CS224W Project Final Report
Supervised Link Prediction in Bipartite Networks

Kameshwar (Kam) Chinta
kchinta@stanford.com

Kevin Clark
kevclark@stanford.edu

Arathi Mani
arathim@stanford.edu

November 13, 2014

1 Introduction

A ubiquitous challenge in the analysis of networks is predicting what new connections will be created in a particular graph at some point in the future. Being able to predict such connections with accuracy has many important practical uses including national defense, where links between terrorists can be deduced from an existing network [5], security, where links might be able to predict future credit card frauds, and social networks, where potential friend relationships could be predicted [1]. Link prediction also can be mutated into variants such as link recommendation (e.g. the "Find Friends" feature in Facebook listing potential acquaintances), link anomaly detection, and other applications.

Current research focuses on link prediction on graphs with one type of node, such as predicting links in a social network made up of people only. We looked at modifying algorithms that are used for these types of graphs so they can be applied to bipartite graphs. In particular, we created a bipartite graph from the Yelp Challenge dataset by connecting users to businesses by the reviews they wrote and then attempted to predict new edges (i.e. reviews) at a later time. We applied several classes of algorithms for link prediction on this bipartite graph: scores based on similarity metrics, scores based on matrix completion, supervised and unsupervised random walks, and a binary supervised classifier.

2 Prior Work

There has been extensive prior work on link prediction in general. Liben-Nowell and Kleinberg [8] experiment with several node similarity metrics including Graph Distance, Common Neighbors, Jaccard's Coefficient, Adamic/Adar Score, Preferential Attachment, Katz, Hitting Time, Page Rank and Sim Rank. The results suggest that Adamic Score and Katz method (or its variants) are relatively close in their performance and provide good results. Many of these measures could be modified so they can be applied to bipartite graphs. We do this with three of their proposed metrics. We also implement a low-rank matrix approximation method described in this paper and discuss how it can be modified to run more effectively on bipartite graphs.

One challenge of link prediction is building a model that combines information about node and edge attributes with network structure in a principled way. A common approach is to extract network features from two nodes, combine them with node attributes, and use them in a supervised classifier. Another common approach to link prediction is using a PageRank style algorithm that assigns scores to nodes based on the stationary distribution of a random walk. Unfortunately, the first approach captures network structure in a quite limited way, while the second approach ignores node attributes. In this work, we apply both of these approaches to our dataset.

However, Backstrom and Leskovec [1] propose an algorithm based on supervised random walks that combines ideas from both of these approaches. Predictions are made based off of the scores of a random walk, but the edge weights for this walk are learned from node and edge attributes in a supervised way. Edge weights are assigned as a function f of the attributes of the edge and the involved nodes. This function is learned with a gradient-based optimization technique that minimizes a loss function over the graph. The supervised random walk algorithm outperformed many other approaches, both supervised and unsupervised, on all datasets. We implement the supervised random walks algorithm and also try random walks with heuristically weighted edges in this work.

Link prediction on bipartite networks specifically is a much less explored topic than general link prediction. Benchettara et al. [2] describe how common link prediction similarity measures can be used as features in the bipartite setting. Instead of applying those metrics to the network as-is, they apply them to projections that convert one side of the bipartite network into a single network with homogeneous nodes. This provides a similarity score between a given node and every other node on the same side of the network. To use these to make a feature for that node, they simply take the maximum similarity over all other nodes on the same side. These features were used in a supervised link prediction classifier and improved accuracy for both the DBLP bibliographic database and a history of transactions on a movie e-commerce site. However, this feature scheme seems quite limited because the features come from one node at a time instead of the pair. We instead apply similarity metrics that can be applied directly to nodes on different sides of the network.

Two other works on bipartite link prediction are those of Yamanishi [10], who uses distance metric learning which is modified to accommodate bipartite graphs and Kunegis et al. [4] who propose a specific class of graph kernels, spectral transformation kernels, generalized for link prediction in bipartite networks. Although we do not implement these techniques, they further illustrate how link prediction techniques can be adapted to be applied on bipartite graphs.

3 Algorithms

We implemented the following algorithms for link prediction in bipartite graphs. Each algorithm takes a candidate (currently non-existing) edge between two nodes on opposite sides of the bipartite graph and produces a score predicting whether that edge will exists in the future.

3.1 Random Baseline

This simply assigns each candidate edge a random score. This is meant to be a benchmark for comparing our other algorithms to, not an effective link prediction algorithm.

3.2 Similarity Metrics

3 similarity measures were used for computing the score of each candidate edge. Most similarity metrics commonly used for link prediction, such as those presented by Liben-Nowell and Kleinberg [8] compare the neighbors of the two nodes in the edge. However, using these sets of nodes does not work in the bipartite setting because the neighbors of nodes on opposite sides of the network do not intersect. Instead, we want both sets of nodes to contain the same type of nodes (that is nodes on the same side of the network). To do this, we chose the the following two sets $S(x), S(y)$ for our similarity metrics given nodes (x, y) on opposite sides of the graph.

- $S(x)$: nodes 2 hops away from x: This produces a set of nodes on the same side of the network as x because traveling distance 2 goes to the other side of the network and back again.
- $S(y)$: y's neighbors: These nodes are on the opposite of the graph of y, so they are on the same side as x.

Thus both sets only contain nodes on x's side, so we can easily use them to compute distance metrics. Intuitively, computing similarity metrics over these sets of nodes is effective for link prediction because $S(x)$ returns the nodes that are similar to x since they are close (2 hops away) and on the same side of the network. Put more concretely, we assume users that review the same businesses you do ($S(x)$

if you are x) are similar to you. Thus it is reasonable to further assume that if a lot of them review the same business y (i.e. $S(y)$, the users that review y has a large overlap with $S(x)$), you are likely to review that business in the future too.

Note that this way of choosing sets can be applied in two ways: taking 2 hops from nodes on the left side of the graph, or taking 2 hops from nodes on the right side of the graph. We tried both of these strategies and report results for each. We used the following distance metrics between the two sets of nodes:

1. Common Neighbors: $|S(x) \cap S(y)|$
2. Jaccard's Coefficient: $\frac{|S(x) \cap S(y)|}{|S(x) \cup S(y)|}$
3. Adamic/Adar: $\sum_{z \in S(x) \cap S(y)} \frac{1}{\log \Gamma(z)}$

The set $\Gamma(z)$ is the set of the neighbors of z in the entire bipartite graph G. In more detail, the common neighbors measure is the most straightforward measure of similarity between the two sets of nodes: how many are in common. Jaccard's coefficient in constrasts, measures the *probability* of a common neighbor by dividing the intersection of the sets by the union. Finally, Adamic/Adar looks at the common neighbors' features (common neighbors' neighbors) and gives weight to the feature by means of the reciprocal of the log of the feature.

3.3 Low Rank Approximation

Link prediction can be viewed as predicting new entries in A, the adjacency matrix of the graph. A commonly used technique for analyzing large matrices like A is computing a low rank matrix A_k of rank k that approximates A. One popular low-rank matrix approximation technique uses the *singular value decomposition* (SVD) of A. A SVD factorization is of the form $A = U\Sigma V^T$ where Σ is a diagonal matrix containing the singular values of A. If Σ is replaced with Σ_k, which only contains the k largest singular values of M, the resulting matrix $A_k = U\Sigma_k V^T$ will have rank k and be a good approximation of A. In fact, A_k will be the matrix that minimizes the squared differences between entries in A and A_k (i.e. the Frobenius norm of $A - A_k$). See [3] for some mathematical details. This approximation can be computed efficiently for small k, even when A is a large sparse matrix such as an adjacency matrix (we use scipy's sparse package to do this). We use the entries of the low rank approximation A_k as scores for candidate edges.

There are two properties of the adjacency matrix for bipartite graphs that we can take advantage of when computing the SVD:

1. The matrix is symmetric because the graph is undirected.
2. A large number of entries (at least half) of the matrix are guaranteed to be 0 because edges can't go between nodes on the same side of the graph.

To take advantage of these properties, we compute the SVD of the user-business matrix M instead of over the adjacency matrix. M is the matrix where the rows represent users and the columns represent businesses and $M_{i,j} = 1$ if there is an edge between user i and business j. This matrix is guaranteed to have half the nonzero entries of A, be at most a quarter of the size, and automatically enforces the constraint that there are no edges between nodes on the same side of the bipartite network. In this decomposition, we are effectively learning k-dimensional feature vectors for each user and each business such that the dot product between them is high if an edge is present. We tried several values of k on our data and found $k = 50$ to produce the best results.

3.4 Unsupervised Random Walks

3.4.1 Mathematical Background

A random walk is a finite Markov chain that moves through a graph $G = (V, E)$. Suppose each edge $(u, v) \in E$ is assigned a weight $w(u, v)$. Then each edge $(u, v) \in E$ can be assigned a transition probability $M_{u,v} = w(u, v)/d(u)$ where $d(u) = \sum_{(u,v)\in E} w(u, v)$ is the weighted degree of u. These transition probabilities can be used in a random traversal of the graph. Each step, if we are currently at node u, we move to node v with probability $M_{u,v}$. A natural question arising from this is which nodes would we tend to visit if we ran this random process for a long time. To find this, we first note that the probabilities $p_t = (p_t(0), p_t(1), ..., p_t(n))$ of ending a random walk at particular node after t steps is given by $p_t = p_{t-1}M$. It turns out that p_t provably converges to a *stationary distribution* p_s as $t \to \infty$, where $p_s = p_s M$ [6].

Another important concept is the idea of running a *personalized* random walk from a particular node u. This is the same as a random walk described above except there is a probability α of jumping back to u during a walk. In this case the probability distribution is given by $p_t = (1-\alpha)p_{t-1}M + \alpha x_u$ where x_u is the vector with 1 at u and zeros for all other entries.

3.4.2 Algorithm Description

We used random walks to produce a score for each candidate edge (u, v). Specifically, we score the edge with the stationary distribution probability of reaching v when running a random walk from u. We also tried weighting the edges before running the random walk. For this approach, we use simple heuristically chosen edge weights instead of learning the weights in a supervised way. Intuitively, we want links that are more "relevant" to have higher weight for the random walk. To capture this, we weighted edges that were created more recently higher. This increases the power of the model by including information about how the network evolved over time as well as information about the network structure. Specifically, we gave each edge a weight of $w_{uv} = 1/(c + a_{uv})$ where a_{uv} is the age of the edge (u, v) and c is a positive constant.

3.5 Supervised Random Walks

Supervised random walks is a technique that ultimately makes predictions using personalized random walks over the network with weighted edges. However, instead of using heuristically chosen edge weights, the edge weights are learned in a supervised way. Edge weights are assigned as a function f of the attributes of the edge and the involved nodes. Ideally, the parameters of this function would be set so the nodes that will link up to a node v in the future are given a higher score by the random walk than nodes that won't. However, finding parameters with this property is a highly constrained optimization problem that may not be solvable. Instead, the problem is relaxed to learning parameters for f that minimize a loss function that captures these constraints in a soft way. For a single node s, this minimization problem is:

$$\min_w F(w) = ||w||^2 + \lambda \sum_{d\in D, l\in L} h(p_l - p_d)$$

Where w are the parameters of the edge strength function f, λ is a constant that governs the regularization strength, h is a loss function, p_u is the random walk score of node u, D is a set of destination nodes that s links to in the future, and L is a set of no-link nodes that s does will not link to.

This loss function is minimized with a gradient-based optimization technique. Although the partial derivatives of the loss function cannot be computed exactly, they can be approximated with an iterative process. However, rather than re-implement this complex gradient computation process, we took the simpler approach of numerically computed the gradient with Newton's difference quotient. This technique is easier to implement while still running with acceptable speed an accuracy. See Backstrom and Leskov [1] for the full details of the supervised random walks algorithm.

We used a sigmoid as our edge strength function f: $f(u, v) = (1+\exp(\psi_{uv} \cdot w))^{-1}$ where ψ_{uv} is the feature vector for edge (u, v) and w is the learned weight vector. We used Wilcoxon-Mann-Whitney loss with width b as our loss function h: $h = (x) = (1+\exp(x/b))^{-1}$. We found that we needed a very low value of b (we found $b = 0.0005$ to work well) because the probabilities produced by the random walk were so low (the probability distribution is over thousands of nodes). Training the algorithm is quite slow and linear in the number of source nodes, so we chose a small set of 400 source nodes to train on. Since the predictions for each node rely on every edge weight in the graph and because the number of features is very small, we believe there still isn't that much danger of overfitting despite doing this. Backstrom and Leskov similarly chose a reduced set of nodes to train on, for example only using 200 for the Facebook dataset, which is comparable in size to the Yelp dataset we used [1]. We trained the algorithm for 50 iterations, which was sufficient for the algorithm to converge (the change in loss function became insignificantly small between iterations). We used the following features on edges:

- Age edge: $(T - t)^{-\beta}$, where T is a time cutoff and t is the edge creation time. We create three features with $\beta = \{1, 0.5, 0.2\}$.
- Rating: The number of stars of the review.
- Positive rating: Indicator on whether the rating was 4 or 5 stars.
- Bias: A constant feature with value 1.

In order to see how effective the learning algorithm was at minimizing the loss function, we computed the loss over the train set with our three random walk approaches. We found constants weights had a loss of 0.4395, heuristic weights had a loss of 0.4359, and the learned weights had a loss of 0.4350. Because of the nature of the WMV loss function and how the learning algorithm is constrained to follow the graph structure, the losses are fairly close in value. However, we do see that the algorithm successfully learned parameters that significantly lowered the loss function. Interestingly, the improvement over heuristically weighted edges was quite small. This is surprising because the algorithm did learn to assign high weights to features other than the edge age feature used in the heuristically weighted random walks, showing the new features were useful. In particular, the "rating" and "positive rating" features both got large positive weights. This makes sense intuitively because users are more likely to visit businesses similar to ones they like, so the edges where a user liked a business should be more useful in predicting what a user will visit next.

One reason the improvement was small could be that the edge age feature was significantly more important than the rating. The learned weights for the edge age features were about one and a half times as high as the rating feature, even though there are three variants of the age feature. Also, the number of reviews with a high age and thus very low weight because of that feature is quite large while the number of reviews with a very low rating and thus low weight because of that feature is quite small, so the edge age feature has a much greater capability for down-weighting "bad" edges. Another possibility for the small improvement in score is that the sigmoid in the edge strength function in supervised random walks reduces the weight of edges that would be given very high strength by the heuristically weighted random walks and this decreases performance. Using a different edge strength function like an exponential would possibly alleviate this problem and would be a good future experiment.

3.6 Supervised Binary Classifier

We also experimented with using a binary classifier to do link prediction. We trained it to predict, given two nodes, whether a link will occur between them in the future. After initially examining several algorithms including Support Vector Machines, Logistic Regression, and Random Forest, we found that Gradient Boosting Trees (GBT) performed the best, so we focused our efforts on fine tuning the GBT. Gradient boosting is an ensembling technique that sequentially builds estimators that improve accuracy on the train set. In the case of GBT, these estimators are small decision trees. GBT is an effective and commonly used technique when the number of features is not too large. Our model was trained with 500 height 4 trees. We used scikit-learn's[9] implementation of the algorithm. We trained this model using the results of all our unsupervised methods as features (each similarity metric both user-centric and business-centric, svd, unweighted and heuristically weighted random walks). We also included a few simple features based on node attributes: the degree of the user, the degree of the business, and the star rating of the business. Adding more node attribute features had little affect on performance, likely because the unsupervised methods provide far stronger signal than the simpler features.

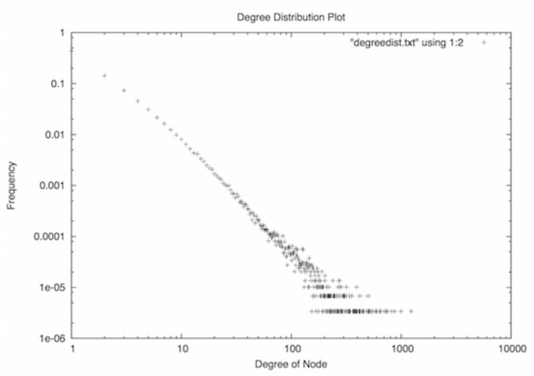
4 Dataset

We applied the algorithms to the Yelp Challenge dataset, which consists of information about businesses, reviews, and users on Yelp. This dataset was represented with a bipartite network where one side of the network represents users and the other side represents businesses. An edge goes between a user and a business if that user has written a review about the business.

4.1 Dataset Analysis

Some basic statistics were collected about the dataset in order to understand the general network structure of the dataset. All the data analysis was performed on the entire dataset including all users, businesses and reviews.

Property	Value
Number of nodes	294,615
Number of edges	1,090,251
Diameter	12
IsConnected?	false
Average Degree of Node	3.70



An analysis was done on the degree distributions of the nodes in the user set. We found out that the degree distributions follows a power law and we calculated the alpha to be 2.682 using the Maximum Likelihood Estimate with an x_{min} of 4. A plot of the degree distribution is above. The overall structure of the graph is that it is composed of one large connected component containing 99.6% of the nodes and the rest (about 250 weakly connected components) are smaller, encompassing up to 10 nodes each.

5 Results

5.1 Evaluation Methodology

Link prediction can be viewed as a binary classification task on whether there will be an edge between two nodes at some future time. To evaluate this, we created two "snapshots" of the network at time t and a later time t' and attempted to predict which pairs of nodes in the earlier snapshot will have an edge between them in the later snapshot. Specifically, we chose t as January 1st, 2013 and t' as July 1st, 2013 for our test set. We also created a training set for our supervised methods using earlier times: t_{train} as January 1st, 2012 and t'_{train} as July 1st, 2012. Because the way users review businesses might evolve over time, our test set may have systematic differences from our train set. However, we found the differences to be small enough that supervised learning was still be effective. Furthermore this division of train and test data matches how a system would run in the real world, where we only have information about prior links. We considered shifting our train set six months forward so it would be closer in time to the test set, but then the train set would be over different months than the test set and which businesses users tend to visit almost certainly depends on the season.

The large size of our graph means the predicting presence or absence of a link forming between every pair of nodes in the network at t is computationally infeasible for most of our algorithms. For example, at time t_{train} there are 128,349 users and 34,842 businesses in the network, resulting in about 4.5 billion candidate edges. Furthermore, the high sparsity of the network means this dataset has an extremely high class imbalance: between t_{train} and t'_{train} only 0.00148% of candidate edges are actually created, meaning negative examples outnumber positive ones over 65,000 to 1. To deal with these two problems, we construct a smaller and more balanced set to evaluate our algorithms on. We do this by applying two heuristics that reduce the size and class imbalance of the dataset. Both of these heuristics remove some positive examples, which will hurt recall, but remove far more negative examples, which will help with precision.

- Only taking users that have been active (i.e. that have written a review) in the last six months. This removes 12% of positive examples and 62% of negative examples.
- Only taking businesses at distance 3 from the current user as candidate edges. This further removes 22% of positive examples and 66% of negative examples.

Although this reduces class imbalance by more than a factor of 5, we unfortunately still required additional downsampling of negative examples to make the dataset manageable. Currently only 1% of negative examples are included in our evaluation. Given access to more computational processing, we would have done no downsampling and run over all candidate edges for each user.

One advantage of random walks (both supervised and unsupervised) is that they automatically produce a score for every candidate business given a user, meaning the second heuristic is unnecessary. However, we found that not applying the heuristic had a negligible impact on performance. We believe this is because that further away businesses (5 hops or more from the user) are given much lower PageRank scores than the 3-hop ones because there is a high chance of teleporting back to the user before reaching them. We also considered using a method like random walks to pick a set of candidate businesses instead of the simple 3-hop heuristic. However, we decided against doing this because combining algorithms like that complicates the evaluation. In particular, it seemed strange to evaluate algorithms weaker than random walks (the majority of our approaches) when we were already using the stronger algorithm to pick candidate businesses.

After applying the heuristics and sampling, we took 10,000 users and all sampled edges connected to those users as testing examples for evaluation.

Our algorithms assign a score to the existence of each possible edge at t'. We evaluated these scores according to two metrics:

- **area under curve (auc):** The area under the ROC curve generated from the scores produced from our classifiers. This reflects the quality of an algorithm as an overall predictor of the network structure in the future.
- **user precision at 20 (prec@20):** The percent of the top 20 highest scoring predicted edges going from a given user that will exist at time t'. This reflects the quality of our algorithm as a recommender since it measures how accurate we will be if we suggest 20 new businesses for a user to visit

5.2 Evaluation Results

A similarity metric followed by (U) means we took nodes at hop 2 from users and neighbors of businesses when computing similarity scores (see section 3.2). (B) means the reverse.

Model	**prec@20**	**auc**
Gold Standard	4.59	1.0
Random Baseline	1.40	0.502
Common Neighbors (U)	3.08	0.814
Common Neighbors (B)	3.03	0.815
Jaccard (U)	3.06	0.700
Jaccard (B)	1.51	0.703
Adamic Adar (U)	3.14	0.827
Adamic Adar (B)	3.06	0.844
Singular Value Decomposition ($k = 50$)	3.02	0.800
Random Walks	3.28	0.808
Weighted Random Walks	3.42	0.831
Supervised Random Walks	3.44	0.834
Supervised Classifier	3.44	0.914

5.3 Results Discussion

Overall, supervised random walks and the supervised classifier proved to be about equal as the most effective technique for prec@20, but the supervised classifier was by far the best for auc. Out of the similarity metrics, Adamic Adar performed the best and Jaccard performed the worst, which matches the results of Liben-Nowell and Kleinberg [8]. For every metric, the user-centric one proved superior, perhaps because most users have reviewed a very small number of businesses, so looking at their neighbors provides little information compared to looking at nodes at distance 2 from a user. SVD performed slightly worse than the similarity metrics. It's poor performance is perhaps because we did not apply any regularization, so the k features it learns for each user could be highly overfit for users with a small number of reviews. Learning edge weights with the supervised random walks algorithm did improve upon heuristically weighted edges, but the improvement was quite small. The supervised random walks section offers some discussion over why this might be.

It is worth noting that two metrics prec@20 and auc proved to be quite different measures of accuracy. For example, Jaccard (U) had a quite high prec@20 score but a low auc. This can be explained by the following difference in the metrics. Having a good auc means that a new link that does occur will be given a high score compared to the scores of potential links that do not occur.

Having a good prec@20 means that a new link that does occur will be given a high score compared to scores of potential links that do not occur *for that user*. In other words, auc requires having a notion of how likely a user is to form links as well as which links for the same user are more likely. This explains why supervised random walks is so unimpressive in terms of auc. It has absolutely no notion of which users are more likely to produce new edges; in fact it must assign scores so that the scores of all candidate edges for a user sum to 1. In contrast, the supervised classifier is trained in a way that maximizes its overall edge prediction ability as opposed to the per-user edge prediction ability, so it unsurprisingly has a much higher auc score.

6 Conclusion and Future Work

In this work, we implemented a variety of link prediction techniques and investigated how they could be applied to bipartite graphs. We found the commonly used link prediction similarity metrics could also be effective on bipartite networks if they were modified to by applied over a larger set of close nodes than just neighbors. We explored a variety of similarity metrics and found that the similarity methods that work well on homogeneous networks also work well on bipartite networks after the modification. We also explain how to exploit the structure of a bipartite graph's adjacency matrix in using matrix approximation methods and report results. Furthermore, we implemented several random walk based methods and found them to be particularly effective, especially when the edges in the network are weighted heuristically or assigned weights that are learned with the supervised random walks algorithm. However, our best performing method was a supervised classifier that combined the results from all of our unsupervised techniques. This demonstrates how unsupervised methods can be quite effective when combined together with a supervised classifier.

Unsurprisingly, our two supervised methods gave the best results, but they differ significantly and there are reasons why one might use either of them. The supervised classifier was by far the best method in terms of auc, meaning it is the strongest at predicting overall what new edges will be in the network in the future. However, it did not outperform supervised random walks in prec@20, meaning it is not necessarily better as a recommender. The supervised model has the disadvantage of being a much more complicated model relying on the results of many other unsurprised techniques, some of which are quite complicated themselves. Furthermore, it has to be applied separately to each candidate edge of interest, meaning it must rely on some other method to provide a reduced set of candidate edges for it to make predictions over. In contrast, supervised random walks can provide the score of every single candidate edge given a user in a reasonable amount of time. However, supervised random walks is slower to train because it requires many iterations of training, each one taking $O(|E|)$ time.

Although supervised random walks was quite effective, we were surprised by the lack of gains from doing supervised random walks over simple heuristically weighted random walks. As we suggest in the supervised random walks section, exploring different edge strength functions or trying new features might help raising this improvement. However, the relatively small improvement also illustrates the power of simple heuristic edge weighting. Heuristically weighted edges greatly outperforms unweighted edges and only slightly under performs supervised random walks despite being much simpler and faster to run. We also believe there is more potential for using matrix approximation algorithms in link prediction over bipartite networks. Our SVD method produces reasonable results, but still failed to outperform most other approaches. We believe this comes in part from lack of regularization causing overfitting, so one interesting future experiment would be trying out this model with regularization.

References

[1] L. Backstrom and J. Leskovec. Supervised random walks: predicting and recommending links in social networks. In *WSDM11*, pages 635644, 2011.

[2] N. Benchettara, R. Kanawati, C. Rouveirol. Supervised Machine Learning applied to Link Prediction in Bipartite Social Networks. In *Proceedings of the International Conference on Advances in Social Network Analysis and Mining.* IEEE, Los Alamitos, CA, 326-330, 2010.

[3] C. Eckart and G. Young. The approximation of one matrix by another of lower rank. In *Psychometrika.* 1(3):211-218, 1936.

[4] J. Kunegis, E. De Luca, and S. Albayrak. The link prediction problem in bipartite networks. In *Computational Intelligence for Knowledge-Based Systems Design.* 2010.

[5] Valdis Krebs. Mapping networks of terrorist cells. *Connections*, 24(3):4352, Winter 2002.

[6] L. Lovasz. Random Walks on Graphs: A Survey. In *Combinatorics, Paul erdos is eighty*, pages 1-46, 1993.

[7] S. Myers and J. Leskovec. On the convexity of latent social network inference. In *NIPS 10*, 2010.

[8] D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. In *12th CIKM*, pages 556-559, 2003.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. In *Journal of Machine Learning Research*, 12:2825-2830.

[10] Y. Yamanishi. Supervised bipartite graph inference. In NIPS, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds. MIT Press, 2008, pp. 18411848.

[11] B. Yan, S. Gregory. Detecting community structure in networks using edge prediction methods. In *Journal of Statistical Mechanics: Theory and Experiment 2012* 2012 P09008.