Group 14
Predicting Yelp Reviews using Internal Links - Link Prediction in Bipartite Networks

Andrew Giel
BS Stanford 2015, Computer Science
agiel@stanford.edu

Siddharth Jain
MS Stanford 2014, Computer Science
sjain2@stanford.edu

Kelly Wang
BS Stanford 2014, Computer Science
kekewang@stanford.edu

***Abstract*—Social Networks may exist as bipartite graphs which evolve with new links being continuously added between the two disjoint set of nodes. To understand the dynamism of such bipartite networks, it is useful to predict links between existing nodes. Using the Yelp social network between users and businesses, we attempt to predict user reviews on businesses and the rating of predicted reviews. In order to accomplish this goal, we implement the algorithm of internal links in bipartite graphs. In this paper, we describe the method used and evaluate its effectiveness in predicting user reviews in the Yelp network. We conclude that this algorithm is not particularly suited to highly clustered social networks due to the chronological disparity in user reviews and relatively low number of reviews per user.**

I. Introduction

Many real-life interactions can be modeled as bipartite social networks in which two distinct groups of nodes arise, with edges from one group to the other group representing some form of interaction. Examples include networks between product-consumer (Goodreads, Yelp), user-tag (Instagram, Delicious), user-comment or user-reviews. These networks are very dynamic as new users/consumers continuously enter the graph and new links are formed as social interactions occur. Therefore, knowledge of how these networks will evolve is important for organizations maintaining these social networks. Predicting user behavior or developing recommendations give these organizations a greater insight about the behaviors of the network and potential monetary gain.

In this paper, we take the Yelp social network around Stanford University, where users review businesses, and predict what new reviews will occur between the original set of users and businesses. New reviews are represented by new links that are formed in the social network. Our goal would be to predict these links along with the associated rating of the review. We accomplish this goal by implementing the internal links method for bipartite link prediction. First, we give an overview of previous work involving link prediction, including the internal links method, which is especially relevant to our project. We then describe our data and some of its characteristics. We preface our algorithm explanation with terms and mathematics necessary before delving into the internal links method. After establishing our evaluation metrics, we present the results of our experiment, and provide an analysis of the results seen. We conclude with our final thoughts on the internal links algorithm and possibilities for future research.

II. Review of Prior Work

A considerable amount of research has examined link prediction in the past. As part of this paper, we narrowed our focus to the task of link prediction for bipartite graphs. The following two papers are our driving force in developing this research and provide a substantial overview of the link prediction methodology.

A. The Link Prediction Problem for Social Networks

Liben-Nowell and Kleinberg, in their paper, provide the use cases for link prediction. They discuss the methods used for link prediction, which are broadly split into neighbor related methods, path related methods or higher-level approaches. Each of the methods aims to capture the similarity between nodes, the intuition being that the higher the similarity, the more likely the two nodes are to form a link.

Neighbor related methods are those where a metric between neighboring nodes in a graph is used to determine the similarity between nodes. Methods that fall into this category are:

- Similarity in distance from nodes to the graphs connected component
- Number of common neighbors
- Jaccards coefficient - The probability that two nodes are neighbors
- Preferential attachment - Link probability based on the product of the number of their neighbors between two nodes

Path related methods try to perform random walks to determine the similarity of nodes:

- Katz - Sum of weighted path lengths between two nodes
- Normalized hitting and commute times - The number of times x hits y on a random walk starting at x
- SimRank - Recursively defining the similarity of two nodes based on how many neighbors they share

Other higher level approaches such as Low-rank approximation take the adjacency matrix of three applications: ranking by Katz, ranking by common neighbors, and defining the weight of (x, y) as the (x, y) entry in the matrix and use that to determine the similarity between nodes.

The paper then applies these techniques to the co-authorship network and analyzes the results. It concludes that results are not very accurate as authors choose to co-author with other authors for reasons not in the scope of a graphs attributes. Though all of these methods present interesting possibilities, we chose to use neighbor related methods as they are more suitable to a bipartite graph. Paths in bipartite graphs are not as meaningful as neighbors, and the graph is too sparse for higher level approaches such as adjacency calculations.

B. Internal Link Prediction: A New Approach for Predicting Links in Bipartite Graphs

Allali, Magnien, and Latapy use variants of the neighbor related methods from the previous paper and apply them to bipartite graphs. Instead of predicting directly on a given graph, the paper makes a projection graph and assigns weights to the projection graphs edges. The paper uses this weighted projection graph to predict links. These concepts will be further elaborated in the algorithm section of this paper. The paper used three different kind of weighting metrics for their edge weights:

- Number of common neighbors
- A variant of the Jaccard coefficient - Accounting for the discrepancy between the number of neighbors two nodes may have.
- A variant of preferential attachment - Weighing node's neighbors by the total number of neighbors present. If the node has fewer neighbors, each neighbor is given a higher weight.

The paper defines a threshold weight which is used to determine if an edge can be predicted. These methods were tested on a user-group dataset, a user-comment graph from Flickr, and user-tag graphs from Delicious. Our paper expands on the methods employed by this paper to perform link prediction on bipartite graphs.

III. Data Collection

We obtained our data from Yelp in the form of the Yelp Academic Dataset. Our data consists of businesses, users, and reviews from around 30 different universities across the United States. The data includes the rating for each review, the text for each review, and other metadata. For this paper we are primarily focused on the structure of the network as opposed to the metadata that surrounds it, yet we leverage the additional data such as the ratings given by the user and the date of the review. We take each review and use the user ID and business ID to identify our nodes while the review itself forms the edge in our graph. The rating by the user and date of the review are placed as weights on the edge in our graph. Using this process we generate a bipartite graph, in which the users and businesses are in two disjoint sets of nodes and the reviews serve as edges from one set to the other.

We used the network based around Stanford University as the dataset to develop and implement link prediction. For this reason, when we refer to the dataset, we mean only the Stanford dataset and not the entire Yelp Academic Dataset. We created the Stanford dataset by selecting the businesses that were identified by Yelp as nearby to Stanford, and then selecting reviews made on these businesses and the users that made those reviews. This way we have each business, review, and user that was associated to be in the Stanford area.

IV. Exploratory Analysis

We ran preliminary analyses on all the universities' data consolidated (All-School network) and the Stanford network in particular. Based on our findings, it is proper to note that both the All-School network and the Stanford network display power law degree distributions for users and businesses. The users have a much higher alpha as compared to businesses, suggesting that there are more users that are outliers and review many businesses. This causes the power-law distributions slope to be steeper in the log-log scale. Businesses on the other hand have a lower alpha, as their degree distribution follows a more uniform pattern. Fig 1. and Fig 2. show the power law distribution of the All-School network users and businesses.

Fig. 1. Power Law distribution of Businesses in the Yelp dataset for all universities.

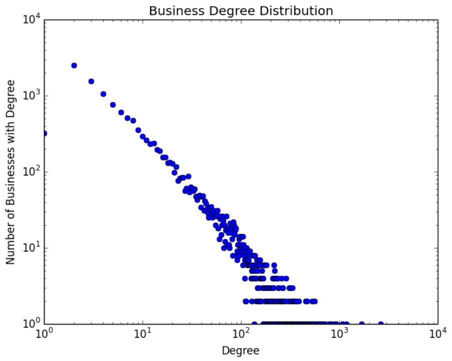
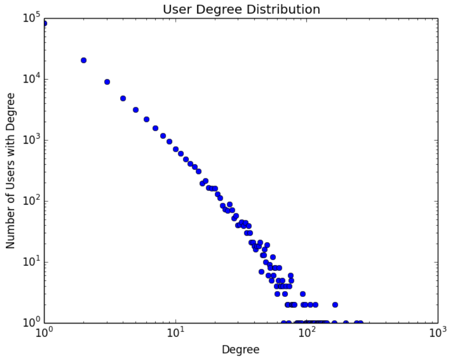


Fig. 2. Power Law distribution of Users in the Yelp dataset for all universities.



The Stanford Network for Yelp has a much lower proportion of businesses as compared to users suggesting that there are more reviews for a given business. This leads to users having a higher alpha in their power law distribution. This collaborates our findings that on average users review about 2 businesses which suggests a sparse bipartite network between users and businesses. The Stanford network is therefore highly clustered around business nodes which tend to have a high in-degree (many reviews for a business).

TABLE I. ALL-SCHOOLS NETWORK - ANALYSIS

Number of Businesses:	13481
Number of Users:	130873
Number of Reviews:	330071
Average Review Rating (out of 5):	3.65
Number of Nodes:	144354
Number of Edges:	330071
Business Alpha :	1.46
User Alpha :	3.09
Alpha (All nodes):	2.58
Number of Weakly Connected Components:	731

TABLE II. STANFORD NETWORK - ANALYSIS

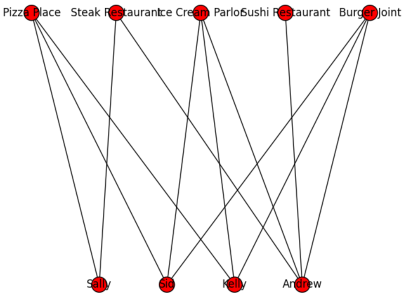
Number of Businesses:	499
Number of Users:	9726
Number of Reviews:	15990
Average Rating:	3.61
Number of Nodes:	10225
Number of Edges:	15990
Business Alpha :	1.44
User Alpha :	4.70
Alpha (All nodes):	3.74
Number of Weakly Connected Components:	47

V. DEFINITIONS

The internal links method for bipartite link prediction relies on understanding two crucial concepts, the induced link and the internal link.

We begin with a bipartite graph $G = (A, B, E)$ where A and B are the two distinct groups of nodes and E is the set of edges where every edge $(a, b) \in E$, $a \in A$, $b \in B$. In order for our algorithm to work, a projection graph, $G_{proj} = (V, E_{proj})$ where $V \subseteq A$, must be created. We create G_{proj} from G, using the notion of induced links. An induced link $(u, v) \in E_{proj}$ where $u \in V$, $u \in A$, $v \in V$, $v \in A$ is created when the condition $\exists i \in B$ where $(u, i) \in E$ and $(v, i) \in E$. Less formally, an induced link is a link added to G_{proj} between two nodes in the same subset of G, where both of the nodes have at least one edge to a node in the other subset of G. Subsequently, G_{proj} is made entirely of induced links, and represents the nodes within G that share endpoints.

Fig. 3. Example of a bipartite graph between users and businesses.



Once a projection graph G_{proj} has been created, internal links can be identified. An internal link is a link $(a, b) \notin E$, $a \in A$, $b \in B$ such that if (a, b) were added, the induced links added to G_{proj} would create a new projection graph G'_{proj} such that $G_{proj} = G'_{proj}$. An internal link is a link within the bipartite graph that does not add any new induced links to a projection graph.

VI. ALGORITHM

In this portion we describe our instantiation of the internal links algorithm for bipartite link prediction. At a high level, this involves creating a projection graph G_{proj} from the original bipartite graph G, weighting the links within G_{proj} via multiple measures of node-to-node similarity, identifying all possible internal links, and finally narrowing down to internal links that are most likely to occur by eliminating those that do not meet a certain threshold of similarity. Once we have predicted links, we describe our own algorithm to provide a user-rating to the link based on neighboring links.

We create our graph $G_{main} = (Users, Businesses, Reviews)$ from our data. This graph is bipartite and representative of the actual Yelp network as shown in Fig. 3. We then create a projection graph.

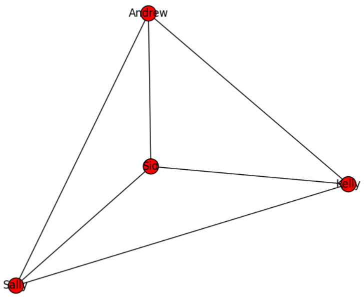
A. Algorithm 1: Creating Projection Graph

Input ← G_{main}- Bipartite Graph
Output → G_{proj} - Projection Graph

```
Gproj = new Graph()
for review in Gmain:
    b = review.business
    u = review.user
    for reviewOfB in Gmain.edges(b):
        v = reviewOfB.user
        if u != v:
            Gproj.add_edge(u, v)
return Gproj
```

This process involves looping through $Users$ and finding all $Reviews$ each user u has made. For each review r, we get a business b. For each business b we see which users v also link to b. If $v \neq u$, then we create the induced link, an edge within G_{proj} (u, v). Once we have iterated through each $User$ then we have our projection graph, G_{proj}.

Fig. 4. Corresponding projection graph of users from Fig. 3.



While creating our projection graph, we weight the induced edges. There are a variety of ways to accomplish this. We outline three of these methods based on Allali, Magnien, and Latapys paper. We then modify each of these to include review ratings from our dataset, providing three more methods.

The first method is simply to sum up the number of common businesses reviewed by the two users. This is the Sum method. Formally, this is

$$Sum = |Reviews(u) \cap Reviews(v)|$$

where u and v are two nodes with an induced edged between them and $Reviews(user)$ returns all of the businesses reviewed by a user. This method is very simple, and gives high weights to users who have a lot of reviews.

The Jaccard method works to reduce this preference to users with high numbers of reviews, by normalizing over the total number of reviews between the two nodes. Formally,

$$Jaccard = \frac{|Reviews(u) \cap Reviews(v)|}{|Reviews(u) \cup Reviews(v)|}$$

again where u and v are two nodes with an induced edged between them and $Reviews(user)$ returns all of the businesses reviewed by a user.

We also use another method, known as the Delta weight function, which assesses the relative importance of a shared review in terms of how many reviews the business has. Formally,

$$Delta = \sum_x \frac{2}{|Reviewed(x)| \cdot |Reviewed(x)| - 1}$$

where

$$x \in Reviews(u)Reviews(v)$$

and where u and v are two nodes with an induced edged between them. $Reviews(user)$ returns all of the businesses reviewed by a user, and $Reviewed(business)$ returns all of the users which reviewed that business. Note that this function works by summing up the inverse of the maximum number of connections between the users which reviewed the business.

The Yelp dataset gives us the ability to add more features to the weighting of our induced edges. Previous work with this algorithm relied only on unweighted bipartite networks. Yelp allows users to rate their experience at a business or establishment on a 1-5 scale, which we use as weights on the bipartite edges. This gives us extra information not only on the nature of the interactions in a network, but also on the similarity between users.

In order to take advantage of this extra information, we have supplemented the original three weighting functions with a similarity coefficient C which is derived from the similarity in ratings between the two users. The intuition is that two users are more likely to review the same business if they have had similar experiences at prior businesses. More formally, within a bipartite graph $G_{main} = (A, B, E)$ we have two user nodes $U \in A$ and $V \in A$ and a business node $W \in B$ where $(U, W) \in E$ and $(V, W) \in E$. Both (U, W) and (V, W) have a weight associated with them $\omega(U, W)$ and $\omega(V, W)$ respectively. We calculate the similarity coefficient C for this triad of nodes as follows:

$$C(U, V, W) = \omega MAX - abs(\omega(U, W) - \omega(V, W))$$

In our dataset, this means the max weight for a review (ωMAX = 5) minus the difference in the reviews (maximum of 5 - 1 = 4). Essentially, this similarity coefficient serves to make reviews on the same business from two different users worth less if the users reviewed the business differently. We believe that this serves as a measure of similarity, or a quantification of similar taste.

We present our previous three weighting techniques modified to include our similarity coefficient here.

$$Rating - Sum = \sum_x C(u, v, x)$$

$$Rating - Jaccard = \frac{\sum_x C(u, v, x)}{|Reviews(u) \cup Reviews(v)|}$$

$$Rating - Delta = \frac{2 \sum_x C(u, v, x)}{|Reviewed(x)| \cdot |Reviewed(x)| - 1}$$

where

$$x \in (Reviews(u) \cap Reviews(v))$$

Without this modification, weighting the edges of the projection graph serves to eliminate internal links that do not have strong enough induced links due to lack of overlap in common edges. This additional similarity coefficient C will eliminate internal links for dissimilar users, thereby reducing recall and increasing precision.

We also created a weighting entirely dependent upon C, which we refer to as the Rating Least-Squares Weight. This weighting works by taking sum of the squares of C. More formally:

$$Rating - Least - Squares = \sum_x (C(u, v, x))^2$$

again where

$$x \in (Reviews(u) \cap Reviews(v))$$

Using these weighting functions we create a total of seven weighted projection graphs for a given bipartite graph, one for each weighting function: Sum-Weight (SW), Jaccard-Weight (JW), Delta-Weight (DW), Rating-Sum-Weight (RSW), Rating-Jaccard-Weight (RJW), Rating-Delta-Weight (RDW) and Rating-Least-Squares-Weight (RLSW).

B. Algorithm 2: Identifying Internal Links

Input ← G_{main}- Bipartite Graph
Input ← G_{proj} - Projection Graph
Output → I - Set of internal links

```
users = Gmain.users
businesses = Gmain.businesses
I = {}
for u in users:
    for b in businesses:
        if (u, b) not in Gmain.edges():
            internal = True
            for review in Gmain.edges(b):
                v = review.user
                if not Gproj.has_edge(u, v):
                    internal= False
                    break
            if internal:
                I.add((u, b))
return I
```

Once we have created a weighted projection graph, we want to identify internal links by iterating through $Users$ and $Businesses$ and identifying $(User, Business) \notin Reviews$ that induce edges which already exist within G_{proj}. Recall, this is the definition of an internal link.

C. Algorithm 3: Predicting Internal Links

Input ← G_{main}- Bipartite Graph
Input ← G_{proj} - Projection Graph
Input ← I - Internal Edges within G_{main} Output → P - Set of predicted links

```
P = {}
for i in I:
    u = i.user
    b = i.business
    for review in Gmain.edges(b):
        v = review.user
        if u != v:
            weight = Gproj.getWeight(u, v)
            if weight >= tau:
                P.add(i)
                break
return P
```

Once we have all internal links, we predict them based on the weight of their induced edges satisfying a certain threshold. More formally, we only keep internal links (u, b) where $u \in Users$ and $b \in Businesses$ where $Induced_{\tau}(u, b)$ is true, where $Induced_{\tau}(u, b)$ returns true if any link induced from (u, b) has a weight of τ or greater within some weighted version G_{proj}. Hence only if the there is an induced edge with weight more than τ do we predict that internal link.

Our algorithm now gives us a list of predicted links for each of the seven weighting methods which can be used to predict the ratings on new reviews.

D. Algorithm 4: Rating Prediction

Input ← G_{main}- Bipartite Graph
Input ← G_{proj} - Projection Graph
Input ← $(u, b) \in P$ - Predicted Internal Link
Output → uB - Predicted rating for (u, b)

```
uReviewed = Gmain.neighbors(u)
ratings = []
for v in Gproj.neighbors(u):
    if (v,b) in Gmain.edges():
        vReviewed = Gmain.neighbors(v)
        toCompare = uReviewed & vReviewed
        totalDiff = 0.0
        for b′ in toCompare:
            uRating = Gmain.weight(u, b′)
            vRating = Gmain.weight(v, b′)
            totalDiff += uRating - vRating
        avgDiff = totalDiff/len(toCompare)
        vB = Gmain.weight(v, b)
        ratings.append(vB + avgDiff)
uB = round(avg(ratings))
uB = min(MAX_RATING, uB)
uB = max(MIN_RATING, uB)
return uB
```

Building off of the internal links structure already established for link prediction, we saw the opportunity to predict ratings for these links as well. Once we have a set of predicted links we estimate the rating of the new edge by using the ratings of neighbors within the projection graph. Since internal links are guaranteed to not change the projection graph, we are guaranteed that the user shares a review with at least one other user, and that the other user has reviewed the business in the internal link. More formally, for an internal link $(u, b) \in P \rightarrow \exists v$ such that $(u, v) \in G_{proj}$, $(v, b) \in G_{main}$ and $\exists b'$ such that $(u, b') \in G_{main}$ and $(v, b') \in G_{main}$. This guarantee of the internal link algorithm makes extension for rating prediction always possible. First, the algorithm identifies the users within the projection graph that are neighbors of the predicted user and also reviewed the predicted business. For each of these users, we average the difference in ratings compared to the predicted user. The algorithm adds this average difference to the rating of the neighboring user to the predicted business, and then averages this across all relevant users. Finally, we round to the nearest integer and make sure the prediction is within the relevant range of possible ratings to give us a predicted rating on a 1-5 scale.

VII. Evaluation

In order to evaluate our algorithm, we take our university dataset (Stanford network) and perform a 50:50 split based on date. That is, we create a $Train$ set on the first half of the reviews, and a $Test$ set on the second half of the reviews. Our bipartite graph for the algorithm is then created based on the $Train$ set, which we use to predict new reviews. We then evaluate our predictions based on new reviews in the $Test$ set, between users and businesses existing in the $Train$ set and the $Test$ set. We consider the links T relevant, where a link $t = (u, b) \in T$ has the following characteristics:

- $t \in Test$
- $u \in Train$ and $u \in Test$
- $b \in Train$ and $b \in Test$
- t is designated as an internal link based on the structure of the network derived from $Train$

In order to understand the effectiveness of the internal links method on the Yelp Academic Dataset, we utilize the statistics of precision and recall. Precision is the percent of predicted links that are relevant. Recall is defined as the percent of relevant links that are predicted. We compute the F1 score, by definition, as a function of precision and recall.

$$F1 = 2 * \frac{Precision \cdot Recall}{Precision + Recall}$$

From here we can formalize the notions of precision and recall. Let us call the set of predicted links P. Using the same definition of T from above, we have:

$$Precision = \frac{|T \cap P|}{|P|}$$

$$Recall = \frac{|T \cap P|}{|T|}$$

Notice that this definition of T means if we were to predict every internal link derived within $Train$ then we would have a recall of 1.0. This is why we consider both the precision and recall when evaluating our method, as either one alone gives us an incomplete description of performance.

Our definition of relevance, or T, is carefully constructed for this algorithm. This is due to the potentially severe limitations of the algorithm. When using this algorithm to predict links in a network, one must recognize that the best-case scenario is that every link created after training is an internal link. This is simply not reality; Allali, Magnien and Latapy claim that internal links represent approximately 25% of newly created links 50 days past training. We found that the numbers within the Yelp Academic Dataset were much lower, as can be seen in the analysis section.

In order to evaluate the predicted rating on the new reviews we get from the internal links algorithm, we took the difference $abs(p - a)$ of the predicted rating p and the actual rating a and created a distribution of the differences. This gives us an empirical probability for how different p will be from a.

VIII. Experiment And Results

A. Link Prediction

We first trained τ, our parameter for link prediction, by running this algorithm on five different schools. We chose different schools with similar sizes to Stanford, essentially as a cross-validation set to give us different τ values that maximize our F1 statistic. The schools we chose to train τ on were Carnegie Mellon University (CMU), Cornell University (COR), University of Maryland - College Park (UMC), University of Illinois - Urbana-Champaign (UIC), and Princeton University (PRC). As defined in our evaluation, we split the data 50:50 by date for each of these schools, meaning we had the chronologically older half of the network to train on, and the newer half to evaluate against.

Since the algorithm uses seven different weighted projection graphs, we established seven different τ values, one for each method. For each of the seven weight methods, we evaluated our algorithm while increasing τ from 0.1 to 5.0 with increments of 0.1. We recorded the precision, recall, and calcuated the F1 score for each weighting function for each τ. We then designated the τ with the highest resultant F1 for each weighting function. This methodology gives us five sets (one for each validation school) of seven τ (one for each weighting method).

We ran our algorithm against the Stanford dataset, using a 50:50 chronological split once again. We used the respective τs acquired from our cross-validation runs. The results can be seen in Table 3. Unfortunately, the F1 scores, precision and recall of the Stanford sets were very low.

Given these results, we decided to tweak the $Train$ and $Test$ sets we used in our algorithm to try and improve performance. Prior work on the internal links algorithm had shown that the number of internal links would decrease as the time period of the $Test$ set was postponed further away from the $Train$ set. To combat this inverse relationship, we created two new experiments to test the effectiveness of the algorithm: year-by-year and month-by-month.

TABLE III. Stanford Network - Results using cross-validated τ

School	Weight	τ	F1	Precision	Recall
CMU	SW	4.1	0.0125	0.0063	0.3333
COR	SW	4.1	0.0125	0.0063	0.3333
PRC	SW	4.1	0.0125	0.0063	0.3333
UIC	SW	3.1	0.0063	0.0032	0.3333
UMC	SW	1.1	0.0012	0.0006	0.6666
CMU	JW	0.1	0.0004	0.0002	1.0000
COR	JW	0.2	0.0000	0.0000	0.0000
PRC	JW	0.1	0.0004	0.0002	1.0000
UIC	JW	0.1	0.0004	0.0002	1.0000
UMC	JW	0.1	0.0004	0.0002	1.0000
CMU	DW	0.2	0.0000	0.0000	0.0000
COR	DW	0.8	0.0000	0.0000	0.0000
PRC	DW	0.4	0.0000	0.0000	0.0000
UIC	DW	0.4	0.0000	0.0000	0.0000
UMC	DW	0.4	0.0000	0.0000	0.0000
CMU	RSW	4.1	0.0161	0.0082	0.3333
COR	RSW	4.1	0.0161	0.0082	0.3333
PRC	RSW	4.1	0.0161	0.0082	0.3333
UIC	RSW	4.1	0.0161	0.0082	0.3333
UMC	RSW	4.1	0.0161	0.0082	0.3333
CMU	RJW	0.5	0.0000	0.0000	0.0000
COR	RJW	1.0	0.0000	0.0000	0.0000
PRC	RJW	0.4	0.0000	0.0000	0.0000
UIC	RJW	0.9	0.0000	0.0000	0.0000
UMC	RJW	0.5	0.0000	0.0000	0.0000
CMU	RDW	3.1	0.0000	0.0000	0.0000
COR	RDW	4.6	0.0000	0.0000	0.0000
PRC	RDW	2.1	0.0000	0.0000	0.0000
UIC	RDW	0.1	0.0256	0.0133	0.3333
UMC	RDW	0.1	0.0256	0.0133	0.3333
CMU	RLSW	2.1	0.0077	0.0038	0.6666
COR	RLSW	4.1	0.0259	0.0135	0.3333
PRC	RLSW	4.1	0.0259	0.0135	0.3333
UIC	RLSW	2.1	0.0077	0.0038	0.6666
UMC	RLSW	4.1	0.0259	0.0135	0.3333

For these two new experiments we split the data by the time interval (either month or year). For example, in the month-by-month experiment we made separate sets of data for each calendar month. For each time-interval dataset, we ran our algorithm and predicted links, then evaluated based upon the next time-interval dataset. For example, in the month-by-month we would predict links (and evaluate success) for December 2010 based on the algorithm trained upon data from exclusively November 2010. For our τ in this experiment we decided that instead of attempting some form of cross-validation with other datasets, we would iterate through τ from 0.1 to 5.0 on increments of 0.1 as we had done before when finding the most effective threshold during validation. Our main aim was to try to find the highest possible F1 score, and therefore chose to find the most optimal τ for the Stanford dataset using this algorithm. This represents a fundamental change of direction in our experimentation, as these experiments no longer model the effectiveness of the algorithm in a real world scenario. Instead, these experiments are meant to display the capabilities of the algorithm in an ideal situation. Our intention for these experiments was to determine whether the internal links algorithm performed better at the granularity of a month or year, as we had already shown it to be ineffective on a 50:50 split over the entire time-span of data.

The results for the two new experiments can be seen in Table IV and V. The F1 score, precision and recall are still very low for the optimal τ parameters for each of the weight methods. This indicates that our original formulation of the algorithm as a 50:50 split across the entire time span was not incorrect, as the internal links algorithm performs ineffectively on the Yelp social network even when using a more granular time scale.

TABLE IV. Stanford Network - Year by Year split - Best τ by F1

Weight	τ	**F1**	**P**	**R**
SW	2.1	0.0344	0.0188	0.2000
JW	0.1	0.0018	0.0009	0.5200
DW	0.3	0.0355	0.0201	0.1600
RSW	4.1	0.0041	0.0021	0.6400
RJW	0.3	0.0023	0.0011	0.7200
RDW	0.6	0.0256	0.0134	0.2800
RLSW	1.1	0.0022	0.0011	1.0000

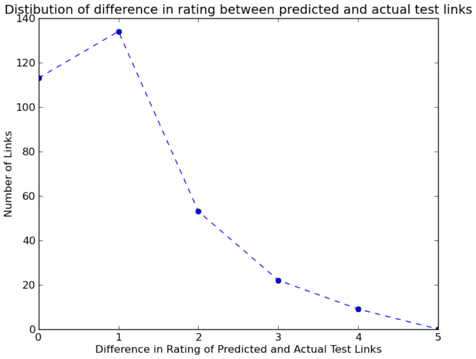
TABLE V. Stanford Network - Month by Month split - Best τ by F1

Weight	τ	**F1**	**P**	**R**
SW	0.1	0.0014	0.0007	1.0000
JW	0.5	0.0028	0.0014	0.5000
DW	0.1	0.0021	0.0011	0.3333
RSW	4.1	0.0021	0.0010	0.5000
RJW	1.9	0.0042	0.0021	0.5000
RDW	0.6	0.0025	0.0012	0.3333
RLSW	4.1	0.0015	0.0007	1.0000

B. *Rating Prediction*

After completing our experiments on link prediction, we used the internal links and projection graph to predict the rating of new reviews within the entire Yelp network. To do this, we made rating predictions for each review within T (See definition in Evaluation section) for each school. Obtaining the set T is equivalent to running the link prediciton algorithm on each school with a weight and threshold that would guarantee a recall of 1.0 (such as a Sum Weight where $\tau = 1$). Using the algorithm defined above we calculated a predicted rating for each of the links in T. We found that the internal links structure is very conducive to ratings prediction, as 247 of 331 (.748) reviews were within 1 point of the actual rating. Fig 5. shows the distribution of the difference in rating between predicted and actual.

Fig. 5. Distribution of difference between Predicted Rating and Actual Rating.



IX. Analysis of Results

Our experiments resulted in data that demonstrated the ineffectiveness of the internal links algorithm when predicting links in a bipartite network. Using cross-validation on other schools datasets to determine our threshold τ with 50:50 splits was largely unsuccessful. We attempted to change the granularity of the data split to year and month. Even with the empirically best possible τ we found that changing the granularity barely helped if at all. The internal links algorithm failed to predict links within the Stanford Yelp network with any acceptable degree of success. But more interesting than how the algorithm performed is why it behaved so.

A. *Low Number of Internal Links Found in the Test Set*

Possibly the largest contributing factor to this algorithm's poor performance on this dataset is the simple fact that $|T|$, the number of internal links within the test set, is very low. In fact, when the Stanford data was split evenly between test and train, $|T|$ = 3. Compare this to $|R|$ = 837 for the same data split, where R is the set of links $(u, b) \in Test$ where $u \in Train$ and $b \in Train$. Notice $T \subset R$. $\frac{|T|}{|R|}$ represents the upper bound of new links the internal links method could possibly predict over all new links capable of being predicted by any algorithm (since both the user and business were found within $Train$). For the Stanford dataset $\frac{|T|}{|R|}$ = 0.0036, an incredibly low percentage. This behavior was fairly common across multiple schools within the Yelp Academic Dataset, with one of the highest ratios found being from the University of Waterloo with a $|T|$ = 8 and $|R|$ = 58, for a $\frac{|T|}{|R|}$ of .1379. With $|T|$ values so low, the algorithm must predict a very small and very particular set of links in order to have reasonable precision, F1. The data shows, even with 7 different weighting functions, that higher thresholds only barely increased precision and instead often resulted in a drop in recall, meaning relevant links (links $\in T$) were no longer predicted. The low number of internal links within test sets undoubtedly harmed the performance of our algorithm immensely.

B. *High In-Degree of Businesses*

Another detractor from our algorithm's effectiveness was the preferential attachment personified in the Yelp network. The power-law business degree distributions reduced the effectiveness of creating projection graphs. The intuition behind a projection graph is that the structure created encapsulates some external similarity between the users: whether that be geography, personal interest, or personal taste. Unfortunately, the Yelp data used for this paper resulted in projection graphs that were not very reprsentative of user similarity due to businesses with very high in-degree. Businesses with high in-degree lead to tightly-woven projection graphs. For example, if a business has 50 unique reviewers then the resultant projection graph results in 2450 edges between all of the 50 reviewers. In fact, we found that the clustering coefficient of the Stanford projection graph on a 50:50 split to be .90, representative of this preferential attachment model. As a consequence of this behavior, more internal links can be identified since the projection graph is so well-connected that new reviews have a low chance of inducing links that change the projection graph. As such, these projection graphs give very little intuition into the similarity of the users. One point worth noting is that the Delta Weight weighting function is constructed to combat this phenomenon by normalizing by the number of reviewers for the businesses, yet found not much better success in terms of precison or F1 than other weight functions.

C. Low Out-Degree of Users

One additional hypothesis is that the frequency of reviews also posed a problem for the internal links algorithm. On average, users in the Stanford dataset had 150 days between any two reviews. Additionally, the average number of reviews per user was less than 2 (1.6). We believe that such low frequency and degree in terms of link creation harmed the algorithm's ability to reason about users, since the projection graphs were often created from one review per user and never supplemented.

X. Future Work

The internal links algorithm for bipartite networks can be improved to increase its effectiveness on social networks. One possible experiment would be to develop a weighting method that helps to address the problem of high in-degree businesses more adequately than the current Delta function. Another interesting avenue for exploration would be to create and weight a projection graph of businesses instead of users. This would make the projection graph more sparse, causing a lesser number of internal links to be possible as inducing changes in the projection graph would be easier. Additionally, this internal links algorithm should be evaluated on other bipartite networks where the number of links per user is higher and link creation is more frequent, such as a check-in network like Foursquare.

XI. Conclusion

We implemented the internal link algorithm for link prediction on bipartite graphs and experimented it on the Yelp dataset. Our conclusion is that this algorithm is not particularly suited to link prediction in social networks that are highly clustered and have a large temporal spacing. We do believe that the internal links algorithm can serve as a useful tool for recommendations within a bipartite network as internal links within this algorithm already have weights associated with them via the weighted induced links in the projection graph. Using these weights could be an approach to rank internal links and make recommendations.

Additionally, the experiments ran in this paper do indicate that the internal links algorithm does provide a foundation from which ratings prediction can be accurately determined. Although the scope of this ratings prediction may be limited due to the small number of internal links, our experiments show its effectiveness on the Yelp dataset. Businesses in the Yelp social network can leverage this algorithm to advertise to targeted users with higher predicted ratings in order to increase their probability of gaining positive reviews.

Acknowledgment

We would like to thank our advisors Christina Brandt and Justin Cheng for their valuable support and encouragement.

References

[1] D. Liben-Nowell and J. Kleinberg, *The link-prediction problem for social networks*, Journal of the American Society for Information Science and Technology, 58(7), pp1019-1031, 2007.

[2] O. Allali, C. Magnien and M. Latapy, *Internal link prediction: a new approach for predicting links in bipartite graphs*, Social Network Analysis and Mining, 3(1), pp 85-91, 2013.

[3] J. Leskovec, D. Huttenlocher and J. Kleinberg, *Predicting Positive and Negative Links in Online Social Networks*, In Proc, WWW, 2010.

[4] D. Liben-Nowell and J. Kleinberg, *The link prediction problem for social networks*, CIKM '03, 2003.

[5] L. Adamic and E. Adar, *Friends and neighbors on the web*, Social Networks, 2003.