Predicting Bitcoin Transactions with Network Analysis

Gabriel Bianconi (bianconi@stanford.edu)
Mahesh Agrawal (mahesha@stanford.edu)

Fall 2017

1 Abstract

Bitcoin has recently soared in popularity, attracting the attention of individuals, corporations, and governments alike. The aim of this paper is to predict the future number of transactions in the Bitcoin network based on the current network architecture. The task is similar to link prediction, but instead computes the expected number of edges in the network in a given time interval. We explored two major approaches: using handcrafted topological and non-topological features, and learning features in an unsupervised approach leveraging `node2vec`. We found that models learning over these features did not outperform a baseline regression based on node and edge counts, and discuss possible reasons why that was the case.

2 Introduction

Bitcoin has captured the interest of governments, corporations, financial institutions, and consumers. Over the past few years, the number of transactions grew vastly. While its adoption benefits the ecosystem, its fast growth also led to many issues. For example, the transaction costs have increased while transaction confirmation times deteriorated. For these reasons, predicting the number of transactions in a future time period could help the community prepare for and handle the network growth and its side effects. We explore this problem from the perspective of network analysis.

The Bitcoin network is composed of a number of wallet addresses. Each of these nodes are used to send and receive the currency. Each time a coin is sent from a wallet to another, a transaction is recorded in the network. The nodes are connected by directed edges (from sender to receiver) representing their involvement in a transaction. In addition, wallets can conduct multiple transactions with each other over time. These properties make the Bitcoin network a directed multigraph.

In this paper, we seek to predict the total number of transactions in a given time period given the state of the Bitcoin network in the previous time period. This is essentially the canonical link prediction problem but at the macro network level. Instead of predicting individual new links that will appear in the network in a subsequent time, we predict the total number of new links that will be created in the network. While much research exists for predicting individual links, there is little work dedicated to predicting the total number of new links created. This problem is also different from the usual link prediction problem in that the number of nodes in the Bitcoin network is constantly growing since new wallets are being created everyday, whereas the traditional link prediction tasks generally looks at predicting new links between existing nodes.

In particular, given all the transactions that occurred in the Bitcoin network in a given interval, our goal is to predict the number of transactions in the following interval. We explored two approaches for this problem. In the first experiment, we extracted various handcrafted topological and non-topological features from the network using 24 months of data and then train different machine learning models. In the second experiment, we use `node2vec` to generate

embeddings for the nodes in the network in an unsupervised way. We then create a matrix of these node embeddings, and compute its SVD to generate features for our models. Due to the computational requirements of `node2vec`, we reduced the dataset to only 25 weeks of data.

3 Literature Review

There has been extensive research in predicting the formation of individual edges in a network. For example, [1] investigated if the evolution of a network can be modeled from its intrinsic features only. In particular, the paper explored if they could perform link prediction using only the graph structure and no attributes about the nodes. Their approach is also dynamic; that is, it considers the evolution of the network over time. (A static approach seeks to find missing links in the network at the same period of time.) The authors explored several predictors for links, including graph distance, common neighbours, Jaccard's coefficient, Adamic/Adar, preferential attachment, Katz_β, hitting time, commute time, rooted PageRank_α, and SimRank_γ. In addition, the authors used meta-approaches that leveraged these predictors, including low-rank approximations, unseen bigrams, and clustering.

In addition to exploring the graph structure (topological features), [2] also include non-topological features for their supervised learning algorithm. They attribute a large part of the success of their work to this feature selection. "Proximity Features" measured the similarity of two nodes in the network, by using attributes such as the count of matching keywords. "Aggregated Features" combine attributes of two nodes, such as summing their papers, number of neighbors, keyword count, research areas, and more. These were in addition to using the standard topological features used by previous researchers.

[3] also investigate the problem of link prediction. Rather than using a single snapshot of the network to predict future links, the authors try to leverage the temporal evolution of the network to make such predictions. In particular, to predict the likelihood of two nodes getting a link in the future, the authors define the concept of frames, time slices of snapshots of the graph, using which they compute a score over time.

In our paper, we extend this prior work to the problem of predicting the total number of new links in a growing network where new nodes are constantly being added. We use a lot of the topological and non-topological features that the authors found useful and also adopt a supervised learning approach as they did to train our predictive model.

While most previous works have been successful in the link prediction task with a number of handcrafted features, unsupervised representation learning has also been successful in the space. Unsupervised approaches enable us to learn based on information in the graph that might not be captured in the handcrafted features. In particular, `node2vec` [4] has achieved state-of-the-art results in link prediction for a number of datasets by using solely features learned in an unsupervised way. We also apply this technique to see how it compares with supervised learning.

4 Dataset

Our work focuses on the historical transaction data for the Bitcoin network. Our goal is to use the transaction data in an interval to predict the number of transactions in the subsequent period. We build our dataset using the transaction graph and timestamp data in [5, 6].

The dataset consists of an edge for each input-output address pair in a transaction. For example, a Bitcoin transaction with 3 inputs and 2 outputs becomes 6 edges in the final dataset. We stored the data in a SQLite3 database and filtered to 24 months of data starting in November 2011 and ending in November 2013 since we do not have timestamp data available beyond 2013. We then split the data by month; that is, each raw datapoint corresponded to the edges for that month. Finally, we partitioned our dataset into training and testing splits, with alternating

months for each. In addition, 30% of the training split was reserved for validation. For unsupervised feature learning, we reduced the dataset to its first 25 weeks due to the computational requirements of the methods used.

As can be seen in Figure 1 below, the number of nodes and edges in the network is growing over time which makes the network highly dynamic. Note that the nodes for a time period correspond to the number of wallets that were involved in at least one transaction in that time period. The number of edges for a time period corresponds to the number of input-output pairs in the transactions in that interval.

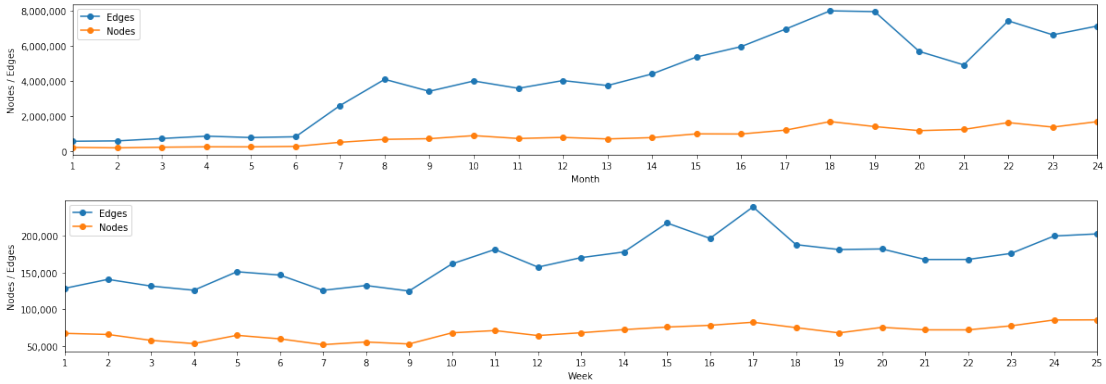

Figure 1: This figure shows the number of nodes and edges per month or week in the constructred dataset.

5 Methodology

For our first experiment, we extract both topological and non-topological features from the network. The features we use, based on [1, 2, 3], are discussed below. There were certain features that were extremely expensive to compute for the entire network, so we instead sampled a subset of edges or nodes to approximate the respective values.

5.1 Topological Features

We describe these as features that capture the overall structure of the network giving insights into network properties such as connectivity, degree distributions, path lengths and other high level network properties. Where a feature computes values for each node, such as with closeness centrality, we use the mean and standard deviation of these values as our topological feature for the entire network. We extract the following topological features from our network:

Average Clustering Coefficient: We calculate the average clustering coefficient $\hat{C}$ as

$$\hat{C} = \frac{1}{n}\sum_{i=1}^{n} C_i,$$

where C_i is the local clustering coefficient for node i.

We expect a higher $\hat{C}$ to predict a greater number of transactions since this indicates the creation of clusters that would lead to increased transaction activity within these clusters.

Network Modularity: The network modularity Q is calculated as

$$Q = \frac{1}{4m}\sum_{i,j}\left(A_{i,j} - \frac{k_i k_j}{2m}\right)$$

where $A_{i,j}$ is the adjacency matrix, m is the number of edges in the graph, and k_i is the degree of node i.

Q measures the strength of division of a network into modules or communities. The stronger this division the higher is the value of Q. This feature should augment the Average Clustering Coefficient in understanding the structure and strength of communities in the Network.

PageRank: We use the mean and standard deviation of the PageRank scores over all the nodes in the network. PageRank measures the relative importance of each node in the network. The standard deviation of the PageRank values should give us insight into how varied the importance of nodes in the Network are. We hypothesize this might have some correlation with the number of transactions in the network.

Node Degree Statistics: We compute the average node degree, along with the standard deviation of the in-degrees and out-degrees. We hope these features will provide insight into the distribution of the network node degrees and that this will have some predictive power over the network transactions.

Path Length Statistics: Path length is the shortest distance between any two given nodes. We compute the mean and standard deviation of the path length of a sample of the nodes in the network. These should indicate how closely connected the network is and one would expect a more connected network structure to lead to a higher number of new transactions in the network.

Closeness Centrality: Closeness centrality $C(x)$ of a node x is defined as the reciprocal of the farness, that is,

$$C(x) = \frac{1}{\sum_y d(y,x)}$$

where $d(y,x)$ is the distance between vertices x and y for all $y \neq x$. We use the mean and standard deviation of $C(x)$ for all nodes x in the network to give us another feature that measures distances between nodes.

WCC and SCC Statistics: We compute the fraction of nodes in the largest Weakly Connected Component and the largest Strongly Connected Component. The size of each of these sub-network generally provides some insight into the network structure and overall network connectivity and thus we expect these to be useful features.

Node Eccentricity: Node eccentricity is the largest shortest-path distance from a given node u to any other node v in the Network for $u \neq v$. We used this metric to give us insight into how far apart the farthest node pairs are from each other. This should indicate whether or not the network exhibits the small world property.

Effective Diameter: The effective diameter is the 90th percentile of the distribution of shortest path lengths. We expect this will give insight into how far apart the farthest nodes are and we expected a smaller value to be positively correlated with increased network transactions. This feature should augment Node Eccentricity above.

Betweenness Centrality: Betweenness centrality $g(v)$ of a node v is given by

$$g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where σ_{st} is the total number of shortest paths from node s to node t and $\sigma_{st}(v)$ is the number of those paths that pass through v. The mean and standard deviation of this metric gives insight into how concentrated the paths through nodes in the network are and we expect this would have some predictive power on the transactions.

Freeman's Network Centrality: Freeman's Network Centrality $FC(G)$ of a graph G is calculated as

$$FC(G) = \frac{\sum_{x \in G} N_{max} - |N(x)|}{(|G|-1) \cdot (|G|-2)}$$

where G is the Network, $|G|$ is number of nodes in G, $N(x)$ is the set of neighbours of node x, and N_{max} is the maximum degree of any node x in G. This measure was proposed in [7] and gives us an indication of how heterogeneous the degree centrality in the network is.

5.2 Non-Topological Features

Non-topological features capture properties about node pairs in a network as opposed to properties about the network as a whole. While traditionally these features have been used to predict individual links between those nodes, we extend their use to also predict the total number of transactions in the network. We do so by computing the values of each feature for different node pairs and then using the mean and standard deviation of all these values as one feature for the entire network.

Pairwise Common Neighbors: We calculate the number of common neighbors over pairs of nodes s and u for $s \neq u$ over the network. We expect this feature would give insight into how many common 'friends' node pairs have with the hope that a greater value would mean a more strongly connected network.

Jaccard Coefficient: This is a commonly used similarity measure in Information Retrieval [8]. It measures the probability that x and y both share a common feature f for a random feature f that either x or y has. In our case, we can treat the neighbors of x and y as features and then the proportion of common neighbors of x and y is a good measure of similarity of x and y. Formally, it is calculated as

$$\frac{|\Gamma(x) \cap \Gamma(y)|}{|\Gamma(x) \cup \Gamma(y)|}$$

where $\Gamma(a)$ is the set of neighbors of node a.

Preferential Attachment Score: Preferential attachment has gained increasing popularity as a model for the growth of networks [9]. The model suggests that the probability that a new edge is formed with node x is proportional to $|\Gamma(x)|$, the number of neighbors of x. Newman [10] and Barabasi et. al [11] have also found, based on empirical evidence, that in the co-authorship domain, the probability of x and y coauthoring together is proportional to $|\Gamma(x)| \cdot |\Gamma(y)|$. We use this measure for our network expecting that the larger the product of the size of neighbors of two nodes, the more likely that they will transact with each other.

Adamic Adar: This measure, also known as Frequency-Weighted Common Neighbors was proposed in [12]. It is calculated as

$$\sum_{z \in \Gamma(x) \cap \Gamma(y)} \frac{1}{log(|\Gamma(z)|)}$$

This measure gives more weight to rarer features. In the context of networks, this means that two nodes x and y are more likely to be connected to each other if their common neighbor z has few neighbors since this increases the likelihood of z connecting x with y.

Neighborhood Connectedness: Another measure we can look at for evaluating whether node x and y will transact with each other is to look at the connectedness of their neighborhoods. The more connections that the neighborhoods of x and y have with each other, the higher the chance of x and y transacting. Formally, we calculate neighborhood connectedness as

$$\sum_{a \in \Gamma(x)} \sum_{b \in \Gamma(y)} \delta(a, b)$$

where $\delta(a, b)$ is defined as

$$\delta(a,b) = \begin{cases} 1 & \text{if } a = b \text{ or } (a,b) \in E \text{ or } (b,a) \in E \\ 0 & \text{otherwise.} \end{cases}$$

5.3 Unsupervised Representation Learning

In the previous section, we outlined a number of handcrafted features we used for prediction. While we believe these features could be helpful for our task (and have had success in past papers), we realize they might not capture all the relevant information present in the graph. Additionally, the need for sampling in computing several of these features increases the potential problem in extracting meaningful features. As an alternative, we propose to learn features by learning a representation of the graph in an unsupervised way.

In particular, we use `node2vec` [4] to create an embedding of the nodes. Using the embedding, we then create a set of features for each interval. For our experiments, we used 128 dimensions, walk length of 80, 10 walks per source, optimization context size 10, and one SGD epoch. We were unable to run `node2vec` in the whole dataset (which contains over 100M edges) because of limited computational power. Thus, we used a dataset composed of the first 25 weeks of the previous dataset, and used each week as the interval for prediction.

For each of those intervals, we build a set containing the embeddings for all the nodes that transacted in that period in a matrix X. Without further preprocessing, these features are not useful since the number of nodes (and thus the matrix dimension) varies per interval and the representation is not permutation invariant with regard to the row ordering. Our solution was to compute the singular value decomposition (SVD) of the matrix, and extract k singular values and their corresponding right singular vectors as features. Note that the singular values and right singular vectors remain unchanged if you permutate the rows of a matrix. To see this, let $X' = PX$ be a permutation of the rows of our original matrix X with SVD $X = U\Sigma V^*$. Then, we easily see that X' has the SVD

$$X' = PX = PU\Sigma V^*$$

since PU is unitary. Thus, X' and X share singular values and right singular vectors.

Computing the SVD of the matrices became unfeasible due to their size. Instead, we used FBPCA [13] to compute an approximate decomposition. With this approach, we were able to compute the features in reasonable time.

We then chose $k = 16$, thus creating features based on the 16 largest singular values and their right singular vectors. Each interval has a total of $128 \times 16 + 16 = 2064$ features.

6 Regression Models

Using the extracted features for each month's edges, we trained a number of regression models to predict the following month's number of transactions. We discuss these models below.

6.1 Decision Trees

Decision Trees have become increasingly popular in recent years [14, 15]. Their success stems from the fact that they are able to self-select features that are important for prediction along with the conditions to split the features on. In addition, unlike most other machine learning models, the results of decisions trees are easily interpretable and explainable. We use a MSE criterion for our decision tree.

6.2 Support Vector Machines (SVMs)

SVMs [16] are another class of supervised machine learning models that work well for regression tasks. In particular, SVMs have been found to be effective in high dimensional spaces as is the case with our feature space. Our SVM was a support vector regressor (SVR) with $C = 1$ and $\epsilon = 0.1$. We experiment with both linear and RBF kernels. Note that SVMs required us to appropriately scale the data.

6.3 Feed Forward Neural Networks

We also tried feed-forward neural networks (multi-layer perceptrons) [17]. These have recently been found to be extremely powerful at tackling complex machine learning tasks. In particular, their non-linear activation functions and model flexibility allows them to learn all sorts of mappings and interactions between features to effectively predict the target value. Our feed-forward neural network had one hidden layer with 24 hidden units and ReLU activation units. We used an Adam optimizer [18] for 10,000 iterations with $\beta_1 = 0.9$ and $\beta_2 = 0.999$ parameters, learning rate of 0.001, early stopping, and $L2$-regularization with weight 0.0001.

7 Experimental Results

We evaluated our models on the testing splits described before. As a baseline, we also evaluated the models on predicting transaction numbers based simply on the node and edge counts in the previous month. In doing so, we're able to see the effects of our other features.

7.1 Topological and Non-Topological Features

In the table below, we report the average relative error and associated standard deviation in predicting the number of transactions over 100 runs per model.

	SVM (Linear)	**SVM (RBF)**	**Decision Tree**	**Neural Network**
Node and Edge Counts	$27.0\% \pm 0.0\%$	$27.8\% \pm 0.0\%$	$29.7\% \pm 2.3\%$	$20.5\% \pm 1.4\%$
All Features	$39.5\% \pm 0.0\%$	$41.0\% \pm 3.7\%$	$29.8\% \pm 3.7\%$	$20.4\% \pm 1.4\%$

When looking at all our features (node and edge counts plus all topological and non-topological features), we find that SVMs performed the worst of all the models with relative error close to 40% for both the linear and RBF kernels. Neural Networks gave the best performance for all features with relative error around 20% while Decision Trees were in the middle with relative error around 30%

Overall, the models augmented by our topological and non-topological features did not improve on the accuracy from the baseline model (using only node and edge counts as features). In fact, for SVMs they performed even worse. These results suggest that the features we are extracting are not actually providing any additional predictive power over the baseline features of node and edge counts. This result seems surprising at first because we use state of the art topological and non-topological features that have been found to be be highly effective at solving the individual link prediction task. However, after some thought it becomes clear why this is so.

First, the literature looks at using these features to predict individuals links and not the total number of new links created. We used aggregate quantities of these features such as the mean and standard deviation to predict the total number of links created in this network. It is very likely that aggregating these feature values in this manner over the entire network reduces their predictive power by making them more noisy. An extension of our work could involve using the individual features between node pairs to predict links between all the possible node pairs in the network and then summing this to get the total number of new links in the network. However, for a network the size of Bitcoin, this is computationally intractable given the compute power available to us. In addition, there would be no way to predict edges between nodes that will only be created in the next time period.

Second, and perhaps more importantly, it seems that the features do not actually add any real predictive power in the Bitcoin network over the baseline nodes and edge counts. Bitcoin transaction numbers are noisy in nature given the relative young age of the network and they are also influenced by a host of different factors. In addition, most of the transaction volume in the Bitcoin network today comprises of trade transactions - bitcoins moving form one wallet to another due to trade on exchanges. These transactions have little correlation with the topological and non-topological features of the network and are more correlated with external factors

such as financial market adoption, regulations and price movement. Since most transactions in the network are of this nature, this explains why the features do not provide much additional predictive power on the number of transactions than simply using the number of nodes and edges. Consider for instance, the case of triadic closure. In regular networks, if node A and node B were friends with node C but not each other, we would expect there to be an edge in the future between node A and node B. However, for the Bitcoin network, if node A and node B had a transaction with node C (which represents an exchange) then A and B are not actually 'friends' with node C and thus there is very little chance of there being a triadic closure. One way to work around this issue would be to strip out all the exchange-related transactions from the network and only run our experiments on 'real' transactions in the network where our features actually provide predictive power. However, given the anonymous nature of the Bitcoin network, this is not a feasible task.

Figure 2 below shows the ground truth number of transactions and the predicted number of transactions for the baseline and our full feature set (topological and non-topological) for the neural network model on the test set. This figure confirms that the features do not add any additional predictive power to the baseline model. Furthermore, we also find that our predictive model, both for the baseline and full feature set, sometimes undershoots the baseline and sometimes overshoots it. This indicates that the results of the model are noisy as it is not really learning anything systematic about the number of transactions based on the network.

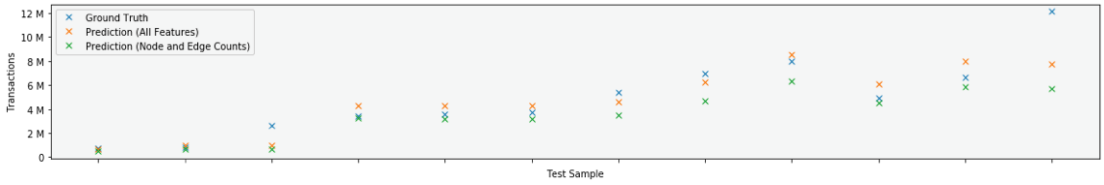

Figure 2: This figure shows the predicted values in comparison to the ground truth values over the test set

7.2 Unsupervised Features

In a similar fashion, we compute the unsupervised features using the methodology based on `node2vec` described in the previous section. We find that, again, the learned features did not outperform a regression on node and edge counts. In particular, a combination of these two (listed as Unsupervised Features + Counts in table) under performed the version based on the node and edge count features for every model except for the neural network. The table and figure below present our results for different variations of the features and the models.

	SVM (Linear)	SVM (RBF)	Decision Tree	Neural Network
Node and Edge Counts	$10.8\% \pm 0.0\%$	$9.4\% \pm 0.0\%$	$9.7\% \pm 0.2\%$	$10.5\% \pm 0.7\%$
Unsupervised Features	$16.4\% \pm 0.0\%$	$16.4\% \pm 0.0\%$	$16.3\% \pm 3.4\%$	$10.9\% \pm 0.2\%$
Unsupervised Features + Counts	$16.4\% \pm 0.0\%$	$16.4\% \pm 3.7\%$	$16.7\% \pm 2.7\%$	$10.5\% \pm 0.8\%$

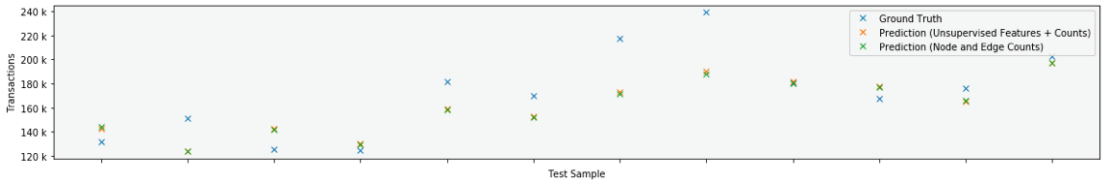

Figure 3: This figure shows the predicted values in comparison to the ground truth values over the test set

Since `node2vec` creates embeddings for nodes and not edges in the graph, we lose valuable information when using only the node embeddings; we effectively predict using the wallets that

transacted rather than the input and output edges of these transactions. The issues mentioned in the previous subsection could have also prevented this experiment from achieving positive results. In addition, the need to approximate computations and the choice of $k = 16$ for PCA could have lead to features that do not accurately represent the graph.

8 Conclusion

A reasonable baseline for our task is to predict the number of transactions in the next month using the number of nodes and edges in the network from the previous month. We then expected that augmenting these two powerful features with more advanced topological, non-topological, and unsupervised features would give us better predictive power. However, as we see from the previous section, this was not the case. For all of our models, we find that that there is no significant difference in predictive power between the baseline model and the baseline model augmented with additional features.

We conclude by mentioning a few additional reasons for why this might be the case. It could be that the number of nodes and edges captures enough information about the network that adding more features does not provide any additional predictive power. This is even more plausible given the Bitcoin network, where most of the transactions are trade transactions and these transactions are directly correlated with how many people traded Bitcoin in the previous month. Finally, it could be that there is actually very little correlation between features of the Bitcoin network and the number of transactions. Several external features such as the price of Bitcoin, news announcements, activity in other cryptocurrencies, and several other factors also have a large role to play in the number of transactions and these could prove to be bigger influences than the network properties itself.

9 Challenges and Future Work

The number of nodes and edges in the network limited the amount of computation that could be performed in reasonable time. For example, there are up 12 million edges for some of the months we analyzed. As a result, certain features with high computational complexity cannot be calculated for the complete network. We had to sample small subsets of the network in order to approximate some of these features and this could have potentially skewed the features due to sampling error.

Our current dataset also only spans until end of 2013. The Bitcoin network was still in early stages back then and thus the network transactions grew fairly stably over time without much variability. Since then, however, the number of transactions per month has seen higher variation and this is probably a more interesting period to predict transactions in that would benefit from the additional features we extract. Currently, there are datasets available beyond 2013 but without time stamps for the transactions which makes it unusable for our purposes. We would be interested in revisiting our approach in the future when this extended dataset is available.

In addition, we could look into exploring varying the time period used for both training and testing. Currently, one time period is represented as a month, but we could in the future explore more granular time periods at the expense of higher computational power. We could also look into using recurrent neural networks to condition the outcome on a series of previous time periods as opposed to just the immediately preceding time period.

Finally, we had to limit our experiment involving `node2vec` due to its computational requirements. It would be interesting to explore other unsupervised learning algorithms that are designed to scale, such as DeepGL [19].

References

[1] David Liben-Nowell and Jon Kleinberg. "The link-prediction problem for social networks". In: *journal of the Association for Information Science and Technology* 58.7 (2007), pp. 1019–1031.

[2] Mohammad Al Hasan et al. "Link prediction using supervised learning". In:

[3] Darcy Davis, Ryan Lichtenwalter, and Nitesh V Chawla. "Multi-relational link prediction in heterogeneous information networks". In: *Advances in Social Networks Analysis and Mining (ASONAM), 2011 International Conference on*. IEEE. 2011, pp. 281–288.

[4] Aditya Grover and Jure Leskovec. "node2vec: Scalable feature learning for networks". In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2016, pp. 855–864.

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

[6] 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.

[7] Linton C Freeman. "Centrality in social networks conceptual clarification". In: *Social networks* 1.3 (1978), pp. 215–239.

[8] Salton Gerard and J McGILL Michael. *Introduction to modern information retrieval*. 1983.

[9] Michael Mitzenmacher. "A brief history of generative models for power law and lognormal distributions". In: *Internet mathematics* 1.2 (2004), pp. 226–251.

[10] Mark EJ Newman. "Clustering and preferential attachment in growing networks". In: *Physical review E* 64.2 (2001), p. 025102.

[11] Albert-Laszlo Barabâsi et al. "Evolution of the social network of scientific collaborations". In: *Physica A: Statistical mechanics and its applications* 311.3 (2002), pp. 590–614.

[12] Lada A Adamic and E Adar. "Predicting missing links via local information". In: *Social Networks* 25.3 (2003), pp. 211–230.

[13] Arthur Szlam, Yuval Kluger, and Mark Tygert. "An implementation of a randomized algorithm for principal component analysis". In: *arXiv preprint arXiv:1412.3510* (2014).

[14] S Rasoul Safavian and David Landgrebe. "A survey of decision tree classifier methodology". In: *IEEE transactions on systems, man, and cybernetics* 21.3 (1991), pp. 660–674.

[15] Leo Breiman. *Classification and regression trees*. New York: Chapman & Hall, 1993. ISBN: 0412048418.

[16] Ingo Steinwart and Andreas Christmann. *Support Vector Machines*. 1st. Springer Publishing Company, Incorporated, 2008. ISBN: 0387772413.

[17] Geoffrey E Hinton. "Connectionist learning procedures". In: *Artificial intelligence* 40.1-3 (1989), pp. 185–234.

[18] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[19] Ryan A Rossi, Rong Zhou, and Nesreen K Ahmed. "Deep Feature Learning for Graphs". In: *arXiv preprint arXiv:1704.08829* (2017).