Enhancing Fraudulent Account Detection on the Ethereum Blockchain: A Novel Feature Selection Approach and Comparative Analysis of Machine Learning Algorithms

by

BORZOUYEH NOROUZI

A thesis submitted to the
Department of Computer Science
in conformity with the requirements for
the degree of Master of Science

Bishop's University
Canada
August 2023

Copyright © BORZOUYEH NOROUZI, 2023

Abstract

Blockchain technology has recently introduced disruption across numerous sectors, with cryptocurrencies like Ethereum's Ether gaining prominence in finance. However, the decentralized nature of Ethereum's platform, which facilitates smart contracts, has also attracted criminals seeking to exploit its versatility. In this thesis, we focus on identifying illicit accounts on the Ethereum blockchain network, where scammers and fraudsters engage in various illegal activities, such as Ponzi schemes, Money Laundering, and Phishing. Utilizing supervised machine learning methods, we differentiate between accounts with reported criminal activities and normal accounts. Our proposed Random Forest classifier achieves the highest average AUC score of 0.998 and an accuracy of 0.979. We employ a novel feature selection approach that combines Recursive Feature Elimination with Cross-validation (RFECV) and leverages Random Forest built-in features importance (RF-FI) to identify six essential features with the critical impact on the models output. These essential features include 'Average minutes between received transactions,' 'Time Difference between the first and last transaction (Minutes),' 'Minimum value of received ERC20 transactions,' 'Encoded type of received ERC20 tokens,' 'Number of sent transactions,' and 'Total number of ERC20 transactions.' Our methodology successfully identifies illicit accounts and provides a valuable and effective framework for identifying and prioritizing the key factors contributing to fraudulent activities on the Ethereum Blockchain network, with potential applications in other cryptocurrencies using the Blockchain protocol.

Keywords: Ethereum, Blockchain, Fraud detection, Supervised classifiers, Feature selection.

Acknowledgments

I want to express my sincere gratitude to my supervisors, Dr. Mohammed Ayoub Alaoui Mhamdi and Dr. Madjid Allili, for their invaluable guidance, support, and advice throughout my time at Bishop's University. Their mentorship has significantly shaped my academic and professional development.

I am deeply grateful to my friend Mehran Attar, a Ph.D. candidate at Concordia University, for his encouragement and inspiration, which have motivated me to strive for excellence and push my limits in my academic pursuits.

I extend my heartfelt thanks to my parents and sister for their unwavering support, kindness, and motivation throughout my journey.

Lastly, I deeply appreciate my wife, Marzieh, for her unwavering love and support. Her presence has been a constant source of strength, helping me overcome challenges during my academic journey. I feel truly blessed to have her by my side.

Contents

1 INTRODUCTION **1**
1.1 Introduction . . . 1
1.1.1 Problem Definition . . . 2
1.2 Motivation . . . 2
1.2.1 Why Ethereum? . . . 2
1.2.2 Why Anomaly Detection? . . . 3
1.2.3 Who Are The Target Audiences? . . . 3
1.3 Aims and Objectives . . . 4
1.4 Literature Review . . . 4

2 BACKGROUND AND PRELIMINARIES **7**
2.1 Background . . . 7
2.1.1 Blockchain and Decentralized Cryptocurrencies . . . 7
2.1.2 Ethereum . . . 8
2.2 Classification Models . . . 10
2.2.1 NuSVC . . . 10
2.2.2 Logistic Regression . . . 12
2.2.3 Random Forest (RF) . . . 14
2.2.4 Neural network Multilayer Perception (MLP) . . . 15
2.3 Feature Selection Methods . . . 16
2.3.1 Recursive Feature Elimination (RFE) . . . 17
2.3.2 Random Forest built-in Feature Importance (FI) . . . 18

3 METHODOLOGY **20**
3.1 Data visualization . . . 20
3.1.1 Description of the Dataset and Target Variable . . . 20
3.2 Data Pre-processing . . . 21
3.2.1 LabelEncoder . . . 21
3.2.2 Identifying & Handling missing values . . . 21
3.3 Models Implementation . . . 24
3.3.1 Hyperparameter Tuning . . . 24
3.4 Performance Measures . . . 25
3.5 Feature Importance . . . 27

4 EXPERIMENTAL RESULTS **28**
4.1 General dataset information 28
4.2 t-SNE Visualization 29
4.3 Classifiers Results 31
4.3.1 NuSVC 31
4.3.2 Logistic Regression 32
4.3.3 Random Forest 34
4.3.4 Multi-Layer perceptron (MLP) 36
4.4 Comparison of Our Proposed Models 37
4.5 Feature Importance 38
4.5.1 Retrain Models with New Feature Sets 42
4.6 Comparison with Other Works 44

5 CONCLUSION AND FUTURE WORKS **48**

Bibliography **50**

List of Tables

3.1 Full set of 42 features of the original dataset. 22

4.1 Results for 3-, 4-, 5- and 10-fold cross-validation of the NuSVC classifier 32
4.2 Results for 3-, 4-, 5- and 10-fold cross-validation of the Logistic Regression classifier 33
4.3 Results for 3-, 4-, 5- and 10-fold cross-validation of the Random Forest classifier. . . 35
4.4 Results for 3-, 4-, 5- and 10-fold cross-validation of the MLP classifier. 37
4.5 comparison of our four proposed ML models by their highest AUC score 38
4.6 Feature selection by RFECV method using 10-fold cross-validation. 39
4.7 List of important features selected by RF-FI method with a threshold value of ”0.015”. 41
4.8 List of 16 important features. 43
4.9 List of 6 important features 44
4.10 Comparison of the RF classifier results with the original 42 features and two new sets of features 44
4.11 Comparison of the MLP classifier results with the original 42 features and two new sets of features 44
4.12 Comparison of our four proposed ML models by the XGBoost model proposed by Farrugia et al. [6] 45

List of Figures

2.1 Schematic visualization of a chain of Blocks[2] 8
2.2 General SVM model for binary classification. 10
2.3 Logistic Regression VS Linear Regression. 13
2.4 An example of a Random Forest classifier's visual representation [55]. 14
2.5 Schematic of Neural Network MLP classifier. 15
2.6 A general framework of feature selection for classification [64]. 16

3.1 Visualizing missing values in the dataset 23
3.2 Confusion Matrix for binary classification [77]. 26

4.1 Visualization of the correlation between features. 29
4.2 2D scatter plot t-SNE projection regarding account's class 30
4.3 3D scatter plot t-SNE projection regarding account's class. 30
4.4 NuSVC hyperparameter tuning heatmap 31
4.5 Average 10-fold cross validation confusion matrix of NuSVC 32
4.6 Box plot illustrating the relationship between the hyperparameter 'C' and the corresponding F1 scores. 33
4.7 Average 10-fold cross validation confusion matrix of logistic regression 34
4.8 Grid-Search 10-fold cross-validation evaluation of n_estimators and max_depth parameters. 34
4.9 Average 10-fold cross-validation confusion matrix of RF model. 36
4.10 Average 10-fold cross-validation confusion matrix of MLP model. 37
4.11 Visualization of the proposed model's performance 38
4.12 Relationship between the number of features and the mean accuracy using RFECV. . 40
4.13 Rank of the most important features determined by RF-FI. 40
4.14 Ranking of Important Features using RFECV Method. 41
4.15 The correlation heatmap of 21 features extracted from the two feature selection methods. 42
4.16 The correlation heatmap of 6 features extracted from the two feature selection methods. 43
4.17 Performance comparison of the proposed models to the ones conducted by Aziz et al. [44] 46
4.18 Comparison of the accuracy score obtained by our proposed models with the ones implemented by Quadir et al. [30] 47

Chapter 1

INTRODUCTION

1.1 Introduction

Blockchain is a decentralized, digital ledger that allows for secure and transparent record-keeping [1]. It has found its most prominent use case in cryptocurrencies, which are digital assets secured by cryptography and traded on decentralized networks. With the creation of Bitcoin in 2009, blockchain technology gained significant attention, and its popularity has only grown since then [2]. In 2021, the total transaction volume across all cryptocurrencies reached 15.8 trillion dollars, up 567% from the previous year, according to Chainalysis [3]. The Ethereum blockchain network, launched in 2015, introduced the concept of smart contracts. These contracts enable trustworthy transactions to be carried out without the need for intermediaries, and their outcomes are recorded on the blockchain. As a result, Ethereum has become the largest blockchain network that supports smart contracts, and its associated cryptocurrency, Ether, is the second-largest cryptocurrency in the market [4]. However, the decentralized nature of blockchain networks also attracts illegal activities, such as smart-Ponzi schemes, phishing, scamming, money laundering, fraud, or any other malicious actions that violate the rules and regulations governing the Ethereum ecosystem [5]. To combat such illicit activities, machine learning methods can be effectively employed through anomaly detection techniques.

Illicit account detection using machine learning techniques in a decentralized blockchain like Ethereum holds significant value in enhancing the platform's security and trustworthiness. "illicit accounts" in the context of the Ethereum blockchain refers to user accounts that engage in suspicious or unauthorized activities discussed earlier that deviate from the blockchain network's standard and legitimate usage patterns [6]. Illicit accounts are a concern for the security and integrity of the blockchain as they can pose risks to other users, undermine trust in the network, and potentially lead to financial losses or reputational damage. The platform can proactively identify suspicious or illicit accounts based on patterns, anomalies, and historical data by employing machine learning algorithms, thereby mitigating potential risks and protecting legitimate users from fraudulent activities. Such detection mechanisms can contribute to maintaining the integrity of the Ethereum network, fostering a safer environment for users, and preserving the overall reputation and credibility of the blockchain platform [7].

In this thesis, we employ machine learning techniques for detecting fraudulent accounts associated with a number of fraudulent transactions in the Ethereum blockchain network. Specifically, we utilize supervised learning, a commonly used approach in anomaly detection, aiming to classify test data as abnormal or normal using labeled data [8]. We employ a newly published dataset by Farrugia et al. [6], which has yet to be broadly explored in the literature. We implemented four

classification algorithms, of which three, namely NuSVC, Logistic Regression, and Multilayer Perceptron, were implemented on this dataset for the first time. Furthermore, we developed two new feature sets by integrating the outcomes of Recursive Feature Elimination with cross-validation and Random Forest built-in Feature Importance methods, prioritizing the essential information in the dataset while reducing the total number of features. We utilize the pipeline tool in this thesis to integrate dataset imputation, normalization, hyperparameter tuning, and model training using grid search cross-validation. Using pipelines made our research results more consistent and transparent, contributing to the broader research community's ability to utilize our findings.

Chapter 2 provides contextual information on blockchain technology, Ethereum, and proposed methods. Chapter 3 discusses the proposed approach to detecting fraudulent accounts in the Ethereum blockchain network. In Chapter 4, we present and analyze the results of each classification model before and after feature selection and compare our findings with the results of existing works. Besides that, To ensure the completeness and accuracy of our thesis, we sought to enhance our findings by incorporating a more extensive dataset. This enabled us to better compare our results against other related works in the field. Adding a larger dataset allowed us to broaden the scope of our research and evaluate our approaches' generalizability. Finally, Chapter 5 concludes the thesis and outlines future research directions.

1.1.1 Problem Definition

1.2 Motivation

1.2.1 Why Ethereum?

As the second-largest cryptocurrency by market capitalization, Ethereum's smart contract functionality allows developers to create decentralized applications (DApps) and platforms that can run without downtime or interference from a third party [9]. Additionally, Ethereum's growing community and ecosystem have resulted in various use cases, and applications researchers can study.

One of the significant advantages of Ethereum is its ability to handle a high volume of transactions. According to data from Etherscan.io [10] The Ethereum daily transaction volume since 2021 has been around 740,000 to 1.8 million, making it one of the most active cryptocurrencies in daily transactions. Flexibility is another advantage that makes it possible to develop specialized and varied decentralized applications. Moreover, Ethereum's use of the proof-of-stake (POS) consensus algorithm, where validators are chosen to create new blocks based on the number of coins they hold, means it consumes far less energy than Bitcoin, which uses the energy-intensive proof-of-work (POW) algorithm, where miners solve complex mathematical puzzles to validate transactions.

The Ethereum network's ability to facilitate the creation of decentralized applications makes it an excellent platform for researching and developing new innovative solutions. Researchers can use Ethereum's vast capabilities to study various areas, such as blockchain governance, scalability, security, and smart contract design. Ethereum's potential extends beyond cryptocurrencies, as it has been used in various industries such as finance, healthcare, IoT, and gaming, to name a few [11].

Interestingly, while there are many edges to this, there are also some drawbacks, particularly security-related ones [12]. Hence, scalability, flexibility, growing ecosystem, and broad applicability make it a selected platform for analysis.

1.2.2 Why Anomaly Detection?

Anomaly detection has long been a subject of research. Anomalies are anomalous or implausible happenings. Thieves and criminal activity are frequently atypical in character in any financial network. Anomalies should be found as soon as feasible in order to prevent damage to the integrity and community of the network [13]. Banks have made significant financial investments in the traditional finance system to identify suspicious transactions across their network, as have the governments, businesses, and organizations controlling and supervising funds transfers. These transactions could be a part of fraud, bribery, money laundering, or other criminal techniques.

As of modern digital financial network, similar activity is also present over the Ethereum as a decentralized network. Entities and users are utilizing the anonymity provided by Ethereum for illicit activities. The ”illicit activities” or behaviors observed on the blockchain include scams and smart-Ponzi schemes, phishing, money laundering, ransomware and extortion, and other crime and fraud actions. Although blockchain technology's internal security mechanisms, like hash functions, can solve issues like double spending, there are currently no tools to automatically detect or flag suspicious behavior or transactions on the network. To address this, techniques like anomaly detection, commonly used in banks for spotting unusual transactions, can be applied to blockchain networks. By analyzing the publicly available distributed ledgers of cryptocurrencies, anomalies can be identified and potential illicit activities detected [14].

Since the publicly available Ethereum blockchain ledger qualifies as big data. Hence, manually identifying anomaly characteristics within this vast amount of data would be impractical and never-ending. To handle the high volume of transactions and smart contracts, machine learning algorithms are necessary for efficient analysis. This approach can help identify past transactions that exhibit similar behavior and classify new transactions with similar features. To this end, features are used as identifiers and are determined by analyzing available transaction data and extracting relevant information. The machine learning algorithms can distinguish between normal and anomalous behavior in user accounts by learning the corresponding features that relate to either normal or anomalous behavior.

1.2.3 Who Are The Target Audiences?

The probable stakeholders of illicit account detection on the Ethereum blockchain using machine learning techniques include:

1. **Ethereum Community:** This includes the Ethereum Foundation, developers, and smart contract auditors invested in maintaining the network's security and integrity.

2. **Regulatory Bodies:** Government entities and regulatory authorities overseeing blockchain and cryptocurrency activities are interested in preventing illicit activities from complying with regulations.

3. **Cryptocurrency Industry:** Cryptocurrency exchanges, wallet providers, financial institutions, and businesses benefit from secure and trustworthy networks to mitigate risks and comply with AML and KYC regulations.

4. **Investors and Users:** Investors and token holders are interested in the network's stability and security, while users of DApps seek increased security against fraudulent activities.

1.3 Aims and Objectives

This thesis aims to develop a methodology that can accurately distinguish between normal and illicit accounts based on their transaction history on the Ethereum blockchain. We also aim to find the most informative features within our dataset in order to optimize execution time without compromising the accuracy of the classification models.

To address above mentioned aims, the following objectives have been defined:

1. Preprocessing and preparing an appropriate dataset to train and test the supervised learning models.

2. Modifying and optimizing NuSVC, Logistic Regression, Random Forest, and Multi-Layer Perceptron classification algorithms using GridSearchCV.

3. Thorough evaluation of the proposed models, avoid overfitting using 3-, 4-, 5-, and 10-fold cross-validation along with various performance metrics such as accuracy, F1 score, and AUC.

4. Extensive feature selection by implementing Recursive Feature Elimination with cross-validation (RFECV) and Random Forest built-in Feature Importance methods to gain an in-depth insight into essential features.

5. A thorough analysis of four classifiers and compare the results with similar works in the field.

1.4 Literature Review

According to the Council on Foreign Relations, Bitcoin is by far the most prominent cryptocurrency, and its market capitalization has peaked at over $1 trillion. Numerous others, including Ethereum, the second-most popular, have proliferated in recent years [15]. The growth in digital currencies could make cross-border payments more efficient and help address the $1.7 trillion global trade financing gap [16]. Nevertheless, as the volume of transactions rises, fraud accounts also arise, resulting in significant losses for the normal accounts involved [17]. By the end of 2020, blockchain became more popular again, and several new studies were conducted regarding anomaly or illicit account/address detection [18].

In 2021, the usage of cryptocurrencies is expanding more quickly than ever. According to 2022 and 2023 Crime Report by the Chainalysis, The entire transaction volume increased to $15.8 trillion, up 567% from the totals for 2020. Criminal activity involving cryptocurrencies also peaked in 2021, with criminal addresses collecting $14 billion during the year, up from $7.8 billion in 2020. Despite the market slump, the volume of illicit transactions increased for a second year, reaching a record high of $20.6 billion at a lower bound estimate. In 2022, illegal activity increased to $18 billion, primarily due to the uncovering of new cryptocurrency frauds [19] [20]. The volume of transactions and the cash value of fraud in cryptocurrencies demonstrate the significance of fraud detection.

Cai et al. [7], highlighted fraudulent activities that were both objective and subjective. The authors come to the conclusion that while subjective fraud cannot be detected by blockchain, it can be mitigated by machine learning. The forms of fraud that blockchain can identify and the ones that blockchain are still susceptible to also have been explored in [1], which opened the door to thoughts on the issues that a machine learning component has to take into account. Machine learning has emerged during the last decade as one of the most effective technologies for identifying financial fraud. Due to their discoveries, researchers began using machine learning algorithms to solve cryptocurrency fraud detection problems [21].

Ostapowicz et al. [22], employed approaches for supervised machine learning to identify fraudulent activity in the Ethereum blockchain. They concentrated on the capability of criminals to steal money by using well-known malware or phony emails. They, therefore, employed a dataset of more than 300,000 accounts to detect such accounts using the capabilities of Random Forests, Support Vector Machines, and XGBoost classifiers.

Steven Farrugia et al. [6] provides a substantial Ethereum dataset consisting of 2179 accounts flagged by the Ethereum community for their illegal activity based on their transactions coupled with 2502 normal accounts. We utilized their comprehensive dataset in this thesis. The author then used the XGBoost classifier plus 10-Fold cross-validation to train the model. The results showed an accuracy of 96.3%. The authors also noted three features that significantly impacted their model's output for detecting illicit accounts: 'Time difference between first and last (Mins),' 'Total Ether balance,' and 'Min value received'. We compared our results with their work in this thesis.

Unsupervised learning algorithms are also employed for detecting fraud in blockchain. Monamo et al. [23], specifically focused on using trimmed k-means for fraud-detection in a multivariate setup in the Bitcoin network.

Li et al. [24] presented a comprehensive survey of data-mining techniques, including the study of deep learning techniques used for anomaly detection. The authors also provided summaries of the various general and specialized detection techniques. They discussed the merits and cons of the various techniques employed as well as what the future may hold for this profession. Bartoletti et al. [25] also used data-mining techniques for detecting Ponzi schemes in the Ethereum blockchain. Ponzi schemes are financial scams that make big profit claims but rely on fresh user investments to pay off old users. When new users cease signing up, the scheme fails.

Yuan et al. [26], developed phishing fraud detection models in the Ethereum exchange. The authors created a transaction network to train & test the model using the transaction records. Next, they trained the model using the node2vec embedding method and Support Vector Machine (SVM) classifier. This model achieved an absolute accuracy of 84.6%.

Aziz et al. [27] utilized various machine learning techniques for fraud detection in the Ethereum blockchain. The authors propose Light Gradient Boosting Machine (LGBM) approach for detecting fraudulent transactions. They employed different models, such as Random Forest (RF), Multi-Layer Perceptron (MLP), KNN, SVM, and XGBoost, and compared their metrics with the LGBM approach. The light gradient boosting machine (LGBM) algorithms and Extreme Gradient Boosting (XGBoost) demonstrate the highest accuracies in their work, while LGBM shows slightly better performance with 98.60%. Finally, they achieved an accuracy of 99.03% by optimizing the LGBM with hyperparameter tuning. Furthermore, it is noteworthy that Aziz, the principal author of the cited paper, has continued to contribute to the field with their recent study [28] using a novel deep learning technique. The performance of the proposed method was compared to that of a variety of well-known techniques, including k-nearest neighbours (KNN), logistic regression (LR), multi-layer perceptron (MLP), XGBoost, light gradient boosting machine (LGBM), random forest (RF), and support vector classification (SVC), in terms of restricted features. The authors also compared their performance and efficiency metrics to the proposed approach to detecting frailties. In the end, deep learning with the recommended optimization method exceeds the RF model, with a slightly higher performance of 99.71% vs 98.33%, while the suggested technique and SVC models outperform the other models with the highest accuracy.

The classification of the fraud account on the Ethereum network is suggested by RN Alief [17] using a CNN-based algorithm. The CNN model is used on a dataset that includes both normal and fraudulent transactions made across the Ethereum network. The results demonstrate that the CNN-based model effectively categorizes the fraudulent account with an accuracy of 98.02

Li et al. [29] developed an end-to-end Phishing Detection Graph Neural Network (PDGNN)

architecture, considering phishing detection to be a graph classification issue. The authors build a simple Ethereum transaction network initially, then extract transaction subgraphs from phishing accounts they have gathered. After that, they suggest an exact distinction between legitimate and phishing accounts using an end-to-end detection model based on Chebyshev-GCN. PDGNN beats conventional phishing detection techniques, according to their testing on five Ethereum datasets, and scales well in massive transaction networks.

The detection of illicit Ethereum accounts using machine learning was studied by Bella et al. [21] based on their transaction history and properties. The authors took a transactional data approach by extending feature computation to every transaction property. While studying three classification models: XGBoost, SVM Classifier, and Logistic Regression, the author uses a feature selection process to emphasize the most important characteristics. A 26-feature dataset produced by their model has an F1 score of 0.9654.

Abdul Quadir et al. [30] propose a novel approach for detecting fraudulent accounts associated with transactions on the Ethereum network. The proposed approach involves implementing machine learning algorithms, including Logistic Regression, Naive Bayes, Decision Trees, Random Forests, AdaBoosts, KNNs, SVMs, and Gradient Boosts, and creating a stacking classifier by combining several standalone classification algorithms and creating a meta-learner based on the output of each base algorithm. The proposed approach achieved a high accuracy of 97.18% with an F1 score of 97.02%. Of all the individual algorithms tested, the Random Forest algorithm had the highest accuracy rate of 95.47%, and the Gradient Boosting algorithm came in second place with an accuracy rate of 94.61%. The paper concludes that algorithms tend to perform better when combined properly.

Chapter 2

BACKGROUND AND PRELIMINARIES

2.1 Background

2.1.1 Blockchain and Decentralized Cryptocurrencies

The fundamental concepts behind blockchain technology emerged in the late 1980s and early 1990s. Leslie Lamport created the Paxos protocol in 1989, and in 1990 he submitted his work The PartTime Parliament to the ACM Transactions on Computer Systems, which eventually led to its publication in a 1998 edition. The study outlines a consensus methodology for determining a result in a computer network when the machines or network may not be reliable [2].

In 1991, a method of digitally signing documents using a chain of information was used as an electronic ledger to effortlessly demonstrate that none of the documents in the collection had been altered [31]. Later, these concepts were integrated and implemented into electronic cash and explained in the paper Bitcoin: A Peer-to-Peer Electronic Cash System [32] in 2008, published pseudonymously by Satoshi Nakamoto. Eventually, in 2009 the Bitcoin cryptocurrency blockchain network was established.

The digital economy now depends primarily on trustworthy authorities, which naturally implies that all online transactions go via the relevant entities like banks or financial institutions for handling financial transactions, payment processors such as PayPal or Stripe, e-commerce platforms such as Amazon or Shopify, and social media platforms like Facebook or Twitter for handling user data and privacy, necessitating our, as end users, implicit faith that the transactions have been completed safely and privately. Both the financial and non-financial domains can use this [14]. Blockchain technology enables the creation of a decentralized system for agreeing on the authenticity of digital information in an online environment [33]. At the same time, all transactions or events are recorded and stored chronologically_ and linearly_ on connected blocks as a public ledger. In fact, the public ledger is a database with all record history visible to everyone. It is built using an append-only data structure and maintained by a peer-to-peer network [34].

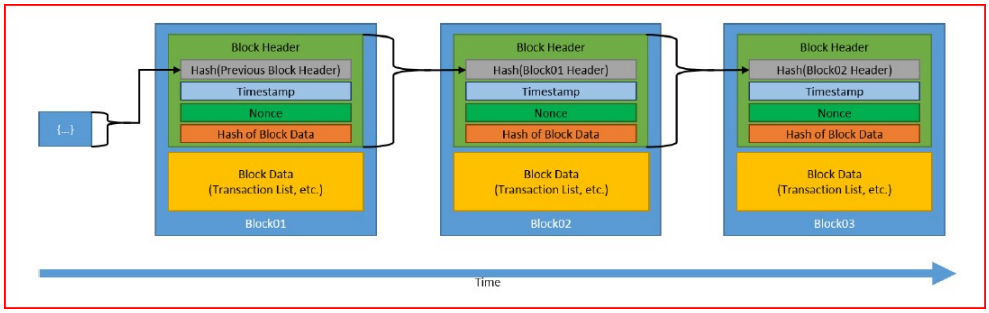

Figure 2.1: Schematic visualization of a chain of Blocks[2]

Transactions must be confirmed by the majority of the system's entities using the distributed consensus algorithm. Immutability, or the inability to change data once it has been added to a ledger, is made feasible by a mix of consensus methods and one-way cryptographic hash functions [35]. The Fig. 2.1 shows a generic chain of blocks.

A block in the blockchain network plays a crucial role in maintaining network integrity [1]. It consists of a block header containing essential information and block data, which collects transactions and other data. The block header includes the previous block's hash, acting as a unique identifier that links blocks together. It also contains a timestamp, nonce, and hash of the current block data to ensure security and maintain the chronological order of blocks in the chain. Once added to the chain, each block's transactions become a permanent part of the blockchain ledger [2].

However, although the immutability of blockchain ledgers is often touted, modifications to the underlying blockchain may be possible in specific scenarios [2]. A sense of confidence in the system is unintentionally created and further reinforced through public/private key technology without any third-party intervention. This may be avoided if a person or group of people controls more than 50% of the network. Nevertheless, if not, the ledger in question is seen as both a record of previous occurrences and a way to confirm the veracity of that record. In this way, double records usually referred to as the double spending problem, are automatically precluded and prevented from being recorded on the distributed ledger [36].

Blockchain technology possesses a trifecta of salient attributes which render it highly beneficial to various business operations and potential uses. These include the ability to implement algorithms via the utilization of smart legal contracts, facilitating secure and reliable digital financial transactions between parties, and providing an immutable repository for records and transactions.

2.1.2 Ethereum

Blockchain technology has undergone recent improvements, resulting in the emergence of several open-source platforms, including Ethereum. Created by Vitalik Buterin in July 2015 [37], Ethereum surpasses Bitcoin in terms of functionality and purpose. Unlike Bitcoin, which is primarily designed as a peer-to-peer electronic payment system and a store of value, Ethereum addresses these limitations by providing full Turing completeness through the use of smart contracts and its programming language [38].

The term "smart contract" was coined by Nick Szabo in 1994, referring to "a computerized transaction protocol that executes the terms of a contract." Smart contract design aims to fulfill typical contractual conditions, including payment terms, liens, confidentiality, and enforcement, while reducing malicious or accidental exceptions and minimizing reliance on trusted intermediaries [39].

A smart contract is a set of code and data, often called functions and state, deployed through cryptographically signed transactions on a blockchain network. Nodes within the blockchain network execute the smart contract, ensuring that all nodes yield identical results from the execution. The outcomes of the execution are then recorded on the blockchain, providing transparency and immutability to the smart contract's operations [2].

Users can send data to a smart contract on a blockchain network to perform a specific service. The code is tamper-resistant and acts as a trusted third party. Smart contracts can perform calculations, store information, and send funds to other accounts [2]. However, smart contracts extend beyond financial applications and can serve various purposes, including healthcare, information security, smart city, and IoT solutions [40].

The cryptocurrency of Ethereum, "Ether," has garnered popularity for its dual purpose. As a tradable asset, it is comparably utilized in a similar fashion to other digital currencies. Additionally, within the Ethereum ecosystem, it serves as a means of remuneration for the execution of transactions within decentralized applications.

One Ether equals 10^{18} Wei, for which Wei is the smallest amount of acknowledged Ether. This granularity permits Ethereum to accommodate an array of computations, monitor the state of executing transactions, and advance the Blockchain architecture. In essence, Ethereum can be considered a conceptual overlay on the fundamental Blockchain technology. It enables end-users to establish their regulations for defining proprietorship rights, formatting transactions, and determining the outcome of state transition functions. This could be achieved by implementing smart contracts, which mandate the satisfaction of specific parameters or conditions before the execution of cryptographic protocols [14].

In addition to components discussed earlier as for Fig. 2.1, the Ethereum blockchain also includes an additional field called the Ethereum accounts, which contains the state of all accounts on the network. Each account is uniquely identifiable via a 20-byte address that points to four fields used to ensure each state is unique; (i) Nonce: this value is incremented by 1 for each transaction, (ii) Contract byte code: is an immutable file because it is held inside the blockchain, (iii) Account balance: the current share of Ether currency (Wei) in the account, (iv)Account storage: externally held account handled by the owner's private key [37], [41].

Ethereum accounts are categorized into EOAs (externally owned accounts) and CAs (contract accounts). EOAs are designed to initiate transactions to other accounts, which may include optional input data and ETH transfers. When an EOA targets another EOA for a transaction, typically involving an ETH transfer, it proceeds without significant impact. However, if the transaction target is an existing smart contract linked by the "tnx to" parameter, Ethereum loads the contract's code and passes the transaction input data to the Ethereum Virtual Machine (EVM) for further processing. The EVM executes the code, and if no exceptions arise during this process, the resulting outcome is shared and synchronized across the entire network. Users can deploy new contracts in addition to engaging with an already-existing smart contract by leaving tnx_to empty and filling in the transaction input (also known as tnx_input) with suitably encoded init code [24]. ERC20 transactions are exclusively related to transferring tokens that constitute the ERC20 interface standard. Thus the transferable token must keep these standards, and only then are these tokens listed as transfer tokens [6].

Recently, the Ethereum blockchain has undergone a transition from proof of work to proof of stake consensus mechanism. Transactions on the Ethereum blockchain are still signed data packets containing six fields: recipient, sender, quantity, data, Startgas, and Gasprice [37]. However, instead of miners using significant computational power to solve a computational puzzle, validators with a stake in the network are chosen to validate transactions based on their stake. The most prominent validators will likely receive more rewards (and therefore validate more transactions) due to their

higher stake in the network. The use of proof of stake reduces the high energy consumption of proof of work and allows for a more environmentally friendly and cost-effective validation process [42]. The Startgas defines the limit, and senders' transaction fees (Gasprice) still serve as a regulatory mechanism to ensure the network is not abused.

2.2 Classification Models

In this thesis, we implement four distinct classification algorithms: NuSVC, Logistic Regression, Random Forest, and Multilayer Perceptron. Based on our literature review, these classification methods are widely used and have been shown to be effective in anomaly detection tasks in various domains [24], [43], [27], [41], [25], [44], [30][8], and [22]. These methods were chosen because they are fundamentally different in nature, and their comparison can provide insight into their strengths and limitations in our specific use case.

It is important to note that although other machine learning models can be used for anomaly detection, there is no one-size-fits-all approach, and the choice of models depends on various constraints. Our focus in this thesis was on hyperparameter tuning and feature selection, which are crucial for building accurate and efficient anomaly detection models. By optimizing these models and identifying the most important features for anomaly detection, we have contributed to the ongoing discourse in the field of anomaly detection.

2.2.1 NuSVC

Support Vector Classification (SVC) is a variant of Support Vector Machine (SVM) designed explicitly for classification. Support vector machines are supervised learning algorithms that analyze data, recognize patterns, and are mainly employed for classification and regression. They were created in order to address the issue of overfitting that was present in traditional neural networks [45]. One of the key components of an SVM is the creation of a hyperplane in an n-dimensional space, which is used to classify training data. The hyperplane's design maximizes the distance between the two classification categories as shown in Fig.2.2, which makes SVM suitable for classification. As a result, an SVM determines the best hyperplane to divide the provided data [46].

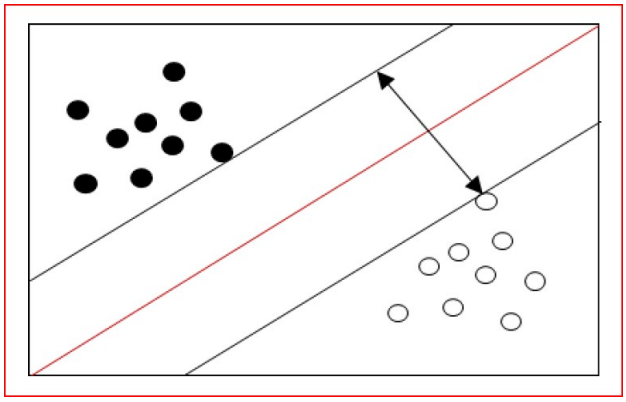

Figure 2.2: SVM for binary classification with the maximum margin hyperplane shown by the red margin [47]

SVC and NuSVC are similar methods but differ in the parameters they accept and their mathematical formulations. Classification of data using NuSVC can be done linearly as well as non-linearly. Based on the non-linearity of our data, we used a non-linear version of the method.

When dealing with a set of data points that cannot be classified into two classes by a linear hyperplane, it is common to map the data to a higher dimension. This allows the data to be separated by a hyperplane linearly. By employing different types of kernel functions, which map the data points to a plane where they are linearly separable, we can do non-linear classification for situations where the data cannot be linearly separable into two clean categories. If the mapping function is given by $\phi(x)$, kernel $K(x, y)$ is given as [47]:

$$K(x, z) = \phi(x)^T \phi(z) \tag{2.1}$$

where $\phi(x)$ and $\phi(z)$ are feature maps that transform the input data x and z into a high-dimensional space.

The kernel function enables the algorithm to construct a hyperplane with a maximum margin in a feature space that can be transformed into a higher dimension. This transformation allows for better separation and classification of data points that may not be linearly separable in their original space. Although the classifier is represented by a hyperplane in the transformed feature space, it might be non-linear in the original input space. Thus, the optimization problem is now given as maximization of the margin of separation or minimization of the kernel function, this problem can be expressed in the dual form using the following equation:

$$\min_{w,b} \max_{\alpha \geq 0} \left\{ \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \right\} \tag{2.2}$$

where α represents the Lagrange multipliers, w and b represent the hyperplane parameters, and $K(x_i, x_j)$ represents the kernel function that computes the similarity between input data points x_i and x_j in the transformed feature space. y_i and y_j represent the class labels or target values associated with the input data points x_i and x_j, respectively. These labels indicate the category or class to which the data points belong. Typically, in binary classification tasks, y_i and y_j can take the values of either -1 or +1 values, where -1 represents one class, and +1 represents the other. The multiplication of y_i and y_j in the equation ensures that the optimization problem considers the correct sign of the terms based on the class labels when computing the kernel function.

In this method, the training and testing data presented in the original feature space may be compared using a kernel function instead of computing the dot product of data that has been changed to a new higher dimension data.

”In contrast to the traditional SVM, where the C parameter is used to control the trade-off between maximizing the margin and minimizing the classification error on the training set, NuSVC employs the parameter ’nu’ to regulate the number of support vectors.

The ’nu’ parameter specifies both the lower bound on the fraction of support vectors and the upper bound on the fraction of training errors. Its value ranges between 0 and 1, representing an upper bound on the fraction of training errors and a lower bound on the fraction of support vectors.

The choice of ’nu’ directly influences the number of support vectors and the margin of the classifier. A smaller ’nu’ value leads to fewer support vectors and a wider margin, while a larger ’nu’ value results in more support vectors and a narrower margin [48].

In our implementation of the NuSVC model, we utilized the Radial Basis Function (RBF) kernel. The RBF kernel is widely recognized in machine learning for its ability to capture complex relationships between features due to its infinite complexity [47].

The RBF kernel equation is defined as:

$$\exp(-\gamma|\mathbf{u}-\mathbf{v}|^2) \tag{2.3}$$

Here, the vectors $\mathbf{u}$ and $\mathbf{v}$ represent the feature vectors of two distinct support vectors. These vectors are employed to compute the similarity between support vectors in the transformed feature space. The constant parameter γ needs to be defined by the user [47]."

2.2.2 Logistic Regression

Logistic regression is a statistical model that can be used to predict a binary outcome based on one or more independent variables [44]. It applies a logistic function to the linear combination of the independent variables, which converts the log-odds of the outcome to a probability between 0 and 1. The coefficients of the linear combination are estimated using maximum likelihood estimation, which maximizes the log likelihood function of the observed data [49].

Letting $\mathbf{X}$ denote the vector of predictors $X_1, X_2, \ldots, X_k$, The standard linear regression model could be used as an initial effort to model the response [50]:

$$\mathbb{E}(Y|\mathbf{X}) = \mathbf{X}^T\boldsymbol{\beta} \tag{2.4}$$

When analyzing binary data, the expectation of the variable Y is represented by the probability that Y equals 1. However, a purely linear model cannot accurately fit the data across all predictor ranges as it may result in probabilities exceeding 1 or dropping below 0. To address this, the binary logistic regression model is preferred for analyzing binary responses, considering the probability of Y equaling 1 based on the predictor values:

$$\text{Prob}(Y=1|\mathbf{X}) = \frac{1}{1+\exp\left(-\mathbf{X}^T\boldsymbol{\beta}\right)} \tag{2.5}$$

where $\mathbf{X}^T\boldsymbol{\beta}$ stands for $\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k$. The regression parameters $\boldsymbol{\beta}$ are estimated by the method of maximum likelihood.

The logistic function:

$$P(x) = \frac{1}{1+\exp(-x)} \tag{2.6}$$

has an unlimited range for x while $P(x)$ is restricted to the range from 0 to 1. Fig.2.3 shows the general representation of Linear and Logistic Regression.

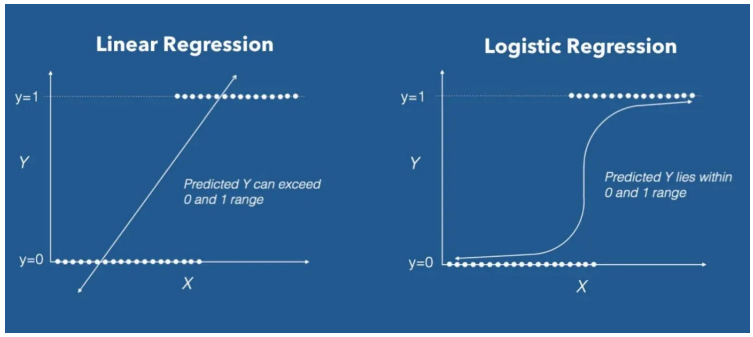

Figure 2.3: Logistic Regression VS Linear Regression [51]. Logistic regression fits a line to the data that separates the two classes, maximizing the likelihood of the data given the line by minimizing a cost function.

However, in order to obtain the best estimates for the regression coefficients, it is necessary to minimize a cost function. The most common cost function used in logistic regression is the cross-entropy loss, also known as the log-loss function:

$$\text{Cost}(\boldsymbol{\beta}) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\text{Prob}(Y=1|\mathbf{X}_i)) + (1-y_i)\log(1-\text{Prob}(Y=1|\mathbf{X}_i))] \tag{2.7}$$

where $\boldsymbol{\beta}$ are the regression coefficients, n is the sample size, $\mathbf{X}_i$ is the vector of predictor values for the i-th observation, and y_i is the binary outcome for the i-th observation.

The goal of logistic regression is to find the values of $\boldsymbol{\beta}$ that minimize the cost function. This is typically done using an optimization algorithm, such as gradient descent, which iteratively updates the values of $\boldsymbol{\beta}$ to move towards the minimum of the cost function.

One of the most important hyperparameters of logistic regression is the regularization parameter, which controls the amount of penalty applied to the coefficients to prevent overfitting. The L_2 (Ridge) regularization term is proportional to the sum of the squares of the regression coefficients. This penalty encourages the regression coefficients to be small, which has the effect of shrinking the coefficients towards zero, but not to zero. This means that the L_2 regularization term can still retain all the predictors in the model, but it downweights the importance of some of the less important predictors [52].

The L_2 regularization formula for logistic regression is [53]:

$$L(\boldsymbol{\beta}) = -\frac{1}{n}\sum_{i=1}^{n} y_i \log(P(\mathbf{X_i}\boldsymbol{\beta})) + (1-y_i)\log(1-P(\mathbf{X_i}\boldsymbol{\beta})) + \frac{\lambda}{2}\sum_{j=1}^{p}\beta_j^2 \tag{2.8}$$

where $\boldsymbol{\beta}$ is the vector of regression coefficients, $\mathbf{X_i}$ is the vector of predictors for the ith observation, y_i is the binary response variable for the ith observation, n is the number of observations, p is the number of predictors, λ is the regularization parameter, and $P(\mathbf{X_i}\boldsymbol{\beta})$ is the predicted probability of the ith observation belonging to class 1, based on the logistic regression model.

The first term in the formula represents the negative log-likelihood of the logistic regression model, which measures the model's goodness of fit to the data. The second term is the regularization term, which penalizes the magnitude of the regression coefficients to prevent overfitting. The magnitude of the penalty is controlled by the regularization parameter λ, with larger values of λ leading to more regularization.

2.2.3 Random Forest (RF)

Random Forest is a supervised machine-learning algorithm introduced by Leo Breiman and Adele Cutler in 2001 [54]. It is an ensemble learning method that combines the output of several decision trees using averaging to improve predictive accuracy and reduce overfitting. It can be used to solve classification or regression problems.

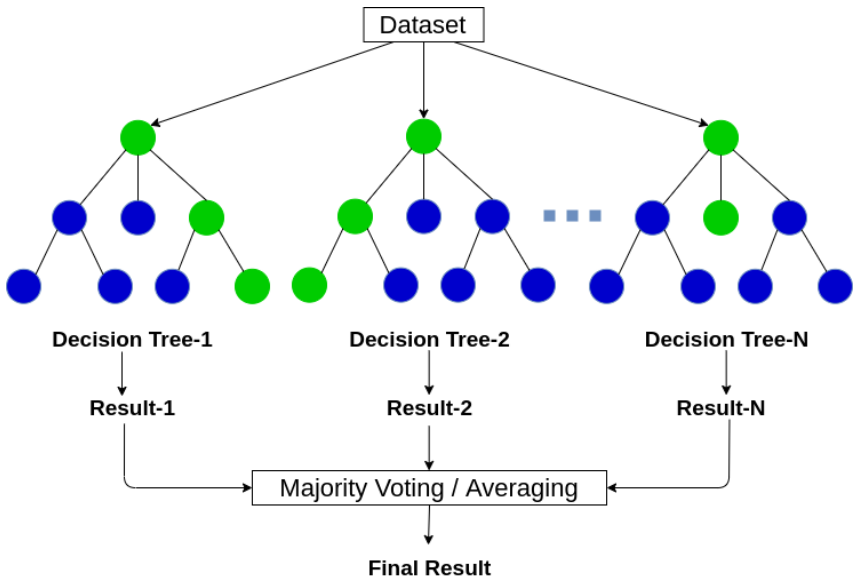

Figure 2.4: An example of a Random Forest classifier's visual representation [55].

The objective of a random forest classifier is to create a prediction function $f(X)$ for a response variable Y, based on an n-dimensional input vector $X = (X_1, X_2, ..., X_n)$. The prediction function aims to minimize the expected value of the loss, which is calculated using a loss function $L(Y, f(X))$. Typically, a zero-one loss function is used, which assigns a value of 0 if $Y = f(X)$ and a value of 1 otherwise [56].

During the tree learning phase, the random forest classifier automatically selects the attributes to divide the data based on a splitting function, such as the Gini index. The Gini index calculates the impurity or inequality of the samples given to a node based on the selected attribute split [57]. For instance, in our classification case, where there are two classes, "illicit and "normal", let p_{mk} be the fraction of samples of class k in node m. Then, the Gini index G_m at node m is defined as:

$$G_m = \sum_{k=1}^{K} p_{mk}(1 - p_{mk}) \tag{2.9}$$

where K is the number of classes.

From a computational standpoint, Random Forests are demanding because they (i) naturally handle both regression and (multi-class) classification, (ii) are relatively fast to train and predict, (iii) depend only on one or two tuning parameters, (iv) have a built-in estimate of the generalization error, (v) can be used directly for high-dimensional problems, (vi) can easily be implemented in parallel [56]. From a statistical standpoint, Random Forests are appealing because they provide additional

features such as (i) measures of variable importance, (ii) differential class weighting, (iii) missing value imputation, (iv) visualization, (v) outlier detection, and (vi) unsupervised learning [56]. These features can be particularly useful in data analysis and modeling for various applications.

With Random Forests, it is possible to obtain more than just a prediction or classification but also additional insights about the data and the underlying relationships.Due to its robustness and flexibility, Random Forest has been widely used in anomaly detection problems, such as fraud detection [58], intrusion detection, and medical diagnosis.

2.2.4 Neural network Multilayer Perception (MLP)

Artificial neural networks (ANN), also known as neural networks (NN), were developed due to respect for how complicated computations are carried out in the human brain, which differs significantly from how traditional digital computers carry them out [59]. The Multilayer Perceptron (MLP) is a neural network architecture widely used for classification tasks due to its ability to model complex non-linear relationships between input features and output class labels. The model consists of multiple layers of neurons, where each neuron in a layer is connected to all the neurons in the previous layer Fig.2.5. The neurons in each layer use a non-linear activation function to transform the weighted sum of inputs into an output signal, which is then fed as inputs to the next layer [60].

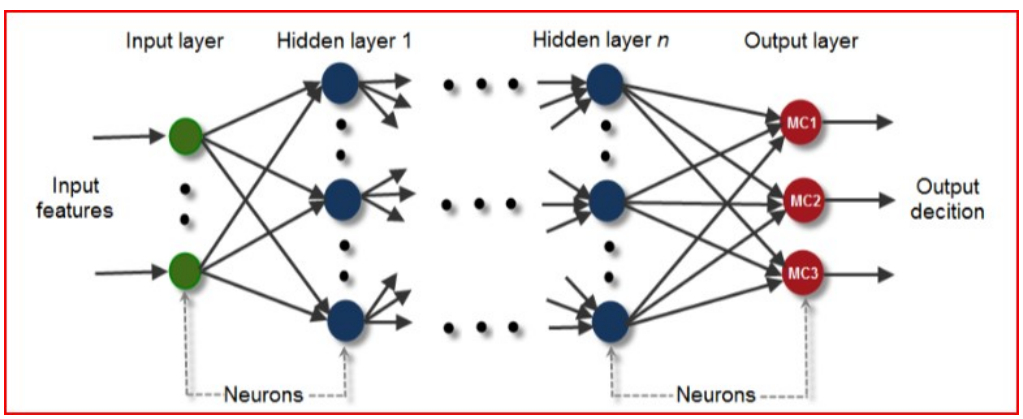

Figure 2.5: Schematic of Neural Network MLP classifier with circles representing neurons and arrows as the connections between Input, Hidden, and Output Layers [61].

The inputs received by the neuron are modified by the synaptic weights and bias to produce the net input, which is evaluated by the activation function to determine the neuron's output. The output can be propagated to other neurons or be the network output. The neurons' interconnection through connection links with associated weights and activation functions allows NN to solve various real-world problems. The activation functions can be nonlinear and vary depending on the context of the problem [59]. The hidden layers in an MLP are responsible for learning increasingly complex and abstract features from the input data [62].The weights and biases of the neurons in each layer are learned during the training process using a backpropagation algorithm that computes the gradients of the loss function with respect to the network parameters. The gradients are then used to update the weights and biases using an optimization algorithm, such as stochastic gradient descent.

The output layer of the MLP model computes the predicted class probabilities using the Softmax function [63]:

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \tag{2.10}$$

where z_i denotes the ith element of the input to softmax, which conforms to class, and k is the number of classes.

The Softmax function transforms the output signals of the neurons in the last hidden layer into a probability distribution over the output classes. The predicted class is then determined by selecting the class with the highest probability.

2.3 Feature Selection Methods

Collecting data is usually accompanied by too much noise in the real world. Various factors play a role in creating noise in this data, but two important ones are flaws in the technologies used to acquire the data and the data's original source itself. It is difficult to extract useful information and patterns from such a large and noisy data set.

One of the most widely employed methods for removing noisy and redundant features is dimensionality reduction. The two primary categories of dimensionality reduction approaches are feature extraction and feature selection. The newly built features created by feature extraction methods are often combinations of the original features, which are projected onto a new feature space with reduced dimensionality. Principle Component Analysis (PCA), Linear Discriminant Analysis (LDA), and Canonical Correlation Analysis (CCA) are a few examples of feature extraction approaches. On the other hand, feature selection procedures, like class labels in classification, try to choose a limited feature group that reduces redundancy and enhances objective relevance [64]. A general framework of feature selection for classification is depicted in Fig2.6.

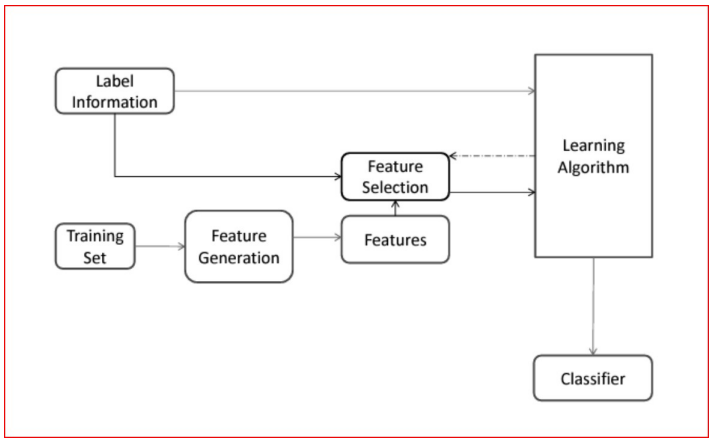

Figure 2.6: A general framework of feature selection for classification [64].

Feature selection is a crucial step in developing machine learning (ML) models, particularly for anomaly detection applications that involve identifying patterns in data that deviate from the norm or expected behavior. Feature selection is primarily focused on removing non-informative or redundant predictors from the model [65]. Using too many or irrelevant features can lead to overfitting, where the model is trained to fit the noise in the data rather than the underlying patterns. On the other

hand, using too few or irrelevant features can lead to underfitting, where the model cannot capture the relevant patterns in the data. Therefore, feature selection plays a critical role in optimizing the performance of ML models [66].

In general, feature selection methods can be classified into filters, wrappers, and embedded methods [67]. Filter-based feature selection ranks the features based on their statistical properties, such as their correlation with the target variable. Additionally, wrapper-based methods involve training the model with different subsets of features and selecting the subset that produces the best performance. Finally, embedded methods, such as regularization techniques, incorporate the feature selection process into the model training process.

2.3.1 Recursive Feature Elimination (RFE)

Recursive Feature Elimination (RFE) is a wrapper method using backward selection strategy for feature selection. The Recursive Feature Elimination with cross-validation (RFECV) general algorithm is depicted in Algorithm 1. In Every iteration, the relevance of the features is evaluated, and the less important ones are eliminated. Another option to speed up the process is eliminating a set of characteristics each time. For some measures, the relative relevance of each feature might change significantly when assessed across a new selection of features throughout the stepwise elimination process, necessitating the recursion (particularly for highly correlated features). A final ranking is created using the features' elimination order (inverse) [68].

Algorithm 1 Recursive Feature Elimination with Cross-Validation (RFECV)

Inputs:

- Dataset $\mathbf{X}$ with n features and m samples, target vector $\mathbf{y}$, estimator E, step size k, number of folds K

Outputs:

- Selected features $\mathbf{X}$selected

Steps:

1. Compute the minimum number of features p to consider
2. Compute the maximum number of features P to consider
3. Create an empty set for selected features $\mathbf{X}$selected
4. For each $d \in [p, P, k]$, do the following:
   (a) Use K-fold cross-validation to estimate the performance of the model with d features
   (b) Compute the average performance score across all folds
   (c) If $d == p$, set the current score to the maximum score and set $\mathbf{X}$selected to include the d features
   (d) If the current score is greater than the maximum score, set the maximum score to the current score and set $\mathbf{X}$selected to include the d features
5. Return $\mathbf{X}$selected

The algorithm starts by initializing an estimator, step size, and the number of folds for cross-validation. Then, it computes the minimum and the maximum number of features to consider, creates an empty set for selected features, and iterates through all possible numbers of features within the specified range. At each iteration, it uses cross-validation to estimate the model's performance with the current number of features, selects the score for the current iteration, and compares it with the previous best score. If the current score is better, it updates the selected features. The process continues until the optimal number of features is determined. The algorithm's output is a subset of features that optimizes the model's performance. The goal of RFECV is to identify the optimal subset of features that maximizes model performance. In theory, any machine learning model can be used inside the RFECV algorithm as long as it can produce a feature importance ranking, including Decision Trees, Random Forest, Support Vector Machines, Gradient Boosting, etc. In [68], the authors compare traditional Support Vector Machine-Recursive Feature Elimination (SVM-RFE) and Random Forest-Recursive Feature Elimination (RF-RFE) algorithms. They employed repeatable experiments and independent test sets to produce objective and consistent performance assessments. They found that RF-RFE has a better performance than SVM-RFE.

2.3.2 Random Forest built-in Feature Importance (FI)

Random forest directly conducts feature importance, while classification rules are built [54]. The importance score for each feature is based on the decrease in node impurity (e.g., Gini impurity for classification and mean squared error for regression) weighted by the probability of reaching that node. The node probability can be computed by dividing the total number of samples by the number of samples that reach the node. The feature that arises from dividing on that trait is of more importance when the value is higher.

We discussed how we calculate a Gini index in the section 2.2.3. The sum of the Gini indexes decrease (from parent to children) over all nodes where the particular feature is utilized to split is then used to establish the Gini importance value of a feature in a single tree. The total or average importance value of all the forest's trees is referred to as the forest's overall importance [57].

We may use the following steps to demonstrate how we determine the significance of each feature using Gini importance in a random forest model with 100 decision trees [69]:

For each feature j, calculate the total decrease in Gini impurity (TDI) at each node m of each decision tree i using the following formula:

$$TDI(j,m,i) = A \cdot (B - C \cdot D - E \cdot F) \tag{2.11}$$

where:

- A: Proportion of samples that reach node m of tree i compared to the total number of samples n_i.

  Formula: $A = \frac{n_{m,i}}{n_i}$

- B: Gini impurity score at node m of tree i.

  Formula: $B = Gini(m,i)$

- C: Proportion of samples that go to the left child node of node m of tree i compared to the samples at node m.

  Formula: $C = \frac{n_{left,m,i}}{n_{m,i}}$

- D: Gini impurity score at the left child node of node m of tree i.

  Formula: $D = Gini(left, m, i)$

- E: Proportion of samples that go to the right child node of node m of tree i compared to the samples at node m.

  Formula: $E = \frac{n_{right,m,i}}{n_{m,i}}$

- F: Gini impurity score at the right child node of node m of tree i.

  Formula: $F = Gini(right, m, i)$

For each decision tree i, calculate the total decrease in Gini impurity over all the nodes where each feature is used to split, as follows:

$$TDI(j,i) = \sum_{m} TDI(j,m,i) \tag{2.12}$$

where the sum is taken over all nodes m where feature j is used to split in tree i.

For each feature j and each decision tree i, calculate the importance of feature j in tree i as the ratio of its total decrease in Gini impurity to the sum of the total decrease in Gini impurity over all features, using the following formula:

$$FI(j,i) = \frac{TDI(j,i)}{\sum_{j'} TDI(j',i)} \tag{2.13}$$

Average the feature importance scores over all decision trees to obtain the mean importance of each feature, as follows:

$$FI(j) = \frac{1}{100} \sum_{i=1}^{100} FI(j,i) \tag{2.14}$$

Normalize the mean importance scores by dividing each score by the sum of all the mean importance scores, using the following formula:

$$FI'(j) = \frac{FI(j)}{\sum_{j'} FI(j')} \tag{2.15}$$

where $FI'(j)$ is the normalized importance score for feature j.

The resulting values $FI'(j)$ are the Gini importance scores for each feature. The higher the value, the more important the feature.These importance scores can be used to rank the features and select the most important ones for further analysis or model building.

Chapter 3

METHODOLOGY

This chapter provides a comprehensive explanation of the implementation of our classification models for the purpose of classifying the dataset that comprises illicit accounts identified by the Ethereum community as well as normal accounts that have not been associated with illegal activity or transactions on the network.

Following data pre-processing, the classification models are trained using the prepared dataset. A pipeline tool was developed to improve the models' performance. We used GridSearch cross-validation to find the best hyperparameters in the training set. Further, feature selection methods RFECV and RF-FI are applied to identify the most important features within the dataset. These essential features are then used to retrain our models, thereby reducing the dimensionality of the dataset and improving the models' interpretability. In Chapter 4, we apply different evaluation metrics such as accuracy, F1 score, and AUC to assess the models' outcomes.

3.1 Data visualization

The t-SNE tool is for high-dimensional data visualisation. It attempts to reduce the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data and translates similarities between data points into joint probabilities. Because the cost function for t-SNE is not convex, we can achieve different outcomes with various initializations [70].

3.1.1 Description of the Dataset and Target Variable

The dataset used in this thesis consists of 4681 rows, comprising 2179 fraudulent accounts and 2502 normal accounts. The accounts are characterized by 42 features that have been extracted based on their transaction history. The dataset was initially created and utilized by Steven et al. [6] and is available online on his GitHub page [71]. The author utilized various resources and conducted a thorough, comprehensive approach to creating this dataset. In their study, they detailed the whole data gathering process and extracting features. Table 3.1 shows a complete list of extracted features [6]. This labeled dataset delivers ground truth labels for training and evaluating the machine learning algorithms. The dataset's features, derived from account historical transactions, offer valuable insights into illicit behavior patterns. These features enable the algorithms to effectively classify accounts and flag suspicious ones, aiding in detecting illicit activities on the Ethereum blockchain.

Our target variable for analysis is the 'FLAG' column, which categorizes accounts as either normal (0) or illicit (1). We set the stratified parameter to maintain the ratio between the two types of accounts. The dataset has significant variations in value ranges and includes missing values. Hence, we performed several vital steps in the pre-processing stage to ensure our dataset is adequately prepared for model training and testing.

3.2 Data Pre-processing

To prepare the dataset for modeling, we applied the LabelEncoder to convert string values to numerical values and the Iterative imputation method to handle missing values. Given the large size and wide range of numerical data in our dataset, we use StandardScaler for standardization within a pipeline. This tool centers each feature by subtracting its mean value and scales it by its standard deviation, resulting in a standardized normal distribution. These pre-processing techniques ensure the dataset was well-formatted and clean, enabling the models to produce accurate and dependable results.

3.2.1 LabelEncoder

Many machine learning algorithms cannot directly process categorical variables. To address this, we need to convert them into numerical values. Two primary methods exist for converting categorical variables: nominal, which has no particular order, and ordinal, which has some order. The categorical variables in our dataset are nominal, so we utilized LabelEncoder technique, a widely used pre-processing tool for converting categorical data into numerical values.

The LabelEncoder assigns a unique integer value to each unique category or class in the categorical column, thereby encoding the data into a numeric format that algorithms can more easily process. The LabelEncoder can handle both string and numeric values, but it is typically used to transform categorical data in a way compatible with various machine learning algorithms. When applying the LabelEncoder to a string column, it first identifies all the unique string values in the column. Then it assigns a unique integer label to each unique string value. We employed this method to transform our dataset's two categorial features 'ERC20_most_rec_token_type' and 'ERC20_most_sent_token_type'.

3.2.2 Identifying & Handling missing values

The heatmap in Fig3.1, depicts the percentage of columns and rows with missing data, suggesting that multiple Ethereum transactions have missing attribute values in our dataset. We employ Iterative imputation method to handle missing values in the dataset.

Imputation is the process of substituting values for missing values in a dataset, and it can be categorized into two main varieties: (i) Univariate imputation: missing values in a feature are replaced with some statistic of the observed data in that feature alone, such as mean or median. (ii)Multivariate imputation: missing values are imputed using observed data from other features, such as Regression imputation, K-Nearest Neighbor (KNN) imputation, and Iterative imputation. The goal of imputation is to minimize the loss of information and reduce the bias that might result from deleting records with missing values.

Based on our literature review, in general, there is no single "best" imputation technique that works well for all types of datasets. Most of studies use simple Univariate techniques using mean or median or even zeros [23] and [6] to handle their missing values and some other just simply remove them from their dataset [27] and [72]. However, we utilize Iterative Imputation method for the first time on Ethereum dataset which is a type of Multivariate Imputation (MI).

Index	Feature's Name	Features' Description	Data Type
0	Avg_min_between_sent_tnx	Average time between sent transactions for account in minutes	Integer
1	Avg_min_between_received_tnx	Average time between received transactions for account in minutes	Integer
2	Time_Diff_between_first_and_last_(Mins)	Time difference between the first and last transaction	Integer
3	Sent_tnx	Total number of sent normal transactions	Integer
4	Received_Tnx	Total number of received normal transactions	Integer
5	Number_of_Created_Contracts	Total Number of created contract transactions	Integer
6	Unique_Received_From_Addresses	Total Unique addresses from which account received transactions	Integer
7	Unique_Sent_To_Addresses	Total Unique addresses from which account sent transactions	Integer
8	min_value_received	Minimum value in Ether ever received	Double
9	max_value_received	Maximum value in Ether ever received	Double
10	avg_val_received	Average value in Ether ever received	Double
11	min_val_sent	Minimum value of Ether ever sent	Double
12	max_val_sent	Maximum value of Ether ever sent	Double
13	avg_val_sent	Average value of Ether ever sent	Double
14	min_value_sent_to_contract	Minimum value of Ether sent to a contract	Double
15	max_val_sent_to_contract	Maximum value of Ether sent to a contract	Double
16	avg_value_sent_to_contract	Average value of Ether sent to contracts	Double
17	total_transactions_(including_tnx_to_create_contract)	Total number of transactions	Integer
18	total_Ether_sent	Total Ether sent for account address	Double
19	total_ether_received	Total Ether received for account address	Double
20	total_ether_sent_contracts	Total Ether sent to Contract addresses	Double
21	total_ether_balance	Total Ether Balance following enacted transactions	Double
22	Total_ERC20_tnxs	Total number of ERC20 token transfer transactions	Integer
23	ERC20_total_Ether_received	Total ERC20 token received transactions in Ether	Double
24	ERC20_total_ether_sent	Total ERC20 token sent transactions in Ether	Double
25	ERC20_total_Ether_sent_contract	Total ERC20 token transfer to other contracts in Ether	Double
26	ERC20_uniq_sent_addr	Number of ERC20 token transactions sent to Unique account addresses	Integer
27	ERC20_uniq_rec_addr	Number of ERC20 token transactions received from Unique addresses	Integer
28	ERC20_uniq_rec_contract_addr	Number of ERC20 token transactions received from Unique contract addresses	Integer
29	ERC20_avg_time_between_sent_tnx	Average time between ERC20 token sent transactions in minutes	Integer
30	ERC20_avg_time_between_rec_tnx	Average time between ERC20 token received transactions in minutes	Integer
31	ERC20_avg_time_between_contract_tnx	Average time ERC20 token between sent token transactions	Integer
32	ERC20_min_val_rec	Minimum value in Ether received from ERC20 token transactions for account	Double
33	ERC20_max_val_rec	Maximum value in Ether received from ERC20 token transactions for account	Double
34	ERC20_avg_val_rec	Average value in Ether received from ERC20 token transactions for account	Double
35	ERC20_min_val_sent	Minimum value in Ether sent from ERC20 token transactions for account	Double
36	ERC20_max_val_sent	Maximum value in Ether sent from ERC20 token transactions for account	Double
37	ERC20_avg_val_sent	Average value in Ether sent from ERC20 token transactions for account	Double
38	ERC20_uniq_sent_token_name	Number of Unique ERC20 tokens transferred	Integer
39	ERC20_uniq_rec_token_name	Number of Unique ERC20 tokens received	Integer
40	RC20_Most_Sent_Token_Type	Most sent token for account via ERC20 transaction	String
41	ERC20_Most_Rec_Token_Type	Most received token for account via ERC20 transaction	String

Table 3.1: Full set of 42 features of the original dataset.

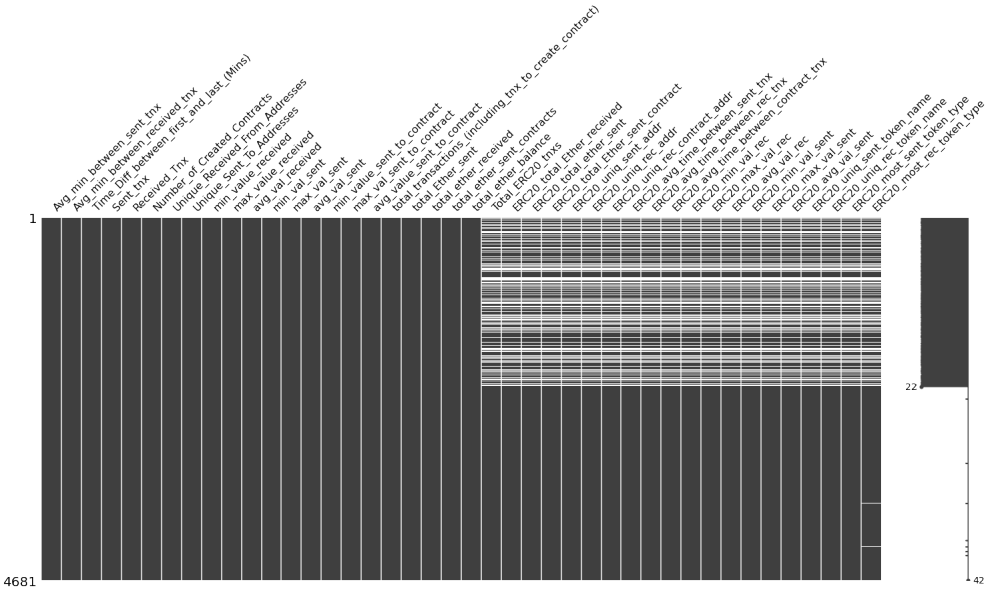

Figure 3.1: Visualizing missing values in the dataset: the white lines indicate existing of missing values in the relative rows and columns.

According to a study by Azur et al [73], iterative imputer is one of the best-performing imputation techniques for categorical data. The study compare several imputation techniques and found that iterative imputer outperformed other methods regarding accuracy and bias, such as mean imputation and k-nearest neighbor imputation. Unlike other imputation techniques, iterative imputer can handle missing values in numerical and categorical features [74].

One of the reasons why iterative imputer works best for categorical data is that it can handle missing values in a way that preserves the underlying structure of the data. It is essential because categorical data often contains meaningful relationships between the categories, and it is vital to preserving these relationships when imputing missing values. Another advantage of iterative imputer is that it can handle missing values in both independent and dependent variables. In many cases, missing values in the dependent variable are the target of the analysis, and it is essential to handle them appropriately.

Algorithm 2 [75], shows how iterative imputer imputes missing values in a dataset. The Bayesian Ridge regression is used as an estimator inside our method, known for its ability to handle complex, high-dimensional data. The method starts by filling in missing values with an initial guess, such as the mean or median value of the column. Then, it uses a regression model to predict the missing values in each column using the observed values of other columns as input. This process is repeated several times until the imputed values converge. We set the ”max-iter” parameter to 10, so the process will be repeated ten times. The resulting imputed dataset can then be used for further analysis or modeling.

Algorithm 2 Iterative Imputer Algorithm for Missing Data Imputation

Inputs:

- **X**: incomplete dataset with missing values
- estimator: regression model used to estimate missing values
- max-iter: maximum number of iterations

Outputs:

- **X**complete: completed dataset with imputed missing values

Steps:

1. Initialize missing values in **X** using a simple imputation method (e.g., mean or median).
2. Repeat until convergence or maximum number of iterations:
   - For each feature i with missing values:
     - Define **X**obs as the subset of **X** where feature i is observed.
     - Define **X**miss as the subset of **X** where feature i is missing.
     - Use estimator to estimate missing values in **X**miss based on observed values in **X**obs.
     - Replace missing values in **X**miss with the estimated values.
3. Return the completed dataset $\mathbf{X}_{\text{complete}}$.

3.3 Models Implementation

For the purpose of model implementation, we build a pipeline tool that incorporated key components such as data standardization, imputation, parameter tuning, model training, and evaluation. We split the dataset into training and testing sets to develop our classification models, with the former accounting for 80% of the data. We use hyperparameter tuning techniques to determine the optimal parameters for each model by testing different parameter combinations iteratively. This streamlines the model development process, allowing us to identify the best models for our use case. We utilize Python programming language and imported relevant libraries such as scikit-learn, Pandas, NumPy, and Matplotlib. We create a classifier object with specified parameters for model execution and trained it on the training set. The object is then used to predict the corresponding test set, and this process was repeated for all classifiers with varying fold numbers. Finally, we analyze and discuss the results in Chapter 4 of this thesis.

3.3.1 Hyperparameter Tuning

Finding the best hyperparameters for a particular task is often challenging since the ideal settings cannot be determined ahead of time. Therefore, different hyperparameter combinations have been tested on the train set and the performance of each model be evaluated. However, this process can lead to overfitting, where a model scores well on the training data but cannot generalize to new data. To account for overfitting, cross-validation is an essential technique which is used in our

hyperparameter tuning.

K-fold Cross-validation is the process of splitting the data into training and testing sets and then further dividing the training set into K subsets, or folds. The model is iteratively trained K times, each time on K-1 of the folds and evaluated on the remaining fold. The average performance of each fold is then used to determine the final validation metrics for the model. Stratified K-fold cross-validation is used in this thesis which is a suitable choice for classification problems.

Stratified K-fold aims to ensure that each cross-validation fold contains approximately the same proportion of each class as the entire dataset. It helps to prevent bias in the evaluation of the model's performance [9]. By stratified sampling set to the output 'FLAG' variable, a variety of folds 3-, 4-, 5-, and 10-folds were tested. We evaluated different selected hyperparameters. We evaluated these configurations using grid-search, a technique for determining the best parameter values based on the context of each model.

Using the same stratified parameter, we apply various configurations to the k-fold cross-validation, including 3-, 4-, 5-, and 10-fold. The results of these evaluations were provided in the results section and were primarily used to identify any potential issues with overfitting or underfitting in our models. By following these steps, we ensure that our dataset was prepared correctly for model training and testing and that our models were capable of accurately identifying illicit accounts while avoiding false positives.

3.4 Performance Measures

We employ several primary evaluation measures to assess the relative performance of our models with different hyperparameters. The Area Under the Receiver Operating Characteristic (ROC) Curve (AUC) is used to evaluate the performance of the binary classifiers, which assesses their ability to distinguish between positive and negative examples. Additionally, we use the Confusion Matrix, a contingency table that displays the number of true positives, true negatives, false positives, and false negatives. In addition to these primary measures, we analyze other indicators such as accuracy and F1 score to evaluate our models comprehensively.

3.4.0.1 Confusion Matrix and AUC

The confusion matrix sometimes called the "Error Matrix", is used as a measurement of accuracy [76]. There are four possible predicted and actual value combinations within the table.

True Positive (TP): Indicates how many target flows were accurately detected.

True Negative (TN): Indicates how many other flows were accurately detected.

False Positive (FP) Type 1 Error: Indicates how many target flows were detected incorrectly.

False Negative (FN) Type 2 Error: Indicates the number of target flows that were not detected.

Comparing classification results with actual measured values is possible by displaying the accuracy of classification results within a confusion matrix [76]. Our scenario's "illicit" or "normal" accounts are TP and TN, respectively [78].

Accuracy counts how many positive and negative observations were accurately categorized.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{3.1}$$

The proportion of correctly classified positive classes divided by the total number of positive classes is known as Recall or Sensitivity. Alternatively, it represents how much of all the actual positive classes we accurately anticipated.

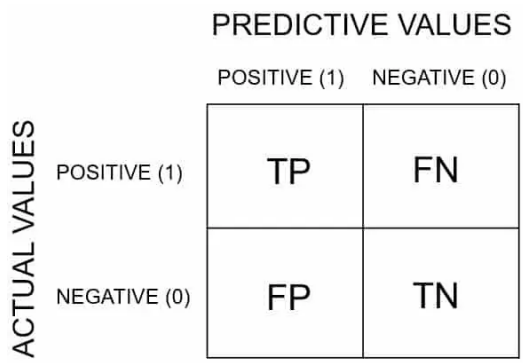

Figure 3.2: Confusion Matrix for binary classification [77].

$$Recall = \frac{TP}{TP + FN} \tag{3.2}$$

We also have precision

$$precision = \frac{TP}{TP + FP} \tag{3.3}$$

It measures how many of the predicted positive instances are positive. In other words, precision is the ability of the model to avoid false positives. A high precision score indicates that the model produces very few false positives. By having Recall and Precision now, we can calculate the F1 Score, the harmonic mean of precision and recall, which is expressed as follows:

$$F1Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3.4}$$

The F1 Score is a commonly used metric to evaluate the performance of a binary classification model. It combines both precision and recall metrics to provide an overall performance measure. Additionally, we have specificity, which defines the ratio of real negative classes that are correctly identified.

$$Specificity = \frac{TN}{FP + TN} \tag{3.5}$$

Now that we have the specificity, we can depict the ROC curve, which aids in achieving the accuracy measure.

The ROC curve plots the True Positive Rate (TPR) or Recall against the False Positive Rate (FPR) for different classification thresholds. We can flip the x-axis and plot the True Negative Rate (TNR) or specificity against the FPR for different classification thresholds to depict the ROC curve with specificity. The TNR is the proportion of actual negative cases correctly classified as negative by the model.

$$FPR = 1 - Specificity = \frac{FP}{FP + TN} \tag{3.6}$$

ROC graphs are two-dimensional graphs in which TPR and FPR are depicted on the Y and X axes, respectively. A ROC graph represents the relative trade-offs between ”illicit” (true positives) and ”normal” (false positives) accounts [79]. In order to determine the optimal performance of our

models for different configurations, we consider AUC as the preferred measurement. The higher the AUC, the better our model's accuracy. It is due to the risk associated with incorrectly classifying accounts as well as its ability to encompass all performance factors [14].

3.5 Feature Importance

A significant contribution of this thesis is performing thorough feature selection to identify the essential features in the dataset. We aim to compare two methods, Recursive Feature Elimination with Cross-Validation (RFECV) and Random Forest built-in Feature Importance (RF-FI), and generate new feature sets with fewer but most informative features to reduce dimensionality and model execution time while maintaining accuracy. RFECV method selects the most prominent features from the dataset, while RF-FI does not have this feature and instead weights all the features from the most important to the least. We set a threshold for RF-FI to select the most important features, resulting in 17 features for each model. We use the outcomes to create two new sets of features: a 16-set that included the union of the two methods' outcomes and eliminated correlated features and a 6-set that included the top three features from each method. We then retrain our models using these feature sets and compared the outcomes to find the best set of features in terms of accuracy and execution time.

Chapter 4

EXPERIMENTAL RESULTS

The results of the approach used are shown in the following sections. The findings are presented in several categories in the order that the implementation will be evaluated. Firstly, we will discuss some general dataset information, followed by a non-linear dimensionality reduction using the t-SNE graph, evaluating the models, using two methods for selecting important features, and re-evaluating our models with new sets of important features.

All implementations are carried out on a regular laptop with an 11th Gen Intel(R) Core i7-1165G7, 2.80GHz, 2.80 GHz processors, and 16GB available RAM using Python v3.8.8.

4.1 General dataset information

After extracting the features from the accessible accounts, we give some general details about the entire created dataset. We give specific information on characteristics that we consider are useful. The Correlation Matrix (Heatmap) of our characteristics are displayed in the Fig. 4.1. It illustrates how closely related different traits are. There are some relationships between the features, as the map demonstrates. We further employ some feature importance strategies to refine our features.

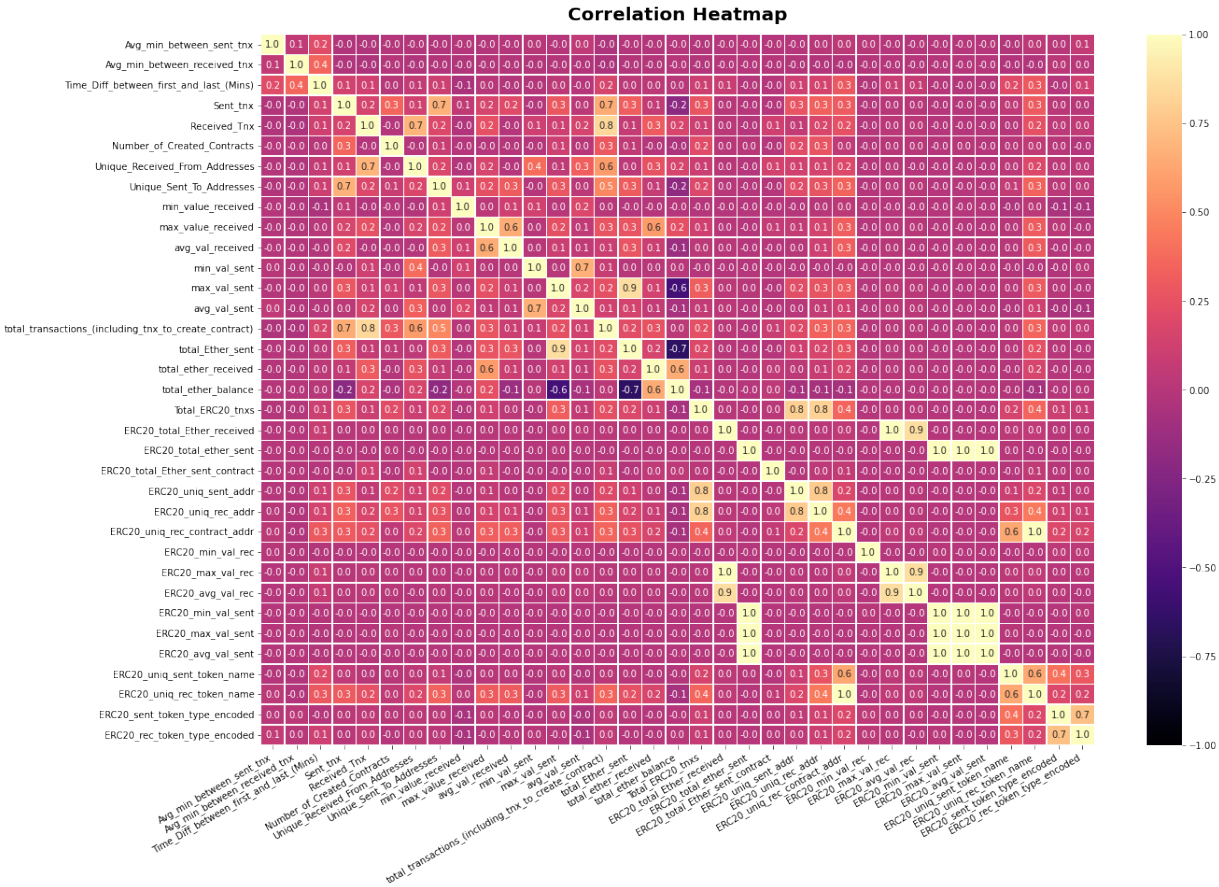

Figure 4.1: Visualization of the correlation between features.

4.2 t-SNE Visualization

We use the t-SNE tool to depict the available illicit and normal accounts in our dataset in 2D and 3D, as shown in Fig 4.2 and Fig4.3 respectively. The normal and illicit accounts are shown by red and blue data points respectively. We can observe a few distinct clusters from both classes in the 2D scatter plot in Fig. 4.2, most notably the three red clusters that are located close to the upper left corner of the plot. However, a significant amount of two different categories' data points overlap throughout the plot, indicating that our dataset is not linearly separable. The figure serves as a visual representation of the data at hand and emphasizes the need for machine learning approaches to aid in differentiating the two binary groups.

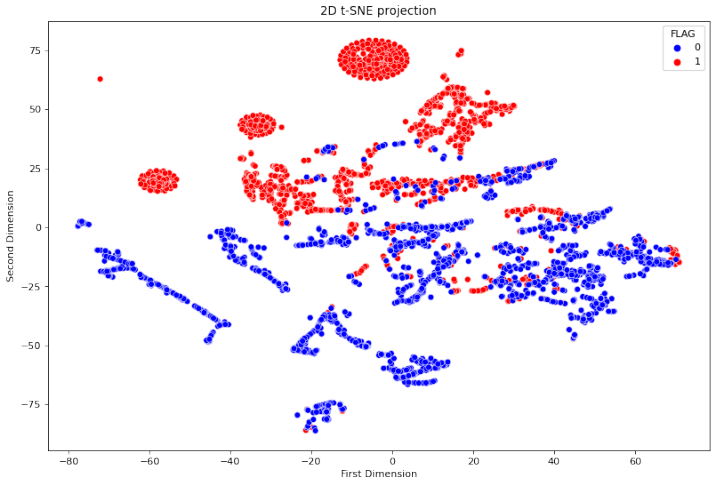

Figure 4.2: 2D scatter plot t-SNE projection regarding account's class. The data points in red represents illicit accounts, while the data points in blue represent normal accounts.

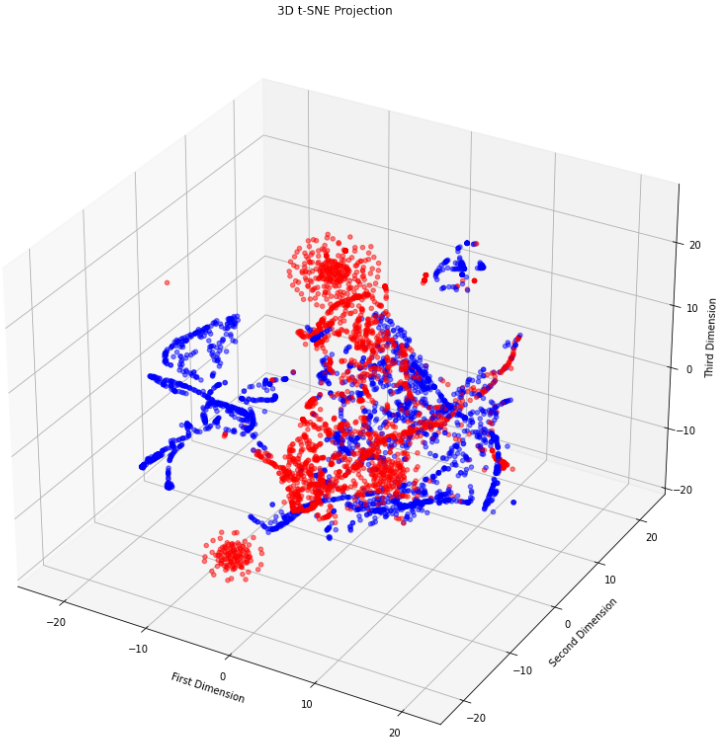

Figure 4.3: 3D scatter plot t-SNE projection regarding account's class. The data points in red represents illicit accounts, while the data points in blue represent normal accounts.

4.3 Classifiers Results

To detect illicit accounts within the Ethereum Blockchain, we employed four distinct classifiers: NuSVC, Logistic Regression, Random Forest, and Multilayer Perceptron. In the subsequent section, we will discuss the findings of the grid search cross-validation for each classifier, including evaluations conducted with 3-, 4-, 5-, and 10-fold methods.

4.3.1 NuSVC

In the proposed NuSVC model, two hyperparameters of interest were ”nu” and ”gamma”. The parameter ”nu” represents the upper bound on the fraction of training errors and the lower bound on the fraction of support vectors. The ”gamma” parameter controls the influence of each individual training sample, with smaller values of ”gamma” indicating a wider Gaussian radius for kernel function and, thus, a more flexible model. The results in Table 4.1 show that increasing the number of folds affects the model's performance. By increasing the number of folds, we can observe a noticeable enhancement in the average AUC. The highest AUC achieved is 0.978 (+/- 0.005) with 10 fold size. Correspondingly, the accuracy and F1 score of the 10 fold is also superior. The ”nu” parameter remains fixed at 0.2, while each fold discovered a distinct optimal value for gamma. It is expected that the execution time will increase when the number of folds increases. Shown in Fig.4.4 is the heatmap of hyperparameters tested in our grid search, displaying their impact on the AUC score. We try various combinations of values for each hyperparameter to obtain these results.

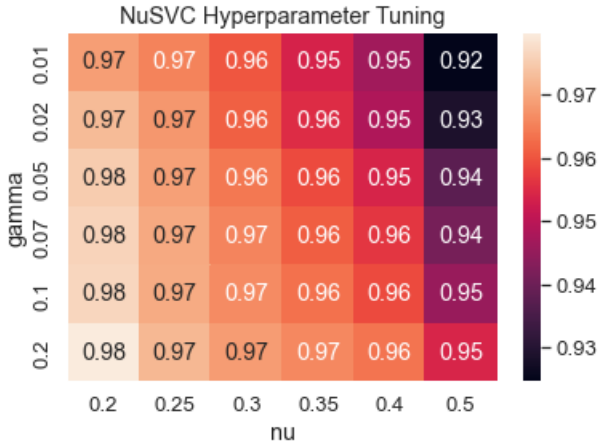

Figure 4.4: NuSVC hyperparameter tuning heatmap

According to the confusion matrix Fig. 4.5 of our NuSVC model, it correctly identifies 472 normal accounts (TN) and 416 illicit accounts (TP). However, it also makes 21 false positive predictions by incorrectly identifying 21 normal accounts as illicit, and 28 false negative predictions by incorrectly identifying 28 illicit accounts as normal.

Folds	nu	gamma	Accuracy	F1-score	AUC	Execution time (sec)
3	0.2	0.07	0.943 (+/- 0.038)	0.943 (+/- 0.007)	**0.97 (+/- 0.007)**	56.928
4	0.2	0.05	0.94 (+/- 0.037)	0.94 (+/- 0.002)	**0.969 (+/- 0.002)**	86.409
5	0.2	0.1	0.945 (+/- 0.038)	0.944 (+/- 0.007)	**0.972 (+/- 0.007)**	113.126
10	0.2	0.2	0.948 (+/- 0.038)	0.948 (+/- 0.005)	**0.978 (+/- 0.005)**	253.193

Table 4.1: Results for 3-, 4-, 5- and 10-fold cross-validation of the NuSVC classifier. Each row represents the optimal parameters of the model along with the respective average performance measures of Accuracy, F1-score, and AUC.

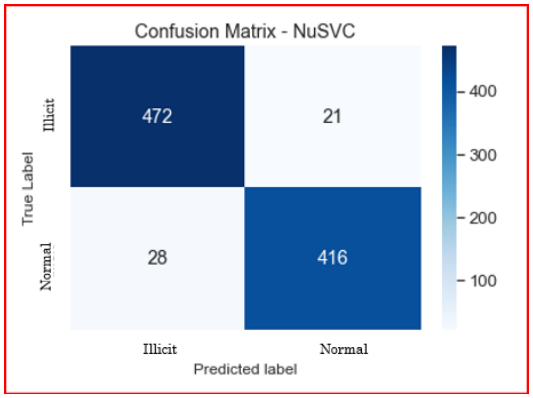

Figure 4.5: Average 10-fold cross validation confusion matrix of NuSVC

4.3.2 Logistic Regression

Two hyperparameters of interest in the proposed Logistic Regression model are"C"and max_iter. The parameter"C"controls the inverse regularization strength, with smaller values of"C"indicating more robust regularization. The parameter max_iter controls the maximum number of iterations for the solvers to converge. We utilize grid search cross-validation to find the optimum value of these two parameters for different numbers of folds. The results in Table 4.2 show that increasing the number of folds does not significantly affect the model's performance. However, we can see a clear improvement in the average AUC using 4-fold, with a highest AUC of **0.91 (+/- 0.009)**. Across all fold sizes, the average accuracy is around 0.878-0.879, an F1-score of around 0.878-0.879. The execution time of the model increases with the number of folds and can range from 13 to 55 seconds. Fig.4.6 shows the box plot of how the F1 score is changed with fold size 4 by testing different values of parameter"C"in our grid search.

Table 4.6 represents the relationship between the hyperparameter "C" and the corresponding F1 scores obtained through a 4-fold grid search cross-validation. As the parameter "C" increases from 0.1 to 1, there is a noticeable improvement in the F1 score, indicating that the model's performance is positively affected by higher values of "C". However, as "C" continues to increase to matters such as 10, 30, and 50, the improvement in the F1 score becomes less significant. This suggests that the model's performance reaches a plateau, and further increases in "C" do not yield substantial gains in the F1 score. These findings highlight the importance of selecting an appropriate value for "C" in logistic regression to achieve optimal performance.

Folds	C	max_iter	Accuracy	F1-score	AUC	Execution time (sec)
3	50	300	0.879 (+/- 0.029)	0.879 (+/- 0.014)	**0.908 (+/- 0.014)**	13.123
4	50	200	0.879 (+/- 0.027)	0.879 (+/- 0.009)	**0.91 (+/- 0.009)**	19.373
5	30	400	0.878 (+/- 0.028)	0.878 (+/- 0.009)	**0.909 (+/- 0.009)**	25.048
10	30	400	0.878 (+/- 0.022)	0.878 (+/- 0.008)	**0.909 (+/- 0.008)**	55.971

Table 4.2: Results for 3-, 4-, 5- and 10-fold cross-validation of the Logistic Regression classifier. Each row represents the optimal parameters of the model along with the respective average performance measures of Accuracy, F1-score, and AUC.

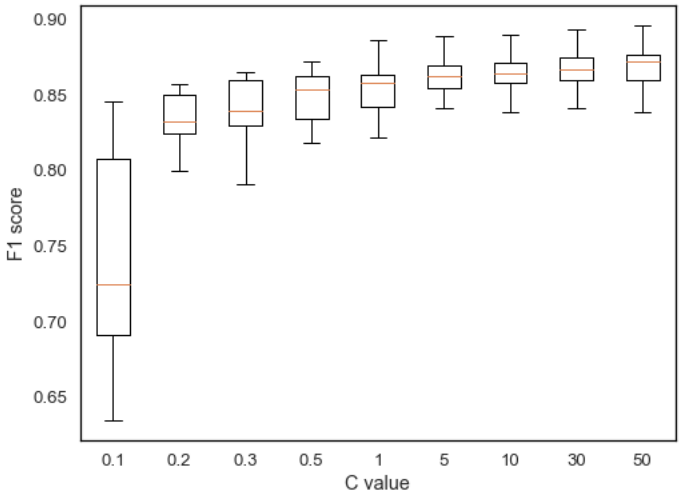

Figure 4.6: Box plot illustrating the relationship between the hyperparameter 'C' in logistic regression and the corresponding F1 scores obtained through 4-fold grid search cross-validation. The F1 score shows notable improvement as 'C' increases from 0.1 to 1, but subsequent increases to values such as 10, 30, and 50 results in less significant improvements.

The confusion matrix of our logistic regression model Fig.4.7 indicates that the model correctly identified 449 normal (TN) and 381 illicit (TP) accounts. The model made 44 false positive predictions, incorrectly identifying 44 normal accounts as illicit. It also made 63 false negative predictions, incorrectly identifying 63 illicit accounts as normal.

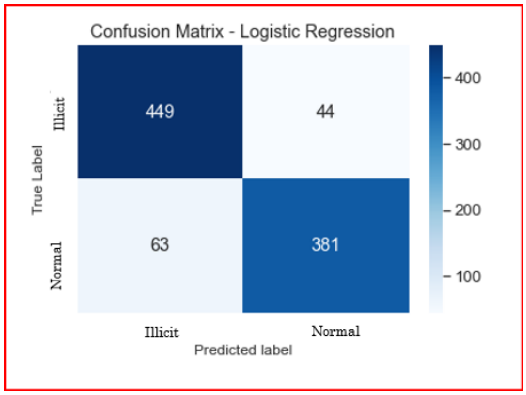

Figure 4.7: Average 10-fold cross validation confusion matrix of logistic regression

4.3.3 Random Forest

The maximum depth (max_depth) is a hyperparameter that controls the depth of each decision tree in the forest, which can prevent overfitting. The number of trees in the forest (n_estimators) is another hyperparameter that determines the number of decision trees to include in the forest and can improve the model's performance by increasing the diversity of the individual trees. We utilize grid search cross-validation to identify the optimal values for our hyperparameters, max_depth, and n_estimators. Furthermore, we rely on AUC evaluation based on the provided parameter values to determine the best model performance. Fig4.8 shows after evaluating over 10-folds, random forest performance grows by higher max_depth and n_estimators.

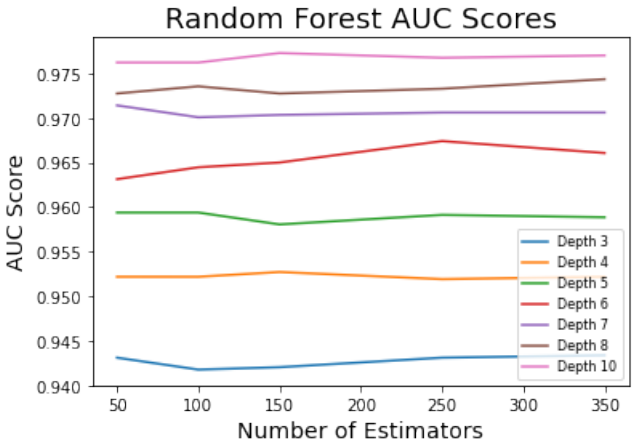

Figure 4.8: Grid-Search 10-fold cross-validation evaluation of n_estimators and max_depth parameters.

From the graph, we can infer that the pink line depicts optimal hyperparameters of the model, with a max_depth of 10 and n_estimators of 150, which produced an AUC score of **0.998**. For these perfect parameters, the equivalent accuracy was **0.978**. It's worth reminding that the graph is

based on the results of cross-validation, meaning that the model was trained and evaluated multiple times with different subsets of the data. The AUC scores are the average of the results from all the iterations, which gives a more robust estimate of the model's performance.

Table 4.3 presents the cross-validation variation results of a random forest model with different splits. The table includes essential performance metrics such as accuracy, F1-score, AUC, and the execution time. All the folds found the same value for best max_depth parameter. The optimal number of estimators varied across the folds, with the highest value of 350 for 3-fold cross-validation and the lowest value of 50 for 4-fold cross-validation. In terms of model performance, the AUC score was consistently high across all folds, ranging from 0.997+/- 0.002 to 0.998 +/- 0.002, with 10-fold cross-validation, achieving an optimum AUC indicating the excellent overall performance of the RF classifier in distinguishing between illicit and non-illicit accounts. Regarding execution time, the RF classifier took longer for higher fold values, with the longest execution time of 508.482 seconds for 10-fold cross-validation. However, the overall execution time remained within reasonable limits, indicating that the RF classifier can be applied efficiently to large datasets.

Folds	**max_depth**	**n_estimators**	**Accuracy**	**F1-score**	**AUC**	**Execution time (sec)**
3	10	350	0.977 (+/- 0.011)	0.976 (+/- 0.001)	**0.998 (+/- 0.001)**	129.324
4	10	50	0.979 (+/- 0.011)	0.979 (+/- 0.002)	**0.997 (+/- 0.002)**	180.115
5	10	150	0.978 (+/- 0.01)	0.978 (+/- 0.001)	**0.998 (+/- 0.001)**	232.974
10	10	150	0.978 (+/- 0.011)	0.978 (+/- 0.002)	**0.998 (+/- 0.002)**	508.482

Table 4.3: Results for 3-, 4-, 5- and 10-fold cross-validation of the Random Forest classifier. Each row represents the optimal parameters of the model along with the respective average performance measures of Accuracy, F1-score, and AUC.

Based on the confusion matrix, the random forest classification model achieved 483 true positives (TP) and 433 true negatives (TN), accurately identifying both normal and illicit accounts in the Ethereum blockchain. However, it also had 10 false negatives (FN), meaning that 10 illicit accounts were misclassified as normal, and 11 false positives (FP), indicating that 11 normal accounts were mistakenly classified as illicit. The precision of the model is 0.98, indicating that it correctly identified 98% of all illicit accounts among its positive predictions. The recall is 0.98 as well, indicating that it correctly identified 98% of all actual illicit accounts. Finally, the F1-score is 0.98, which indicates that the model has a balanced performance between precision and recall. Overall, the random forest classification model performs well in detecting both normal and illicit accounts in the Ethereum blockchain, with high accuracy and a balanced performance between precision and recall.

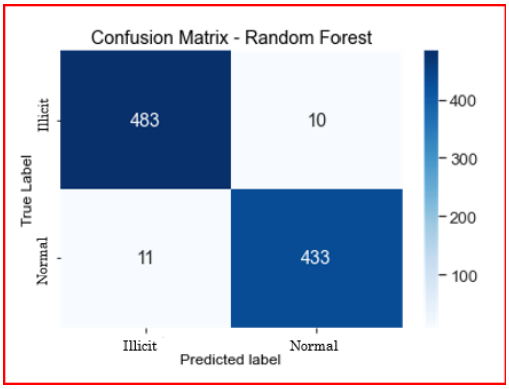

Figure 4.9: Average 10-fold cross-validation confusion matrix of RF model.

4.3.4 Multi-Layer perceptron (MLP)

We employ the same approach to optimize the performance of our Multi-Layer Perceptron (MLP) classifier. Grid search cross-validation is used inside the pipeline to find the ideal values of the relevant hyperparameters, including hidden layer sizes, which determines the number of neurons in each hidden layer, and alpha, which regulates the strength of L_2 regularisation, a method for preventing overfitting by adding a penalty term to the loss function that dissuades the model from giving heavy weights to the features.

Table4.4 presents the results of the MLP classifier using grid search cross-validation with different numbers of folds. Each row shows the optimal hyperparameters of the model, including the learning rate (alpha) and the number of neurons in each hidden layer (h_layer_size). We compare results using the average performance measures of accuracy, F1-score, and AUC, along with the execution time of each fold in seconds. The results show that the model's performance varies slightly across different folds. For example, the 4-fold cross-validation achieved the highest AUC score of **0.995 (+/- 0.001)**, indicating a good ability of the model to distinguish between positive and negative classes. On the other hand, the 10-fold cross-validation achieved the highest accuracy and F1-score of **0.971 (+/- 0.001)**, suggesting that the model can correctly classify the target variable. However, the differences in performance among folds are relatively small, and the model shows consistent results across all folds. The execution time of the model increases with the number of folds, with the 10-fold cross-validation taking the longest time (1050.172 seconds), which is expected since the model needs to fit and evaluate the data multiple times in each fold, and the more folds, the more time-consuming the process becomes. In summary, the MLP classifier using grid search cross-validation achieves consistent and relatively high performance across different folds, with slightly different optimal hyperparameters and performance measures.

Considering all the metrics and variability across the folds, we chose 10-fold as the best fold for this model. Even though the AUC score for fold 10 is not the highest, it is still within the range of the other folds and indicates excellent performance. Furthermore, the higher accuracy and F1-score for fold 10 suggest that the model performs well in terms of discrimination and overall classification accuracy.

Fig4.10 shows the MLP classifier confusion matrix. The model correctly identified 483 normal accounts as normal (true positives) and 427 illicit accounts as illicit (true negatives). However, it misclassified 17 normal accounts as illicit (false positives) and 10 illicit accounts as normal (false negatives). Overall, the model's performance can be considered good, with an accuracy of 95.7%.

Folds	alpha	3 hlayers_size	Accuracy	F1-score	AUC	Execution time (sec)
3	0.001	(300, 150, 100)	0.963 (+/- 0.002)	0.963 (+/- 0.002)	**0.991 (+/- 0.002)**	231.649
4	0.001	(150, 100, 50)	0.964 (+/- 0.001)	0.964 (+/- 0.001)	**0.995 (+/- 0.001)**	341.020
5	0.0001	(300, 150, 100)	0.961 (+/- 0.002)	0.96 (+/- 0.001)	**0.993 (+/- 0.001)**	475.908
10	0.001	(50, 150, 50)	0.971 (+/- 0.001)	0.971 (+/- 0.001)	**0.993 (+/- 0.001)**	1050.172

Table 4.4: Results for 3-, 4-, 5- and 10-fold cross-validation of the MLP classifier. Each row represents the optimal parameters of the model along with the respective average performance measures of Accuracy, F1-score, and AUC.

Comparing the results of the MLP classifier with the random forest classification model, we see that both models correctly identified 483 normal accounts as normal and 10 illicit accounts as illicit. However, the random forest model had a lower false positive rate, with only 11 normal accounts misclassified as illicit compared to the MLP classifier's 17. Based on these results, both models detected normal and illicit accounts well. However, the random forest model performs slightly better regarding false positive rates. It is also worth noting that the MLP classifier had longer execution times than the random forest model.

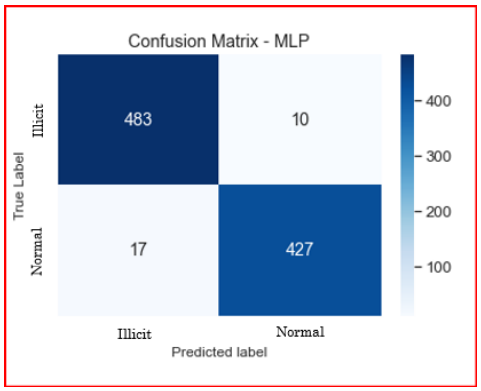

Figure 4.10: Average 10-fold cross-validation confusion matrix of MLP model.

4.4 Comparison of Our Proposed Models

In this thesis, we have trained several models on different folds of our dataset and evaluated their performance using various metrics. To compare the models and determine the most effective, we have decided to opt for the best AUC score of each model. By selecting the best AUC score from each model, we aim to create a fair and unbiased comparison between the models and determine the most optimal for our problem. This approach allows us to focus on the most important metric for our task and identify the model with the highest predictive power, ultimately leading to better results and insights. Table 4.5 indicates the best scores of four proposed machine learning models, including Accuracy, F1-score, AUC, and Execution time for their respective optimal hyperparameters and cross-validation folds. Based on the evaluation metrics and execution time, RF and MLP are the best algorithms for this classification task. It is evident that RF takes the longest time to execute compared to other algorithms. However, despite having the shortest execution time, Logistic Regression achieved the lowest accuracy, F1 score, and AUC score, suggesting that there might be

better options for this task. Overall, the results indicate that RF and MLP are excellent options for the classification task, depending on the specific trade-offs between accuracy and execution time.

Methods	Folds	Accuracy	F1-score	AUC	Execution time (sec)
NuSVC	10	0.948 (+/- 0.038)	0.948 (+/- 0.005)	**0.978 (+/- 0.005)**	253.193
Logistic Reg	4	0.879 (+/- 0.027)	0.879 (+/- 0.009)	**0.91 (+/- 0.009**	19.373
RF	10	0.978 (+/- 0.011)	0.978 (+/- 0.002)	**0.998 (+/- 0.002)**	508.482
MLP	4	0.964 (+/- 0.001)	0.964 (+/- 0.001)	**0.995 (+/- 0.001)**	341.020

Table 4.5: comparison of our four proposed ML models by their highest AUC score

In Figure 4.11, we visualize our proposed models' results to better understand each model's performance. The radar chart indicates that the random forest model outperformed all other models in every metric. In contrast, the logistic regression model scored the lowest and was noticeably distant from the other methods. Additionally, the RF model has the highest execution time and significantly differed from the other models.

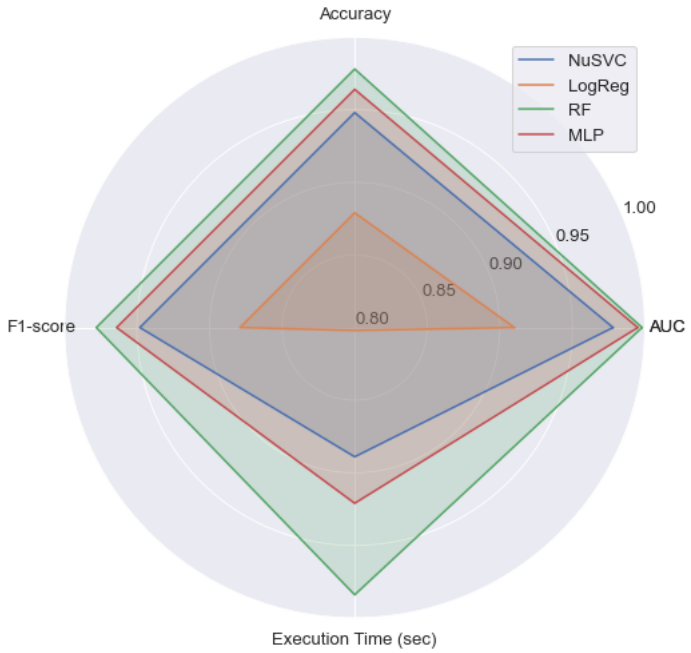

Figure 4.11: Visualization of the proposed model's performance using various metrics with normalized Execution time for clarity.

4.5 Feature Importance

We utilize the RFECV method to rank important features and determine the optimal number of features for optimal performance. The results are displayed in Table 4.6, where the model identified 17 features that provide nearly all the necessary information from our dataset.

	Index	Feature's Name	Rank
0	1	Avg_min_between_received_tnx	1
1	2	Time_Diff_between_first_and_last_(Mins)	2
2	4	Sent_tnx	3
3	6	Received_Tnx	4
4	10	Unique_Received_From_Addresses	5
5	17	total_transactions_(including_tnx_to_create_contract)	6
6	19	total_ether_received	7
7	21	total_ether_balance	8
8	22	Total_ERC20_tnxs	9
9	23	ERC20_total_Ether_received	10
10	27	ERC20_uniq_rec_addr	11
11	28	ERC20_uniq_rec_contract_addr	12
12	32	ERC20_min_val_rec	13
13	33	ERC20_max_val_rec	14
14	34	ERC20_avg_val_rec	15
15	39	ERC20_uniq_rec_token_name	16
16	41	ERC20_rec_token_type_encoded	17

Table 4.6: Feature selection by RFECV method using 10-fold cross-validation.

The RFECV method identified the most important features as "Avg_min_between_received_tnx" followed by "Time_Diff_between_first_and_last_(Mins)" and "Sent_tnx." These results suggest that time-related factors has a significant impact on the dataset. Conversely, "ERC20_rec_token_type_encoded" was found to be the least significant feature. Figure 4.12 depicts the association between the number of features and the accuracy achieved through Recursive Feature Elimination with Cross-Validation (RFECV). The graph shows that the model's accuracy gradually improves as more features are added, peaking at 17 features. Subsequently, the accuracy experiences slight fluctuations, indicating that the model attains its highest performance after incorporating these 17 features. This demonstrates the significance of feature selection in optimizing the model's accuracy.

The RF-FI method ranks features based on their average decrease in impurity level, which provides a measure of their importance in the model. However, this method does not directly provide the number of important features to select. Therefore, a threshold value is typically set to determine the number of most important features to select. In our thesis, we use a threshold value of 0.015 to select the most important features and have listed them in Table. 4.7. This approach allows us to focus on the features that have the highest impact on the model's performance while reducing the complexity and redundancy of the feature set.

The RF-FI method identifies the most important feature as 'ERC20_min_val_rec' with the contribution level to the overall predictive power of the Random Forest model of **0.134** followed by 'ERC20_rec_token_type_encoded' and 'Total_ERC20_tnxs' with significant contribution of 0.092 and 0.086 on average. The rest of the features are sorted in descending order. The results show that the proposed RF-FI method concentrates more on ERC20 token transactions, unlike RFECV, which was focused more on time-related features.

Fig.4.13 and Fig4.14 provide better understanding of the performance of these two methods by visualizing their results. Both methods shared 13 features and has four features that are unique to them. We represents the unique features to each set by the color "yellow".

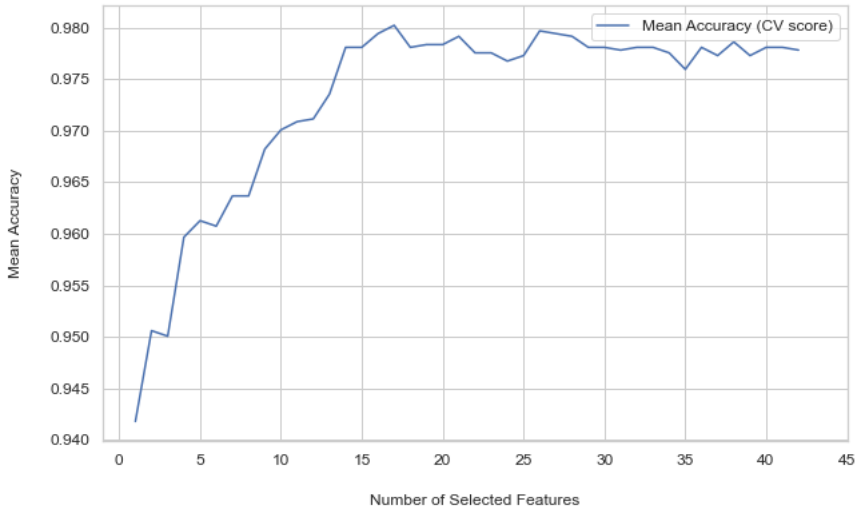

Figure 4.12: Relationship between the number of features and the mean accuracy using RFECV. The graph demonstrates the incremental improvement in accuracy as more features are added, reaching its peak at 17 features. Subsequent fluctuations indicate attaining the highest accuracy after incorporating these 17 features.

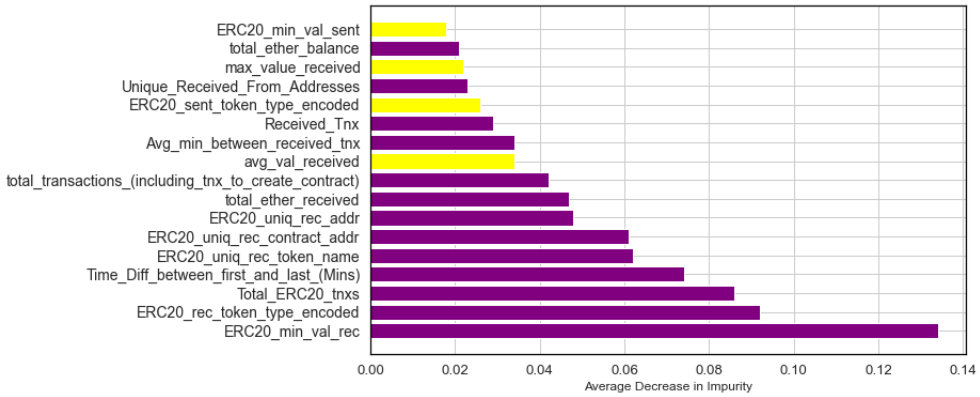

Figure 4.13: Rank of the most important features determined by RF-FI with respect to ”Contribution to the overall predictive power of the Random Forest model” and the threshold value of 0.015. The frequency defines the average accumulation of the impurity decrease within each tree. Yellow color bars represents the unique features selected by RF-FI model.

	Index	Feature's Name	FI
0	32	ERC20_min_val_rec	0.134
1	41	ERC20_rec_token_type_encoded	0.092
2	22	Total_ERC20_tnxs	0.086
3	2	Time_Diff_between_first_and_last_(Mins)	0.074
4	28	ERC20_uniq_rec_contract_addr	0.062
5	39	ERC20_uniq_rec_token_name	0.061
6	27	ERC20_uniq_rec_addr	0.048
7	19	total_ether_received	0.047
8	17	total_transactions_(including_tnx_to_create_contract)	0.042
9	10	avg_val_received	0.034
10	1	Avg_min_between_received_tnx	0.034
11	4	Received_Tnx	0.029
12	40	ERC20_sent_token_type_encoded	0.026
13	6	Unique_Received_From_Addresses	0.023
14	9	max_value_received	0.022
15	21	total_ether_balance	0.021
16	35	ERC20_min_val_sent	0.018

Table 4.7: List of important features selected by RF-FI method with a threshold value of "0.015".

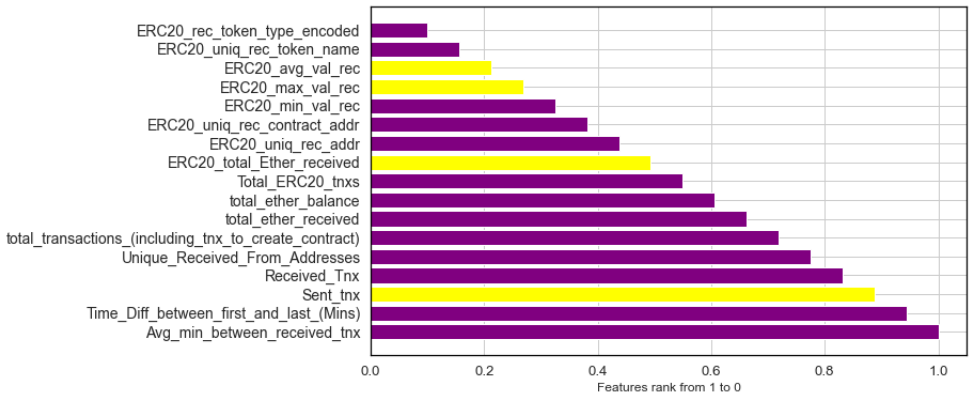

Figure 4.14: Ranking of Important Features using RFECV Method. In order to have a better visualization, we scaled rank features from 1 to 0. Yellow color bars represents the unique features selected by RFECV model.

Based on the RF-FI and RFECV results, it can be concluded that the time-related features such as 'Avg_min_between_received_tnx' and 'Time_Diff_between_first_and_last_(Mins)' are among the most important features for predicting the outcome variable. Moreover, features related to ERC20 token transactions, such as 'ERC20_min_val_rec' and 'ERC20_rec_token_type_encoded', are also important.

Both methods shared 13 features and has four features that are unique to them. Features that are only present in the RF-FI method are 'max_value_received', 'ERC20_sent_token_type_encoded', 'ERC20_min_val_sent', and 'avg_val_received '. Features that are only present in the RFECV method are 'ERC20_total_Ether_received', 'ERC20_max_val_rec', 'ERC20_avg_val_rec', and 'Sent_tnx'.

4.5.1 Retrain Models with New Feature Sets

As part of our contribution to this thesis project, we developed two new sets of features using two different selection techniques introduced in the previous section. We then used these new sets to retrain our models and analyze the results. The first list refers to as ”16-set”, which combined the outcomes of both techniques, resulting in 21 features - 13 shared and four unique to each set.

As previously discussed, some of the features have correlation with each other. In Fig4.15, we represented the correlation between these features in the ”16-set”. After eliminating these features with a correlation degree of 0.8 or higher we were left with 16 independent features in the ”16-set”, which are a combination of the most important features achieved by both feature selection methods without their correlated counterparts. The table 4.8 lists these 16 features.

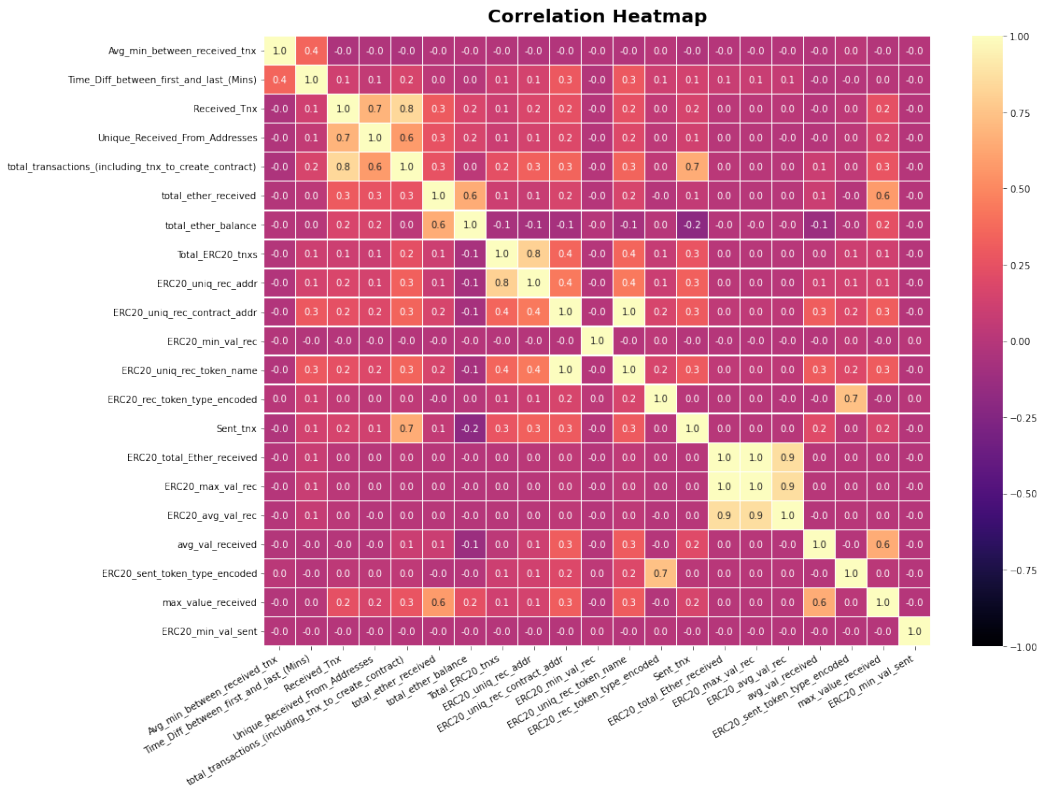

Figure 4.15: The correlation heatmap of 21 features extracted from the two feature selection methods.

In order to further reduce the dimensionality of our dataset, we chose the three most important features from each approach as the second step. This new collection of features is referred to as ”6-set” Fig.4.16 shows the correlation heatmap of this new set of features.

Index	Feature's Name
1	Avg_min_between_received_tnx
2	Time_Diff_between_first_and_last_(Mins)
3	Sent_tnx
4	Received_Tnx
5	Unique_Received_From_Addresses
6	max_value_received
7	total_ether_received
8	total_ether_balance
9	Total_ERC20_tnxs
10	ERC20_total_Ether_received
11	ERC20_min_val_sent
12	ERC20_uniq_rec_contract_addr
13	ERC20_min_val_rec
14	avg_val_received
15	ERC20_sent_token_type_encoded
16	ERC20_rec_token_type_encoded

Table 4.8: 16-set: the summation of significant features identified by the RFECV and RF-FI methods after removing correlated features.

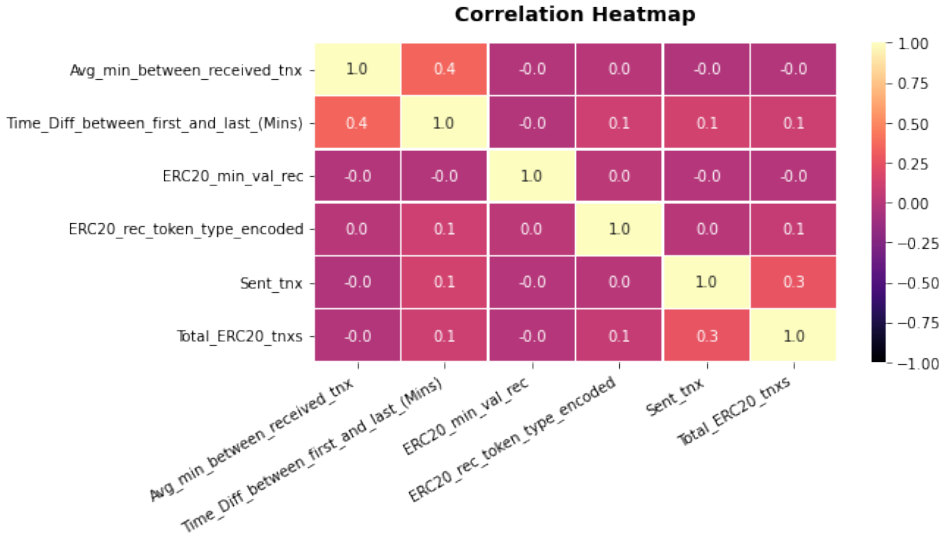

Figure 4.16: The correlation heatmap of 6 features extracted from the two feature selection methods.

As we can see from the Fig.4.16, there is no correlation between these features. A complete list of "6-set" was depicted by table 4.9. Finally, We selected two proposed models - RF and MLP - for retraining our models due to their better performance. To ensure a fair comparison, we chose the number of folds for each model that produced the highest AUC score values. We then compared the outcomes of the two new sets of features with the original set with 42 features. The comparison

Index	Feature's Name
1	Avg_min_between_received_tnx
2	Time_Diff_between_first_and_last_(Mins)
3	ERC20_min_val_rec
4	ERC20_rec_token_type_encoded
5	Sent_tnx
6	Total_ERC20_tnxs

Table 4.9: 6-set: the summation of the top three significant features identified by the RFECV and RF-FI.

results for our RF and MLP models are shown in Table 4.10 and Table 4.11, respectively, using the same procedure to retrain our models.

n_features	max_depth	n_estimators	Accuracy	F1-score	AUC	Execution time (sec)
42	10	150	0.978 (+/- 0.011)	0.978 (+/- 0.002)	**0.998 (+/- 0.002)**	508.482
16	10	100	0.979 (+/- 0.003)	0.979 (+/- 0.001)	**0.998 (+/- 0.001)**	482.880
6	10	100	0.972 (+/- 0.004)	0.972 (+/- 0.001)	**0.996 (+/- 0.001)**	350.976

Table 4.10: Comparison of the Random Forest classifier results with the original 42 features and applying our two new feature sets using 10-fold cross validation. 'n_features' shows the number of features used in each train and test implementation.

The results presented in Table 4.10 and Table 4.11 indicate that our new feature sets did not significantly impact the evaluation metrics of our proposed RF and MLP models. This demonstrates that we successfully identified the most informative features within our dataset. Specifically, the "16-set" yielded slightly better accuracy and F1 score in both classifiers than the original feature set, while the AUC score remained consistent. On the other hand, the "6-set" had the lowest execution time without a substantial decrease in evaluation metrics, suggesting that it may be the optimal choice in terms of balancing execution time and accuracy. In summary, selecting an appropriate feature set is crucial when building a classification model, considering both performance and execution time respects.

n_features	alpha	3 hlayers_size	Accuracy	F1-score	AUC	Execution time (sec)
42	0.001	(150, 100, 50)	0.964 (+/- 0.001)	0.964 (+/- 0.001)	**0.995 (+/- 0.001)**	341.020
16	0.001	(50, 150, 50)	0.975 (+/- 0.002)	0.975 (+/- 0.002)	**0.987 (+/- 0.002)**	334.759
6	0.0001	(150, 100, 50)	0.955 (+/- 0.002)	0.955 (+/- 0.001)	**0.98 (+/- 0.001)**	274.120

Table 4.11: Comparison of the MLP classifier results with the original 42 features and applying our two new feature sets using 4-fold cross validation. 'n_features' shows the number of features used in each train and test implementation.

4.6 Comparison with Other Works

This section begins by evaluating the performance of the proposed models against the XGBoost classifier, which was previously used by Farrugia et al. [6] in their study. The authors originally applied the XGBoost classifier to identify illicit accounts in the Ethereum blockchain. We employed the same comprehensive dataset created by the authors and used it in their work for the first time to

conduct this comparison. This comparison allows us to assess our proposed models' effectiveness and ability to detect similar fraudulent activities in the same dataset.

The data in Table 4.12 indicates that our RF and MLP models performed better than the XGBoost model in all evaluation metrics. While the NuSVC model had reasonable results, the logistic regression model stood out from the other models. Additionally, the XGBoost model had a better execution time than our proposed models, except for logistic regression.

Methods	Folds	Accuracy	F1-score	AUC	Execution time (sec)
Proposed NuSVC	10	0.948 (+/- 0.038)	0.948 (+/- 0.005)	**0.978 (+/- 0.005)**	253.193
Proposed LogReg	4	0.879 (+/- 0.027)	0.879 (+/- 0.009)	**0.91 (+/- 0.009**	19.373
Proposed RF	10	0.978 (+/- 0.011)	0.978 (+/- 0.002)	**0.998 (+/- 0.002)**	508.482
Proposed MLP	4	0.964 (+/- 0.001)	0.964 (+/- 0.001)	**0.995 (+/- 0.001)**	341.020
Farrgia XGBoost	10	0.963 (+/- 0.006)	0.960 (+/- 0.006)	**0.994 (+/- 0.0007)**	230.60

Table 4.12: Comparison of our four proposed ML models by the XGBoost model proposed by Farrugia et al. [6]

As a means of ensuring the comprehensive and thorough nature of our thesis, we have taken steps to further enhance our evaluation of classification models by introducing a new dataset, in addition to the well-designed and comprehensive dataset utilized in this thesis which was obtained from various sources, as detailed in previous sections. This new dataset has been included to augment the evaluation of our classification models further and provide an even more detailed analysis of their performance.

For this purpose, we obtained the dataset from the Kaggle website [80]. This new dataset includes 9841 Ethereum transactions. It specifically consists of 2179 fraud transactions and 7662 non-fraud transactions and comprises 51 features, which are made up of several detailed transactions. Since this new dataset is larger in size, it helps us to ensure that our models are accurate and reliable on the original dataset and capable of generalizing well to new and unseen data. It also allows us to investigate whether our models overfit or underfit on the original dataset. This expansion of our thesis will provide valuable insights into our models' performance and our approach's robustness.

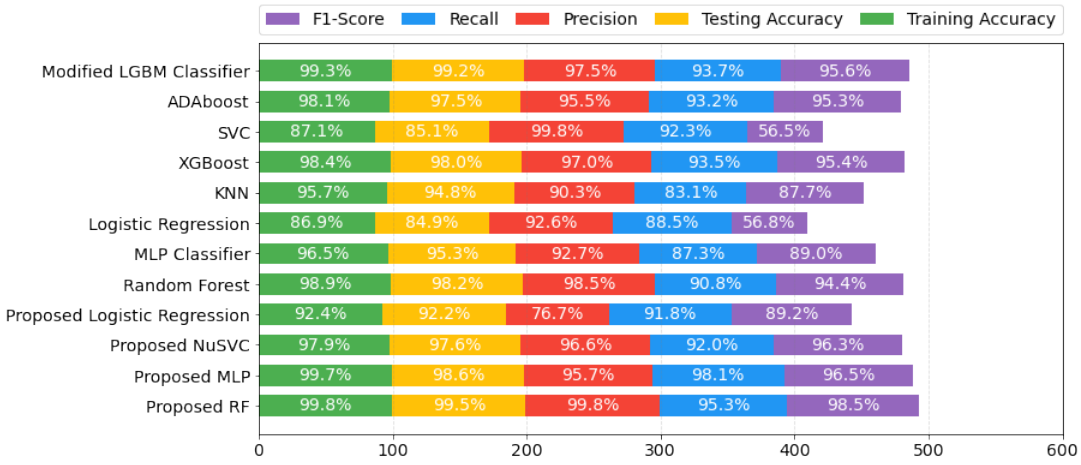

Figure 4.17: Performance comparison of the proposed models to the ones conducted by Aziz et al. [44] using Training Accuracy, Testing Accuracy, Recall, Precision and F1 score.

We trained and tested our models with the new dataset following the same methodology as our previous implementations. It is noteworthy that this dataset was old enough; therefore, more previous studies had already been conducted on it, which provided a better basis for comparison with our results. In contrast, the primary dataset utilized in our thesis was relatively new, and no similar works were found that could provide a benchmark for our classification models.

In their study, Aziz et al. [44] utilized this dataset to implement their models. Their approach involved modifying the Light Gradient Boosting Machine (LGBM) classifier and comparing its performance with well-known techniques such as k-Nearest Neighbors (KNN), XGBoost, Support Vector Classification (SVC), Logistic Regression, and AdaBoost. As shown in Figure 4.17, our proposed models, particularly the proposed RF model, demonstrated superior performance compared to Aziz's modified LGBM model and the other classification models. The proposed MLP model also performed well, and after the proposed RF and Aziz's modified LGBM, it was the best classifier in terms of the evaluation metrics presented.

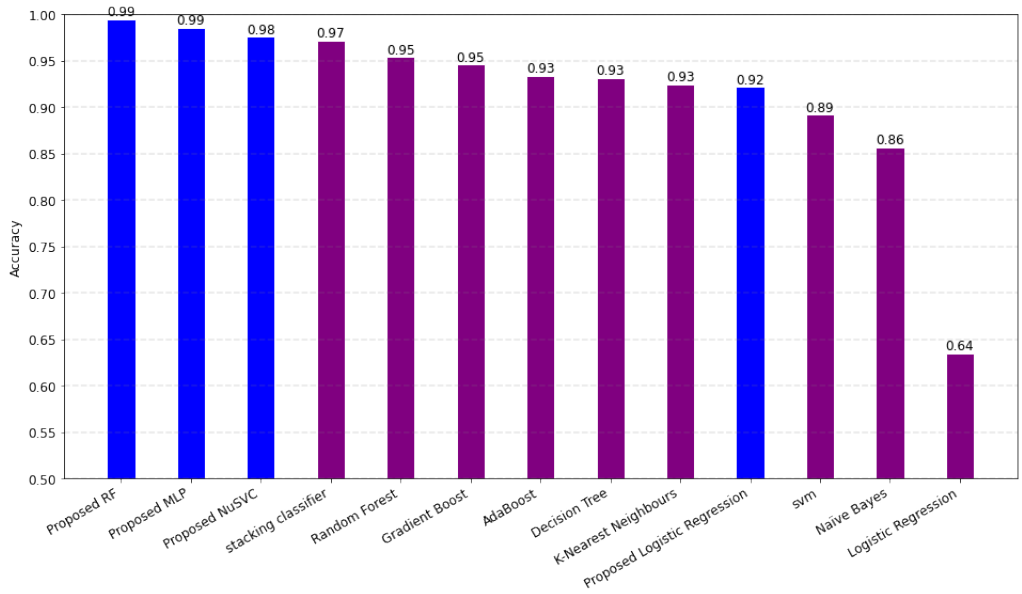

Figure 4.18: Comparison of the accuracy score obtained by our proposed models with the ones implemented by Quadir et al. [30] in their study. The blue bars represent our proposed models while the purple bars represents Quadir's results. We sorted the bar chart to have a better comparison visually. Our proposed Random Forest model achieved 0.99 accuracy score following by MLP model 0.99 and NuSVC 0.98 while the best accuracy achieved by Quadir et al. was 0.97 for their proposed Staking Classifier.

Abdul Quadir et al. [30] also utilized this dataset to conduct their study. They present a new method for identifying fraudulent accounts connected to transactions on the Ethereum network. The strategy involves utilizing various machine learning algorithms such as Logistic Regression, Naive Bayes, Decision Trees, Random Forests, AdaBoosts, KNNs, SVMs, and Gradient Boosts. They also develop a stacking classifier that combines multiple independent classification algorithms and a meta-learner based on the outcome of each base algorithm.

Fig. 4.18 illustrates a comparison of our proposed models with those presented in the Quadir et al. [30], based on their accuracy scores. Our results demonstrate that our Random Forest, MLP, and NuSVC models outperform all the models proposed by the authors. Although our proposed logistic regression model ranks towards the lower end of the chart with an accuracy score of 0.92, it still surpasses the accuracy score of the referenced study's proposed logistic regression model, which only achieved a score of 0.64. Therefore, our proposed models are more effective and reliable for detecting fraudulent transactions than the methods proposed in the referenced study.

Chapter 5

CONCLUSION AND FUTURE WORKS

Our method for detecting illicit accounts on the Ethereum network involves a combination of ideas and techniques from similar works, including fine-tuning models and feature selection. Using four supervised machine learning models including NuSVC, Logistic Regression, Random Forest, and Multilayer Perceptron, we classified accounts based on their Ethereum blockchain activity. Two different datasets, one balanced and one imbalanced, containing normal and illicit accounts, were used to evaluate the models. The Random Forest model demonstrated exceptional effectiveness, achieving an average AUC score of 0.998 with a standard deviation of 0.002, while the Multilayer Perceptron classifier followed closely with an average AUC score of 0.995 and a standard deviation of 0.001. The models exhibit excellent generalization abilities without overfitting, evident from the low standard deviations in both AUC and accuracy. Additionally, the results from a smaller number of k-folds used for cross-validation further support this conclusion.

In addition, this thesis introduces a novel feature selection approach that combines Recursive Feature Elimination Cross-Validation and Random Forest Feature Importance to identify the most crucial features. Additionally, we created two new feature sets to enhance efficiency by combining the most essential features from both methods. By doing so, we identified six informative features, including 'Average minutes between received transactions,' 'Time Difference between the first and last transaction (Minutes),' 'Minimum value of received ERC20 transactions,' 'Encoded type of received ERC20 tokens,' 'Number of sent transactions,' and 'Total number of ERC20 transactions.', that significantly reduced execution time without a significant decrease in accuracy. The results also revealed the crucial influence of time-related features. Notably, the time duration between the first and last transaction significantly impacted the account classification. We observed that normal accounts have an average duration of approximately 136.8 days, while illicit accounts have a much shorter average duration of around 38.3 days. This finding indicates that illicit accounts might be created solely to carry out illegal activities, as their usage periods are notably shorter than legitimate accounts. The average total ERC20 transactions of normal accounts were also approximately 237.42% higher than illicit accounts. This suggests that illicit accounts conduct fewer ERC20 transactions, possibly due to their involvement in specific illicit activities that do not require frequent transactions or to avoid detection. In contrast, normal accounts, representing legitimate users, engage in more frequent ERC20 transactions for various purposes, such as legitimate trading, DeFi activities, or blockchain-based applications.

Despite the substantial contributions made in this thesis, it is paramount to acknowledge limitations and challenges encountered. A significant limitation of this thesis was finding a suitable dataset for classifying accounts on the Ethereum blockchain. Creating a novel dataset was not feasible within the scope of this thesis due to the significant resources and complex process involved, which is time-consuming. Additionally, the scarcity of relevant Ethereum datasets aligned with our thesis's objectives limited the depth and breadth of our analysis. It is important to consider these limitations when interpreting the research findings. In future studies, addressing these limitations may be possible by exploring alternative approaches to dataset creation or collaborating with other researchers to develop more comprehensive datasets.

In light of these limitations, several avenues for future research are worth exploring. Researchers may investigate the feasibility of adapting the proposed approach to detecting illicit activities on other blockchain networks, such as Bitcoin, Cardano, Polkadot, and Solana. Moreover, developing this methodology in traditional (fiat) currencies and financial transactions could be beneficial in combating financial crimes.

One may also look into developing and implementing a tool that incorporates our approach to flag suspicious accounts in cryptocurrency exchanges and other financial platforms. By providing alerts to users when interacting with such accounts, this proactive measure would significantly enhance the security and trustworthiness of these platforms.Another possibility would be testing and developing other machine learning models along with suitable feature selection methods to reduce the execution time of the models without significantly lowering the model's accuracy.

Bibliography

[1] Jennifer J Xu. Are blockchains immune to all malicious attacks? *Financial Innovation*, 2(1):1–9, 2016.

[2] Dylan Yaga, Peter Mell, Nik Roby, and Karen Scarfone. Blockchain technology overview. *arXiv preprint arXiv:1906.11078*, 2019.

[3] Chainalysis, url : https://www.chainalysis.com/.

[4] Shuai Wang, Liwei Ouyang, Yong Yuan, Xiaochun Ni, Xuan Han, and Fei-Yue Wang. Blockchain-enabled smart contracts: Architecture, applications, and future trends. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 49(11):2266–2277, 2019.

[5] Agarwal R Barve S Shukla SK. Detecting malicious accounts in permissionless blockchains using temporal graph properties appl. *Network Sci*, 6(1):1, 2021.

[6] Steven Farrugia, Joshua Ellul, and George Azzopardi. Detection of illicit accounts over the ethereum blockchain. *Expert Systems with Applications*, 150:113318, 2020.

[7] Yuanfeng Cai and Dan Zhu. Fraud detections for online businesses: a perspective from blockchain technology. *Financial Innovation*, 2:1–10, 2016.

[8] Sheraz Naseer, Yasir Saleem, Shehzad Khalid, Muhammad Khawar Bashir, Jihun Han, Muhammad Munwar Iqbal, and Kijun Han. Enhanced network anomaly detection based on deep neural networks. *IEEE Access*, 6:48231–48246, 2018.

[9] Bruno Mazorra, Victor Adan, and Vanesa Daza. Do not rug on me: Zero-dimensional scam detection. https://arxiv.org/abs/2201.07220, 2022.

[10] Ethereum Daily Transactions Chart, 2023, url : https://etherscan.io/chart/tx.

[11] Teng Hu, Xiaolei Liu, Ting Chen, Xiaosong Zhang, Xiaoming Huang, Weina Niu, Jiazhong Lu, Kun Zhou, and Yuan Liu. Transaction-based classification and detection approach for ethereum smart contract. *Information Processing & Management*, 58(2):102462, 2021.

[12] Arkan Hammoodi Hasan Kabla, Mohammed Anbar, Selvakumar Manickam, Taief Alaa Al-Amiedy, Peterson Bernabe Cruspe, Ahmed K. Al-Ani, and Shankar Karuppayah. Applicability of intrusion detection system on ethereum attacks: A comprehensive review. *IEEE Access*, 10:71632–71655, 2022.

[13] Thai Pham and Steven Lee. Anomaly detection in bitcoin network using unsupervised learning methods. *arXiv preprint arXiv:1611.03941*, 2016.

[14] steven farrugia. Classification of illicit activity over the ethereum network using xgboost. Master's thesis, University of Groningen, 2019.

[15] Cryptocurrencies, Digital Dollars, and the Future of Money, 2023, url : https://www.cfr.org/backgrounder/cryptocurrencies-digital-dollars-and-future-money.

[16] 3 ways digital currencies could change global trade, 2022, url : https://www.weforum.org/agenda/2022/01/digital-currencies-international-trade/.

[17] Revin Naufal Alief, Syifa Maliah Rachmawati, Muhammad Rasyid Redha Ansori, Jae-Min Lee, and Dong-Seong Kim. Cnn-based fraud account classification for ethereum blockchain's accounts. *amazonaws*, pages 690–691, 2022.

[18] Baran Kılıç, Alper Sen, and Can Özturan. Fraud detection in blockchains using machine learning. In *2022 Fourth International Conference on Blockchain Computing and Applications (BCCA)*, pages 214–218, 2022.

[19] The Chainalysis 2022 Crypto Crime Report, 2022, url : https://go.chainalysis.com/2022-crypto-crime-report.html.

[20] The Chainalysis 2022 Crypto Crime Report, 2023, url : https://go.chainalysis.com/2023-crypto-crime-report.html.

[21] Amel Bella Baci, Kei Brousmiche, Ilias Amal, Fatma Abdelhédi, and Lionel Rigaud. Detecting illicit ethereum accounts based on their transaction history and properties and using machine learning. In *The International Conference on Deep Learning, Big Data and Blockchain (DBB 2022)*, pages 97–108. Springer, 2022.

[22] Michał Ostapowicz and Kamil Żbikowski. Detecting fraudulent accounts on blockchain: a supervised approach. In *International Conference on Web Information Systems Engineering*, pages 18–31. Springer, 2020.

[23] Patrick Monamo, Vukosi Marivate, and Bheki Twala. Unsupervised learning for robust bitcoin fraud detection. In *2016 Information Security for South Africa (ISSA)*, pages 129–134. IEEE, 2016.

[24] Ji Li, Chunxiang Gu, Fushan Wei, and Xi Chen. A survey on blockchain anomaly detection using data mining techniques. In *Blockchain and Trustworthy Systems: First International Conference, BlockSys 2019, Guangzhou, China, December 7–8, 2019, Proceedings 1*, pages 491–504. Springer, 2020.

[25] Massimo Bartoletti, Salvatore Carta, Tiziana Cimoli, and Roberto Saia. Dissecting ponzi schemes on ethereum: identification, analysis, and impact. *Future Generation Computer Systems*, 102:259–277, 2020.

[26] Qi Yuan, Baoying Huang, Jie Zhang, Jiajing Wu, Haonan Zhang, and Xi Zhang. Detecting phishing scams on ethereum based on transaction records. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2020.

[27] Rabia Musheer Aziz, Mohammed Farhan Baluch, Sarthak Patel, and Abdul Hamid Ganie. Lgbm: a machine learning approach for ethereum fraud detection. *International Journal of Information Technology*, pages 1–11, 2022.

[28] Rabia Musheer Aziz, Rajul Mahto, Kartik Goel, Aryan Das, Pavan Kumar, and Akash Saxena. Modified genetic algorithm with deep learning for fraud transactions of ethereum smart contract. *Applied Sciences*, 13(2):697, 2023.

[29] Panpan Li, Yunyi Xie, Xinyao Xu, Jiajun Zhou, and Qi Xuan. Phishing fraud detection on ethereum using graph neural network. In *Blockchain and Trustworthy Systems: 4th International Conference, BlockSys 2022, Chengdu, China, August 4–5, 2022, Revised Selected Papers*, pages 362–375. Springer, 2022.

[30] Abdul Quadir Md, SM Satya Sree Narayanan, H Sabireen, Arun Kumar Sivaraman, and Kong Fah Tee. A novel approach to detect fraud in ethereum transactions using stacking. *Expert Systems*, page e13255, 2023.

[31] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton University Press, 2016.

[32] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.

[33] Michael Crosby, Pradan Pattanayak, Sanjeev Verma, Vignesh Kalyanaraman, et al. Blockchain technology: Beyond bitcoin. *Applied Innovation*, 2(6-10):71, 2016.

[34] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. A survey of attacks on ethereum smart contracts (sok). In *International conference on principles of security and trust*, pages 164–186. Springer, 2017.

[35] Tomaso Aste, Paolo Tasca, and Tiziana Di Matteo. Blockchain technologies: The foreseeable impact on society and industry. *discovery.ucl.ac.uk*, 2017.

[36] Zexi Xing and Zhengxin Chen. A protecting mechanism against double spending attack in blockchain systems. In *2021 IEEE World AI IoT Congress (AIIoT)*, pages 0391–0396, 2021.

[37] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 3(37):2–1, 2014.

[38] Chris Dannen. *Introducing Ethereum and solidity*, volume 1. Springer, 2017.

[39] Nick Szabo. Smart contracts http://www. fon. hum. uva. nl/rob/courses/informationinspeech/cdrom/literature. *LOTwinterschool2006/szabo. best. vwh. net/smart. contracts. html*, 1994.

[40] Chibuzor Udokwu, Aleksandr Kormiltsyn, Kondwani Thangalimodzi, and Alex Norta. The state of the art for blockchain-enabled smart-contract applications in the organization. In *2018 Ivannikov Ispras Open Conference (ISPRAS)*, pages 137–144, 2018.

[41] Rahmeh Fawaz Ibrahim, Aseel Mohammad Elian, and Mohammed Ababneh. Illicit account detection in the ethereum blockchain using machine learning. In *2021 International Conference on Information Technology (ICIT)*, pages 488–493. IEEE, 2021.

[42] Cong T. Nguyen, Dinh Thai Hoang, Diep N. Nguyen, Dusit Niyato, Huynh Tuong Nguyen, and Eryk Dutkiewicz. Proof-of-stake consensus mechanisms for future blockchain networks: Fundamentals, applications and opportunities. *IEEE Access*, 7:85727–85745, 2019.

[43] Meng Li. A survey on ethereum illicit detection. In *International Conference on Artificial Intelligence and Security*, pages 222–232. Springer, 2022.

[44] Rabia Musheer Aziz, Mohammed Farhan Baluch, Sarthak Patel, and Pavan Kumar. A machine learning based approach to detect the ethereum fraud transactions with limited attributes. *Karbala Int. J. Mod. Sci*, 8:139–151, 2022.

[45] Vladimir N Vapnik. An overview of statistical learning theory. *IEEE transactions on neural networks*, 10(5):988–999, 1999.

[46] Arpit Aggarwal, Tanvi Sahay, Annu Bansal, and Mahesh Chandra. Grid search analysis of nu-svc for text-dependent speaker-identification. In *2015 Annual IEEE India Conference (INDICON)*, pages 1–5, 2015.

[47] KE Manjunath, SB Sunil Kumar, Debadatta Pati, Biswajit Satapathy, and K Sreenivasa Rao. Development of consonant-vowel recognition systems for indian languages: Bengali and odia. In *2013 Annual IEEE India Conference (INDICON)*, pages 1–6. IEEE, 2013.

[48] Bernhard Schölkopf, Alex J Smola, Robert C Williamson, and Peter L Bartlett. New support vector algorithms. *Neural computation*, 12(5):1207–1245, 2000.

[49] Michael P LaValley. Logistic regression. *Circulation*, 117(18):2395–2399, 2008.

[50] Frank E Harrell, Jr and Frank E Harrell. Binary logistic regression. *Regression modeling strategies: With applications to linear models, logistic and ordinal regression, and survival analysis*, pages 219–274, 2015.

[51] Introduction to Logistic Regression, 2019, url : https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148.

[52] A Mkhadri, G Celeux, and A Nasroallah. Regularization in discriminant analysis: an overview. *Computational Statistics & Data Analysis*, 23(3):403–423, 1997.

[53] Fariborz Salehi, Ehsan Abbasi, and Babak Hassibi. The impact of regularization on high-dimensional logistic regression. *Advances in Neural Information Processing Systems*, 32, 2019.

[54] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[55] Random forest vs Desicion tree, 2022, url : https://www.analyticsvidhya.com/blog/2020/05/decision-tree-vs-random-forest-algorithm/.

[56] Adele Cutler, D Richard Cutler, and John R Stevens. Random forests. *Ensemble machine learning: Methods and applications*, pages 157–175, 2012.

[57] Cha Zhang and Yunqian Ma. *Ensemble machine learning: methods and applications*. Springer, 2012.

[58] Siddhartha Bhattacharyya, Sanjeev Jha, Kurian Tharakunnel, and J Christopher Westland. Data mining for credit card fraud: A comparative study. *Decision support systems*, 50(3):602–613, 2011.

[59] Osval Antonio Montesinos López, Abelardo Montesinos López, and Jose Crossa. Fundamentals of artificial neural networks and deep learning. In *Multivariate Statistical Machine Learning Methods for Genomic Prediction*, pages 379–425. Springer, 2022.

[60] Binjie Chen, Fushan Wei, and Chunxiang Gu. Bitcoin theft detection based on supervised machine learning algorithms. *Security and Communication Networks*, 2021:1–10, 2021.

[61] Ebrahim Ebrahimi, Kaveh Mollazade, and Arman Arefi. An expert system for classification of potato tubers using image processing and artificial neural networks. *International Journal of Food Engineering*, 8(4), 2012.

[62] Mohamad H Hassoun et al. *Fundamentals of artificial neural networks*. MIT press, 1995.

[63] Neural network models (supervised), 2023, url : https://scikit-learn.org/stable/modules/neural_networks_supervised.html.

[64] Jiliang Tang, Salem Alelyani, and Huan Liu. Feature selection for classification: A review. *Data classification: Algorithms and applications*, page 37, 2014.

[65] Max Kuhn, Kjell Johnson, et al. *Applied predictive modeling*, volume 26. Springer, 2013.

[66] Huan Liu and Hiroshi Motoda. *Feature selection for knowledge discovery and data mining*, volume 454. Springer Science & Business Media, 2012.

[67] Jason Weston, André Elisseeff, Bernhard Schölkopf, and Mike Tipping. Use of the zero norm with linear models and kernel methods. *The Journal of Machine Learning Research*, 3:1439–1461, 2003.

[68] Pablo M Granitto, Cesare Furlanello, Franco Biasioli, and Flavia Gasperi. Recursive feature elimination with random forest for ptr-ms analysis of agroindustrial products. *Chemometrics and intelligent laboratory systems*, 83(2):83–90, 2006.

[69] Treemodels.scala, url : https://github.com/apache/spark/blob/master/mllib/src/main/scala/org/apache/spark/ml/tree/treeModels.scala.

[70] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[71] Ethereum Fraud Detection, sfarrugia15 2022, url : https://github.com/sfarrugia15/Ethereum_Fraud_Detection/blob/master/Account_Stats/Complete.csv.

[72] Patel Nikunjkumar Sureshbhai, Pronaya Bhattacharya, and Sudeep Tanwar. Karuna: A blockchain-based sentiment analysis framework for fraud cryptocurrency schemes. In *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6. IEEE, 2020.

[73] Melissa J Azur, Elizabeth A Stuart, Constantine Frangakis, and Philip J Leaf. Multiple imputation by chained equations: what is it and how does it work? *International journal of methods in psychiatric research*, 20(1):40–49, 2011.

[74] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011.

[75] Oliver Lüdtke, Alexander Robitzsch, and Simon Grund. Multiple imputation of missing data in multilevel designs: A comparison of different strategies. *Psychological methods*, 22(1):141, 2017.

[76] Hanyang Wang, Sirui Zhou, Honglei Li, Juan Hu, Xinran Du, Jinghui Zhou, Yunlong He, Fa Fu, and Houqun Yang. Deep learning network intrusion detection based on network traffic. In *International Conference on Artificial Intelligence and Security*, pages 194–207. Springer, 2022.

[77] Understanding AUC - ROC Curve, Sarang Narkhede 2018, url : https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5.

[78] Douglas G Altman and J Martin Bland. Diagnostic tests. 1: Sensitivity and specificity. *BMJ: British Medical Journal*, 308(6943):1552, 1994.

[79] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[80] Fraud Detection: ETHEREUM transactions, Chiticariu Cristian 2022, url : https://www.kaggle.com/code/chiticariucristian/fraud-detection-ethereum-transactions.