Classification of Bitcoin transactions based on supervised machine learning and transaction network metrics.

Archie Norman
Department of Computer Science
University College London
Gower Street, London, WC1E
archie.norman.15@ucl.ac.uk

This report is submitted as part requirement for the MSc in Web Science Big Data Analytics at University College London. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged.

Abstract

The Bitcoin currency is a publicly available, transparent, large scale network in which every single transaction can be analysed. Multiple tools are used to extract binary information, pre-process data and train machine learning models from the decentralised blockchain. As Bitcoin popularity increases both with consumers and businesses alike, this paper looks at the threat to privacy faced by users through commercial adoption by deriving user attributes, transaction properties and inherent idioms of the network. We define the Bitcoin network protocol, describe heuristics for clustering, mine the web for publicly available user information and finally train supervised learning models. We show that two machine learning algorithms perform successfully in clustering the Bitcoin transactions based on only graphical metrics measured from the transaction network. The Logistic Regression algorithm achieves an F1 score of 0.731 and the Support Vector Machines achieves an F1 score of 0.727. This work demonstrates the value of machine learning and network analysis for business intelligence; on the other hand it also reveals the potential threats to user privacy.

Contents

List of Figures

List of Tables

1 Introduction

We first introduce this project by discussing the purpose of the research, how the changing landscape of the Bitcoin network means that questions need to be asked around the motives of commercial entities within the network and how they may conflict with that of the consumers. We show the networks increasing popularity and show a similar use case of web analytics, the privacy issues around it and how this can be applied to the Bitcoin network. We then introduce the core concepts within this field by defining the blockchain protocol, characteristics of transactions within the network and the mining process to generate Bitcoins. Recent work is discussed and we show the evolution of the network by identifying two main trading phases over the course of the Bitcoin lifetime. We investigate work carried out by [9] examining the relationship between the network structure and the price of a Bitcoin.

Classification metrics are then defined which are then used later in the paper to evaluate the supervised machine learning models we create. We focus on recall, precision, accuracy, F1 score, Receiver Operating Characteristic (ROC Curve) and log loss as suitable techniques to determine the success of our model.

We then present a section discussing recent work in this field and describe the decisions made over the course of this project based on other works. In particular we look at unsupervised learning techniques to detect spam transactions in the network, which inspired the decision to create a ground truth data-set. We also look at price prediction techniques, which inspired the decision to focus on network metrics even more so that macro events.

The project plan defines the problem we are investigating, the objectives of the research and the steps to carry out the work. The methodology describes the setup of the Bitcoin client, how we acquire the data-set and the pre-processing carried out to make the data suitable as input to the machine learning algorithms. Steps to configure the tools are presented with instructions to install GraphLab Create and to activate the working environment. We present the Logistic Regression and Support Vector Machine algorithms and apply them to our pre-processed data-set.

The results section shows the successful implementation of the project by analysing the network metrics and using these results as pre-computed input to our algorithms. We return the results of the trained models and show each the results over each iteration on the models learning phase. We evaluate the results presented by applying the evaluation metrics defined in Section 1, resulting in an F1 score of 0.731 and the Support Vector Machines achieves an F1 score of 0.727. We perform feature sensitivity analysis and tweak the models accordingly to achieve the highest possible scores.

Finally we discuss the project in its entirety by looking at the goals we set out to achieve. The results of our low level implementation are successful in that we show techniques can be applied to provide commercial entities within the Bitcoin network with tools to cluster addresses. As such we show that this commercial gain must come at the cost of the consumer who would likely forgo a certain amount on anonymity. The complexity of the project is then outlined by presenting the challenges faced, these are mostly surrounded with the size of the data-set and the lack of research in the area of clustering transactions using supervised learning techniques. The main limitation of this project is centered around user behaviour, we know from previous research that users tend to re-use addresses, however if a new address were to be generated for every transaction then we would likely see a lot of noise within the models training data. We propose potential contributions to the field in the technique used to conduct this research and finally present a view of future work on this project. The main future improvement would involve building an api to return pre-calculated address statistics and to develop the machine learning algorithms into an online-learning environment.

2 Why is this project interesting?

The Bitcoin system was proposed in 2008 by Satoshi Nakamoto and released in January 2009 [14]. Bitcoin is a decentralised transaction system, its purpose is to allow its users to transfer fungible value to one another. The most important feature of Bitcoin is its anonymity. In a similar fashion to cash, Bitcoin does not specify the sender or the receiver of the transaction rather users are able to send and receive the currency through public keys which are then cryptographically signed with private keys without revealing the identity of either participant [21]. Other notable advantages include; a lack of taxes, no transaction costs, no charge-backs and no third party seizures. The cost of one Bitcoin has now settled at $573.32 while reaching $1,000 in June 2016. Bitcoin has gained popularity rapidly over the years with its market capital valued at $9,119,135,742 with over 15 million Bitcoins in circulation. As of October 7, 2014 more than 64,000 businesses were reported to accept Bitcoin payments globally [19]. A Bitcoin does not have any underlying association with sovereign currency or commodities and is not acknowledged by global banks as legal tender. It relies on the participating nodes within the network to verify all transactions. There has been much discussion on the privacy of bitcoins users and whether or not they can operate anonymously.

As user adoption steadily grows, so do does the number of businesses that accept the controversial crypto-currency. It is now not so uncommon to find users order food for delivery, engage in online dating , and purchase everything from babyfood to video game consoles with the new digital currency. A growing list of firms accept Bitcoin in London, including Tesla, Microsoft, CheapAir.com and Britains first Bitcoin pub, The Pembury Tavern in Hackney, London, where a pint of Milton Pegasus costs 3.70 or 0.0084 Bitcoin. The key question this paper aims to answer is whether or not the inherent protocol structure will allow these businesses to exploit their customers privacy for their own business intelligence purposes. We must consider the privacy implications associated with the use of bitcoins in commercial transactions. As we discuss later in the protocol overview section, each Bitcoin transaction is recorded on a public ledger, commonly referred to a blockchain. While each Bitcoin and Bitcoin wallet address is only identified by a public address, it is not hard to find the wallet address and trace the entire transaction history of that particular Bitcoin or wallet address. Later in this paper we demonstrate an example where the seizure of Bitcoins from the Silk Road founder by the FBI can be traced back through the blockchain.

Online analytics companies such as Google adverts use a number of techniques to track users across the web. By tracking these users, analytics companies are able to build profiles and determine which users are likely to purchase certain items [1]. In this manner both the analytics company and the websites gain insight about people as they browse web pages, or in this case, transact with entities [1].

A persons Bitcoin wallet contains a public ledger that allows anyone to identify the exact Bitcoins contained in each wallet, as part of of the classification investigation carried out in this project, we mine the current balance for each Bitcoin addresses within our data-set. As people spend the bitcoins in their wallet, it is possible to view the history of each transaction in near real-time. Previously when two users sent each other money, in most cases both sides of the transaction were anonymous. As we see the Bitcoin use case change, where users might purchase common goods or services, there has to be a point when the consumer hands over personal information to the commercial entity. The user might also provide an e-mail address to receive an electronic receipt or confirm acceptance of payment and delivery [1]. The anonymity of the user can perhaps be protected if the company were able to provide a one time only public address for the user to send Bitcoins to [1]. However, without a privacy-protecting measure on the buyers side, the company is now able, if it so chooses, to associate the buyers identity with her Bitcoin wallet address [1].

In isolation, knowing the identity of a buyer and her Bitcoin address may not pose significant privacy concerns since a company would still not know the identity of all the other sellers that the buyer transacts business with, however as we show in this paper, it is possible to mine a large number of Bitcoin addresses and associate them to specific public entities. As discussed here [1], it is possible that commercial analytics companies could create Bitcoin identification networks modeled after current third-party ad networks. Were companies to begin sharing de-identified, non-personally identifiable Bitcoin wallet addresses with each other, they would effectively have access to peoples complete purchase histories [1].

Name	**Category**	**Name**	**Category**
Name	Category	Name	Category
School of English	Education	Multichannel Studios	Film Studios
Artexor Media	Software/Development	Network Fish	IT Support
BackupVault	Data Storage	Nincomsoup	Cafes/Delicatessens
Burger Bear	Pubs/Restaurants	online-mixing.com	Production/Distribution
ByMarieLind	Arts/Crafts	Overhang	Sports/Recreational
Cable Studios	Marketing/Advertising	Paddock Studio	Recording Studios
CeX Bromley	Marketplaces	Rathbone News	Newsagents
CeX Harrow	Marketplaces	Rentadesk	Office Space
Clink78	Travel/Accommodation	SENSUOUS	Masseurs
Dream State	Arts/Crafts	Sheridans	Law Firms
La Porca	Pubs/Restaurants	Verrien	Arts + Crafts
Light and Stories	Photography	We Love Flatpacks	Services
London Hackspace	Office Space	Man With a Van	Home/Garden
Sound/Light	Services	Your Sushi London	Food/Consumables
Speech Therapy	Therapy/Counselling	Zaar Properties	Estate Agents
Makers Academy	Education	Zed's Computers	IT Support
Maxbox VR	Electronics	Zen Head Shop	Head Shops
Microworld	Electronics	n/a	n/a

Table 1: Sample of London based businesses now accepting Bitcoin

3 Background

Bitcoin is a complex system composing of cryptography, distributed algorithms and incentive driven behaviour [2]. The blockchain lies at the heart of Bitcoin and is a public ledger in which each transaction is recorded. Each transaction is recorded by Bitcoin users themselves and every user maintains a copy of the blockchain locally, hence the decentralised aspect of Bitcoin. Users can solve a computationally intensive cryptographic problem to successfully commit a block to the blockchain and if successful they are rewarded in bitcoins. For the purpose of this introduction we will assume 'users' are running full nodes in the network. A full node means the user actively maintains a copy of the blockchain on their local machine and keeps it connected to the Bitcoin network. It is possible for users to transact Bitcoin without running a node if they so wish.

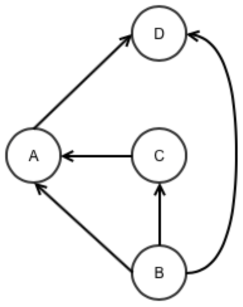


Figure 1: Diagram showing a simple transaction graph subset.

3.1 Understanding the Bitcoin Protocol

3.1.1 The Blockchain

The blockchain prevents users from double spending and restricts users from modifying previous transactions or blocks. A node will maintain its own version of the blockchain with blocks validated by the node itself. When multiple nodes share the same blockchain they are considered to be in consensus [4]. All nodes abide by the same consensus rules, however in some cases new rules can be added or updated causing some nodes to become out of sync with other nodes. In this instance we see a fork in the blockchain whereby some nodes following the upgraded rules accept blocks and out of date nodes reject blocks, or vice versa. This will cause multiple chains, however the Bitcoin core detects out of date nodes or chains and is able to rectify the difference through a proof of work method [4].

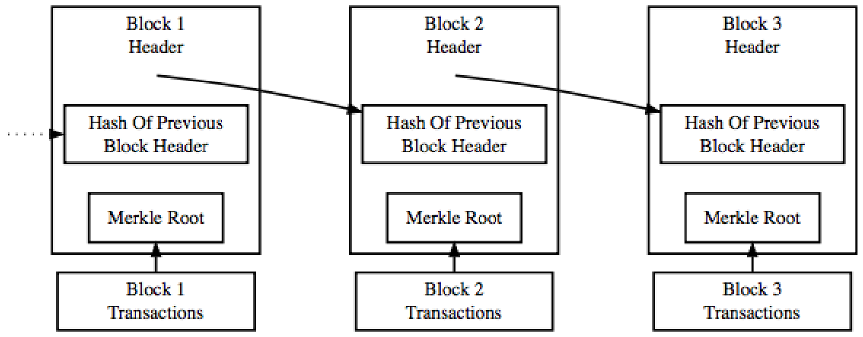


Figure 2: Blockchain protocol structure [4].

A block within the blockchain is split in to three parts; the block header, the number of transactions in the block and the raw block transactions themselves. As shown in figure 2, the block header contains information linking it directly to the previous block (except for the first block, this is called Genesis).

The first transaction within a new block is always the coinbase transaction. This collects and spends any transaction fees paid by the blocks transactions. This is the reward to the user (miner) for appending the block to the blockchain. The block reward originally started at 50.00 BTC per block, this however has been reduced to 12.50 BTC since many more users have started mining themselves.

Towards the end of 2015 the complexity involved in mining has greatly increased and will continue to do so over time.

Name	**Bytes**	**Data Type**	**Description**
Version	4	uint32_t	Set of block validation rules to follow.
Previous block header hash	32	char32	SHA256(SHA256()) hash in internal byte order of the previous blocks header. This ensures no previous block can be changed.
Merkle root hash	32	char32	SHA256(SHA256()) hash in internal byte order. The merkle root is derived from the hashes of all transactions included in this block, ensuring that none of those transactions can be modified.
Time	4	uint32_t	Unix epoch time when the miner started hashing the header (according to the miner). Must be greater than or equal to the median time of the previous 11 blocks.
nBits	4	uint32_t	Encoded version of the target threshold this blocks header hash must be less than or equal to.
Nonce	4	uint32_t	Arbitrary number miners change to modify the header hash in order to produce a hash below the target threshold.

Table 2: Bitcoin block header definition

3.1.2 Transactions

It is important to make clear that the currency moves from transaction to transaction and not wallet to wallet.

- The input of one transaction is output of another.
- A single transaction can create multiple outputs.
- The output of a transaction can only be used as an input.
- Each transaction record involves one or more sending addresses and one or more receiving addresses.

Bitcoin ownership is a matter of public record in the blockchain. Proving ownership resides in the users access to the private key associated with a given public key. In order for a user to be able to send a Bitcoin they must be able to sign the transaction with their private key. Lets assume A would like to send 10 BTCs to B. A must create a message and sign it with their private key. Users of the Bitcoin are able to see the message/transaction and verify that it was A who sent it. Considering the entity sending the transaction must sign the input hash with their private keys, we can assume that multiple input address are owned by the same user. This is an important clustering heuristic discussed later in this paper.

3.1.3 Mining

In order for the transaction to be committed to the blockchain the miner must first verify that the transaction has indeed been correctly signed by A and that A has a sufficient balance to pay B. At this point miners will bundles these verified transactions into a block. A header is then created and inserted into the newly created block, the header contains a hash of the most recently created block. The next step is solving a computational task known as the 'proof of work' problem. If the miner is successful the new block is added to the local blockchain which is they propagated through the network. The miner is then entitled to the block reward in the 'coinbase' transaction. The mining reward means that new Bitcoins are generated and circulated within the network. Presently a block is mined every 9 minutes.

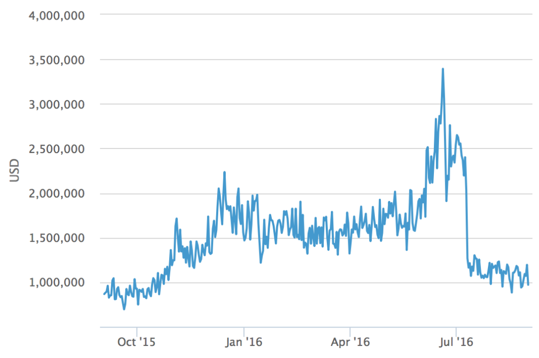


Figure 3: Transaction fees paid to miners.

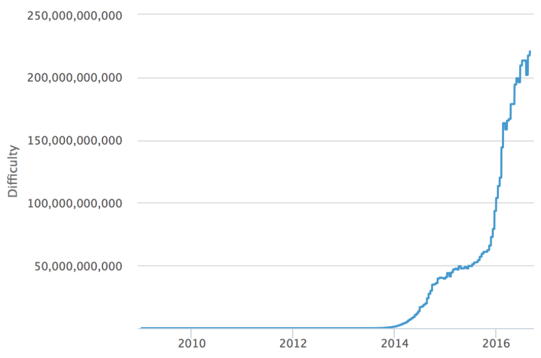


Figure 4: Measure of how difficult it is to find a new block.

The purpose of the 'proof of work' is to make sure that the task of mining is time consuming and costly in hardware and energy to miners. Other users in the network can easily confirm that the miner has successfully met all requirements. If we assume a miner has constructed a new block, inserted a header with a hash of the previous block and inserted a nonce (an incremental id), the miner must then create a hash of the block which must be less than the value of the set target value (a 256-bit number). The target value is recalculated every 2016 blocks and is adjusted based on the mining difficulty value. This difficulty metric enables a self regulating mining eco-system. If for example more miners join the network, the number of blocks being created would increase, therefore the average mining time would decrease which in turn would cause the mining difficulty to increase. As the difficulty rate increases, the block creation rate would decrease and thus return to an average rate.

3.1.4 Analysis of Bitcoins growing adoption.

There are a number of ways user might enter the Bitcoin network. Firstly through mining, this however has become so computationally expensive that it would be rare to find individuals mining coins by themselves. It is more likely that users would participate in mining pools in which many users distribute the work and maintain shares in the reward if a block is successfully mined. It is most likely that users will buy coins from exchanges in exchange for fiat currencies and then store their coins with wallet services or just hold the coin on their local machine. As the mining complexity increases so does the number of users. There are currently over 8,000,000 Bitcoin wallets created and over 392,000 unique address used per day. We would expect the number of unique address to increase significantly if users within the network maintained high standards of privacy by generating a new address for each transaction. Research carried out by [9] confirms this heursitc by showing the beginning and ending balance of each public key over a month period. A number of addresses/keys with decreasing balances tend to loose all value rather than just some over the month period.

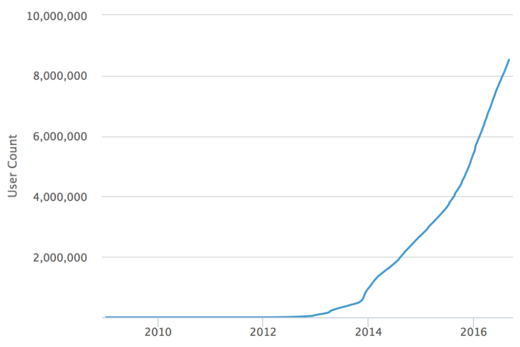


Figure 5: Number of Blockchain wallets.

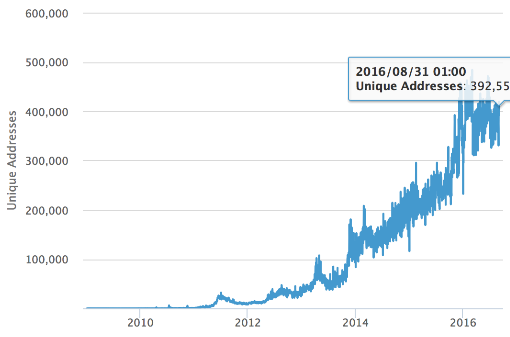


Figure 6: Number of unique addresses.

As we can see from the increasing number of wallets and unique Bitcoin address, as well as increasing media attention, it is clear that the Bitcoin network is gaining traction. Not only are the adoption numbers increasing over time but also the number of transactions per day where as the value of transactions maintain a more conservative increase. This shows that more users are transacting with smaller amounts of money, perhaps spending coins on smaller spend services rather than large purchases.

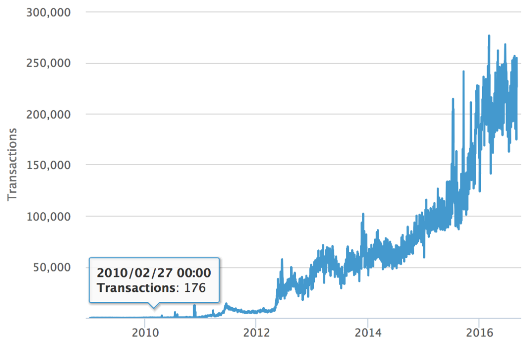


Figure 7: Daily confirmed Bitcoin transactions.

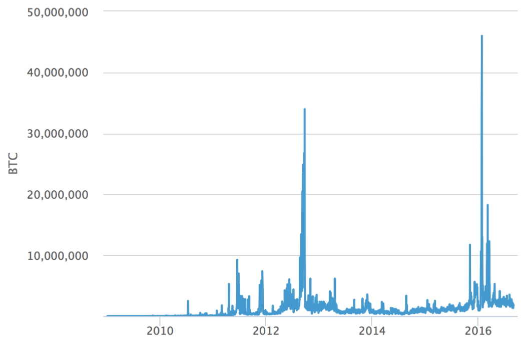


Figure 8: Value of all transaction outputs per day.

The fantastic returns early adopters of Bitcoin users also plays a part in the surge in popularity of the controversial crypto-currency. An attraction for some new users is the potentialy gains by investing in the currency. As we can see from Figure 9 there have been opportunities to earn solid returns, however volatile the currency is. The sporadic change in Bitcoin value can be mostly associated with media attention as well as macro political influences. A notable example of the change in Bitcoin value is when in 2009 two pizzas were bought for 10,000 BTCs, 4 years later in 2013 those Bitcoins would have been valued at $10,000,000. On the other side of the coin, in August 2016 almost 120,000 BTCs, worth around $78m, were stolen from Hong Kong-based Bitfinex, one of the most popular cryptocurrency exchanges, causing a 20% decline in the value of the currency, this is shown in the right hand side of Figure 9.

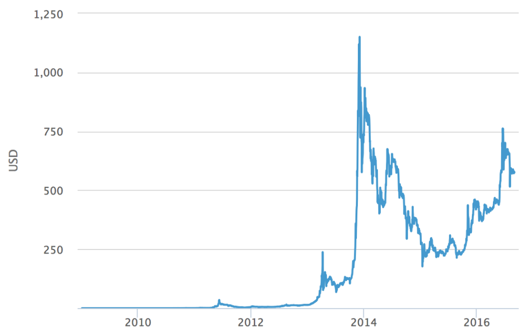


Figure 9: Average USD market price.

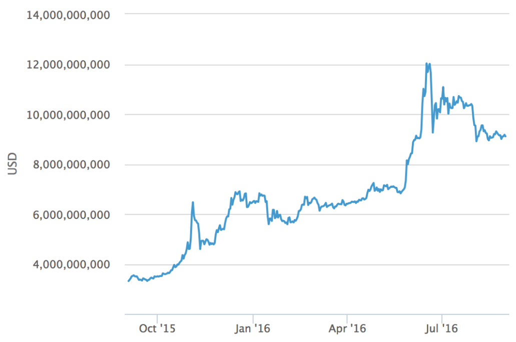


Figure 10: Total USD value of Bitcoin supply in circulation.

3.1.5 Evolution of the Bitcoin Structure

The Bitcoin network structure follows a ”rich get richer” phenomenon, recent work by [9] studies the evolution of basic network characteristics over time, such as the degree distribution, degree correlations and clustering. Each node is represented as a Bitcoin address (or a public key as discussed earlier) and each transaction represents an edge. As of 2011 the Bitcoin network had 13,086,528 nodes and 44,032,115 edges. We know that each block maintains a timestamp and this enables us to investigate the evolution of the network over a given time period. The paper indicates two main stages of evolution. The initial phase, lasting until 2010, whereby the currency is mainly used for experimental purposes. Secondly the trading phase, this represents a period when users began trading the currency for value purposes. Kondor, Posfai, Csabai and Vattay verify this by using the Gini co-efficient, a measurement of wealth distribution in economics. $G = 0$ represents perfect equality and $G = 1$ inequality (one person owns all of the wealth). The wealth distribution during the initial phase was close to 1, this is because only a few users had bitcoins and were not able to trade them. According to this report the coefficients decrease to G_{in} 0.629 and G_{out} 0.521 throughout the trading phase.

The clustering coefficient, a degree in which nodes of a graph tend to cluster together is measured:

$$C = \frac{1}{N}\sum_{v}\frac{\Delta v}{d_v(d_v-1)/2}$$

Figure 11: Mathematical definition of clustering coefficient

As you would expect the clustering coefficient is high during the initial phase (around 0.15 according to this paper). However it drops to 0.05 during the trading phase as we see the currency reach a wider audience and business starting to accept the currency for goods or services.

3.1.6 Inferring the interplay between network structure and market effects in Bitcoin

Recent work by [10] investigates the possibility that external macroeconomic events can be associated with changes in the network structure. The paper reconstructs the entire Bitcoin network and examines the effects the most active users have on the sub-graph. Principle Component Analysis is applied to snapshots of the market at different points to show a change in network structure. The changes are then compared to macroeconomic events, in this case market price.

Firstly the transaction list is contracted by clustering addresses that belong to the same user. This is done by by a heuristic we discuss later on this paper that multiple addresses in one transaction are the same entity. Two sub-networks are derived from the data. It is assumed that users with a start and end activity period of 600 days are long term users (LT). The LT sub network consists of 1639 nodes and 4,333 edges with these users controlling 1,894,906 addresses and participating in 4,837,957 transactions. The second sub network is the top 2,000 most active users (AU). Addresses associated with the SatoshiDice gambling site are excluded from this sub network. The AU sub network has 1,288 nodes and 7,255 edges; the users in this subgraph have a total of 3,326,526 individual Bitcoin addresses and participated in a total of 12,900,964 transactions during the two years. The total number of Bitcoin transactions in this period is 27,930,580, meaning that the two subgraphs include 17.3% (LT) and 46.2% (AU) of all transactions respectively [10].

Going forward we assume $u \rightarrow v$ where u and v are nodes or users and the edge represents a transaction within the two year period. The authors of the paper aim to obtain a base set of networks and represent each days snapshot as a linear combination of these base network where each snapshot is a weighted network, and the weight of link $u \rightarrow v$ is equal to the number of transactions that occurred that day between u and v [10]. The snapshot for each day is represented as an adjacency matrix and allows us to generate a graph time series matrix.

The long term user network shows the top 20 ranking edges (determined by the value of the weight) yields only 46 out of 200 distinct edges. Of these 46, 95% are weakly connected with only 29 users owning 1,349,815 addresses. Of these, 20 form a strongly connected component.

This research by [10] concludes that there are high correlation coefficients between the most active 2,000 Bitcoin users and the market price. Using a linear combination of four vectors this paper was able to reproduce features contributing to the market price.

3.1.7 Demonstration of basic transaction analysis

The purpose of this section is to show how one can target a specific Bitcoin address or transaction and trace the users or services it has passed through to end up at this position in the network graph. As described in the protocol overview section, each output address must become the input address in another transaction. We can therefore visualise ans track the flow of coins with ease. The diagram below shows the transaction history of a specific address, this address is well know as it was used to seize Bitcoins from the founder of Silk Road shortly after his arrest. As shown, once the FBI had seized the coins there were able to move the coins into various other wallets, and twice sent coins to themselves, this is shown in the self loop one hop after the seizure.

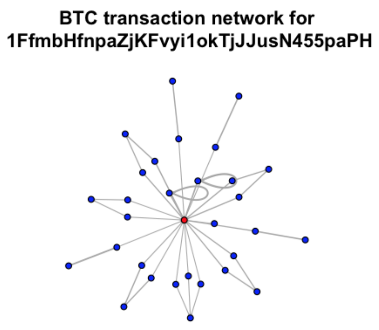

BTC transaction network for
1FfmbHfnpaZjKFvyi1okTjJJusN455paPH

Figure 12: Transaction structure of the FBI Bitcoin address.

The next visualisation shows the point when the FBI auctioned of the seized bitcoins. At this point we know that the community has accepted the coins in to circulation, the blocks have since been appended to the blockchain but we can still trace the flow of coins and determine whether or not they are returned to the FBI wallet or spent on certain products/services.

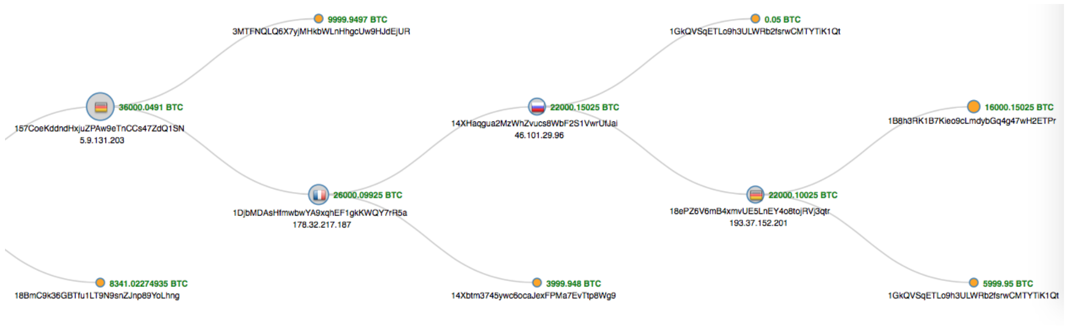

9999.9497 BTC
3MTFNQLQ6X7yjMHkbWLnHhgcUw9HJdEjUR
36000.0491 BTC
157CoeKddndHxjuZPAw9eTnCCs47ZdQ1SN
5.9.131.203
26000.09925 BTC
1DjbMDAsHfmwbwYA9xqhEF1gkKWQY7rR5a
178.32.217.187
8341.02274935 BTC
18BmC9k36GBTfu1LT9N9snZJnp89YoLhng
0.05 BTC
1GkQVSqETLo9h3ULWRb2fsrwCMTYTiK1Qt
22000.15025 BTC
14XHaqgua2MzWhZvucs8WbF2S1VwrUfJai
46.101.29.96
3999.948 BTC
14Xbtm3745ywc6ocaJexFPMa7EvTtp8Wg9
22000.10025 BTC
18ePZ6V6mB4xmvUE5LnEY4o8tojRVj3qtr
193.37.152.201
16000.15025 BTC
1B8h3RK1B7Kieo9cLmdybGq4g47wH2ETPr
5999.95 BTC
1GkQVSqETLo9h3ULWRb2fsrwCMTYTiK1Qt

Figure 13: Demonstration of targeted transactional flow analysis.

This technique of tracking coins and users shows that with some machine learning input and perhaps some label propagation we can generate similar visualisations or analytics application for businesses to track transactions that could be useful for their own marketing purposes. We can now aggregate and label enough of the network using the clustering methods discussed in the next section and the train statistical models to predict which of these entities, based on their trading history and node properties within the network, are likely to interact with categories. The combination of derived node attributes, graphical properties and the inherent structure of the network will allow us to potentially threaten consumers privacy as the network becomes more commercialised.

3.2 Definitions of evaluation metrics

The following definitions are shown here for completeness. More information can be found here [22], this is the source of information for this section, which has been presented here.

3.2.1 Recall

Recall answers the question: Out of all the items that are true, how many are found to be true by the classifier? [22]

$$recall = \frac{\#true\ positive}{\#true\ positive + \#false\ negative}$$

Figure 14: Mathematical definition of recall

3.2.2 Precision

Precision answers the question: Out of the items that the classifier predicted to be true, how many are actually true? [22] The precision score quantifies the ability of a classifier to not label a negative example as positive. The precision score can be interpreted as the probability that a positive prediction made by the classifier is positive. The score is in the range $[0, 1]$ with a value close to 0 being the worst and closer to 1 being the best score. The precision is defined as:

$$precision = \frac{\#\, true\, positive}{\#\, true\, positive + \#\, false\, positive}$$

Figure 15: Mathematical definition of precision

3.2.3 Accuracy

The accuracy metric tells us how often our model has made a correct prediction. Its the ratio between the number of correct predictions and the total number of predictions (the number of test data points) [22].

$$accuracy = \frac{\#\, correct}{\#\, predictions}$$

Figure 16: Mathematical definition of accuracy

It is possible to get a very high accuracy score but a badly performing model. For example if our model predicts everything to be in class 0, then the model will still correctly classify a large number of predictions, accuary does not tell us how many times the model actually predicted something correctly.

3.2.4 F1 Score

The F1-score is a single metric that combines both precision and recall via their harmonic mean:

$$F_1 = 2\frac{precision * recall}{precision + recall}$$

Figure 17: Mathematical definition of the F1 score

The score lies in the range $[\mathit{0}, \mathit{1}]$ with *1* being ideal and *0* being the worst.

$$F_\beta = (1+\beta^2)\frac{precision * recall}{\beta^2 precision + recall}.$$

Figure 18: Mathematical definition of the F1 beta score

3.2.5 Receiver Operating Characteristic (ROC Curve)

The ROC curve shows the sensitivity of the classifier by plotting the rate of true positives to the rate of false positives. In other words, it shows you how many correct positive classifications can be gained as you allow for more and more false positives [16]. A successful model would show a true positive rate of 100%, without incurring any false positives. This rarely happens. A good ROC curve will show a large space below, ideally showing a steep incline. A bad model will show a very small amount of space below the ROC curve.

3.2.6 Log Loss

Log-loss, or logarithmic loss, will be used in our logistic regression model. This evaluation metric can be used with the prediction returns a probability value of classification. If the true label is ”0” but the classifier thinks it belongs to class ”1” with probability 0.51, then the classifier would be making a mistake. But its a near miss because the probability is very close to the decision boundary of 0.5 [16].

$$log - loss = \sum_{i=1}^{N} y_i \log(p_i) + (1 - y_i) \log(1 - p_i)$$

Figure 19: Mathematical definition of Log Loss

Here, p_i is the probability that the i_{th} data point belongs to class ”1”, as judged by the classifier. y_i is the true label and is either ”0” or ”1”. By minimizing the cross entropy, we maximize the accuracy of the classifier [16].

4 Key literature in the field and influences on this project

To the best of our knowledge there has not been any work carried out using supervised learning to classify Bitcoin transactions using network metrics. However, there have been two popular areas of research within this field that have been useful to this project.

Firstly, much work has been carried out using unsupervised learning techniques to detect anomalies within the Bitcoin transaction network. Inspired by [7] and [15], an investigation was carried out to use the k-means algorithm as an unsupervised learning approach to clustering users into categories. The main reason not pursuing this approach is the lack of ground truth data and during the process of mining data, the problem became suitable for supervised learning and less unsupervised which in turn was able to generate much better results.

The second popular topic within Bitcoin and machine learning is price prediction. There have been a few research papers in this area, with the most notable being [20] which uses Bayesian regression to predict whether the price of Bitcoin will be positive or negative over given time interval periods. The paper claims to double its money within 60 days of trading.

Thorough research within both of these sub-fields led to this paper. This investigation led to two core decisions when carrying out this research. Firstly, much recent work has been based on the same data-set, this therefore inspired us to extract our own, up to date data-set rather than using the standard 2013 data-set kindly provided by [5]. Therefore the work carried out during our research is using current and up to date transactions. We feel it is important to reflect the current state of the evolving Bitcoin network. The second decision was to work with both network metrics and transactional data we would expect to see in the price prediction field. Therefore our feature set will include not only PageRank, in-degree etc. but also include the average price per transaction and number of transactions as both sets of information can provide valuable insight to our clustering predictions.

The lack of ground truth data within the Bitcoin network had great influence on the direction of this project. As stated, the unsupervised learning approach to clustering transactions has been a topic of interest for a few years now, however in each case it becomes hard to evaluate the work carried out. At best the user is able to identify a handful of well known anomalies and show whether or not their work can detect them. Firstly, this means that their evaluation methods are limited to a very small test set, secondly that they have no way of verifying false positives and even true positives because is it very unlikely that any users in the Bitcoin network would be willing to reveal their identities. We therefore decide to apply some data mining techniques to seek out an acceptable data-set that can be used as both a training, test and validation set.

5 Project plan

5.1 Problem statement

The goal of this project is to demonstrate that as Bitcoins popularity increases so will the number of ways in which it is used, namely commercial adoption will inevitably create demand for targeting and identifying clusters of users for business intelligence purposes. This use case highlights a potential threat to user privacy.

5.2 Objectives

We first provide a complete introduction to the Bitcoin network protocol, transactions within the network, the purpose of mining and importantly the growing adoption of Bitcoin as a currency. We provide network metric analysis, showing a number of values such as PageRank, in-degree, out-degree and various other graph attributes as well as describing the separate user and transaction graphs. After providing a foundation of the Bitcoin network aim to provide insight into the most recent work in this field and describe the decisions made from previous work when implementing this project.

We then implement data mining techniques, various pre-processing methods and apply two machine learning techniques, presenting our results and ultimately showing that categorical clustering can be used for intelligence purposes.

- Discuss current work in the field, examine the successes and flaws in both price prediction and anomaly detection. Section 4.1.7 discusses the influences on this project from previous works.
- Extract a recent segment from the Bitcoin network. This involves taking the binary data and transforming the data into a graphical format.
- Write web scraping scripts to extract known Bitcoin addresses from the web and provide a categorical tag with each address. This data is used to build a ground truth data-set.
- Create a user graph with the categorical tags and basic information representing the users transactional behaviour.
- Build two supervised machine learning models. We use Support Vector Machines and Logistic Regression as both solve classification problems. The models will take a transaction and predict with or not they would be used in a specified category.
- Create a training, validation and test data-set.
- Train the machine learning models.
- Define and present evaluation metrics used to assess the quality of the implementation. We use recall, precision, F1, accuracy, ROC and log assess as evaluation metrics.
- Analyse the evaluation metrics and look for improvements within each model based on the feature sensitivity analysis coefficient metrics.

5.3 Inspiration

The topic of crypto-currencies is a new and exciting area to work in. The inspiration for this work comes from a desire to work on something novel and current. Although there has been work carried out in the field of anomaly detection and price prediction, to the best of our knowledge, there has not been any supervised learning applications to network metrics and transactional data to categorise entities.

5.4 Methodology

5.4.1 Extracting the data-set

An up to date snapshot of Bitcoin transactions were taken using the bitcoingraph tools [3]. An extract was taken from blocks 417,500 to 424,572, which consisted of 62,418,698 transactions. Parsing the data-set requires that $BitcoinCore(v.11.1)$ is installed and running as a local node. The blockchain currently consists of 133GB of data, therefore a p2p network was first used to bulk load the bulk chain, this massively reduced the initial load time. That being said, the full data load and the re-indexing of the chain took about one week to complete. The client provides access to three programs: bitcoind (= full peer), Bitcoin-qt (= peer with GUI), and Bitcoin-cli (RPC command line interface). Running Bitcoin with the -server argument (or running bitcoind) tells it to function as a HTTP JSON-RPC server, but Basic access authentication must be used when communicating with it, and, for security, by default, the server only accepts connections from other processes on the same machine [4]. The following configuration must be set for bitcoingraphs to be able to access teh raw blockchain data:

```
# server=1 tells Bitcoin-QT to accept JSON-RPC commands.
server=1

# You must set rpcuser and rpcpassword to secure the JSON-RPC api
rpcuser=your_rpcuser
rpcpassword=your_rpcpass

# How many seconds Bitcoin will wait for a complete RPC HTTP request.
# after the HTTP connection is established.
rpctimeout=30

txindex=1

# Listen for RPC connections on this TCP port:
rpcport=8332
```

Once configured the Bitcoin command line interface is able to accept connections through the REST interface.

```
bitcoind -daemon -rest
```

We can now extract a specified number of Bitcoin blocks by using the following command:

```
bcgraph-export 0 1000 -u your_rpcuser -p your_rpcpass
```

The following CSV files are created (with separate header files):

File	Description
addresses.csv	sorted list of Bitcoin addressed
blocks.csv	list of blocks (hash, height, timestamp)
transactions.csv	list of transactions (hash, coinbase/non-coinbase)
outputs.csv	list of transaction outputs (output key, id, value, script type)
rel_block_tx.csv	relationship between blocks and transactions (block_hash, tx_hash)
rel_input.csv	relationship between transactions and transaction outputs (tx_hash, output key)
rel_tx_output.csv	relationship between transactions and transaction outputs (tx_hash, output key)
entities.csv	list of entity identifiers (entity$_id$)
rel$_address_entity.csv$	assignment of addresses to entities (address, entity$_id$)

Table 3: Bitcoingraph output files

['tx_hash', 'coinbase', 'block_hash', 'height', 'timestamp', 'id', 'value', 'script_type', 'output_address', 'input_address', 'year', 'month', 'day', 'close_price']

We then configured a local server to act as an interface to the local blockchain. The local web interface is used to query the blockchain and check results thorughout the process.

```
server=1
whitelist=127.0.0.1
txindex=1
addressindex=1
timestampindex=1
spentindex=1
zmqpubrawtx=tcp://127.0.0.1:3001
zmqpubhashblock=tcp://127.0.0.1:3001
rpcallowip=127.0.0.1
rpcuser=archienorman
rpcpassword=Gold2020

reindex=1
```

5.4.2 Configuring GraphLab Create

GraphLab Create, is a graph-based, high performance, distributed computation framework written in C++. "The GraphLab framework is a parallel programming abstraction targeted for sparse iterative graph algorithms. GraphLab provides a high level programming interface, allowing a rapid deployment of distributed machine learning algorithms" [3]. Main features of GraphLab include: a unified multicore and distributed API, optimized C++ execution engine leverages extensive multi-threading and asynchronous IO as well as maintaining powerful machine learning toolkits [8]. GraphLab exploits the sparse structure and com- mon computational patterns of ML algorithms. GraphLab enables ML experts to easily design and implement efficient scalable parallel algorithms by composing problem specific computation, data-dependencies, and scheduling [11].

Ensure Python 2.7.x

```
# Create a new conda environment with Python 2.7.x
conda create -n gl-env python=2.7 anaconda

# Activate the conda environment
source activate gl-env
```

Ensure pip version $= 7$

```
# Ensure pip is updated to the latest version
# miniconda users may need to install pip first, using 'conda install pip'
conda update pip
```

Install GraphLab Create

```
# Install your licensed copy of GraphLab Create
pip install --upgrade --no-cache-dir https://get.graphlab.com/ \
GraphLab-Create/2.1/your registered email address here/your product \
key here/GraphLab-Create-License.tar.gz
```

Ensure installation of IPython and IPython Notebook

```
# Install or update IPython and IPython Notebook
conda install ipython-notebook
```

5.4.3 Pre-processing the data-set

A sample was taken of 15% resulting in a graph of 9,368,903 transactions and 5,588,275 edges. We first perform sanity checks by iterating through the dtaframes and removing any rows where the sender or receive is not a hash number as well as removing any punctuation in the address strings. The categorised Bitcoin address data-set was next loaded into a data-frame and joined with the transaction data-set resulting in a tagged transaction list. The next step was to split out the transaction list into a user list, while maintaining information about whether the user was a sender or receive of a particular category. We then compute transaction based metrics by counting the number of transactions per address and calculating an average transaction value. Finally in-degree, out-degree, PageRank, k-core and the connected component metrics were added to the user list resulting in the following data-set for input to our supervised learning algorithms.

['address', 'received_charity', 'received_finance', 'received_gambling', 'received_junk', 'received_pools', 'received_services', 'sent_charity', 'sent_finance', 'sent_gambling', 'sent_junk', 'sent_pools', 'sent_services', 'transaction_count', 'value', 'category', 'is_gambler', 'is_charity', 'is_finance', 'is_junk', 'is_pools', 'is_services', 'PageRank', 'delta', 'in_degree', 'out_degree', 'component_id', 'core_id', 'avg_value']

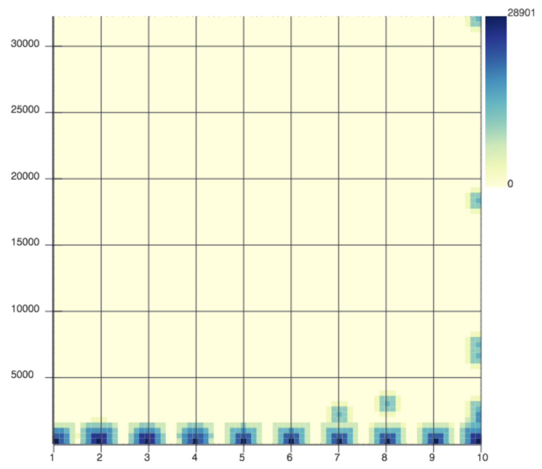


Figure 20: Logarithmic plot of in-degree vs core id

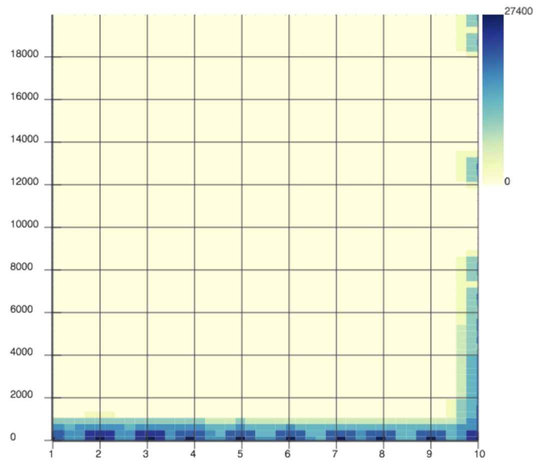


Figure 21: Logarithmic plot of out-degree vs core id

After defining our transaction graph into k-core components, we can see that core id 10 holds a much larger number of transactions that the rest of the cores, with only a slightly higher mean value. By diving a little deeper we can see that core 10 contains a large number of transactions with gambling sites and the services category which would explain the high number of transactions.

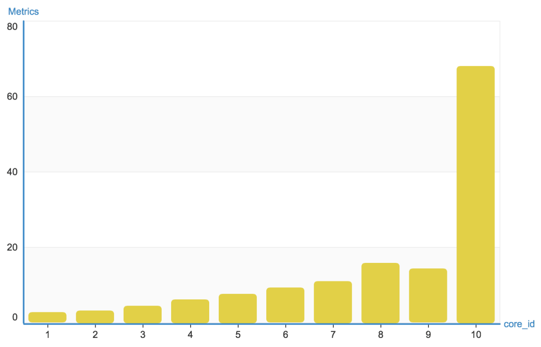


Figure 22: Plot showing mean number of transactions per core id

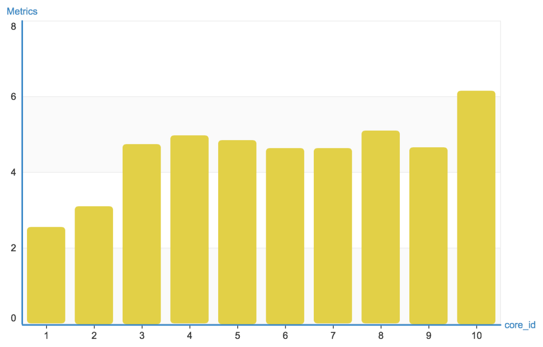


Figure 23: Plot showing mean value of transactions per core id

A well known obstacle with Bitcoin data is that there is no ground truth data-sets available due to the anonymity within the network. By spending extra time mining tagged addresses we are able to build ourselves a data-set in which we can train supervised learning models.

The final step of the pre-processing stage is to generate a binary True or False value for each user per category. We take the generated columns 'is_services' and assign a binary value depending on whether or not the user has transacted with the given category multiple times. To reduce noise within the data we build the data-set using only services, finance, pools and gambling. We decided to remove the junk and charity categories because there was not enough tagged addresses found to make the training of a model worthwhile.

5.4.4 Clustering heuristics

The aim of this paper is to cluster transactions based on a combination of mostly network properties and a small amount of transactional information. In this section, we discuss the work carried out by [12], their work highlights two important clustering heuristics which can be inferred from the Bitcoin protocol. The clustering enables us to view users as entities rather than simply Bitcoin addresses. An entity can be considered a user that controls multiple public addresses.

First consider two graphs, the transaction graph and the public address graph. The transaction graph represents in and out degree as the number of inputs or outputs on a per transaction basis. The vertices represent transactions, and a directed edge from a transaction t_1 to a transaction t_2 indicates that an output of t_1 was used as an input in t_2. However, in the public address graph we represent vertices as public keys and directed edges again represent the flow of money from one public key to another.

The Bitcoin protocol specifies that when the bitcoins are the output of a transaction they must be spent all at once: the only way to divide them is through the use of a change address. The change address is the public key in which the remaining balance is returned in change to the sender.

Research carried out by [17] [18] has shown that the multiple input addresses can be aggregated to represent one entity as previously discussed. If we observed one transaction with addresses A and B as inputs, and another with addresses B and C as inputs, then we conclude that A, B, and C all belonged to the same user. It is also quite safe: the sender in the transaction must know the private signing key belonging to each public key used as an input, so it is unlikely that the collection of public keys are controlled by multiple entities (as these entities would need to reveal their private keys to each other) [12].

As discussed above, change addresses are the mechanism used to give money back to the input user in a transaction, as bitcoins can be divided only by being spent[12]. The change address clustering heuristic was developed in 2013 by [12]. One of the key foundations of the change address is that it is generated internally by the Bitcoin client and never re-used and we can therefore assume that it is very unlikely to give out their change address to receive payments from other users. This heuristic does come with added risk as assumptions are made on the use of the network rather than an inherent idiom. [12] define the following logic when mining for change addresses, this is the first appearance of this public key, the transaction is not a coin generation, there is no self change address.

5.4.5 Mining for Bitcoin identities

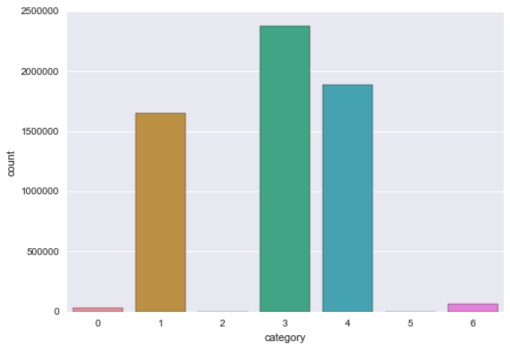
The goal of the mining phase is to extract publicly available information and build a ground truth data-set. In order to do so, we first define six categories to group the addresses into: gambling, charity, finance, services, junk and mining pools. Three separate web scrapers were built to parse the html structures of three websites. Each website maintains a number of known Bitcoin addresses and groups them by the entity that owns them. The following websites were mined:

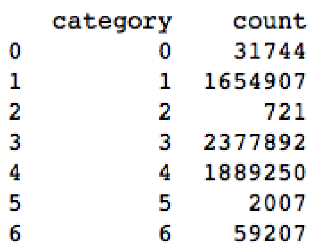
https://blockchain.info/de/tags
http://bitcoinwhoswho.com
https://www.walletexplorer.com

The blockchain.info website did not perform any categorisation on addresses but only provided verified tags. In order to classify this data into categories a small program was written to present a tag to the command line in which the user could manually classify the tag. The classification was then applied to all address with similar tags to reduce the manual effort presented to the user. This is an example of the raw data extracted by the crawler, which was then used to classify addresses through program.

1change6feSoBHr957Zh5qmBrebA8oea7	SatoshiBONES	Hot Wallet
1bonesmxY3yyK74cjHPKoTQjmHBR4xZ6	SatoshiBONES	0.09766pct
1bonesG6wgmA8612BAHc8HHJUezQe38V3	SatoshiBONES	1.562pct

The rest of the category classification was achieved through the automated web crawler. This was achieved by identifying the category of address by the url, storing the address and integer category as a key, value pair and writing the pair to a csv file. Over the course of the address mining phase, 6,015,728 Bitcoin addresses were collected each with an assigned category.





```
   category    count
0         0    31744
1         1  1654907
2         2      721
3         3  2377892
4         4  1889250
5         5     2007
6         6    59207
```

Figure 24: Breakdown of categorised addresses mined.

5.4.6 Supervised Learning approach to classifying services

Now we have out input data for the algorithms we first split the data-set into training and test sets. The training set consists of 80% of the whole data-set. This consists of the labelled pre-processed data and will be used to fit the Logistic Regression model and the Support Vector Machine models.

A validation data set during the model creation, it is used for monitoring and tuning the model to improve its generalization performance. The validation set consists of 5% of the training model. For each row of the progress table, the chosen metrics are computed for both the provided training data-set and the validation set. The tuning process optimizes the selected model on the validation data. Consequently, a further holdout sample is needed for a final, unbiased assessment.

The test set is used to evaluate the abilities of our generated model. The data in the test set are treated in the same way that new predictions would, ie. they are not able to see the target value shown in the training data. They cannot be involved in the determination of the fitted prediction model. For the the purpose of this paper we will be classifying addresses that are likely to use services rather than gambling, finance or mining pools.

6 Results

6.1 Stage 1: Network analysis

6.1.1 Degrees

For a node, the number of directed edges coming into the node is know as the in-degree, whereas the number of direct edges leaving the node is know as the out-degree. In this instance the degree shows the number transaction to and from a Bitcoin address. We can formulate this by defining $G = (V, E)$ and vV. The in-degree of v is denoted $deg(v)$ and its out-degree is denoted $deg + (v)$. A vertex with $deg(v) = 0$ is called a source. A vertex with $deg + (v) = 0$ is called a sink. If a vertex is neither a source nor a sink, it is called an internal. If for every $vertex\ v \in V, deg+(v) = deg(v)$, the graph is called a balanced directed graph. We can define the degree distribution the Bitcoin network as the fraction of nodes in the network with degree k.

$$\sum_{v \in V} \deg^{+}(v) = \sum_{v \in V} \deg^{-}(v) = |A|$$

Figure 25: Mathmetical defintion of degree

address	in-degree
1LuckyR1fFHEsXYyx5QK4UFzv3PEAepPMK	2175
1Y2sKhUZvsP2EANpKra3FaFZufHq5VEwT	1972
3NVXE53nFT5x6D6V6j4qAZGcw8RR1wrNV9	1646
1FAv42GaDuQixSzEzSbx6aP1Kf4WVWpQUY	1339
1MFXYK1XucKFfhPhW9HDHD3vsM9BKey4qm	1241
13vHWR3iLsHeYwT42RnuKYNBoVPrKKZgRv	1160
19QC9XoMVWxP43TkzkSKZhtcY4oZFqWb1d	1067
3JCEg58NFBuwdoUP9M3HcnUQN4D4K5A9gE	965
3Pfd4r6HipsdFxbBgvKKVTRkEzLYT2K1yQ	945
3AbixYB8q3hHuAkFWSxUnTtqncFgRFYGDb	927

Table 4: Computing the in-degree.

address	out-degree
0	7901
3NVXE53nFT5x6D6V6j4qAZGcw8RR1wrNV9	3058
3HNSiAq7wFDaPsYDcUxNSRMD78qVcYKicw	2912
1GX28yLjVWux7ws4UQ9FB4MnLH4UKTPK2z	2741
13pucx6gHP2vyBLc88QfcGivjkhK63PeVg	2429
3BQjEhqd7ug4eoA7g1RTXNztwp89c9Qyv7	1790
12oGycatuaruzCDLgZj79R22ppAHd6cUqj	1697
3AbixYB8q3hHuAkFWSxUnTtqncFgRFYGDb	1653
1MFXYK1XucKFfhPhW9HDHD3vsM9BKey4qm	1404
3MWy1dmEEz6xBHA47Lj969QNmzjrFGukPu	1321

Table 5: Computing the out-degree.

6.1.2 PageRank

The Pagerank algorithm is also particularly useful when analysing nodes within a network. It was first introduced by Larry Page and Sergey Brin while they were graduate students at Stanford, it then became a key factor in Google Search in 1998. The premise behind Pagerank is that the importance of a node can be determined by the recursively counting the importance of the nodes that link into the page.

If we create a node address n_1 and send Bitcoin to another node n_2, this means that there is some form of interaction or agreement between these two users. If there are a lots of transactions to node n_2, this means that the common belief is that node n_2 is important. If on the other hand, node n_2 has only receives one transaction, but that comes from an authoritative node n_3, (an address held by Satoshi Namokoto) we say that node n_3 transfers its authority to node n_2; in other words, node n_3 asserts that node n_2 is important. Either through popularity or authority, we can iteratively assign a rank to each node, based on the ranks of the nodes that point to it. We can recursively define PageRank as:

$$pr(i) = reset\_probability + (1 - reset\_probability) \sum_{j \in N(i)} pr(j)/out - degree(j)$$

Figure 26: Mathematical definition of PageRank

where N(i) is the set containing all vertices j such that there is an edge going from j to i. Self lops (i.e., edges where the source vertex is the same as the destination vertex) and repeated edges are treated like normal edges in the above recursion.

address	pagerank	delta
1ZYSHNtBxpaf2XtGdhng6gpqSZc6UX2Vj	239.642706498	8.65280023277e-05
1Y2sKhUZvsP2EANpKra3FaFZufHq5VEwT	212.798935725	0.000200349821768
1HVgipffWqRhRWJqapeN23RGigzpxtGW6n	198.796794492	0.0874415460842
1FAv42GaDuQixSzEzSbx6aP1Kf4WVWpQUY	178.48923347	0.000313952016597
13vHWR3iLsHeYwT42RnuKYNBoVPrKKZgRv	162.804430048	0.000285055618605
19QC9XoMVWxP43TkzkSKZhtcY4oZFqWb1d	145.669657199	2.3712794615e-05
3Pfd4r6HipsdFxbBgvKKVTRkEzLYT2K1yQ	121.727165361	0.000760422400901
3JCEg58NFBuwdoUP9M3HcnUQN4D4K5A9gE	112.694756169	7.28129777485e-05
1MFXYK1XucKFfhPhW9HDHD3vsM9BKey4qm	83.0274824207	2.13922151744e-05
1LApZ4cAv5VsJzHS44V4A8wsvW7meBHg6H	79.5513260453	0.000598371525015

Table 6: Computing the PageRank.

6.1.3 Connected components

A know that a graph is connected if there is a path between every pair of vertices. Therefore in a connected graph there are no unconnected nodes. A graph with two or more nodes but with no edges is known as a disconnected graph. A directed graph is called weakly connected if replacing all of its directed edges with undirected edges produces a connected (undirected) graph. In the Bitcoin network all transactions are connected because there must always be a sender and a reviver in the transaction. It is connected if it contains a directed path from u to v or a directed path from v to u for every pair of $vertices\ u, v$ [25]. It is strongly connected or strong if it contains a directed path from u to v and a directed path from v to u for every pair of $vertices\ u, v$. We show a sample of connected components with the component id.

address	component_id
3R2a9rtVgiz9XP1f6oFcFmyRYMxuGz5YF9	738798
36vpdsTCXJfNumcZHxgVhzF6nAme5zxbgn	738798
32VQYvcXcxzmqqMuk9a5mafaoLeopzJrYv	738796
3R2VHNXVcdPfjmAYyKy48k26LVHWrJ9coW	738796
3PuhFK1gAJhkgEG91h78HbYbkb8oKVthxS	738791
3PyvGYk8fUmvUFcjceedg834CHQnAUGkaZ	738791
3R2EFhcgfSufXg7Fr8M5dek6TQfAbCWBYP	738791
1HrYwxq3iRDvGSQ164vDPy9U8hscW6wypX	738777
3R1WDnpe2hhStAm5aWd3RNHKJHBPGVBBfk	738777
353qvnxRFCa7qvffqTDGyQjMwdNRC4gP69	738773

Table 7: Weakly connected components.

6.1.4 k-core

A k-core model contains a core ID for each vertex, and the total number of cores in the graph. The core ID of a vertex is a measure of its global centrality. The algorithms iteratively remove vertices that has less than k neighbors recursively. The algorithm guarantees that at iteration $k + 1$, all vertices left in the graph will have at least $k + 1$ neighbors. The vertices removed at iteration k is assigned with a core ID equal to k [23]. After analysis we notice that the categorised transaction are relatively evenly distributed across the core ids, we do however see a larger number of gambling, financial and services transactions in cores 10, 2 and 1.

6.1.5 Analysis of network metrics

Preliminary analysis shows the following in-degree, out-degree and PageRank of a sample of transactions over July 2016 with 624,529 nodes and 738,800 edges. As expected the network follows a power distribution with a very high number of nodes within the network maintaining a in-degree, out-degree or PageRank value of close to 0. We then see a very few nodes with extremely high values shown in the results above (the results are sorted by the nodes with the highest degrees and PageRank respectively). It is likely that these nodes are exchanges, sending and receiving large numbers of transactions per day.

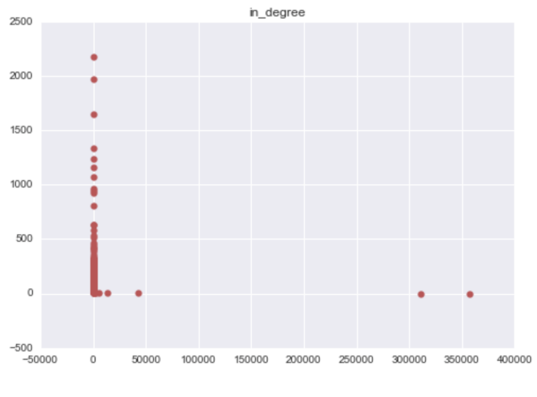


Figure 27: Network in-degree.

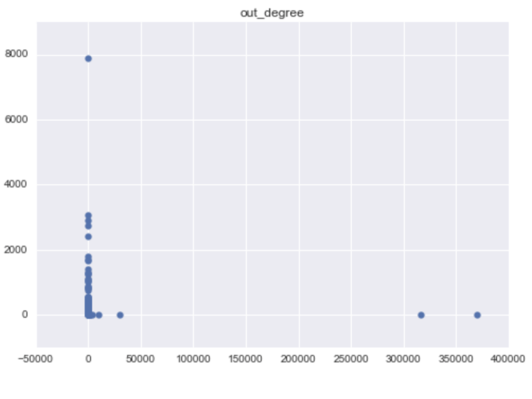


Figure 28: Network out-degree.

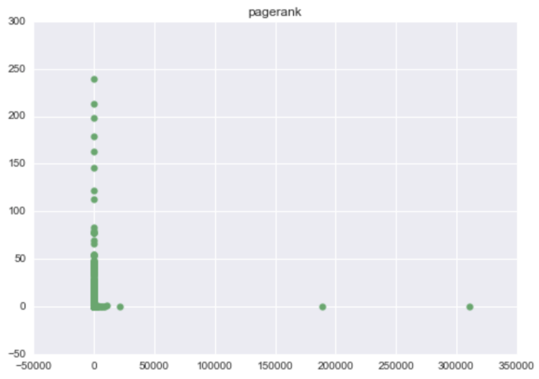


Figure 29: Network PageRank.

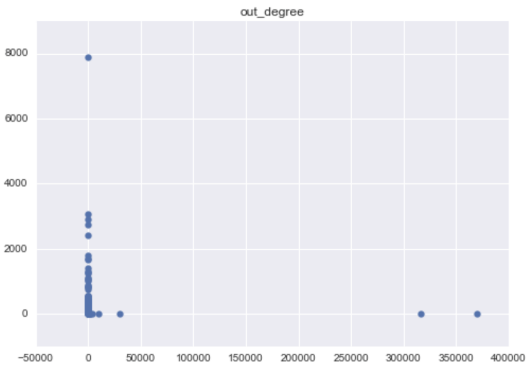


Figure 30: Network out-degree.

The network is also sparsely connected, like most real networks. This is shown in the data and one would not expect every single user to trade with every other user in the network. Therefore we can assume:

$$E < E_{max} for E_{max} = N(N-1)/2$$

Figure 31: Mathematical definition of sparse networks with remote nodes

The results also show this network to be a scale-free (power-law) node-degree distribution. There are few well linked nodes and many sparsely linked nodes within the same network. The network metric results shown here will be used as input features in the next stages of this report.

6.1.6 Transaction network characteristics

The blockchain network has mostly been analysed as a transaction graph. Due to processing limitations one day's worth of transactions were analysed (1,994,006 nodes and 1,521,030 edges). We assume each node to be a transaction and the edges represent the output of one transaction and the input of another resulting in a directed graph. At present there are over 100,000,000 transactions in the blockchain. We see a very high number of in and out degrees with a value of 1, which decreases down the scale with very few transactions nearing the 1,000 mark. The network features a giant connected component if we disregard the directional aspect of the graph, connecting 99% of all transactions. The transaction network shows a relatively high connected core of input/outputs with a very sparse set of transactions, however this is to a lesser extent than the user network.

6.1.7 User network characteristics

In the user network graph, we see an average degree of 1.861 and a network diameter of 10. We find pockets of active users within the topology, these dense areas most likely represent Bitcoin exchanges. The network is relatively sparse given that many users will perhaps trade only once or twice, but in this instance this could be because of the timescale chosen for the sample.

6.2 Stage 2: Logistic Regression

6.2.1 Definition of Logistic Regression

Logistic regression is a regression model is well known the machine learning world for performing classification tasks. A logistic function is used to predict whether or not a binary y value is true or false. Given a set of features x_i, and a label $y_i \in \{0, 1\}$, logistic regression interprets the probability that the label is in one class as a logistic function of a linear combination of the features [6]:

$$f_i(\theta) = p(y_i = 1|x) = \frac{1}{1 + \exp(-\theta^T x)}$$

Figure 32: Mathematical definition of Logistic Regression

As the input to a logistic function gets larger and positive, it becomes closer to 1, whereas as the input becomes larger and negative, it becomes closer to 0 [6]. An intercept term is added by appending a column of 1's to the features with the aim to optimise:

$$\min_{\theta} \sum_{i=1}^{n} f_i(\theta) + \lambda_1 |\theta|_1 + \lambda_2 |\theta|_2^2$$

Figure 33: Mathematical definition of Logistic Regression showing the intercept term.

where λ_1 is the L1_penalty and λ_2 is the L2_penalty.

6.2.2 Logistic Regression settings

There are just over 114,539 transactions with which to train the logistic regression model for the services category. After initial experimentation we found that introducing a penalty to the model did not improve results at all, therefore we left the penalty at 0. We attempted with a penalty of 5, 10, 20 and 50. We define the maximum iterations at 50 and specify the following five features:

feature1	feature2	feature3	feature4	feature5
PageRank	in_degree	out_degree	avg_value	transaction_count

Table 8: Initial Logistic Regression features.

Model parameter	**Value**
Training set size	114,539
Number of classes	2
Number of feature columns	5
Number of unpacked features	5
Number of coefficients	6

Table 9: Logistic Regression model settings

6.2.3 Logistic Regression results

Iteration	Passes	Elapsed Time	Training-accuracy	Validation-accuracy
1	2	0.111053	0.521111	0.541994
2	3	0.177214	0.668216	0.661546
3	4	0.241410	0.678711	0.673902
4	5	0.302273	0.688803	0.683921
5	6	0.400226	0.692272	0.689598
6	7	0.543189	0.692124	0.689097
11	12	0.836860	0.692884	0.689932
25	26	1.948253	0.692884	0.689932

Table 10: Training Logistic Regression model showing training vs validation accuracy at each iteration.

6.3 Stage 3: Support Vector Machines (SVM)

6.3.1 Definition of Support Vector Machines (SVM)

Please note the definitions and the example in this section are taken from [13]. They are shown here for completeness.

Support Vector Machine (SVM) is an optimisation method that performs classification tasks by constructing hyperplanes into a multidimensional space that separates cases of different class labels. SVM can be used in both regression and classification tasks and can also take multiple, continuous and categorical variables [13]. Support Vector Machines are based on the concept of decision planes that define decision boundaries. A decision plane is one that separates between a set of objects having different class memberships. The objects belong either to class green or red. The separating line defines a boundary on the right side of which all objects are green and to the left of which all objects are red. Any new object (white circle) falling to the right is classified as green (or classified as red should it fall to the left of the separating line) [13].

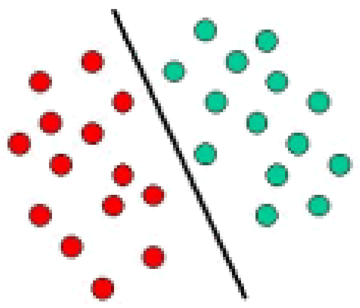


Figure 34: Define the decision plane [13].

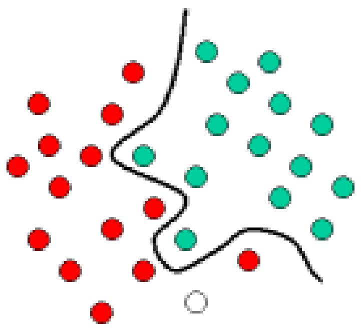


Figure 35: Classify white dot [13].

The above is a classic example of a linear classifier, i.e., a classifier that separates a set of objects into their respective groups (green and red in this case) with a line. Most classification tasks, however, are not that simple, and often more complex structures are needed in order to make an optimal separation, i.e., correctly classify new objects (test cases) on the basis of the examples that are available (train cases) [13]. This situation is depicted in the illustration below. Compared to the previous schematic, it is clear that a full separation of the green and red objects would require a curve (which is more complex than a line) [13]. Classification tasks based on drawing separating lines to distinguish between objects of different class memberships are known as hyperplane classifiers. Support Vector Machines are particularly suited to handle such tasks [13]. There are cases when finding an optimum hyper plane is not possible. Occasionally it is possible to overcome this by transforming the data into a higher dimensional space. We can use a kernel trick to compute the dot products in the higher dimensional space and use those to find a hyperplane.

In the image below we see the original objects (left side of the schematic) mapped, i.e., rearranged, using a set of mathematical functions, known as kernels. The process of rearranging the objects is known as mapping (transformation). Note that in this new setting, the mapped objects (right side of the schematic) is linearly separable and, thus, instead of constructing the complex curve (left schematic), all we have to do is to find an optimal line that can separate the green and the red objects [13]. We therefore find the hyperplane that maximises the distances to the nearest point in each class.

In this paper we implement a linear C-SVM (SVC) using GraphLab Create. In this model, given a set of features x_i, and a label $y_i \in \{0,1\}$ the linear SVM minimizes the loss function:

$$fi() = max(1Tx, 0)$$

Figure 36: Mathematical definition of SVM minimising the loss function

As with other models, an intercept term is added by appending a column of 1's to the features. The composite objective being optimized for is the following:

$$\min_{\theta} \lambda \sum_{i=1}^{n} f_i(\theta) + ||\theta||_2^2$$

Figure 37: Mathematical definition showing the penalty parameter

where λ is the penalty parameter (the C in the C-SVM) that determines the weight in the loss function towards the regulariser. The larger the value of λ, the more is the weight given to the mis-classification loss. GraphLab Create solves the Linear-SVM formulation by approximating the hinge-loss with a smooth function [26].

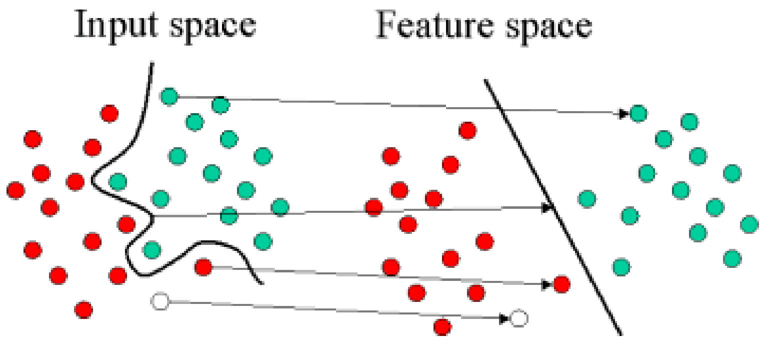


Figure 38: Optimal decision plane [13].

6.3.2 Support Vector Machine settings

The Support Vector Machine implementation uses Limited-memory BFGS (L-BFGS or LM-BFGS) for optimisation. We use the same baseline features as with Logistic Regression.

feature1	feature2	feature3	feature4	feature5
PageRank	in_degree	out_degree	avg_value	transaction_count

Table 11: Initial Support Vector Machine features.

Model parameter	**Value**
Training set size	114,090
Number of classes	2
Number of feature columns	5
Number of unpacked features	5
Number of coefficients	6

Table 12: Support Vector Machine model summary

6.3.3 Support Vector Machine results

The SVM model was able to successfully able to classify 60.6% of predictions. The model achieved an F1 score of 0.727, recall of 0.907 and precision 0.907.

The model aimed to predict a binary output of either True or False as to whether or not the transaction is likely to be interacting with the 'services' category. A target value was set using the column 'is services' which defined during the pre-processing phase. The model then used this labelled data-set to classify each transaction [24]. In a similar fashion to the Logistic Regression model a 5% validation was taken from the training set and used validate the training of the model at each iteration.

Iteration	Passes	Step size	Elapsed Time	Training-accuracy	Validation-accuracy
1	4	0.000044	0.060040	0.515070	0.521307
2	6	1.000000	0.103437	0.515070	0.521307
3	7	1.000000	0.131290	0.637579	0.643840
4	8	1.000000	0.161072	0.542631	0.539380
5	14	1.161132	0.251702	0.695629	0.701874
6	15	1.161132	0.279608	0.579126	0.580501
11	28	1.236735	0.513037	0.484930	0.478693
25	77	1.557945	1.309792	0.491355	0.499751
50	147	0.001807	2.582619	0.657089	0.667385

Table 13: Training SVM model showing training vs validation accuracy at each iteration.

7 Evaluation

7.1 Stage 4: Logistic Regression

The Logistic Regression model was able to successfully able to classify 68% of predictions. The model achieved an F1 score of 0.731, recall of 0.89 and precision 0.62.

F1 Score	AUC	Recall	Precision	Log Loss	Accuracy
0.731	0.826	0.890	0.621	0.626	0.680

Table 14: Logistic Regression evaluation metrics.

True Positive	False Positive	True Negative	False Negative
10773	6538	6084	1345

Table 15: Logistic Regression confusion matrix.

The ROC curve returned by the logistic regression classifier has a reasonable amount of space under it, the true positive rate increases relatively quickly. With the mark false positive rate really kicking in at the the 70% mark. As discussed in the definition of the ROC curve, we would not expect to see a classifier just to 100% except in very rare examples.

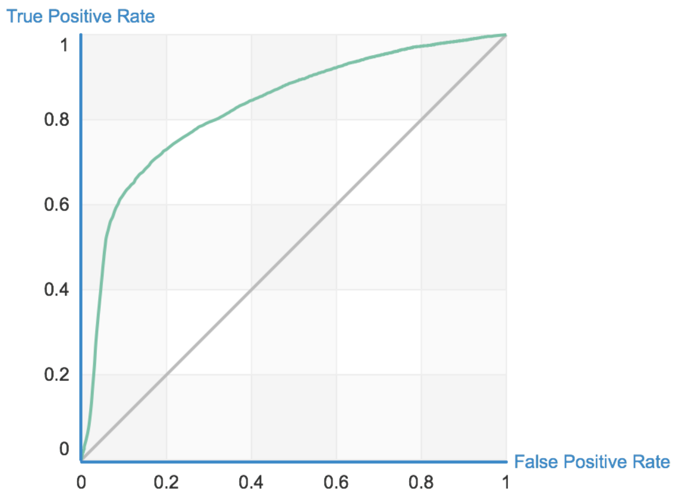


Figure 39: Logistic Regression model true positive vs false positive rate.

7.1.1 Logistic Regression feature sensitivity

Logistic regression can provide valuable insight about the relationships between the target and feature columns in your data, revealing why the model returns the predictions that it does. The coefficients (θ) are what the training procedure learns. Each model coefficient describes the expected change in the target variable associated with a unit change in the feature. The bias term indicates the ”inherent” or ”average” target value if all feature values were set to zero.

We can return the coefficients which can tell us the impact a feature may have when predicting value. The absolute value of the coefficient for each feature indicates the strength of the feature's association to the target variable, holding all other features constant. The sign on the coefficient (positive or negative) gives the direction of the association [13].

The standard error is the deviation on the estimated coefficient. For example, a coefficient of 1 with a standard error of 0.5 implies a standard deviation of 0.5 on the estimated values of the coefficients. Standard errors capture the confidence we have in the coefficients estimated during model training. Smaller standard errors implies more confidence in the value of the coefficients returned by the model [13].

Standard errors on coefficients are only available when solver=newton or when the default auto solver option chooses the newton method and if the number of examples in the training data is more than the number of coefficients. If standard errors cannot be estimated, a column of None values are returned.

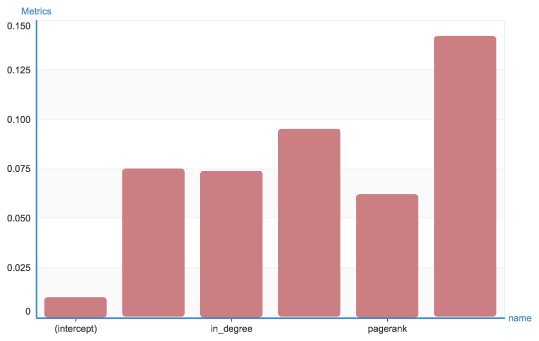


Figure 40: Logistic Regression standard error plot

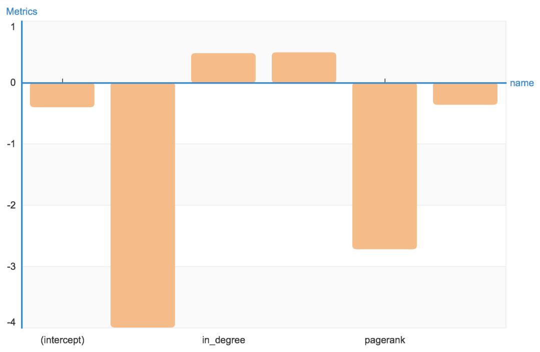


Figure 41: Feature absolute values.

As shown in the results of the coefficients, the PageRank and average value of the transaction contributed most to the models prediction of the correct outcome. We can see that transaction count made the smallest contribution to the model. Both in-degree and out-degree made positive contributions to the correct classification. We run a quick test that removing the PageRank and average value of transactions will not improve F1 the classification scores.

F1 Score	AUC	Recall	Precision	Log Loss	Accuracy
0.704	0.739	0.775	0.645	0.665	0.686

Table 16: Logistic Regression evaluation metrics after feature sensitivity adjustment.

The feature adjustments manage to improve true positive, false positive and true negative results significantly, however there is large increase in the number of false negatives that we see.

True Positive	False Positive	True Negative	False Negative
11134	3226	9348	6120

Table 17: Logistic Regression confusion matrix after feature sensitivity adjustment.

We next attempted to decrease the number of false negatives having adjusted our features. We introduce the $L1$ and $L2$ regularizes. The larger the $L2$ weight, the more the model coefficients shrink toward 0. We can therefore add bias into the model but also decrease variance, potentially leading to better predictions. In a similar fashion to the $L2$ penalty, the higher the $L1$ penalty, the more the estimated coefficients shrink toward 0. The $L1$ penalty, however, completely zeros out sufficiently small coefficients, automatically indicating features that are not useful for the model.

7.1.2 Logistic Regression adjusted results after analysis

Using a combination feature sensitivity analysis and penalty value to punish feature with low contributions to the model outcome, we were able to increase our results further.

feature1	feature2	feature3	feature4
PageRank	in_degree	out_degree	avg_value

Table 18: Adjusted Logistic Regression selected features.

Model parameter	**Value**
Training set size	114,539
Number of classes	2
Number of feature columns	4
Number of unpacked features	4
Number of coefficients	5

Table 19: Adjusted Logistic Regression model settings

The model settings remain largely similar with transaction count removed and $L1$ and $L2$ penalties of 0.95 added. The results show an increase across each metric showing the model has improved, however the increase was only a couple of percent, not as much as expected.

F1 Score	AUC	Recall	Precision	Log Loss	Accuracy
0.733	0.822	0.881	0.628	0.624	0.691

Table 20: Adjusted Logistic Regression evaluation metrics.

The confusion matrix shows better results in classifying true positive and true negative values, however the number of false positives and false negatives almost swaps. There are many more false negatives classified and far fewer false positives which would suggest that our model has become much more conservative given the penalties introduced.

True Positive	False Positive	True Negative	False Negative
12653	1707	7977	7491

Table 21: Adjusted Logistic Regression confusion matrix.

7.2 Stage 5: Support Vector Machine

The SVM model was able to successfully able to classify 60.6% of predictions. The model achieved an F1 score of 0.727, recall of 0.907 and precision 0.907.

As shown in the predictions table, the trained model was able to classify transactions that were likely to be trading within the services category with over 11,000 correct predictions. Equally so, the model correctly determined when a transaction was not likely to be associated with 'services' nodes with 5,762 predicted False values [24].

F1 Score	Recall	Precision	Accuracy
0.727	0.907	0.907	0.606

Table 22: Support Vector Machine evaluation metrics.

True Positive	False Positive	True Negative	False Negative
10632	1486	6569	6053

Table 23: Support Vector Machine confusion matrix.

7.2.1 Support Vector Machine feature sensitivity

The feature sensitivity analysis showed that the out-degree and average value per transaction were the two least valuable features. A number of tests were run to optimise the model further but each iteration generated worse results that the initial feature selection. We therefore do not present an adjusted model for Support Vector Machines.

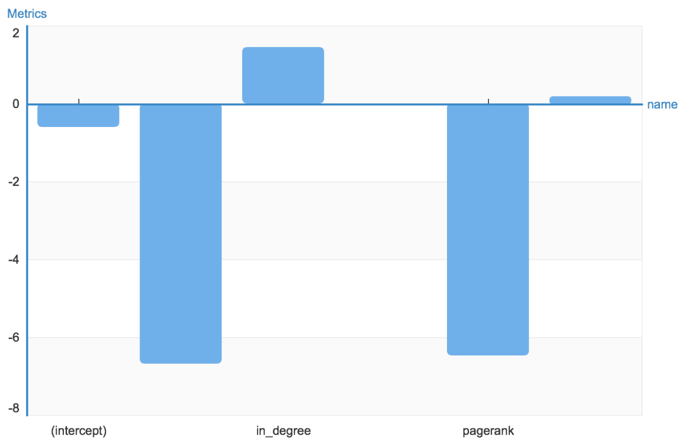


Figure 42: SVM feature absolute values.

8 Discussion

8.1 How do the results support the problem of this paper

We have described, discussed and demonstrated how the Bitcoin protocol can be used to identify trends in transaction behaviour. We prove that by using publicly available information we can predict whether or not a user transaction is likely to be associated with a certain category. The method implemented here is arguably harder to achieve because we do not maintain detailed user profiles, nor do we know personal details of users. In this paper we argue that commercial entities could potentially share information and therefore benefit from business intelligence. This assumption remains true, however what we show is that it is not necessary for businesses to share this information in the first place because we show that it can be done using network metrics and transactional behaviour already. Commercial adoption only means that it will become even easier to derive this information if businesses hold customer information, such as email address or delivery address.

The key result of this paper is showing that we can predict a whether a transaction will be associated with the services category with an F1 score of 0.733 with absolutely no personal information or any other kind of information except the network metrics and the number of transaction made by the address and their average value. This means there are opportunities for businesses to potentially discover new customers and avenues for generating sales.

One area that we did not set out to investigate is user privacy. This is because there has been plenty of work in this field. As a result of this research we can deduct that the monetary gains of business intelligence will come at a cost to user privacy. There is a significant threat that if business share customer information, we will most certainly see less anonymity in the network. By sharing personal information and using clustering and predictive techniques shown in this paper, it will become harder harder for anonymous transactions to take place. As we discuss early on in this paper, in the early years of Bitcoin, we can assume that both parties in a transaction are anonymous, but by making one entity know to the world, it becomes harder to hide the anonymous party.

8.2 Evidence of the project complexity and deliverables

This project has entailed a wide range of skill-sets. The data-set itself currently weighs in at 168GB on a local hard disk, taking at least 7 days to download and index the blockchain. The core client must run 24/7 in order to keep an up to date blockchain. The tools used to extract data from the blockchain are unsupported code base from Github which often means a significant amount of time testing the tools with no support. Extracting the sample of 62,000,000 transactions took 12 hours alone which then meant further time pre-processing. The information retrieval tasks required custom scripts for each source scraped as well as concatenation scripts to normalise the 6,000,000 tagged addresses. By experimenting with multiple tools for data analysis, both iPython and GraphLab Create proved a successful combination for analysing such a large data-set.

As well as technical complexities, the deliverables themselves were tough at times because, to the best of our knowledge, there is no comparison to this work. A significant amount of work had to be carried out to ensure that the supervised machine learning models were able to successfully classify transactions. Experiments were largely done with unsupervised techniques (k-means) to discover feature relationships.

8.3 Limitations of the project

There are two notable limitations to this project. Firstly, if users within the network transact in a manor that involves a completely new public address for every single transaction then it become much harder to cluster entities. Work carried out by [12] shows that at the moment most users to do not follow this trend, but if they were to do so then they would increase their chances of remaining anonymous and therefore make it harder for analytics companies to track their transaction history. The second limitation is the number of dormant addresses in the network increase the amount of noise in the training data and will therefore reduce the quality of the results. To overcome this in this paper we do not include these transactions and remove any address that is not active within the network. The final limitation would be if companies made their public address secret and could therefore not be identified using information retrieval techniques, this would remove the ground truth data-set and therefore make it harder to train an up to date model.

8.4 Contributions to the field

The work carried out in this project combines existing techniques of clustering within the Bitcoin network but also implements supervised learning techniques on network metrics which is yet to be explored within the field. We achieve this by providing a solid ground truth data-set and as such are able to generate promising results in proving insight for business intelligence purposes. By providing predictions on the type of transactions an address is likely to interact with on the transaction graph we can suggest clusters of users that may be interested in a type of service. This work also raises the question of anonymity within the network while showing evidence of the changing use case of this crypto-currency.

8.5 Future work

There are two main elements to this future work section. Firstly the technical implementation. As discussed the data-set at times was tricky to manage. Going forward we install the Bitcoin client on an AWS instance. Over the course of this project it did not seem appropriate to upload 168GB of data to an AWS instance. If we had the client and the data-set on a virtual machine we could have scaled the instance when needed which would have reduced the computation time when pre-processing the data. Next we would have been able to write an api to sit on the AWS instance which would have made it easier to query the data-set without having to use third party services. We found that third party services often had restrictions on usage and by implementing our own api, we could have generated more insightful address statistics.

We would also like to turn this project into a online learning implementation by regularly optimising the model and as well as continuously updating the feature set. This would be implemented by running a crawler to query the api described above which would then return updated address statistics. This would improve accuracy by providing a near real-time snapshot of addresses used in transactions.

8.6 Code and documentation

https://github.com/archienorman11/thesis-bitcoin-clustering

References

[1] *Are User Identification Networks the Future of Commercial Bitcoin Transactions?* 2016. URL: `https://freedom-to-tinker.com/blog/jaredho/are-user-identification-networks-the-future-of-commercial-bitcoin-transactions/` (visited on 08/26/2016).

[2] Anton I Badev and Matthew Chen. “Bitcoin: Technical Background and Data Analysis”. In: (2014).

[3] Bitcoingraph. 2016. URL: `https://github.com/behas/bitcoingraph` (visited on 08/26/2016).

[4] Bitcoin.org. *Developer Guide - Bitcoin*. 2016. URL: `https://bitcoin.org/en/developer-guide#block-chain` (visited on 01/04/2016).

[5] Ivan Brugere. “Bitcoin Transaction Network Dataset”. In: *Data Set* (2013).

[6] Joel Grus. *Data Science from Scratch: First Principles with Python*. ” O’Reilly Media, Inc.”, 2015.

[7] Jason Hirshman, Yifei Huang, and Stephen Macke. *Unsupervised Approaches to Detecting Anomalous Behavior in the Bitcoin Transaction Network*.

[8] Joey Jegonzal. *PowerGraph: A framework for large-scale machine learning and graph computation.* 2012.

[9] Dániel Kondor et al. “Do the rich get richer? An empirical analysis of the Bitcoin transaction network”. In: *PloS one* 9.2 (2014), e86197.

[10] Dániel Kondor et al. “Inferring the interplay between network structure and market effects in Bitcoin”. In: *New Journal of Physics* 16.12 (2014), p. 125003.

[11] Yucheng Low et al. “Graphlab: A new framework for parallel machine learning”. In: *arXiv preprint arXiv:1408.2041* (2014).

[12] Sarah Meiklejohn et al. “A fistful of bitcoins: characterizing payments among men with no names”. In: *Proceedings of the 2013 conference on Internet measurement conference*. ACM. 2013, pp. 127–140.

[13] N/A. *Support Vector Machines (SVM) Introductory Overview*. 2016. URL: `http://www.statsoft.com/textbook/support-vector-machines` (visited on 09/01/2016).

[14] Satoshi Nakamoto. “Bitcoin: A peer-to-peer electronic cash system”. In: (2008).

[15] Phillip Thai Pham and Steven Lee. “Anomaly Detection in Bitcoin Network Using Unsupervised Learning Methods”. In: ().

[16] David Martin Powers. “Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation”. In: (2011).

[17] Fergal Reid and Martin Harrigan. “An analysis of anonymity in the bitcoin system”. In: *Security and privacy in social networks*. Springer, 2013, pp. 197–223.

[18] Dorit Ron and Adi Shamir. “Quantitative analysis of the full bitcoin transaction graph”. In: *International Conference on Financial Cryptography and Data Security*. Springer. 2013, pp. 6–24.

[19] Richelle Ross et al. *Bitcoin Price Index - Real-time Bitcoin Price Charts*. 2015. URL: `http://www.coindesk.com/price/` (visited on 01/04/2016).

[20] Devavrat Shah and Kang Zhang. “Bayesian regression and Bitcoin”. In: *Communication, Control, and Computing (Allerton), 2014 52nd Annual Allerton Conference on*. IEEE. 2014, pp. 409–414.

[21] Michele Spagnuolo, Federico Maggi, and Stefano Zanero. “Bitiodine: Extracting intelligence from the bitcoin network”. In: *Financial Cryptography and Data Security*. Springer, 2014, pp. 457–468.

[22] Turi. 2016. URL: `https://turi.com/products/create/docs/graphlab.toolkits.evaluation.html#classifier-metrics` (visited on 08/26/2016).

[23] Turi. 2016. URL: `https://turi.com/products/create/docs/generated/graphlab.kcore.KcoreModel.html` (visited on 08/26/2016).

[24] Turi. 2016. URL: `https://turi.com/learn/userguide/evaluation/classification.html#precision_recall` (visited on 08/26/2016).

[25] Eric W. Weisstein.

[26] Jian Zhang et al. “Modified logistic regression: An approximation to SVM and its applications in large-scale text categorization”. In: *ICML*. Vol. 3. 2003, pp. 888–895.

Appendices

A Bitcoin address mining

A.1 https://blockchain.info/de/tags

```python
#!/usr/bin/env python

from bs4 import BeautifulSoup
import unicodecsv as csv
import requests

def scrape_page(writer, list_key, offset):
    r = requests.get('https://blockchain.info/de/tags',
                     params={'filter': list_key, 'offset': offset})
    soup = BeautifulSoup(r.text, 'html.parser')
    for tr in soup.tbody.find_all('tr'):
        if tr.img['src'].endswith('green_tick.png'):
            tds = tr.find_all('td')
            address = tds[0].get_text()
            tag = tds[1].get_text()
            writer.writerow([address, tag])
    page_links = soup.find_all('div', class_='pagination')[0].find_all('a')
    last_offset = int(page_links[-2]['href'].rpartition('=')[2])
    print(last_offset)
    return last_offset


with open('output/blockinfo.csv', 'w') as identities_file:
    identities_writer = csv.writer(identities_file)
    identities_writer.writerow(['address', 'tag'])
    for list_key in [8, 16, 2, 4]:
        last_offset = scrape_page(identities_writer, list_key, 0)
        for off in range(200, last_offset + 1, 200):
            print('(list {}) scrape identity {} of {}'.format(list_key, \
            off, last_offset))
            scrape_page(identities_writer, list_key, off)
```

A.2 http://bitcoinwhoswho.com

```python
#!/usr/bin/env python

from bs4 import BeautifulSoup
import requests
import json

addresses = []

def lets_get_scraping(url, id):
    r = requests.get(url)
    soup = BeautifulSoup(r.text, 'html.parser')

    for tr in soup.tbody.find_all('tr'):
        for strong_tag in tr.find_all('strong'):
            if id == 1:
                temp = {strong_tag.get_text(): 1}
                addresses.append(temp)
            elif id == 2:
                temp = {strong_tag.get_text(): 2}
                addresses.append(temp)
            elif id == 3:
                temp = {strong_tag.get_text(): 3}
                addresses.append(temp)
            elif id == 4:
                temp = {strong_tag.get_text(): 4}
                addresses.append(temp)
    return addresses


for i in range(17):
    gambling = 'http://bitcoinwhoswho.com/search/index/index/keyword/gambling/ \
    page/{0}'.format(i)
    lets_get_scraping(gambling, id=1)

print('Completed: Gambling')

for i in range(2):
    charity = 'http://bitcoinwhoswho.com/search/index/index/keyword/charity \
    /page/{0}'.format(i)
    lets_get_scraping(charity, id=2)

for i in range(34):
    charity = 'http://bitcoinwhoswho.com/search/index/index/keyword/donate/ \
    page/{0}'.format(i)
    lets_get_scraping(charity, id=2)

print('Completed: Charity')

for i in range(17):
    finance = 'http://bitcoinwhoswho.com/search/index/index/keyword/finance/ \
    page/{0}'.format(i)
    lets_get_scraping(finance, id=3)

print('Completed: Finance')

for i in range(162):
    exchange = 'http://bitcoinwhoswho.com/search/index/index/keyword/exchange/ \
    page/{0}'.format(i)
    lets_get_scraping(exchange, id=3)

print('Completed: Exchanges')

print(len(addresses))

with open('output/whoiswho.json', 'w') as jsonfile:
    json.dump(addresses, jsonfile, encoding='utf8')
```

A.3 https://www.walletexplorer.com

```python
#!/usr/bin/env python

from bs4 import BeautifulSoup
import requests
import json
import re
import pandas as pd

addresses = []

# wallet_links = open('data/wallet.csv')
# wallet_links = csv.reader(wallet_links)

df = pd.read_csv('data/wallet.csv')
df = df[['Old/Historic_link', 'Gambling_link', 'Services_link', 'Pools_link', \
'Exchanges_link']]
df_gambling = df['Gambling_link']
df_services = df['Services_link']
df_pools = df['Pools_link']
df_exchanges = df['Exchanges_link']
df_gambling = df_gambling.dropna()
df_services = df_services.dropna()
df_pools = df_pools.dropna()
df_exchanges = df_exchanges.dropna()

df_gambling = pd.DataFrame(df_gambling)
df_services = pd.DataFrame(df_services)
df_pools = pd.DataFrame(df_pools)
df_exchanges = pd.DataFrame(df_exchanges)

df_gambling.columns = ['link']
df_services.columns = ['link']
df_pools.columns = ['link']
df_exchanges.columns = ['link']

def lets_get_scraping(stem, id, num):
    for i in range(1, num):
        url = stem
        url = url + '/addresses?page={0}'.format(i)
        print(url)
        r = requests.get(url)
        soup = BeautifulSoup(r.text, 'html.parser')
        try:
            for tr in soup.table.find_all('tr'):
                if tr:
                    for td in tr.find_all('td'):
                        if td:
                            for a in td.find_all('a'):
                                if a:
                                    if id == 1:
                                        temp = {a.get_text(): 1}
                                        addresses.append(temp)
                                    elif id == 6:
                                        temp = {a.get_text(): 6}
                                        addresses.append(temp)
                                    elif id == 3:
                                        temp = {a.get_text(): 3}
                                        addresses.append(temp)
                                    elif id == 4:
                                        temp = {a.get_text(): 4}
                                        addresses.append(temp)
                else:
                    print('error')
        except:
            print('error')
    return addresses


for index, row in df_gambling.iterrows():
    temp = row['link']
    r = requests.get(temp)
    soup = BeautifulSoup(r.text, 'html.parser')
    for foo in soup.find_all('div', attrs={'class': 'paging'}):
        num = foo.text
    pages = re.search('(?<=/\s)\d+', num).group(0)
    lets_get_scraping(row['link'], id=1, num=int(pages))

print('Completed: Gambling')

for index, row in df_services.iterrows():
    temp = row['link']
    r = requests.get(temp)
    soup = BeautifulSoup(r.text, 'html.parser')
    for foo in soup.find_all('div', attrs={'class': 'paging'}):
        num = foo.text
    pages = re.search('(?<=/\s)\d+', num).group(0)

    lets_get_scraping(row['link'], id=4, num=int(pages))

print('Completed: Services')

for index, row in df_pools.iterrows():
    temp = row['link']
    r = requests.get(temp)
    soup = BeautifulSoup(r.text, 'html.parser')
    for foo in soup.find_all('div', attrs={'class': 'paging'}):
        num = foo.text
    pages = re.search('(?<=/\s)\d+', num).group(0)

    lets_get_scraping(row['link'], id=6, num=int(pages))

print('Completed: Pools')

for index, row in df_exchanges.iterrows():
    temp = row['link']
    r = requests.get(temp)
    soup = BeautifulSoup(r.text, 'html.parser')
    for foo in soup.find_all('div', attrs={'class': 'paging'}):
        num = foo.text
    pages = re.search('(?<=/\s)\d+', num).group(0)

    lets_get_scraping(row['link'], id=3, num=int(pages))

print('Completed: Exchanges')

print(addresses)
```

```python
print(len(addresses))

with open('output/pools.json', 'w') as jsonfile:
    json.dump(addresses, jsonfile, encoding='utf8')
```

A.4 Program to manually sort addresses based on tag

```python
#!/usr/bin/env python

import csv
import collections
import json

cnt = collections.Counter()

sort = open('output/sort.csv')
gambling_sample = open('output/gambling.csv')
finance_sample = open('output/finance.csv')
services_sample = open('output/services.csv')
charity_sample = open('output/charity.csv')
junk_sample = open('output/junk.csv')

sort = csv.reader(sort)
gambling_sample = csv.reader(gambling_sample)
finance_sample = csv.reader(finance_sample)
services_sample = csv.reader(services_sample)
charity_sample = csv.reader(charity_sample)
junk_sample = csv.reader(junk_sample)

sort = list(sort)

gamble = []
for j in gambling_sample:
    for val in j:
        gamble.append(val)

finance = []
for j in finance_sample:
    for val in j:
        finance.append(val)

services = []
for j in services_sample:
    for val in j:
        services.append(val)

charity = []
for j in charity_sample:
    for val in j:
        charity.append(val)

junk = []
for j in junk_sample:
    for val in j:
        junk.append(val)

```

```
def update_all(name, id):
        for b in dict_list:
            if b['source'] == name:
                b['tag'] = id

def update_from_lists():
    for j in dict_list:
        for k in gamble:
            if str(k) in str(j['source']):
                update_all(j['source'], 1)
        for f in services:
            if str(f) in str(j['source']):
                update_all(j['source'], 4)
        for l in finance:
            if str(l) in str(j['source']):
                update_all(j['source'], 3)
        for t in charity:
            if str(t) in str(j['source']):
                update_all(j['source'], 2)
        for v in junk:
            if str(v) in str(j['source']):
                update_all(j['source'], 5)


def categorise_to_list():
    for item in dict_list:
        if item['tag'] == 0:
            answer = int(input("%s - (1=Gambling, 2=Charity, \
             3=Finance, 4=Services, 5=Scam/Junk): " % item['source']))
            if answer == 1:
                with open('gambling.csv','a') as g:
                    g.write(item['source'] + "\n")
                    update_all(item['source'], 1)
            elif answer == 2:
                with open('charity.csv','a') as c:
                    c.write(item['source'] + "\n")
                    update_all(item['source'], 2)
            elif answer == 3:
                with open('finance.csv','a') as f:
                    f.write(item['source'] + "\n")
                    update_all(item['source'], 3)
            elif answer == 4:
                with open('services.csv','a') as s:
                    s.write(item['source'] + "\n")
                    update_all(item['source'], 4)
            elif answer == 5:
                with open('junk.csv','a') as j:
                    j.write(item['source'] + "\n")
                    update_all(item['source'], 5)
            elif answer == 10:
                get_stats()
            elif answer == 100:
                create_json_output()
                break
            elif answer == 1000:
                break

        else:
```

```python
            print('done')

def get_stats():
    un_tagged = 0
    tagged = 0
    for p in dict_list:
        if p['tag'] == 0:
            un_tagged += 1
            cnt[p['source']] += 1
        else:
            tagged += 1

    print(cnt.most_common(30))
    print(tagged)
    print(un_tagged)

def create_json_output():
    temp_list = []
    for i in dict_list:
        temp = {i['address']: i['tag']}
        temp_list.append(temp)
    with open('output/output.json', 'w') as jsonfile:
        json.dump(temp_list, jsonfile)

dict_list = []
for i in sort:
    dict = {"address": i[0], "source": i[1], "tag": 0}
    dict_list.append(dict)

update_from_lists()
categorise_to_list()
```

A.5 Script to look-up wallet statistics of each each address in the transaction list

```python
#!/usr/bin/env python

import json
import pandas as pd
from blockcypher import get_address_overview


transactions = '../../notebooks/output/data_addresses'
addresses = []

df = pd.read_csv(transactions)

for index, row in df.iterrows():
    # print(get_address_overview('1DEP8i3QJCsomS4BSMY2RpU1upv62aGvhD'))
    data = (get_address_overview(row['address']))
    final_n_tx = data['final_n_tx']
    n_tx = data['n_tx']
    unconfirmed_balance = data['unconfirmed_balance']
    final_balance = data['final_balance']
    balance = data['balance']
    total_sent = data['total_sent']
    address = data['address']
    total_received = data['total_received']
    print(address, total_received, balance, final_n_tx, total_sent)
    temp = {
        "address": address,
        "final_n_tx": final_n_tx,
        "n_tx": n_tx,
        "unconfirmed_balance": unconfirmed_balance,
        "final_balance": final_balance,
        "balance": balance,
        "total_sent": total_sent,
        "total_received": total_received,
    }
    addresses.append(temp)

print(len(addresses))

with open('output/cypher_stats.json', 'w') as jsonfile:
    json.dump(addresses, jsonfile, encoding='utf8')
```

B Pre-processing

B.1 Input data-set

```
+------------------------------------+------------------+------------------+-------------------+---------------+
|              address               | received_charity | received_finance | received_gambling | received_junk |
+------------------------------------+------------------+------------------+-------------------+---------------+
| 111ZVmQMacfXiEpm9hrgWTjzbvHFRmiC1  |       0.0        |       0.0        |        0.0        |      0.0      |
| 111rk7dfDvQLx5FtywePeNzvDVGTJhKrz  |       0.0        |       1.0        |        0.0        |      0.0      |
| 1123JJPvduSJe6JSiab2zqcRU7Rkdjf3ZK |       0.0        |       1.0        |        0.0        |      0.0      |
| 1123xwCN4XaF3yLztPYwWMVXaCQcYR1aTF |       0.0        |       0.0        |        0.0        |      0.0      |
| 1124K1jJmvbazM9RS1FRVRGPTe86b11Yok |       0.0        |       1.0        |        0.0        |      0.0      |
+------------------------------------+------------------+------------------+-------------------+---------------+
+----------------+-------------------+--------------+--------------+---------------+-----------+------------+
| received_pools | received_services | sent_charity | sent_finance | sent_gambling | sent_junk | sent_pools |
+----------------+-------------------+--------------+--------------+---------------+-----------+------------+
|      0.0       |        0.0        |     0.0      |     2.0      |      0.0      |    0.0    |    0.0     |
|      0.0       |        0.0        |     0.0      |     2.0      |      0.0      |    0.0    |    0.0     |
|      0.0       |        0.0        |     0.0      |     11.0     |      0.0      |    0.0    |    0.0     |
|      0.0       |        0.0        |     0.0      |     2.0      |      0.0      |    0.0    |    0.0     |
|      0.0       |        0.0        |     0.0      |     4.0      |      0.0      |    0.0    |    0.0     |
+----------------+-------------------+--------------+--------------+---------------+-----------+------------+
+---------------+-------------------+-------------+----------+------------+------------+------------+
| sent_services | transaction_count |    value    | category | is_gambler | is_charity | is_finance |
+---------------+-------------------+-------------+----------+------------+------------+------------+
|      0.0      |         2         |  0.02000108 |    6     |     0      |     0      |     1      |
|      0.0      |         3         | 39.86275797 |    9     |     0      |     0      |     1      |
|      0.0      |         12        |  0.56035218 |    36    |     0      |     0      |     1      |
|      0.0      |         2         |     0.12    |    6     |     0      |     0      |     1      |
|      0.0      |         5         | 23.22186286 |    15    |     0      |     0      |     1      |
+---------------+-------------------+-------------+----------+------------+------------+------------+
+---------+----------+-------------+----------------+-------------------+-----------+------------+
| is_junk | is_pools | is_services |    pagerank    |       delta       | in_degree | out_degree |
+---------+----------+-------------+----------------+-------------------+-----------+------------+
|    0    |    0     |      0      | 0.83549177044  | 1.41351756411e-07 |     2     |     0      |
|    0    |    0     |      0      | 0.277501081498 |        0.0        |     2     |     1      |
|    0    |    0     |      0      | 0.67788978633  | 1.31144808624e-06 |     11    |     1      |
|    0    |    0     |      0      |   0.41020625   |        0.0        |     2     |     0      |
|    0    |    0     |      0      | 1.46046619159  | 9.32111184815e-05 |     4     |     1      |
+---------+----------+-------------+----------------+-------------------+-----------+------------+
+--------------+---------+-------------+---------------+----------------+
| component_id | core_id |  avg_value  | in_degree_std | out_degree_std |
+--------------+---------+-------------+---------------+----------------+
|    107928    |    1    |  0.01000054 | 93.9320422623 |  97.912067833  |
|    107928    |    2    | 13.28758599 | 93.9320422623 |  97.912067833  |
|    107928    |    9    | 0.046696015 | 93.9320422623 |  97.912067833  |
|    837894    |    2    |     0.06    | 93.9320422623 |  97.912067833  |
|    107928    |    2    | 4.644372572 | 93.9320422623 |  97.912067833  |
+--------------+---------+-------------+---------------+----------------+
```

Figure 43: Sample of supervised learning input data-set

B.2 Category breakdown

Xapo.com	BitPay.com
AlphaBayMarket	BitoEX.com
NucleusMarket	CoinPayments.net
Cubits.com	BitcoinFog
BTCJam.com	Cryptonator.com
HaoBTC.com	HolyTransaction.com
CoinKite.com	HelixMixer
FaucetBOX.com	CoinJar.com
Cryptopay.me (old)	OkLink.com
BitcoinWallet.com	Purse.io
ePay.info	Loanbase.com
GermanPlazaMarket	MoonBit.co.in
CryptoStocks.com	StrongCoin.com-fee
CoinApult.com	Paymium.com
Genesis-Mining.com	ChangeTip.com
Bitbond.com	DoctorDMarket
GoCelery.com	BTCPop.co
BTCLend.org	CoinURL.com
CoinBox.me	BitNZ.com
CoinWorker.com	WatchMyBit.com
BitLaunder.com	Vic-Socks.to
SecureVPN.to	Bylls.com
BitClix.com	GreenRoadMarket

Table 24: Sample sources for the services category

```
print(pools[['PageRank', 'delta', 'in-degree', 'out-degree', \
'avg_value', 'value', 'transaction_count']].describe())
```

metric	PageRank	delta	in_degree	out_degree	avg_value	value	transaction_count
count	251.000	2.510	251.000	251.000	251.000	251.000	251.000
mean	0.407	2.046	12.549	10.721	7.255	3633.886	23.270
std	0.697	7.833	24.113	67.553	44.010	56499.497	88.621
min	0.150	0.000	2.000	0.000	0.004	0.020	2.000
25%	0.157	0.000	3.000	0.000	0.045	0.282	5.000
50%	0.166	0.000	6.000	2.000	0.403	4.557	9.000
75%	0.290	2.757	12.500	7.500	3.276	46.336	22.000
max	6.583	5.559	303.000	1060.000	656.774	895183.259	1363.000

Table 25: Transactions summary for pools category

```
print(finance[['PageRank', 'delta', 'in-degree', 'out-degree', \
'avg_value', 'value', 'transaction_count']].describe())
```

metric	PageRank	delta	in_degree	out_degree	avg_value	value	transaction_count
count	60704.000	6.070	60704.000	60704.000	60704.000	6.070	60704.000
mean	0.850	5.504	10.317	7.989	10.188	2.124	18.497
std	6.041	1.104	114.616	118.843	40.150	6.538	228.762
min	0.150	0.000	1.000	0.000	0.000	2.053	2.000
25%	0.227	2.295	2.000	0.000	0.073	3.505	3.000
50%	0.393	2.574	4.000	1.000	1.397	1.103	6.000
75%	0.671	2.144	8.000	5.000	7.485	8.555	15.000
max	666.600	2.380	24300.000	26539.000	5450.126	1.514	50839.000

Table 26: Transactions summary for finance category

```
print(services[['PageRank', 'delta', 'in-degree', 'out-degree', \
'avg_value', 'value', 'transaction_count']].describe())
```

metric	PageRank	delta	in_degree	out_degree	avg_value	value	transaction_count
count	72674.000	7.267	72674.000	72674.000	72674.000	72674.000	72674.000
mean	0.355	1.112	7.326	11.607	0.841	12.883	19.281
std	1.243	1.611	15.051	216.782	23.566	259.363	271.817
min	0.150	0.000	2.000	0.000	0.000	0.000	2.000
25%	0.152	4.113	2.000	0.000	0.000	0.000	3.000
50%	0.160	1.017	4.000	0.000	0.002	0.014	5.000
75%	0.304	1.432	8.000	1.000	0.045	0.305	10.000
max	161.016	3.454	1736.000	24936.000	5716.656	34299.937	49926.000

Table 27: Transactions summary for services category

```
print(gambling[['PageRank', 'delta', 'in-degree', 'out-degree', \
'avg_value', 'value', 'transaction_count']].describe())
```

metric	PageRank	delta	in_degree	out_degree	avg_value	value	transaction_count
count	16630.000	1.663	16630.000	16630.000	16630.000	16630.000	16630.000
mean	0.694	8.204	9.258	5.176	0.607	5.924	15.173
std	3.636	3.122	341.376	96.340	2.881	49.083	422.991
min	0.150	0.000	1.000	0.000	0.000	0.000	2.000
25%	0.277	0.000	2.000	0.000	0.006	0.020	3.000
50%	0.405	9.080	2.000	1.000	0.010	0.040	4.000
75%	0.587	1.223	5.000	3.000	0.085	0.574	8.000
max	303.661	2.802	42750.000	8934.000	129.398	4160.338	50120.000

Table 28: Transactions summary for gambling category