Building a Machine Learning Model to Verify Wallet Addresses

In my previous post, I mentioned my contribution to our project: a Wallet Verification System. This system allows users to input a wallet address, retrieve its transaction history, and test it against a machine learning model to determine whether the wallet is associated with illicit activity. In this post, I will walk through the process I followed to develop this system.

Technology Used

- **Programming Language:** Python
- **APIs:** Moralis API
- **Libraries:** Streamlit, Moralis, Flask, Requests, Pandas, Scikit-Learn
- **Dataset:** Elliptic dataset

Development Process

Creating the Machine Learning Model

The first step was developing the machine learning model. I used Scikit-Learn to create a **Random Forest Classifier (RFC)**. The model was trained on a dataset provided by Elliptic. However, before training, I needed to modify the dataset so it would be compatible with the transaction data retrieved from the API.

I chose **Moralis' API** because it provides a simple and efficient way to retrieve blockchain data through its Python library. Compared to other indexers I tested, Moralis was much easier to set up and integrate into the project.

Once the API was established, I developed a script to fetch wallet transaction data, clean it, and ensure it matched the format required by the machine learning model. Another script was then created to combine the data retrieval process with the machine learning model. This script evaluated the wallet's transaction history, provided a classification (licit or illicit), and displayed the confidence level of the prediction on the front end.

Results

The model was able to classify wallets semi-effectively but was largely inaccurate due to the limited data. However, there were some cases where it was consistently distinguishing between licit and illicit wallets.

Evaluation and Challenges

Dataset Limitations

One of the biggest challenges was dataset compatibility. The Elliptic dataset consists entirely of **Bitcoin** transactions, while my Wallet Verification System was designed for **Ethereum** addresses. This difference required extensive data cleaning and conversion. However, the modification significantly reduced the number of features available for training, from 58 parameters in the original dataset down to only 8.

Bitcoin and Ethereum also differ in transaction structure. Bitcoin's transaction history is linked to previous transactions, whereas Ethereum operates on an account-based model. Additionally, timestamps in the dataset were recorded in "timesteps" (each representing a two-week period), while Moralis provided timestamps as precise date-time values. These differences further contributed to inaccuracies in the model.

Classification Report

The effectiveness of the model was assessed using Scikit-Learn's classification report, which includes:

- **Class:** The labels assigned by the model (1 = illicit, 2 = licit)
- **Precision:** The percentage of correct predictions for each class
- **Recall:** The percentage of actual instances correctly identified
- **F1-score:** A balance between precision and recall
- **Support:** The number of samples for each class in the dataset

Here are the classification results:

Full dataset, original parameters:

- Illicit: Precision 96%, Recall 79%, F1-score 87%
- Licit: Precision 99%, Recall 100%, F1-score 99%

Adapted dataset, limited parameters:

- Illicit: Precision 87%, Recall 85%, F1-score 86%
- Licit: Precision 89%, Recall 91%, F1-score 90%

Full dataset, adapted parameters:

- Illicit: Precision 77%, Recall 56%, F1-score 65%
- Licit: Precision 98%, Recall 99%, F1-score 98%

As shown, reducing the number of features made the model about **10% less precise** when trained on the adapted dataset. Additionally, because the original dataset contained far more licit transactions than illicit ones (over **10 times more**), the model struggled with class imbalance.

Improving the Model

To balance the dataset, I removed a significant portion of licit transactions. However, a better approach could be to use **SMOTE (Synthetic Minority Over-sampling Technique)**, which generates artificial data points for the underrepresented class. SMOTE works by creating synthetic examples using **K-nearest neighbor (KNN)** to make the dataset more balanced. However, due to the extreme imbalance between licit and illicit transactions, SMOTE might not generate realistic enough data.

Additionally, **Random Forest Classifier (RFC)** may not have been the best choice for this project. RFC models are highly sensitive to class imbalances, as seen in my classification

reports. A better alternative could be **XGBoost (Extreme Gradient Boosting)**, which improves model accuracy by correcting errors iteratively during training. Other possible algorithms include:

- **LightGBM** – Optimized for large datasets
- **CatBoost** – Designed for datasets with more categorical variables

Performance Issues with Large Wallet Histories

Another issue I encountered was **processing time**. Wallets with extensive transaction histories required significantly more time to process. This also resulted in higher compute unit usage on Moralis, increasing costs.

Potential solutions:

- **Limit API pages:** Reducing the number of transaction history pages retrieved would speed up processing.
- **Filter necessary data:** Currently, my script fetches all transaction details, even though only three features are needed. However, Moralis does not currently offer fine-grained filtering options.

Final Thoughts

Although my model is not yet fully accurate, this project was an exciting opportunity to experiment with machine learning and blockchain data. I learned a great deal about **machine learning models, data preprocessing, and API integration**. If I were to improve the model further, I would focus on:

- Finding an Ethereum-specific dataset with more parameters
- Implementing an alternative algorithm like XGBoost
- Optimizing API calls to reduce processing time

A huge thanks to **@Moralis** for their smooth API and **@Elliptic** for providing an extensive dataset.

If you want to explore my work, check it out here:
GitHub Repository

Glossary

- **Random Forest Classifier (RFC):** A machine learning algorithm that trains multiple decision trees using different parts of the dataset. Each tree votes on the classification, and the majority decision is chosen. The RFC also provides a confidence score indicating how many trees agreed on the classification.
- **Indexer:** A tool that collects and organizes blockchain data, making it more accessible for users.
- **Bitcoin & Ethereum:** Cryptocurrencies used on the blockchain, each with its own transaction structure and data formats.