Université de Montréal

Prediction of the Transaction Confirmation Time in Ethereum Blockchain

par

Harsh Jot Singh

Département d'informatique et recherche opérationnelle

Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures et postdoctorales

en vue de l'obtention du grade de

Maître ès sciences (M. Sc.)

en Informatique

Août 2019

Université de Montréal
Département d'informatique et recherche opérationnelle
Faculté des arts et des sciences

Ce mémoire intitulé

Prediction of the Transaction Confirmation Time in Ethereum Blockchain

Présenté par

Harsh Jot Singh

A été évalué par un jury composé des personnes suivantes

Alain Tapp
Président-rapporteur

Abdelhakim Senhaji Hafid
Directeur de recherche

Fabian Bastin
Membre du jury

Résumé

La blockchain propose un système d'enregistrement décentralisé, immuable et transparent. Elle offre un réseau de nœuds sans entité de gouvernance centralisée, ce qui la rend "indéchiffrable" et donc plus sûr que le système d'enregistrement centralisé sur papier ou centralisé telles que les banques. L'approche traditionnelle basée sur l'enregistrement ne fonctionne pas bien avec les relations numériques où les données changent constamment. Contrairement aux canaux traditionnels, régis par des entités centralisées, blockchain offre à ses utilisateurs un certain niveau d'anonymat en leur permettant d'interagir sans divulguer leur identité personnelle et en leur permettant de gagner la confiance sans passer par une entité tierce.

En raison des caractéristiques susmentionnées de la blockchain, de plus en plus d'utilisateurs dans le monde sont enclins à effectuer une transaction numérique via blockchain plutôt que par des canaux rudimentaires. Par conséquent, nous devons de toute urgence mieux comprendre comment ces opérations sont gérées par la blockchain et combien de temps cela prend à un nœud du réseau pour confirmer une transaction et l'ajouter au réseau de la blockchain.

Dans cette thèse, nous visons à introduire une nouvelle approche qui permettrait d'estimer le temps il faudrait à un nœud de la blockchain Ethereum pour accepter et confirmer une transaction sur un bloc tout en utilisant l'apprentissage automatique. Nous explorons deux des approches les plus fondamentales de l'apprentissage automatique, soit la classification et la régression, afin de déterminer lequel des deux offrirait l'outil le plus efficace pour effectuer la prévision du temps de confirmation dans la blockchain Ethereum. Nous explorons le classificateur Naïve Bayes, le classificateur Random Forest et le classificateur Multilayer Perceptron pour l'approche de la classification. Comme la plupart des transactions sur Ethereum sont confirmées dans le délai de confirmation moyen (15 secondes) de deux confirmations de bloc, nous discutons également des moyens pour résoudre le problème asymétrique du jeu de données rencontré avec l'approche de la classification. Nous visons également à comparer la précision prédictive de deux modèles de régression d'apprentissage automatique, soit le Random Forest Regressor et le Multilayer Perceptron, par rapport à des modèles de régression statistique, précédemment proposés, avec un critère d'évaluation défini, afin de déterminer si l'apprentissage automatique offre un modèle prédictif plus précis que les modèles statistiques conventionnels.

Mots clés: Apprentissage automatique, Ethereum, Blockchain, Régression, Classification, Random Forest, Naïve Bayes, Multilayer Perceptron, Transaction

Abstract

Blockchain offers a decentralized, immutable, transparent system of records. It offers a peer-to-peer network of nodes with no centralised governing entity making it 'unhackable' and therefore, more secure than the traditional paper based or centralised system of records like banks etc. While there are certain advantages to the paper based recording approach, it does not work well with digital relationships where the data is in constant flux. Unlike traditional channels, governed by centralized entities, blockchain offers its users a certain level of anonymity by providing capabilities to interact without disclosing their personal identities and allows them to build trust without a third-party governing entity.

Due to the aforementioned characteristics of blockchain, more and more users around the globe are inclined towards making a digital transaction via blockchain than via rudimentary channels. Therefore, there is a dire need for us to gain insight on how these transactions are processed by the blockchain and how much time it may take for a peer to confirm a transaction and add it to the blockchain network.

In this thesis, we aim to introduce a novel approach that would allow one to estimate the time (in block time or otherwise) it would take for Ethereum Blockchain to accept and confirm a transaction to a block using machine learning. We explore two of the most fundamental machine learning approaches, i.e., Classification and Regression in order to determine which of the two would be more accurate to make confirmation time prediction in the Ethereum blockchain. More specifically, we explore Naïve Bayes classifier, Random Forest classifier and Multilayer Perceptron classifier for the classification approach. Since most transactions in the network are confirmed well within the average confirmation time of two block confirmations or 15 seconds, we also discuss ways to tackle the skewed dataset problem encountered in case of the classification approach. We also aim to compare the predictive accuracy of two machine learning regression models- Random Forest Regressor and Multilayer Perceptron against previously proposed statistical regression models under a set evaluation criterion; the objective is to determine whether machine learning offers a more accurate predictive model than conventional statistical models.

Keywords: Machine learning, Ethereum, Blockchain, Regression, Classification, Random Forest, Naïve Bayes, Multilayer Perceptron, Transaction.

Content

List of Figures

List of Tables

List of Abbreviations

MLP	Multilayer Perceptron
RF	Random Forest
ETH	Ether
GHOST	Greedy Heaviest Observed Subtree
EOA	Externally Owned Accounts
RLP	Recursive Length Prefix
MSE	Mean Square Error
MAE	Mean Absolute Error
RMSE	Root Mean Square Error
MAPE	Mean Absolute Percentage Error
SMOTE	Synthetic Minority Over-sampling Technique
TXN	Transaction

Acknowledgment

First and foremost, I would like to express my deep gratitude to my supervisor, Abdelhakim Senhaji Hafid, for allowing me the opportunity to work under his guidance. His direction and support were invaluable in the outcome of this research, but his enthusiasm made working with him a thoroughly enjoyable experience. Thank you for everything.

To the members of the Montreal Blockchain Lab and DIRO who are always willing to assist graduate students, I express sincere thanks for their help. I would also like to thank Prof. Guillaume Rabusseau, of DIRO, for the fruitful discussions we had.

To my family and friends, I cannot thank you enough for your encouragements and support. To my godparents, Saran, Bal, Pammi and Harjeet, my grandfather Didar, I am thankful for you always being there for me.

Finally, to my friends Gabriel, Christy and Lea. Going through my master's degree and being able to rant to you two about my frustration truly got me through the good and the worst of times. I am genuinely grateful for your friendship. Thank you!

1. Introduction

Currencies are defined as an economic buffer; an entity that allows individuals to convert their efforts into something tangible. It allows their efforts to maintain a certain value which can be converted into goods or services as desired. It is governed by a central entity such as a bank and is state backed. Blockchain, fundamentally, was meant to disrupt the whole socio-economic dynamic. It was meant to transform the way we interact and exchange value by rendering the need of a middleman or a governing third-party entirely inconsequential.

This chapter introduces blockchain technology and discusses its basic structure. It then states the problem statement for the thesis and the motivation behind it. It also presents briefly the thesis contribution. Finally, it presents the organization of the rest of this thesis.

1.1 Blockchain: An Overview

Blockchain Technology is a distributed database shared between nodes in a peer-to-peer network (e.g., more than 10000 nodes in Ethereum). Basically, each network node can receive and broadcast transactions. Blockchain, as its name suggests, records transactions into linked blocks [1]. When a user wants to interact with the blockchain (e.g., to transfer cryptocurrency or store a testament), they create and sign, using their private key, a transaction; note that blockchain, in itself, uses public key encryption. Then, it sends the transaction to the blockchain network; a node that receives the transaction, validates the transactions (e.g., verifies the user's signature) and, if valid, stores the transaction in its pending list of transactions and transmits it to its neighbouring nodes. Periodically, a node is selected to create a block; the selection is based on the consensus protocol in use. In the case of proof-of-work (PoW) consensus protocol [2], the node that first solves a mathematical puzzle, is the one that creates the new block. It is important to emphasize that there should be no shortcuts to solve the puzzle in order to guarantee that nodes are selected randomly. PoW consists in determining a string (called nonce) such that when combined with the block header and hashed results in hash that includes a given number of leading bits 0 (this number represents the difficulty to solve the puzzle). Nodes are incentivised to create new blocks because they are rewarded by newly minted coins (e.g., in the bitcoin blockchain, the reward is 12.5 bitcoins as of 2019) as transactions fees. The

time it takes to generate a block, called block time, is specific to the blockchain in use; for example, the block time for bitcoin is 10 minutes whereas it is 15 seconds for Ethereum.

Given the network delays, two or more nodes may create new blocks that reference the same previous block; this may cause diverging views of the blockchain (i.e., this is called "natural" fork). To solve this problem, nodes consider the longest blockchain (in terms of the number of blocks) as the correct/consensus blockchain to build on.

A block consists of block header and a list of transactions; the node selects, from its pending list of transactions, the transactions to include in the block. The maximum number of transactions in a block depends on the blockchain that is used. The blockchain header includes (a) a timestamp that represents the time when the block is created; the timestamp is generated by secure mechanisms of timestamping to guarantee the chronological creation of blocks in the chain; (b) a hash of the previous block; this makes blockchain immutable since it is not possible to modify any block without changing the entire chain (see Figure 1.1).

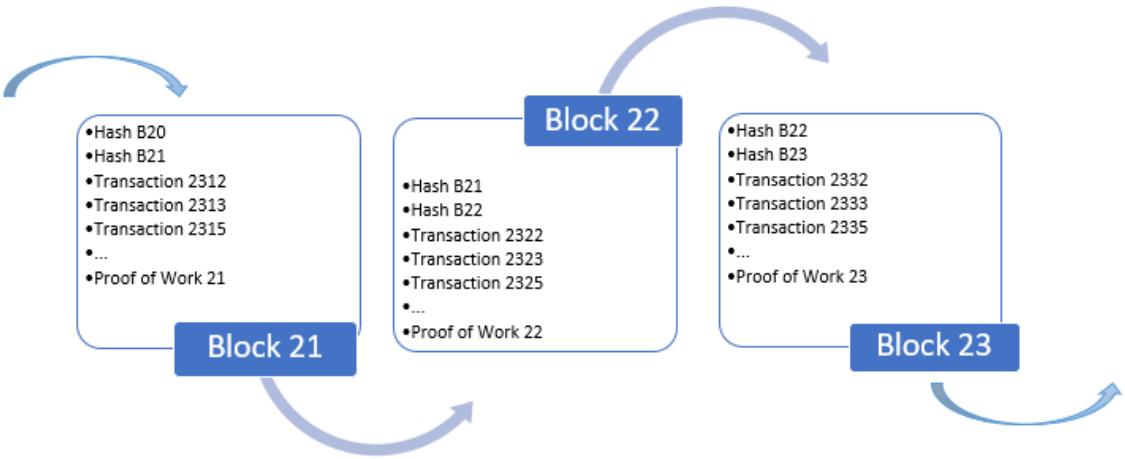

Fig 1.1. A depiction of Blocks in a Blockchain

Indeed, changing even one bit in a block changes the value of its hash; (c) a nonce that solves the mathematical puzzle when proof-of work consensus mechanism is used; and (d) root node of Merkle tree that represents the hash of all transactions inside the block. The node that creates the new block, appends it to its copy of the blockchain and sends it to its neighbouring nodes. Upon receipt of the new block, a node validates the block and, if valid, appends it to its copy of the blockchain and sends it to its neighbouring nodes. The block validation process includes (a) checking the Merkle tree's root hash; (b) checking the validity of all transactions in the block; and (c) checking the hash of the previous block. Broadcasting and validating the new block are what keep all the nodes on a network synchronized with each other. When a new block is appended to the blockchain, the transactions in this block are said to be confirmed.

1.1.1 Blockchain: Nodes

According to [3], there are three types of blockchain nodes: (a) mining nodes: they are responsible for producing blocks; each time a mining node produces a block, it is rewarded; (b) full nodes: they are responsible for maintaining and distributing copies of the entire blockchain; in particular, they are responsible for the validation of blocks produced by the mining nodes. The more full nodes, the more decentralized the blockchain and the harder to hack; and (c) light nodes: They perform similar functions as full nodes without maintaining the entire blockchain; in general, a light node connects to a full node, downloads only the headers of previous blocks; they connect to a full node.

1.1.2 Blockchain: Categories

Blockchain comes in many different types. More specifically, there are three types of blockchains: permission less blockchain also known as public blockchain (e.g., Bitcoin and Ethereum), permissioned blockchain also known as consortium blockchain (e.g., Hyperledger fabric), and private blockchain. In public blockchains, any participant/user can write data to the blockchain and can read data recorded in the blockchain; anybody can be a full node, a miner or a light node. Thus, there is little to no privacy for recorded data and there are no regulations or rules for participants to join the network. Generally, pubic blockchains are considered pseudo-anonymous (e.g., bitcoin and Ethereum); a participant does not have to divulge her identity (e.g., name) instead she is linked to an address (i.e., hash of public key). Providing anonymity is difficult but it is feasible (e.g., Zcash [4]). The success of this type of blockchains depends on the number of participants; it uses incentives to encourage more participation. Consortium blockchains put restrictions on who can participate. In particular, the creation and validation of blocks are controlled by a set of pre-authorized nodes; for example, we have a consortium of 10 banks where each bank operates one node. The right to read data recorded in the blockchain can be public or restricted to the participants. Even participants may be restricted on what they can do in the blockchain; for example, transactions between 2 participants may be hidden from the rest of participants. In a private blockchain, write permissions are centralized and restricted to one entity; read permissions may be public or restricted.

1.1.3 Blockchain: Characteristics

The key characteristics of Blockchain can be summarized as follows:

- Decentralization: there is no central entity that controls the blockchain; indeed, blockchain nodes are responsible, via consensus, for the maintenance of the blockchain in a trust less environment. Nodes can join or leave the network at will. Each node of the network has its own copy of the blockchain making it resilient to single point failure.

- Transparency: all transactions that are stored in the blockchain are visible to anyone (e.g., suitable for audit). Privacy can be supported but it is difficult to deploy (e.g., Zcash [4] supports privacy).
- Immutable: When transactions are confirmed in the correct/consensus blockchain, it is impossible to remove/alter them. This is because block N includes the hash of block N-1; thus, any modification in block N-1 will break the link to block N.
- Secure: blockchain is "unhackable"; this assumes that no entity can control 51% of the hashing power of the blockchain network; since its inception, bitcoin has never been hacked.

1.2 Motivation and Problem Statement

Blockchain offers a way for users to exchange value with all the capabilities offered by state-backed currencies but is more secure and does not require central governing entity. The biggest application of blockchain is cryptocurrency. Cryptocurrency is a digital or virtual type of currency that uses encryption techniques to convert plain text into unintelligible text and vice-versa. It is designed to work as a medium of exchange as well as to control the creation of additional units and validate transactions and allow their transfer through the blockchain network. Bitcoin, Namecoin, Ethereum etc. are a few examples of most commonly used cryptocurrencies.

The market capitalization of publicly traded cryptocurrencies as of April 10, 2019 is about 176.3 billion out of which Bitcoin makes up to 141.6 billion and Ethereum makes up to 26.6 billion dollars [5]. On average, about 12 billion USD is transferred via at least a million transactions each day. The amount of capital involved alone makes it necessary for one to be able to perform analysis on historical and real-time data and allow one to infer and/or predict a certain level of details about future trends in the market. These trends can range from analysis of address space, price prediction, transaction confirmation time prediction, etc.

Bitcoin, when it first came to be in 2009, was not simply meant to be a tool to spend money digitally. It was meant to be a convergence of networking, cryptography and open source software technologies with the aim to completely eradicate the need of state-backed currencies by crossing international boundaries and nullifying the usage of banks as a mean to store money [6]. Since then, there has been a gradual increase in the awareness and excitement in regard to cryptocurrencies and the rudimentary distributed ledger (or Blockchain) technology. On the grassroot level, these blockchain based cryptocurrencies are meant to provide complete anonymity (or rather pseudo-anonymity) to its users by providing them capabilities to operate via a set of addresses without having to disclose any of their personal details [7] all the while providing high level of transparency on past transactions.

Ethereum, the second most commonly used cryptocurrency was launched in 2015, is the most well-established, largest open-ended, public and blockchain-based software platform.

Unlike Bitcoin, which only allows for value exchange, Ethereum permits the utilization of Smart Contracts. These are snippets of code or protocols that digitally facilitate, verify and ensure performance of a contract [8]. It also offers a programming language allowing users to publish Distributed Applications without external interference, fraud or downtime using its own decentralized public blockchain technology [9] [10].

Ethereum is a decentralized technology. It runs on a network of machines (or nodes) that are distributed globally. Since there is no central point of failure, Ethereum is also immune to hacking or any attack preventing its operation. These characteristics of Ethereum allow it to be more formidable in comparison to its counterparts and hence, make it possible for Ethereum data to be used for knowledge inference and/or analysis. As such, in this thesis, we particularly focus on Ethereum and the time it takes for a mining node to confirm a transaction on the said platform. A transaction is how the external world interacts with the Ethereum network. Each time there is a need to modify or update the state of the network, a transaction has to be made. These transactions can be of three types: (1) Fund transfer between two accounts; (2) Deployment of a contract on the Ethereum network; and (3) Execution of a function on a deployed contract.

Any change in the state of the network is considered to be a transaction. The transaction carries information of the user, via the user interface (e.g., browser), to the network, to another endpoint on the network or back to the user's station. It could carry large amount of capital in form of ether or data through contracts that a transaction can call for execution.

Due to the importance of these transactions and the amount of capital these might carry within them, it is very important for a user to gain some insight on how much time it might take for the transaction to be processed based on the network traffic. By gaining these insights ahead of time, a user can infer whether right now would be the right time for her to send this transaction to the network. In case where a transaction is highly time sensitive, prediction of confirmation time can assist a user in making changes to the transaction to ensure that a mining node confirms it as swiftly as possible. Not only this, by understanding how much time the miners will take to process a transaction, user can gain insight on miner policies, i.e., what factors are taken into consideration by mining nodes while choosing one transaction over the others. In general, a miner would choose a transaction where they would get the most incentive. But, since there is no set policy for the Ethereum blockchain, these policies can change at any time and by analysing the most recent network trend, the user can keep up to date with these policies.

Each time a user makes a transaction, she has to pay a fee. Again, while miners will process transactions with higher fees first, it is not efficient for a user to send a transaction with a value so low that the transaction would never be picked up and she would have to resend the transaction at a higher value. Similarly, it would not be helpful (or rather it would be wasteful) for the user to make a transaction with fees higher than what miners are accepting to prioritize transactions. Gaining insight on the current network state will assist the user to determine "optimal" fees she needs to pay for her transaction.

1.3 Contribution

In this thesis, we propose to use Machine Learning in order to predict confirmation time for a transaction on the Ethereum blockchain network. The prediction problem, in a simplified manner, can be seen as a classification problem in machine learning terminology. A classifier would predict which of the eight classes of confirmation time frame, the transaction in question belongs to. These eight classes, set as per analysis of transaction data, are: within 15 seconds (or approximately 1 block time), within 30 seconds, within 1 minute, within 2 minutes, within 5 minutes, within 10 minutes, within 15 minutes, within 30 minutes or longer [11]. We explore the Naïve Bayes, Random Forest (RF) as well as Multilayer Perceptron (MLP) classifiers. For our exploration, we employ Naïve Bayes classifier with Gaussian distribution and MLP with batch normalization and dropout with SoftMax output layer.

It should be noted that out of the million transactions used as historical data for the classification problem, 49.4% of the transactions belonged to class 1 (within 15 seconds) and 24.8% belonged to class 2, making the dataset highly imbalanced and the classification task becomes challenging since the classification models have a tendency of always classifying the dominant class. This problem would bias the classifier to always predict a transaction as belonging to class 1. To overcome this, we also explore re-sampling methods for the dataset.

We also consider the problem at a more fundamental level as a machine learning regression problem. A regressor would take in as input the transaction metadata and based on the data learned in the past would make a prediction for the transaction in question. We explore random forest and MLP regressors. The Naïve Bayes algorithm is not suitable for regression problem and, therefore, was not explored in this case.

While there has been some previous literary work on Ethereum network analysis that uses machine learning, most of it has been focussed on analysis of user address space or price prediction. These experiments were done mostly in a restricted environment and only considered historical data for prediction or analysis. In case of transaction confirmation time prediction, there has not been any work with machine learning and has mostly been focused on statistical regression algorithms. While statistical modelling has been shown to work well for prediction problem, the complexity of a statistical algorithm is compared with the two machine learning algorithms, random forest and multilayer perceptron (MLP), to explore which of the three algorithms would work best with our problem statement.

We show that our proposed models achieve a performance of 83.6% in case of classification and a remarkable 89.36% in case of regression approach. In addition, by exploring multiple variants of the machine learning algorithms proposed in this thesis as well as comparing them with statistical algorithms already used for prediction purposes, we draw insight on which of the two, machine learning or statistical analysis, is more viable. We conclude that the flexibility with handling complex relationships among dataset

features offered by machine learning makes it an extremely promising option for further research and analysis of Ethereum blockchain network.

1.4 Organisation of this Thesis

In Chapter 2, we introduce fundamentals of Ethereum and discuss how a transaction is executed on the Ethereum network. In chapter 3, we discuss basic concepts and terminologies in machine learning. We briefly discuss different machine learning models such as classification, regression, etc. We also present the criteria of evaluation for these machine learning models.

In Chapter 4, we present our proposed method that is used to predict transaction block confirmation time. Chapter 5 presents the details of the evaluation and the implementation of our proposed model. Finally, Chapter 6 concludes the thesis and presents future work.

2. Ethereum

Ethereum is a public, open-source, distributed ledger computing platform that aims to achieve a new, more secure and trust-oriented internet. It is a supercomputer running on blockchain technology that permits utilization of smart contracts to make digital payments, to securely transfer data and to provide access to an open financial system. It was designed to utilize the internet to create a decentralized value-transfer system, globally available and *virtually* free to use. It is meant to exist without a centralised governing entity but to be operated and governed by all who share the network. Fig 2.1 shows a graphical illustration of the Ethereum Network.

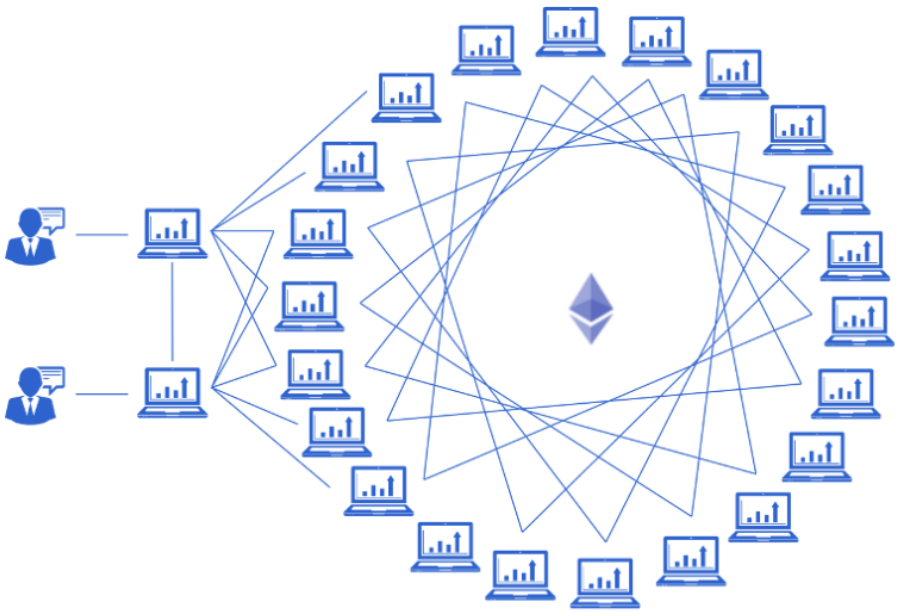

Fig 2.1. A depiction of the Ethereum Network.

Ethereum facilitates exchange between consenting individuals who, without it, would have no mean to trust one another. It allows individuals who are geographically separated, unwilling and incompatible to make any transactions via traditional systems due to lack of trust or because of their rudimentary legal system. Ethereum offers this by providing a

secure, trusted mechanism to make such exchanges all the while providing a degree of anonymity to its users.

According to Dr. Gavin Wood, the founder of Ethereum [2], "*Ethereum is a project which attempts to build the generalized technology; technology on which all transaction-based state machine concepts may be built. Moreover, it aims to provide to the end-developer a tightly integrated end-to-end system for building software on a hitherto unexplored compute paradigm in the mainstream: a trustful object messaging compute framework.*"

In this chapter, we present the Ethereum blockchain paradigm and the associated terminologies. We will then discuss the transaction execution model in Ethereum blockchain network.

2.1 Ethereum Blockchain Paradigm

Ethereum was developed to facilitate transactions among individuals without requiring them to disclose their personal details, i.e., by developing trust among two anonymous entities. As a whole, it is a transaction-based state machine beginning at the genesis state [12] and moves on to the final state via incremental execution of transactions [13]. The final state of the machine is what is perceived as the canonical version of Ethereum network. It includes details such as account balances, trust arrangements, reputations, etc. In between the two states, there can be valid or invalid changes. The invalid state changes may or may not imply an invalid account balance modification in either the sender's or the receiver's account. Formally [2],

$$\sigma_{t+1} = Y(\sigma_t, T)$$

where, Y is Ethereum state transition function allowing components to carry out computations and σ allows them to store arbitrary state between transactions.

These transactions are accumulated into blocks by utilizing Merkle trees [14]. These blocks work as journals recording the transactions and are connected to one another using a cryptographic hash. Since a transaction changes the state of the network, it must be valid. For a transaction to be valid, it has to go through a process called mining. Blocks punctuate transactions with incentives for the mining nodes. This incentivization is a state-based function that adds value to a nominated account [2].

Mining is the process of allocating effort on one set of transaction series or block over the others. It is achieved via a cryptographically secure process called 'proof-of-work'. Formally [2],

$$\sigma_{t+1} \equiv \Pi(\sigma_t, B)$$

$$B \equiv (\dots, (T_0, T_1, \dots))$$

$$\Pi(\sigma, B) \equiv \Omega(B, Y(Y(\sigma, T_0), T_1) \dots)$$

where, Ω is the block-finalisation state transition function used to reward the nominated account; B is the block in question; T_n represents the transaction associated with the block and Π is the block-level state function.

This represents the basic blockchain paradigm on which Ethereum and all blockchain based technologies operate on. Similar to other blockchain technologies, like Bitcoin or Namecoin that came before it, Ethereum has its own cryptocurrency called Ether (ETH). ETH is digital money. It is the token whose blockchain is generated by Ethereum platform. It can be transferred among accounts or mining nodes to compensate for goods and services offered on or outside the Ethereum network. The smallest denomination of Ether is Wei. One Ether is defined as 10^{18} Wei. The denominations of Ether are given in Table 2.1.

TABLE 2.1 DENOMINATION OF ETHER

Multiplier	Name
10^0	Wei
10^{12}	Szabo
10^{15}	Finney
10^{18}	Ether

As discussed, Ethereum is a state-based machine decentralized machine. This means that each node has equal opportunity and capability to create a new block on some pre-existing block. This may lead to a block tree and deciding, which path may be the best root to leaf traversal path, may be not trivial. If the chain were to diverge, the user might end with two distinct states and there will be no way to determine which state is most 'valid.' This is known as a *fork* and must be avoided at all cost as the uncertainty might compromise the whole system. Fig 2.2. illustrates forks in blockchain[1].

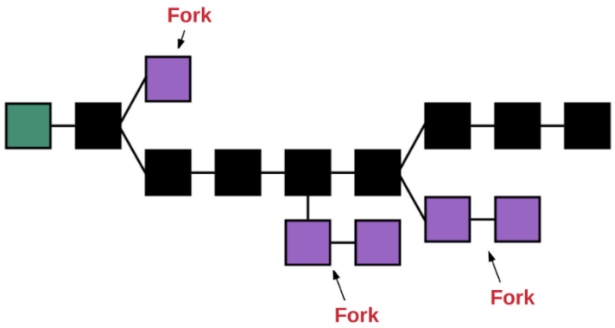

Fig 2.2. Forks in blockchain.

[1] Figure source: Kasireddy, P., How does Ethereum work, anyway?, Medium, https://medium.com/@preethikasireddy/how-does-ethereum-work-anyway-22d1df506369 , last accessed 2019/04/15.

To draw consensus on which path is most viable, GHOST (Greedy Heaviest Observed SubTree) protocol is used [15]. With GHOST, the path that has had most computation done on it is selected as the consensus path. This is done by looking at the block number of the most recent block on each possible path. The higher the block number, the more the number of blocks in that chain which implies that higher amount of effort has been put into this chain. For instance, in case of the tree illustrated in Fig 2.2., GHOST would choose the chain with three blocks in it as illustrated in Fig 2.3.

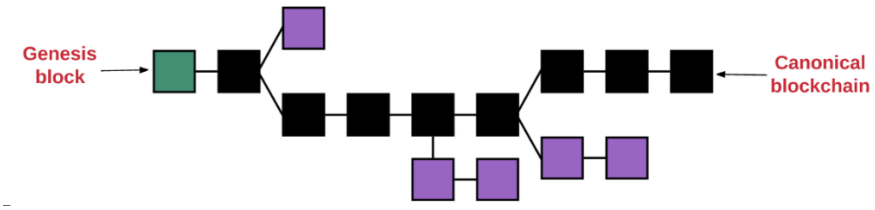

Fig 2.3. Illustration of GHOST protocol[1].

2.2 Accounts in Ethereum

The basic unit of Ethereum is account. For an individual to make a transaction she needs to have an account. Each account has a 20-byte address and a state associated with it. These accounts are of two types: Externally Owned Account (EOA) and Contract Accounts. EOA are controlled by private keys whereas Contract Accounts are controlled by their contract code and can only be activated by an EOA. Irrespective of the type of account in question, each account state has four components given in Table 2.2. and Fig 2.4.

TABLE 2.2: COMPONENTS OF ACCOUNT STATE IN ETHEREUM

Component	
nonce	For EOA, this represents the number of transactions sent from the account's address. For contract account, nonce represents number of contracts created by the account.
balance	Number of Wei currently owned by the account.
storageRoot	A 256-bit hash of Merkle tree's root node that encodes the storage content of the account. This is empty by default.
codeHash	For contract account, the Ethereum Virtual Machine code of the account gets hashed and stored in codeHash while for EOA, codeHash contains the hash of an empty string.

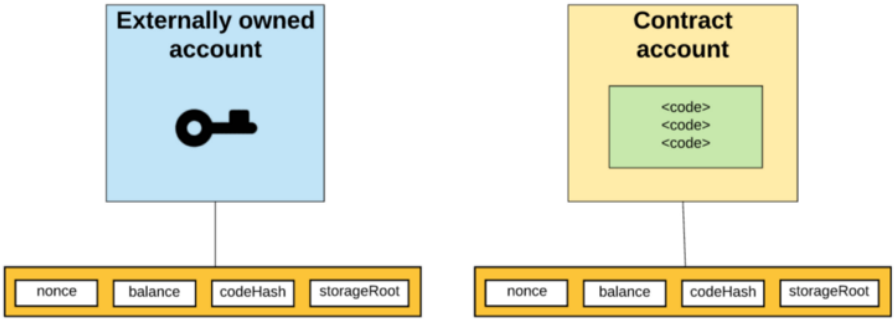

Fig 2.4. Components of Account State[2]

2.2.1 Externally Owned Account

An externally owned account, controlled by a private key, has an ether balance and can send transactions to other accounts. These transactions can be to transfer ether, trigger contract accounts or update the state of the machine. There is no code associated with an externally owned account. Fig 2.5 illustrates the operation of an externally owned accounts that calls a contract account.

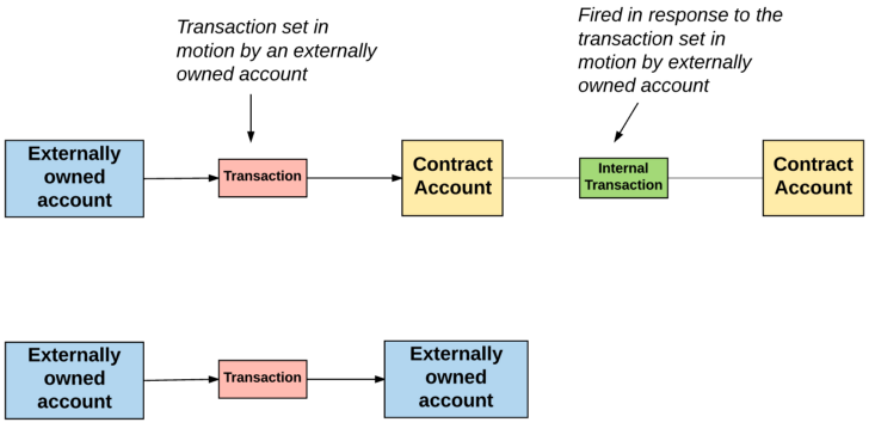

Fig 2.5 An Externally Owned Account[2].

2.2.2 Contract Account

A contract account too has an ether balance but unlike EOA has some kind of code associated with it. The code execution is triggered by transactions from EOA or message

[2] Figure source: Kenneth, H., Ethereum Account, Medium, https://medium.com/coinmonks/ethereum-account-212feb9c4154, last accessed 2019/04/15.

calls received from other contracts. These message calls, unlike transactions, do not broadcast or publish anything on the blockchain. A message call does not consume any ether and discards all state changes when the call is done. Whenever a contract account is executed, it performs operations or arbitrary complexities, manipulates its own persistent storage or its own permanent state and/or call other contracts.

A contract account can list all the incoming transactions and is of two types: Simplet account and Multisig account. A Simplet account is created and owned by a single user account but a Multisig (Multisignature) account can have multiple owners one of which would also be the creator account.

2.3 The Transactions

A transaction is how the external world interacts with the Ethereum Network. It is a cryptographically signed instruction sent by an account holder to change the state of Ethereum at any given time. There are two types of transactions in Ethereum: (1) Transactions made to generate message calls or (2) To create new accounts with code, i.e., contract accounts. Both of these transactions have a common set of components [2] discussed in Table 2.3 (a), (b) and (c).

TABLE 2.3 (a): COMPONENTS OF A TRANSACTION IN ETHEREUM

Component	
nonce	Represents the number of transactions sent by the sender (T_n).
gasPrice	Number of Wei to be paid per unit of gas for the computation cost incurred due to the execution of a transaction (T_p).
gasLimit	Represents the maximum amount of gas that should be used to execute a transaction. This is paid upfront and cannot be modified later (T_g).
to	160- bit receiver's address or that of a contract creation transaction (T_t).
value	Value of Wei to be transferred (T_v).
v, r, s	Transaction signature; it is also used to determine the transaction source or sender (T_v, T_r and T_s).

TABLE 2.3 (b): ADDITIONAL COMPONENTS OF CONTRACT CREATION TRANSACTION IN ETHEREUM

Component	
init	An unlimited size byte array that specifies the EVM-code for account initialization procedure (T_i). It is an EVM code fragment that returns the ***body***, the second code fragment to be executed each time the account receives a message call while ***init*** is executed only once.

TABLE 2.3 (c): ADDITIONAL COMPONENTS MESSAGE CALL IN ETHEREUM

Component	
data	An unlimited sized byte array containing the input data of the message call (T_d).

2.4 The Block

An Ethereum block is a collection of the block header (H), all the data relevant to the transactions (T) it includes, and a set of other block headers (U) that are known to have a parent block equal to the current block's parents' parents, known as ommers. Table 2.4 shows the components of a block header [2] and Fig 2.6 describes these components.

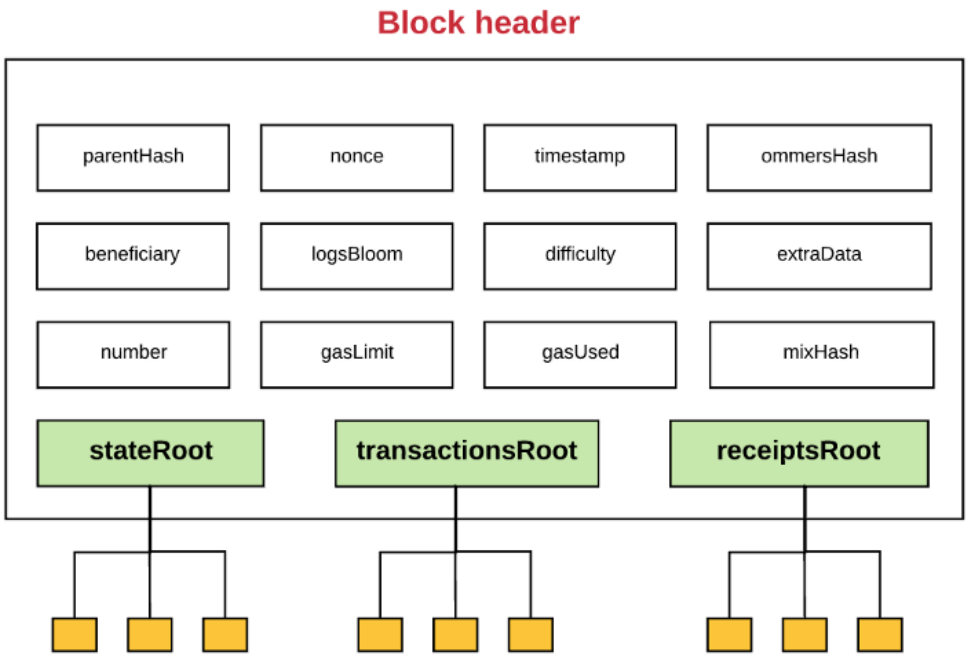

Fig 2.6 A block header in Ethereum blockchain.

TABLE 2.4: COMPONENTS OF A BLOCK HEADER IN ETHEREUM

Component	
parentHash	256-bit hash of the parent block's header (H_p).
ommersHash	256-bit hash of the ommers list part of the block (H_o).
beneficiary	160-bit address of the mining node to which all the incentive for this block would go to (H_c).
stateRoot	256-bit hash of the root node of state trie (H_s).
transactionRoot	256-bit hash of the root node of transaction trie (H_t).
receiptsRoot	256-bit hash of the root node of transaction trie (H_e).
logsBloom	Bloom filter composed of logger address and log topics contained in each log entry from the receipt of reach txn in txn list (H_b).
difficulty	Scalar corresponding to the difficulty level of the block (H_d).
number	Number of ancestor blocks (H_i). For genesis block, this is zero.
gasLimit	Current limit of gas expenditure for the block (H_l).
gasUsed	Total gas used in transactions belonging to this block (H_g).
timestamp	Unix time stamp for when the block was created (H_s).
extraData	Bit array containing relevant data to the block (H_x). This must be 32 byte or fewer.
mixHash	256-bit hash which when combined with nonce proves the amount of computation done on the block (H_m).
nonce	64-bit hash which when combined with mixHash proves the amount of computation done on the block (H_n).

2.5 Gas and Payment

In order to prevent network abuse, all transactions on Ethereum are subject to a fee. This fee is paid in 'gas'. Gas is the fundamental unit used to measure the cost of computation and execution of a transaction on Ethereum.

Every transaction has a specified amount of gas associated with it known as gasLimit (see Table 2.3 (a)). This is the prepaid amount that is charged to the sender account before the transactions is processed. The amount is charged based on another entity included in a transaction called gasPrice. Consider an example, if the sender sets the gas limit to 50,000 and a gas price of 20 gwei. Then, this would mean that the sender is willing to spend at most 50,000 x 20 gwei or 10^{-3} Ether in order to execute that transaction.

If an account cannot pay for a transaction it wants to process, the transaction would be considered failed or invalid. On the other hand, if a transaction requires less amount of gas that what is paid by the account then the remaining gas is refunded back to the user's account. Fig 2.7 shows that a transaction failed because there wasn't enough gas to cover execution cost; the gas spent is not refunded.

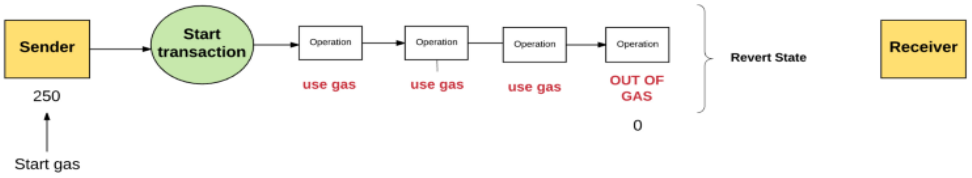

Fig 2.7 (a) Gas and Payment for an invalid transaction in Ethereum[3].

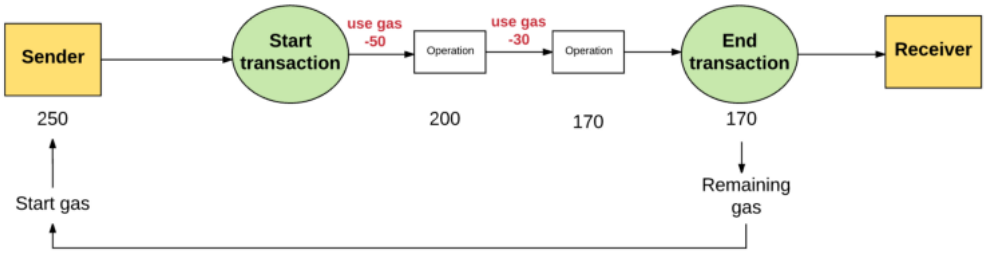

Fig 2.7 (b) Gas and Payment for a valid transaction in Ethereum[3].

[3] Figure source: Kasireddy, P., How does Ethereum work, anyway?, Medium, , last accessed 2019/04/15.

The fee that each account holder is charged for making a transaction on Ethereum network is sent to the miner since they are the ones who had put in the effort to run computations and validate these transactions. Typically, the higher the fees a sender is willing to spend, the more a miner can derive from and better are the chances for the transaction to get validated right away.

2.6 World State

The world state in Ethereum is the mapping between the 160-bit identifier addresses and account states that is stored in a modified Merkle tree (trie) [2]. A Merkle tree is a binary tree composed of a set of nodes where a large number of leaf nodes contain underlying data and all the intermediate nodes store a hash for their two child nodes with a root node, also containing a hash of its child nodes represents the top of the tree. Fig 2.8. illustrates a simple Merkle tree.

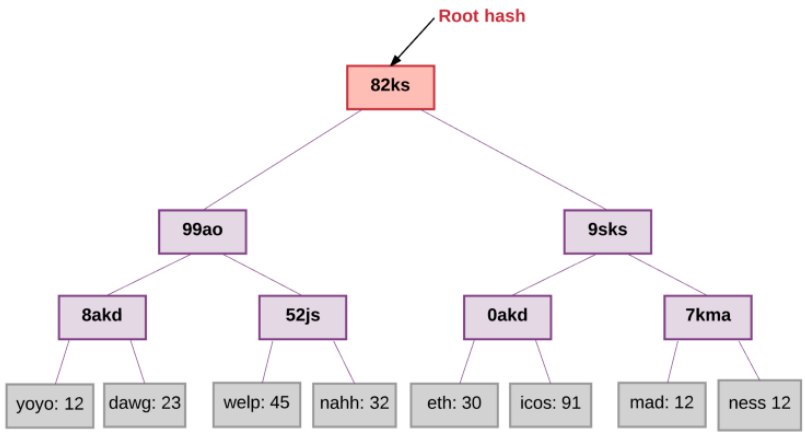

Fig 2.8. A Merkle tree is a binary tree with leaf nodes containing underlying data and the other nodes containing hash of their two child nodes.

The trie stores bytearrays to bytearrays mapping in a simple backend database known as the state database. The data contained in the leaf nodes is generated by splitting our data in chunks and then splitting those chunks into buckets. Then, we hash the newly created buckets and we continue hashing until there is only one left that represents the root node. This hash is known as the root hash. The trie has a key for each node. Starting from the root node, each key contains the information about which node should be traversed next to get to the desired leaf node.

Similar trie structure is used to store transactions and receipts. The receipts in Ethereum are used to store the state after a transaction has been executed. These receipts are also used to record the cumulative gas used for the execution of the said transaction.

2.7 Transaction Execution

Execution of a transaction in Ethereum is defined by the state transition function Y. Before any transaction is executed, it has to go through an intrinsic test of validity [2]. This, according to Dr. Gavin Woods, includes:

1 The transaction is well-formed RLP (Recursive Length Prefix), with no additional trailing bytes;
2 The transaction signature is valid;
3 The transaction nonce is valid (equivalent to the sender account's current nonce);
4 The gas limit is no smaller than the intrinsic gas, g_0, used by the transaction. The intrinsic gas includes:

   - A predefined cost of 21,000 gas.

   - A gas fee for data sent with the transaction (txn). This fee greatly depends on the size of that is being sent with the txn (4 gas for each byte of zero data and 68 gas for each non-zero byte of data).

   - In case of contact creation txn, another 32,000 in gas.

5 The sender account balance contains at least the cost, v_o, required in up-front payment.

Only when a txn meets all of the above requirements is it considered valid. We then consider the upfront cost of computation for the transaction and deduct from the sender's account. The nonce for the sender is also increased by 1. Then, the transaction is executed and Ethereum tracks the transaction via substate which contains:

1 Self-destruct set: A set of contract accounts (if any) to be discarded after the execution of the txn. These contract accounts have a self-destruct function that is used to free space on blockchain by clearing up contract data.
2 Log series: archived and indexable checkpoints of the virtual machine's code execution.
3 Refund balance: the amount to be refunded to the sender account after the transaction.

Once the sender is refunded (a) miner is paid for the effort put in for the txn; (b) gas used is added to the gasUsed counter in the block; (c) data in self-destruct set is discarded; and (d) new state of Ethereum network is achieved.

3. Prediction Models

Machine learning provides explicit learning capabilities to a computing system by using statistical techniques. It is the study of programs and algorithms that can learn from historical data and based on that, it can make predictions for new data. A ML algorithm takes a set of input samples called training set and uses the set to learn in three fundamental ways: supervised, un-supervised and reinforcement learning [16].

- **Supervised Learning** algorithm also has knowledge of the target output for the training set, known as labels, and the algorithm learns the input to achieve the target.
- **Unsupervised Learning** has no such labels as in supervised learning. It aims to draw structure or pattern from a given input dataset.
- **Reinforcement Learning** deals with the problem of learning the appropriate action(s) in order to maximize payoff.

The focus of this thesis is entirely on supervised learning. Consequently, after defining certain machine learning terminologies, this chapter discuss *supervised learning* in detail. We then discuss some key concepts in supervised machine learning such as *Generalisation, Overfitting,* etc. We also present the evaluation criterion for machine learning algorithm, namely *classification* and *regression* algorithms, used in this thesis.

3.1 Terminology

This section introduces the basic terminology of machine learning which will be used in the rest of the thesis. For any supervised learning algorithm, there exists a *dataset* which is a collection of data in a set of rows and columns. Each row corresponds to a distinct instance of the data also known as a *training example* or an *instance*. The columns, on the other hand, are called *input variables, attributes* or *features.* Each dataset is also associated with one or more *target(s)*, *label(s)* or *output variables*. These target(s), label(s) or output variable(s) represent the result computed from the set of input variables.

The dataset in machine learning is typically divided into two subsets; *training set* and *testing set.* The training set is what is used by a machine learning algorithm to learn underlying features of the dataset while the testing dataset is used to evaluate how well the model learned the training set and how accurate its predictions are.

3.2 Supervised Machine Learning: Concepts and Definitions

In supervised learning, the dataset in question has two parts which are fed to a machine learning algorithm. These parts are: (1) a set of input instances $X = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3}, \dots, \boldsymbol{x_m}\}$ and (2) a set of target values $Y = \{y_1, y_2, y_3, \dots, y_m\}$. Each input instance $\boldsymbol{x_i}$ has a set of n features $\boldsymbol{x} = \{x_1, x_2, x_3, \dots, x_n\}$. Moreover, each feature x_i can either take real numerical values or categorical values belonging to an unordered set. These features can be modified to another form depending on the complexity of the problem at hand. This process is known as *data pre-processing*.

There also exists an unknown function $f^*(\cdot)$ that maps each input instance $\boldsymbol{x}$ to the best possible target value y. The main idea behind supervised learning is to approximate this unknown function, known as the true function, based on the given set of input X and output Y. This process of approximating $f^*(\cdot)$, using a function $f_\theta(\cdot)$ where θ is a set of parameters, is known as *learning*.

Typically, a supervised learning algorithm aims to learn the parameters θ of the function $f_\theta(\cdot)$ by minimizing the prediction error made by the model. Formally, the function that maps the relative discrepancies between estimated and actual target values into a real number is known as a *loss function* [16].

Generally, a learning process can be defined as the process of finding the best parameters θ in order to minimize loss function over all instances of the dataset. Formally, the learning process is as follows:

$$\hat{\theta} = argmin_\theta \left\{ \sum_{i=1}^{m} l(y_i, o_i; \theta) \right\},$$

where $\hat{\theta}$ is the learned set of parameters; y_i is the set of target labels and o_i is the output given by the learning algorithm for i^{th} input instance.

Whenever the target values are an unordered set of discrete values, the machine learning task is known as a *classification* problem. Whereas, when the target label is a set of continuous values, the task is known as a *regression* problem. The two learning algorithms are discussed in detail in subsequent sections.

3.2.1 Classification

In machine learning, classification is defined as the task of predicting the category to which the data instance in question belongs to. These predictions are made based on one or more independent variables in the dataset. The output given by a classification algorithm is discrete and belongs to an unordered set of C distinct classes *{1, 2, ..., C}*.

In order to make predictions about the new examples, the model can either output a class label from the set of classes or can output a set of probabilities. Each probability would

correspond to a distinct class and indicates the probability that the given instance belongs to a specific class. In classification models that provide probabilistic outputs, either the class with the highest probability is chosen to be the predicted label or the class label is drawn by sampling the output distribution. Fig 3.1 illustrates a simple binary classification.

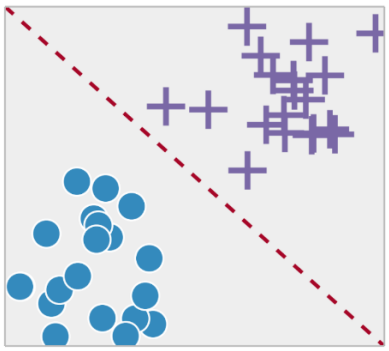

Fig 3.1 A simple binary classification.

3.2.2 Regression

Regression, in machine learning, is defined as a statistical process of prediction target values given a set of input instances when the target label in question is a continuous quantity. It aims to find a relation between different variables of the dataset and output a set of continuous numeric values. Fig 3.2 illustrates an example of regression.

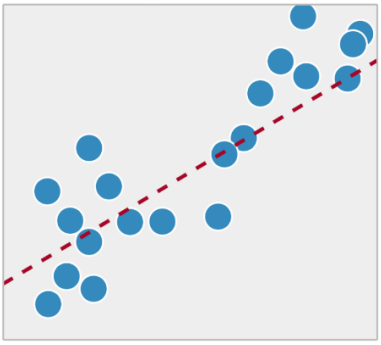

Fig 3.2 A simple regression instance.

In regression problems, Mean Square Error (MSE) is one of the most commonly used loss functions used to evaluate model performance. The algorithm aims at minimizing this loss function which maps the variation between the numerical values predicted by the model and those observed or *true targets.* Formally,

$$MSE(O,Y) = \sum_{i=1}^{n} \|o_i - y_i\|_2^2$$

3.3 Generalisation

Machine learning aims to develop models that are able to make predictions for unforeseen examples. Henceforth, generalisation to new instances is an important part of learning algorithms. We seek models that can work accurately not just with the training dataset but also the testing set. Therefore, instead of simply memorizing the training set, we want the algorithms to learn the underlying factors of variation.

3.3.1 Bias-Variance Trade-off

Bias and variance of a machine learning algorithm helps in formal understanding of the reason behind errors encountered during the prediction task. To understand bias and variance in machine learning, we assume that the model in question can be trained multiple times with different randomly selected datapoints. Errors in prediction, caused due to bias and variance, are known as errors due to bias and errors due to variance respectively [17] [18].

Let $f(x)$ be the prediction model; bias can be defined as the difference between the expected output of the model and the target value [19]. Formally,

$$bias = E[f(x)] - y$$

where $E[\cdot]$ is the expected value and y represents the true target. Variance, on the other hand, is the variability in different predictions of multiple trained instances of the model [19]. Formally,

$$variance = |f(x) - E[f(x)]|^2$$

The total error observed in the model, in terms of bias and variance, is given as;

$$error = E[(f(x) - y)^2] = bias^2 + variance$$

Given that the amount of data available is limited, there is always a trade-off between bias and variance. One may decrease one at the cost of the other. Therefore, the error can only be minimized by finding a balance between the two. Fig 3.3 illustrates and example of trade-off.

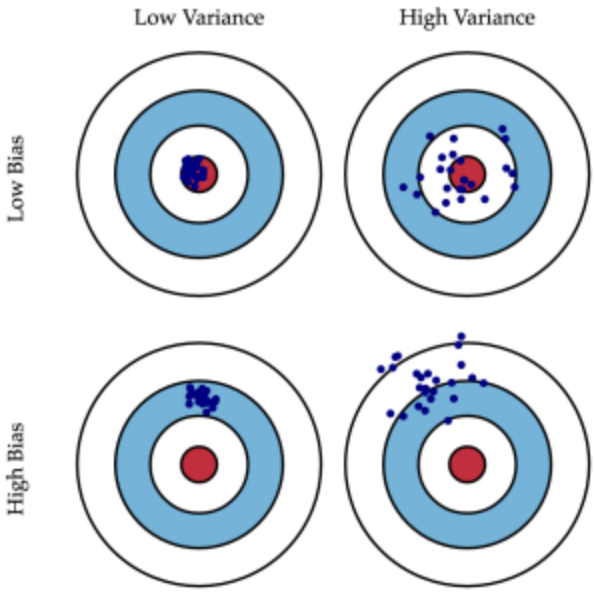

Fig 3.3 Dart chart: A graphical illustration of bias-variance trade-off. Consider a classification problem as throwing darts at a board. If darts land in very different parts of the board, the model has "high variance". If their mean is close to the centre of the board, the model has "low bias". Similarly, "low variance" and "high bias" can be defined. The above four dart boards correspond to these situations in [20].

3.3.2 Overfitting Problem

Overfitting is a problem encountered in machine learning where the model learns the training data so well but is unable to make prediction for the unseen instances from the testing dataset. It means that the model has learned everything from the training dataset including noise instead of learning the underlying relations among variables in the dataset. According to Bishop [21], overfitting can occur due to limited availability of the dataset or too many model parameters. Another reason for overfitting can be the presence of imbalanced dataset which is a problem encountered mostly in classification problems; it has been discussed with respect to the classification model proposed in this thesis in later sections. Fig 3.4 illustrates overfitted, underfitted and a fit model.

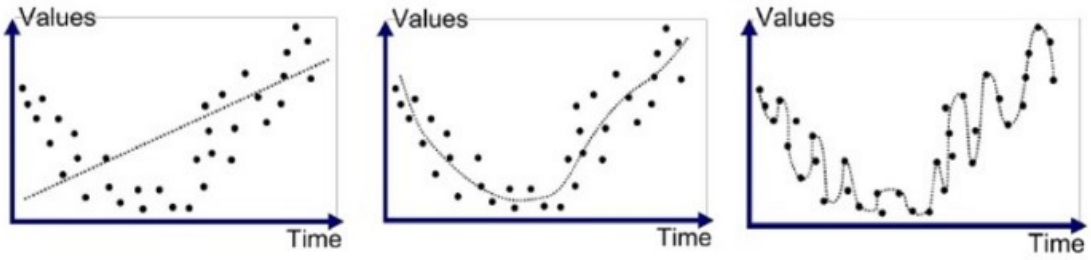

Fig 3.4 Left: the model is underfitted or equivalently has high bias. This is because a linear function was used to approximate a second order polynomial function. Right: the model is overfitted because a high order polynomial function is used. This model has high variance. Middle: the model is just fitted. The Figure is adopted from Bishop [21].

Overfitting Due to Unbalanced Data

A dataset is said to be imbalanced or skewed if majority of instances belong to one class than other(s). In such cases, the classifiers would be inclined to always predict the majority class rendering the model weak and inaccurate. This type of overfitting simply happens because the classification objective assumes that errors from different classes have the same costs [22].

3.4 Performance Evaluation for Classification

It is important to evaluate the strength of any given model. Error metrics help us determine how good the model will perform when exposed to unseen data. This section gives a brief description about the error metrics used for performance measure of a classification model.

3.4.1 Accuracy

Accuracy of a model is defined as the percentage if correct predictions made by a model. Formally,

$$Accuracy = \frac{\#\ correct\ predictions}{total\ examples}$$

However, with imbalanced data, one must face the accuracy paradox. If a class is abundant in a dataset, the model may be tempted to always classify each entry with this class. The accuracy may be high, but the model may not be able to find the entries of the under-represented class(es). In those cases, the accuracy must be compared with the null accuracy, which is a metric defined as the percentage of times the majority class is predicted by the model irrespective of the target for a given example. Formally,

$$Null\ Accuray = \frac{\#\ majority\ class\ predictions}{total\ examples}$$

3.4.2 Kappa Score

Cohen's Kappa is a statistical measure used to assess the reliability of an agreement between a fixed numbers labels whilst assigning categories. In case of more than two labels, we utilise Fleiss Kappa. It tells us how much better our model is performing over the performance of another model which will simply guess the result based on the frequency of each class. Formally,

$$\kappa = \frac{\bar{P} - \bar{P_e}}{1 - \bar{P_e}}$$

Where, $\bar{P}$ represents the relative observed agreement among the raters and $\bar{P_e}$ is the hypothetical probability of agreement. The factor $1 - \bar{P_e}$ provides degree of agreement that is attainable and $\bar{P} - \bar{P_e}$ gives the degree of agreement actually achieved. A value of 0 indicates that the classifier is useless. The interpretation of the Kappa score is given in Table 4.1 [23].

TABLE 3.1: INTERPRETATION OF KAPPA SCORE.

Value of Kappa	Level of Agreement	% of Data that is Reliable
<0	None	0-4%
0.1 – 0.20	Minimal	4–15%
0.21 – 0.40	Weak	15–35%
0.41 – 0.60	Moderate	35–63%
0.61 – 0.80	Strong	64–81%
Above 0.90	Almost Perfect	82–100%

3.5 Performance Evaluation for Regression

These error metrics help measure the strength and accuracy of a regression model when it is introduced to unseen data. These metrics are: *Mean Absolute Error, Root Mean Square Error, Mean Absolute Percentage Error* and *Prediction Value (PRED)*.

3.5.1 Mean Absolute Error

Mean Absolute Error (MAE) is defined as the mean of absolute error or difference between predicted and actual values. Mathematically, this can be defined as follows:

$$MAE = \frac{1}{n}\sum_{i-1}^{n}|act_i - est_i|$$

3.5.2 Root Mean Square Error

RMSE represents the standard deviation of the magnitude difference between predicted and actual values. It measures the square root of the average of the squared difference between predicted values and the true values. Formally,

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(act_i - est_i)^2}$$

3.5.3 Mean Absolute Percentage Error

MAPE is percentage equivalent of MAE. It is most commonly used as a loss function in regression problems and model evaluation due to its intuitive interpretation of relative error. Formally,

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{act_i - est_i}{act_i}\right| \times 100\%$$

3.5.4 Prediction Value (PRED)

PRED(n) represents the percentage of absolute percentage errors that are less than or equal to the value n among N transactions.

$$PRED(n) = \frac{1}{N}\sum_{1}^{N}\begin{cases} 1, & if\ MAE \leq n \\ 0, & otherwise \end{cases}$$

In this chapter, we discussed, in detail, the fundamentals behind supervised learning and presented some key concepts in supervised machine learning, such as Generalisation, Overfitting, etc. We also presented the evaluation criterion for the two machine learning approaches, namely classification and regression algorithms, used in this thesis. The following chapter presents the methodology employed in this work. We discuss the classification and regression models employed as well as the characteristics of the Ethereum blockchain dataset.

4. Methodology

Ethereum is a state-based machine. It changes the state of the network with every block confirmation and transaction made. Due to the volatile nature of the network, any prediction model must be able to keep up with the current state of the network in real time.

In this chapter, we propose machine learning models and techniques to tackle the transaction confirmation time prediction problem; we consider both classification and regression approaches. In case of classification, we present three machine learning algorithms that will be able to predict an approximate time window in which a transaction would be confirmed to a block. After general dataset analysis in terms of classification, we observed that the dataset was highly skewed meaning that most of the transactions belonged to one class over the others. Thus, dealing with an imbalanced classification problem, we made use of techniques to make the dataset more balanced. On the other hand, in case of regression approach, we present two machine learning algorithms that would be able to predict the time it would take for a transaction to be confirmed in terms of real values as opposed to the categorization done in classification.

In section 4.1, we will briefly discuss the limited research work done on the subject in the past and its limitations. In section 4.2, we analyse our dataset and understand its characteristics, features and class distribution, in case of our classification approach, and then present techniques to overcome those limitations. In section 4.3, we present the two approaches we adopt to tackle the imbalanced nature of our classification problem: algorithmic approach (Ensemble methods in Section 4.4) and the data-based approach (Re-sampling techniques in Section 4.5). Section 4.6 presents the three machine learning algorithms that we adopt in order to tackle the confirmation time prediction problems for both the classification and regression approach.

4.1 Related Work

Due the volatile nature of the Ethereum transaction dataset, its predictability has been sparsely covered in published literature. However, this section briefly discusses the work involving Ethereum transaction confirmation time prediction and their limitations.

Eth Gas Station [24] proposed a Poisson regression model to estimate the expected number of blocks it would take for a transaction to be confirmed based on the amount of

gas used by the transaction and the gas price. The model outputs real values (in seconds and in blocks). The model makes its estimations based on the data from the last 10,000 blocks at any given time. The model uses statistical analysis and outputs the confirmation time prediction based on the percent of blocks that had a similar transaction confirmed within them in the past. The Poisson regression model [24] is periodically retrained to keep up with the most recent state of the Ethereum network. The model does not take into consideration the transactions in the past that are still pending and how the data from those transactions could relate to current transactions. Their proposed statistical model is used as the basis of this thesis. It is used to analyse which of the two, machine learning or statistical modelling, would be more efficient for the prediction problem.

4.2 Dataset Analysis

The initial dataset consists of about one million transactions that were made on Ethereum with the last transaction in the set dating at November 18, 2018. The data was extracted from etherscan.io API [25] and included the gas price, gas limit set by the user, gas used, timestamp for when the transaction was confirmed and when the transaction was made as well as the number of transactions made by the sender. While there is no given field in the API that would suggest when the transaction was made, we wrote a python script to extract the timestamp from the API's pending transaction pool [26]. These transactions were used as the historical data to train the model before the actual time estimations were made.

For the purpose of classification, we divided the confirmation time in 8 categories: within 15 seconds (or approximately 1 block time), within 30 seconds (approximately 2 blocks time), within 1 minute (approximately 4 blocks time), within 2 minutes (approximately 8 blocks time), within 5 minutes (approximately 20 blocks time), within 10 minutes (approximately 40 blocks time), within 15 minutes (approximately 60 blocks time), within 30 minutes or longer. The classes were defined based on the Ethereum network trend of creating new block about every 15 seconds. Most of the transactions (49.4%) belonged to class one (confirmed within 15 seconds) with only 0.112% of the transactions having to wait more than 30 minutes before being confirmed by a mining node. This implies the skewedness of the dataset. In general, classification algorithms have an affinity towards always predicting the majority class. This imbalanced nature of the dataset is why it is vital to adopt some kind of balancing techniques to ensure accurate predictions by the classification model.

It was also observed that out of the million transactions, about 1% of these were reverted transactions or failed. This can be visualised in Fig 4.1 while Table 4.1 presents the categorization of the dataset.

TABLE 4.1: STATISTICS OF DATASET

Class	Number of transactions
1. Within 15 seconds	494,071
2. Within 30 seconds	248,609
3. Within 1 min.	173,047
4. Within 2 min.	38,918
5. Within 5 min.	33,951
6. Within 10 min.	31,349
7. Within 15 min.	8,934
8. Within 30 min. and longer	1,121

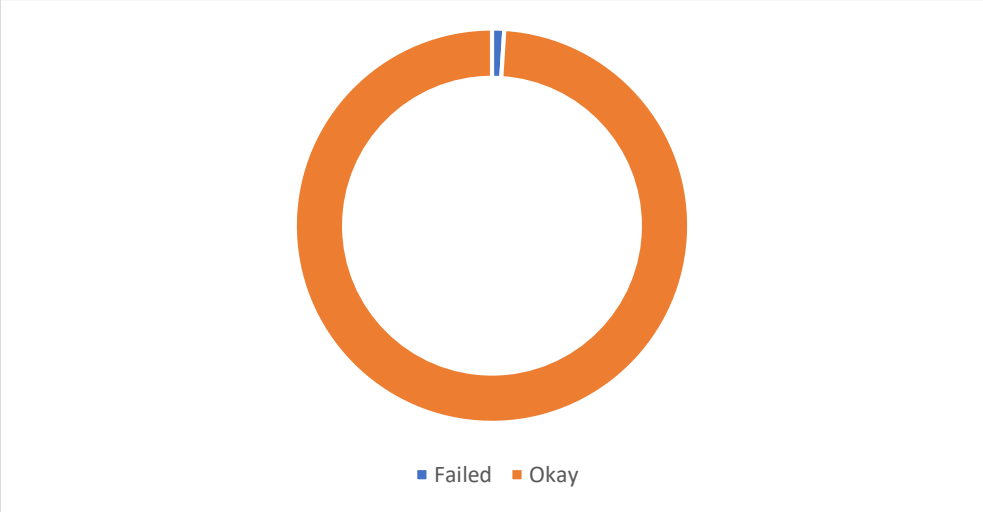

Fig 4.1 Total to okay and failed transactions.

In order to classify a transaction into one of the 8 aforementioned classes, the independent variables which includes gasPrice, gasLimit, value and receipt_status are used to determine the class to which the transaction would belong.

On the other hand, in terms of regression, in order to predict discrete valued output defining the time taken by a mining node to confirm a transaction, no categorisation of the dataset is needed. The independent are simply passed through a regressor to determine the

dependent variable, conf_time. This dependent is the difference between the two timestamps: (1) transaction's introduction to the network and (2) the timestamp for its confirmation. Table 4.2 lists the variable in the input and output vector for our machine learning model.

TABLE 4.2: DEPENDENT AND INDEPENDENT VARIABLE OF THE DATASET

Variable	Type	Description
gasPrice	Independent	Gas price provided by the sender in Wei.
gasLimit	Independent	Represents the maximum amount of gas that should be used to execute a transaction.
value	Independent	Value transferred in Wei
receipt_status	Independent	Either 1 (success) or 0 (failure). This marks whether the transaction was successfully confirmed or not.
timestamp_0		Timestamp for when the transaction was added to the pending pool.
timestamp_1		Timestamp for when the transaction was confirmed by the mining node.
conf_time	Dependent	Difference between timestamp_0 and timestamp_1. This is used as the dependent variable in case of regression.
class	Dependent	In case of classification, valued between 1 to 8. This defines the class to which a transaction belongs.

When a user wishes to make an exchange or execute a contract, initially, the data would have to be converted into a raw transaction data. This raw transaction contains the gasPrice, the gasLimit set by the user, the destination addresses as well as value; this value is the total amount of Ether they wish to send. It would then include the data into it which would be the hash of the function in the contract that the account holder wishes to execute.

In order to ensure that the transaction is made by the account holder and not by an unauthorized entity, the transaction would then have to be signed with the user's private key. This is also used to ensure accountability of the sender.

The transaction is then broadcasted to the Ethereum network and the local geth (or parity) node will generate a transaction ID that the user will be able to use to track its status. These geth (GO-Ethereum) or parity (Parity-Ethereum) nodes run Ethereum protocols that define how a user would operate, how the network works and the rules each user must follow to be a valid part of the Ethereum network [27]. These geth and parity nodes communicate with other nodes on the network and hold the ability to mine new transactions, to validate transactions in the block, to add new transactions to the block as well as to execute them. Fig 4.2 illustrates the broadcasting process.

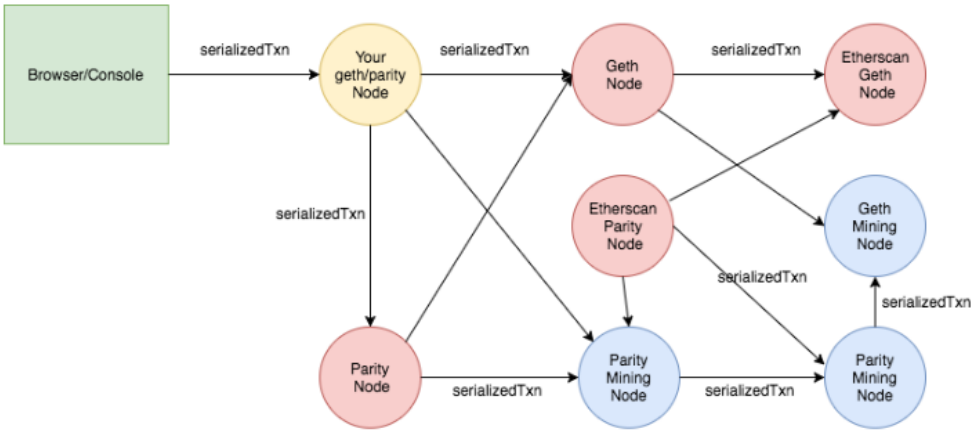

Fig 4.2 Signed Transaction propagating through the network. The transaction is broadcasted by sender's local geth (or parity) node to its peers who then broadcast it to their peers and so on until the whole network has a signed copy of the transaction. This figure is adopted from M. Murthy[4]. The Etherscan geth and parity nodes are responsible for updating the etherscan API [25].

As discussed in Section 2.1, in the Ethereum network, some nodes work as full nodes or mining nodes. These nodes pick up a transaction and put in the effort to include the said transaction into a block. Mining nodes have a transaction pool where each transaction exists as pending before it is picked up for evaluation. These transactions are stored in the pool as per the gas price associated with each one of them. Higher the price, more likely is the node to evaluate it first. A sample pending pool is illustrated in Fig 4.3.

[4] Figure source: Murthy, M., Life Cycle of an Ethereum Transaction, Medium, https://medium.com/blockchannel/life-cycle-of-an-ethereum-transaction-e5c66bae0f6e, last accessed 2019/04/20.

Fig 4.3 Mining node's pending transaction pool.

It should be noted that the pending transaction pool can only hold a limited number of transactions. If account holders keep on generating transactions with higher gas price, as is the trend, the transaction with the minimum gas would be discarded and the transaction would have to be re-broadcasted to the network.

Next, once the miner has picked up the transaction to include in the block, the transaction would be validated, included into the pending block and the proof of work begins. Then, once the block is added to the blockchain, the valid block is then broadcasted to the entire network by the mining node.

Finally, the local node will receive the new block and will then sync its local copy of the blockchain. It would then execute all the validated transactions in the block.

In order to build a model that keeps up with the ever-changing state of the Ethereum network, we wrote a python script with the purpose of monitoring a local Geth node. This was done to extract data about mined as well as pending transactions. This allowed the model to access real-time data and make predictions based on current market trends rather than simply relying on historical data.

The data extracted from monitoring the mining node, again, included gasPrice, gasLimit, value, timestamp for when the transaction was confirmed as well as the timestamp for when the transaction was added to the pending pool of the geth node. The two timestamp variable were used to calculate the dependent variable for the regression model. This variable represented the time that it took for the mining node to confirm a transaction once it was added to its pending pool. This was calculated as the different of the two independent timestamps.

It should be noted that while on average, it takes up to 2 blocks for a transaction to be confirmed, it varies depending on network traffic as well as the number of transactions that

a mining node is assigning to a block. Another parameter that affects the confirmation time for a transaction is the gas limit the user has imposed on the transaction in question. A mining node gets an incentive for working on each transaction (see Chapter 2, Section 2.1). Higher the price the user is willing to pay, the higher the incentive the miner would get for the effort put in to perform computations on the transaction. Therefore, the mining node would be inclined to confirming such transactions over others leading to lower confirmation time. The variation in number of pending transactions per minute can be observed in Fig. 2 [28] and the block confirmation time variation can be observed in Fig. 3 [29].

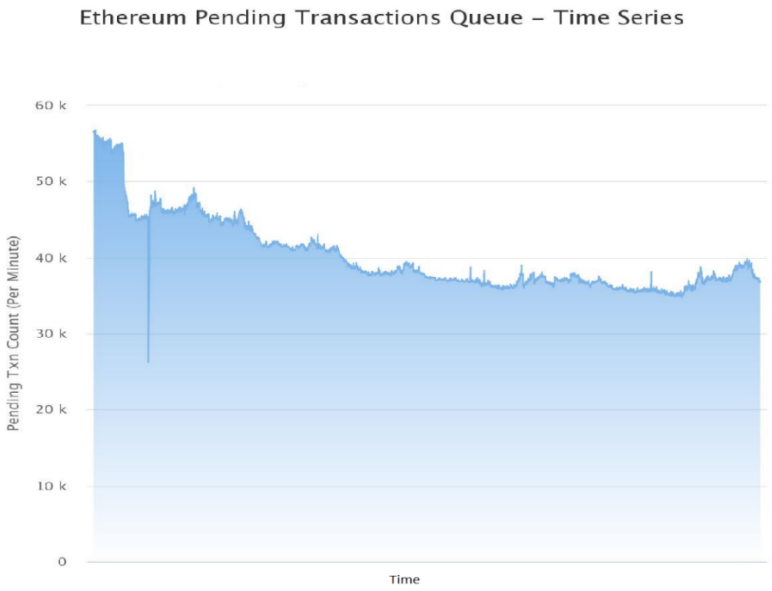

Fig 4.4 Ethereum pending transaction queue (per min.).

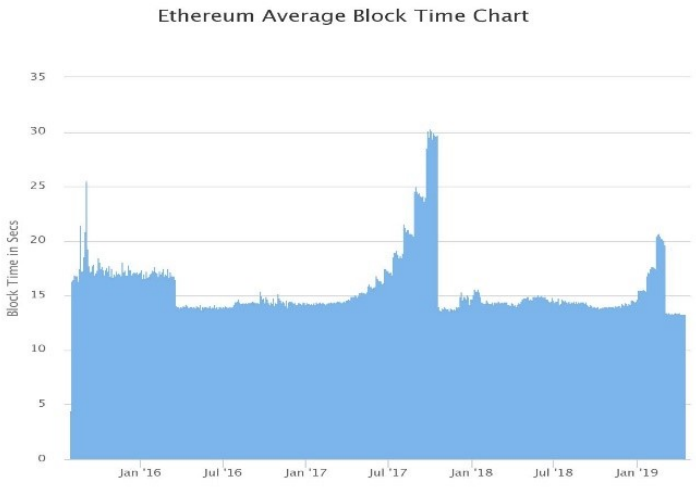

Fig 4.5 Ethereum Average Block time chart. (in seconds).

4.3 Tackling Unbalanced Data

Classification algorithms, in general, assume that the class distribution is balanced, either implicitly or explicitly [30]. Therefore, employing any classification algorithm on unprocessed skewed dataset would lead to an inaccurate classifier.

Moreover, in a simple classification model such as a Naïve Bayes model or a Support Vector Machine (SVM) the probability that one of the minority classes would be predicted is almost zero. Whereas, in case of complex classifiers such as Decision Trees, the feature selection criterion would simply ignore those minority classes [31]. Given a small amount of data for one class, feature selection methods could easily fail as there is no significant change in model performance by adding or eliminating a feature [30].

Assuming that these minority classes are the main cause of inaccurate classifiers, for the classification approach towards predicting confirmation time, we mostly focus on adapting our learning algorithm to accommodate the minorities. We will look into two approaches; (1) Algorithmic approach and (2) Data centric approach. These are discussed in detail in subsequent sections.

4.3.1 Algorithmic Approach

The algorithmic approach for handling imbalanced class distribution uses ensemble methods. Ensemble method is a form of regularization technique that compensates for the drawbacks of a simple classifier by designing a complex structure of multiple classifiers. It aims to train multiple instances of the classifier and aggregate their outputs to get the final result with higher predictive accuracy. In terms of bias-variance trade-off (see Chapter 3), combining and then aggregating multiple classifiers is meant to help reduce the high variance observed in a classifier due to skewed or imbalanced dataset. One of the most important ensemble method algorithms is Random Forest which is based on multiple instances of Decision Trees. Random forest is introduced in detail in Section 4.6.3.

4.3.2 Data-centric Approach

Data-centric approach towards handling imbalanced dataset is based on re-sampling the dataset in order to make the classifier equally biased to the minority class(es) as it is towards the majority class. There are two types of re-sampling techniques: (1) Under-Sampling and (2) Over-Sampling. In under-sampling, the algorithm would delete datapoints from the majority class whereas in over-sampling, the algorithm creates new data points for the minority class(es).

4.4 Ensemble Methods

The main idea behind ensemble methods is to generate complex classification models that combine prediction results of much simpler ones. These simpler models are known as base models. In general, base algorithms used in ensemble methods are Naïve Bayes, Decision Tree and Support Vector Machines. Ensemble methods aim to improve predictive performance of a model by reducing bias, variance or both [32].

The two common ensemble methods towards combining classifiers are: Bagging and Boosting. There are discussed in subsequent sub-sections and are illustrated in Fig 4.6.

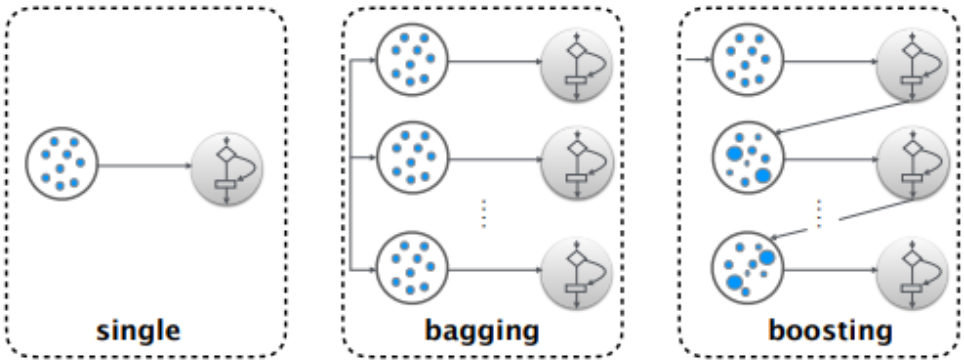

Fig 4.6 Left: Represents a single classifier where the whole dataset is fed to the model in a single instance. Centre: Represents bagging where multiple variants of the dataset are created, each different from the other in a randomly sampled manner and are then fed to a number of models. The result is an aggregation of results from each model instance. Right: Represents boosting where random sampling with replacement over weighted dataset is used.

4.4.1 Bagging

Bagging aims to reduce variance and avoid over-fitting [32]. It generates several same sized instances of the original dataset by sampling it *with replacement*. There may or may not be some datapoints from the original dataset that are repeated in these instances.

This process is known as bootstrapping. Bootstrapping increases the size of the training set which reduces the variance. The sets are fed to the model and their results are combined by either averaging (regression) or choosing the majority class (classification) [33].

4.4.2 Boosting

Boosting aims to reduce both bias and the variance [32]. It aims to combine weak learner to generate a stronger one. At each training instance, the dataset is fed to a weak model which then gets added to a stronger model with weight associated with its accuracy. Each time the output from a weak learner is added to next, stronger one, the datapoints that have

been classified correctly, lose weight. This allows for the model to focus on misclassified datapoints. The process is repeated n times and the result will be a combination of n weak learners as a strong learner [34].

4.5 Data Resampling

As discussed in section 4.3.2, re-sampling is a data-centric approach towards tackling imbalanced data. It involves generation of samples of the dataset such that there is no bias in the model towards one class over others. Some of the generally used re-sampling techniques are discussed below.

4.5.1 Random Over-sampling

In random over-sampling, the algorithm aims to generate new datapoints for the minority class. These new datapoints are simply copies of some randomly selected datapoints from the original dataset. The algorithm keeps on generating new datapoints until all the classes in question have same number of datapoints.

4.5.2 Random Under-sampling

In random under-sampling, the algorithm aims to randomly delete datapoints from the majority class. The process continues until all classes have same number of datapoints. One drawback of this technique is that it might delete the datapoints in the decision boundary that are important in the process of decision making.

4.5.3 SMOTE

Synthetic Minority Over-sampling Technique (SMOTE) [35] is a re-sampling technique that utilizes interpolation between current datapoints to generate new datapoints in the minority class. A major drawback of this technique is that it might also generate new datapoints for the majority class.

4.6 Prediction Models

In this section, we introduce the machine learning models used in the thesis for the prediction of confirmation time in Ethereum blockchain network. These models are: Naïve Bayes Classifier, Decision Tree and its implementation: Random Forest as well as Multilayer Perceptron (MLP).

4.6.1 Naïve Bayes Classifier

Let K be the number of classes in a distribution. The conditional probability of class k given a vector of n distinct features $\boldsymbol{x} = \{x_1, x_2, \dots, x_n\}$ can be written as $P(C_k|\boldsymbol{x})$. According to Bayes theorem [36], this probability can be formulated as:

$$P(C_k|\boldsymbol{x}) = \frac{P(\boldsymbol{x}|C_k)P(C_k)}{P(\boldsymbol{x})}$$

where, $P(C_k|\boldsymbol{x})$ is the posterior, i.e., our updated knowledge conditioned on the observed data while $P(\boldsymbol{x}|C_k)$ and $P(C_k)$ are called the likelihood and the prior respectively.

In terms of classification, $P(\boldsymbol{x})$ remains same for all classes and the training of a classifier is meant to be such a way that the nominator is maximized for the target class and minimized for all others [37]. This nominator is the joint probability of all the classes C_k and the features $\boldsymbol{x}$; $P(C_k, x_1, x_2, \dots, x_n)$. This, by chain rule [38], can be formulated as:

$$\begin{aligned} P(C_k, x_1, x_2, \dots, x_n) = P(C_k) * \\ P(\boldsymbol{x_1}|C_k) * \\ P(\boldsymbol{x_2}|x_1, C_k) * \\ P(\boldsymbol{x_3}|x_2, \boldsymbol{x_1}, C_k) * \\ \dots \\ P(\boldsymbol{x_n}|\boldsymbol{x_{n-1}}, \dots, \boldsymbol{x_1}, C_k) \end{aligned}$$

Consequently, since $P(C_k|x_1, \dots, x_n) \propto P(C_k, x_1, x_2, \dots, x_n)$, a classifier can be formally defined as:

$$\hat{c} = \underset{k \in \{1,2,\dots,K\}}{\operatorname{argmax}} P(C_k, x_1, x_2, \dots, x_n)$$

In practice, training such a classifier with large number of features can be complex and challenging. A simpler yet effective variant of a probabilistic classifier is a Naïve Bayes classifier [39]. It assumes that given the class label C_k, all features $\boldsymbol{x} = \{x_1, x_2, \dots, x_n\}$ are independent from one another. A graphical illustration of a Naïve Bayes classifier is given in Fig 4.7.

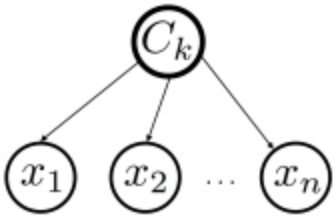

Fig 4.7 Bayesian network representation of the naive Bayes classifier. According to the graph representation, conditioned on the class C_k,, all the features $\boldsymbol{x} = \{x_1, x_2, \ldots, x_n\}$ are independent of each other.

4.6.2 Decision Tree

Decision tree is a decision support tool that employs a tree-like structure of decisions as well as their consequences. It operates like a flow-chart where each node represents a test on an attribute and each branch represents the outcome of the test. The leaf node represents a decision on the numerical target [40].

A simple example [41], consider a set of features *{Outlook, Humidity, Wind}* and the target is *Play Tennis* which takes values of *"Yes"* or *"No"*. A decision tree is illustrated in Fig 4.8

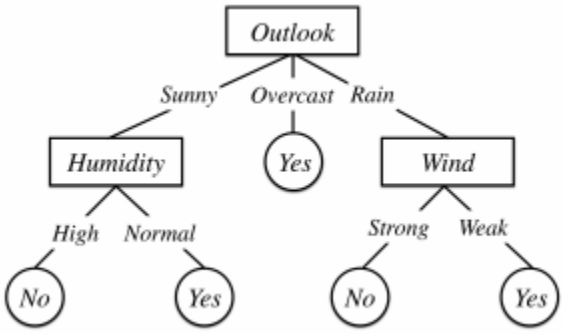

Fig 4.8 A graphical representation of a decision tree classifier. The classification starts from the top and moves downwards to the leaf nodes which represent the outcome of whether playing tennis would be a possibility or not. The example has been adopted from Mitchel [41].

4.6.3 An Ensemble of Decision Trees: Random Forest

Random Forest is an ensemble learning method. The algorithm is based on combining decision trees to build a stronger prediction model. A decision tree leads to hierarchical partitioning of the feature space. Starting from the root node, each node divides the feature space in two or more partitions. The partitioning becomes more and more complex with each new decision node. This can lead to over-fitting where in, due to the complexity of

partitions, the training set yields almost no error whereas the testing set yields more or less very inaccurate results.

This is where combining decision trees can be helpful [40]. Random forest is an ensemble of decision trees in which the final prediction is the result of aggregation of the output from each of the individual decision trees.

Given a training set $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$ and a corresponding set of targets $Y = \{y_1, y_2, \ldots, y_m\}$ where $\boldsymbol{x}_i$ has a set of n features $\boldsymbol{x} = \{x_1, x_2, \ldots, x_n\}$, the following steps are done k times to train a random forest model:

- **Select a random set of samples with replacement from X called X_k.**
- **Build a decision tree on X_k which is trained on a random subset of features ($n' < n$).**

This process can be visualized in Fig 4.9. For classification task, the prediction of class for an unseen sample can be done by choosing the majority class from each of the *k* decision trees. Whereas, for regression, the output prediction can be the aggregation of the output of all the *k* decision trees.

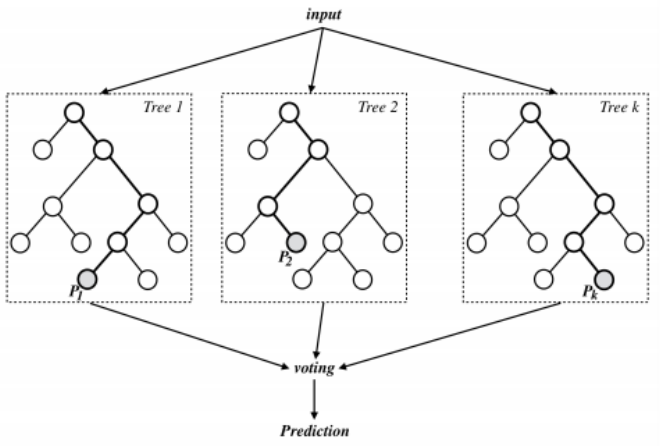

Fig 4.9 In a random forest, *k* different decision trees are trained using *k* different subsets of the dataset. During test time, a sample input point is fed to all trees and predictions P_1, P_2, P_3,…, P_k are generated. A voting is then applied on all predictions to make a single final prediction.

The performance of a random forest model is mostly dependent on two factors [42]: correlation between two trees and the accuracy of each tree. By increasing the feature subset (*n'*) size, the tree's accuracy would increase but so would the correlation. Hence, n' should be optimized. Random forest model, in general, performs well at learning complex, highly non-linear relationships; like between time and both gas price and gas used in Ethereum blockchain dataset. The model is known to outperform fundamental

classification and regression models like naïve Bayes, polynomial and linear regressors [43].

4.6.4 Multilayer Perceptron (MLP)

A perceptron is a linear classifier that classifies input by separating two categories with a straight line [44]. The perceptron produces single output based on linear combination of several revalued inputs and input weights. Formally,

$$y = \varphi\left(\sum_{i=1}^{n} w_i x_i + b\right) = \varphi(\boldsymbol{w^T x} + b)$$

where, x_i represents the inputs to the unit, w_i represents the weights, b represents the bias and φ is the non-linear activation function. In terms of classification, this non-linear activation function φ is a hard threshold at zero. Whereas, in terms of regression, it uses a linear model such that $\varphi(z) = z$ [44].

A perceptron in itself, is single layered. It is unable to formulate model hierarchy as in an artificial neural network. It is, therefore, shallow neural network that is unable to perform non-linear operation such as XOR function [45]. The term shallow implies that the neural network does not have more than one or two hidden layers.

MLP is an adaptive neural structure made up of at least three layers constituting an input layer which is made up of a number of perceptions equal to the number of attributes of the dataset. The network must have at least one hidden layer and an output layer made up of one perceptron in case of regression and a number of perceptrons equal to the number of classes in case of a classification problem. Every unit in one layer is connected to every unit in the next layer making it a fully connected network. These layers communicate among one another via synaptic connections represented by weights [46]. Each layer except the input layer has a non-linear activation function.

MLP is a supervised learning technique which learns a function $f(\cdot) : R^n \rightarrow R^o$ by learning a dataset with n input dimensions and o output dimensions. Training involves adjusting parameters, weights or biases for the model with an aim to minimize error. These adjustments are generally made via *Backpropagation.*

MLPs, in general, have two ways of operation: Forward pass or Backward pass. In forward pass, the signal flows from input layer through hidden layer to the output layer where the decision accuracy is measured against target labels. In backward pass, however, the chain rule of calculus i.e., partial derivatives w.r.t. the weights and biases are back propagated through MLP. This differentiation provides us with a gradient along with which the parameters can be adjusted further so as to minimize error. The process continues until the error measure reaches a set limit and the model is said to be a convergence state. Fig. 4.10 shows a simple MLP regressor.

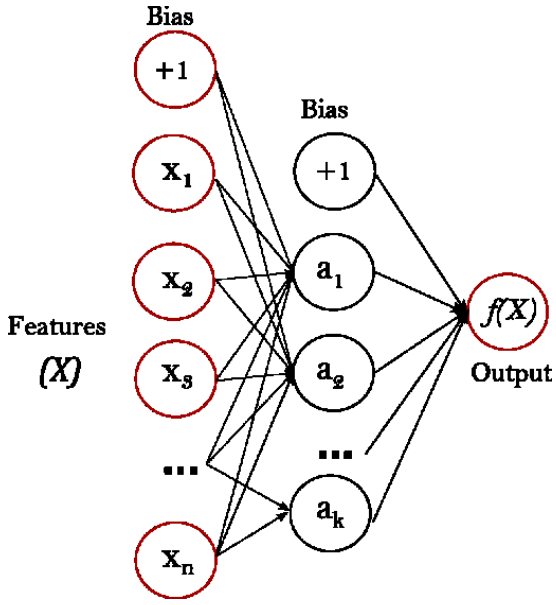

Fig 4.10 A simple MLP regressor.

MLP can work with as many hidden layers and parameters as one might require, all with non-linearities between them. This allows the model to be able to learn any complex non-linear relationship in a dataset, as observed with random forest. MLP is known to be flexible and adaptive to any feature-variable relationships and hence, does not require one to stress over the structure of the network [43].

This chapter presented the limited research work that has been done in order to make confirmation time predictions in Ethereum blockchain as well as the machine learning models employed in our thesis. We also discussed the techniques to tackle the problems associated with our classification approach and the skewed nature of the dataset. The following chapter discusses the complexities of the three models proposed in this chapter and presents the results observed from our experiments with the three models in terms of both classification and regression approaches.

5. Results and Discussion

In the previous chapters, several methods and algorithms for confirmation time prediction in Ethereum blockchain network have been introduced in terms of both, classification and regression. In this chapter, Section 5.1 explains the toolkit and environments used for our experiments. Section 5.2 discusses the computational complexities of the machine learning techniques used in this thesis as well as the statistical regression technique that is currently being used [24] to make prediction for Ethereum network. Section 5.2 implementation details of the classification approach. We report and analyse the performance of several machine learning and re-sampling algorithms in Section 5.3. Section 5.4 discusses the model implementation details of regression approach. Then, in Section 5.5, we analyse the performance of the machine learning regression algorithms. Finally, in section 5.6, we discuss our results and observations.

5.1 Environments and Toolkits

In this section, we present the details about the programming environment and the toolkits that were employed during our experiments. Both the classification and regression implantation of our problem statement were set up in Python details of which are discussed in the following subsections.

5.1.1 Python

Python is a general-purpose, interpreted, dynamic and most widely used programming language when it comes to data analysis or any of the related problems. It offers a robust collection of scientific, mathematical and statistical tools that allows for easy implementations of varying degree of machine learning algorithms.

It offers many distinct libraries, each with their own specific features and capabilities. These libraries include, but are not limited to, *NumPy* (Python's Numerical library) [47], *SciPy* (Scientific library) [48], *scikit-learn* and *Imbalanced-learn*. In our work, we employ Python 3.0 among other libraries facilitating data and model performance analysis.

5.1.2 Scikit-learn

Scikit-learn [49] is a machine learning oriented python library build atop NumPy and SciPy. It offers varying degree of data-mining and analysis tools and is open source. It is capable of implementing multiple machine learning algorithms for the fundamental machine learning tasks of classification, regression or clustering.

5.1.3 Imbalanced-learn

Imbalanced-learn [50] is a python library built atop scikit-learn, SciPy and NumPy that offers many re-sampling techniques to handle data imbalance such as over-sampling, under-sampling, SMOTE etc.

5.2 Computational Complexities

The computational complexity of an algorithm can be defined as the measure of resources required for running it. For a given instance or input vector [51], it is the measure of the number of steps required to compute the output, assuming this to be the worst-case scenario. In general, complexity of machine learning algorithms is highly dependent on their implementations, dataset properties, etc. [52]. While discussing the complexity of the two machine learning algorithms employed in this paper, we consider upper bounds in case of dense data as is the case with our dataset.

Let n denote the number of instances or transactions in our training subset, p denote the number of parameters passed as input vector (seven in our case), h_{li} denote the number of nodes in each layer i of MLP, e denote the number of epochs and t denote the number of trees in the random forest method. Table 5.1 shows the complexities of the three approaches.

TABLE 5.1: COMPLEXITY OF MACHINE LEARNING ALGORITHMS

Model	Training	Prediction
Naïve Bayes	$O(np)$	$O(p)$
MLP	$O(e * n * (ph_{l1} + h_{l1}h_{l2})$	$O(ph_{l1} + h_{l1}h_{l2} + \cdots)$
Random Forest	$O(n^2pt)$	$O(pt)$
Eth Gas Station	$O(p^2n + n^3)$	$O(p)$

We observe that in the case of Naïve Bayes, which is only employed for classification purposes, the training complexity is entirely dependent on the number of instance and number of parameters passed as input. Whereas, random forest, which is an ensemble

method, simply multiplies the prediction accuracy, of a simple regressor used by Eth Gas Station, by the number of trees or 'voters' employed. While, one interpretation of this could be increased complexity, the presence of multiple voters also ensures stronger model with a chance at better predictive accuracy.

Similar inference can be made in case of MLP. The complexity of MLP depends on the number of layers, number of neurons in each layer as well as the number of training epochs. While the network structure may increase, it also leads to a stronger more accurate predictive model; this is not the case when we consider a liner regressor like the one used by Eth Gas Station.

5.3 Implementation Details for Classification Approach

As discussed in Section 4.2, the data set was distributed into 8 classes making this a multi-class classification problem. Once data was obtained, the dataset was split into 2 disjoint sets. The first set which comprises 80% of the data was used for training and validation. The second set was used as a test data set in order to compare the performance of each model on identical data instead of a randomly split dataset.

In the subsequent sections, we discuss the implantation detail of the three algorithms- Naïve Bayes classifier, Random Forest Classifier and Multilayer Perceptron.

5.3.1 Naïve Bayes Classifier

We know that Naïve Bayes classifier employs Bayes theorem (see Section 4.6.1). In most implementations, the prior is calculated by using an estimate for the class probability from the training set.

$$P(C_k) = \frac{number\ of\ data\ points\ belonging\ to\ C_k}{Total\ number\ of\ data\ points}$$

To model the likelihood term, one can assume a distribution over the data represented by feature vector. In our work, we use Gaussian distribution. While, there are many other functions (e.g., Bernoulli) that could have been employed, Gaussian or Normal distribution is easiest to work with since it can approximate wide range of data by simply calculating mean and variance. Formally [53],

$$P(\boldsymbol{x}|\boldsymbol{C_k}) = \frac{1}{\sqrt{2\pi\sigma_{C_k}^2}} exp\left(-\frac{(x_i - \mu_{C_k})^2}{2\sigma_{C_k}^2}\right)$$

where σ_{C_k} and μ_{C_k} are estimated via maximum likelihood.

5.3.2 Random Forest

Random forest is an ensemble of decision trees. As the number of trees in the forest increases, better overall performance is achieved [54]. However, it can also be computationally expensive to create forests with large number of trees. In our experiment, since the model is expected to keep up with the most recent state of the Ethereum network, the number of transactions that the model will learn at time would be of a relatively small dataset. Therefore, it is possible to seek out more accurate predictions by employing a forest with higher number of trees. Each tree in the forest would classify any given transaction into one of the eight labels (as discussed in Chapter 4 Section 2). Then, the final classification for the transaction would be predicted by taking a mode over the prediction by each tree in the forest. During our experiments, we observed that by increasing the number of trees in the forest, the model accuracy increased. However, after employing a forest with 500 trees, the increase in model's predictive accuracy was insignificant as compared to the training time for the random forest model.

5.3.3 Multilayer Perceptron (MLP)

An MLP with batch normalization [55] and dropout [56] is used for the dataset with a Softmax output layer [57]; Cross entropy [58] was used as the loss function for the network. The model with only one hidden layer was used and the optimization of the neural network was performed using AdaGrad optimization [59]. Furthermore, the learning rate was decayed exponentially with each epoch of training in order to optimally converge the learning process for the neural network.

5.4 Results with Classification Approach

The three models were first evaluated with the static data (as discussed in Section 4.2) and then, they were also evaluated as incrementally re-trained models. This section presents the results observed with the three models when used on both static and incrementally updated real time dataset for our classification approach towards predicting the transaction confirmation time in Ethereum blockchain network.

5.4.1 Initial Performance

During the evaluation process of the three models, a randomly selected data subset of 100,000 transactions was used and the prediction accuracy, null accuracy and kappa score for each model, was calculated for the three re-sampling techniques: Under- Sampling, Over-Sampling and SMOTE.

It should be noted that while the data acquired from the Ethereum API had several attributes, they did not all have the same effect on the prediction accuracy. We observed that the gas price and gas limit have the most effect on the confirmation time for a transaction and that the model accuracy can be increased by a large factor (as per our

experiments) if weighted models are employed. Out of three, only MLP offers weighted nodes for parameters. The other two model were modified as locally weighted Naïve Bayes [60] and weighted random forest [61]. The evaluation results can be observed in Table 5.2, 5.3 and 5.4.

TABLE 5.2: INITIAL EVALUATION RESULTS WITH CLASSIFICATION APPROACH WITH UNDER-SAMPLING

Time to train (in minutes)	Accuracy	Null Accuracy	Kappa Score	Model used
7	57.78%	54.85%	21.54%	Naïve Bayes
13	94.56%	93.23%	87.00%	Random Forest
35	71.35%	68.42%	77.13%	MLP

TABLE 5.3: INITIAL EVALUATION RESULTS WITH CLASSIFICATION APPROACH WITH OVER-SAMPLING

Time to train (in minutes)	Accuracy	Null Accuracy	Kappa Score	Model used
20	57.78%	54.85%	21.54%	Naïve Bayes
38	96.88%	95.67%	88.78%	Random Forest
89	73.35%	72.45%	77.36%	MLP

TABLE 5.4: INITIAL EVALUATION RESULTS WITH CLASSIFICATION APPROACH WITH SMOTE

Training Data	Time to train (in minutes)	Accuracy	Null Accuracy	Kappa Score	Model used
100,000	15	99.54%	98.85%	26.74%	Naïve Bayes
100,000	20	97.39%	97.46%	85.00%	Random Forest
100,000	53	85.27%	55.32%	73.61%	MLP
1,000,000	25	92.08%	90.5%	27.70%	Naïve Bayes
1,000,000	40	96.78%	95.13%	83.42%	Random Forest
1,000,000	130	92.91%	67.39%	78.50%	MLP

The naïve Bayes model with SMOTE resulted in best prediction accuracy; however, the null accuracy for the model was also very similar to the prediction accuracy hence rendering the model inaccurate. The random forest model with SMOTE came out with the best kappa score but also had high null accuracy relative to prediction accuracy; this makes MLP with SMOTE the best but moderately accurate model for prediction of the classes for the testing data. The performance of the three models with under-sampling and over sampling was observed to be subpar at best.

The three models were then executed with the whole dataset providing similar results (See Table 5.4) but with better accuracy and Kappa score implying that these models learn better with more data available to train.

5.4.2 Real-time Dataset

In order to accurately predict the confirmation time for a transaction, the model needs to keep the network congestion in mind. Hence, there is a need for the model to be re-trained incrementally. For our classification approach, this was only implemented in the MLP model; this is because Naïve Bayes classifier offered a weaker model, as discussed above, and Random Forest model is incapable of storing weights, i.e., each time there is new dataset available, the model needs to be re-trained from scratch.

To train the model from scratch at each interval would prove to be inefficient due to high training time. To make it work, after training the network, the weights associated with it are saved and loaded when new data becomes available. This allows the model to be trained from where it was last trained.

According to the data collected from Ethstats.net [62], each block in Ethereum has a gas limit of 7,999,992 and let's assume that each transaction costs 21,000 gas, assuming nothing else is attached to it and that the mining nodes are in fact generating full blocks. This implies that there are ~380 transactions (upper limit) in each block with a block time of ~15.03 seconds. This would result in about 25.346 txn/second. The MLP model discussed in section 5.1.3 was re-trained at 10-minute, 15-minute and 30-minute intervals; Table 5.5 shows the results.

TABLE 5.5: REAL-TIME DATA EVALUATION RESULTS WITH CLASSIFICATION APPROACH

Time Interval (in minutes)	No. of transactions (thousands)	Time to train (in minutes)	Model Accuracy	Kappa Score
10	15.6	~14	83.61%	78%
15	23.4	~15	82.34%	78.12%
30	46.8	~17	82.18%	78.77%

We observed that MLP worked with average accuracy of about 82.7% irrespective of the time interval at which the model was trained with the new dataset. It was also observed that the training time did not vary much with the increased time interval. Therefore, in order to prevent the model from learning a large number of transactions at a time, we decided to re-train the model at a 15 minutes interval.

5.5 Implementation Details for Regression Approach

Before we assess the performance of any given model, especially in case of regression where the output is a set of continuous values and not a set of discrete values, it is necessary to understand and optimize the model structure. Since the continuous valued output expected from a regressor can either be less or more than the known value, it is important to employ absolute error (discussed in Chapter 3 Section 5) as the evaluating factor in order to evaluate the performance accuracy of a regression model. This is because the error metric such as RMSE would severely penalize extreme outlier (or very bad predictions) since it is root of the square of the difference between the estimated and actual target values. However, this is not the case with MAE where the metric is not as sensitive to outlier as MSE.

In the subsequent section, we present the implementation details for the two regression models: Random Forest regressor and Multi-Layer Perceptron regressor. This is because Naïve Bayes, as utilized in case of classification approach does not offer an alternative for the regression approach. Whereas, as discussed in Chapter 4, both MLP and Random Forest, in general, perform well at learning complex, highly non-linear relationships (e.g., relationship between time and both gas price and gas).

5.5.1 Random Forest

As discussed earlier, the performance of a random forest algorithm greatly depends on the number of tress in the forest. After a certain point, using large number of trees only makes the model more complex rather than improving its accuracy. In our experiment, we employed multiple variants of random forest model: with 250 and 500 trees with at least 5 samples at the leaf node, with 1000 trees and at least 10 samples at the leaf nodes, 1500 trees and at least 15 samples as well as one with 2000 trees and at least 20 samples at leaf nodes. Before the data features and the dependent variable (time taken to confirm a transaction) are passed through the random forest regressor function available in scikit-learn [63], we passed also employed a grid search operation available in the library [64] to optimise the model depth (max_depth) for each of the random forest variant to optimise their performance accuracy. Each tree in the forest would take the set of independent variables, as discussed in Chapter 4 Section 4.2, and make discrete valued prediction for the conf_time which is time taken by the mining node to confirm a given transaction. A random forest regressor will then take an average over the discrete valued results calculated by each tree in the forest [42] and generate the final result as the estimated prediction for the transaction.

5.5.2 Multi-Layer Perceptron (MLP)

In case of MLP, we trained multiple variations of MLP with one hidden layer, two hidden layers as well as with three hidden layers to compare the effect of added layer on

training complexity and the prediction accuracy of the model. All variants of MLP employ backpropagation and no activation function (or identity function) at the output layer. This is because in case of regression, we are interested in predicting numerical values and do not require any transformations as was the case with the classification approach. MLP also employs square error as the loss function. Formally,

$$Loss\ (\hat{y}, y, W) = \frac{1}{2}\|\hat{y} - y\|_2^2 + \frac{\alpha}{2}\|W\|_2^2$$

where, $\alpha\|W\|_2^2$ is an L2- regularisation term (or penalty) that penalises the MLP model; and $\alpha > 0$ represents the non-negative hyperparameter that controls the magnitude of penalty. MLP regressor, starting from random weights W, works on the minimizing the aforementioned loss function by running multiple iteration and updating the weights at every step as it backpropagates through the output layer.

5.6 Results with Regression Approach

Similar to the case of classification, we initially split the historical data in two parts where 80% of the transactions were fed to both of our proposed models, i.e., MLP and random forest, and the remaining 20% was used to validate the models after the training phase. This was done to make sure that the model was able to properly predict the block confirmation time for the transaction before real-time data was involved.

5.6.1 Initial Performance

The computational complexity of MLP is dependent on the number of hidden layers used to design the network and that of random forest depends mostly on the number of trees used (see Section 5.1). In order to further understand the complexity, we executed multiple variants of the two machine learning models (MLP and Random Forest). The results of these experiments can be observed in Table 5.6, 5.7 and Figs. 5.1 and 5.2.

TABLE 5.6: TRAINING TIME FOR THE THREE MLP VARIANTS

Model	Time to train (in minutes)
MLP with one hidden layer	130
MLP with two hidden layers	210
MLP with three hidden layers	350

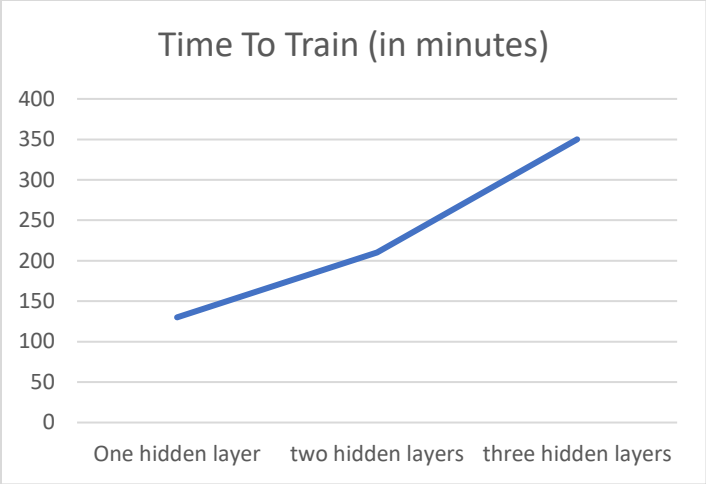

Fig 5.1 Training time for different variants of MLP (in minutes).

TABLE 5.7: TRAINING TIME FOR THE FIVE RANDOM FOREST VARIANTS

Model	Time to train (in minutes)
250 trees/5 samples	10
500 trees/5 samples	22
1000 trees/10 samples	40
1500 trees/15 samples	53
2000 trees/20 samples	71

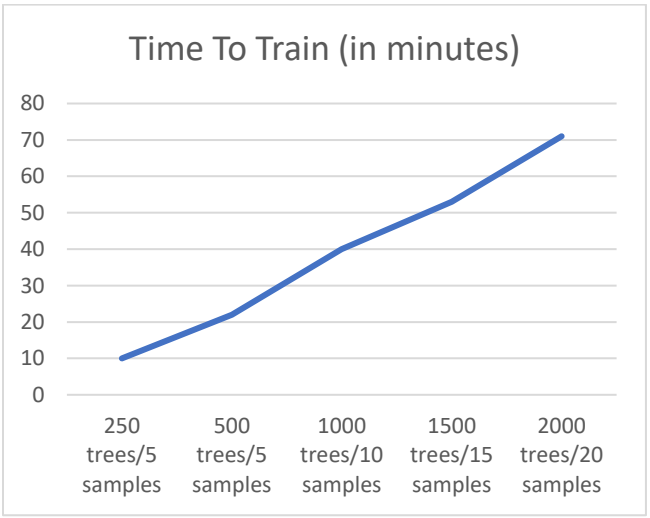

Fig 5.2 Training time for different variants of Random Forest (in minutes).

We also observed that there was a gradual increase in prediction accuracy of the model as we added more layers, in case of MLP and more trees, in case of random forest. However, the increase was so slight that when considering the time taken to train these variations, the increased model accuracy turns out to be a bad trade off. The performance of the two models and the variants can be observed in Table 5.8, 5.9 and Fig 5.3 and 5.4.

TABLE 5.8: RESULT ANALYSIS FOR STATIC DATASET WITH MLP WITH REGRESSION APPROACH

Model	MAE	RMSE	MAPE	Pred (0.20)
One hidden layer	0.167	0.408	7.40%	87.51%
Two hidden layers	0.153	0.391	6.89%	88.02%
Three hidden layers	0.1493	0.386	6.57%	88.44%

TABLE 5.9: RESULT ANALYSIS FOR STATIC DATASET WITH RANDOM FOREST WITH REGRESSION APPROACH

Model	MAE	RMSE	MAPE	Pred (0.20)
250 trees/5 samples	0.118	0.343	5.07%	89.54%
500 trees/5 samples	0.112	0.334	4.83%	93.79%
1000 trees/10 samples	0.109	0.33	4.81%	93.91%
1500 trees/15 samples	0.1087	0.329	4.809%	93.92%
2000 trees/20 samples	0.1087	0.329	4.809%	93.92%

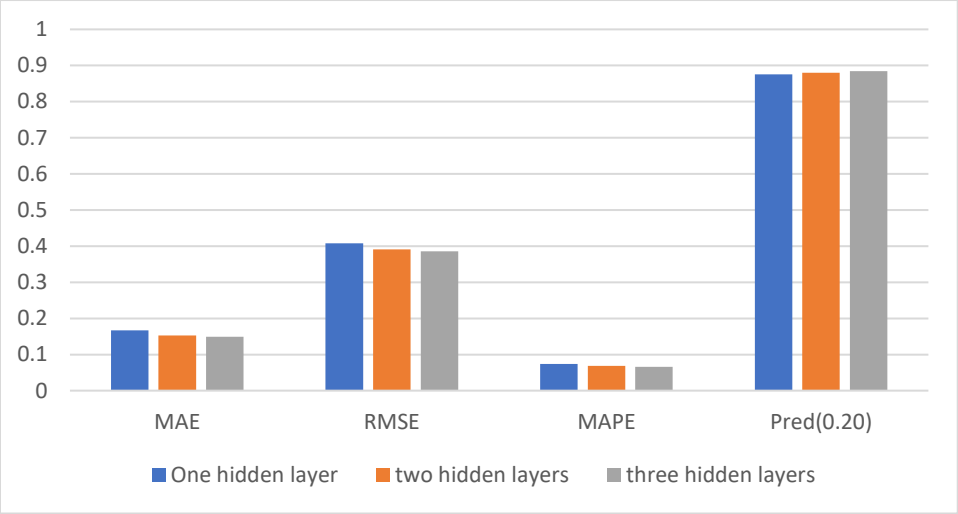

Fig 5.3 Performance comparison between variations of MLP, with static data and under the set evaluation criteria.

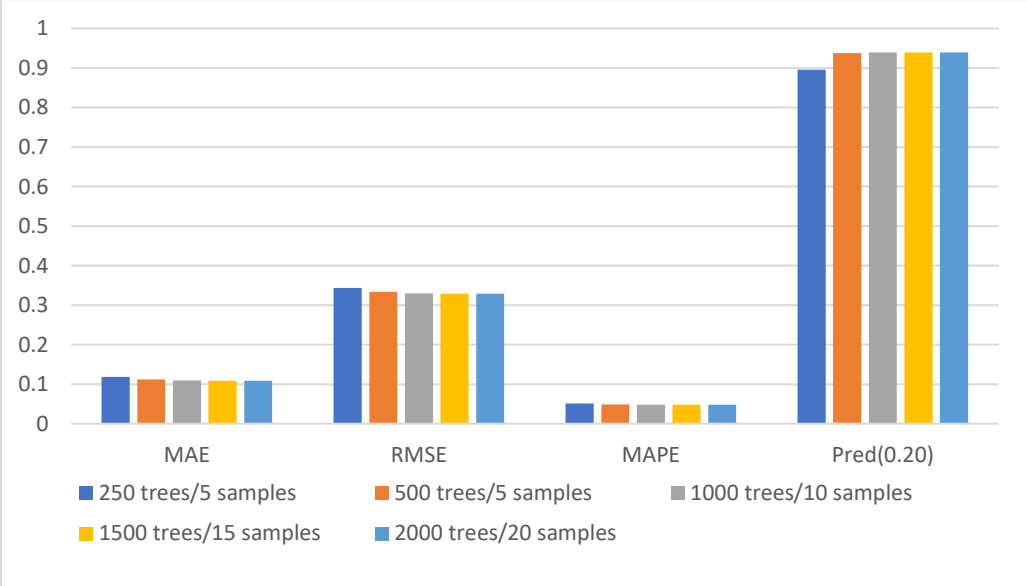

Fig 5.4 Performance comparison between variations of, Random Forest with static data and under the set evaluation criteria.

5.6.2 Real-time Dataset

For real time data, the local geth node that was being monitored (see Section III) has been recording details of the transactions as they are made. These details include the timestamp for when the transaction was made, the number of blocks that were created before it was confirmed, the gas price as well as the gas limit set by the user. Whenever there is a set of new transactions for which the model is required to make confirmation time predictions, it would do so based on the data collected by geth node for the most recent confirmed transactions (from the past 100 blocks). These are then fed to the model with the main strategy of predicting the confirmation time for a transaction given a certain gas price between 0-100 gwei at the current state of the transaction pool.

The two machine learning regression techniques proposed were compared with Poisson Regression Statistical model used by Eth Gas Station [24]. Since there is no literature available confirming the accuracy of this statistical model; to measure the prediction accuracy of Eth Gas Station, a randomly generated subset of the data gathered from the local geth node was passed through a regression model similar to the one used by Eth Gas Station as available on their GitHub page [65]. Eth Gas Station performs their predictions in terms of number of blocks a transaction had to wait before it was confirmed by a mining node instead of number of seconds as done in this work [66]. It should be noted that, on average, in Ethereum, a new block is created every 15 seconds. This implies that if a transaction had to wait 29 seconds before it was confirmed by a mining node then it waited for two block confirmations. This interchangeability between the two allowed us to compare the performance of our machine learning algorithms against Eth Gas Stations Poisson regression model.

According to the data collected from BitInfoCharts [67], on average, each block has 138 transactions in it. Considering that our models consider data from 100 most recent block to predict the confirmation time for transaction(s) in question, the models will have to learn about 13.8 thousand transactions. Table 5.10 illustrates the time taken by each model to learn these transactions.

TABLE 5.10: TRAINING TIME FOR THE THREE MODELS TO LEARN 100 BLOCKS

Model	Time to train (in minutes)
Eth Gas Station	2.91
MLP	7.31
Random Forest	2.78

Taking into consideration the complexity of each variant as observed in previous section, comparison of statistical approach taken by Eth Gas Station was done against MLP with

one hidden layer and random forest with 500 trees and at least 5 samples at leaf nodes. It should be noted that for Eth Gas Station, we implemented a Poisson regression model to calculate the time taken by the model to learn 13.8k transactions.

Since users are allowed to make transactions at any given time, our model should remain up to date with the most recent Ethereum blockchain network state. Given the time taken by each of the three models as seen in Table 5.10. we trained the random forest model at an interval of 3 minutes whereas the MLP was trained at an interval of 8 minutes. Table 5.11 and Fig 5.5 shows the results for the two models when compared with Eth Gas Station.

TABLE 5.11: RESULT ANALYSIS FOR THREE REGRESSION MODELS

Model	MAE	RMSE	MAPE	Pred (0.20)
MLP	0.21	0.424	10.54%	82.74%
Random Forest	0.13	0.36	5.70%	89.36%
Eth Gas Station	0.174	0.417	9.31%	84.79%

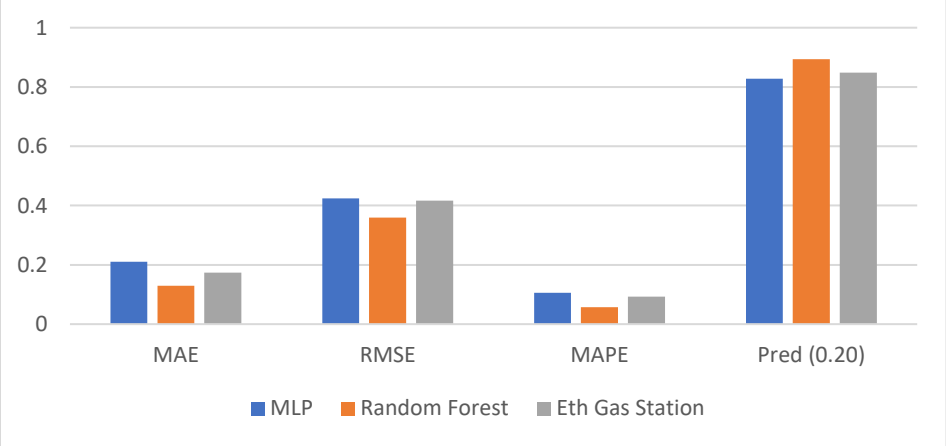

Fig 5.5 Performance comparison between MLP, Random Forest and Eth Gas Station under the set evaluation criteria.

The Multi-Layer Perceptron produces a mean absolute error of about 0.21 with the root mean square error of 0.424 and pred(n) of 82.74% at n being 0.20. While analysing hyperparameters, it was observed that RMSE can be decreased at the cost of prediction accuracy, resulting in a weak or inaccurate model. The mean absolute percentage error observed with MLP was about 10.54%. Similarly, with random forest, MAE was observed

to be good at 0.13 with RMSE at 0.36, MAPE at 5.7% and pred (0.20) of a solid 89.36%. When evaluating the statistical Eth Gas Station, the mean absolute error of 0.174 was observed with mean absolute percentage error at 0.31%, RMSE at 0.417 and a prediction accuracy of 84.79%. We conclude, from above discussion, that the random forest model performs better than the other two models in all the evaluation criterions employed.

5.7 Discussion

In terms of classification approach, prediction accuracy and Cohen's Kappa score are one of the most important performance measurement metrics. We observed that random forest and MLP both perform well under the two metrics. However, it is not possible to store the weights in random forest therefore the model does not perform well with real-time data which is the most important aspect of our problem statement. MLP, on the other hand, presents a model with relatively good accuracy and moderate strength in terms of Cohen's kappa score. The MLP model work with an average accuracy of about 82.5%. We know that on average, a transaction has to wait for two block confirmations (~30 seconds) before it is confirmed. However, in cases where the model would predict that the transaction belongs to 'within 5 minutes' class, there is no way for the user to know if it would take 3 minutes, 4 minutes or more. Hence, while the model performs with good prediction accuracy, it can only provide a user with an approximation of time it would take for their transaction to be confirmed which may or may not always be ideal. Due to the large amount of capital involved and the sensitivity of data that is to be transferred via a transaction through the Ethereum blockchain network, the user may want more precise predictions in terms of transaction confirmation. This is where regression approach comes in.

Unlike the classification approach, the regression approach to our problem results in real valued predictions for the confirmation time. The principal metric for evaluation of a regression model is Root Mean Square Error. We observed that out of the two machine learning algorithms we proposed, i.e., random forest regressor and MLP regressor and the previously employed Poisson regression model, random forest perform the best with an RMSE score of about 0.36 and a prediction accuracy (PRED (0.20)) of 89.36%. This is because of random forest model's inherent capability of handling variations and noises in the dataset all the while keeping the model unbiased and stable.

As discussed in the previous chapter, Random Forest is an ensemble method and reduces the error by reducing the variance. There are two features contributing in the reduction of variance: averaging and random sampling. Each tree in the Random Forest has a very high variance but averaging reduces the variance as long as the trees are not co-related. Since training many trees on the same dataset generates strongly co-related trees, different subsets of the dataset are fed to decision trees. By adding randomness to the sampling process, trees are trained even more independently, which in case of large number of trees, the gain for averaging can be more dramatic. However, we also observed that there is a certain point at which adding more trees to the forest does not significantly improve the model performance but indeed increases the computational complexity of the model.

We also observed that MLP underperforms in comparison to the other two techniques. While MLP is capable of learning any non-linear relationship among data features, its performance is greatly dependent on the number of epochs and iterations used. Due to the need for the model to be retrained periodically and the time taken by MLP to learn new data, it is not the most viable model for confirmation time prediction. On the other hand, random forest needs to be trained from scratch each time we want the model to learn new data. However, it is observed that it learns new data much faster with lesser complexity as compared to MLP.

Despite the limitations of MLP and random forest in terms of complexity and training time, machine learning does provide a novel approach when it comes to these confirmation time predictions and can, as observed, outperform statistical methods.

6. Conclusion

The task of predicting the confirmation time for a transaction in Ethereum network is nontrivial and a necessity considering the amount of capital involved. In this thesis, we did consider two approaches towards the problem at hand. A simpler classification approach, where we categorized a transaction into eight classes based on the transaction confirmation time, and a more complex but precise regression approach. We systematically compare the performance of three machine learning classification algorithms and tackle the skewed nature of the class distribution. We also compare the two machine learning regression algorithms to the previously employed statistical technique and draw insight on which model works the best for our prediction problem.

To summarize out contribution, we found that classification, while it provides relatively accurate predictions, is not the best approach to the confirmation time prediction problem since more precise prediction may be more viable considering the sensitivity of data and the capital involved in Ethereum transactions. We also found that out of machine learning and statistical regression techniques, Random Forest algorithm outperforms all other techniques presented in this thesis. We observed that while there is an increase in the prediction accuracy of the two algorithms, i.e., Random Forest and Multilayer Perceptron Regressors, under the more complex variants, there is only a slight increase in the accuracy when these complex variants are compared with their simpler counterparts. Thanks to the relatively fast learning observed with random forest, it outperforms the other two algorithms with real-time data as well.

One of the future works might be to explore other models like deep neural networks, which like MLP, can easily update their knowledge base with new data. This in in addition to exploring these and other neural structures to determine whether changes made to the hyper-parameters (e.g., the learning rate or the size of hidden layer(s)) can decrease relative error magnitude without affecting model accuracy, i.e., presenting stronger regression or prediction model which would be more suited to the ever-changing Ethereum network. Another question that remains to be answered would be to understand and explore how the knowledge of time prediction can help user manipulate the network into making transactions more economic or faster in the future. We also intent on publishing a web application based on the model proposed in this thesis which would be available for usage

for any Ethereum user to make confirmation time predictions before they make a transaction to the network.

References

[1] Blockchains: The great chain of being sure about things, The Economist, http://www.economist.com/briefing/2015/10/31/the-great-chain-of-being-sure-about-things, last accessed 2019/04/12.

[2] Wood, G., Ethereum: A Secure Decentralised Generalised Transaction Ledger, https://gavwood.com/paper.pdf, last accessed 2019/04/12.

[3] What is a node in a Blockchain Network? Nodes.com, https://nodes.com/, last accessed 2019/06/04.

[4] Zcash, How it works, https://z.cash/technology/, last accessed 2019/06/04.

[5] CoinMarketCap, https://coinmarketcap.com/, last accessed 2019/04/10.

[6] Greenberg, A., Crypto Currency https://www.forbes.com/forbes/2011/0509/technology-psilocybin-bitcoins-gavin-andresen-crypto-currency.html#7c44500c353e, last accessed 2019/04/10.

[7] Payette, J., Schwager, S., and Murphy, J., Characterizing the Ethereum Address Space, http://cs229.stanford.edu/proj2017/final-reports/5244232.pdf, last accessed 2019/04/10.

[8] Smart Contracts, https://en.wikipedia.org/wiki/Smart_contract, last accessed 2019/04/12.

[9] Jagers, C. What is Ethereum, https://www.investopedia.com/articles/investing/022516/what-ethereum.asp, last accessed 2019/04/10.

[10] Bajpai, P., Bitcoin Vs Ethereum: Driven by Different Purposes, https://www.investopedia.com/articles/investing/031416/bitcoin-vs-ethereum-driven-different-purposes.asp, last accessed 2019/04/10.

[11] Singh, H. J., Hafid, A. S., Prediction of Transaction Confirmation Time in Ethereum Blockchain using Machine Learning, Blockchain and Applications, Springer, DOI 978-3-030-23813-1_16, 2020, pp. 126-133.

[12] Genesis Block, https://en.bitcoin.it/wiki/Genesis_block 2019/04/12.

[13] Saraf, C. and Sabadra, S., Blockchain platforms: A compendium, IEEE International Conference on Innovative Research and Development (ICIRD) 2018, Bangkok,2018, pp. 1-6.

[14] Merkle Tree, https://en.wikipedia.org/wiki/Merkle_tree, last accessed 2019/04/12.

[15] Sompolinsky, Y. and Zohar, A., Accelerating Bitcoin's Transaction Processing. Fast Money Grows on Trees, Not Chains, Cryptology ePrint Archive, Report 2013/881, http://eprint.iacr.org/, 2013, last accessed 2019/04/15.

[16] Murphy, K. P., Machine learning: a probabilistic perspective, MIT press, 2012.

[17] Sammut, C. and Webb G.I., Encyclopedia of machine learning, Springer Science & Business Media, 2011.

[18] Geurts, P., Contributions to decision tree induction: bias/variance tradeoff and time series classification, Ph. D. thesis, University of Liege Belgium, 2002.

[19] Wasserman, L., All of statistics: a concise course in statistical inference, Springer Science & Business Media, 2013.

[20] Moore, D. S. and McCabe, G. P., Introduction to the Practice of Statistics, WH Freeman/Times Books/Henry Holt & Co, 1989.

[21] Bishop, C. M., Pattern recognition, Machine Learning 128, 2006.

[22] Ganganwar, V., An overview of classification algorithms for imbalanced datasets, International Journal of Emerging Technology and Advanced Engineering 2 (4), 2012, pp. 42–47.

[23] Landis, J. R. and Koch, G. G. "The measurement of observer agreement for categorical data" in Biometrics. Vol. 33, 1977, pp. 159–174.

[24] Eth Gas Station, https://ethgasstation.info, last accessed 2019/04/15.

[25] Ethereum Transaction Information, https://etherscan.io/, last accessed 2019/04/15.

[26] Ethereum Pending Transactions, https://etherscan.io/txsPending, last accessed 2019/04/15.

[27] Ethereum StackExchange, What's the difference between geth/mist/parity?, https://ethereum.stackexchange.com/questions/62653/can-anyone-explain-whats-the-difference-between-mist-geth-parity-in-simple-term, last accessed 2019/07/24.

[28] Ethereum Pending Transaction Queue, https://etherscan.io/chart/pendingtx, last accessed 2019/04/20.

[29] Ethereum, Block Size History, https://etherscan.io/chart/blocksize, last accessed 2019/04/20.

[30] Provost, F., Machine learning from imbalanced data sets 101, In Proceedings of the AAAI'2000 workshop on imbalanced data sets, 2000, pp. 1–3.

[31] Drummond, C. and Holte, R. C., Exploiting the cost (in) sensitivity of decision tree splitting criteria, In ICML, 2000, pp. 239–246.

[32] Opitz, D. and Maclin, R., Popular ensemble methods: An empirical study, Journal of Artificial Intelligence Research 11, 1999, pp. 169–198.

[33] Breiman, L., Bagging predictors, Machine learning 24 (2), 1996, pp. 123–140.

[34] Freund, Y. and Schapire, R. E., A desicion-theoretic generalization of on-line learning and an application to boosting, In European conference on computational learning theory, 1995, pp. 23–37, Springer.

[35] Chawla, N. V., Bowyer, K. W., Hall, L. O. and Kegelmeyer, W. P., Smote: synthetic minority over-sampling technique, Journal of artificial intelligence research 16, 2002, pp. 321–357.

[36] Vapnik, V. N., An overview of statistical learning theory, IEEE transactions on neural networks 10 (5), 1999, pp. 988–999.

[37] Friedman, J., Hastie, T. and Tibshirani, R., The elements of statistical learning, Volume 1, Springer series in statistics Springer, 2001, Berlin.

[38] Schum, D. A., The evidential foundations of probabilistic reasoning, Northwestern University Press, 1994.

[39] Duda, R. O., Hart, P. E., Pattern classification and scene analysis, Wiley New York. Volume 3, 1973.

[40] Decision Tree-Regression, https://www.saedsayad.com/decision_tree_reg.htm, last accessed 2019/04/25.

[41] Mitchell, T. M., Machine learning, McGraw-Hill, 1997.

[42] Breiman, L., Random forests, Machine learning 45 (1), 2001, pp. 5–32.

[43] Seif, G. Selecting the best Machine Learning algorithm for your regression problem, https://towardsdatascience.com/selecting-the-best-machine-learning-algorithm-for-your-regression-problem-20c330bad4ef, last accessed 04/27/2019.

[44] Grosee, R., Intro to Neural Networks and Machine Learning, Lecture 5: Multilayer Perceptron, https://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/readings/L05%20Multilayer%20Perceptrons.pdf, last accessed 2019/09/21.

[45] A Beginner's Guide to Multilayer Perceptrons (MLP), Skymind, https://skymind.ai/wiki/multilayer-perceptron, last accessed 2019/07/24.

[46] Arouri, C., Nguifo, E. M., Aridhi, S., Tsopze, N., Towards a constructive multilayer perceptron for regression task using non-parametric clustering. A case study of Photo-Z redshift reconstruction, https://arxiv.org/abs/1412.5513, last accessed 2019/04/14.

[47] Van Der Walt, S., Colbert, S. C., and Varoquaux, G., The numpy array: a structure for efficient numerical computation, Computing in Science & Engineering, vol. 13, no. 2, 2011, pp. 22-30.

[48] Jones, E., Oliphant, T. and Peterson, P., SciPy: Open source scientific tools for python, http://www.scipy.org/, 2001.

[49] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E., Scikit-learn: Machine learning in Python, Journal of Machine Learning Research, 2011, pp. 2825-2830.

[50] Lemaitre, G., Nogueira, F. and Aridas, C. K., Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. CoRR abs/1609.06570.

[51] Hall, L., Computational Complexity, John Hopkins University, http://www.esi2.us.es/~mbilbao/complexi.htm, last accessed 2019/04/25.

[52] Computational Complexity of Machine Learning Algorithms, The Kernel Trip, https://www.thekerneltrip.com/machine/learning/computational-complexity-learning-algorithms/, last accessed 2019/04/25.

[53] McCallum, A., Nigam, K., A comparison of event models for naive Bayes text classification, In AAAI-98 workshop on learning for text categorization, Volume 752, 1998, pp. 41–48, Citeseer.

[54] Oshiro, T. M., Perez, P. S. and Baranauskas, J. A., How many trees in a random forest? In International Workshop on Machine Learning and Data Mining in Pattern Recognition, 2012, pp. 154–168, Springer.

[55] Ioffe, S., and Szegedy, C., Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift: Cornell University Library, arXiv:1502.03167, March 2015.

[56] Srivastava, N., Hinton G., Krizhevsky A., Sutskever, I., and Salakhutdinov, R., Dropout: A Simple Way to Prevent Neural Networks from Overfitting: Journal of Machine Learning Research: June 2014.

[57] Multi-Class Neural Network: Softmax, Machine Learning, https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax, last accessed 2019/07/25.

[58] Loss Functions, ML Cheatsheet, https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html, last accessed 2019/07/25.

[59] https://www.tensorflow.org/api_docs/python/tf/train/AdagradOptimizer, last accessed 2019/02/05.

[60] Frank, E., Hall, M. and Pfahringer, B., Locally Weighted Naïve Bayes, https://www.cs.waikato.ac.nz/~eibe/pubs/UAI_200.pdf last accessed 2019/02/05.

[61] Winham, S., Freimuth, R. and Biernacka, J., A Weighted Random Forest Approach to Improve Predictive Performance, Statistical analysis and data mining vol. 6,6, pp. 496-505 (2013).

[62] Ethereum Network Status, https://ethstats.net/, last accessed, 2019/05/12.

[63] Random Forest Regression, Scikit-learn, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html, last accessed 2019/04/15.

[64] GridSearchCV, Scikit-learn, https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html, last accessed 2019/04/15.

[65] EthGasStation, https://github.com/ethgasstation, last accessed 2019/04/15.

[66] FAQ, EthGasStation, https://ethgasstation.info/FAQcalc.php, last accessed 2019/09/21.

[67] BitInfoCharts, Ethereum (ETH) price stats and information, https://bitinfocharts.com/ethereum/, last accessed 2019/05/25.