Imperial College
London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Identifying users on the Bitcoin network using machine learning and graph visualisations

Author:
Aditya Chaturvedi

Supervisor:
Dr. David Birch

Submitted in partial fulfillment of the requirements for the MSc degree in MSc Computing (Machine Learning) of Imperial College London

September 2018

Abstract

The anonymous yet public nature of the Bitcoin blockchain has left many wondering about the inner activities occurring on the network. Breakthroughs in identifying these activities will enable quantitative insights into this massive public database, and help curb the prevailing illicit activities on the system. If this is achieved successfully, public confidence in this currently volatile technology is likely to increase, paving the way for a more stable and widely adopted technology that can change the way the world thinks of money [30]. As a continued step towards this goal, this project builds a framework that takes a combined a top-down and bottom-up approach to analysing the network.

Presented in this study is a novel system that combines machine learning and graph based analysis to extract useful information on real world actors. Given unlabelled address and transaction data, the system is able to cluster addresses belonging to individual users by implementing a refined pair of ownership heuristics. Using these clusters (identified as 'users'), self defined features are generated and combined with ground truth labels scraped from off-chain sources. By training machine learning models with this feature set, commercial Bitcoin services, such as mining pools, exchanges, gambling sites and tumbling services are identified with high average Recall values of up to 80%. Given enough training data, this system can be utilised to identify services that haven't been tagged, thereby giving a more encompassing view of network activity. The changing behaviour of services on the network over time is also explored.

Finally, new supplementary top-down visualisations are showcased, adding to previous work done in this space. This study presents the clustering of the address to the user graph in a visual manner. The graphs can be generated for any block height and help derive interesting visual insights that would have otherwise been hidden away. Previous attacks on the network are also explained in the context of these visualisations.

Acknowledgments

I would like to thank Dr. David Birch for his continued encouragement and guidance throughout this project, Mr. Dan McGinn for his insightful conversations and advice on the neo4j database and Prof. William Knottenbelt for his invaluable feedback and suggestions.

I would also like to thank my colleagues Jack Collins and Manish Sharma for their insightful discussions on machine learning and blockchain technology throughout this amazing year.

None of this work would be possible without the constant support and love of my family and friends, for whom I am truly grateful.

Contents

Chapter 1

Introduction

The recent widespread evolution and adoption of machine learning methods has allowed for unprecedented levels of insight into data. Machine learning is the practice of using certain algorithms to parse and learn from data in order to make a strong prediction about some underlying function describing something about the world. Alongside this, blockchain technology has taken the world by storm with its promise of systems that are, theoretically, cheat-proof, immutable and free from the influence of external bodies due to their anonymous and decentralised nature. Bitcoin is the most used and recognisable form of such a system implemented currently. This project aims use graph visualisations to take a novel look at activity on the Bitcoin blockchain, as well apply machine learning methods in order to make predictions on the type of users that may be present.

1.1 Motivation

The motivation to undertake this study initially came about from a desire and curiosity to learn more about blockchain and the hype around it. As more was learned about its pseudo-anonymous nature and inner workings, there was a clear opportunity to try and apply traditional machine learning methods to attain greater insights about the data. There was a perfect mix between the two principles as one produces publicly available data in vast quantities by the minute, and the other, being a data hungry science, allows meaningful analysis.

Due to the pseudo-anonymous nature of the Bitcoin blockchain, there is an incentive for bad actors to abuse its system. Previous works carried out by the Imperial College London Data Science Institute (DSI) have been able to visually show the presence of algorithmic behaviour, possible laundering chains and distributed denial-of-service (DDoS) attacks on the system [26]. This study looks to supplement these existing DSIs transaction graphs with two additional graphs. One which describes interacting addresses within a block, and another which condenses this address graph into a much simpler, more interpretable ‘user’ graph. Furthermore, the current vi-

sualisations are without any additional information. Appending quantitative and behavioural data to these visualisations, such as attaching labels of known services to nodes, enables a fresh view into activity on the system.

The need for graph visualisations becomes apparent due to the connective nature of Bitcoin transactions and their in and out addresses. The system can be naturally represented as a graph, as has been successfully demonstrated by the DSI for Bitcoin transactions in real-time as well as at a block-level [26]. The representation of the dense and indecipherable binary Bitcoin data as graphs recovers great value from this publicly available dataset, providing insights to the general public as well as people of domain knowledge. It is also useful for vanilla machine learning methods as shown later, with the graph data being used to extract learn-able features per user.

1.2 Objectives

The aims of this project can be summarised as follows:

1. Create an address graph to accompany the current transaction graphs from previous projects, using similar graph layout algorithms.
2. Develop a system to condense this address graph, which is complex and bipartite, into a more intuitive and insightful 'user' graph.
3. Tag these graphs with labelled data for better visual indications of user activity per block.
4. Extract features from the user graph in order to learn, using machine learning, how certain commercial services operate and interact across blocks.
5. Present findings in a visually pleasing and aesthetic manner using the KPMG Global Data Observatory (GDO).

1.3 Challenges

A classic problem in applying machine learning to Bitcoin is the dearth of ground truth data. Even if a few labels exist, there is a heavy imbalance in the ratio between what is labelled and what is unknown. Websites such as blockchain.info have a tool which allows for manually labelling addresses but due to the sheer size of the data this proves to be an arduous task. Additionally, the labels can be expected to be noisy and the quality is highly dependant on the annotator. The amount of research done in this space is also recently emergent. Previous works by Pham et al. [29] and Hirshman et al. [20] have experimented with using unsupervised learning methods to cluster transactions but their results have been limited.

This project involves the use of many different technologies and a major task was getting these technologies to work together as a system. The data collection was done using a neo4j database. Results from this were fed into a python framework which carried out data manipulation, statistical analysis and machine learning. The creation and manipulation of graphs were done using python-igraph and Gephi framework.

1.4 Contributions

The specific contributions of this dissertation are described below:

1.4.1 Address-to-User Clustering Tools

This project produces a novel system that successfully deploys clustering algorithms over previously mined blocks in the Bitcoin network. As such, it can successfully cluster and identify potential users given a set of addresses along with the transactions they were involved in. The algorithm is able to function across a continuous slice of multiple blocks and also accommodates for asynchronous blocks. Another useful feature is that, given an address owned by a user, it can trace other potential addresses belonging to that user.

1.4.2 Annotated Address and User Graphs

Novel annotated address and user graphs were devised to supplement previous work visualising Bitcoin transactions by McGinn et al. [26]. The address graph describe relationships between addresses that have interacted via transactions, and the user graph clusters and condenses the address graph into a much simpler, more interpretable view of the network. These annotated graphs allow for new insights into activity on the network, by enabling visual observations of real world entities on the blockchain.

1.4.3 Machine Learning Analysis

A system is developed to extract spending, receiving, mining and behavioural features from the user graph. By using classic machine learning models on this self-generated dataset of features, average recall up to 80% was returned when trying to classify different kinds of active commercial services on the system.

Chapter 2

Background and Related Work

This chapter describes the background theory and research that was needed to be understood for this project. In the first section, a recap of Bitcoin fundamentals is given, while also looking at some of the services that are available for Bitcoin. The second part briefly looks at graph theory and the Force-Atlas 2 layout algorithm. Next, a description of basic machine learning theory is given, along with an explanation of the pros and cons of the different algorithms chosen and a justification for their usage. Finally, the chapter concludes detailing previous work that has been done in this space, with clear referencing of the works built upon for this study.

2.1 Bitcoin

2.1.1 Blockchain and Cryptocurrencies

According to Google Trends, noteworthy internet searches for the term 'blockchain' first appeared around November 2013 [5]. Of course, searches reached an all time high during December 2017, when media coverage was plentiful and cryptocurrency prices soared. Simultaneously, searches for the term 'cryptocurrency' and 'bitcoin' were also relatively low up until the latter part of 2017 where they saw a similar widespread increase. These technologies go hand in hand, however the common public belief that they are one and the same is incorrect. Blockchain is the technology that underpins digital currencies, along with many other potential use cases from the automotive and IOT sectors to cloud storage.

The blockchain is simply a large distributed ledger that keeps a record of all verified transactions that occur on it. It is called the "blockchain" because transactions are not added to the ledger individually. Instead, they are bunched together and added in 'blocks'. This database is distributed peer-to-peer and continually altered by participating nodes. As identical information is stored in its entirety across multiple nodes in the network, the blockchain is very robust to failure because it is not controlled by a single user and has no single point of failure.

Cryptocurrencies are a use case of blockchain technology. They utilise a public and distributed version of the system to regulate the generation of currency units and also verify all fund transfers. Due to the lack of a centralised authority, both privacy (no authority has any knowledge of whom, for what and why the transaction is occurring) and portability (transfer of funds to anywhere in the world) are possible. The most recognisable cryptocurrency is Bitcoin.

Blockchain technology was the brainchild of the person or group of people known as Satoshi Nakomoto. They described a peer-to-peer decentralised currency named Bitcoin in an anonymous white paper released in 2008 [28]. The Bitcoin protocol is based of the ”proof-of-work” algorithm, which requires ”miners” to ”mine” blocks that are difficult to create but easy to validate. The network is run peer-to-peer with new nodes being added as users run their own local copy of the complete blockchain database. The key idea behind Bitcoin is to replace the tiered architecture of trust and central authority with its decentralised, public ledger transactions. This eliminates central authority and the need for parties to trust each other. All transactions made on the network are public, but the participants are (pseudo)anonymous.

2.1.2 Bitcoin Mining

Mining is the only process through which new coins are minted and injected into the network. Mining involves bundling transactions to a block and appending it to the chain of blocks. The bitcoin protocol is designed to add a new block in this way approximately every ten minutes. Mining also verifies transactions on the system. To create value out of Bitcoin, the mining process needs to be non-trivial, otherwise there would be no incentive for competition and no trust in the chain. This is achieved by hashing the header of the block with a random number (nonce) using the SHA256 cryptographic hash algorithm. The block is mined if this hash value has a certain number of leading zeroes (determined by the difficulty level). As a reward for their efforts, the miner is rewarded with newly issued bitcoin, as well as any fees they charge for transactions. This supply is controlled by the reward era, the value of which is halved every few years. The maximum supply of total bitcoin is limited to roughly 21 million bitcoins.

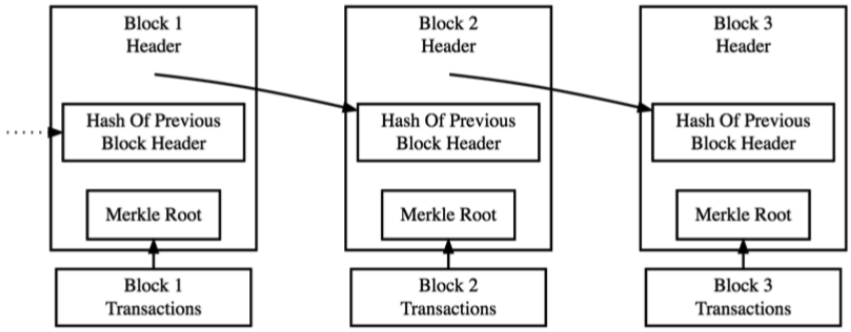


Figure 2.1: Simplified Bitcoin Blockchain architecture [33]

A block is essentially a data structure, with the bitcoin block size being limited to 1MB, hence limiting the number of transactions it can store. The block consists of the header which contains the previous block hash, the nonce (constantly changed by miners during mining), merkle root (hash of the transactions in the block) and timestamp. This chain of blocks goes all the way back to the original hard-coded block called the ”genesis block”. Blocks are only joined to the longest chain already existing in the network (as this is the ledger agreed upon by the majority of nodes). This consensus mechanism makes the network resilient to the double spending problem where malicious users could propagate two transactions in separate locations but use the same bitcoin, and also prevent a malicious node from altering historical transactions.

2.1.3 Bitcoin Transactions

Transactions are created, validated and verified on the bitcoin network [11]. The first transaction in a block is always the coinbase transaction, which is created by the miner. Nodes that receive transactions verify that they are consistent and well formed and only then are these validated transactions propagated to the network. Valid transactions are stored in each nodes mempool and mining nodes proceed to mine their mempool into a block. Transactions that are recorded in a block become verified. Any transactions that are valid but not mined are added to a queue to be mined in later blocks. A key concept to understand is the UTXO (unspent transaction outputs). This is the amount of bitcoins locked to an owner, recorded on the blockchain and recognised by the network. Hence, a bitcoin address contains no explicit balance but just UTXOs that are linked to that address. For a transaction, input UTXOs are consumed *entirely* and any remainder after the transaction is taken as change or fees.

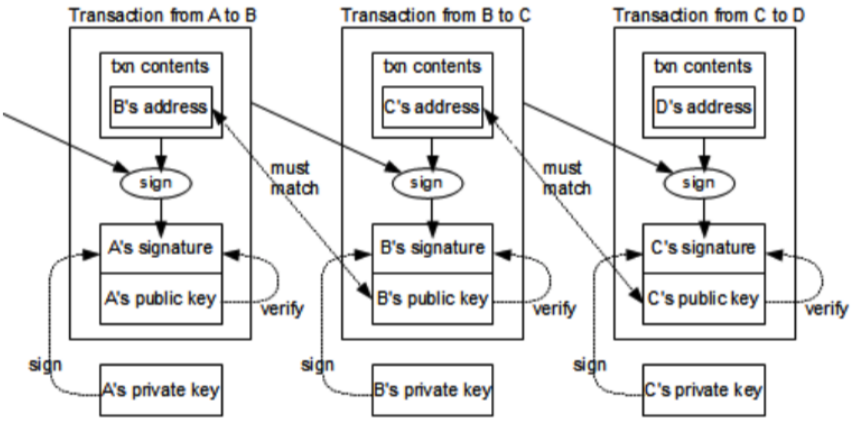


Figure 2.2: Transaction chain between inputs and outputs [33]

Inputs into a transaction can either be the coinbase transaction or the output of a previous transaction. The output is bitcoin to a specified address and is only spent

if the unlocking script for that address is provided. When spent, the value is exchanged from the previous owner/address to the next. Due to the public nature of the blockchain, all sources of funds of bitcoin can be traced, and all payments between address can be seen. Therefore, if an address can be linked to an owner or an algorithmic actor, it will be possible to see all the transactions their address makes.

Most common types of Transactions:

- One input (source of funds), two outputs (payment and change)
- Many inputs, one output (assembly of wealth)
- One input, many outputs (dispersion of wealth)
- Many inputs, many outputs

2.1.4 Bitcoin Services

There are many different businesses that have emerged from the bitcoin platform. In this study, four such services are sought to be identified. The following is a brief description on each of these:

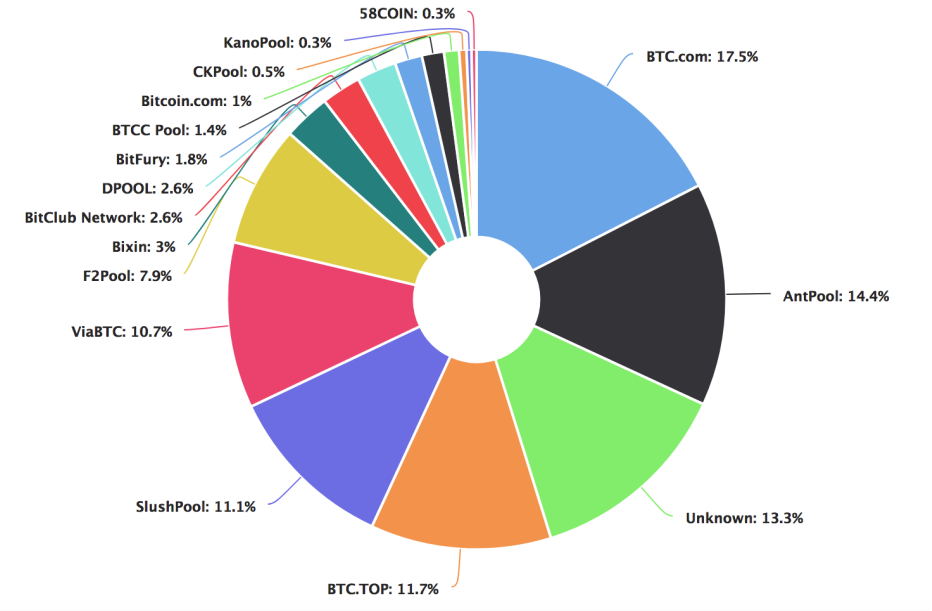


Figure 2.3: Market share of the most popular Bitcoin mining pools [7]

- **Mining Pools**
  These are groups of miners who want a share of the profits from mining a block. They agree to share this reward in proportion to the respective mining hash power that each of them contributed. Mining pools are collaborative efforts by people who have the hardware power necessary for mining. Early on in Bitcoins history, mining was possible on a home computer. However, due

to exponential increase in competition and complexity, mining is now largely done in specialised warehouses with massive amounts of mining hardware and power consumption. These warehouses usually direct their hashing power towards mining pools. The majority of these warehouses are currently in China [35].

- **Gambling Service**
  A bitcoin gambling website operates like any other gambling service with the difference being that they accept payments in bitcoin, as well as other cryptocurrencies. Common gambling games includes dice, casino, blackjack and roulette.

- **Exchanges**
  A bitcoin exchange is a digital platform where traders can purchase and sell bitcoins using different fiat currencies or other cryptocurrencies. An exchange looks to match buyers with sellers and acts as their intermediary. There are transaction fees applied to each buy and sell order executed. The fee taken is based on the amount of transactions that are executed. A well known exchange was Mt. Gox, which had monopolistic control over the bitcoin-dollar trading volume (estimated 70%) [22]. After it's security breach and collapse, many exchanges claim to improve on Mt. Gox with better security features. Exchanges are not to be confused with wallets. A wallet is a digitally secure service for storing bitcoin. Wallets store private keys which are used to authorise transactions from the address of the user. Most exchanges provide wallets for their customers, but this may incur additional fees.

- **Mixers**
  Bitcoin mixing or tumbling is a paid service which looks to obscure the connection between sending and receiving addresses that are transacting. Mixers make it difficult to prove where the sent bitcoins originated from, where they are received and where they are stored. They do this by mixing a users coins with coins sent by other users or with those in their reserve. Subsequently, these coins are sent to the address specified by the user (evidently from a different address). Hence, by having two independent wallets, they break the link between the two addresses. Due to their business model, tumbling services can be used for money laundering and therefore it can be illegal to mix large amounts as per anti-structuring laws [34]. Mixing services charge a fee per transaction.

2.2 Graph Visualisations

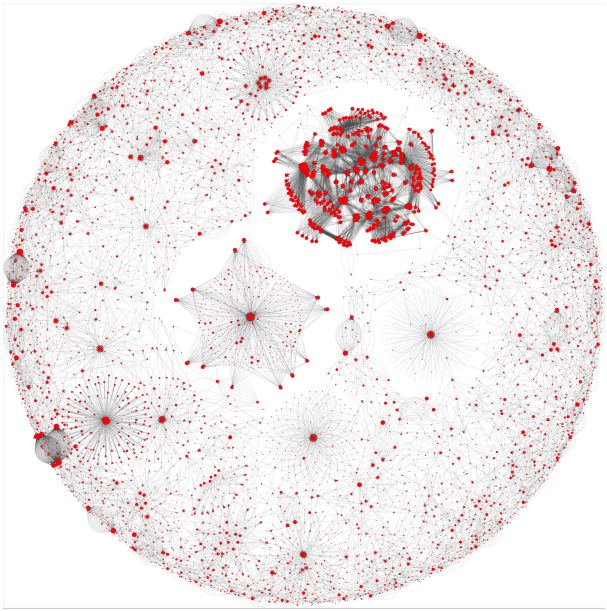


Figure 2.4: Early stage address graph of Block 400000 using the Kamada-Kawai layout.

2.2.1 Basic Graph Theory

Graph theory is essentially the study of lines and points. The lines are called edges (E) and the points are called vertices (V). This field has seen widespread application in modern society due to the abundance of large-scale and highly interconnected networks. Examples of such networks include the internet, social networks, biological structures and linguistics. Mathematically, a graph can be described as sets of vertices and edges (V, E). Edges can come in two forms, directed and undirected. This is used to described the relationship between the two nodes that the edge is connecting.

A graph that is represented mathematically, whilst providing structure, does not provide the locations of where to place the nodes in space. By visualising the network, a human can be helped in identifying underlying patterns in the network. Graph layout algorithms aid in this by describing ways in which to display graph characteristics. Initial experiments with the address graph in this study were done using the well known Kamada-Kawai algorithm [24]. However, because the DSI has historically used ForceAtlas2 for its Bitcoin visualisations, the primary layout algorithm was quickly chosen to be ForceAtlas2. This gave a better a visual benchmark with which to compare previous works.

2.2.2 ForceAtlas2

ForceAtlas2 is a force-directed layout algorithm. This means that it attempts to recreate physical forces in order to render a layout. Inspired by real force effects, it applies 'spring-electric' forces to nodes [21].

In force-directed algorithms, forces are assigned vertices and edges to simulate motion as these force repel and attract, until overall energy tends to zero. More specifically:

- All pairs of nodes repel each other
- Adjacent pairs of nodes (with an edge between them) attract
- A gravitation force is used to pull nodes towards a central fixed point.

Due to gravity, the resulting graph is round and disjointed sub-graphs are minimised. ForceAtlas2 applies repulsive forces dependent on the degree of nodes. Optimal performance is achieved by balancing a trade-off between speed and precision. ForceAtlas2 reduces computational complexity by using the Barnes-Hut optimisation ($\mathcal{O}(n^2)$ of ForceAtlas to $\mathcal{O}(n \log n)$) and as such is a good choice for a force-directed layout algorithm. It is scalable to millions of nodes.

Talk about different parameters?

2.2.3 Data Observatory Bitoin Visualisations

Utilising the KPMG Data Observatory (see Chapter 3), prior work at the DSI has been successful in graphing Bitcoin transactions. These are useful when compared to other bitcoin visualisations in that they provide systematic, top-down snapshots of the network [26]. This work is applicable in both a real time view of transactions in the mempool, and also at a per block basis. The first case allows transactions to be tracked as they propagate through the network. A future application can also be that once the presence of a particular known malicious/algorithmic structure is seen in the mempool, it can be tracked for further analysis. The second, per block, visualisation proves to be more useful for this study, by providing a look at the historical activity within a block. Work done by Rastogi [32] built upon these visualisations to extend across multiple blocks to give a more wholistic view of activity over time.

The graph layout algorithm used by McGinn et al. [26] was ForceAtlas2 and the graph design was as follows:

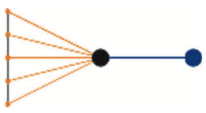


Figure 2.5: Transaction sourcing five equal input amounts (orange node and gray edge) from a single address (grey edges) and paying 25 BTC to a new address (blue node and edge).

- **Transactions** are displayed as nodes in a neutral colour, whose size is scaled based on the coinbase reward.
- **Input** nodes and edges are orange
- **Output** nodes and edges are blue.
- **Address** is a gray associative edge only if more than one input or output references the same address within scope.

The size of the inputs and outputs depends on their value. Associations with corresponding transactions are made by edges of the same colour as the input/output node. A chain of spends (blue edge to node to orange edge) can form if an output node becomes an input node. As shown in Figure 2.6, the graph structures produced per block can have various forms.

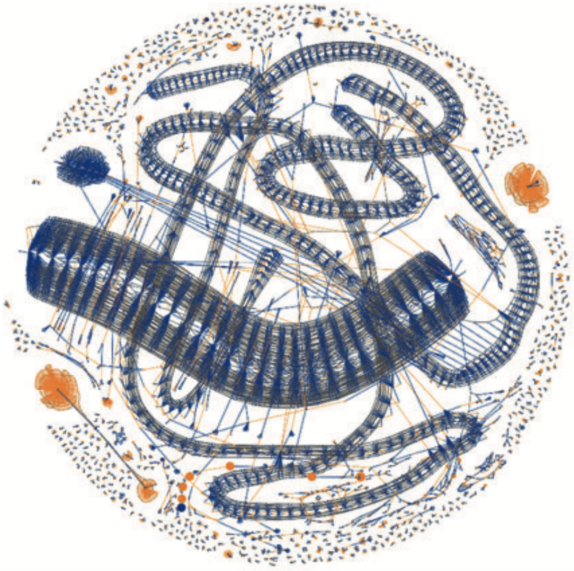


Figure 2.6: Parasitic worm structures (predominantly in blue) demonstrating a transaction rate Denial-of-service attack.

2.2.4 Other Existing Visualisations

- **BitConeView:**
  BitConeView is an online tool that visualises the Bitcoin network to trace the flow of bitcoin at a transactional level. It can be used to investigate mixing services and flows of historical funds. The analysis tools available include 1. UTXO, 2. Purity, 3. Budget and 4. Transfer analysis [15]. Whilst this is a useful tool for analysis if the transaction for investigation is specifically known, it is limited by its bottom up view of the network since it cannot detect groups of transactions or services themselves.
- **BlockSeer**
  BlockSeer is also an online analysis tool but it differs from BitConeView as it

does not perform analysis. However, it generates a transaction graph that can be traversed backward and forward to trace a coins origin and further movements [1]. Additionally it collects tags for known addresses and transactions using a crowd sourcing feature. This tool is useful in traversing the blockchain once a transaction, address or block are known, but does little to detect unknown services, or perform analysis on the flow of UTXOs.

- **Chainalysis**
  Chainalysis provides software tools as services for network analysis using graph based methods. Given an address, their Refactor tool can visualise, track and analyse associated transactions [2]. It can also find paths between different wallets and suspects under investigation. Chainalysis's services have been used by law enforcement agencies such as the IRS for its crackdown on cryptocurrencies since 2015 [19].

- **BitConduite**
  Bitconduite is a tool (still under development at the time of writing) which looks to analyse actors on the Bitcoin network, and understand the control and purpose of bitcoin flow [23]. It tries to categorise these actors (who are either individuals or services) based on their transaction characteristics. It groups similar entities using k-means clustering with a predefined number of clusters. Rather than analysing short term activity, Bitconduite aims to monitor long term activity such as frequency of transactions and the amount transferred per transaction. The system is limited in use due to its bottom up view of the network.

2.2.5 Discussion

Whilst all of these previous visualisations have been useful, they have not fully taken advantage of the relationships available from their data. All of the previous approaches have either focused on a top-down or bottom-up approach to analysis of the blockchain, and have not yet implemented a machine learning approach to their methods. This project combines the two approaches, along with incorporating machine learning models into the system design.

2.3 Machine Learning

Machine learning is a subset of artificial intelligence for automatic model building. The field can be broken into three main areas: supervised, unsupervised and reinforcement learning. Supervised learning is where the data being used to train the model has been labelled. Algorithms looks to generate a function that maps inputs to desired outputs. In unsupervised learning modelling is carried out on a set of unlabelled examples, in an attempt to infer underlying patterns or clusters. In reinforcement learning, software agents look to take actions in their environment in

order to maximise a cumulative reward. The machine learning field has seen a massive resurgence in recent times due to hardware advancements and increased source and aggregation of data.

Given the anonymous nature of the data at hand, a lot of the literature applying machine learning to Bitcoin has focused on unsupervised learning. Due to this, and suggestions from McGinn et al. [25], clustering methods such as K-Means clustering and clustering evaluation metrics were studied initially. However, as shown in the Results section, the extracted features did not lead to good separation in euclidean space, implying that K-Means clustering (and other distance based algorithms) would prove ineffective. As such, unsupervised methods were abandoned in search of a more supervised approach.

2.3.1 Classification Measures

For supervised learning models, the following measures are useful in evaluating performance:

1. Accuracy: Ratio of correctly predictions to total observations. This metric can be extremely misleading for imbalanced datasets (such as the one used throughout this project), and therefore needs to be cautiously interpreted with class balance in mind.

$$Accuracy = \frac{T_p + T_n}{T_p + T_n + F_p + F_n} \tag{2.1}$$

2. Precision: Ratio of correctly predicted positive observations to total number of predicted positive observations. Low precision is indicative of many false positive classifications.

$$Precision = \frac{T_p}{T_p + F_p} \tag{2.2}$$

3. Recall: Ratio of correctly predicted postive observations to all observations in the postive class. Low recall is indicative of many false negatives. Given that the deployed machine learning models will be trying to detect services on an anonymous dataset, it is important that this metric is high for this task.

$$Recall = \frac{T_p}{T_p + T_n} \tag{2.3}$$

4. F1 score: Weighted average of both Precision and Recall.

$$F1 = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{2.4}$$

2.3.2 Decision Tree

Decision tree learning is a method for approximating discrete classification functions by means of a tree-based representation. They can also be used for regression. A learned decision tree can be thought of as a set of if-then rules, which are used to classify an unseen data point. Decision trees consists of two main components: branches and nodes. A tree typically starts with a single root node, which branches into possible leaf nodes or decision nodes.

- Node: Test for a specific attribute of the instance (leaf node) or make classifications (decision node).
- Branch: Denote possible values for the attribute under test.

Typical decision tree learning algorithms include the ID3 and CART algorithms. In python scikit-learn, an optimised version of CART is implemented [4]. By employing a top-down greedy search through the space of possible solutions, a typical decision tree learning algorithm will do the following:

1. Perform a statistical test (information gain or gini coefficient) of each feature to determine how well it classifies the training examples when considered alone.
2. Select the feature that performs best as the root of the tree.
3. To decide the descendant node down each branch of the root (parent node), sort the training examples according to the value related to the current branch and repeat the process described in steps 1 and 2.

Advantages:

- They were chosen as they are simple to understand, interpret and visualise.
- Extend to multi-class problems

Disadvantages:

- Have a tendency to overfit by creating over-complex trees that do not generalise well to the underlying data. In order to combat this, parameters such as *maximum tree depth* and *minimum number of samples* were optimised. Pruning was also implemented, where poor predicting sections of the tree are removed to reduce complexity.
- Susceptible to outliers.
- An unbalanced dataset can heavily bias the tree. Because of this, class imbalance was tackled in various ways, as described in Chapter 5.

2.3.3 Random Forest

Random forest algorithm is an ensemble learning method that can perform both classification and regression. Random forests perform bootstrap aggregation (bagging),

meaning that they combine many weak learners (decision trees) into an ensemble that is much stronger. Several decision trees are built from random subsets drawn with replacement from the original training data. When splitting a node during construction of a tree, the split is taken on the best performing feature of a random subset of the overall features provided. By combining multiple decision trees whose predictions are achieved independently and then averaged, they reduce variance and overfitting to improve accuracy.

Advantages:

- Widely used in industry due to their effectiveness, scalabilty and parallelisabilty.
- Maintains accuracy even when a large proportion of data is missing.

Disadvantages:

- They are not easily interpretable as they are grey box learners.
- With a lot of data, random forest models can become quite large in size and slow to evaluate.

2.3.4 K-Nearest Neighbours (kNN)

Nearest neighbour methods look to find a predefined number of training samples closest in distance to an unseen data point, and make a prediction based on the class label which has the largest number of close samples. The number of neighbours (k) for kNN is predefined by the user, and the distance metric is most commonly chosen to be Euclidean distance.

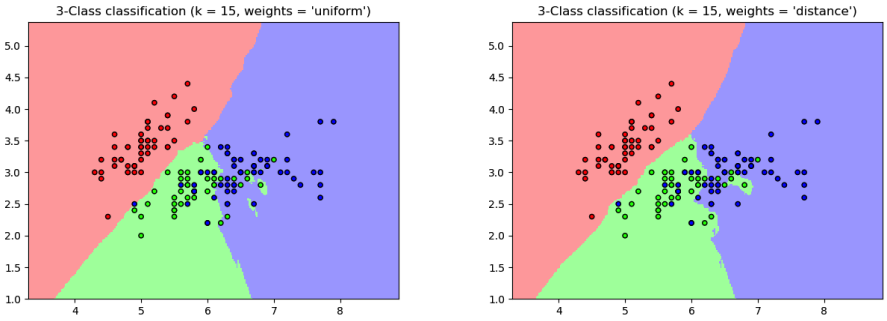


Figure 2.7: An example of classification using kNN [8]

Advantages:

- Requires no information on any prior probability distributions on the input data
- Little to no training time is required.

Disadvantages:

- It does not 'learn' anything from the training data, instead using the training data itself for classification (lazy learner).
- Does not perform well if the data is not linearly separable in vector space. Experimentation with 'k' value and distance metric is required.
- Computationally expensive making it an impractical and unscalable option with greater amounts of data (predicition scales linearly in time with data).
- Accuracy can be severely degraded with high-dimension data.

To improve accuracies, Principal Component Analysis (PCA) dimensionality reduction was used prior to kNN classification. PCA is also used to visualise users (described in feature vectors) in 2D-space. The data points were seen to be overlapping considerably, explaining the poor results of the KNN classifier. These are shown in the Results section.

2.3.5 Discussion

In order to derive insights into the behaviour of users, there was a need to choose classification methods that are interpretable to some level. For this reason, tree-based and nearest neighbour algorithms were chosen at the outset. They also had few hyperparameter tuning requirements when compared to SVMs (kernel, regularisation, slack variable) and neural networks.

The `scikit-learn` implementation of decision trees and random forests provide a `feature_importances_` metric that attaches a relative importance to features based on how much they aid in classification. This feature proves to be highly useful in determining the specific attributes of classes of users that differentiates them from each other.

Deep learning algorithms have been shown to be incredibly effective for complex problems such as image classification, natural language processing, and speech recognition. However, they need large datasets and considerable training time to work well [17]. They are also difficult to interpret due to their highly blackbox nature. Perhaps in extending the methods described in this study over the entire blockchain, there is greater merit to a deep learning approach. Over a larger timeframe, the underlying behaviour of users will be a lot more complex and a lot more data will be available, making it a suitable deep learning problem.

2.4 Previous Work Detecting Activity on the Bitcoin Network

The majority of past efforts have focused on using unsupervised learning methods and behavioural heuristics to cluster transactions and addresses.

2.4.1 Unsupervised Learning

Anomaly detection is a classic problem in data science that looks to discover rare occurrences in datasets. On the bitcoin network, illicit activities such as money laundering or dark web markets are considered to be anomalous, with the assumption that the majority of users act legally on the network. The research conducted by Pham et al. [29] looks to identify these anomalous activities using unsupervised support vector machines. They create two network graphs, a user and a transaction graph, from which they extract certain features used in detection. The dataset of users for this study is predefined, and as such their user graph is created from known users and is not algorithmically generated using the heuristics described in Chapter 4. The technique is shown to identify two out of a known thirty cases of theft which, while being a good step towards detecting suspicious transactions, is limited in scope and correctness. Baquer et al. [12] used k-means clustering combined with a feature set of transactions to differentiate between spam and non-spam transactions. This work gave good results, successfully detecting spam transactions during the DoS attack in 2015, and showing the negative impacts this attack had on the network. However, the lack of ground truth limited validation outside this known attack period.

2.4.2 Heuristics Based Detection

The work done by Meiklejohn et al. [27] is widely cited and has inspired many bitcoin analysis tools such as `walletexplorer.com` and `chainanalysis.com`. By using their own ground truth data, they are able to build upon heuristics developed by Androulaki et al. [10] and conclusively test its effectiveness. In this study, transactions are made with known services on the bitcoin network, and these known addresses are used as labels. Illegal activities are then investigated using traffic flow tracking techniques. These heuristics are employed in this project along with some self-devised additions. Further detail on the method and implementation is given in Chapter 4. The paper by Bitfury [16] looks to further the clustering heuristics with off-chain tags and a voting system for the address clusters. They were able to reduce clustering errors that these heuristics are prone to. This paper inspired the use of external off-chain information as labels for this project.

Chapter 3

Software Stack and System Design

This chapter describes the technologies used in this project and the key decisions made during implementation. The data collection phase is detailed, along with the problems faced. An overview of the overall system architecture is also given. User generation and machine learning analysis are discussed in later chapters.

3.1 Approach

By combining a bottom up and top down approach to the problem at hand, the system looks to use the data on the blockchain and combine it with features extracted from graph based analysis for machine learning models. Data is collected from a neo4j database, and fed into python code in dataframe form. By manipulating these dataframes, transactions are traversed to cluster addresses based on the heuristics described in Section 4.2. Using `Beautiful Soup` and `multiprocessing` packages, HTML documents on `walletexplorer.com` were scraped for address labels. Finally, all of these components were combined to perform service classification tasks, and graph visualisations (using Gephi and igraph) on the data observatory.

3.2 DSI Neo4j Bitcoin Database

DSI researchers have created a neo4j bitcoin database as part of the research carried out by McGinn et. al [25]. This database contains the Bitcoin blockchain up until block 426565. By using neo4j, the raw binary format of blockchain data is abstracted away to allow for much more structure, readability and ease of querying. As such, the primary source of data for this project was this database. It provided a large, clean dataset at the outset and allowed for focusing efforts elsewhere rather than wrestling with raw data collection and parsing. Neo4j represents data in graph format. As such, the key concepts to understand are 'nodes' (vertices) and 'relationships' (edges). Both these data types can have properties, such as type of node or

type of relationship. The DSI neo4j database schema is described in Figure 3.1. As shown, all relationships in the database are directed, and nodes have different labels specifying their unique roles.

As this database is not up to date with the current blockchain size, 540,036 at the time of writing, investigations are limited. However, this problem is easily resolved in future iterations, once a real time update to the database is made. The remaining system designed in this study will be able to accommodate for this, as the format of the data extracted from the database will remain the same.

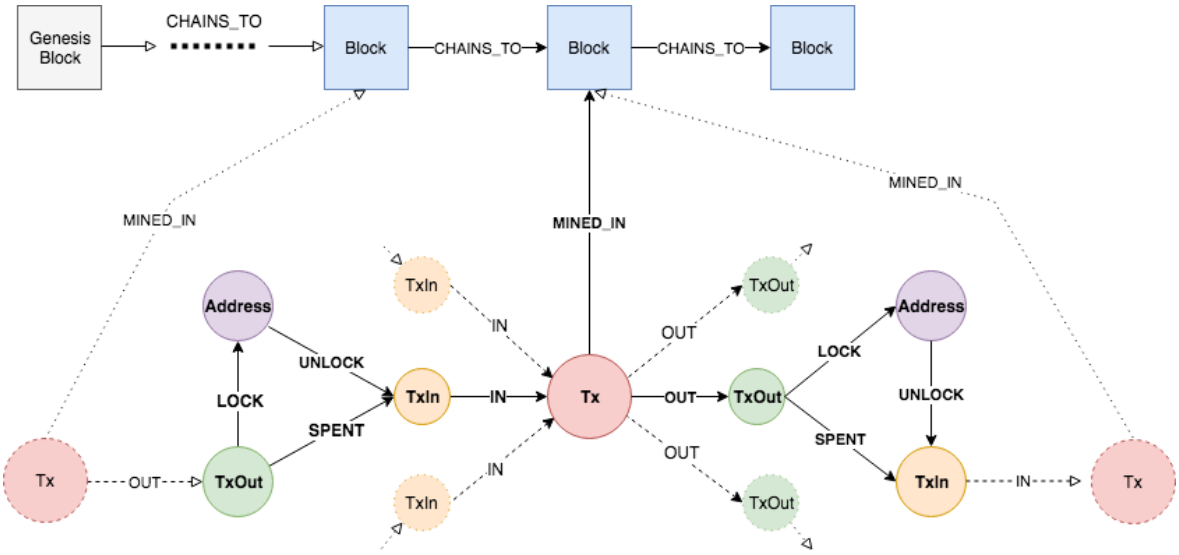


Figure 3.1: Visualisation of the nodes and relationships of the DSI Bitcoin neo4j database

3.2.1 Data Collection

The DSI bitcoin database was accessed via the internet and also an online platform. This platform was very useful as it provided a terminal from which to query the data database, along with a screen which returned results as graphs or tables. With this, it became easy to visually explore the database and write quick queries to check efficiency and correctness. The query language for neo4j is Cypher, which performs queries in a pattern matching style. Cypher was learnt during the background study period of the project, with ramp up being relatively quick due to the easy-to-read nature of the language. The py2neo python package was used to easily call Cypher queries from python applications. py2neo has good support documentation and was chosen because of its ability to execute directly from terminal and functionality to export extracted data as a pandas dataframe [9]. Dataframe conversion was crucial for further manipulations and machine learning analysis of the data.

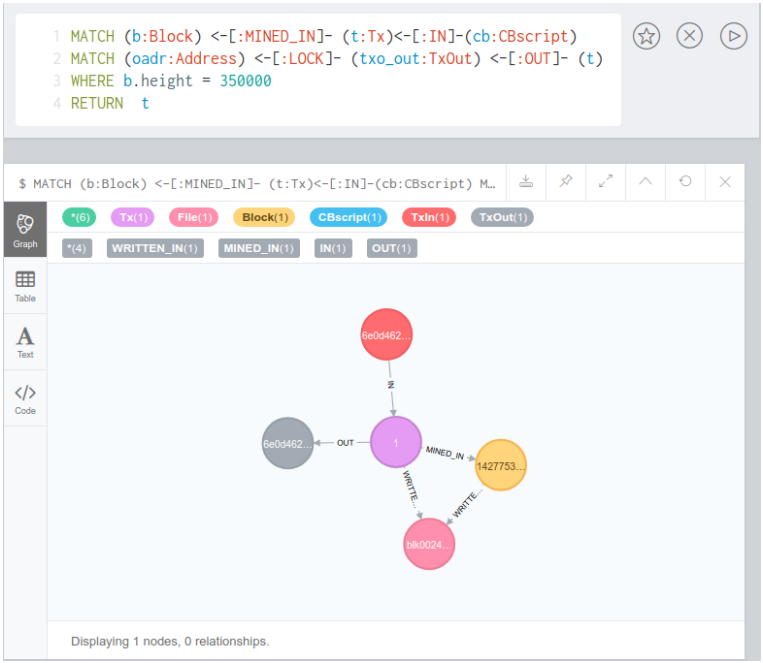


Figure 3.2: Snapshot of neo4j online platform, with query and result.

During the initial planning phase of data collection and usage, it was assumed that the database would be regularly queried during graph generation and model training. However, due to the fact that the data was accessed via the web, latency issues caused inconsistencies in the time taken for results to be returned. Additionally, the Cypher queries written for accessing data over thousands of blocks or for obtaining one-time change addresses (OTC - the meaning of which is described in Chapter 4) caused the database to regularly crash (even if queries were limited with the LIMIT keyword). One reason for these crashes may have been due to the fact that there were multiple users accessing the database at the same time. It is possible that heavy queries or queries in quick succession resulted in a lack of RAM, causing the system to blow up. The solution for this was to run queries overnight, gathering subsets of graph data, and storing these as local pickled copies for quicker access in subsequent cycles. Admittedly, a more robust solution would have been to design a new neo4j database as per requirements, such as an address graph, and query this local version for faster throughput.

3.2.2 Problems when moving from graph to dataframe

A possible source of error was discovered early on when trying to extract dataframes from graphs. This problem must be considered when representing a bipartite graph as rows and columns. Given below is simple example:

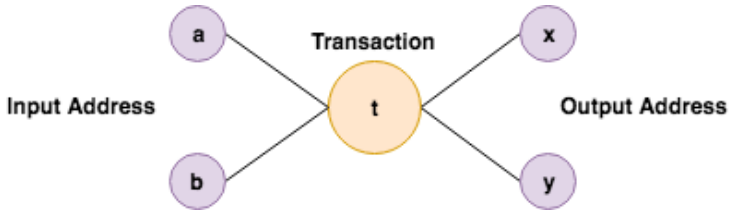


Figure 3.3: Simplification of DSI graph structure showing input and output addresses of a transaction

A dataframe representation of this graph is given in Table 3.1. As shown, all nodes are repeated multiple times. These repeated values have to be accounted for when taking the average of rows or calculating statistics on individual columns. Using the unique IDs of inputs and outputs to a transaction (TxIn and TxOut shown in Figure 3.1 resolved this issue.

Input	Transaction	Output
a	t	x
a	t	y
b	t	x
b	t	y

Table 3.1: Dataframe representation of Figure 3.3.

With this considered, per block data from the DSI database was extracted in a pandas dataframe consisting of the following columns:

- Input address per transaction (sending address node)
- Output address per transaction (receiving address node)
- Amount of Bitcoin from each unique sending address (In value)
- Amount of Bitcoin to each unique receiving address (Out value)
- ID of the unspent transaction output node being used as input - TxOut (IN)
- ID of created unspent transaction output node - TxOut (OUT)
- ID of transaction node (Tx)
- ID of each unique transaction input (TxIn)

```
MATCH (b:Block) <-[:MINED_IN]- (t:Tx) <-[:IN]- (txi:TxIn) <-[:UNLOCK]- (iadr:Address)
WHERE b.height = {BLOCK_NUMBER}
MATCH (txi) <-[:SPENT]- (txo_in:TxOut)
MATCH (oadr:Address) <-[:LOCK]- (txo_out:TxOut) <-[:OUT]- (t)

RETURN iadr.address as iadr, oadr.address as oadr, txo_in.value as input_val, txo_out.value as output_val, ID(txo_in)
as id_txo_in, ID(txi) as id_txi, ID(t) as id_t, ID(txo_out) as id_txo_out
```

Figure 3.4: Cypher query used to get individual block data from DSI neo4j database. The resulting dataframe from this query will not include the coinbase transaction. To accommodate for this, another query was run which looked for a transaction that had the coinbase script as input.

3.3 Python and igraph

Python3 was the primary language of development. Python offers many useful libraries in graph visualisation and analysis, as well as robust machine learning libraries that are widely used in industry. Due to its active community and widespread adoption, Python is an ideal platform for further development and future work. `pandas`, `numpy`, `scikit-learn` and `multiprocessing` are examples of the main python libraries used. Their implementation is discussed in later chapters.

Complex relationships of the bitcoin blockchain encourage a graph representation, however the sheer amount of data that is present requires a graph tool that can manipulate and analyse networks at a very large scale. There are two popular graphing tools available in python, `igraph` and `NetworkX`. While `NetworkX` (implemented in pure Python) performs well for small and dynamic graphs, `igraph` (implemented in C) is more performant in terms of speed and RAM usage for larger networks. As such, `igraph` was chosen over `NetworkX`.

Algorithm	igraph	NetworkX
Single-source shortest path	0.012 s	0.152 s
PageRank	0.093 s	3.949 s
K-core	0.022 s	0.714 s
Minimum Spanning Tree	0.044 s	2.045 s
Betweenness	21.6 mins	15.4 hours

Table 3.2: Performance comparison between igraph and NetworkX [6]

igraph offers common graph data structures for efficient and convenient graph generation. In particular, the TupleList class was used extensively. Tuplelist 'constructs a graph from a list-of-tuples representation' and 'assumes that the edges of the graph are encoded in a list of tuples'. This easily generated user or address graphs once fed a predefined list of edges between these nodes. Abstracting away from slow and complex graph structures, TupleList allowed very fast graph generation.

3.4 Gephi

Gephi is an open-source network analysis and visualization software package [13]. It can be accessed via a Java API or a graphical user interface. Gephi provides numerous graph layout libraries and advanced network analysis tools. ForceAtlas2 was of particular interest for this project and Gephi implements this with the Barnes Hut optimisation. However, Gephi's Java API is not intuitive and does not support edge attributes. Therefore it was reasonable to use python-igraph for graph generation and manipulation, and subsequently use the Gephi ForceAtlas2 library to apply the graph layout. This workflow is described in greater detail in Section 3.5.

3.5 KPMG Data Observatory

The largest of its kind in Europe, the KPMG Data Observatory (DO) at the DSI is an insightful tool that enables visualisations on a large scale. Consisting of 64 monitors placed circularly, 130 million total display pixels and 313 degrees surround vision, it allows for immersive and interactive view of data sets [3]. The observatory is accessed using a Microsoft Surface Tablet which runs a custom-built interactive app designed by DSI researchers. The DO was the last step in the overall system design, taking in GraphML files of structured, labelled and coloured graphs from igraph and Gephi to visualise on its screens.

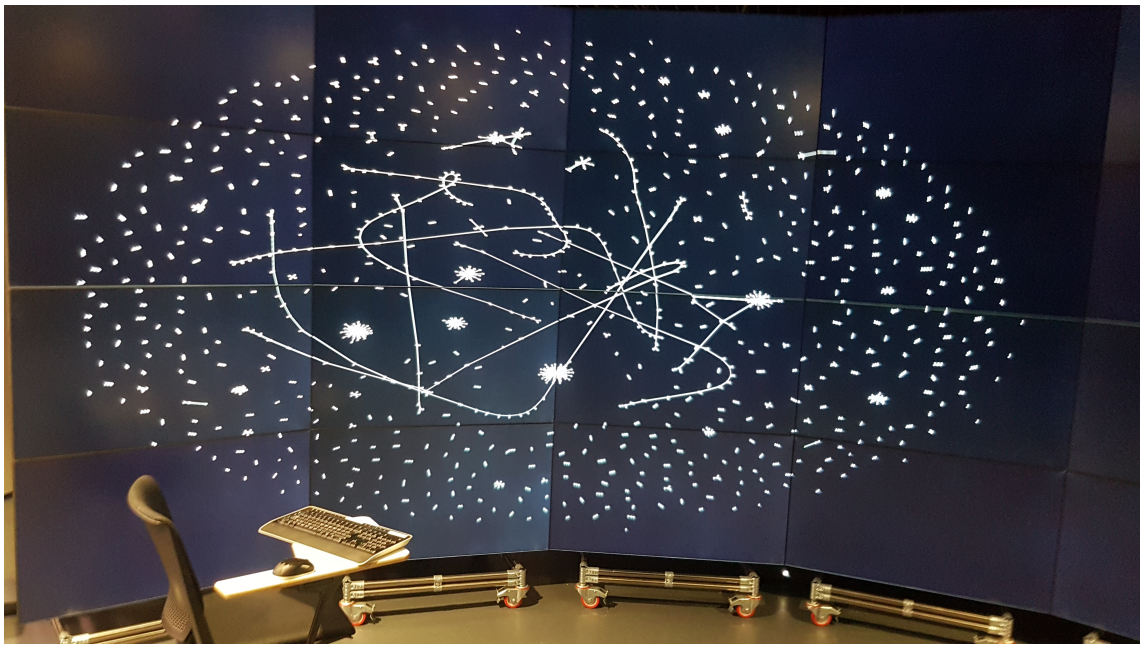


Figure 3.5: Deploying the user graph on the observatory

Using the DO, the visualisations produced from this project, along with other results, were shown on a large scale not possible on smaller screens. When compared along-

side previous transaction graphs, this allowed for a collaborative viewing experience to reveal interesting patterns and structures in address and user behaviour.

3.6 Overall System Design

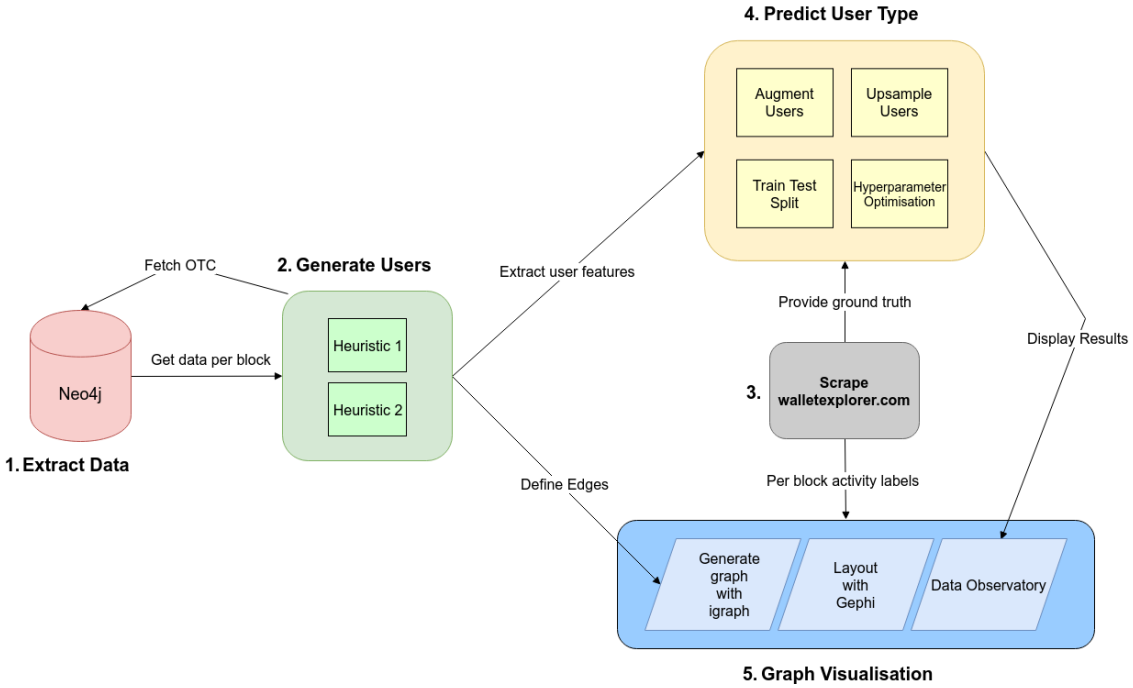


Figure 3.6: System design

There are four major components to the system architecture, as shown in Figure 3.6.

- The first step involves collecting block data as described in Section 3.1.
- The user generation block applies address clustering heuristics, while also extracting relevant features specific to users and defining the user graph. Ground truth labels for addresses and users were obtained by scraping the open datasets available on walletexplorer.com (Chapter 4)
- Generated users and their features are then used to train and test machine learning models defined in `scikit-learn`. Many manipulations to the data were required due to class imbalance, and hyperparameter optimisation of the models was done using a combination of grid search and cross-validation. (Chapter 5)
- The edges between users and addresses describing the user graph were fed into igraph for graph generation. Then, by exporting this graph as a GraphML file, Gephi was used for graph layout in 2D space. Finally, the DO platform was utilised to showcase results and visualisations. (Chapter 6)

Chapter 4

Generating users from addresses

One of the primary contributions of this project is the clustering of addresses into users for any block or range of blocks. This chapter will continue this discussion by explaining the clustering heuristics, detailing the algorithms used for user generation and their limitations. Methods for scraping off-chain resources and tagging users with these labels are also explained. However, the chapter starts with address graphs and the problems associated with them.

4.1 The Need for a User Graph

Initially, given the labels obtained from `walletexplorer.com`, it was thought that the address graph would suffice in characterising the behaviour of different services on the bitcoin blockchain. However upon generation of the address graph, it revealed inherent inaccuracies due to the pseudo-anonymous nature of bitcoin transactions. The address graph is generated by defining edges between pairs of addresses that are linked by a transaction. As such, the address graph describes all addresses and their interconnections (via executed transactions) within a block. However, the fundamental problem with this is that transactions abstract away from actual interactions between specific addresses. For example, as shown in Figure 4.1, suppose three address a, b & c are used as inputs to the transaction t with output addresses being x, y & z. Even though it is known that the six addresses are connected by the transaction, specifics such as which input address sent what amount to which output address are unclear. Hence, the features extracted per address in the graph cannot be considered wholly accurate.

By exploiting characteristics that are intrinsic to the usage of the bitcoin protocol, a user graph can be constructed where addresses belonging to entities are gathered. An entity (user) that controls an address, is expected to participate in all transactions which involves that address. As per the bitcoin protocol, this would mean that the user would have knowledge of the private key corresponding to that address, which is used to sign for transactions and validate the transfer of funds. Hence, the address

graph is condensed into users which control sets of addresses. While this clustering may not be 100% accurate and resulting users remain anonymous, the user graph is able to reveal user-to-user and user-to-address level transactions clearly. Interactions between tangible entities and the flow of funds between them can be analysed more confidently than just at an address level. At a high level, this process consists of considering each transaction along with its input and output addresses. From these, using the two heuristics described in the next sections, addresses are collated into users.

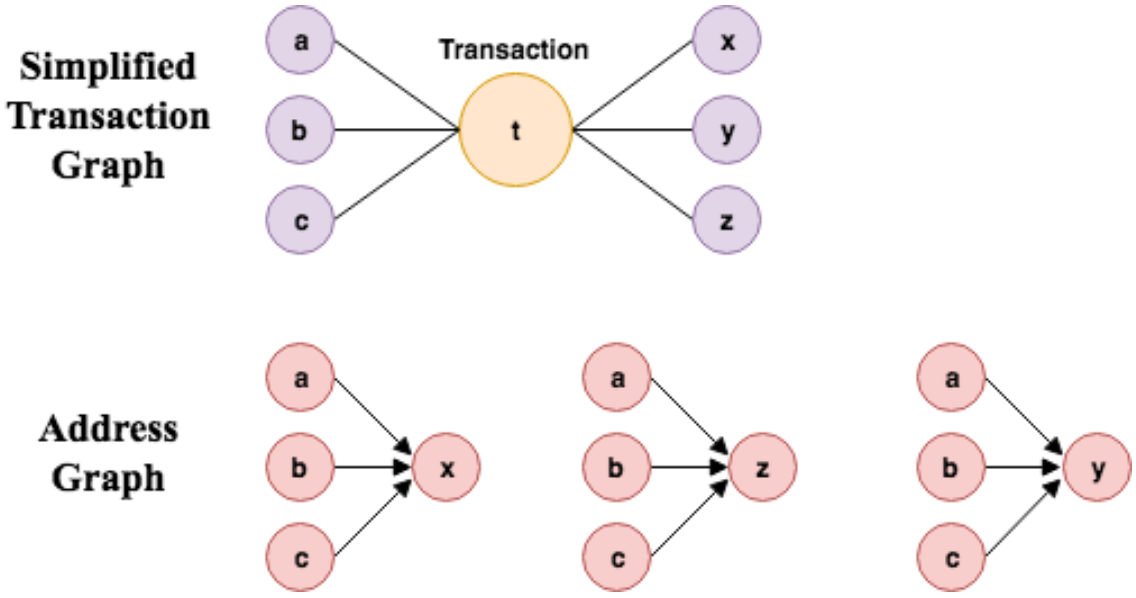


Figure 4.1: Converting from the transaction graph to the address graph

4.2 Address-to-User Clustering Heuristics

These address clustering methods have been detailed by Androulaki et al. [10] and Meiklejohn et al. [27], and their methods have been replicated in this study. There are other additional ways to cluster addresses (as discussed in Section 4.3.1), but as a first step towards analysis, only the following two heuristics were implemented in this project.

4.2.1 Heuristic 1

Consider a transaction that has multiple input addresses. It is assumed that all input addresses belong to the same user. The basis for this assumption is the fact that by signing the public key for a transaction, a user must have knowledge of its private key. Hence, it is assumed in a situation where there are multiple keys that need to be signed, a single user would own all of the associated private keys for the majority of cases.

This heuristic is formally explained below:

> **HEURISTIC 1:** *If two (or more) addresses are inputs to the same transaction, they are controlled by the same user; i.e., for any transaction t, all $pk \in$ inputs(t) are controlled by the same user.*
>
> Here pk refers to a set of public keys and inputs(t) is the set of addresses used as inputs for t.

The outcome of such a heuristic is demonstrated in Figure 4.2. Consider three different transactions that have already been executed on the network. These transaction can be occurring in the same block, or in different blocks at different points in time. Given transaction $t1$, it can be assumed that if there are multiple inputs to the transaction then all of these addresses belong to the same user. As such, User 1 is assigned the addresses A, B and C. In a similar manner the addresses, F and G, of transaction $t3$ are all clumped as belonging to User 2. This is possible because there is no intersection between the two sets of addresses assigned to each user. In the case of transaction $t2$, where there is an overlap of the input set with the addresses of User 1, the union of both sets is assigned to User 1. Hence, eight input addresses have been condensed down to two distinct users.

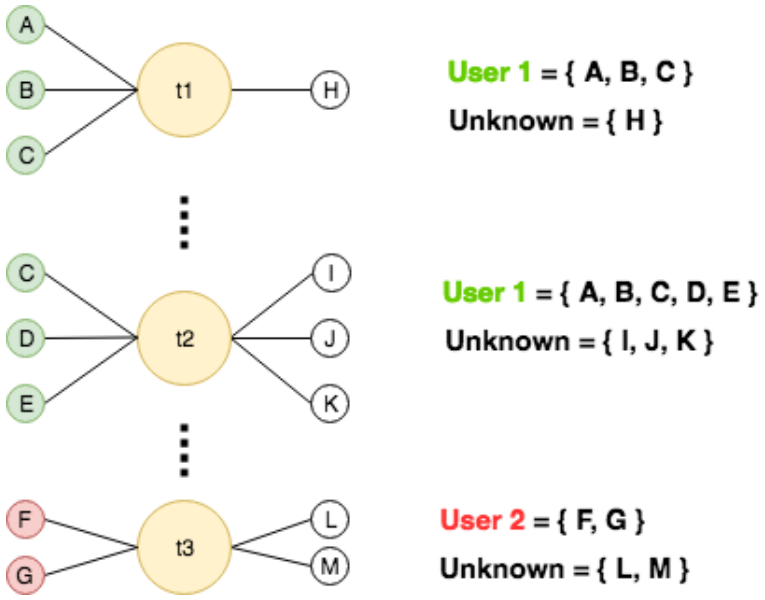


Figure 4.2: Associating multiple input addresses to users over multiple transactions using heuristic 1

Intuitively, this heuristic makes a lot of sense. Considering a scenario in which a user of the network needs to transfer funds, and the amount required is greater than any that can be unlocked by any *one* of their owned addresses. In this situation, the user will clump multiple addresses together, and satisfy the amount required to be transferred in one go. The less efficient and more costly alternative would be to send incremental amounts one at a time from each owned address, incurring small

transaction fees every time. Additionally, multiple users are unlikely to reveal their private keys to each other, which will happen if they collectively sign the public keys to be used as inputs to the same transaction.

4.2.2 Heuristic 2

Another intrinsic property of the bitcoin protocol is the requirement that users cannot predivide bitcoins before spending them through a transaction. In other words, bitcoin can only be divided by being spent. For example, if a user has 5 BTC and wants to send 3 BTC to another, the bitcoin protocol requires that they send the entire 5 BTC amount during payment, and then return the 2 BTC change amount to themselves, hence splitting the funds. An idiom of use of this is that the Bitcoin creates a new one-time change address (OTC) internally that will likely never be reused (such as for accepting payments), and may be unknown to the user. By identifying these change address in conjunction with Heuristic 1, the address graph can be further clustered by not only using input addresses but also some output addresses. Heuristic 2 looks to identify these change addresses and is a bit more involved and dangerous than the first. If done incorrectly, it can collapse the entire graph.

Formally, this heuristic is described as follows:

HEURISTIC 2: Given a set of transactions arranged chronologically (over one or multiple blocks). Consider a transaction t within this set. A public key pk is a one-time change address if it satisfies the following conditions:

1. This is the first appearance of pk in the blockchain.
2. t is not a coinbase transaction
3. $pk \notin inputs(t)$, i.e. pk is not a self-change address.
4. $\forall\ outputs(t)$, condition 1 is met only for pk, i.e. t contains only one output address appearing for the first time.

Combining with Heuristic 1, the OTC address is controlled by the same user as the input addresses. Therefore, the owner of $inputs(t)$ also owns the OTC address $pk \in outputs(t)$ (if OTC exists).

4.3 Limitations of Heuristics

The first heuristic is expected to be very accurate as common spending by different users would occur only if there exists a high degree of trust between them (which is somewhat of a rarity given the anonymous and trustless nature of the network). However, different from the publishing year of the work by Meiklejohn et al. [27], multiple services now exist for multi-input transactions. Additionally, as OTC is

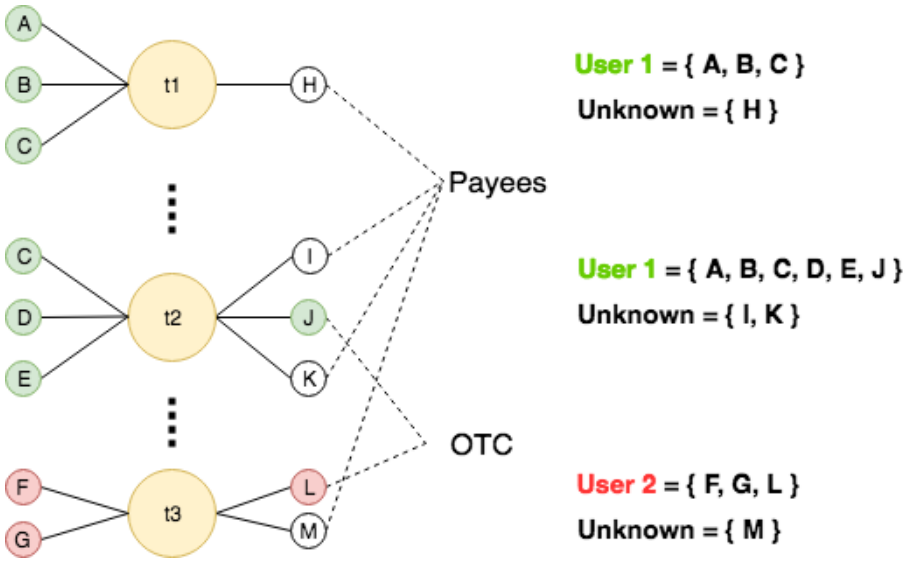


Figure 4.3: Combining both heuristics 1 and 2. Transactions $t1$, $t2$ and $t3$ are not coinbase transactions. Output addresses J and L were used for the first time in the blockchain, and are not inputs to their respective transactions. If assuming a scenario where neither L and M had been seen before, neither would have been allocated as OTC.

more of an idiom of use rather than explicitly defined in the bitcoin protocol it lacks robustness, being highly prone to any change to changing or adversarial usage of the network. Suggested in Section 9.1 are some additional heuristics that can be implemented, considering the Bitcoin network as of the time of writing. Given more time, these heuristics would have been relatively easy to integrate with the current system.

4.4 User Generation Algorithm

After running the Cypher queries on the neo4j DSI database as described in Section 3.1, a chronologically sorted dataframe is obtained. This dataframe contains block data ordered by when the blocks were mined. It can consist of information on one block, multiple sequential blocks, or multiple non-sequential blocks in the chain. It is important that the dataframe is sorted chronologically, therefore the Cypher queries were specifically executed on a block number sequence presorted in ascending order.

When a block is mined, the reward value (newly minted bitcoin) is transferred to the address of a miner. Knowing that this process is true for every block that is mined, the algorithm first goes through the entire dataset and creates users from miner addresses. It does this by executing a slightly tweaked version of Algorithm 1, in which it looks for transactions that have no defined input address. Hence coinbase transactions, which are slightly different to regular transactions, are accounted for. An imaginary 'Bitcoin' user is also created to keep track of the amount of bitcoin that was realised and number of coinbase transactions that occurred in the block slice

under analysis. But this user is ignored in all further analyses, as it not an actual entity.

A User class was defined to store information relevant to a user. This class has five attributes:

- **addrs** [$set()$] All input addresses belonging to user
- **cadrs** [$set()$] All one-time change addresses belonging to user
- **sending_tx** [$set()$] All transactions in which user sent bitcoin
- **receiving_tx** [$set()$] All transactions in which user received bitcoin
- **blocks** [$set()$] All blocks in which the user has been active

Once Algorithm 1 has generated all regular and miner users using heuristic 1, Algorithm 2 finds all potential OTC that belong to these users as per the second heuristic. The neo4j database greatly showed its usefulness here. Given the block size under analysis, it enabled the discovery of all addresses used for the first time ever in the blockchain. In other words, all addresses that had never been used as either input or output addresses were flagged.

To validate the correctness of the algorithms, the researchers from [27] were contacted. They generously provided their labelled dataset of hand-made users and participated transactions from their study. Using this as ground truth, the algorithm was validated to see if it was correctly identifying the users. A high user detection accuracy was achieved on random sub samples of the dataset, allowing progression to the next steps of the project.

4.5 Constructing the User Graph

After generating all possible users from the block sequence under analysis, a directed user graph can be constructed, with edges representing transactions between nodes. This graph is essentially a condensed version of the address graph, with the difference being that all addresses belonging to a user are grouped into their respective user super-nodes. Hence, the user graph consists of all users found by the two heuristics, along with all other output addresses that remain unassigned. As such, certain unassigned output addresses may merit further exploration if many users are seen to be transacting with them. Algorithm 3 shows how this graph is generated. The $user\_tags$ input parameter (described in more detail in Algorithm 4) is a dictionary mapping addresses to service labels.

Algorithm 1 Generate users using first heuristic

```
    procedure GET_USERS_HEURISTIC_1(block_dataframe)
        input_addresses ← all input addresses in block_dataframe
        users ← ∅
        already_seen_addrs ← ∅
 5:     for input_address ∈ input_addresses do
            if input_address ∉ already_seen_addrs then
                user ← GET_ONE_USER(input_address, block_dataframe)
                if address of user ∩ output address of transaction ∈ block_dataframe
    then
                    insert transaction into user receiving transactions
10:             insert user into users
                insert input_address into already_seen_addrs

    function GET_ONE_USER(input_address, block_dataframe)
        addresses_to_investigate ← {input_address}
        seen_transactions ← ∅
15:     all_input_addresses_of_user ← ∅
        User ← initialise empty User class
        while addresses_to_investigate do
            current_address ← remove first address in addresses_to_investigate
            for every transaction involving current_address do
20:             if transaction ∉ seen_transactions then
                    input_addrs ← all inputs to transaction
                    insert transaction into seen_transactions
                    insert input_addrs \ current_address into addresses_to_investigate
                    insert input_addrs into all_input_addresses_of_user
25:     insert all_input_addresses_of_user into user addresses
        insert seen_transactions into user sending transactions
        return user
```

Algorithm 2 Modify existing users using second heuristic

procedure GET_USERS_HEURISTIC_2($block\_dataframe$, $users$)

$possible\_otc\_addrs \leftarrow$ all new unseen addresses appearing only once as output addresses in $block\_dataframe$ and not being used as input addresses and not receiving addresses in coinbase transactions

for $user \in users$ **do**
  $user\_otc \leftarrow \emptyset$
  for $transaction \in sending\_tx$ of $user$ **do**
    $potential\_otc \leftarrow \emptyset$
    $oadrs\_from\_tx \leftarrow$ all output addresses of $transaction$
    $iadrs\_from\_tx \leftarrow$ all input addresses of $transaction$
    for $address \in oadrs\_from\_tx$ **do**
      if $address \in possible\_otc\_addrs$ and $address \notin iadrs\_from\_tx$ **then**
        insert $address$ into $potential\_otc$
    if only 1 $potential\_otc$ **then**
      insert $potential\_otc$ as change address of $user$

Algorithm 3 Construct user graph from generated users

procedure CONSTRUCT_USER_GRAPH($block\_dataframe$, $users$, $user\_tags$)
  $edges\_dataframe \leftarrow$ iadr and oadr columns of $block\_dataframe$
  for $user \in users$ **do**
    for $address \in$ addresses belonging to $user$ **do**
      Replace $\#user$ in place of $address$ in iadr or oadr column of $edges\_dataframe$
  $input \leftarrow$ iadr column from $edges\_dataframe$ now containing users
  $output \leftarrow$ oadr column from $edges\_dataframe$ now containing users
  $edge\_value \leftarrow$ sum transaction amount for each unique input-output edge in $edges\_dataframe$

  $edge\_tuple\_list \leftarrow [(input, output, bitcoin\_value)]$ ▷ $bitcoin\_value$ *is an edge attribute between transacting nodes*

  $graph \leftarrow$ `igraph TupleList`($edge\_tuple\_list$)
  for $vertex \in graph$ **do** ▷ reminder: *vertex can be user or unallocated address*
    if $vertex$ contains $address \in user\_tags$ **then**
      annotate $vertex$ with label of $address$
  Adjust size & colour for $vertices$ & $edges$ of top 10 users with most addresses in $graph$
  Write $graph$ to `GraphML` file

4.6 Tagging Users

With users and the user graph generated, it is necessary to label them with any corresponding tags found from off-chain sources. This provides annotations to visualisations and allows for supervised machine learning models. There are two ways in which these labels can be attained. The first would include making hand-made transactions the Bitcoin network by self-owned addresses and transacting with addresses of known services. In the interest of time, this method was not chosen. The alternative is to use off-chain blockchain exploration tools, and for this two free and publicly available datasets were identified: `blockchain.info/tags` and `walletexplorer.com`. Whilst `blockchain.info` is the more popular site, it has a relatively small number of tags of services, and many tags are self-identified addresses of individual users using the site. Hence, a lot of the tags are prone to error and are generally unreliable. To effectively collect tags from this source, there needs to be a human in the loop to review each tag and assess its correctness. Such a system would clearly be impractical on a large scale. Perhaps a more efficient system would be to implement a natural language processing algorithm that can filter ambiguous tags.

Address	Tag	Link	Verified
12iL16ZCSyjU7gEaeLDasiEfsUBUaEozJ3	jojo was here	https://j0j0gr.wordpress.com/2015/03/02/hello-world/	✗
134ZnmvWpGDGSwU6AnkgSEqP3kZ2cKqruh	asdfafdsarewr	https://blog.blockchain.com/2014/08/29/how-to-understand-the-bloc...	✗
1AfT83fMEGa43JQYAo8j9HMPouaZzzCyn7	abcde	http://localhost:8080/tx/21581c0f34ea4927684d6046118ca9d601046ab6...	✗
1GLENCoJHdKjVnLUmQGnbL2SjYWKKKeRc9	1GLENCo	http://uranus.glencooper.com/	✗
1HeyNSA2TW3TixWRD6DnXipq4jhDVXePCP	1GLENCo	http://1glenco.glencooper.com/	✗
1GLENCoSJJHH7FahyhTi16iwZQcHtMbxA4	1GLENCo	http://1glenco.glencooper.com/	✗
3QercMjovsSjuPtJSaYNpzMtgiRb5SJuVd	BX.in.th Cold Storage	https://bx.in.th/info/transparency/BTC	✗
1FR6uK7J8JakrcEcuLbopjpnmL2hYK6wAa	Leon03	http://block.okcoin.cn/btc/search/1FR6uK7J8JakrcEcuLbopjpnmL2hYK6...	✗
1CUBJE4zo7B5W54Wnm8FH41NGkvetZKKoL	To build a church in Poland UE	http://block.okcoin.cn/btc/search/1CUBJE4zo7B5W54Wnm8FH41NGkvetZK...	✗
1F9v3DZKmaesQntE7imZt7581cqw6Zd51i	Jesus Christ is risen !!!!!!!	http://block.okcoin.cn/btc/search/1F9v3DZKmaesQntE7imZt7581cqw6Zd...	✗
1ApPvzbESUjW1mrWwmyP6uxeQWXbMkX17f	Europe ONLY to Christians !!!!	http://block.okcoin.cn/btc/search/1ApPvzbESUjW1mrWwmyP6uxeQWXbMkX...	✗
1HZo14xXKRrzt1H3o6PpQBvPJbtbR4KFe2	Bad dude tagggg	https://www.reddit.com/r/SheepMarketplace/comments/1rvlft/i_just_...	✗
1CbR8da9YPZqXJJKm9ze1GYf67eKAUfXwP	testing	https://www.reddit.com/r/SheepMarketplace/comments/1rvlft/i_just_...	✗

Figure 4.4: Sample of tagged addresses available on `blockchain.info`, showing the high level of variance in the quality of tags

For these reasons, `walletexplorer.com` was chosen as the off-chain source for address labels. Additionally, `walletexplorer.com` has collected its tags using the first address clustering heuristics described in Section 4.2.1. In order to get an initial address tag for a service, the site could have scraped public forums, used bots to trade on exchanges for their addresses and also collected verified tags on `blockchain.info/tags`. With these initial tags, `walletexplorer.com` uses the first address clustering heuristic (described in Section 4.2.1) to find more owned addresses. Hence the labels collected from `walletexplorer.com` are a good fit for the users generated in this project, as both use similar methods to collate addresses.

`walletexplorer.com` contains a very large database of labelled addresses. To scrape this website for all necessary labels in a reasonable timeframe, the python `multiprocessing` package was used [31]. 64 cores were used to scrape the site in parallel and collect tags for each service, e.g. `Bittrex.com`, `SatoshiDice.com`, etc., in a dataframe. Previously generated users from Algorithms 1 & 2 were subsequently tagged using these dataframes containing tagged addresses of services.

Algorithm 4 Label users with address tags

procedure TAG_USERS($block\_dataframe$, $users$, $services\_dataframe$)
  $types\_of\_service \leftarrow$ [exchanges, gambling, pool, mixer]
  $address\_tags \leftarrow$ initialise empty dictionary
  for $service \in types\_of\_service$ **do**
    $tagged\_addrs \leftarrow$ tags $\in$ $services\_dataframe$ $\cap$ unique addrs $\in$ $block\_dataframe$
    if any $tagged\_addrs$ **then**
      for $addr \in tagged\_addrs$ **do**
        insert $addr$ with $service$ tag into $address\_tags$
  $user\_tags \leftarrow$ initialise empty dictionary
  for $user \in users$ **do**
    if $address$ of $user \in address\_tags$ AND $address$ tagged as only 1 $service$ type **then**
      insert $user$ with $service$ tag into $user\_tags$

When using the generated $address\_tags$ dictionary (described in Algorithm 4) to label users, if a user owns two or more addresses labelled as different types of services, then the user is not considered to be any of those services. Evidently, the clustering heuristics have failed here. In such a case, the user itself may be a service that allows for multi-input transactions, resulting in it being multiply labelled by the address tags of its client users. However, as shown in Figure 5.2, when looking over a large number of blocks only a very small percentage of users have been mislabelled, indicating good robustness of the clustering heuristics.

Chapter 5

Detecting users on the network

Having generated users and the user graph as well as labelling them with off-chain sources, it is now possible to apply machine learning models for user detection. This chapter gives an overview of the class imbalance problem and how it was dealt with. Features extracted from users are also explained, along with the techniques used for hyper-parameter optimisation.

5.1 Class Imbalance

There are five categories of bitcoin services available on `walletexplorer.com`: 1. Exchanges, 2. Pools, 3. Gambling, 4. Services/Others, 5. Old/Historic. Underneath each of these categories are the links to tagged addresses and transactions of individual services that belong to that category.

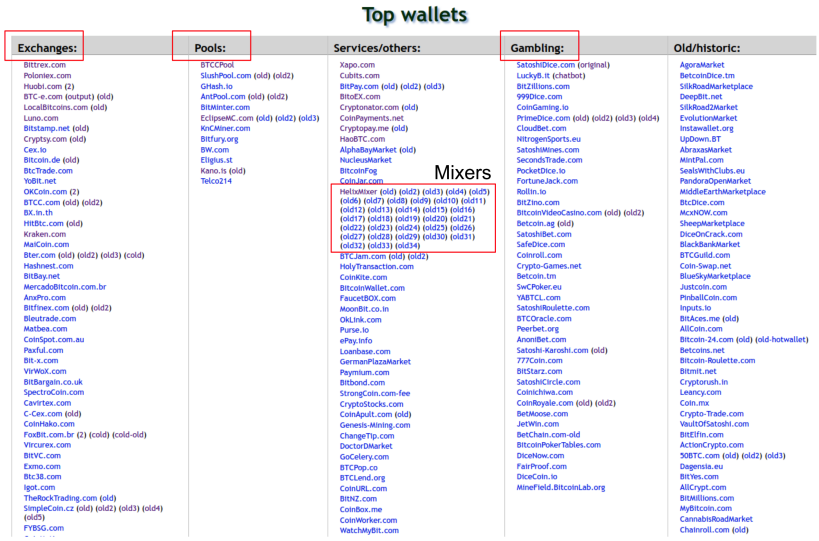
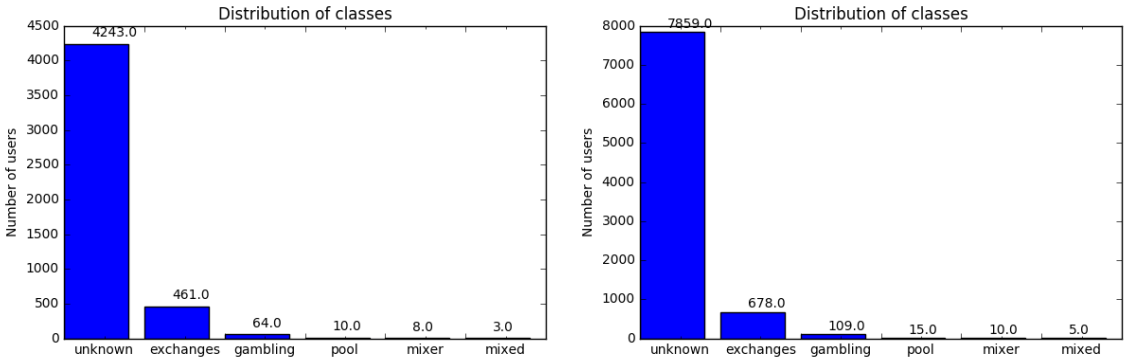


Figure 5.1: Screenshot showing the various categories and websites available on `walletexplorer.com`

Using the algorithms described in Section 4.6, tags for the services of interest (namely those from Section 2.1.4) were collected and used to label generated users. For the user detection task, it was decided to learn whether or not a user belonged in a particular category, rather than trying to identify their specific website or service. For example two different users, `bittrex.com` and `btctrade.com`, both belong to the 'Exchanges' category. Rather than identifying which user is `bittrex.com` and which is `btctrade.com`, the model will look to classify both users as an exchange. The reason for this was because more utility is derived from the model learning how exchange services generally behave to subsequently be able to identify new, unseen exchanges that have not been tagged.



(a) 10 Blocks

(b) 20 Blocks

Figure 5.2: Class imbalance for a continous range of 10 vs 20 blocks (observed between blocks 400500 & 400520)

The first experiment looked over a random subset of 10 continuous blocks, to gauge the distribution of users belonging to each category. A class imbalance problem became immediately apparent. Shown in Figure 5.2(a), the highest proportion of users have no tags and are placed in the 'Unknown' category. If a user in the unknown/-control class was actually an unlabelled sample belonging to one of the other 4 categories, the training set would be fundamentally flawed. Therefore, this 'unknown' or control class was ignored, with only labelled samples from one of the known categories being used for learning.

As expected, the 'exchanges' class is next to contain the most amount of users. This is because `walletexplorer.com` contains a relatively large amount of address tags for exchanges when compared to its other categories. Since it is dependent on the unequal amount of information that `walletexplorer.com` has for its categories, this imbalance problem stems from the data source and will persist in all labelled user sets. Admittedly, the only way to truly fix the class imbalance problem would simply be to gather more data by aggregating from other sources.

In an attempt to alleviate this issue, the subset of blocks over which to generate users was increased to 20 (Figure 5.2(b)). However it was quickly realised that the imbalance problem will remain no matter the size of the subset. This is a result of the address clustering heuristics working correctly. Given a scenario in which a

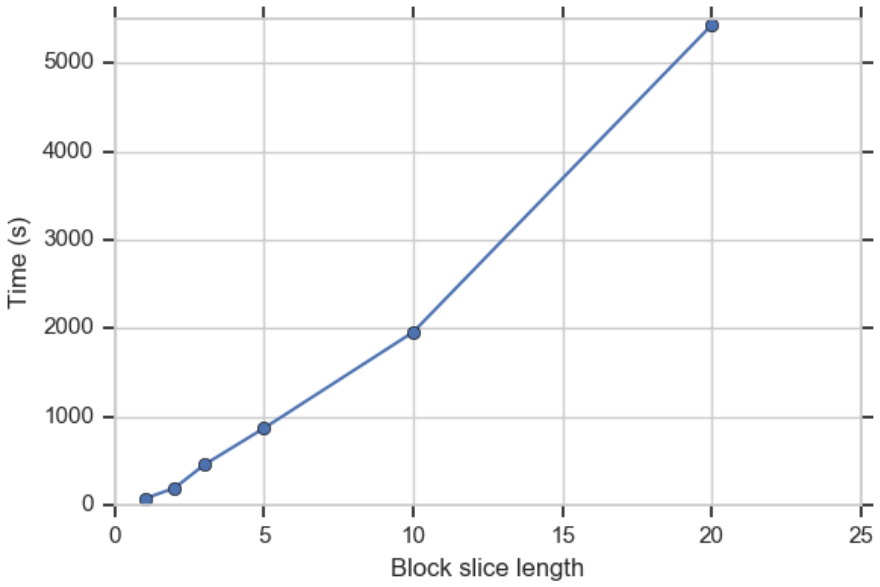


Figure 5.3: Time vs size of observed range

gambling user is active over a number of blocks, even if the observed number of blocks is increased, the *same* user will be seen over the larger slice as well. In other words, even if different owned addresses are used over the additional blocks, most will be clustered to the same owner by Algorithms 1 & 2. Thus, even though more information is appended some existing data points in the feature dataset, the number of data points themselves will not increase significantly.

5.1.1 Looking over larger blocks

Class imbalance is a common occurrence in many machine learning applications, and there is plentiful literature on how to tackle this issue. Evident from Figure 5.2(a), balancing methods would fail as 8 `mixer` users are just not enough data to oversample on, and downsampling from 461 `exchanges` users down to 8 would be highly unrepresentative of the `exchanges` class. Therefore, there was a need to gather more data on users, and naturally this was to be done by generating them over a much larger number of blocks. However, by requiring many dataframe manipulations, the current algorithms (1 & 2) for user generation are quite computationally intensive. As shown in Figure 5.3, as the number of blocks under analysis is increased, the time taken to generate users and user graph increases extensively. Evidently, looking over an even larger range than shown would have been infeasible.

To overcome the issue, a fast workaround was implemented to create a lot more data points. Rather than generating all users within a large range, e.g. 1000 blocks, many more were generated by independently sampling from much smaller slices over a 1000 blocks. Even if a user observed in one slice is seen again in another slice, it will be counted as two different users (belonging to the same class). As a

result, two different data points are observed in the feature set rather than just one. This operation can be parallelised since the generation of users between different slices is done independently, resulting in much faster generation over large numbers of blocks (distributed over 64 CPUs, 10 block slices of users over a range of 1500 blocks can be generated in 5 hours). **For all user type prediction experiments, the block slice length was set to 10 blocks (results can be seen in Chapter 6).**

Algorithm 5 Generating users over a large number of blocks (parallelisable)

```
1: procedure ADD_USERS(users, services_dataframe, step_size, block_range)
2:     feature_set ← initialise empty dataframe
3:     for ∀block_slice ∈ block_range do
4:         extract features from users
5:         label users using services_dataframe
6:         discard all users other than pools, exchanges, mixers & gambling services
7:         append features to feature_set
8:         slide to next block_slice depending on step_size
```

To ensure that there is no overlap between the slices (which could repeat observations in the feature set) the step size between block slices needs to be greater than or equal to the number of blocks in a slice. It is also important that users are sampled from block slices of equal length to ensure uniformity in user features.

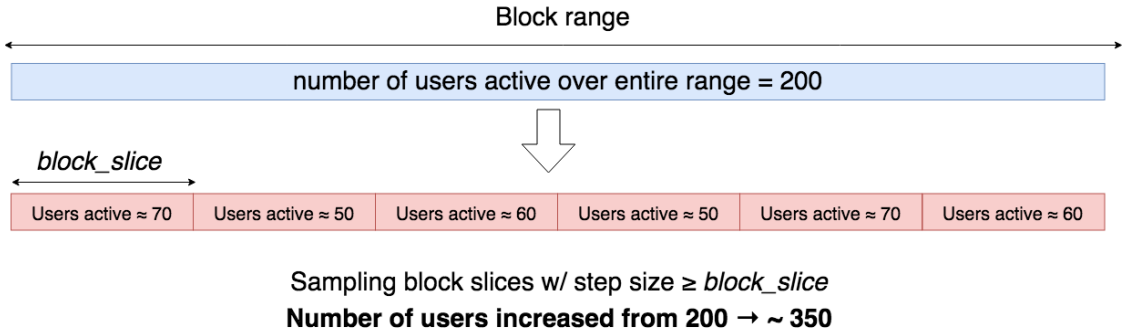


Figure 5.4: In a simplified scenario, if given only 200 active users over a block range, generating users from small, non-overlapping slices of the range aggregates many more data points.

Short term changes in user activity patterns between block slices are accounted as these change will be reflected in the feature set of observations. A disadvantage, however, is that the overarching behaviour of users over a longer time frame is lost. To resolve this, longer subsets of blocks over longer ranges of the blockchain need to be sampled.

5.1.2 Overcoming continued class imbalance

After generating many more data points using by using method described in Section 5.1.1, the following techniques were applied (with greater confidence) to fix the

persisting class imbalance problem, and are briefly discussed below:

- Undersampling
  Sample, with replacement, random data points of the majority class to match the number of samples in the minority class. This risks removing highly representative instances of the majority class, thereby discarding useful information. As shown in the results section, resulting accuracies after undersampling were relatively low.

- Oversampling
  Random samples of the minority class are replicated until there are as many samples as the majority class. This technique is implemented in `sklearn` by using the `class_weight="balanced"` parameter. Overfitting may occur as too many replications of minority class data points can bias the classifier.

- Hybrid
  By undersampling the majority class, and oversampling the minority class to a number of samples somewhere in between, a trade off between the two approaches is reached. Therefore, while this can reduce overfitting it may also leave the majority class underrepresented.

- Synthetic Minority Oversampling Technique (SMOTE) Instead of replicating existing samples of the minority class, SMOTE technique adds synthetic minority instances to the existing data. Taking n nearest neighbours to the existing data point to be replicated, a new synthetic point on the line between the existing point and a randomly chosen neighbour is created. Due to lack of time, this technique was not explored. However, since kNN does not perform well (Figure 6.2) it is possible that the euclidean distance metric is ill defined for these features. Therefore, SMOTE is unlikely to work well either.

5.2 Extracted Features of Users

To describe user activity and behaviour across blocks, features were extracted by combining the information collected during user generation (Algorithms 1 & 2), and transactional relationships between users in the user graph. The feature set contains a row of information for each user observed, while ignoring any unallocated addresses and transactions. Below is a table summarising all of the features that were extracted and used for model training.

Consider a scenario where two users of the same class were active during the same range of blocks, e.g. two gambling sites, but one of them was far more active than the other. In this case, the feature sets of both observations would be very different. The more active user would have a much greater number of transactions and larger flow of bitcoins. This was anticipated to become problematic with the feature set having high variance and noise, resulting in poor classification. To solve the issue, relevant

Feature	Description
max_sent, *min_sent*	Max and min bitcoins sent by user
max_rec, *min_rec*	Max and min bitcoins received by user
total_sent, *total_rec*	Total bitcoin sent and recieved by user
unique_rec, *unique_sent*	Num of unique addresses and users transacted with
unique_rec_adr, *unique_sent_adr*	Num of unique addresses transacted with
unique_rec_user, *unique_sent_user*	Num of unique users transacted with
num_sending_tx, *num_receiving_tx* (in/out degree)	Number of edges directed to and from user node
ratio	Ratio of number of inputs to number of outputs
is_miner	Has the user ever performed a mining transaction?
num_adr	Number of addresses owned by user
num_cadr	Number of change addresses owned by user
num_blocks_active	Number of blocks user performs transactions in

Table 5.1: Overview of features used

features in Table 5.1 were normalised over the number of addresses owned by a user and also the number of blocks a user was active in. After adding these features, accuracies were greatly improved. In total, users were described by a feature set containing 37 features.

5.3 Model Hyperparameter Tuning

There were two ways to utilise the available data for the machine learning task: 1. Train-Test split and 2. K-Fold cross validation. While the first method is K times faster and simpler to examine, cross-validation allows for an accurate estimate of out-of-sample accuracy. Data is also used more efficiently as every observation is used for both training and testing. The dataset is split into K equal partitions, with 1 partition being used as the testing set and the rest as the training set. The testing accuracy is calculated K times, each with a different fold being used for testing. An estimate of the out-of-sample accuracy for the parameter combination being used is then calculated by averaging the testing accuracies across all K folds. In this project, cross validation was implemented along with grid search to determine optimal hyperparameters for the different classification models. During grid search every parameter combination is iteratively tested. For every combination, the average accuracy result is noted and the set of parameters which perform the best out of them all are returned. The `scikit-learn` library provides the `GridSearchCV` function that does this very efficiently by utilising all CPU cores available during the search. `GridSearchCV` was used to construct models with combinations of different hyperparameters, shown in Table 5.2. 10-fold cross validation was done on each individual model.

To ensure good validation of results, data over a sequence of blocks was divided into a 70/30 split. 70% was used in grid search and cross validation to produce the best model. This model was then tested on the completely unseen remaining 30% of the data, which gave a reasonable sample size to test model performance and check for overfitting. Since PCA, KNN and SVM are sensitive to the scales of features, Z-score

K-Nearest Neighbours	Decision Tree	Random Forest
n_neighbors	criterion	n_estimators
	p	max_depth
	min_samples_leaf	max_features
	min_samples_leaf	min_samples_split
	max_leaf_nodes	min_samples_leaf
		bootstrap

Table 5.2: Parameters of each model to be tuned during grid search

normalisation was carried out on the dataset before training these classifiers.

Chapter 6

Results and Evaluation

This chapter details and reasons the results obtained during this project. It explores the data used for training, and showcases the performance of the various models tested. The best performing classifer is used to perform further analysis on a range of blocks much earlier in the blockchain. Finally, the user and address graphs are shown, along with some interesting insights that can be derived from them.

6.1 Comparing Classifiers

This section begins by taking a deeper look at the user feature set to be learned. For analysis, a random, continuous range of blocks were selected. The number of blocks needed to be small enough to keep user generation time within reasonable limits but also large enough for the model to learn varying user behaviour over time. Therefore, a range of 500 blocks was selected (block height 400000 to block height 400500), equating to approximately 83 days of network activity. Given below is the class distribution of users within this range, generated per 10 block slices.

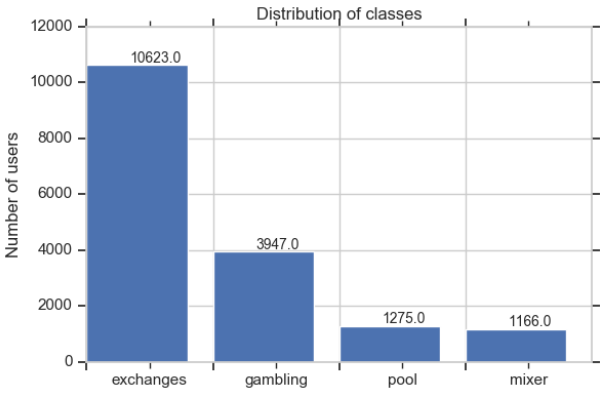


Figure 6.1: Distribution of labelled users active per 10 block slice over a 500 block range (400000 to 400500).

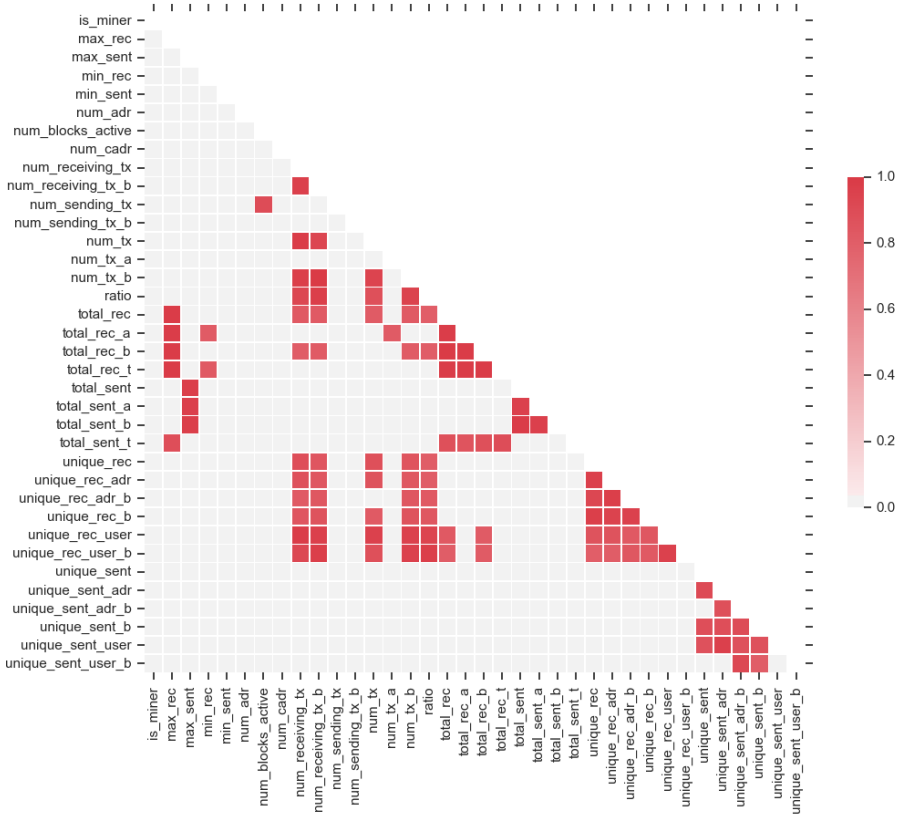


Figure 6.2: Correlation matrix of user features. Only highly correlated features with correlation values 0.8 or higher are shown.

Over the observed range, any 'unknown' or 'mixed' label users were ignored as they did not belong to the classes of interest. For each remaining labelled user, the features described in Table 5.1 were extracted. Shown below are the highly correlated features, extracted from users in the block range 400000 to 400500. To explain some of these correlations, take `max_sent` as an example. `max_sent` is highly correlated with `total_sent`. This is expected over a short 10 block slice, since a users maximum sent amount in bitcoins will likely make up most of their total amount sent in bitcoins. In order to ascertain the most relevant features for classification (hence remove unnecessary ones) the random forest feature importance metric was utilised. This describes how relevant a feature is during model construction. It was found that the additional normalising features appended to the feature set in Table 5.1 were not contributing to the model. Also, as seen from the matrix below, these features are highly correlated with many others. Some of these can be again explained using the same reasoning as before. It is expected that most of these correlations will subside if the block slice is taken over a much larger range, lending motivation to retain these correlated features in future experiments. However, for the following tests, these additional features were removed.

Given that this is a multi-class problem, the string labels of the classes were encoded under the scheme in Table 6.1. This scheme is used in all of the confusion matrices and accuracy tables that follow.

Category	Label
exchanges	0
gambling	1
mixer	2
pool	3

Table 6.1: Label encoding scheme

6.1.1 kNN Performed on Principal Components

The kNN algorithm is not influenced by the size of classes and as such the unbalanced original feature dataset was used as input. kNN is not expected to perform well as shown by the principal component analysis plot in Figure 6.4. There is significant overlap between principal components, indicating that the data is noisy, and the euclidean distance measure is not well defined for the feature vectors. This hypothesis is reflected in the results where kNN predicts quite poorly, and heavily overfits on the exchanges class.

Labels	0	1	2	3
Recall	0.95	0.02	0.00	0.02
Precision	0.71	0.27	0.00	0.25
F_1 score	0.81	0.08	0.00	0.04
Overall Accuracy	0.66			

Table 6.2: Test results from nearest neighbour classification on PCA data showing overfitting. Three out of four class perform very poorly. Even with the exchanges class, while recall accuracy is high, the precision result indicates that there are few relevant instances of services in the retrieved instances of services. This system would not be suitable for deployment.

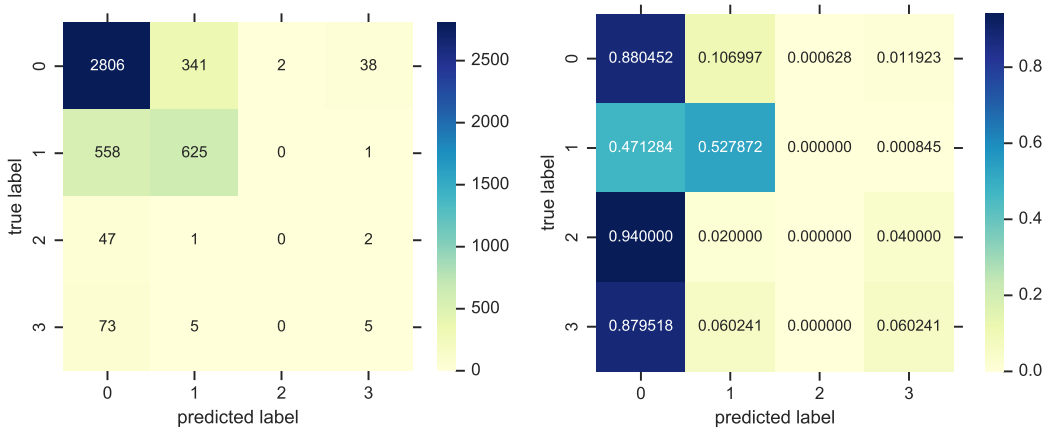


Figure 6.3: Normalised confusion matrices for k-Nearest Neighbours again clearly shows overfitting. Unable to distinguish between which service is which, the nearest neighbour classifer has assigned most labels to the class with the highest number of data points (exchanges).

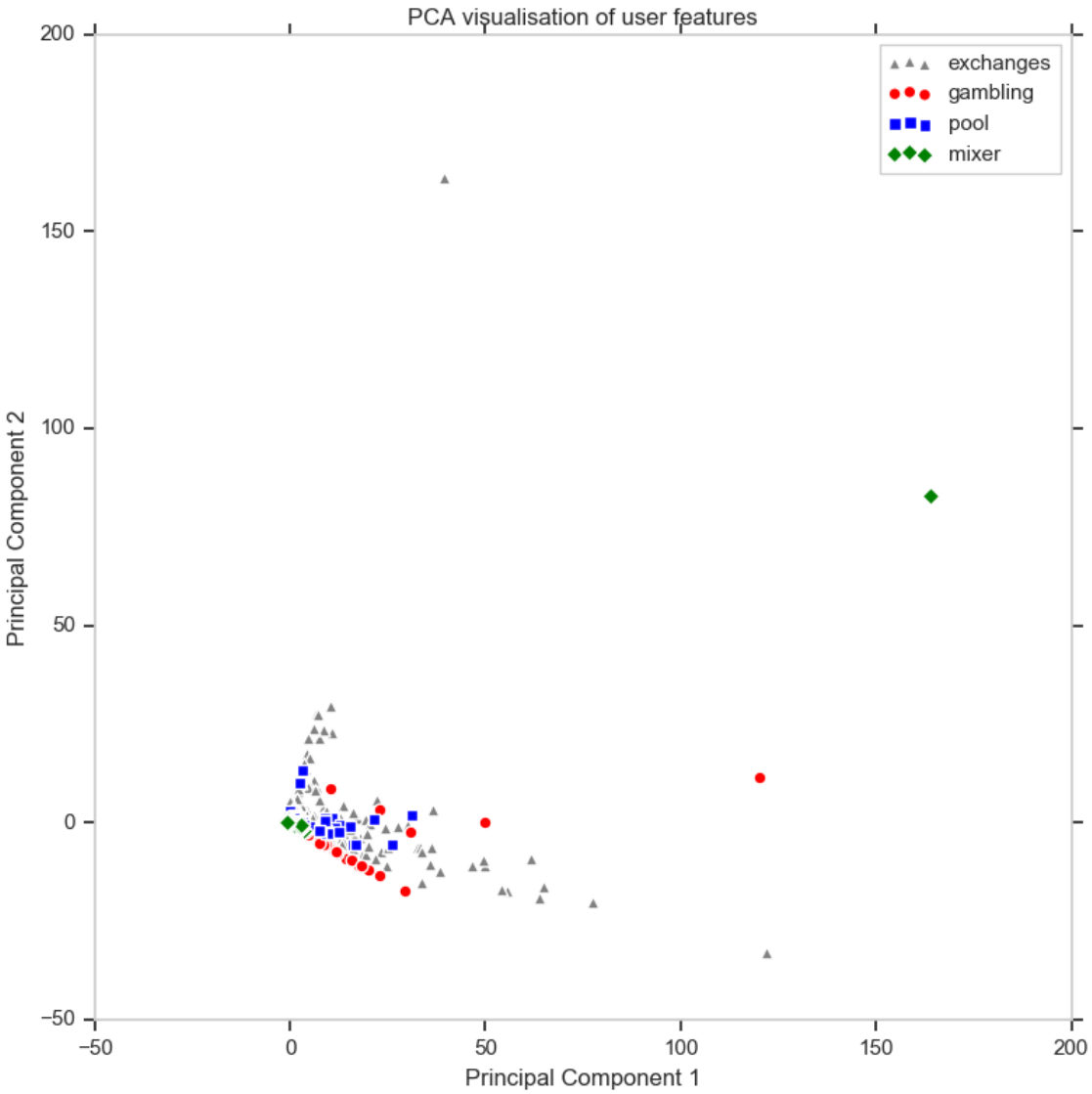


Figure 6.4: Principal Component Analysis showing significant overlap of the feature set in vector space. The lack of separability indicates that the data is noisy, and that tests in higher dimensions may perform poorly. This is confirmed by the results.

6.1.2 Decision Tree

	Upsampled				Downsampled				Hybrid			
Labels	0	1	2	3	0	1	2	3	0	1	2	3
Recall	0.88	0.81	0.54	0.83	0.62	0.76	0.82	0.88	0.82	0.86	0.50	0.93
Precision	0.92	0.72	0.38	0.85	0.90	0.61	0.07	0.32	0.94	0.65	0.32	0.82
F_1 score	0.90	0.77	0.45	0.84	0.74	0.68	0.12	0.47	0.87	0.74	0.39	0.87
Overall Accuracy	0.85				0.66				0.83			

Table 6.3: Test results for decision tree classifier. Training on the upsampled and hybrid data provide the best classification results.

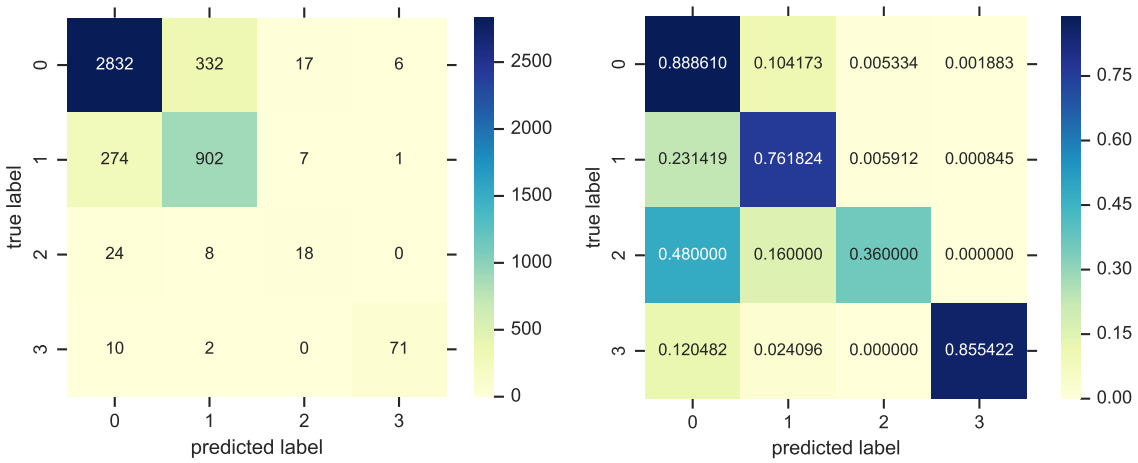


Figure 6.5: Consfusion matrix for the decision tree classifier on upsampled data. Exchanges and mining pools are clearly being classified well by the model. Mixers get wrongly classified as exchanges & gambling services quite often, and 25% of gambling samples are mistaken to be exchanges.

Focusing on the recall metric, it is evident that keeping all the data helps in retrieving the number of relevant samples of the majority class over all the samples. When downsampled, this recall value drops down from 88% from 82%. For the other classes, except mixers (reasoning described in Section 4.1.6), a noticeable increase in Recall is observed. This is because by using the downsampled data, the model has overfit less on the majority class. More weighting is now given to the other samples, resulting in better classifications for minority classes. Evidently the downsampled data returns the worst test results, with the model unable to learn the features for any of the classes. This is expected as many data points have been thrown away when downsampling from 10623 and 3947 to 1116 data points.

Decision trees can be visualised and hence are interpretable. Figure 6.6 is analysed (zooming in is required). Please note that this tree has only a max depth of 4 and is not representative of the final model, which was too complex to be visualised here. As a result, the accuracies for this model were not very high. Despite this, it is still useful to interpret how some of the classifications were made:

The root node of the tree contains all samples, with the most discriminative attribute being `max_sent`. Moving down the tree, the samples are divided based on attributes of the decision nodes. Some of the paths reaching final decision nodes are interpreted below:

1. If a test point's maximum sent amount is very low, and it has sent amounts to many users, it is most likely to be a gambling service. This is characteristic of how a blockchain gambling site would operate with many users betting small amounts.
2. If the total sent amount of a sample is greater than 105 bitcoins sent to many users/addresses, then it is classified as an exchange. This makes sense as an exchange would be expected to send large amounts to multiple unique users.
3. If a test point has more than 1 change address, it is likely to be classified as a mining pool. Since the business model of pools requires them to send bitcoins to multiple users, it is expected that they will have many change addresses.

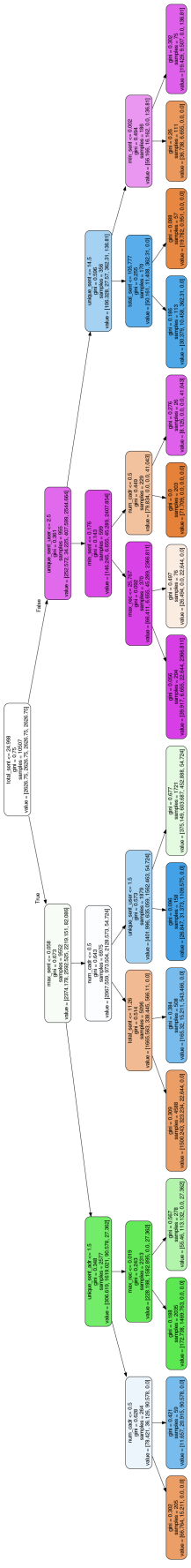


Figure 6.6: Visualisation of decision tree upto a max depth of 4. The final tree model generated during training was much larger, leaving it unsuitable for visualisation.

6.1.3 Random Forest

	Upsampled				Downsampled				Hybrid			
Labels	0	1	2	3	0	1	2	3	0	1	2	3
Recall	0.93	0.75	0.26	0.86	0.69	0.79	0.92	0.86	0.86	0.88	0.50	0.80
Precision	0.90	0.82	0.48	0.99	0.94	0.68	0.08	0.40	0.94	0.71	0.47	0.92
F_1 score	0.92	0.78	0.34	0.92	0.79	0.73	0.14	0.54	0.90	0.78	0.49	0.85
Overall Accuracy	0.88				0.72				0.86			

Table 6.4: Test results for the random forest classifier. Again, training on upsampled and hybrid data provide the best classification results.

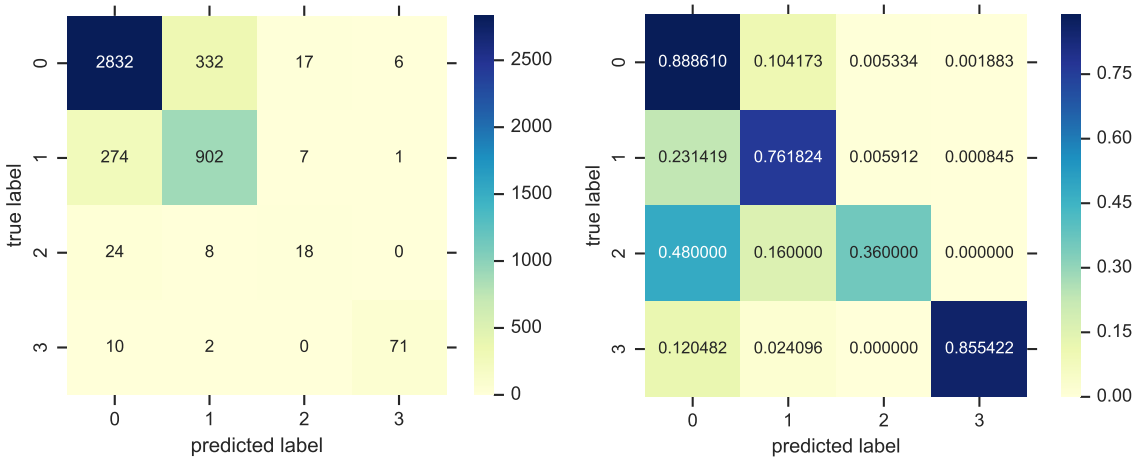


Figure 6.7: Confusion matrices for random forest on up-sampled data. Similar errors as those made by the decision tree are also here, namely gambling and tumbling services being wrongly classified as exchanges.

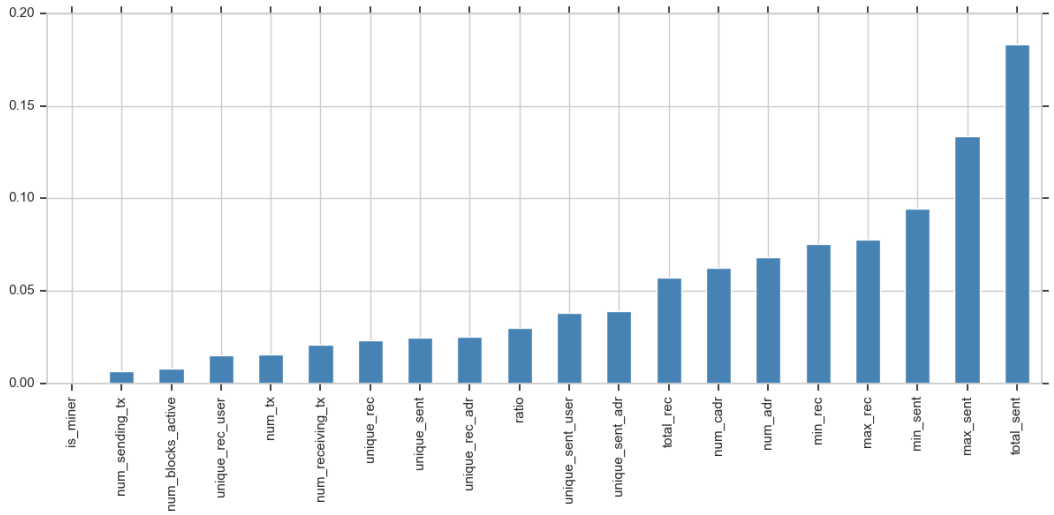


Figure 6.8: Random forest feature importance plot. Evidently, most decisions have been split on features describing spending characteristics.

6.1.4 Discussion

As explained in Section 6.1.1, the nearest neighbour classifier is unable to classify services well based on the given feature set. However, the random forest and decision tree classifiers give promising results and will be the focus of further analysis and future experiments. For both of these models, the downsampled data set gives the worst test results. By throwing away samples, the remaining data is left under-representative of the gambling and exchange classes. The best performing technique in terms of accuracy is oversampling, which is expected as there had been no instances of training data removed. However this is misleading given that recall metrics for classes other than the majority exchanges class are lower than in the hybrid model. The hybrid technique performs quite well, with results almost on par with the oversampled model. This balancing technique is deemed the best as it helps to prevent the models learning on the majority class too well and hence overfitting on them. For a heavily imbalanced dataset such as the one under observation, the hybrid dataset will produce more generalised models. Continued justification for the usage of a hybrid dataset is given in Section 6.2.1. Additionally, the fact that the accuracies for exchanges remain high, is indicative of it's remaining data being largely representative of the entire class even post downsampling.

By observing the feature importance plot in Figure 6.8, the `total_sent` feature was used most often during splitting. A possible explanation for this may be due to Heuristic 1. Because all users have been generated by collating multiple inputs to a transaction, they have all participated in transferring at least some amount of bitcoin. As such, users will always have data for features relating to spending characteristics. Alternatively it is possible, and even likely for a user to not receive any amount at all in a short block slice under test. Therefore, it is reasonable that tree based learners will use features relating to spending characteristics for many of the classifications.

In all cases, the 'mixer' class has been largely misclassified. Reasons for this can be two-fold: The first and most obvious conclusion is that there simply isn't enough data points in the feature set to fully learn the characteristics of a mixing service. Because the `walletexplorer` dataset was limited for this class, this is a likely outcome. However, another more interesting reason may be due to how the address clustering heuristics work. As explained in Section 4.3, the heuristics fail for services that allow multi-input services. Mixers are well known examples of this, and evidently the user heuristics have been unable to correctly cluster addresses belonging to these services. The hypothesis that all input private keys to a transaction belong to the same user fails here as, by design, a tumbling services is a utility that aggregates bitcoins from multiple users to obscure traceability. Fixes to the heuristics to accommodate for this are suggested in Section 7.2.1.

6.1.5 Limitations

The machine learning models employed in the predictive classification of users were trained only on 4 different known categories of Bitcoin services. Therefore, if the current system was deployed in a real-time, anonymised setting, it will look to always place a new user into one of these 4 classes. Evidently, the Bitcoin network will consist of many more different types of users than those described in these 4 classes. However, the predictive power of the model is given confidence here, as it has correctly distinguished between the existing categories themselves, with a high average recall of 80%. This strongly indicates that if more classes were used in training, the decision tree and random forest models would be able to learn the varying characteristic behaviours of these other categories of users, using the feature set that was formulated in this study.

6.2 Determining Changes in User Behaviour

To determine whether or not the behaviour of services had changed over time, it was decided to test a trained model on users generated much earlier on in the blockchain. As such, the random forest model was trained on block slices of length 10 over 1500 blocks (block height 400000 to 401500). This model was then used to test for users over blocks 200000 to 200500. Shown below are the confusion matrices from these tests:

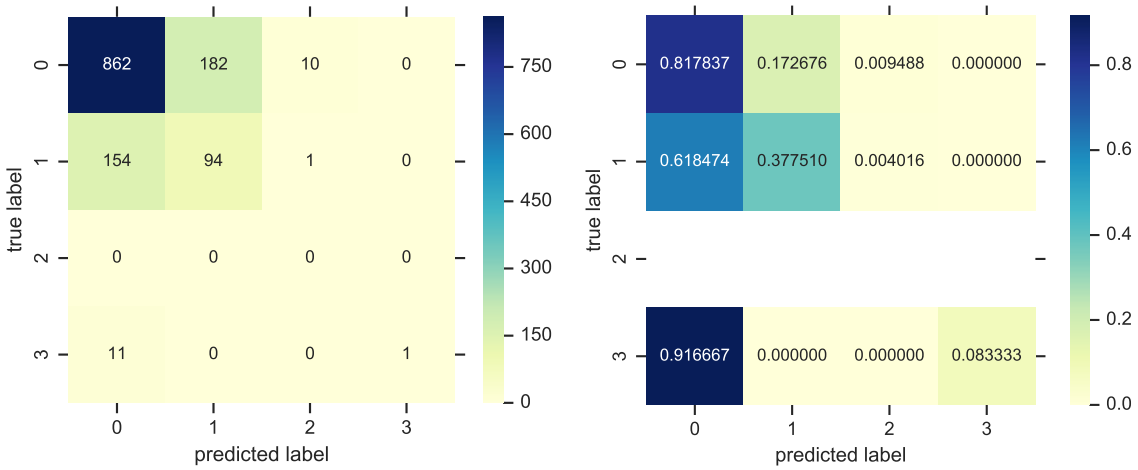


Figure 6.9: Testing a random forest model much earlier on in the blockchain.

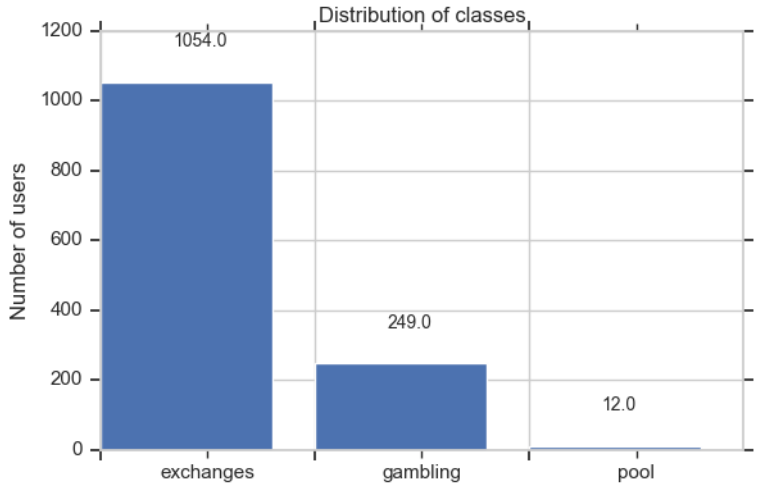


Figure 6.10: Distribution of labelled users over blocks 200000 to 200500.

Evaluation

These results reveal the following insights:

- There seems to be a change in the activities of services over time since prediction results are very poor. Whilst exchange services retain a high accuracy, this is indicative of the model overfitting due to a heavily imbalanced dataset. It has not been able to generalise well when looking for services in blocks too distant from the block range used for training. This would mean that in a real time implementation of the system, quick training would be required over an immediate number of previous blocks. This gives merit to the use of a dataset that is a hybrid of upsampling and downsampling, which would require less data & storage, while also training models quicker (if class imbalance exists).
- Figure 6.10 describes the distribution of labelled services active over the test range. Clearly seen is the reduced activity of users when compared to Figure 6.1. Hence as expected, along with changed behaviour, there are fewer services active earlier on in Bitcoin's history. The blocks under test would have been mined around September of 2012, during the nascent years of Bitcoin.
- One possibility is that `walletexplorer.com` could have very limited tags for services active in 2012. Evidently, `walletexplorer.com` is quite limiting on its own considering the entire network. Therefore, there is a need for more labels to be collected by aggregating from off-chain sources.

6.3 Insights Derived from User Graphs

Before commencing further in this section, it is important to understand the types of graphs presented along with their colour schemes. 4 styles of graphs were deemed

necessary to extract useful information from visualisations. These can be placed in two categories:

1. **Service annotated** address and user graphs are annotated and coloured based on relevant services tagged from `walletexplorer.com`

Common to both	Exchanges	Gambling	Pool	Mixer
Address graph specific	OTC			
User graph specific	Unlabelled User	Labelled Address		

Figure 6.11: Colouring scheme of nodes in the service annotated address and user graphs. Please note that all edges are white in colour, and any nodes not described above are coloured in white. 'Common to both' describes nodes that share a similar tag in both graphs, i.e. A blue colour node will denote an exchange address in the address graph and an exchange user in the user graph.

2. **"Top 5"** address & user graphs have coloured nodes and edges for only the top 5 users owning the most addresses.

Top 5 specific	Big user 1	Big user 2	Big user 3	Big user 4	Big user 5	OTC

Figure 6.12: Colouring scheme for "top 5" address and user graphs. OTC nodes here only correspond to one time change addresses belonging to any one of the top 5 users.

A lot of testing for the user and address graphs was done using information from block 400000. The graphs were layed out with Force Atlas 2, and are showcased below.

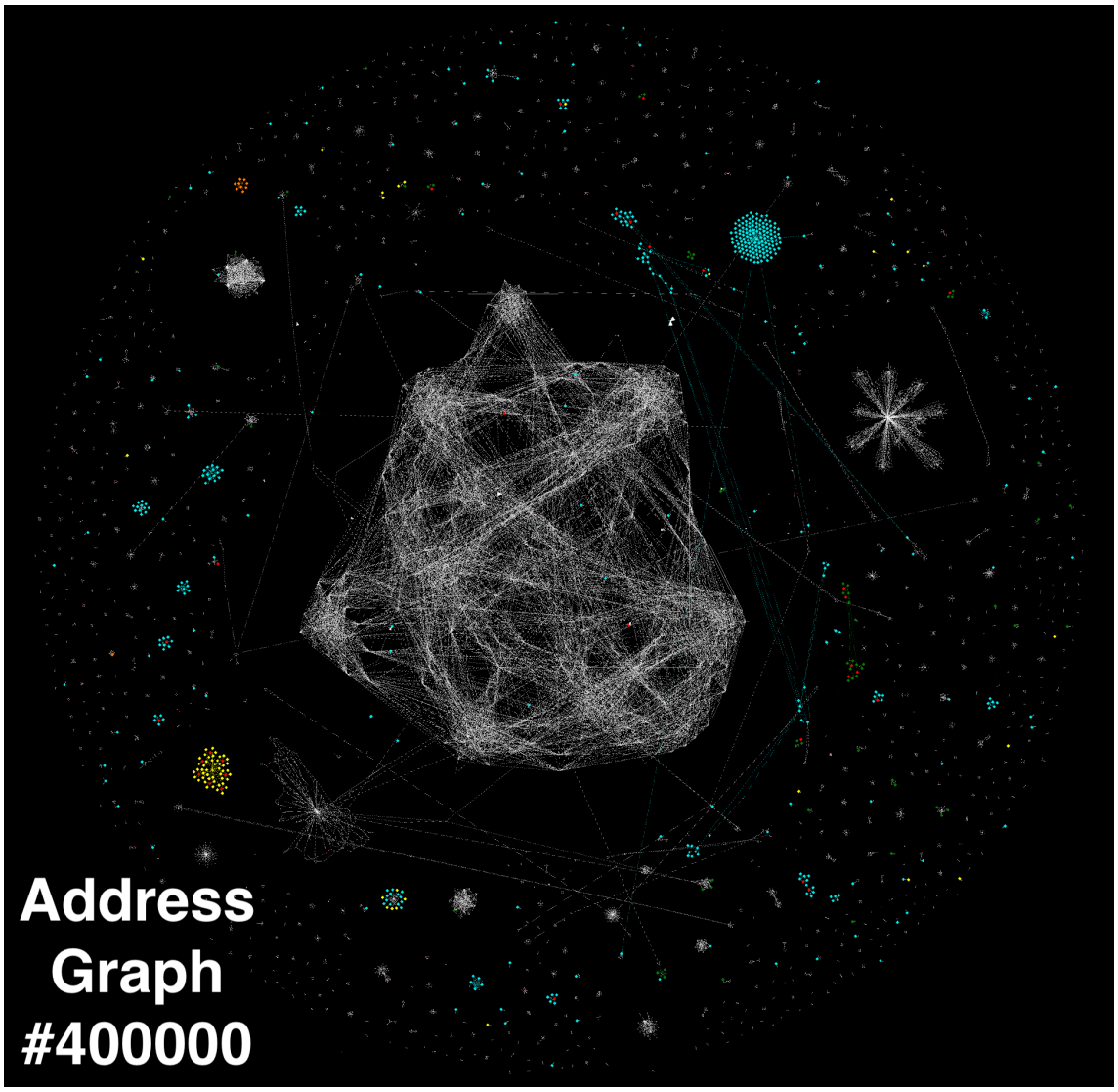


Figure 6.13: Zoomed out service annotated address graph of block 400000, with colours corresponding to tagged services. Remaining white nodes represent untagged addresses. Annotated labels are not shown here but can be easily toggled on.

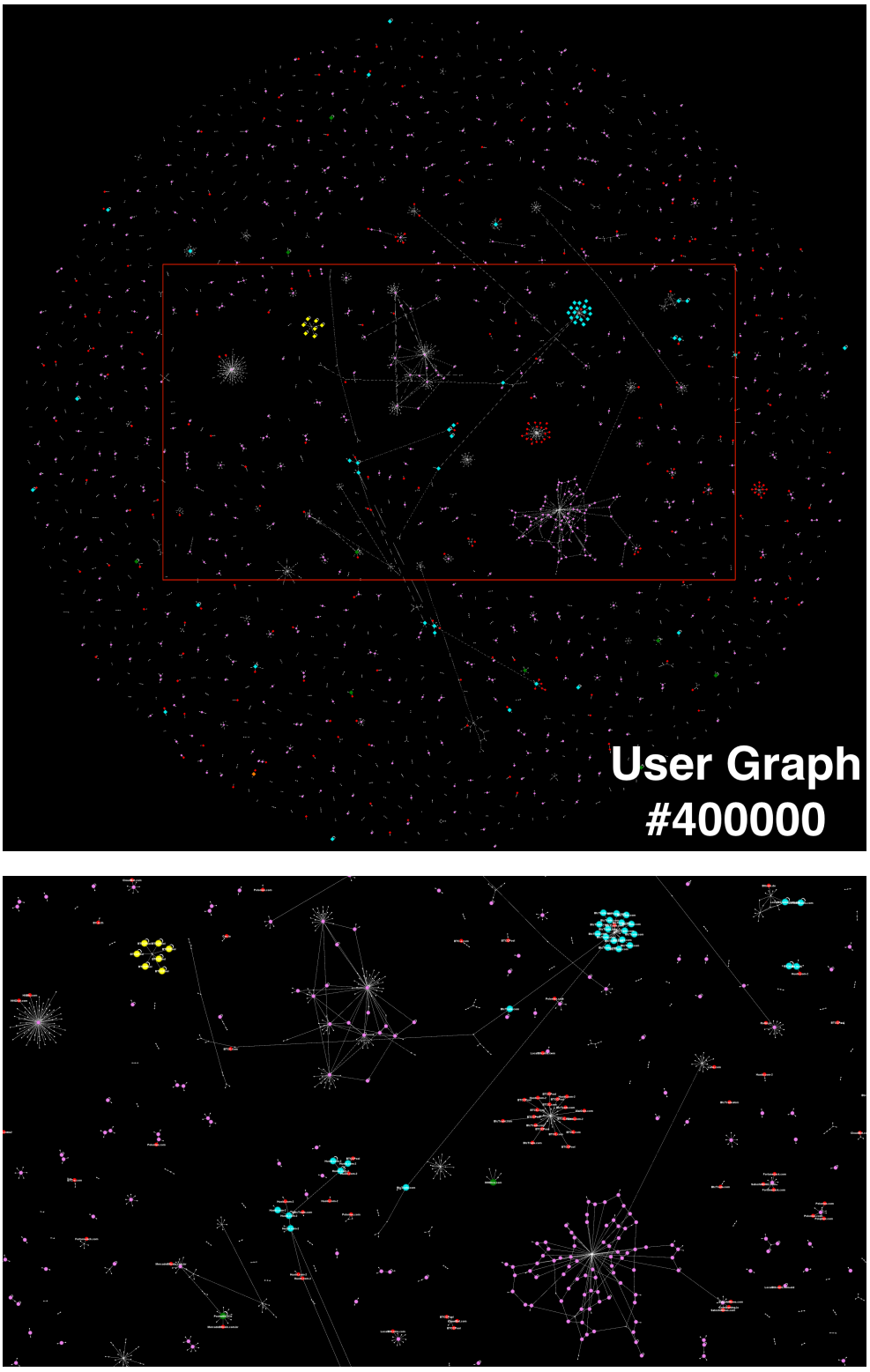


Figure 6.14: Service annotated user graph for block 400000. Image below zooms in on the middle section of the user graph. Colours correspond to tagged services (See first column of Figure 6.11). **Violet** nodes represent users that aren't tagged, **red** nodes represent tagged addresses (which did not get clustered, and therefore are most likely to be addresses of services used only for receiving bitcoins), and the remaining **white** nodes represent untagged addresses.

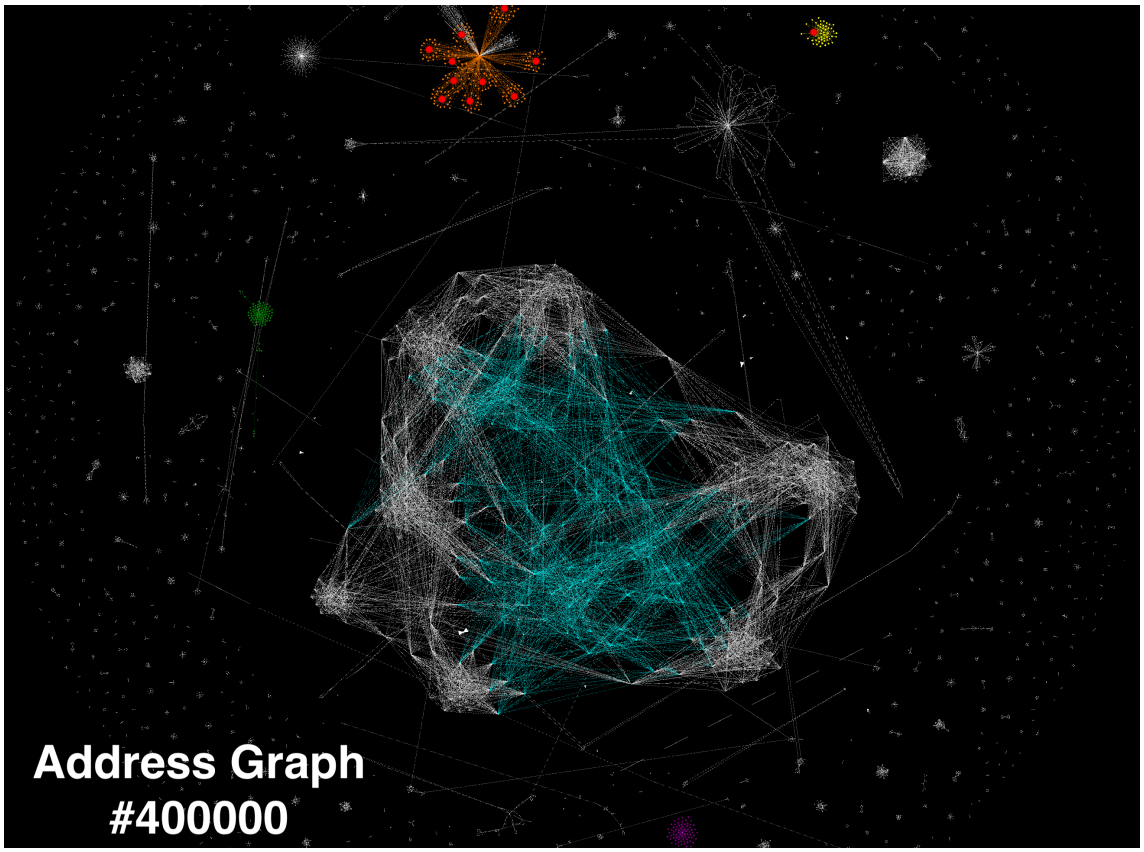


Figure 6.15: Slightly zoomed-in screenshot of the ”Top 5” address graph, showing the 5 most populous address clusters generated from heuristics 1 and 2 in block 400000. This figure has the same structure as Figure 6.13, but shows different information.

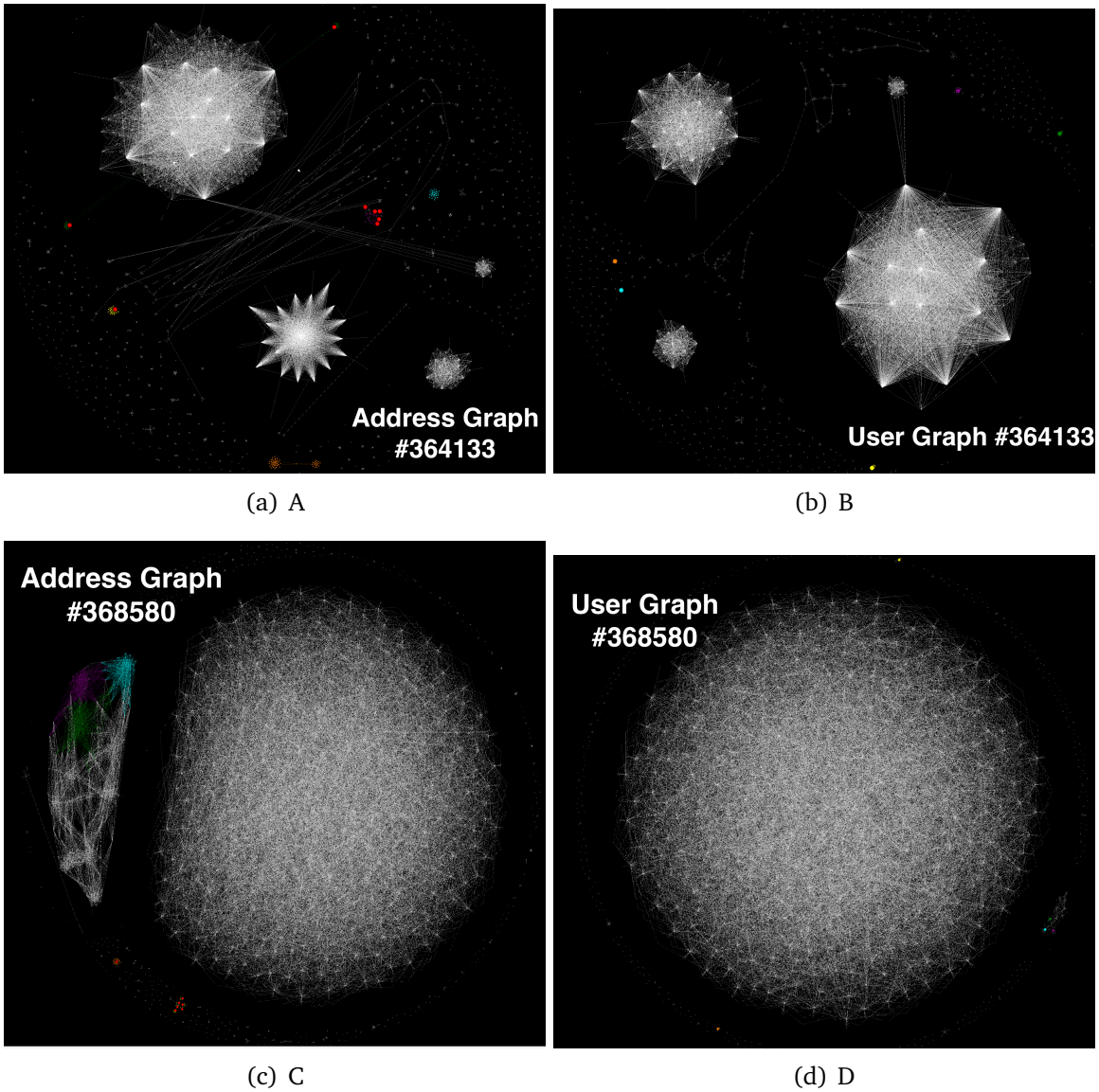
6.3.1 Discussion

The address and user graphs give an insightful view into the activity occurring in block 400000. Focus is drawn to the address graph first. Looking at Figure 6.13, a lot of the transactions occur involving exchange addresses. This is expected, since the majority of available labels were for these services but also because exchanges are widely used on the network. Given that bitcoin is still not a universally adapted means of payment, users on the network should be expected to quite often use these services to switch to fiat currencies, ethereum and altcoins. Addresses belonging to particular services have been clustered together by the layout algorithm due to their connected nature (seen visually in the blue and yellow circular graphs in Figure 6.13).

The user graph of block 400000 helps to significantly condense and simplify the address graph. The massive cluster of addresses has vanished and different interactions between actors on the network become clearer. For example, the supercluster of addresses at the centre of Figure 6.13 have a minimal number of labels. However, by clustering these addresses based on heuristics 1 and 2, a user and their interac-

tions definitively emerge (blue coloured nodes and edges in Figure 6.15). This lends further confidence to the use of the clustering heuristics as they were, in a way, able to deanonymise a user out of a very ambiguous and uninterpretable cluster of addresses.

6.3.2 Visualising Attacks on the Network



(a) A (b) B

(c) C (d) D

Figure 6.16: ”Top 5” Address and User graph representations of the ”parasitic worm” transaction rate attacks, and density-based ”tumor” attacks described by McGinn et al. [26].

6.3.3 Discussion

During the summer of 2015, DSI researchers noticed sustained attacks on the bitcoin network [26]. These attacks stemmed from the long-running dispute over the block size limit. To this day, the size of a block is limited to the arbitrary amount of 1MB. Originally put in place to limit denial of service attacks, the size limit has prevented the system in scaling to allow for higher transactions rates. Community members believe that if bitcoin is to become competitive against the likes of Paypal and Visa, this limit needs to be lifted. Because of this, in 2015, unknown users of the network generated insignificant spamming transactions to artificially increase throughput.

A and B (Figure 6.16), show the first attack taking place. Evidenced by the coloured addresses, there are still normal users transacting on the block. Indeed, the service annotated version of this graph (not shown) shows many active services. However, two massive superclusters of addresses also emerge. These addresses are heavily clustered together due to high frequency algorithmic divisions of bitcoins that are being sent to the same address in small increments. Interestingly, the user graphs retain these clusters. The reasoning for this is that the high frequency transactions would have been occurring with different singular input addresses each time. Therefore, by Heuristic 1, these are ignored and cannot be clustered. Figures C and D show quite a resemblance to the ”tumour” visualisation shown in the previous transaction graph of block 368580 [26]. This is because the second attack was much more data intensive and hence makes up most of the population of address in block 368580. Again, the user graph retains the massive superclustering of attacking addresses, as they have not been clustered into users. However, the left segment of Figure C shows that users were detected by the clustering framework, and as such this second cluster has been condensed in the user graph (Figure D). This is reaffirmed by the separation of these clusters by ForceAtlas2, indicating different behaviours of the different parties owning the addresses (Figure C).

6.3.4 Survey

In order to evaluate the effectiveness of the visualisations, fellow students in the computing department were asked to interpret the graphs and give feedback. This provided valuable assessment of the visualisations and helped focus attention on what was missing from them. Many of the students described how the address graph looked strikingly similar to an image of the universe. As such it drew their attention and increased interest in this space. One essential early piece of feedback was the lack of a link between the address and user graphs. Taking this into account, the graphs were regenerated with the colour schemes as shown in previous Figures. People knowledgeable in the blockchain space were more interested in seeing the interactions of the actual services themselves on the network. With this in mind, the address and user nodes of exchange, gambling, mixer and mining pool services were slightly enlarged and highlighted.

Chapter 7

Conclusion

7.1 Summary of Achievements

This dissertation addresses the task of classifying user activity on the Bitcoin blockchain in a quantitative manner. As such, this project has been successful in providing a more complete representation of the network by combing top down and bottom up approaches.

The infrastructure developed to cluster addresses is versatile and can cluster for any block and also over multiple, synchronous, asynchronous blocks. Given a block range and any starting address, relevant transactions are traversed to infer clusters of ownership. The framework generates features that can evolve with the number of blocks under observation, hence being more flexible when compared to features extracted from just individual transactions or addresses. This allows user activity to be characterised in a much more encompassing manner.

Taking advantage of this, machine learning models were applied and tested to detect for commercial services on the blockchain. By training and testing decision tree and random forest classifiers over an 83 day period, services were detected with high recall scores (averaging at 80%). To an extent, the system has been successful in deanonymising user activity. Additionally, it was shown that the activities of users has changed as the network has grown, with services behaving differently and being more active than before.

The visualisation framework generating address and user graphs take a top down approach to observing activity over an entire block. By appending these graphs with annotations from off-chain sources and data from the address clusters of users, insights into interesting activity on the network were visually derived. The user graphs made it possible to alleviate a lot of the noise present in the address graph, and helped extract meaningful interactions between real world entities. Combined with the data observatory, these visualisations enabled further analysis on emergent patterns and interesting addresses, along with collaborative viewings of notable activity.

7.2 Future Work

Suggested areas for future work involve improving the robustness of the user generation algorithms, gathering more data for training and applying newer technologies that merge graph theory and machine learning.

7.2.1 Additional Heuristics

As demonstrated earlier, while the user heuristics were largely successful in clustering addresses belonging to an entity, the following updates look to adapt the heuristics to the Bitcoin network of more recent times:

1. If the number of input addresses matches the number of output addresses, then the transaction is likely being carried out by a shared send mixer. Mixers try to match inputs having similar values.
2. Consumer wallets only ever create two outputs from a transaction. Hence, if an UTXO from a transaction having three outputs is spent, it is unlikely to be change. [14]
3. Commercial wallet software are assumed not to spent UTXOs unnecessarily. Therefore, a unique output is likely to be change if it has a smaller value than any of the other outputs of the transaction. [14]
4. Certain wallets types can be traceable if they have distinct features during transactions. For example, multi-signature wallets use P2SH ('Pay To Script Hash' is logic that checks for multiple signatures before accepting a transaction) for change during a transfer, whereas the payee is unlikely to. Therefore, a highly accurate estimate of the change amount can be calculated.

7.2.2 Graph Metrics and Isomorphisms

One aspect that can be explored for future work is the addition of more graph-centric features. Useful information can be extracted by looking at features such as page rank, centrality, shortest path and betweeness of nodes. One idea not explored due to time constraints aimed to learn the types of graphs that neighboured known users. From this, not only would user behaviour be learned, knowledge on other users and addresses they interact with would also be obtained. The emerging field of graph convolutional neural networks is promising for this space. Several methods of graph embeddings have been proposed in literature which can be used to identify isomorphism in graphs. These methods try to exploit features like neighbourhood, pagerank and centrality measures to convert a node into a vector. The vector then represents node's behaviour. Few of the most popular approaches to graph embeddings are:

1. Locally linear embeddings : This method postulates that a node in embedding space can be represented as a weighted linear combination of its neighbours. By solving the corresponding optimisation problem over the whole graph, embeddings can be obtained [18].
2. Deepwalk: This method uses the behaviour of stationary probabilities of a node in several random walks and performs optimisation over sum of log-likelihoods for each random walk. From this it can generate a vector representation of the nodes [18].
3. Node2Vec: Very similar to Deepwalk but uses a trade off between breadth-first search and depth-first search to preserve community structure [18].

7.2.3 Improving the Current System

As mentioned previously, in order to enable the machine learning models to generalise over changing user behaviour, they need to be fed more data. To enable this, the following extensions are suggested:

1. A custom real-time address graph database, which is updated with transactions and their addresses every time a new block is mined. Graph databases perform well in handling queries over connected data, even with a large number of relationship traversals. This would greatly speed up the clustering algorithms.
2. More data collected by scraping off-chain sources, similar to the work done by Bitfury researchers [16].
3. Use the observatory to single out interesting entities and their graph structure. In a semi-supervised approach (such as crowd-sourced labelling via an online application), difficult to distinguish structures can be labelled thereby increasing the variety of training data.

Bibliography

[1] About blockseer. https://www.blockseer.com/about. Online; accessed: 2018-09-04. pages 13

[2] Chainalysis product description. https://www.chainalysis.com/. Online; accessed: 2018-09-04. pages 13

[3] Data observatory technical specification. https://www.imperial.ac.uk/data-science/data-observatory/technical-specifications/. Online; accessed: 2018-09-02. pages 24

[4] Decision trees. http://scikit-learn.org/stable/modules/tree.html. Online; accessed: 2018-09-02. pages 15

[5] Google trends search result for blockchain. https://trends.google.com/trends/explore?q=blockchain. Online; accessed: 2018-08-12. pages 5

[6] Graph-tool performance comparison. https://graph-tool.skewed.de/performance. Online; accessed: 2018-09-04. pages 23

[7] Hashrate distribution an estimation of hashrate distribution amongst the largest mining pools. https://www.blockchain.com/en/pools. Online; accessed: 2018-09-02. pages 8

[8] Nearest neighbors. http://scikit-learn.org/stable/modules/neighbors.html. Online; accessed: 2018-09-02. pages 16

[9] Py2neo graph databases. https://py2neo.org/v4/database.html?highlight=pandas. Online; accessed: 2018-09-02. pages 20

[10] Elli Androulaki, Ghassan O Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. Evaluating user privacy in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 34–51. Springer, 2013. pages 18, 27

[11] Andreas M Antonopoulos. *Mastering Bitcoin: unlocking digital cryptocurrencies*. "O'Reilly Media, Inc.", 2014. pages 7

[12] Khaled Baqer, Danny Yuxing Huang, Damon McCoy, and Nicholas Weaver. Stressing out: Bitcoin stress testing. In *International Conference on Financial Cryptography and Data Security*, pages 3–18. Springer, 2016. pages 18

[13] Mathieu Bastian, Sebastien Heymann, Mathieu Jacomy, et al. Gephi: an open source software for exploring and manipulating networks. *Icwsm*, 8(2009):361–362, 2009. pages 24

[14] Jonas David Nick. Data-driven de-anonymization in bitcoin. Master's thesis, ETH Zurich, 2015. pages 61

[15] Giuseppe Di Battista, Valentino Di Donato, Maurizio Patrignani, Maurizio Pizzonia, Vincenzo Roselli, and Roberto Tamassia. Bitconeview: visualization of flows in the bitcoin transaction graph. In *Visualization for Cyber Security (VizSec), 2015 IEEE Symposium on*, pages 1–8. IEEE, 2015. pages 12

[16] Dmitry Ermilov, Maxim Panov, and Yury Yanovich. Automatic bitcoin address clustering. In *Machine Learning and Applications (ICMLA), 2017 16th IEEE International Conference on*, pages 461–466. IEEE, 2017. pages 18, 62

[17] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016. pages 17

[18] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018. pages 62

[19] Stan Higgins. The irs has been using bitcoin tracking software since 2015. `https://www.coindesk.com/irs-using-bitcoin-tracking-software-since-2015/`. Online; accessed: 2018-09-02. pages 13

[20] Jason Hirshman, Yifei Huang, and Stephen Macke. Unsupervised approaches to detecting anomalous behavior in the bitcoin transaction network. Technical report, Technical report, Stanford University, 2013. pages 3

[21] Mathieu Jacomy, Tommaso Venturini, Sebastien Heymann, and Mathieu Bastian. Forceatlas2, a continuous graph layout algorithm for handy network visualization designed for the gephi software. *PloS one*, 9(6):e98679, 2014. pages 11

[22] Chuck Jones. Blaming mt. gox for bitcoin's recent price drop just doesn't compute. `https://www.forbes.com/sites/chuckjones/2018/03/12/blaming-mt-gox-for-bitcoins-recent-price-drop-just-doesnt-compute/#5bf0caf7d615`. Online; accessed: 2018-09-02. pages 9

[23] Christoph Kinkeldey, Jean-Daniel Fekete, and Petra Isenberg. Bitconduite: Visualizing and analyzing activity on the bitcoin network. In *EuroVis 2017-Eurographics Conference on Visualization, Posters Track*, page 3, 2017. pages 13

[24] Stephen Kobourov. Force-directed drawing algorithms. In *Handbook of Graph Drawing and Visualization*, pages 383–408. 2013. pages 10

[25] D McGinn, D McIlwraith, and Y Guo. Toward open data blockchain analytics: A bitcoin perspective. *arXiv preprint arXiv:1802.07523*, 2018. pages 14, 19

[26] Dan McGinn, David Birch, David Akroyd, Miguel Molina-Solana, Yike Guo, and William J Knottenbelt. Visualizing dynamic bitcoin transaction patterns. *Big data*, 4(2):109–119, 2016. pages 2, 3, 4, 11, 58, 59

[27] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. A fistful of bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 127–140. ACM, 2013. pages 18, 27, 29, 31

[28] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008. pages 6

[29] Thai Pham and Steven Lee. Anomaly detection in bitcoin network using unsupervised learning methods. *arXiv preprint arXiv:1611.03941*, 2016. pages 3, 18

[30] Kenneth Rapoza. Goldman sachs caves: Bitcoin is money. `https://www.forbes.com/sites/kenrapoza/2018/01/10/goldman-sachs-caves-bitcoin-is-money/#78ee2e6c74b7`. Online; accessed: 2018-09-04. pages i

[31] Sebastian Raschka. An introduction to parallel programming using python's multiprocessing module. `https://sebastianraschka.com/Articles/2014_multiprocessing.html`, 2014. Online; accessed: 2018-09-02. pages 35

[32] Suryansh Rastogi. Bitcoinviz, a set of visual tools for bitcoin transactions. Master's thesis, Imperial College London, 2016. pages 11

[33] Ken Shirrif. Bitcoin the hard way: the algorithms, protocols, and bytes. 2016. pages 6, 7

[34] Kyle Torpey. Bitcoin laundering study: Where do criminals turn to mask illicit cryptoassets? `https://bitcoinmagazine.com/articles/bitcoin-laundering-study-where-do-criminals-turn-mask-illicit-cryptoassets/`. Online; accessed: 2018-09-02. pages 9

[35] Jordan Tuwiner. Bitcoin mining pools. `https://www.buybitcoinworldwide.com/mining/pools/`. Online; accessed: 2018-09-02. pages 9

Ethics Checklist

SECTION 1: HUMAN EMBRYOS/FOETUSES	
Does your project involve Human Embryonic Stem Cells?	No
Does your project involve the use of human embryos?	No
Does your project involve the use of human foetal tissues / cells?	No
SECTION 2: HUMANS	
Does your project involve human participants?	No
SECTION 3: HUMAN CELLS / TISSUES	
Does your project involve human cells or tissues? (Other than from Human Embryos/Foetuses i.e. Section 1)?	No
SECTION 4: PROTECTION OF PERSONAL DATA	
Does your project involve personal data collection and/or processing?	No
Does it involve the collection and/or processing of sensitive personal data (e.g. health, sexual lifestyle, ethnicity, political opinion, religious or philosophical conviction)?	No
Does it involve processing of genetic information?	No
Does it involve tracking or observation of participants? It should be noted that this issue is not limited to surveillance or localization data. It also applies to Wan data such as IP address, MACs, cookies etc.	No
Does your project involve further processing of previously collected personal data (secondary use)? For example Does your project involve merging existing data sets?	Yes
SECTION 5: ANIMALS	
Does your project involve animals?	No
SECTION 6: DEVELOPING COUNTRIES	
Does your project involve developing countries?	No
If your project involves low and/or lower-middle income countries, are any benefit-sharing actions planned?	No
Could the situation in the country put the individuals taking part in the project at risk?	No
SECTION 7: ENVIRONMENTAL PROTECTION AND SAFETY	
Does your project involve the use of elements that may cause harm to the environment, animals or plants?	No

Does your project deal with endangered fauna and/or flora /protected areas?	No
Does your project involve the use of elements that may cause harm to humans, including project staff?	No
Does your project involve other harmful materials or equipment, e.g. high-powered laser systems?	No
SECTION 8: DUAL USE	
Does your project have the potential for military applications?	No
Does your project have an exclusive civilian application focus?	No
Will your project use or produce goods or information that will require export licenses in accordance with legislation on dual use items?	No
Does your project affect current standards in military ethics e.g., global ban on weapons of mass destruction, issues of proportionality, discrimination of combatants and accountability in drone and autonomous robotics developments, incendiary or laser weapons?	No
SECTION 9: MISUSE	
Does your project have the potential for malevolent/criminal/terrorist abuse?	No
Does your project involve information on/or the use of biological-, chemical-, nuclear/radiological-security sensitive materials and explosives, and means of their delivery?	No
Does your project involve the development of technologies or the creation of information that could have severe negative impacts on human rights standards (e.g. privacy, stigmatization, discrimination), if misapplied?	No
Does your project have the potential for terrorist or criminal abuse e.g. infrastructural vulnerability studies, cybersecurity related project?	No
SECTION 10: LEGAL ISSUES	
Will your project use or produce software for which there are copyright licensing implications?	No
Will your project use or produce goods or information for which there are data protection, or other legal implications?	No
SECTION 11: OTHER ETHICS ISSUES	
Are there any other ethics issues that should be taken into consideration?	No

Project involves working with the publicly available dataset of the bitcoin blockchain.