COS 424 Homework 3

Vlad Feinberg
Undergraduate
Princeton University
vyf@princeton.edu

Bill Van Cleve
Undergraduate
Princeton University
wmcleve@princeton.edu

Abstract

Link prediction in social networks is a commonly-studied challenge. The Bitcoin network presents several additional complications: there is a sender-receiver direction to each relationship, and multiple links can exist between users (representing multiple payments). We use about a year's worth of Bitcoin data in order to predict interactions between users in the next month who never interacted with each other during that year. We primarily use matrix decomposition methods. The best model, which uses non-negative matrix factorization, has AUC 0.783 and best F_1 0.491 for a positive-emphasized test sample.

1 Introduction and Related Work

Bitcoin is an online currency that has exploded in popularity in recent years. One of the reasons for its growth has been the anonymity that it affords users. While every bitcoin transaction is publicly available in what is known as the blockchain, the parties are identified only through a public key [7]. The availability of this blockchain means that anyone can access all previous Bitcoin transactions, though they cannot link transactions to specific parties. The purpose of this paper is to use this transaction history to predict future transactions.

Existing literature on the subject focuses heavily on graph-based methods. The transaction-prediction problem is essentially a link-prediction problem, so approaches developed for social networks are a logical starting point. Methods frequently rely on similarity measures between nodes [5]. Mutual neighbor count is one example. Adamic/Adar refines that approach, taking into account neighbors shared by each node and weighting rarer neighbors (isolated nodes that are mutually interacted with) more heavily [1]. Their similarity function for nodes a and b is $\sum_{z \in \Gamma(a) \cap \Gamma(b)} \frac{1}{log|\Gamma(z)|}$ for the neighbor operator Γ [1]. Other methods, known as global methods, examine all paths between two nodes. SimRank is one commonly used global method, which purports that similar nodes will be connected to similar neighbors. SimRank defines the similarity between two nodes a and b recursively as: $S(a,b) = \frac{C}{|I(a)||I(b)|} \sum_{i=1}^{|I(a)|} \sum_{j=1}^{|I(b)|} s(I_i(a), I_j(b))$ where C is a constant between 0 and 1, $I(v)$ is the set of in-neighbors of node v, and correspondingly O is the out-neighbor operator [3] [10]. Finally, there exist several supplemental methods used in conjunction with the approaches mentioned above. One of the more common is pruning, where the weakest links by similarity measure are deleted from the graph before the primary method is trained. By doing this, the predictions are formed based only on the edges for which the predictor itself is most confident [5].

2 Dataset

Our training dataset consists of a list of sender-receiver pairs of Bitcoin public keys that interacted between March 2012 and March 2013, with a count of the number of transactions for each pair. The dataset is quite sparse: out of 443,652 observed senders and 439,602 observed receivers, only

3,348,026 pairs transact with each other. This means that the sparsity rate is more than 99.998%. The test dataset consists of a list of 10,000 sender receiver pairs with a binary variable indicating whether or not the pair transacted. 1,000 of the 10,000 transacted, while 9,000 did not. Note that none of the data is dated, but the test data comes from a different time period than the training data.

In order to get a sense of what patterns might exist in the data, we examine the frequency of how often pairs interact with each other. Figure 1 presents this distribution.

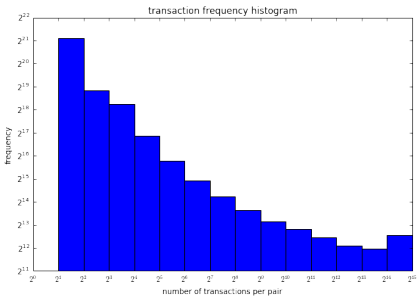

Figure 1: Histogram of number of transactions per pair. Presented on a log-log scale, the trend appears to be roughly linear, meaning the actual relationship is roughly exponential. This will have important implications when computing various similarity metrics - outlier interaction counts may artificially inflate similarity, implying certain numerical transforms may be necessary.

We also examine the distribution of the incoming and outgoing transactions for every address, presented in Figure 2. Here, the distribution appears to be roughly Poisson with a heavy tail on the right.

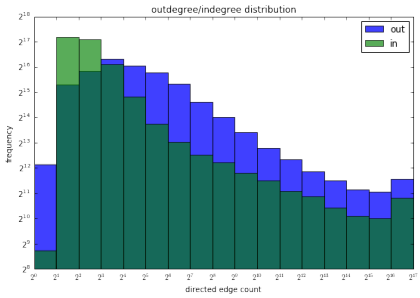

Figure 2: Histogram of in-degree and out-degree count distribution. The median address is a sender in five transactions and a receiver in four. The most prolific sender engaged in 742,100 transactions while the most active receiver participated in 1,696,391. Now that's an address likely to transact again! However, our prediction focuses on only new interactions (the test set focuses on these).

Much of our analysis will work with the the data in the form of a directed graph; as Figure 2 demonstrates, the distributions of the senders and receivers are distinct, so symmetrized approaches would relinquish information [5]. The digraph we operate on has addresses as nodes and weighted edges with direction of payment flow. Weight is equal to the number of transactions observed in the training data.

Figure 3: At least 90% of the SCCs in the data are of size one, meaning that they consist of nodes that are not strong connected to any other nodes. In addition to these isolated points, there are also a few extremely large SCCs: the biggest consists of 433,676 nodes. So rather than there being many small clusters of users interacting with each other, it appears that most users are either relatively isolated or part of a massive SCC - this renders analyses with a "small worlds" assumption useless.

SCC Size Distribution

Quantile	Size of SCC
0.9	1
0.99	2
0.999	9.747
0.9999	32866.5749
0.99999	393595.0575
0.999999	429667.9057
1	433676

Simrank based approaches were disqualified with reasoning based on Figure 3: while transaction data may be sparse, second-order interactions between adjacencies in the graph are not - indeed, the matrix product of the digraph's adjacency matrix - an essential operation in the Simrank computation. A 1000 row sample of the dot product had only 80% sparsity, which would require several hundred GB RAM to represent. Further Simrank update iterations would reduce sparsity further.

3 Methods

All techniques considered were forms of matrix factorization. For some measure of distance d on matrices and a given matrix X, each method minimizes $d(X, AB)$ for A, B constrained in some manner, where the rank of A, B is much less than X. This low rank truncation brings out what we assume is a signal to the data. We take X to be our directed adjacency matrix.

Truncated Singular Value Decomposition (tSVD). The $U\Sigma V^\top$ decomposition for truncated U, V identifies "sending" and "receiving" singular vectors, where the rows of both matrices are the linear combinations of the singular vectors. A Σ-weighted dot product is then the assigned score. We tuned the number of components, trying up to 20. We also performed normalization, wherein we ran the SVD value through a logistic function [8].

Non-negative Matrix Factorization (NMF). NMF decomposes a non-negative matrix X into a features matrix A and coefficient matrix B, with $d = \ell_2$. We tuned the following list of hyperparameters: number of components, ℓ_1, ℓ_2 regularization for A, B, the ratio of regularization between the two norms (as in elastic net), and the initialization (random, zero, or nonzero constant) - both norm operators are the Frobenius p-norms [8].

Latent Dirichlet Allocation (LDA). LDA decomposes a count matrix X into a document-topic matrix A and topic-word matrix B, with d as the negative log likelihood bound from Bayesian variational inference. Here documents are senders and words are receivers. Only up to 4 topics were analyzed because of tractability constraints.

For a baseline, we used **Jaccard similarity**, scores a new transaction $i \to j$ by matching i's sending profile to j's receiving one - the model assumption here is that the adjacency graph tends to its transitive closure over time. The similarity can be expressed as $\frac{|O(i) \cap I(j)|}{|O(i) \cup I(j)|}$.

Count scaling was introduced because of the issue mentioned in Figure 1: appropriate scaling may put the importance of repeat transactions in proportion - scaling by $x \mapsto \log(x + 1)$ (log1p) or converting to binary, throwing repeat transaction data away, were considered.

Edge pruning was attempted to potentially improve generalization. Upon recommendation from Liben-Nowell and Kleinberg, using a scoring metric for edges (in particular, Adamic-Adar) could help remove transactions which are not indicative of latent structure [5].

3.1 Implementation

The adjacency matrix for the digraph had a sparse CSR representation [4]. Similarity metrics were expensive to compute and assumed undirected representation. For undirected algorithms, we symmetrized the sparse adjacency matrix (then removed both directions in the original digraph if the symmetric graph edge was pruned). Computation issues were addressed because the processing was feasible from a memory perspective: using Unix `fork()` semantics, only one copy of the graph needed to be held in memory at a time. 4 machines with 38 parallel processes (each machine had 48 CPUs) were able to compute the Adamic-Adar similarity index for each edge. Cutting out the bottom 10% and 50% of edges according to their score was considered - see Figure 4 for details.

Each method's hyperparameter grid was explored in parallel as well with the same setup as above. Grid search was applied several times, starting from logarithmic scales.

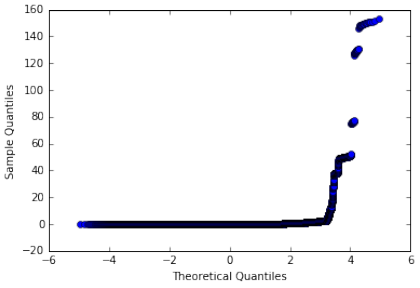

Figure 4: QQ-plot of Adamic-Adar scores (compared to normal). Mean is 3.152, standard deviation is 41.390. Note extreme right skew of this distribution implies low similarity for a majority of the nodes connected by edges, so in theory lots of edges are "tenuous" under this measure.

Method	Validation AUC	Count Scaling	Rank	Other Hyperparameters
NMF	0.8724	log1p	35	$\alpha = 50, \beta = 0$
Normalized SVD	0.8543	binary	4	N/A
SVD	0.8541	binary	4	N/A
$\frac{1}{2}$-Pruned SVD	0.6630	binary	5	N/A
$\frac{1}{10}$-Pruned SVD	0.5482	binary	30	N/A

Table 1: A few notes: for SVD, we set the tolerance for singular values to be 10^{-10}, and we looked for the largest singular values. For NMF, α refers to the sum of the regularization coefficients for both norms, and β refers to the ratio of the regularization on the ℓ_1 norm. Optimal initialization for NMF uses 0-initialization. Note that the $\frac{1}{10}$-pruned SVD used all 20 components in the first grid search, so search was extended to all multiples of 10 up to 100. The lower performance of the pruned algorithms indicates that symmetrization may lose too much information. LDA had an average CV of 0.954 with rank 4 on the log-counts, but this was prone to overfitting on other folds.

3.2 Evaluation

To select the best method, we performed 10-fold cross-validation on the training set: random 1000-sample transactions were removed in full. Then, the method in question was fitted to the data without those 1000 transaction. Validation score was computed by finding the AUC on whether the tested method predicted if the 1000 witheld transactions would appear and if 9000 randomly-sampled transactions from the sparse elements of the full training matrix would not appear. This matches the test set's positive and negative ratios.

Table 1 presents the parameters that performed the best for each method.

4 Results

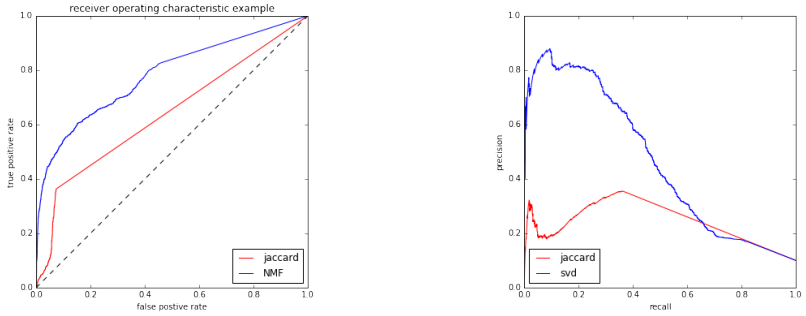

Figure 5: The Jaccard AUC is 0.640; its average F_1 score along the curve is 0.172; and the maximum F_1 is 0.359. Our best validation model (NMF) has an AUC of 0.783, average F_1 of 0.376, and max F_1 of 0.491.

5 Conclusion

By using a variety of dimensionality reduction methods and feature engineering in the form of count scaling, we outperformed a baseline measure similar to the "common neighbors" baseline introduced by [5], with the top model being NMF.

In the future, attempts at different measures of score for the edges may improve pruning quality, which, as can be seen from the heavily skewed Figure 4, was inappropriate for the problem.

6 Acknowledgments

The authors would like to thank Prof. Engelhardt and Mehmet Basbug for their advice on this project and provision of data.

In addition, the authors leveraged the Princeton CS Department's `cycles` systems for many of the parallelizable grid-search and preprocessing tasks.

Finally, the authors heavily relied on `iPython` [9] notebooks, `sklearn` [8] for algorithm implementations, `scipy` [4] and `pandas` [6] for scientific computing functions, and `matplotlib` [2] for graphing.

References

[1] ADAMIC, L. A., AND ADAR, E. Friends and neighbors on the web. *Social networks 25*, 3 (2003), 211–230.

[2] HUNTER, J. D. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering 9*, 3 (2007), 90–95.

[3] JEH, G., AND WIDOM, J. Simrank: a measure of structural-context similarity. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining* (2002), ACM, pp. 538–543.

[4] JONES, E., OLIPHANT, T., PETERSON, P., ET AL. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 2016-04-12].

[5] LIBEN-NOWELL, D., AND KLEINBERG, J. The link-prediction problem for social networks. *Journal of the American society for information science and technology 58*, 7 (2007), 1019–1031.

[6] MCKINNEY, W. pandas: a foundational python library for data analysis and statistics.

[7] MEIKLEJOHN, S., POMAROLE, M., JORDAN, G., LEVCHENKO, K., MCCOY, D., VOELKER, G. M., AND SAVAGE, S. A fistful of bitcoins: Characterizing payments among men with no names. In *Proceedings of the 2013 Conference on Internet Measurement Conference* (New York, NY, USA, 2013), IMC '13, ACM, pp. 127–140.

[8] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., BLONDEL, M., PRETTENHOFER, P., WEISS, R., DUBOURG, V., VANDERPLAS, J., PASSOS, A., COURNAPEAU, D., BRUCHER, M., PERROT, M., AND DUCHESNAY, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[9] PÉREZ, F., AND GRANGER, B. E. IPython: a system for interactive scientific computing. *Computing in Science and Engineering 9*, 3 (May 2007), 21–29.

[10] WIKIPEDIA. Simrank — wikipedia, the free encyclopedia, 2016. [Online; accessed 12-April-2016].