training set and the verification set. The curves of the two are close and the accuracy is similar, indicating that the model does not suffer from overfitting.

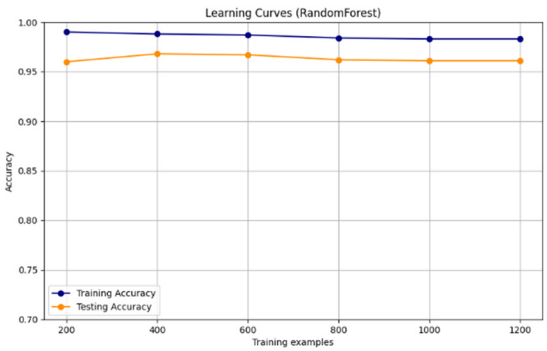

Figure 4. learning curve chart

4. Conclusion

This article mainly proposes a traffic detection model based on the BG-XGBoost algorithm, which is designed based on the characteristics of Eclipse attack traffic and Ethereum network traffic. The model includes three modules: data collection, data stream processing and attack detection. Through this design, the model can effectively solve performance problems such as data skew caused by imbalanced classification problems (such as Eclipse attack traffic detection), and show significant advantages in accuracy and other aspects.

References

[1] Zheng, Zibin, et al. "Blockchain challenges and opportunities: A survey." International journal of web and grid services 14.4 (2018): 352-375.

[2] Liu, Liang, and Budong Xu. "Research on information security technology based on blockchain." 2018 IEEE 3rd international conference on cloud computing and big data analysis (ICCCBDA). IEEE, 2018.

[3] Heilman, Ethan, et al. "Eclipse attacks on {Bitcoin's} {peer-to-peer} network." 24th USENIX security symposium (USENIX security 15). 2015.

[4] Xu, Guangquan, et al. "Am I eclipsed? A smart detector of eclipse attacks for Ethereum." Computers & Security 88 (2020): 101604.

[5] Mufleh, Alaa. Bitcoin Eclipse Attack-Statistic Analysis on Selfish Mining and Double-Spending Attack. Diss. Doctoral dissertation, JOHANNES KEPLER UNIVERSITY LINZ, 2019.

[6] Atzei, Nicola, Massimo Bartoletti, and Tiziana Cimoli. "A survey of attacks on ethereum smart contracts (sok)." Principles of Security and Trust: 6th International Conference, POST 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings 6. Springer Berlin Heidelberg, 2017.

[7] Anderson, Luke, et al. "New kids on the block: an analysis of modern blockchains." arXiv preprint arXiv:1606.06530 (2016).

[8] Marcus, Yuval, Ethan Heilman, and Sharon Goldberg. "Low-resource eclipse attacks on ethereum's peer-to-peer network." Cryptology ePrint Archive (2018).

[9] Rottondi, Cristina, et al. "Detection and mitigation of the eclipse attack in chord overlays." International Journal of Computational Science and Engineering 13.2 (2016): 111-121.

[10] Fantacci, Romano, et al. "Avoiding eclipse attacks on Kad/Kademlia: an identity based approach." 2009 IEEE International Conference on Communications. IEEE, 2009.

[11] Tran, Muoi, Akshaye Shenoi, and Min Suk Kang. "On the {Routing-Aware} peering against {Network-Eclipse} attacks in bitcoin." 30th USENIX Security Symposium (USENIX Security 21). 2021.

[12] Chicarino, Vanessa, et al. "On the detection of selfish mining and stalker attacks in blockchain networks." Annals of Telecommunications 75 (2020): 143-152.

[13] Ismail, Hatem, Daniel Germanus, and Neeraj Suri. "Detecting and mitigating P2P eclipse attacks." 2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS). IEEE, 2015.

[14] Alangot, Bithin, et al. "Decentralized and lightweight approach to detect eclipse attacks on proof of work blockchains." IEEE Transactions on Network and Service Management 18.2 (2021): 1659-1672.

[15] Dai, Qianyi, Bin Zhang, and Shuqin Dong. "Eclipse attack detection for blockchain network layer based on deep feature extraction." Wireless Communications and Mobile Computing 2022 (2022).