Detecting Fraudulent Transactions on the Ethereum Blockchain

Aman Gupta [*](aag9131)
Prakriti Sharma [†](ps4425)
Rishabh Guha [‡](rg4023)

December 20, 2022

Abstract

In this project, we build and test multiple Machine Learning models to detect fraudulent transactions on the Ethereum blockchain. We have achieved this by using a combination of predefined models such as Logistic Regression, SVM Classifier, Random Forest Classifier, and XGB Classifier along with also developing our own classification model using Keras for identifying fraudulent Ethereum transactions. We then calculate and compare the accuracy of each model to determine the best model that correctly classifies the transactions.

1 Introduction

Ethereum is a decentralized blockchain that is home to digital money, global payments, and applications. Even though blockchain technology records information about each transaction, it also assures person anonymity, as long as there is no link between the wallet and its owner's identity. Due to this reason, cryptocurrencies are more frequently used for fraudulent activities. The great thing about Ethereum is that it offers a wide range of data regarding each transaction. A number of parameters such as the amount of eth transferred along with loads of metadata regarding the sender and the receiver can help us develop a Supervised Machine Learning based novel approach that will use this data to predict the probability of a transaction being fraudulent accurately. Our motivation to work on Ethereum fraud detection is twofold. First is the fact that little work has been done in this area before and most with low classification accuracy. This is also coupled with the fact that Ethereum is This is why we aimed to create new classification models and use some existing machine learning models to identify fraudulent transactions in the cryptocurrency world with

[*]
[†]
[‡]

higher efficiency and accuracy.
We start by collecting the Ethereum transaction data set and identifying the structure of the transactions (column names, etc). We further created a logistic regression model and trained it on the retrieved dataset. The model assigns 'Fraud' and 'Non-Fraud' labels to each transaction. The accuracy of classification is calculated and documented for the best model selection. We then repeated the same steps on other machine learning models like an SVM Classifier, Random Forest Classifier, and XGB Classifier. The accuracy for each of these models is also calculated and stored. After this, we created our own classifier using Keras which is trained on the same dataset and gives a higher classification accuracy. For implementation and data visualization, we have used python libraries such as pandas, matplotlib, sklearn, numpy, xgboost, and pickle.

2 Dataset

The transaction dataset that we used contains rows of known fraud and valid transactions made over Ethereum. There are 7662 'non-fraud' transactions (77.86 percent) and 2179 'fraud' transactions (22.14 percent) to train the models accurately. Each row has 51 columns. Some of the columns are Index(the index number of a row), Address(the address of the Ethereum account), FLAG(whether the transaction is fraud or not), Avg min between sent tnx(average time between sent transactions for an account in minutes), Avg min between received tnx(average time between received transactions for an account in minutes), etc. The initial step was to pre-process and clean the data before training the model. First, we identified the columns that are not helpful in determining if a transaction is a fraud or not during classification. For example, we do not need 'Index' and 'Address' and therefore we have removed these columns from the dataset. For all the missing values in the numerical columns, we have populated those with the median of all the values in that column. Furthermore, we identified highly correlated features in the data using a correlation matrix. We found features with high correlation and thus dropped one of them as we do not need both for training our models. We also calculated the variance of all the features and identified the features with zero variance. These features will not help in training the model either and hence we have dropped these columns too. Next, we used a box plot to check the distribution of values across columns. We identified that there are 2 columns with a very low distribution of values and discarded these too. After removing the columns, we split our dataset into the train (80 percent) and test (20 percent) data. We then normalized the training data using power transforms. Since the dataset is imbalanced, we handled the imbalance in our project by oversampling the training data using SMOTE (synthetic minority oversampling technique). This is an important step because imbalanced classification can affect the performance of our machine-learning algorithms. By using SMOTE for oversampling, we have balanced the class distribution by randomly increasing the minority class examples by replicating them. Finally, we used this modified data for training our models and calculating the classification accuracy.

3 Models

In this section, we will discuss in detail the approach and results for all the machine learning models that we have used for detecting fraudulent transactions on the Ethereum blockchain.

3.1 Logistic Regression

Logistic Regression is a supervised classification algorithm that estimates the probability of an event occurring based on a given dataset of independent variables. In the classification problem, the target variable(output), y, can only take discrete values for a given set of features(inputs), X. The model builds a regression model to predict the probability that a given data entry belongs to the category number '1'. Logistic Regression models the data using the sigmoid function. Based on the number of categories, logistic regression can be classified as binomial (target variable can have 2 possible types, '0' or '1'), multinomial (target variable can have 3 or more possible types which are not ordered), ordinal (it deals with the target variables with ordered categories).
In our project, we have used a basic binomial logistic regression model to classify the transactions into 2 classes. Target variable '0' classifies the 'non-fraud' transactions and target variable '1' classifies the 'fraud' transactions. We have initialized our model as 'LR=LogisticRegression(random_state=42)', where random_state is the lot number of the set generated randomly for our task. This gives us a non-optimal model as this model misclassifies a lot of transactions.

3.2 SVM Classifier

Support Vector Machines (SVMs) are a set of supervised learning methods used for classification, regression, and outliers detection. A support vector machine constructs a hyper-plane or a set of hyper-planes in a high or infinite dimensional space, which can be used for classification, regression, or other tasks. Intuitively, we achieve a good separation by the hyper-plane that has the largest distance to the nearest training data points of any class because usually the larger the margin, the lower the generalization error of the classifier.
In our project, we have used the SVC class for our model which is capable of performing binary classification on a dataset. We have initialized our model as 'model=SVC(kernel ='rbf', C=9)', where 'rbf' is the kernel and the value of the constant 'C' is set to 9. SVM Classifier gives us the worst accuracy of all the 5 models we have tested.

3.3 Random Forest Classifier

Empirical evidence suggests that Random Forest classifiers have been known to produce high accuracies on classification problems.

Random Forest classifiers are made up of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction. The reason Random Forest works so well is that when used in large numbers, the decision trees collectively protect each other from individual errors. The outcome of a single decision tree may be wrong but most other trees will be right and will move in the right direction. The only prerequisite for this to work is that individual trees cannot be highly correlated. In the case that this happens, one tree making a wrong decision might influence its neighboring trees to make wrong decisions solely based on its outcome.
In our object, we have used a basic random forest classifier initialized as 'RF=RandomForestClassifier(random_state=42)', where random_state is the lot number of the set generated randomly for our task. This gives us a sub-optimal model with mediocre accuracy over our dataset.

3.4 XGBoost Classifier

XGBoost classifier is a machine learning algorithm that is applied for structured and tabular data. Hence, we have chosen XGB as our last predefined algorithm for modeling this problem. XGBoost is an implementation of gradient-boosted decision trees designed for speed and performance. It is an extreme gradient boost algorithm which means it is an algorithm with lots of parts. XGBoost is an ensemble modeling technique that works with large, complicated datasets. It is both a classification and regression algorithm and we have used it as a classification algorithm for our project. Ensemble learning offers a systematic solution to combine the predictive power of multiple learners. The resultant is a single model that gives the aggregated output from several models. In our project, we have initialized the XGBoost model as xgb-c = xgb.XGBClassifier(random_state=42), where random_state is the lot number of the set generated randomly for our task. This classifier gives us the best accuracy in our dataset.

3.5 XGBoost with hyperparameter tuning

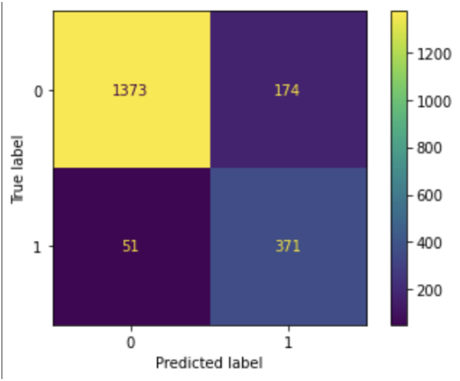
We created an XGBoost model and modeled it with hyperparameter tuning to identify the optimal model architecture. Parameters that define the model architecture are referred to as hyperparameters and thus this process of searching for the ideal model architecture is referred to as hyperparameter tuning. To do this, we explored a range of possibilities on the parameters of our XGB model. First, we created a matrix of parameters called the 'paramsGrid' that contains multiple values of the parameters that we will use to train the model. These parameters are the 'learning_rate', 'n_estimators', 'subsample', 'max_depth', and the 'colsample_bytree'. After training the model on all the combinations of values of these parameters, we calculate the accuracy on test data and identify the set of parameter values that give the most optimal model architecture with the highest accuracy on test data.

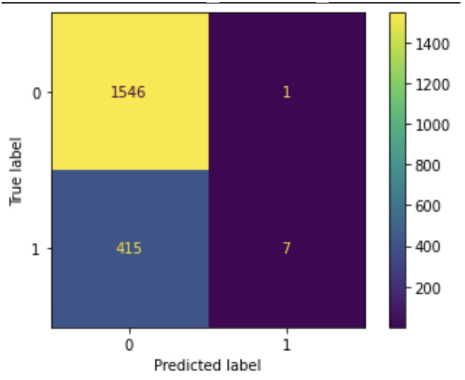
3.6 Keras Classifier

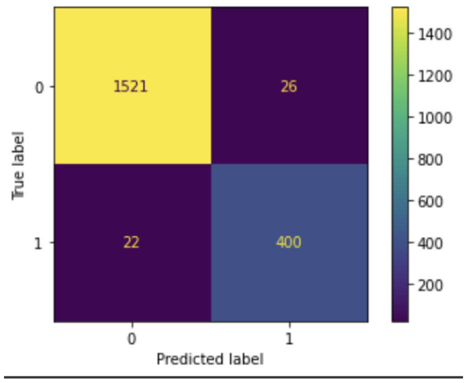
We try to build a custom sequential classifier that would predict fraudulent transactions. We define a sequential model and initialize it with Dense layers with different activation functions. The resulting model has a high training accuracy of 99.7 but when on testing we see that a lot of values are misclassified and the testing accuracy decreases to nearly 60. We then try to optimize and get better accuracy by adding more layers and changing some activation functions. On checking the confusion matrix, we see that there is no significant improvement over the old Keras model. Hence we conclude that we cannot use the current Keras models for this dataset.

4 Results

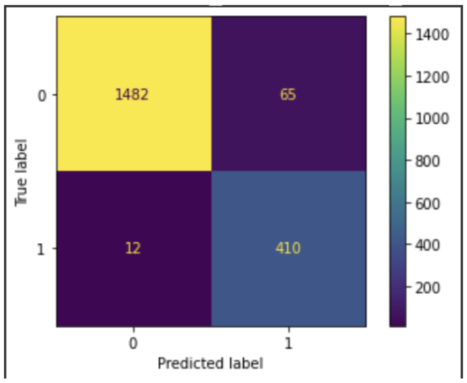
On testing various classifiers and building our own model, we conclude from the confusion matrices that the XGB Classifier works the best in detecting fraudulent transactions. To fine-tune the model, we then perform hyper-parameter tuning to the original model to check what parameters give us the best boost in accuracy. We get the following parameters as most optimal - colsample_bytree: 0.7, learning_rate : 0.5, max_depth: 4, n_estimators: 200, subsample: 0.9
Below are the confusion matrices for all the classifiers and custom models tested:



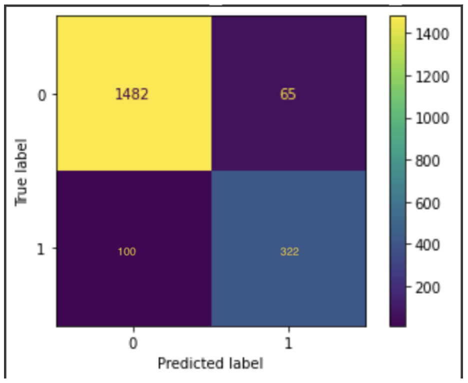




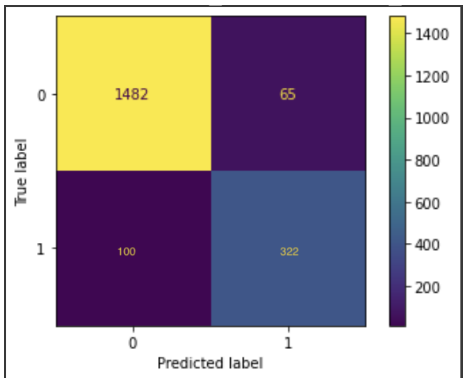
1521
26
22
400
True label
Predicted label



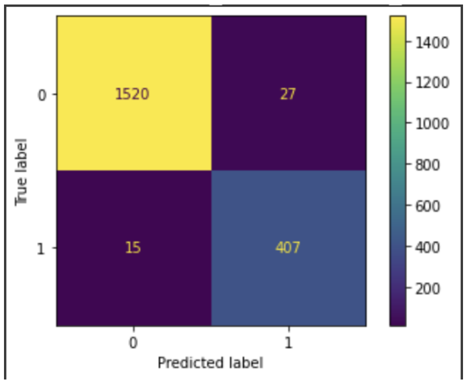
1482
65
12
410
True label
Predicted label



1482
65
100
322
True label
Predicted label



1482
65
100
322
True label
Predicted label



1520
27
15
407
True label
Predicted label

References

[1] Dataset - https://www.kaggle.com/datasets/vagifa/ethereum-frauddetection-dataset.

[2] Implementing Random forest classifier - https://medium.com/@hjhuney/implementing-a-random-forest-classification-model-in-python-583891c99652

[3] Classification using Keras models - https://www.activestate.com/resources/quick-reads/what-is-a-keras-model/

GitHub Link

Here is the GitHub link to the repository containing our project codebase:
https://github.com/amangupta42/ml-final-project

Previous Topic: Class Incremental Learning (CIL)

This paragraph aims to introduce and explain our previous topic and the reason for switching to the current topic. Our initial proposal was to implement class incremental learning on an existing dataset. The purpose of incremental learning is to make models robust to real-world situations. In reality, a machine learning model will come across a lot of situations where the input data belongs to a previously unseen class. Class incremental learning aims to counter this by training the model in such a way that it becomes robust to inputs from previously unseen classes. We aimed to create and implement an algorithm for class incremental learning on a dataset on which incremental learning was not implemented before.
After implementing the CIL algorithm on the new dataset, we planned to calculate the new classification accuracy and compare it with the existing classification accuracy on that dataset. However, it was soon imminent that the hardware and GPU specifications we had access to would not be enough to implement CIL. The main issue is that CIL requires a lot of memory to store the information of previously seen classes. This would mean that we would have to store class information for all classes in the data set as well as for each iteration of batch training. Additionally, all previously stored data would then have to be fetched to be used in every batch of training. This would make the whole project very computationally intensive. This step in the process is essential to the working of CIL as it helps avoid a phenomenon known as Catastrophic Forgetting. Hence, the inability to execute or remove this step was one of the major reasons for having to discard this topic.
Though we made some progress on this topic, we did not foresee these challenges and had to shift to the new (current) topic of detecting fraudulent transactions on the Ethereum blockchain.