PREDICTION OF ILLICIT TRANSACTIONS ON THE BITCOIN BLOCKCHAIN USING MACHINE LEARNING

An Undergraduate Research Scholars Thesis

by

JACK SEBASTIAN

Submitted to the LAUNCH: Undergraduate Research office at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by
Faculty Research Advisor: Dr. James Caverlee

May 2023

Majors: Computer Science
Applied Mathematics

RESEARCH COMPLIANCE CERTIFICATION

Research activities involving the use of human subjects, vertebrate animals, and/or biohazards must be reviewed and approved by the appropriate Texas A&M University regulatory research committee (i.e., IRB, IACUC, IBC) before the activity can commence. This requirement applies to activities conducted at Texas A&M and to activities conducted at non-Texas A&M facilities or institutions. In both cases, students are responsible for working with the relevant Texas A&M research compliance program to ensure and document that all Texas A&M compliance obligations are met before the study begins.

I, Jack Sebastian, certify that all research compliance requirements related to this Undergraduate Research Scholars thesis have been addressed with my Faculty Research Advisor prior to the collection of any data used in this final thesis submission.

This project did not require approval from the Texas A&M University Research Compliance & Biosafety office.

TABLE OF CONTENTS

Page

ABSTRACT

Prediction of Illicit Transactions on the Bitcoin Blockchain Using Machine Learning

Jack Sebastian
Department of Computer Science and Engineering
Texas A&M University

Faculty Research Advisor: Dr. James Caverlee
Department of Computer Science and Engineering
Texas A&M University

With the emergence of cryptocurrencies and blockchain technology, the paradigm of the structure of data storing and distribution has completely changed. And while the central goal of Bitcoin's 2008 whitepaper was to create internet-based peer-to-peer money without a central third party, there are some unforeseen issues that come with having a completely decentralized ledger. Money laundering is possible by moving illicit funds through hundreds of wallets before depositing the funds and cashing out with a crypto exchange. And with these methods of money laundering becoming more advanced over the decades, the advent of cryptocurrency means a new venue for criminals to carry out more malicious schemes. Anti-money laundering techniques have given way to software within the world of technology that is becoming heavily used by financial institutions to analyze and detect suspicious data.

The Elliptic Data Set, which maps Bitcoin transactions to real entities categorized as either from a licit or illicit group, is utilized in this study. It is known as the largest data set that is publicly available by any cryptocurrency and is made up of over 200,000 nodes and 230,000 edges. To aid in the fight against money laundering schemes, we discuss and analyze areas of machine learning that can benefit AML. Visualization of the graph constructed from the data set is examined to show any promising patterns in the location of licit vs illicit nodes. Furthermore, various binary

classification models are used on the Elliptic data set in analyzing performance of prediction of illicit nodes within the network. The classification models include Logistic Regression, K-Nearest Neighbours, Decision Trees, Multilayer Perceptron, Random Forest, and Graph Attention Networks. The results of running these models on the data set provided insight into how nodes in the blockchain network are structured while also demonstrating innovate ways that machine learning can be leveraged within the AML industry.

ACKNOWLEDGMENTS

Contributors

I would like to thank my faculty advisor, Dr. Caverlee, for his guidance and support throughout the course of this research. Thanks also go to my friends and colleagues and the department faculty and staff for making my time at Texas A&M University a great experience.

The data analyzed/used for this paper were provided by the Kaggle platform. The technology given to me in order to run some of the models on stronger GPUs for quicker runtime was provided by Dr. Caverlee's lab.

All other work conducted for the thesis was completed by the student independently.

Funding Sources

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

NOMENCLATURE

BTC	Bitcoin
AML	Anti-Money Laundering
ML	Machine Learning
MLP	Multilayer Perceptron
RF	Random Forest
GAT	Graph Attention Network

1. INTRODUCTION

Cryptocurrency has become one of the fastest growing technological trends of the decade. Somewhere between 2.9 and 5.8 millions of private and instituional users actively participate in the crypto system and run various transaction networks [1]. Having originally been associated with the dark web and illicit trading, it has become a staple of digital transactions and offers promising solutions in financial speed and security. Instead of being physical money that is carried around and exchanged, crypto payments are completely digitized and are stored in an online database. Unfortunately, crimes within the world of cryptocurrency is exponentially growing and are susceptible to things like scams, Ponzi schemes, and fake websites posing as authentic exchanges. Before diving into the new threats and challenges that cryptocurrencies pose on AML methods, its important to understand how cryptocurrency and blockchain technology works.

1.1 Cryptocurrency and blockchain

Cryptocurrencies are digital assets that are designed to be used as forms of exchange. Inherently, its a form of digital currency where all transactions are verified and records maintained by a decentralized system. The term “crypto“ is in reference to the various encryption algorithms and cryptographic methods that safeguard these transaction entries in the blockchain ledger. At this point in time, Bitcoin is the most popular and valuable cryptocurrency. Invented by Satoshi Nakamoto, Bitcoin is most traded cryptocurrency in the world and has a total market cap of around $447 billion [2].

The term “blockchain“ is thrown around when discussing anything related to crypto, and there seems to be a common mystery behind what blockchain is and how it works. Blockchain technology is a decentralized, distributed ledger that stores record of ownership of digital assets. A blockchain is also shared among all nodes in the computer network, which is the backbone behind its security. As the name suggests, blockchain are connected blocks of information that store data related to transactions, with each independent block pertaining to a set of transactions that have

been verified by members of the network. Decentralized blockchains are also immutable - data entered and stored in a block is irreversible and cannot be changed.

1.1.1 Decentralization

Decentralization within blockchain ensures that there is no governing body or group that has power over the ledger, contrasting to how a bank owns and controls the the ledger of everyone at the institution. This decentralized aspect of blockchain relieves anyone from needing to trust other users within the network, as changes in the ledger (through tampering or malicious means) are reflected on everyone's copy and this allows for it to be checked for legitimacy. And while this might seem like a total advantage over having a governing body in charge of the ledger, there are numerous disadvantages to this kind of system. The obvious drawback is a reduced performance and higher cost of maintenance. Being reliant on a distributed network over a single authority heavily reduces the networks performance since multiple members need to reach a consensus on verifying transactions.

1.2 Money Laundering

Money laundering is the process of giving large amounts of illegally obtained money the appearance of having originated from a legitimate source. Some of the most common money laundering methods include drug trafficking, terrorist funding, etc. And while the main goal of these criminal acts are generating huge profits for the individual or organization, it can have serious negative effects on the economy. For example, carrying out tax evasion can result in "loss of government revenue, which affects government's potential spending on development schemes" [3] and therefore indirectly affecting any social group that could've benefited from this potential government expenditure. A high-level United Nations panel estimated that money laundering flows at $1.6 trillion on an annual basis, which accounts for 2.7% of global GDP in 2020 [4]. As money laundering professionals become more specialized and sophisticated, the need for anti-money laundering regulations is essential for maintaining stability within the global sector.

1.2.1 Anti-Money Laundering

Anti-money laundering are the combative methods of fighting against money laundering schemes. These can be through laws, regulations, and procedures that are aimed at revealing efforts to disguise illicit funds as legitimate. There have been strides in AML legislation that date as far back as 1970, with the Bank Secrecy Act that passed in part to prevent organized crime. More recently, the UN included AML provisions in multiple international conventions to address drug trafficking, organized crime, and corruption. The Anti-Money Laundering Act of 2020 that passed was the most extensive overhaul of US AML regulations, which included the Corporate Transparency Act which made it more difficult to use shell companies to evade AML and economic sanctions measures [5].

1.2.2 AML in cryptocurrency

AML measures are an essential component of the regulatory framework surrounding cryptocurrencies. Because cryptocurrencies are decentralized and often pseudonymous, they present unique challenges for AML efforts. However, many jurisdictions have implemented regulations and guidelines designed to increase transparency and combat illicit financial activity in the cryptocurrency space. For example, the Financial Action Task Force (FATF), an intergovernmental organization, has established guidelines for regulating virtual assets and service providers. Additionally, some countries require cryptocurrency exchanges to register as money services businesses and comply with AML and know-your-customer (KYC) requirements. According to a report by Chainalysis, a blockchain data analytics firm, compliance with these requirements has improved over time, suggesting progress in the fight against illicit financial activity in the cryptocurrency ecosystem [6]. While there are still concerns about the effectiveness of AML measures in the cryptocurrency space, ongoing efforts to address these challenges indicate a commitment to combating financial crime and ensuring the legitimacy of the cryptocurrency industry.

1.3 Elliptic Data Set

The data set that is utilized in this study is the Elliptic Data Set. Released in 2019 by Elliptic, it is the world's largest data set of labeled transactions publicly available for any cryptocurrency. Elliptic is one of the leaders in blockchain monitoring solutions for regulatory compliance and risk management by cryptocurrency businesses and financial institutions [7]. The Elliptic data set maps Bitcoin transactions to real bodies that belong to either two categories: licit (exchanges, wallet providers, miners) or illicit (scams, malware, terrorist, organizations). That is, a given transaction in the data set is licit if the entity that generated it was licit.

There are 203,769 nodes and 234,555 edges. Nodes in this case are node transactions and edges are described as "the flow of Bitcoin". 2% of the nodes are illicit and 21% are licit, with the other 77% of the transactions in the data set being unknown in their nature. Each node has an associated 166 features with it, with the first 94 features represent local information about the node (transaction fee, output volume, time step, number of inputs/outputs). The other 72 features are aggregated features that are obtained by aggregating transaction information such as maximum, minimum, standard deviation, correlation coefficients of neighbor transactions. As stated by Elliptic, this was carried out by using transaction information one-hop backward/forward from the center node. In addition to the features, there is also temporal information about the data. A time stamp is associated with each node, representing an estimated of the time when the transaction is confirmed. There are 49 distinct timesteps evenly spaced with an interval of 2 weeks.

1.4 Future plans and goals

The overall objective of the experiments performed is to accurately predict the classification of nodes in the Bitcoin blockchain. However, other goals include highlighting the strengths (and weaknesses) of various binary classification models in their ability to capture the relationship of graph-structured data. Furthermore, graph neural networks (GATs) will be explored in their capability and power of representing graph-structured data and will be expected to outperform other binary classification models.

2. METHODS

It's imperative that our data is processed and formatted in the way we need it to be in order for it to work with the models we are using. Our Elliptic dataset undergoes data preprocessing for the utilization of several binary classification methods in this study. This section will aim to describe the importance and applicability of data preprocessing, what binary classification is and describe each of the binary classifiers used, and demonstrate. All of these things are important in providing insight into the overall structure of the Elliptic dataset.

2.1 Data Preprocessing

Data preprocessing is an essential step in machine learning because it ensures that the data used to train the model is of high quality. Real-world data can be messy, inconsistent, or incomplete, making it challenging for machine learning models to learn from. By preprocessing the data, we can transform it into a format that is easier for the machine learning algorithm to understand and learn from. Data preprocessing is crucial for improving the accuracy and efficiency of machine learning models. By preparing the data properly, we can ensure that the model is more accurate and less prone to errors, making it more useful for real-world applications. Poorly preprocessed data can lead to inaccurate and unreliable results, which can have severe consequences in real-world scenarios. Therefore, data preprocessing is a critical step in the machine learning pipeline and should be given due attention and care.

To make things simpler for our models to read our data, the data preprocessing method on the Elliptic dataset will undergo the following:

1. Merge txId, features, and class into one dataframe.

2. Take useful dataframe as all transactions that are either licit or illicit and do not have unknown transactions.

3. Re-encode licit nodes to be 0 instead of 2.

4. Take X to be features and y to be class.

Step 1 helps make things easier to process for future steps. Step 2 is necessary since we have no use of the unknown transactions in binary classification when we only want to predict licit or illicit transactions. We perform Step 3 because 0 and 1 are easily distinguishable and unambiguous, and also because they have a natural interpretation in terms of Boolean logic (where 0 represents false and 1 represents true). Below is the actual code used for data preprocessing all of our models:

```
    import pandas as pd

    edges = pd.read_csv("elliptic_bitcoin_dataset/elliptic_txs_edgelist.csv")
    features = pd.read_csv("elliptic_bitcoin_dataset/elliptic_txs_features.csv",
                           header = None)
    classes = pd.read_csv("elliptic_bitcoin_dataset/elliptic_txs_classes.csv")

    loc_features = ["loc_feat_" + str(i) for i in range(2,95)]
    agg_features = ["agg_feat_" + str(i) for i in range(1,73)]
    features.columns = ["txId", "time_step"] + loc_features + agg_features
    features = pd.merge(features, classes, left_on="txId", right_on="txId",
how='left')
    features['class'] = features['class'].apply(lambda x : '0' if x == "unknown"
                                                else x)
    features.drop(index=features.index[0], axis=0, inplace=True)

    data = features[(features['class'] == '1') | (features['class'] == '2')]
    X = data[loc_features + agg_features]
    y = data['class']
    y = y.apply(lambda x: 0 if x == '2' else 1)
```

2.2 Binary Classification

Binary classification is a type of machine learning task where the goal is to predict one of two possible outcomes or classes, where the word "binary" refers to the fact that there are only two possible outcomes. In our case, the goal is to predict a transaction either being licit or illicit. Binary classifiers, or binary classification models, are what carry out the task of binary classification. The basic idea behind a binary classifier is to learn a decision boundary that separates

the positive and negative examples in the input space. This decision boundary is often represented as a hyperplane in a high-dimensional space, although other models such as decision trees or support vector machines may also be used. Once the decision boundary has been learned, new inputs can be classified based on which side of the boundary they fall on. Below is a diagram to show the hyperplane separating class points in a 3-dimensional space.

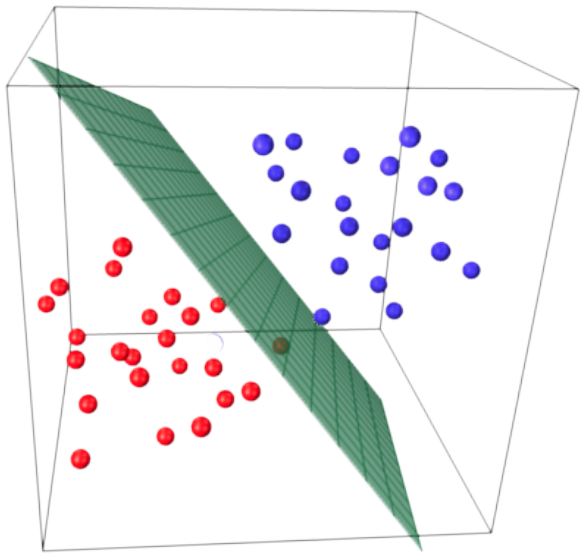

Figure 1: Rahul, Sai. "Hyperplane separating red and blue points within a 3D space." 17 January 2014, http://blog.sairahul.com/2014/01/linear-separability.html, Accessed 25 February 2023

To train a binary classifier, a set of labeled examples is required. These examples consist of input vectors, each of which is associated with a binary label indicating whether it belongs to the positive or negative class. The binary classifier is trained using these labeled examples to learn the decision boundary that separates the two classes. Our Elliptic dataset provides all of these requirements for us to utilize binary classification. The main goal of binary classification is to accurately predict new, unseen data. In other words, once the binary classifier has been trained, it can be used to classify new inputs. To do this, the input is transformed into a feature vector and then passed through the binary classifier to determine which side of the decision boundary it falls

on. If the input falls on the positive side of the decision boundary, it is classified as belonging to the positive class, and if it falls on the negative side, it is classified as belonging to the negative class. Once again, points falling on either side of our decision boundaries will show if the transaction is most likely licit or illicit.

2.2.1 Logistic Regression

The logistic regression model works by estimating the probability of the binary outcome based on a set of input variables. It does this by fitting a logistic function (also known as the sigmoid function) to the data. The sigmoid function is an S-shaped curve that maps any real-valued number to a value between 0 and 1, which can be interpreted as a probability.

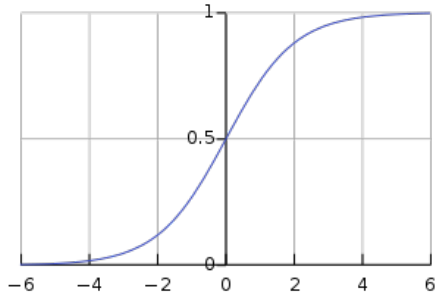

Figure 2: The logistic/sigmoid function $\sigma(x)$

Note that in Figure 2, $\sigma(x) \in (0, 1)$ for all x. The equation for the sigmoid function is:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

where x is the input variable.

In logistic regression, the input variables are combined linearly using weights or coefficients, and the resulting linear combination is transformed using the sigmoid function to obtain the predicted probability of the positive outcome. The equation for the logistic regression model is:

$$P(Y = 1|X) = \sigma(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p) \tag{2}$$

where Y is the binary outcome, X is a vector of input variables, β_0 is the intercept term, $\beta_1, \beta_2, ..., \beta_p$ are the coefficients, and σ is the sigmoid function.

To train the logistic regression model, we use our labeled Elliptic dataset consisting of input variables and their corresponding binary outcomes. The model learns the optimal values of the coefficients by minimizing a cost function, such as the negative log-likelihood or the mean squared error. A regularization term is a penalty term added to the cost function that encourages the model to avoid overfitting the training data. The regularization term is typically a function of the model's parameters, and it is added to the cost function to control the complexity of the model. The most common types of regularization used in binary classification are L1 and L2 regularization. L1 regularization adds the sum of the absolute values of the model's parameters to the cost function, while L2 regularization adds the sum of the squares of the model's parameters. Both types of regularization penalize large parameter values and encourage the model to find simpler solutions that generalize better to new data.

The strength of the regularization term is controlled by a hyperparameter called the regularization strength or regularization parameter. Increasing the regularization strength increases the penalty for large parameter values and reduces the complexity of the model, which can improve generalization performance on new data. However, too much regularization can lead to underfitting, where the model is too simple and fails to capture important patterns in the data. In our case, our cost function that is being minimized is:

$$\min_w C \sum_{i=1}^{\infty} (-y_i \log(\hat{p}(X_i)) - (1 - y_i) \log(1 - \hat{p}(X_i))) + r(w) \tag{3}$$

Below is a breakdown of each variable in this cost function:

- w: the vector of coefficients (weights) to be estimated by the logistic regression model.
- C: a hyperparameter that controls the strength of regularization.

- y_i: the binary target variable (0 or 1) for the i-th training example.
- X_i: the feature vector for the i-th training example.
- $\hat{p}(X_i)$: the predicted probability that the target variable is 1 given the feature vector X_i and the current values of the coefficients w.
- $\log(\hat{p}(X_i))$: the natural logarithm of the predicted probability of the positive class (i.e., $y_i = 1$) for the i-th training example.
- $\log(1 - \hat{p}(X_i))$: the natural logarithm of the predicted probability of the negative class (i.e., $y_i = 0$) for the i-th training example.
- $r(w)$: the regularization term.

The minimization process is typically done using gradient descent or a similar optimization algorithm. For our LR model, we use an optimization algorithm based on gradient descent to minimize the cost function called the limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) algorithm, which is a quasi-Newton method that approximates the inverse Hessian matrix of the cost function using past gradient evaluations.

Once the logistic regression model has been trained, we can use it to predict the probability of the positive outcome for new input data. If the predicted probability is greater than a threshold value (usually 0.5) we classify the outcome as positive; otherwise, we classify it as negative. The top of page 15 includes the code used to implement the logistic regression classifier.

2.2.2 Decision Trees

A decision tree is a commonly used ML algorithm for binary classification that predicts the value of a target variable based on a set of input features. Decision trees have a tree-like structure where each node in the tree represents a decision based on one of the input features, and each leaf node represents a class label or a probability distribution over the class labels. The decision tree algorithm works by recursively splitting the input data into smaller subsets based on the values of

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.3,
                                                    shuffle=False)

model_lr = LogisticRegression(penalty="l2",
                              solver="lbfgs",
                              multi_class="ovr",
                              max_iter=2000).fit(X_train, y_train)
preds_lr = model_lr.predict(X_test)
prec_lr, rec_lr, f1_lr, num_lr = precision_recall_fscore_support(y_test,
                                                                 preds_lr,
                                                                 average = None)
```

the input features. At each node, the algorithm selects the feature that best separates the data into the two classes, and then creates a decision rule based on the threshold value of that feature. The algorithm continues this process until it reaches a stopping criterion, such as a maximum depth of the tree or a minimum number of samples at each leaf node.

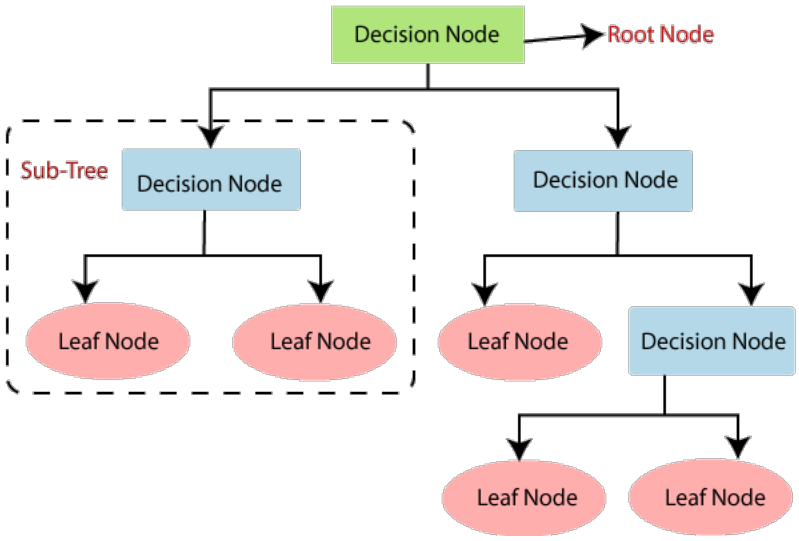

Figure 3: "Decision Tree Classification Algorithm" Javat Point, https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm. Accessed 23 February 2023

To train a decision tree model, it once again requires a labeled dataset consisting of input features and their corresponding class labels (which Elliptic provides). The decision tree algorithm learns the optimal splitting rules by maximizing a criterion that measures the purity of the resulting subsets after each split. The most commonly used criterion and the criterion we will use for our decision tree classifier is the Gini impurity. It measures the probability of misclassifying a random sample from the subset. The Gini impurity can be computed as follows:

$$I_G(p) = 1 - \sum_{i=1}^{c} p_i^2 \tag{4}$$

where p_i is the proportion of samples in class i in the subset, and c is the number of classes.

Once the decision tree model has been trained, we can use it to predict the class label for new input data by traversing the tree from the root node to a leaf node, based on the values of the input features (see Figure 3). At each node, we compare the value of the input feature to the threshold value of the splitting rule, and follow the path corresponding to the true branch if the feature value is greater than or equal to the threshold, and the false branch otherwise. The class label associated with the leaf node reached by the traversal is the predicted class label for the input data. In binary classification, the decision tree algorithm can be used to learn a model that predicts the probability of the positive class, based on the proportion of positive samples in each leaf node. The predicted probability is calculated by normalizing the number of positive samples in the leaf node by the total number of samples in the leaf node. The predicted class label is then obtained by comparing the predicted probability to a threshold value, usually 0.5. Below is the code used to implement the logistic regression classifier:

2.2.3 Multilayer Perceptron

A multilayer perceptron (MLP) is a type of artificial neural network that consists of multiple layers of interconnected nodes (or neurons) that are organized into an input layer, one or more hidden layer, and an output layer. The input layer receives the input features, and the output layer produces the predicted class label or probability distribution over the class labels. The neurons in

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.3,
                                                    shuffle=False)

model_dt = LogisticRegression(criterion="gini",
                              splitter="best",
                              ).fit(X_train, y_train)
preds_dt = model_dt.predict(X_test)
prec_dt, rec_dt, f1_dt, num_dt = precision_recall_fscore_support(y_test,
                                                                 preds_dt,
                                                                 average = None)
```

the MLP are organized into layers and each neuron in any layer is connected to every neuron in the next layer. Each neuron receives inputs from the previous layer, performs a weighted sum of the inputs, and applies an activation function to produce the output. The output of each neuron is then propagated forward to the next layer as inputs to the neurons in that layer.

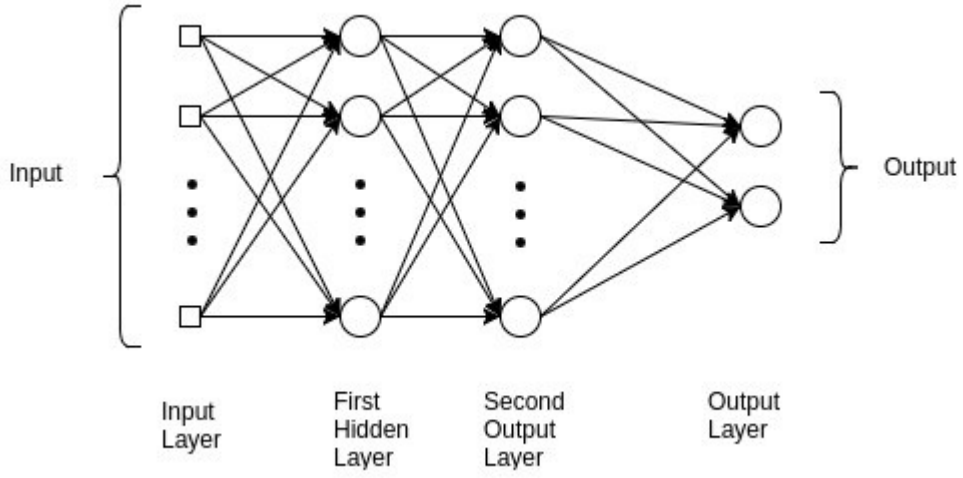

Figure 4: Structure of Multilayer Perceptron (MLP) model with 2 hidden layers

To train an MLP for binary classification, the model learns the optimal weights and biases

for the neurons by minimizing a cost function that measures the difference between the predicted class label and the true class label. These "optimal weights and biases" are performed by something called an "optimizer", which is nothing more than a function that computes the gradient of the cost function with respect to the weights and biases and adjusts them in the direction of the negative gradient. For our MLP classifier, we use something called the Adaptive Moment Estimation (Adam) optimizer. I went with this because Adam has shown to perform well on large and particularly sparse data. The cost function we use for our MLP classifier is a variant of the cross-entropy loss function called the binary cross-entropy loss function, defined as follows:

$$L(y, t) = -t \log(y) - (1 - t) \log(1 - y) \tag{5}$$

where y is the predicted probability of the positive class and t is the true label (which is either 0 or 1). The optimization of the cost function is done through the backpropagation algorithm, which computes the gradient of the cost function with respect to the weights and biases of the neurons. The gradient is then used to update the weights and biases in the opposite direction of the gradient, with the aim of minimizing the cost function.

The activation function used in the neurons is an important component of the MLP, as it determines the nonlinearity of the model and allows it to capture complex nonlinear relationships between the input features and the target variable. The activation we use for our MLP classifier is known as rectified linear unit (ReLU). ReLU is most commonly used in deep learning because it is simple, efficient to compute, and has been shown to work well in practice. Below is the equation of this activation function:

$$\mathrm{ReLU}(x) = \max(0, x) \tag{6}$$

The number of hidden layers and the number of neurons in each layer are hyperparameters of the MLP that need to be tuned during the training process. While a larger number of hidden layers and neurons can increase the capacity of the model and allow it to capture more complex

patterns in the data, it can consequently increase the risk of overfitting and require longer training times. We have set the number of hidden layers for our MLP classifier to be 100.

The output layer of the MLP consists of a single neuron with a sigmoid activation function, which produces a predicted probability of the positive class. The predicted class label is then obtained by comparing the predicted probability to a threshold value, usually 0.5. Below is the final code used to implement our MLP classifier:

```
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.3,
                                                    shuffle=False)

model_mlp = MLPClassifier(hidden_layer_sizes=100,
                          activation='relu',
                          solver='adam').fit(X_train, y_train)
preds_mlp = model_mlp.predict(X_test)
prec_mlp, rec_mlp, f1_mlp, num_mlp = precision_recall_fscore_support(y_test,
                                                                     preds_mlp,
                                                                     average = None)
```

2.2.4 Random Forest

Random Forest is an ensemble learning algorithm where the main idea is to create a "forest" of decision trees. Each decision tree is built using a random subset of the data and a random subset of the features and each tree in the forest outputs a class label and the final output of the random forest is determined by aggregating the outputs of all the decision trees in the forest. The randomness in the algorithm helps to reduce overfitting and improve the accuracy of the model.

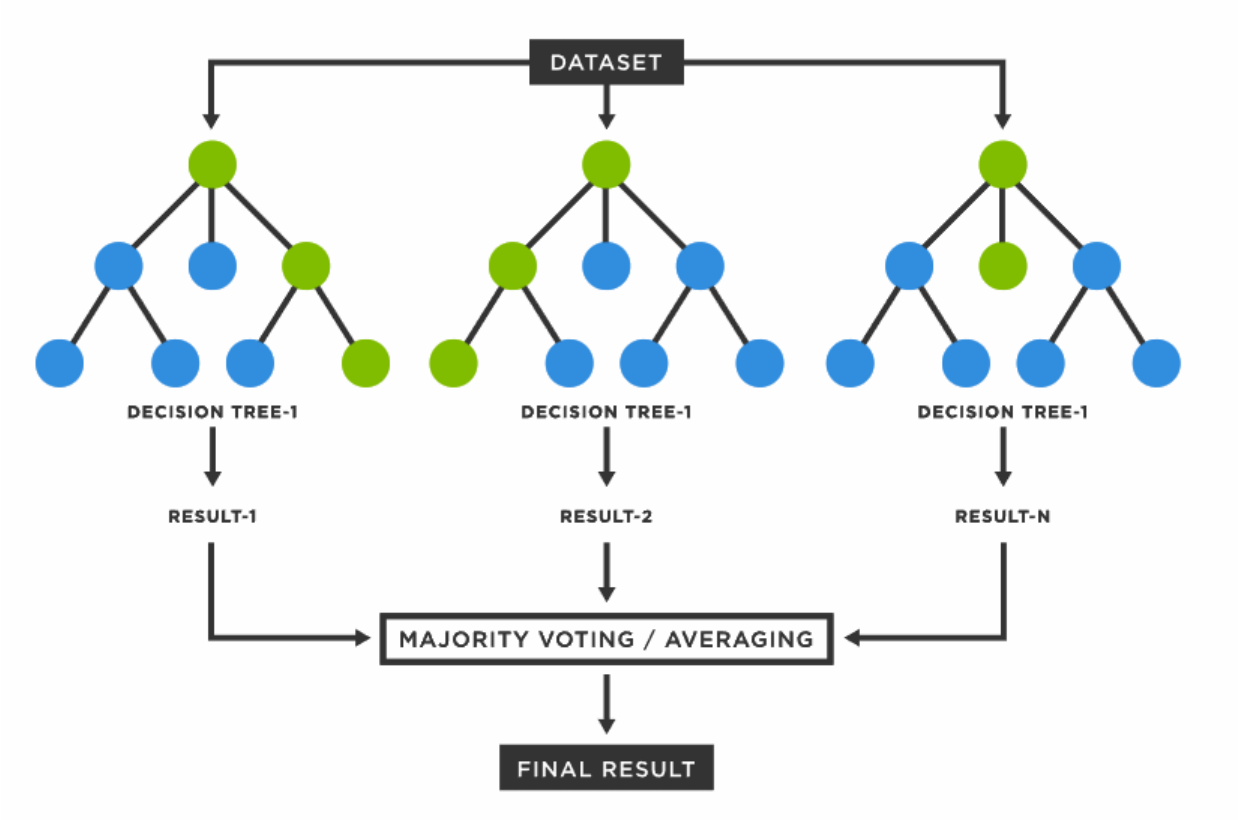

Figure 5: "Flow diagram of Random Forest" TIBCO, https://www.tibco.com/reference-center/what-is-a-random-forest, Accessed 25 February 2023

Below are the steps involved in using the random forest algorithm for binary classification:

1. Data Preparation: The first step is to prepare the data by splitting it into training and testing sets. The training set is used to build the random forest model, while the testing set is used to evaluate the performance of the model.

2. Random Subset of Features: At each node of each decision tree, a random subset of features is selected to split the data. This helps to reduce the correlation between the trees and improve the accuracy of the model.

3. Random Subset of Data: A random subset of the training data is selected for each decision tree. This helps to reduce overfitting and improve the generalization of the model.

4. Decision Tree Construction: A decision tree is constructed for each random subset of data and features. The decision tree is built by recursively splitting the data based on the selected features until the tree reaches a stopping criterion (e.g. maximum depth of the tree or minimum number of samples per leaf).

5. Aggregation of Decision Trees: The final step is to aggregate the outputs of all the decision trees in the forest. The most common method for aggregating the outputs is to use a majority vote, with the class label receiving the most votes being selected as the final output of the random forest.

The number of decision trees in a random forest is an important hyperparameter that can affect the performance of the algorithm, where an increase the number of decision trees can typically lead to a better-performing model up to a certain point where further increasing the number of trees does not improve the performance. Bias/variance, computation time, and randomness are factors that are considered when deciding the amount of trees in a random forest. For our RF classifier, we use a standard 100 decision trees. And as previously mentioned in the section 2.2.2, the Gini impurity criterion will also be used in the decision trees of our RF classifier. Below is the code used for implementing our RF classifier:

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.3,
                                                    shuffle=False)

model_rf = RandomForestClassifier(n_estimators=100,
                                  criterion='gini').fit(X_train, y_train)
preds_rf = model_rf.predict(X_test)
prec_rf, rec_rf, f1_rf, num_rf = precision_recall_fscore_support(y_test,
                                                                 preds_rf,
                                                                 average = None)
```

2.2.5 Graph Attention Network

Graph Attention Networks (GATs) are a type of neural network architecture designed to process graph-structured data. GATs use an attention mechanism to compute a weighted sum of node features from a graph neighborhood, with the weights computed based on the similarity between the features of the central node and those of its neighbors. The attention mechanism allows the network to focus on the most relevant neighbors for each node in the graph, thus enabling it to learn complex relationships between nodes.

The key idea behind GATs is to compute a node representation by taking into account the representations of its neighbors, weighted by an attention coefficient. The attention coefficient measures the similarity between the central node and each of its neighbors, and is computed using a shared learnable function that takes as input the features of the central node and the neighbor node. This function is typically implemented as a multi-layer perceptron (MLP). More formally, given a graph G = (V, E) with node feature matrix X, the output of a GAT layer with K attention heads can be computed as follows:

$$h_i^k = \sigma\left(\sum_{j \in N_i} \alpha_{ij}^k W^k x_j\right) \tag{7}$$

where h_i^k is the output representation of node i in the k-th attention head, N_i is the set of neighbors of node i, σ is a non-linearity function (e.g. ReLU), W^k is a learnable weight matrix for the k-th attention head, and α_{ij}^k is the attention coefficient between node i and its neighbor j in the k-th attention head, computed as follows:

$$\alpha_{ij}^k = \frac{\exp(\text{LeakyReLU}(a^k[Wx_i|Wx_j]))}{\sum_{l \in N_i} \exp(\text{LeakyReLU}(a^k[Wx_i|Wx_l]))} \tag{8}$$

where $\text{LeakyReLU}(x) = \max(x, \alpha x)$ is a variant of the ReLU activation function with a small slope for negative values and a^k and W are learnable parameters. The attention coefficients are normalized using the softmax function over the neighbors of node i, so that they sum up to 1.

The final output of the GAT layer is obtained by concatenating the outputs of the K attention heads:

$$h_i = \text{concat}([h_i^1, h_i^2, \ldots, h_i^K]) \tag{9}$$

GATs can be stacked to form a deep neural network, with each GAT layer processing the node features in a different way based on the attention mechanism. The resulting representation of the graph can be used for various downstream tasks, such as node classification or link prediction.

The GAT model implements multi-head graph attention layers. The $MultiHeadGraphAttention$ layer is simply a concatenation of multiple graph attention layers ($GraphAttention$), each with separate learnable weights W. For the $GraphAttention$ layer, consider inputs node states h^l which are linearly transformed by W^l, resulting in z^l. For each target node:

1. Compute pair-wise attention scores $\mathbf{a}^{(l)\top}([\mathbf{z}^{(l)}i||\mathbf{z}^{(l)}j])$ for all j, resulting in e_{ij} (for all j). $||$ denotes a concatenation, i corresponds to the target node, and j corresponds to a given 1-hop neighbor/source node.

2. Normalize e_{ij} via softmax, so that the sum of incoming edges' attention scores to the target node ($\sum_k e_{\text{norm},ik}$) will add up to 1.

3. Apply attention scores $e_{\text{norm},ij}$ to $\mathbf{z}j$ and add it to the new target node state $\mathbf{h}^{(l+1)}i$, for all j.

The complete implementation of my Graph Neural Network can be found in the GitHub repository at [https://github.com/jacksebastian17/graph-attention-network].

2.3 Evaluation Metrics

Before delving into the results of our study, it is crucial to discuss the criteria for evaluating the performance of the binary classification models used for both our own knowledge and understanding of the results presented as well as applicability the real-world scenario of illicit predictions within AML. We will use four key performance metrics: precision, recall, F1-score, and micro-average F1-score.

Precision measures the ratio of true positive predictions (i.e., correct predictions of illicit transactions) to the total number of positive predictions made by a model. A high precision value

indicates that the model can accurately identify illicit transactions with few false positives (i.e., misclassifying legitimate transactions as illicit). This metric is particularly useful when the cost of false positives is high, as it helps minimize the risk of wrongly flagging innocent transactions.

Recall, on the other hand, measures the ratio of true positive predictions to the total number of actual positive instances (i.e., all illicit transactions) in the dataset. A high recall value indicates that the model can correctly identify a large proportion of the illicit transactions, even if it may also generate some false positives. This metric is crucial when it is essential to detect as many illicit transactions as possible (i.e. AML efforts).

The F1-score is a harmonic mean of precision and recall, providing a balanced assessment of both metrics. It ranges from 0 to 1, with a higher value indicating better performance. The F1-score is particularly useful when there is an uneven class distribution (as is often the case in detecting illicit transactions), as it prevents the model from being biased towards the majority class.

Lastly, the micro-average F1-score is a performance metric that aggregates the contributions of all classes to compute the overall model performance. It is calculated by considering the total number of true positives, false positives, and false negatives across all classes. This metric is particularly important to us for the following reasons:

1. Imbalanced dataset: In the context of illicit transactions detection, the dataset is typically imbalanced, with the majority of transactions being licit and a smaller proportion being illicit. In such cases, the micro-average F1-score is advantageous as it takes into account the total number of true positives, false positives, and false negatives across all classes, giving equal importance to each prediction. This prevents the evaluation metric from being biased towards the majority class (licit transactions) and ensures that the performance of the model on the minority class (illicit transactions) is also considered.

2. Focus on minimizing false negatives: In illicit transaction detection, it is often crucial to minimize false negatives (i.e., failing to identify actual illicit transactions). Since the micro-average F1-score considers the total number of false negatives in its calculation, it emphasizes the model's ability to correctly identify illicit transactions, which is essential for effec-

tive anti-money laundering efforts.

3. Robustness to class distribution changes: In the rapidly evolving cryptocurrency landscape, the distribution of illicit transactions may change over time. The micro-average F1-score is less sensitive to changes in class distributions, making it a more reliable performance metric when dealing with a dynamic environment.

Therefore, the micro-average F1-score is going to be the more emphasized and useful evaluation metric for us when analyzing the results of running our models.

3. RESULTS

Here we discuss the results of the models:

Table 1: Illicit Classification Results

Method	Precision	Recall	F1	Micro-F1
Logistic Regression	0.499	0.632	0.558	0.936
Decision Tree	0.599	0.670	0.609	0.945
MLP	0.821	0.598	0.692	0.966
Random Forest	0.986	0.650	0.784	0.977
GAT	0.967	0.994	0.980	0.967

The Graph Attention Network (GAT) model exhibited the highest performance across most evaluation metrics, achieving a Precision of 0.967, Recall of 0.994, F1 score of 0.980, and Micro-F1 score of 0.967. This superior performance can be attributed to GAT's ability to leverage both local and global information from the underlying transaction graph structure, making it particularly suited for this task.

In comparison, the Random Forest model also performed well, with a Precision of 0.986 and Micro-F1 score of 0.977 (higher than that of GAT). This is most likely a result of RF using a voting mechanism to "ensemble" the prediction results from a number of decision trees, each trained by using a subsample of the data set. So to no surprise, the single Decision Tree classifier did not perform nearly as well as RF. However, RF's Recall of 0.650 and F1 score of 0.784 were lower than those of GAT, indicating that it might miss some true positives despite its high Precision.

The MLP model achieved a Precision of 0.821, Recall of 0.598, F1 score of 0.692, and Micro-F1 score of 0.966. The relatively lower Recall suggests that the model might be more conservative in labeling transactions as illicit, potentially missing some true positives.

The Decision Tree model displayed a moderate performance, with a Precision of 0.599, Recall of 0.670, F1 score of 0.609, and Micro-F1 score of 0.945. The decision tree's ability to

capture non-linear relationships between features and the target variable can be a result of this. However, the single tree's tendency to overfit may have limited its generalization to the test data.

Finally, the Logistic Regression model exhibited the lowest performance among the evaluated models, with a Precision of 0.499, Recall of 0.632, F1 score of 0.558, and Micro-F1 score of 0.936. The relatively low scores comes from the model's limited ability to capture complex relationships (i.e. a graph network) between features and the target variable given its linear nature.

4. CONCLUSION

The results suggest that Graph Attention Networks are the most effective approach for predicting illicit Bitcoin transactions on the blockchain among the models evaluated in this study. The GAT model's ability to leverage the graph structure of blockchain transactions and capture both local and global information has led to its superior performance across all evaluation metrics.

These findings suggest that GATs may be a more effective tool for identifying and preventing criminal activity on the blockchain. Moreover, your research contributes to the growing field of cryptocurrency forensics and highlights the importance of developing new methods to fight criminal activity in financial systems. Through the use of labelled transaction data sets and advanced algorithms, this research offers a prototype for visualization of such data and models that can aid in augmenting human analysis and explainability. Overall, this work inspires others to tackle the critical societal challenge of making our financial systems safer and more inclusive.

While the results show that GATs seemed to outperform the other binary classification models, it is important to ask: What are the limitations of these approaches and how can these limitations be addressed? Some binary classification models, such as neural networks, may lack interpretability, which can make it difficult to explain how the model arrives at its predictions. Also, there exists a limited scope where only a focus on specific aspects of illicit transactions on the Bitcoin blockchain may not capture the full range of criminal activities that occur in cryptocurrency ecosystems.

Another important concluding question: What are the ethical implications of using machine learning algorithms to detect illicit transactions, and how can these implications be addressed? The ethical implications of using ML algorithms to detect illicit activity on the Bitcoin blockchain are complex and multifaceted. One potential concern is that the use of these algorithms could result in false positives or negatives, leading to wrongful accusations or missed criminal activity. Additionally, the use of these algorithms may raise privacy concerns, as transaction data on the blockchain

can reveal sensitive information about individuals and their financial activities. Another ethical concern is the potential for algorithmic bias, whereby the ML algorithms may perpetuate and amplify existing social and economic inequalities. For example, the algorithms may disproportionately target certain demographics or geographic regions, leading to unequal treatment and potential harm to marginalized communities.

Further research is needed to determine the generalizability of these findings to other cryptocurrencies and to inform the development of more effective regulations and policies to combat financial crimes in cryptocurrency ecosystems. Overall, this research inspires further reflection on the critical societal challenge of making financial systems safer and more inclusive.

REFERENCES

[1] M. Rauchs and G. Hileman, *Global Cryptocurrency Benchmarking Study*. No. 201704-gcbs in Cambridge Centre for Alternative Finance Reports, Cambridge Centre for Alternative Finance, Cambridge Judge Business School, University of Cambridge, 2017.

[2] "Bitcoin (btc) price, charts, and news | coinbase: Bitcoin price ...."

[3] V. A. Kumar, "Money laundering: Concept, significance and its impact," *Money Laundering: Concept, Significance and its Impact*, vol. 4, Jan 2012.

[4] "Tax abuse, money laundering and corruption plague global finance | un desa department of economic and social affairs," Sep 2020.

[5] U. Government, "Fincen issues proposed rule for beneficial ownership reporting to counter illicit finance and increase transparency," Dec 2021.

[6] Chainalysis, "2020 geography of cryptocurrency report." https://go.chainalysis.com/rs/503-FAP-074/images/Chainalysis_2020_Geography_of_Cryptocurrency_Report.pdf, 2020.

[7] London, "Blockchain analytics amp; crypto compliance solutions."