DATA MINING PROJECT REPORT: FRAUD DETECTION ON ETHEREUM CLASSIC TRANSACTIONS

2021-2022

Contents

Introduction:

A cryptocurrency is a digital only currency in which transactions are validated by a decentralized network , not by a centralized authority. The history of the transactions is recorded on a record called the blockchain, which is basically a long ledger divided into blocks. A new block can only be added after it gets signed by individuals (or groups) of workers called miners, thus the decentralization. Bitcoin was the first application to use the blockchain platform, and it was proposed by Satoshi Nakamoto (a person, or group of persons, which the identity remains unknown). In order to correct some of the flaws that Bitcoin had, and palliate how hard mining it became, other cryptocurrencies were introduced. The most important one after Bitcoin is to this day Ethereum (ETH). It was created by Vitalik Buterin in 2014, and was the second implantation to have ever used to the blockchain. The figure below represents the roadmap of Ethereum . In the following figure, we can see that in 2016 there was a hard fork that gave birth to a new cryptocurrency : Ethereum Classic (ETC). This transition was consequence of an attack called the DAO attack.

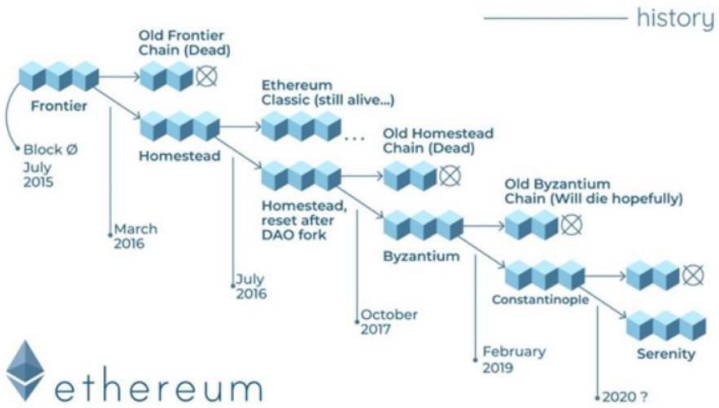

Figure 1 : Ethereum roadmap

In the following project, we will be working on the Ethereum classic (ETC) cryptocurrency (even if, in some works the latter is sometimes confused with ETH since the two of them share the same history of before march 2016).

REMARK: This project was a collective work. The tasks were shared equitably and we did almost everything together.

I. Project motivation:

The Ethereum network, is subject to a number of assaults. The most famous attack, that we already mentioned, is known as a Distributed Autonomous Organization (DAO) attack, and it happened in 2016. The DAO is a contract code that has a flaw that has resulted in the theft of more than USD 50 million [1]. The phishing scam is another sort of fraud that has emerged on the Ethereum network since its inception. It is a type of fraud that obtains personal information and disseminates fraudulent account addresses. Phishing websites try to steal Ethereum wallet keys, or online to acquire money directly. There are lots of other kinds of fraud: Ponzi schemes, Hake, Scamming, … According to cnbc.com, over 10$ billion were lost to decentralized finance scams in 2021.

All of these frauds make it mandatory to have intrusion detection system (IDS) on blockchain networks. However, such a system mining a (big enough) dataset, labeling it, and making a good classifier. We will try in the following project to cover all these steps.

We will try to answer the following questions:

1. Are scammers databases enough to create anomaly detection models ?
2. Which classifier works the best for this task?
3. Which features are best determining the fraudulent nature of a transaction, in this context?

II. Data acquisition workflow:

1. Data mining:

We will analyze the Ethereum classic transactions. Those transactions are available on a dataset of google public datasets database, that is accessed here : https://console.cloud.google.com/marketplace/product/ethereum-classic/crypto-ethereumclassic. The first step is to understand the database metadata, we start by running an Sql Big query on the google cloud console. The query results are available with the project appendix as ETC_table_list.csv.

```
#standardSQL
SELECT *
FROM `bigquery-public-data.crypto_ethereum_classic`.__TABLES__ ;
```

Emplacement de traitement : US

Résultats de la requête ENREGISTRER LES RÉSULTATS EXPLORER LES DONNÉES

Requête terminée (durée : 0,3 s/octets traités : 0 octets)

Informations sur la tâche | Résultats | JSON | Détails de l'exécution

Ligne	project_id	dataset_id	table_id	creation_time	last_modified_time	row_count	size_bytes	type
1	bigquery-public-data	crypto_ethereum_classic	balances	1579787245638	1606139534380	93200492	5592029520	1
2	bigquery-public-data	crypto_ethereum_classic	blocks	1548355775086	1606138851446	11661235	12890285980	1
3	bigquery-public-data	crypto_ethereum_classic	contracts	1548355742492	1606138961489	76444737	14553037294	1
4	bigquery-public-data	crypto_ethereum_classic	logs	1548355748843	1606138889078	12143899	5136290446	1
5	bigquery-public-data	crypto_ethereum_classic	token_transfers	1548355712053	1606138888457	725394	223995450	1
6	bigquery-public-data	crypto_ethereum_classic	tokens	1548355516717	1606138888759	10415	1763461	1
7	bigquery-public-data	crypto_ethereum_classic	traces	1548355780184	1606138962741	227130552	86922492631	1
8	bigquery-public-data	crypto_ethereum_classic	transactions	1548355780439	1606138962132	63560582	25665563036	1

Figure 2 : BigSql query to display tables

The table “transactions”, marked on the figure, will be our table of interest. We then run the following query in order to display the columns of our table. The result is displayed in columns.csv.

```
SELECT table_name, column_name, is_nullable, data_type, is_partitioning_column
FROM `bigquery-public-data.crypto_ethereum_classic`.INFORMATION_SCHEMA.COLUMNS as Trans
WHERE Trans.table_name = 'transactions';
```

Emplacement de traitement : US

Résultats de la requête ENREGISTRER LES RÉSULTATS EXPLORER LES DONNÉES

Requête terminée (durée : 0,4 s/octets traités : 10 Mo)

Informations sur la tâche | Résultats | JSON | Détails de l'exécution

Ligne	table_name	column_name	is_nullable	data_type	is_partitioning_column
1	transactions	hash	NO	STRING	NO
2	transactions	nonce	NO	INT64	NO
3	transactions	transaction_index	NO	INT64	NO
4	transactions	from_address	NO	STRING	NO
5	transactions	to_address	YES	STRING	NO
6	transactions	value	YES	NUMERIC	NO

Figure 3 : columns of the transactions table

Below is a summarized description of the fields.

Field	Field type	Description
Hash	STRING	The transaction’s hash
Nonce	INT64	How many transactions the sender has performed
transaction_index	INT64	Index of the transaction in its block
From_address	STRING	Sender address
To_address	STRING	Receiver address
Value	NUMERIC	The value transferred in Wei. Wei is a fraction of Ethereum (classic, eventually).
Gas	INT64	The amount of consumed gas. Gas is the cost of the transaction and is calculated in Wei or GWei.
Gas_price	INT64	The current gas_price
Input	STRING	Data transmitted along transaction
receipt_cumulative_gas_used	INT64	The amount of gas consumed upon the execution of the block.
receipt_gas_used	INT64	The total amount of gas the transaction used by itself.
receipt_contract_address	STRING	The address of the smart contract related to the transaction.
receipt_root	STRING	The root of the receipt.
receipt_status	INT64	Indicated whether the transaction has been completed of not.
Block_timestamp	STRING	The time of execution of the transaction.
Block_number	INT64	The number of the block in which the transaction belongs.
Block_hash	STRING	The hash of the block.

Table 1 : Fields of the transactions table

2. Data cleansing:

As the objective of our project is to identify fraud within transactions, many fields that are unrelated to that were to be eliminated. We will list them in this section and explain our choice:

- Hash : this information is just a cryptographic text our of which our model will not extract information.

- From_address, To_address : those hexadecimal values cannot be encoded into suitable formats, and hence will be dropped, also keeping them will deviate the project into only classifying the already existing adresses into Fraud/Non-Fraud.
- Receipt_root & Receipt_status : Those fields are related to whether transactions have been completed or not. However, we are only interested in those that has been completed. Thus we will only keep the rows having 'receipt_status' = 1.
- Block_number : This feature is only to be kept on an eternally learning model, but it is not the case for our model. To explain: suppose we train our model with data provided from 2018 to 2019. 2018 was a year on which much more scams happened than the adjacent years. A model that's not receiving new entries can understand that blocks with lower indexes are more prone to scams, and hence become unscalable.
- Block_hash : same as hash.

3. Data labeling :

The raw data do not display either a transaction is fraudulent or not. However, a good classification model needs accurate labels. We propose, for this task, to use the opensource API provided by the database cryptoscamdb.org. A python program (provided in the appendix, at the end of our Jupyter notebook) extracts the scammers addresses.

```
import requests

# the scam database api-endpoint
URL = "https://api.cryptoscamdb.org/v1/addresses"
r = requests.get(url = URL)

# extracting data
query_result = r.json()
res = query_result['result']

# extraction the addresses of scammers
scammers = []
if ( query_result['success']):
    for adr in res:
        dic = res[adr][0]
        if (dic['type'] == 'scam'):
            scammers.append(dic['address'])

print("The number of fraudulent addresses in the database is :" ,len(scammers))
```

```
The number of fraudulent addresses in the database is : 4228
```

```
scammers
```

```
'0x4307e7D64a0F936bB719DDa5CA177F493F846228',
'0x4314AcF1cbbd1081cE588D933a17995fce349C0B',
'0x433c216A36681787A4603f2EECBd2e5F4085C7fb',
'0x4349B24Fe0Bb23c51569E63B2c1d457Dd754647D7',
'0x435fe8272698cA7032c41CBe5d412e0369d159fF',
```

Figure 4 : extraction of the list of 4228 scammers

As we can see, we have the addresses of only 4228 scammers. However, according to , there are about 50000000 unique ETC addresses active by 2021. We clearly see that using what was provided from EtherscamDn will lead us to have a highly unbalanced dataset. This is why, for convenience reasons, we will recur to using an opensource dataset that was shared by researcher Salam R. El Amari on GitHub [2] . This dataset was created with the same method we used but instead of CryptoScamDb, it used EtherScamDb which is now inaccessible.

4. Handling imbalance:

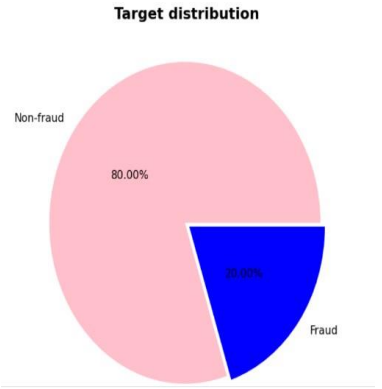

Figure 5 : Fraud / Non Fraud distribution

After studying the target distribution of the data, the two following screenshots show that it is imbalanced:

A lot of techniques can be presented to handle this problem. Among them, here is a list of some possible solutions:

a. Ensemble different resampled datasets

Using all the samples from the minority class, splitting the majority class in n equal smaller datasets and training them with the rare class. At the end, collecting the results together to build the final model.

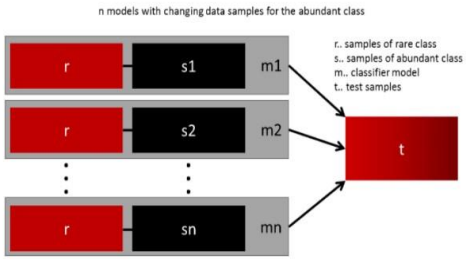

Figure 6 : Ensemble different resampled datasets

b. Resample with different ratios

Same as the first technique, but this time the n smaller datasets from the majority class are in different proportions.

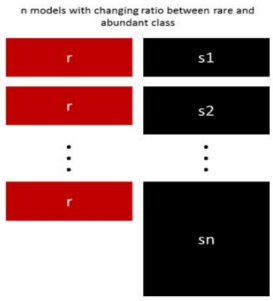

Figure 7 : resample with different ratio

c. Cluster the abundant class

Clustering the majority class in m different clusters and using only the medoids of each cluster.

d. Resample the training set

1. Under-sampling

It consists in picking random samples from the majority class and keeping all the minority class, so that we have a balanced dataset. It is used when the dataset is sufficient.

2. Oversampling

On the contrary, when the quantity of data is insufficient it is preferable to keep the majority class and generate other samples of the majority class. In this sense, many oversampling techniques are available and ready to be implemented. We chose the SMOTE technique: Synthetic Minority Oversampling Technique. It consists of selecting examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample at a point along that line.

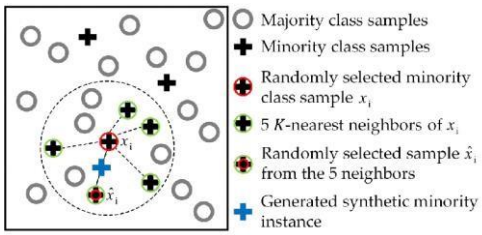

Figure 8 : oversampling with smote[4]

These two last techniques will be used to handle the problem of unbalance in the considered dataset.

Remark: Oversampling must be done after splitting test and train data, unless results can be very confusing and misleading because many redundant samples can be exist simultaneously in the train and the test sets.

III. Models and evaluation:

1. Logistic regression

The confusion matrix of the logistic regression is the following :

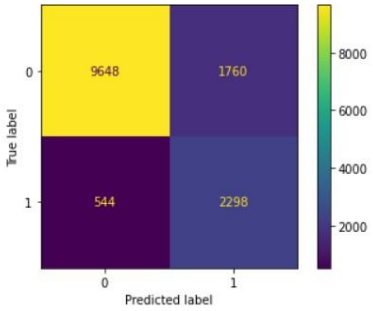

Figure 9 : confusion matrix of the Logistic Regression model

This model, correctly identified 2298 (TP) of FRAUD cases, out of 2832 (P).

flagged as FRAUD 1760 (FP) out of 11408, when this cases were actually NON-FRAUD.

Dealing with a fraud detection scenario, we care more about the transactions that were actually FRAUDS, but which were treated as NON-FRAUD by our model (FN - 544)

Thereby, we have to try to increase the precision using other methods

2. Random forest classifier

Random Forest Classifier model, correctly identified 2623 (TP) of FRAUD cases, out of 2832 (P).

Random Forest Classifier model flagged as FRAUD only 16 (FP) out of 11408, when this cases were actually NON-FRAUD

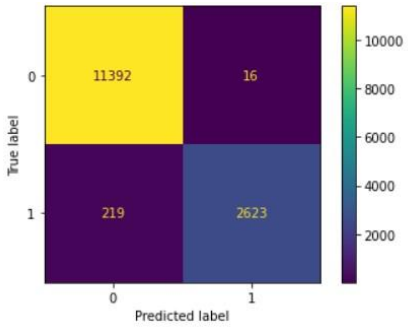

Figure 10 : confusion matrix of the random forest classifier

We can conclude that :

- The Random Forest Classifier model seems to produce more effective results
- Both FP and FN are reduced considerably increasing the recall & precision

3. KNN Classifier:

The challenge in the K-Nearest Neighbors algorithm is the hyperparameter-tuning phase: To find the best value of K, that is associated to the optimal performance.

i. Training the model: With an initial value of k equal to three.

j. Evaluating the model: on both the training and testing datasets

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

Figure 11 : RMSE expression

The RMSE or Root Mean Square Deviation reflects the dispersion/ variability in the quality of prediction; it can be related to the variance of the model.

Naturally, the RMSE reflects a higher value with the test data, which is the case:

```
#evaluation on training data:
from sklearn.metrics import mean_squared_error
from math import sqrt
y_train_predicted = knn_model.predict(x_tr_resample2)
mse = mean_squared_error(y_tr_resample2, y_train_predicted)
rmse = sqrt(mse)
rmse
```

```
0.3261232563188105
```

```
[77] #evaluation on test data:
     test_preds = knn_model.predict(x_test)
     mse = mean_squared_error(y_test, test_preds)
     rmse = sqrt(mse)
     rmse
```

```
0.3336606580012929
```

Figure 12 : RMSE for both training and test sets

Other evaluation metrics:

```
metrics.accuracy_score(y_test, test_preds)
0.8651929824561404

print (confusion_matrix(y_test, test_preds))
[[10561   847]
 [ 1074  1768]]
```

Figure 13 : confusion matrix for the KNN with k=3

k. Tuning of the hyperparameter k:

In order to choose the value of k, with which the model shows the optimal prediction performances (Mainly, the goal here is to reduce the error), we tried to train the model with different values of k, ranging from 1 to 10.

The GridSearchCV pre-defined function in SK-learn allows to do so.

The obtained results show the optimal performances of the model with a value of k equal to=5

```
gridsearch.best_params_
{'n_neighbors': 5}
```

Figure 14 Best value of k

And the optimal performances are:

```
Errors For the train dataset:  0.3031217353700718 For the test dataset:  0.3483051696832394
[[9929 1479]
 [ 812 2030]]
0.8392280701754385
```

Figure 15 Evaluation for the KNN model with k=5: Errors, Confusion Matrix and accuracy

4. SVM Classifier:

a. Results before any data transformation:

	precision	recall	f1-score	support
0	0.87	0.91	0.89	11408
1	0.55	0.44	0.49	2842
accuracy			0.82	14250
macro avg	0.71	0.67	0.69	14250
weighted avg	0.80	0.82	0.81	14250

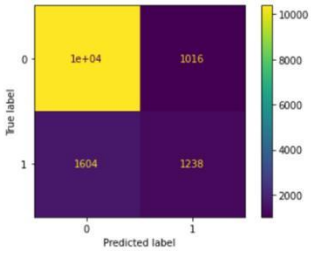

Figure 16 : Evaluating SVM before any data oversampling/undersampling

The confusion matrix shows that the non-fraud class, which is the majority class is better predicted than the samples of the minority class.

b. Results after oversampling:

The dataset we obtained after oversampling is a huge dataset with almost 91000 data samples.

The SVM classification method is not a good choice, in this case, because the training complexity of SVM is highly dependent on the size of data set.

```
from collections import Counter
counter = Counter(y_tr_resample)
print(counter)
```

```
Counter({0: 45592, 1: 45592})
```

Figure 17 Dataset distribution after oversampling

The SVM with over-sampled data, did not execute in more than 2 hour-time and is up to even more.

Executing (2h 30m 1s) Cell > fit() > _dense_fit()

Figure 18 Execution time

So to palliate this problem, we thought about the under-sampling technique below.

c. Results after under-sampling:

	precision	recall	f1-score	support
0	0.89	0.81	0.85	11408
1	0.44	0.61	0.51	2842
accuracy			0.77	14250
macro avg	0.67	0.71	0.68	14250
weighted avg	0.80	0.77	0.78	14250

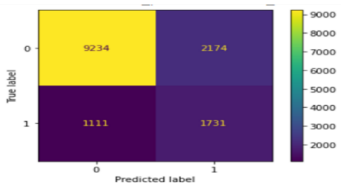

Figure 19 : Evaluating SVM after data undersampling

The idea of under-sampling came in order to reduce the execution time.

The results we obtained below, are better than those obtained without any data manipulation (ie with unbalanced data) as the numbers of well-predicted samples for both 1 and 0 labels is now bigger than those for false predictions. (Which wasn't the case for the class 1 before undersampling).

d. Evaluations and interpretations: I. <u>Confusion matrix:</u>

The confusion matrices by the SVM compared, show that only the 1-class predictions improved after under-sampling. The results aren't satisfying, even though the majority of samples are well predicted in both classes.

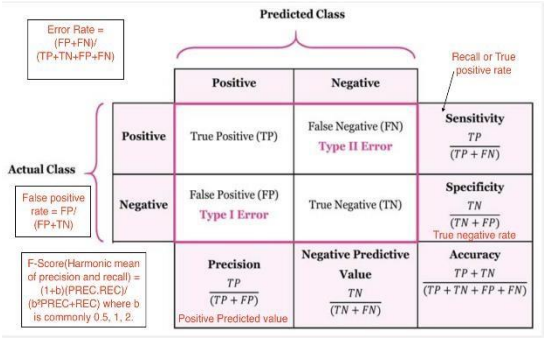

Figure 20 Confusion matrix and evaluation scores

II. However, the <u>accuracy</u> shows good results with the first, imbalanced data (=0.82). And a lower value with the undersampled data(=0.77).

This can be explained by the fact that accuracy do not distinguish between correct attributes of the two classes. In this sense, if the model is predicting good enough the majority class and not even predicting a correct value from the minority class, it will have a good accuracy.

==> Accuracy is not a good metric in our case.

5. Neural Network : (MLPClassifier)

`MLPClassifier` implements a multi-layer perceptron (MLP) algorithm that trains using Backpropagation.

The neural network model, correctly identified 1188(TP) of FRAUD cases, out of 2832 (P).

It flagged as FRAUD 65(FP) out of 11408, when this cases were actually NON-FRAUD

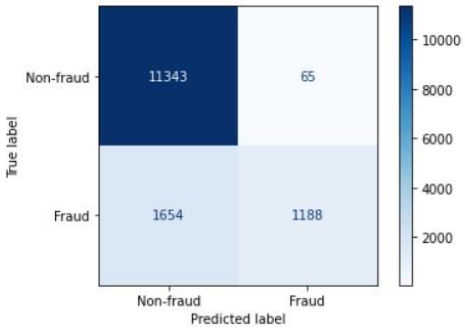

Figure 21 : confusion matrix of the neural network

IV. Conclusion:

After checking our results, we can come out with the following conclusions:

- Out of the different models we used, the random forest classifier seems to work best with our data.
- Labeling blockchain transaction with basing ourselves ONLY on scam detection databases is not enough to the very high number of transactions and also the high number of non-reported scammers. This creates unbalanced datasets and we need to palliate to their problems.
- Detecting scam based on pure transactions data might not be the best approach to flag fraud within a blockchain network, however, it is indeed effective (as do show the results of random forest classifier) and we can use it in practice to help another system decide.

Tables of figures

Bibliography:

[1] Al-Emari, S. R. (n.d.). A Labeled Transactions-Based Dataset on the Ethereum Network.

[2] Mehar, M. e. (2019). Understanding a revolutionary and flawed grand experiment in blockchain:.

[3] Radewagen, Y. W. (n.d.). *kdnuggets*. Retrieved from https://www.kdnuggets.com/2017/06/7-techniques-handle-imbalanced-data.html

[4] *Integrating Growth and Environmental Parameters to Discriminate Powdery Mildew and Aphid of Winter Wheat Using Bi-Temporal Landsat-8 Imagery, link:*

https://www.researchgate.net/figure/The-basic-principle-of-the-synthetic-minorityoversample-technique-SMOTE-algorithm_fig2_332279195