Link Prediction in Augmented Bipartite Graphs CS224W Project Report

Josh Grinberg, Prasad Kawthekar, and Manik Dhar
Stanford University

We are partnering with a start-up whose mission is to be a personal recruiter for a network of top companies in every industry. The goal of our project is to provide company recommendations for job-seeking candidates using data from LinkedIn provided by the start-up. This task can be framed as a network edge prediction problem: given a bipartite graph consisting of users and companies, predict which are the most likely user-company edges to be created in the future. Extensive research exists on link prediction, and link prediction for bipartite graphs specifically. Our project is unique because we have additional metadata: each user profile has a list of other user profiles viewed. Using this information, we can generate a user-user similarity network. We leverage this supplementary similarity network to improve upon existing edge prediction methods.

:

1. INTRODUCTION

Link prediction is an important problem when it comes to social and information network analysis. Recommendation systems for social networks effectively solve this problem. Liben-Nowell and Kleinberg [Liben-Nowell and Kleinberg 2003] solved this problem by using a number of similarity metrics over nodes. These metrics are defined beforehand and are not tuned to each specific graph. Other works involve methods which are trained on the graphs[Li and Chen 2013][Al Hasan et al. 2006][Kashima et al. 2009].

In the special case of bipartite graphs, general methods are not directly available. For example, similarity metrics like the Common Neighbors, Jaccard Coefficient, and Adamic-Adar are not directly applicable. Work has been done on modifying Kernel based methods for bipartite graphs[Liu and Yang 2015].

Our project involves a bipartite graph of users and companies, representing users' work history. Link prediction can be used to recommend jobs to users. Bipartite link prediction is applicable to other networks as well. For example, a Pinterest dataset may involve a split into two groups: the pins and the boards. Unlike other bipartite link prediction problems, our dataset has additional user-user edges which join similar users together. The edges connect the users whose accounts were most visited by the same people. We leverage these edges to improve upon existing link prediction methods. We show for a number of simple methods that using user-user edges significantly improves results.

2. LITERATURE REVIEW

For the goal of link prediction in networks, several heuristic-based approaches have been proposed. These include similarity-based techniques [Resnick et al. 1994][Sarwar et al. 2001], eigenvector-based node ranking [Huang et al. 2004][Huang et al. 2007a], node position-based [Fouss et al. 2012], and clustering coefficient-based techniques [Huang et al. 2007b]. All of these methods, except [Fouss et al. 2012] which uses random-walks to compute similarities, are not directly applicable to our method because they do not involve bipartite graphs. Approaches which combine node metadata can be tweaked to leverage user graph edges. We modify some similarity metrics to make them compatible for the given graph.

Model-based approaches that use statistical machine learning, in contrast to heuristics, have also recently enjoyed attention in literature owing to their success across tasks. The success of the collaborative filtering paradigm in heuristic based approaches has led many learning approaches to attempt to extend the paradigm in a dynamic setting. [Getoor and Sahami 1999] employ probabilistic relational models (PRM) with collaborative filtering. This work is extended to dynamic hierarchical class learning using hierarchical PRMs by [Newton and Greiner 2004]. An active learning method that uses probabilistic models with collaborative filtering to query user preferences in order to achieve better performance for new users is presented in [Yu et al. 2004].

A supervised learning technique that uses node proximity features, aggregated linkage features, and topological features for link prediction is presented in [Al Hasan et al. 2006]. Another supervised approach that uses logistic regression with topological and content-based similarity measures is due to [Wang et al. 2007]. Semi-supervised [Kashima et al. 2009] and unsupervised clustering based techniques [Reddy et al. 2002] have also been used for link prediction. Features to compute node similarity can leverage user graph edges.

As an alternative machine learning approach, graph kernel methods can be used to extract features from graph structure. As compared to feature-based methods, kernel methods do not require explicit node featurization, which can be computationally expensive or require extensive domain knowledge. Marginalized kernel [Kashima et al. 2003], Diffusion kernel [Kondor and Lafferty 2002], Commute time kernel [Fouss et al. 2007], Laplacian kernel [Yajima 2006] have all been used to capture features from graphs that can be then used for link prediction tasks. Another kernel based method, due to [Li and Chen 2013], can in fact incorporate both graph structural and node level features and combine them for its kernel calculation. We apply one graph kernel based approach which uses user graph edges as features.

3. PROBLEM DEFINITION

3.1 Input

—Bipartite graph $G_{uc} = (V_u, V_c, E_b)$ linking users to companies.

—User network $G_u = (V_u, E_u)$ linking users to other users.

—Set of users $U \subset V_u$ for whom to generate predictions.

3.2 Output

Set of companies $C \subset V_c$ for each user $u \in U$.

3.3 Evaluation Metric

Mean average precision at k (MAP@k)[kaggle]: The average precision at n for a user is:

$$ap@n = \sum_{k=1}^{n} P(k)/min(m,n)$$

where $P(k)$ means the precision at cut-off k in the item list, i.e., the ratio of number of recommended nodes followed, up to the position k, over the number k; $P(k)$ equals 0 when the k-th item is not followed upon recommendation; m is the number of relevant nodes; n is the number of predicted nodes. If the denominator is zero, $P(k)/\min(m,n)$ is set to zero. The mean average precision for N users at position n is the average of the average precision of each user, i.e.,

$$MAP@n = \sum_{i=1}^{N} ap@n_i/N$$

As the equation above shows, the order of predictions matters for this metric, but only if there is at least one incorrect prediction.

4. DATA

4.1 LinkedIn Data

Our start-up partner has provided us with sample user profiles and company profile data from LinkedIn in order to experiment on real-world data. The data has the following format:

—User profiles: These are the profiles created by LinkedIn users. For each user, the dataset contains previous work experiences (including company name and start date). The previous work experiences are used to build G_{uc}, the graph containing undirected edges between users and companies. Additionally, for each user, the dataset contains a list of other user profiles that are commonly co-viewed by other LinkedIn users ("People Also Viewed"). These adjacency lists are used to populate the graph G_u.

—Company profiles: These are the profiles created by LinkedIn companies (or auto-generated by LinkedIn otherwise). For each company, similar to the user profiles, the dataset contains a list of other company profiles that are commonly co-viewed by LinkedIn users ("Companies Also Viewed"). Since the "Companies Also Viewed" lists are very sparse, they were not used to generate company-company edges when populating the graph G_c.

A key challenge involved with using the provided LinkedIn data is that the dataset is large and edges are sparse. In total, there are 9 million company profiles and 100 million user profiles. The company dataset totals 27 gigabytes and the user dataset totals 240 gigabytes. Unfortunately, taking a random sample of the dataset is not effective because doing so does not produce enough edges. In fact, when sampling random sets of as many as 100,000 users, the average degree in the resulting G_u graphs is less than 1.0.

In order to identify a dense subgraph G_u of the user profile dataset, we developed the following algorithm. By processing small batches of the dataset at a time, and by making greedy (i.e. locally optimal) decisions, we successfully produced a subgraph with 7,304 users and an average degree of 5.958.

- Let G be the graph of all user profiles, n be the target number of nodes in the output subgraph, and $O(m)$ be the maximum number of nodes that can be stored in memory.
- Initialize the current subgraph S to be the empty graph.
- Phase 1. For each batch $B \subset G$ of m nodes:
  —Add B to S.
  —Remove nodes with degree of 0 from S.
  —Compute the largest connected component S_C of S.
  —If $|S_C| \geq n$, let $S = S_C$ and break.
  —Else, remove the $m-n$ nodes from $S \backslash S_C$ with the smallest degrees.
- Phase 2. Initialize a priority queue q storing the degree of each node in S in increasing order. Denote q_i as the i^{th} element in q. For each batch $B \subset G$ of m nodes:
  —For each node $i \in B$: If removing q_0 from S and adding i to S would increase the total number of edges in S (i.e. if i would have a larger degree in S than q_0 has), add i to S, remove q_0 from S, pop q_0 from the q, and insert i into q.
- Return S.

Phase 1 identifies a connected component S in G whose size is approximately n, and Phase 2 repeatedly swaps new nodes with nodes in S if they increase the total number of edges.

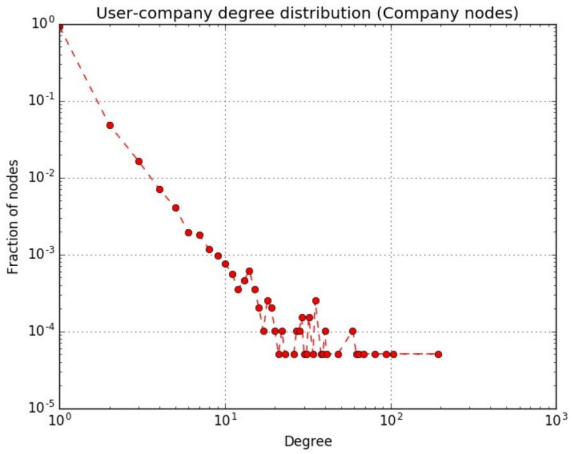

Fig. 1. User-company degree distribution for company nodes in LinkedIn dataset

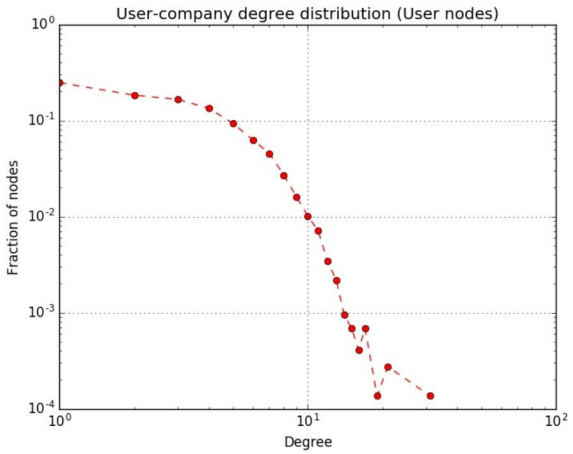

Fig. 2. User-company degree distribution for user nodes in Linkedin dataset

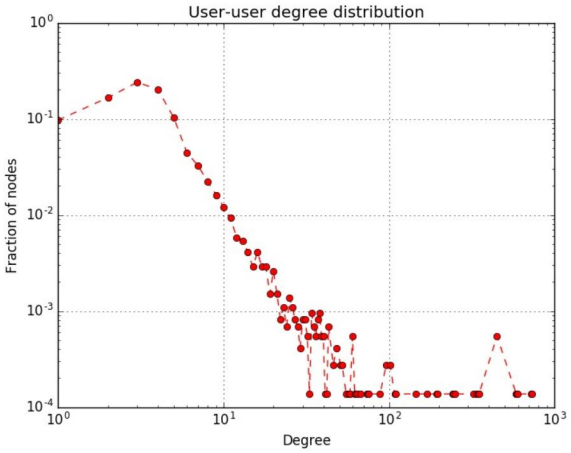

Fig. 3. User-degree distribution for Linkedin dataset

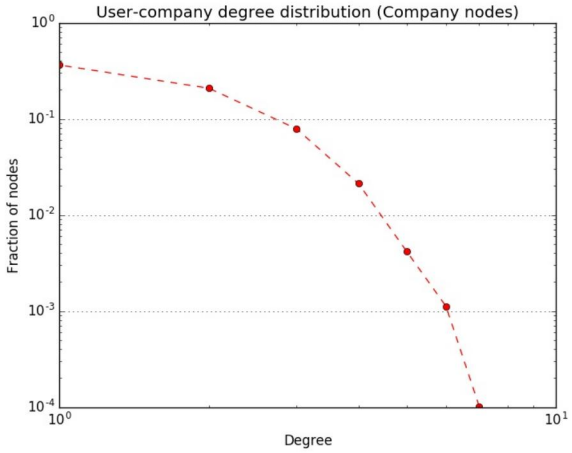

Fig. 4. User-company degree distribution for company nodes in Random graph

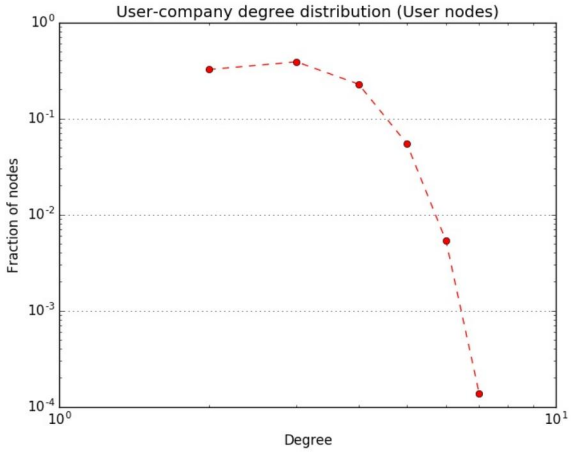

Fig. 5. User-company degree distribution for user nodes in Random graph

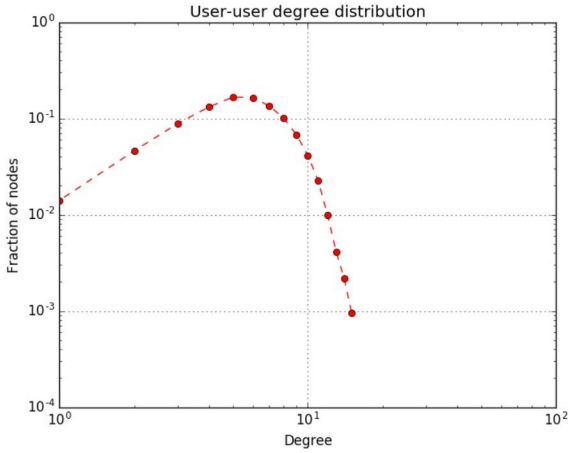

Fig. 6. User-degree distribution for Random graph

The figures above show the plots for the 'densified' graphs generated using the LinkedIn data. The first two subplots show the degree distribution for company and user nodes in G_{uc} respectively. The degree distribution of company nodes in the G_{uc} graph appears to follow the power law, and so does the the degree distribution of G_u which is shown alongside.

4.2 Synthetic Data

In addition to real-world data, we synthesize pseudo-random data using several different graph models described below. By comparing results on real-world data to results on synthesized data, we can better understand the successes and limits of our models.

—Erdos-Renyi Model: The user-user graph G_u and the company-company graph G_c are generated using a standard Erdos-Renyi random model. The user-company graph is generated by creating edges between each user node u and d companies sampled uniformly at random. The degree d of each user node is sampled from a normal distribution. This graph model helps serve as a benchmark for evaluating the extent to which our models can make use of structure found in other graphs.

—Latent Factors Model: In order to model our hypothesis that users and companies in the real-world are more likely to link to each other if they exhibit similar traits (such as personality/culture, interests/industry, location, etc.), we make use of a latent factor model when generating edges. Each user and company is represented as a vector $v \in \mathbb{R}^n$ of real values chosen uniformly at random. Each value in v represents the magnitude of the user or company's expression of some latent variable.

In order to mimic power-law node degree distributions found in real-world graphs, we developed the following algorithm to generate the user-user graph G_u and user-company graph G_{uc}:

- Number the nodes $n_1 \ldots n_k$.
- For each node n_i:
  —Sample a value d from a power-law distribution X.
  —Compute the cosine similarity, denoted s_{ij}, between latent factor vectors v_i and v_j for all $j \in \{1, \ldots, k\}$ and $i \neq j$. Let $s_{ij}^{(m)}$ denote the m^{th} largest s_{ij} for all $j \in \{1, \ldots, k\}$ and $i \neq j$.
  —Create an edge between nodes i and j for $j \in \{1, \ldots, d\}$.

This algorithm produces a graph whose degree distribution follows the power-law. To see this, observe that when each node n_i

is considered, d drawn from the distribution X edges are created. Each node n_i can also have edges if any other node happens to link to it. Thus, the expected degree of each node n_i is $2{*}\mathrm{E}[X]$, which is a scaled power-law distribution.

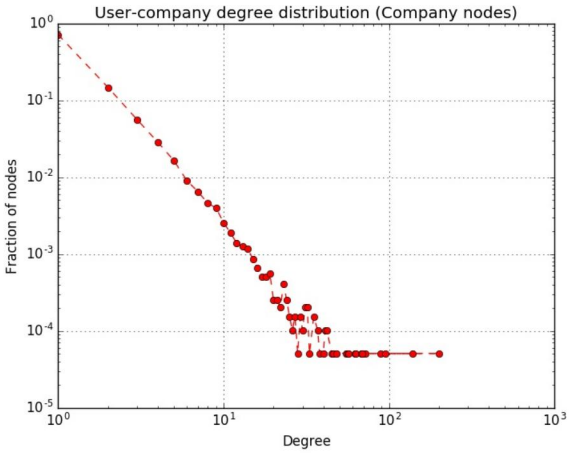

Fig. 7. User-company degree distribution for company nodes in Latent Factors graph

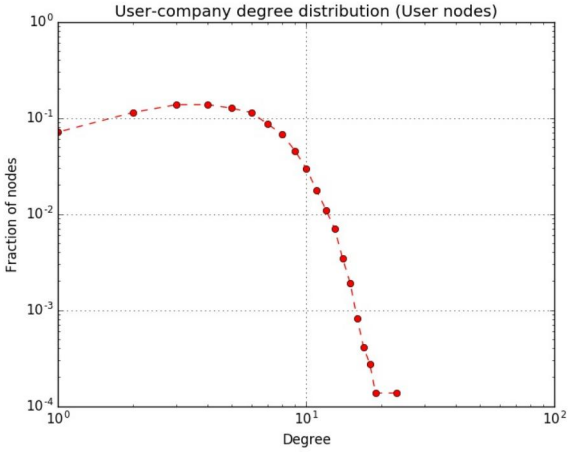

Fig. 8. User-company degree distribution for user nodes in Latent Factors graph

The degree distributions for user nodes and company nodes in the Latent Factors graph appear to mimic the degree distributions for the LinkedIn dataset, more so since the parameters for the latent-factors model are fine tuned for this task.

—Rewired LinkedIn Data: We run a rewiring algorithm on our graph where we randomly pick 2 edges and their end-points and switch their wiring (the metadata, such as employment start date, which we need for our train-test split, is kept with the same edges). The wiring is done for the user-company edges in a way to ensure bipartitenes. We randomly perform rewiring 15,000 times for the user-user graph and 15,000 times for the user-company graph.

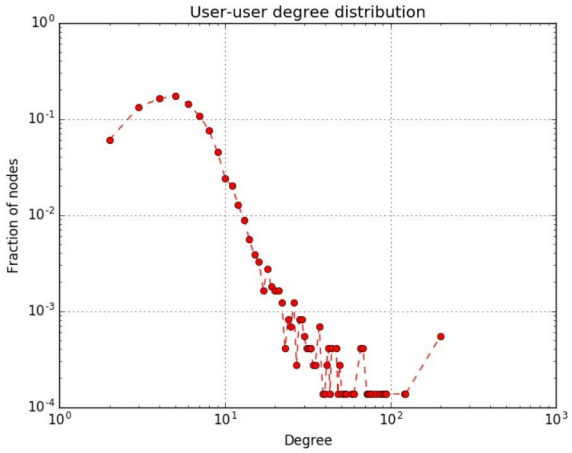

Fig. 9. User-degree distribution for Latent Factors graph

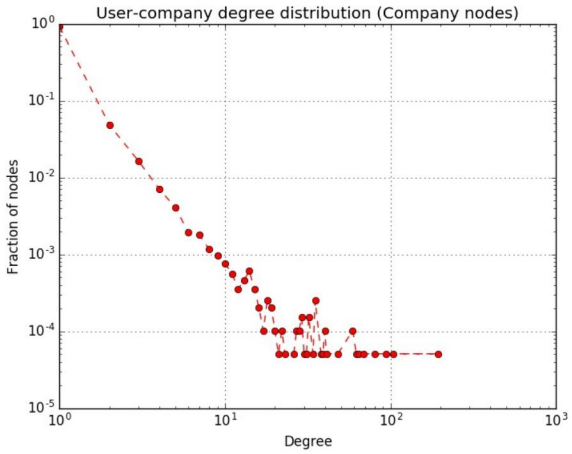

Fig. 10. User-company degree distribution for company nodes in Rewired graph

As a sanity check for the rewiring, above we plot the degree distribution for the nodes in the LinkedIn graph after rewiring. Clearly the degree distribution of the nodes remains the same before and after rewiring the graph, which confirms our implementation of the rewiring algorithm.

4.3 Data Splitting

—Train-test split: We randomly assign 70% of users for training, and the remaining 30% of users for testing. The graphs $G^{train} = \{G_u^{train}, G_c^{train}, G_{uc}^{train}\}$ and $G^{test} = \{G_u^{test}, G_c^{test}, G_{uc}^{test}\}$ represent the subgraphs of G_u, G_c, and G_{uc} restricted to the train or test users, respectively (6 graphs in total). Note that both the G^{train} and G^{test} graphs contain all companies. When testing, we evaluate predictions for one test user at a time. This means that when generating predictions for a test user i, a model can use all of the G^{train} graphs, but only the edges in G^{test} involving user i. Note G_c is just a set of nodes and has no edges and is the same for training and testing.

—Features-labels split: In addition, we hold out certain edges from the G_{uc}^{train} and G_{uc}^{test} graphs in order to determine which user-

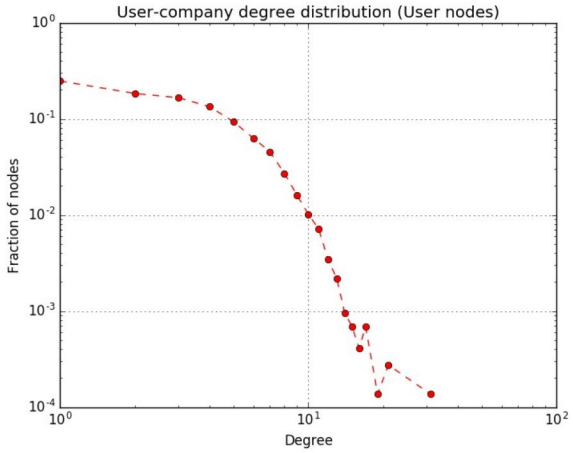

Fig. 11. User-company degree distribution for user nodes in Rewired graph

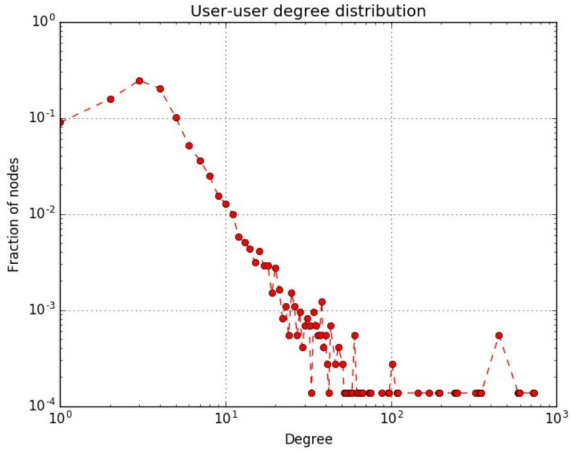

Fig. 12. User-degree distribution for Rewired graph

company edge predictions are correct. These held-out edges represent the ground truth. Rather than holding out a random set of edges, we determine a certain cutoff date in the past and hold out all edges that were created after this date. We are able separate edges temporally because user-company edges have timestamps. The reason we separate edges temporally is that this approach simulates the start-up's use of our system: the system is trained and tested using all the data available from the past, then when a new user creates an account, the system needs to predict edges that will be created in the future. We denote all the data before the cutoff date as *features* and all the data after the cutoff date as *labels*. Thus, the train and test user-company graphs are each split in two, resulting in $G_{uc}^{train,f}$, $G_{uc}^{train,l}$, $G_{uc}^{test,f}$, and $G_{uc}^{test,l}$. The cutoff date is set such that 70% of the user-company edges occur before it, and 30% of the edges occur after it.

—Train features-labels split: Since some of the models require labeled training data for supervised learning, we also split the training user-company data temporally. This results in $G_{uc}^{train,f,f}$ and $G_{uc}^{train,f,l}$. The train cutoff date is set such that 70% of the training user-company edges occur before it, and 30% of the training edges occur after it.

To summarize, in total we split our data into 9 graphs:

—Train graphs: G_u^{train}, G_c^{train}, $G_{uc}^{train,f,f}$, $G_{uc}^{train,f,l}$, and $G_{uc}^{train,l}$.

—Test graphs: G_u^{test}, G_c^{test}, $G_{uc}^{test,f}$, and $G_{uc}^{test,l}$.

4.4 Graph Statistics

As described earlier there are 4 graphs on which we test our algorithms: the LinkedIn Data, Erdos-Renyi graph model, latent factor graph model, and rewired LinkedIn Data. [t]

5. MODELS

We implement the following models in order to predict user-company edges that will be created in the future given data from the past.

—Random Baseline: The random baseline picks a company randomly from the set of companies a user hasn't worked at. All methods should exceed this performance if they are doing something intelligent.

—Metric Based Methods: One simple method for link prediction is to define a node similarity metric and order node pairs by decreasing similarity [Liben-Nowell and Kleinberg 2003]. These metrics are not directly applicable to Bipartite graphs. Also, they need to be modified to leverage the special user-user edges.
We define some notation for this discussion. $N(x)$ is the neighborhood set of a node x in a general graph G. Let u and c be a user and company node respectively. Given the nature of our problem, our similarity metrics will give us a number to compare u and c and not any 2 general nodes in our graphs. $N_{uu}(u)$ and $N_{uc}(u)$ is the neighborhood of u in the graph G_{uu} and G_{uc}. We also define $N_{ucu}(u)$ as the set of all nodes which are at a distance of two edges away from u in G_{uc}. Note that all of these nodes will be user nodes because G_{uc} is bipartite.
Let us look at 4 common similarity metrics used for link prediction.

- Common Neighbors: For two nodes x and y in a graph G we output $|N(x) \cap N(y)|$ which is the number of neighbors shared by x and y. In a bipartite graph this will fail because two nodes in the two parts will not share any neighbors. We propose two variants here for nodes u and c: $|N_{ucu}(u) \cap N_{uc}(c)|$ and $|N_{uu}(u) \cap N_{uc}(c)|$. The former will work on any bipartite graph and the latter leverages the extra G_{uu} structure. We see that we are basically just considering the user-neighborhood of each node. For user-user edges this will be the same as calculating this metric normally but normalizing it will be different as we discuss next.
- Jaccard Coefficient: For two nodes x and y in a graph G we output $\frac{|N(x) \cap N(y)|}{|N(x) \cup N(y)|}$. Our versions are: $\frac{|N_{ucu}(u) \cap N_{uc}(c)|}{|N_{ucu}(u) \cup N_{uc}(c)|}$ and $\frac{|N_{uu}(u) \cap N_{uc}(c)|}{|N_{uu}(u) \cup N_{uc}(c)|}$. This metric removes the effect of the relative sizes of the neighborhood sets. We don't use the number of companies a user connects to when considering user-user edges.
- Preferential Attachment: For two nodes x and y in a graph G we output $|N(x)||N(y)|$. Our versions are: $|N_{ucu}(u)||N_{uc}(c)|$ and $|N_{uu}(u)||N_{uc}(c)|$. This metric is based on the preferential attachment model and works with the intuition that neighbors with greater degrees will link together. For recommendation this is a very poor metric as this will pick the biggest company irrespective of industry. One

Statistics for the graphs we run our tests on

Graph Model	# of users	Avg user-user graph degree	# of user-user edges	# of companies	# user-company edges	Clustering coefficient in user-user graph	Average # of companies per user	Average # of users per company
LinkedIn Data	7304	5.958	21758	19659	25337	0.720	3.469	1.288
LinkedIn Data Rewired	7304	5.957	21755	19659	25337	0.114	3.469	1.288
Random Latent Factors	7304	5.957	21755	19659	36058	0.231	4.937	1.834
Random Erdos Renyi	7304	5.958	21758	19659	22141	0.001	3.031	1.126

modification we propose to fix this is to only look at companies which share common neighbors with u (the u-neighbor can be defined as $|N_{ucu}(u)|$ or $|N_{uc}(c)|$). We present results with our variant. With this modification it still prefers larger companies but in a more guided fashion.

- Adamic-Adar Coefficient: For two nodes x and y we output $\sum_{z \in N(x) \cap N(y)} (\log(|N(z)|))^{-1}$. This method also considers common neighbors between two nodes but gives more weight to nodes with smaller degrees. Our versions are: $\sum_{z \in N_{ucu}(u) \cap N_{uc}(c)} (\log(|N_{ucu}(z)|))^{-1}$ and $\sum_{z \in N_{uu}(u) \cap N_{uc}(c)} (\log(|N_{uu}(z)|))^{-1}$. We can instead of looking at the number of user neighbors $N_{uu}(z)$ (or $N_{ucu}(z)$ if not using uu edges), look at the number of company neighbors when $N_{uc}(z)$. We believe it makes more sense to use $N_{uc}(z)$ as selectivity should measured on the basis of the number of companies a person has worked at. We call these two variants Adamic-Adar user and Adamic-Adar company.

It is important to note we have moved away significantly from the original definition because for the neighborhood definition we selectively look at only user neighbors of the graph. For the plain bipartite case we look at users two hops away to construct such a neighborhood.

—User-User Collaborative Filtering: Similarity scores are computed for pairs of users based on the similarity of their user-company edges. The recommended companies for a given user i are the companies at which the users who are most similar to i have worked. In order to use information provided by user-user edges, we experimented with computing the weight of a user-company edge between user u and company c as the Jaccard Similarity of u's neighbors and the users who worked at c. Another idea we have for future exploration to incorporate user-user edges is to develop a hybrid recommender system that also makes use of content. In our case, we could represent the content of two users as features involving the user-user social network.

—Personalized Random Walks: Whereas simple metric-based methods are very local in nature, random walks can be used to create metrics leveraging large sections of the network structure. Generalizing random walks to include user-user and company-company edges is straightforward and may yield promising performance improvements. The hitting time $H_{x,y}$ from x to y is defined as the expected number of step it will take for a random walk to reach y starting from x. The negative of this can be taken as a similarity metric which would make nodes with smaller paths more similar. This metric can be very small if y happened to be a node with high stationary probability π_y irrespective of what x is. The measure is normalized to $-\pi_y H_{x,y}$. To reduce the dependency of this metric on nodes far away from x and y, the random walk is modified to the the page rank method where the walk is reset to x with some probability β. For our graph the random walk can also leverage the uu edges for its predictions. We run random walk simulations instead of calculating stationary values to allow for quick runs through the dataset.

—Graph Kernel-based Machine Learning:

[Li and Chen 2013] introduced a novel graph kernel that incorporates both node level features and graph structural context. This kernel can be used to measure the similarity between different edges in the network, and differentiate possible edges from impossible ones using a kernel-based classifier such as one-class SVM. The kernel similarity between two user-object pairs (in a bipartite graph G) is defined as:

$$k_g(\overline{uo}, \overline{u'o'}) = \sum_{uo \subset h} \sum_{u'o' \subset h'} \left[k_{path}(h, h') P(h|G) P(h'|G) \right]$$

Where h, h are random walk paths on G starting from node pairs uo and uo, and where

$$k_{path}(h, h') = k_{node}(n_x^u, n_x^u) \times k_{node}(n_{x-1}^u, n_{x-1}^u) \times \ldots \times k_{node}(u, u') \times k_{node}(o, o') \times \ldots \times k_{node}\left(n_{y-1}^o, n_{y-1}^o\right) \times k_{node}\left(n_y^o, n_y^o\right)$$

$$P(h|G) = p_s(n_x^u) p_t(n_x^u | n_{x-1}^u) \cdots p_t(n_2^u | n_1^u) p_t(n_1^u | u) p_t(n_1^o | o) p_t(n_2^o | n_1^o) \cdots p_t\left(n_y^o \middle| n_{y-1}^o\right) p_s\left(n_y^o\right).$$

k_{node} measures the similarity between two nodes, $p_s(i)$ is the stop probability of random walks at node i, and $p_t(i|j)$ is the transition probability from node j to node i.

The kernel k_g lends a natural interpretation to the task of link prediction in the graph. It measures the joint similarity of the structure of subgraphs around user-item pairs (approximated by random walks upto a fixed length) and features of nodes in these subgraphs. To adapt this algorithm to our use-case we extended K_node as follows. Since LinkedIn dataset is not strictly a bipartite graph, it also contains edges between user pairs and between company pairs. These edges correspond to the People Also Viewed feature on LinkedIn, which indicates users/companies (henceforth, nodes) that are similar (according to some hidden distributed metric of similarity determined by the populace) to a given node. These edges were held as representative of node similarity, such that the similarity of a pair nodes (measured by K_node) 1 if either of the nodes occurs in the others' "People Also Viewed" list.

6. RESULTS AND ANALYSIS

The table on the next page tabulates the results for all the methods on the 4 graphs we discussed about earlier. We discuss the results of our experiments and make some interesting observations. We consider Graph Kernel-based Learning separately as its execution was extremely slow so we don't have comprehensive results. Also we don't have a non-uu variant of it (for that we need additional metadata to exploit).

6.1 Best Performance

Simulation of random walks gives us the best results in all cases except one. This shows the power of these walks to exploit the structure of the graphs to make such predictions without any modifications.

6.2 User-user vs. no user-user Graphs

We consistently observe for all methods that leveraging uu edges consistently yields better performance. Which makes sense as these edges provide added structure for the methods to exploit.

6.3 Variation over Graphs

We see the best results are possible on the random latent factor graph. This is because the nodes are connected in a very structured way (leading to more regularity as compared to the real world dataset). The closer the latent factors are on the hypersphere the more likely it is that the nodes will have an edge between them.
We note that rewiring destroys the underlying structure of the graph (as expected) and leads worsening in the performance of the algorithms. It is ineresting to not that uu edges still provide additional benefits even though the rewiring of user-user graph and user-company graph was independently done. This is probably because the degree distribution still carries some useful information which the methods can exploit.
We observe the most extreme scenario in the Random Erdos Reyni model which makes sense as the user-user connections are independent of the user-company edges. Therefore, uu edges are completely misleading and lead to all methods failing. All methods not using uu edges give the same performance. We note the random walk method uses uu edges to give better performance. This is probably because the random uu edges let the random walk explore farther away nodes which can come into easily easily in an erdos-reyni graph model where each edge has equal probability.

6.4 Everyone wants to work for Google

We observe that among all methods based on simple similarity metrics preferential attachment gives the best results. This lends some credibility to the intuition that people on average prefer working (or end up working) in big companies.
We note preferential attachment works extremely well in the latent factor model. This is probably because the latent factor model will prefer joining users to high degree company nodes. This also indicates that there might be more structure at play in the LinkedIn dataset which the latent factor method can not capture.

6.5 Collaborative Filtering

We note that when not considering uu edges collaborative filtering gives the best performance. The only exception is in the case of the rewired graph. This is probably because they loose a lot of their structure. Interestingly we get the best performance on Erdos-Renyi with collaborative filtering. Perhaps it captures uniformly random distributions better than other methods.
Unfortunately, our variant with uu edges performs worse than vanilla collaborative filtering. Even so the performance degradation is not very bad. We believe further exploration in this direction can lead to better results.

6.6 Graph Kernel-based Learning

Although this method gives reasonable results, it takes a lot of time to run (approx 4 hrs per experiment). We believe such a method would require far more richer metadata to reach its full potential. Even so on the linkedIn dataset we get the second best performance using Graph Kernel Learning.

6.7 Very Low values

Even though all methods perform much better than the random baseline, we note that the values is still very low. This is simply because the problem of predicting links in large graphs with sparse edges is difficult. Interestingly a graph in a given state can presumably evolve in a huge number of ways. Our methods are supposed to act as recommender systems and what would be interesting to measure is how useful they are to users when deployed on a professional network. This analysis is not possible with our dataset.

6.8 Effect of density on random walk performance

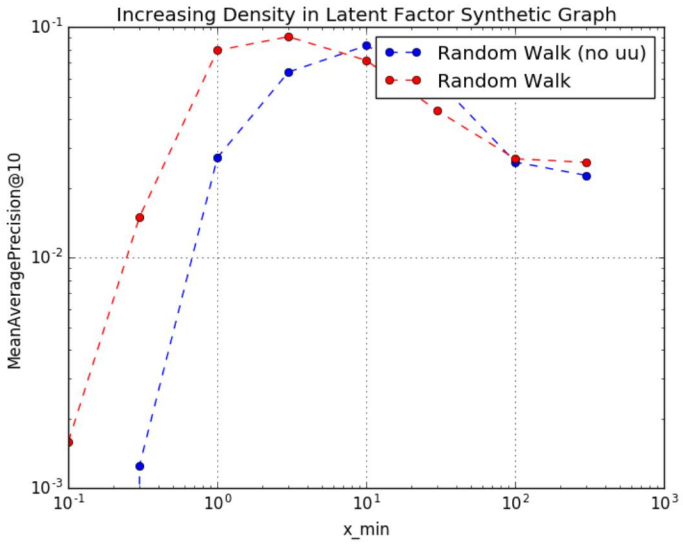

Fig. 13. Change in results with density for latent factor model

The chart above plots the performance of the Random Walk model (with and without user-user edges) on latent factor synthetic graphs as the density of the latent factor synthetic graphs increases (as the parameter x_{min} increases, the number of edges in the latent factor synthetic graphs increases linearly). The results demonstrate that for sparse graphs, user-user edges provide valuable information. However, as the graphs become very dense, performance decreases. This seems reasonable because when there are many edges and as the graph becomes a fully connected graph, each edge becomes less significant. The result is that the Random Walk model is able to extract less information from the structure of the graph.

MAP@10 results of various methods on the 4 graphs

Algorithm	LinkedIn Data	Rewired LinkedIn Data	Random Latent Factors	Random Erdos Renyi
Random Baseline	0.000094	0.000109	0.000104	0.000000
Common Neighbors (with uu)	0.021499	0.000782	0.055694	0.000000
Common Neighbors (no uu)	0.003346	0.000084	0.010583	0.000188
Jaccard Coeff. (with uu)	0.016202	0.000494	0.035209	0.000000
Jaccard Coeff. (no uu)	0.001998	0.000000	0.005144	0.000188
Adamic-Adar user (with uu)	0.020798	0.000796	0.053531	0.000000
Adamic-Adar user (no uu)	0.003031	0.000380	0.008658	0.000126
Adamic-Adar comp. (with uu)	0.021499	0.000782	0.055694	0.000000
Adamic-Adar comp. (no uu)	0.003346	0.000084	0.010583	0.000188
Preferential Attachment (with uu)	0.027375	**0.002850**	0.062309	0.000000
Preferential Attachment (no uu)	0.007967	0.001393	0.021054	0.000188
Random Walk Sim (with uu)	**0.043612**	0.002102	**0.067047**	0.000220
Random Walk Sim (no uu)	0.007440	0.001976	0.021455	0.000188
Collaborative Filtering (with uu)	0.010293	0.000477	0.029650	**0.000377**
Collaborative Filtering (no uu)	0.012354	0.000587	0.031656	**0.000377**
Graph Kernel-based Learning (with uu)	0.034408	(not run)	0.034408	(not run)

REFERENCES

Mohammad Al Hasan, Vineet Chaoji, Saeed Salem, and Mohammed Zaki. 2006. Link prediction using supervised learning. In *SDM06: workshop on link analysis, counter-terrorism and security*.

François Fouss, Kevin Francoisse, Luh Yen, Alain Pirotte, and Marco Saerens. 2012. An experimental investigation of kernels on graphs for collaborative recommendation and semisupervised classification. *Neural networks* 31 (2012), 53–72.

Francois Fouss, Alain Pirotte, Jean-Michel Renders, and Marco Saerens. 2007. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Transactions on knowledge and data engineering* 19, 3 (2007), 355–369.

Lise Getoor and Mehran Sahami. 1999. Using probabilistic relational models for collaborative filtering. In *Workshop on Web Usage Analysis and User Profiling (WEBKDD'99)*.

Zan Huang, Hsinchun Chen, and Daniel Zeng. 2004. Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering. *ACM Transactions on Information Systems (TOIS)* 22, 1 (2004), 116–142.

Zan Huang, Daniel Zeng, and Hsinchun Chen. 2007a. A comparative study of recommendation algorithms in e-commerce applications. *IEEE Intelligent Systems* 22, 5 (2007), 68–78.

Zan Huang, Daniel D Zeng, and Hsinchun Chen. 2007b. Analyzing consumer-product graphs: Empirical findings and applications in recommender systems. *Management science* 53, 7 (2007), 1146–1164.

kaggle. Mean Average Precision. (????). https://www.kaggle.com/wiki/MeanAveragePrecision

Hisashi Kashima, Tsuyoshi Kato, Yoshihiro Yamanishi, Masashi Sugiyama, and Koji Tsuda. 2009. Link propagation: A fast semi-supervised learning algorithm for link prediction.. In *SDM*, Vol. 9. SIAM, 1099–1110.

Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. 2003. Marginalized kernels between labeled graphs. In *ICML*, Vol. 3. 321–328.

R Kondor and J Lafferty. 2002. Diffusion kernels on graphs and other discrete structures. In *Proc. Int. Conf. on Machine Learning*. 315–322.

Xin Li and Hsinchun Chen. 2013. Recommendation as link prediction in bipartite graphs: A graph kernel-based machine learning approach. *Decision Support Systems* 54, 2 (2013), 880 – 890. DOI:http://dx.doi.org/10.1016/j.dss.2012.09.019

David Liben-Nowell and Jon Kleinberg. 2003. The link prediction problem for social networks. In *CIKM '03 Proceedings of the twelfth international conference on Information and knowledge management*. ACM Press, New York, NY, USA, 556–559. DOI:http://dx.doi.org/10.1145/956863.956972

Hanxiao Liu and Yiming Yang. 2015. Bipartite Edge Prediction via Transductive Learning over Product Graphs. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*. 1880–1888.

Jack Newton and Russell Greiner. 2004. Hierarchical probabilistic relational models for collaborative filtering. In *Proc. Workshop on Statistical Relational Learning, 21st International Conference on Machine Learning*.

P Krishna Reddy, Masaru Kitsuregawa, P Sreekanth, and S Srinivasa Rao. 2002. A graph based approach to extract a neighborhood customer community for collaborative filtering. In *International Workshop on Databases in Networked Information Systems*. Springer, 188–200.

Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. 1994. GroupLens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*. ACM, 175–186.

Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*. ACM, 285–295.

Chao Wang, Venu Satuluri, and Srinivasan Parthasarathy. 2007. Local probabilistic models for link prediction. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*. IEEE, 322–331.

Yasutoshi Yajima. 2006. One-class support vector machines for recommendation tasks. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 230–239.

Kai Yu, Anton Schwaighofer, Volker Tresp, Xiaowei Xu, and H-P Kriegel. 2004. Probabilistic memory-based collaborative filtering. *IEEE Transactions on Knowledge and Data Engineering* 16, 1 (2004), 56–69.