Link Prediction in Bipartite Graphs

Anirudh Narasimhamurthy
u0941400
University Of Utah

Soumya Smruti Mishra
u0926085
University Of Utah

Abstract

One of the challenges in network analysis is to predict what new connections will be created in the particular graph/network at some point in future. Many real world networks like user-products or user-business relations have a bipartite nature to them and evolve during time. Predicting links that will appear in them is one of the main approaches to understanding them. This was our main point of interest and we decided to understand link prediction problem in bipartite networks in our project. We also see how this can be extended or formulated to be applied to a recommendation problem.

Classification Keywords

link prediction, collaborative filtering, recommendation, bipartite graph,

1 Motivation

The primary motivation for selecting this project was based on two reasons:

1. Firstly we wanted to explore and learn about link prediction and bipartite graphs. From our literature survey and reading papers [2] [6] [7] [8] we realized that link prediction in bipartite graphs was something which was less explored and hence we were interested to learn more about it through our project.

2. Secondly we were also trying to see if we could use any of the Machine Learning techniques which we had learned about in this project and we believed the problem which we chose would probably help us in achieving it.

2 Introduction

Link prediction is an important research problem in dynamic network analysis. Most of the problems which can be modeled as networks today are dynamic in nature i.e they evolve with node and link additions and removals. Typical examples include user - product relations where users are linked to the products they bought.

[9] states that social networks are not static and keep growing with time with addition of new nodes and links. Similarly user-product networks also evolve with time. This deals with link formation in the graphs i.e given nodes in a network the network grows by forming new relationships with existing nodes. We in this project deal with the question of predicting if a link will be formed between user and product/business.

Being able to predict such links accurately can have important implications in different networks including uses in national defense, predicting future credit card frauds and other related problems.Also a lot of the current research from what we understood focuses on link prediction on graphs with one type of node such as predicting links in social networks(Twitter, Facebook, LinkedIn) which is made up of people only. We wanted to explore

methods which could be applied for a bipartite graph setting.

In this project we attempted to use our understanding of the concepts we learned in this course in line with different methods we came across the literature and used the following methods for link prediction in bipartite graph:

1. Similarity measures using Jaccard similarity and common neighbors
2. Approach based on completing the matrix entries using decomposition techniques
3. A page rank style approach using Random walks.
4. Supervised binary classification method.

3 Formal Problem Description

Our project attempts to predict new links which can potentially be formed in a network in future, given a snapshot of the current network and to evaluate its effectiveness. We are also primarily involved in applying the different metric to bipartite graphs.

4 Definitions

Bipartite graph: A bipartite graph or bigraph is a set of graph vertices decomposed into two disjoint sets such that no two graph vertices within the same set are adjacent to each other.

5 Assumptions

1. One of the primary assumptions that we make is although collaborative filtering recommendation technique might work with the data set, our interest was to learn about link prediction in bipartite graphs and so we model our data set as bipartite graph.

2. Link prediction is our primary goal and recommendation is a side product of this one and we assume the recommendation or result which we state in our later sections might be a possible result although the sparsity and other factors might point otherwise.

3. Also for our data set without taking the location into consideration, we assume the prediction that we make is reasonable one although in practice there could be a lot of other things needed to make such a prediction.

6 Scope

Our main stating is that the approaches used in our project could potentially be applied to a different data set which exists as or can be modeled as bipartite graph and even if the example data set which we have taken might not sufficiently convince that, we wanted to make it clear that our scope lies in the understanding of methods/measures which could be applied for link prediction in bipartite graphs. Results and other observations stated in later sections are our understanding but the bigger picture could or could not be different.

7 Prior Work

- A lot of work has been done on link prediction in general.In [4] they experiment with several similarity metrics including Graph Distance, Preferential attachment, Katz method and others. These could be modified so that they can be applied to bipartite graphs. In our project we experiment with two similarity metrics stated in Introduction section.

- Link prediction in bipartite networks has not been explored extensively. In [5] they describe how similarity metrics could be used as features in bipartite setting and they describe machine learning approaches to solving this problem. Supervised link prediction methods were also suggested.

- Other works in bipartite link prediction include [2, 3]. In [2] they use a kernel based approach to link prediction while [3] uses a different distance metric approach.

- One of the co-authors of our prescribed text book Jure Leskovec has also done research in this area and in [1] they propose a random walk

approach to link prediction which makes use of both a distance metric and Page Rank style approach.

- Our approaches used in our project are primarily influenced by the above mentioned works.

8 Data

8.1 Dataset used for the Project

We used the TripAdvisor dataset for our project. It was obtained from Database and Information Systems Laboratory of University of Illinois, Urbana Champaign from the following url `http://times.cs.uiuc.edu/~wang296/Data/`

8.2 Dataset description

The raw dataset consisted of set of reviews written by different users for different hotels along with the ratings given by the user to different aspects of the hotel. Every review also had a date associated with it. The date range for the reviews in the dataset was from 2002 till 2012. The size of the dataset was 2.06 GB. A snapshot of a the raw review data is shown below for better understanding:


```
{"Reviews": [{"Ratings": {"Service": "5", "Cleanliness": "5", "Overall": "5.0", "Value":
"5", "Sleep Quality": "5", "Rooms": "5", "Location": "5"}, "AuthorLocation": "Albany,
Oregon", "Title": "\u201cSeattle\u201d", "Author": "Sharon B", "ReviewID": "UR126815874",
"Content": "This past fall I went with a friend to Seattle. We stayed at the Best Western
Loyal Inn and we enjoyed our stay. The front desk was very helpful and when you call with
a request there was follow through. The room was clean and fresh smelling. The breakfast
was good too. I would return again. Walking distance to the Space Needle and sights close
by.", "Date": "March 28, 2012"},
```


Figure 1: Raw review data

8.3 Data processing

The most important aspect of our problem is to model the given dataset as a bipartite graph. From the definition mentioned earlier, to model the given data as a bipartite graph, the following processing was required to be done:

- In order to lessen the load on our processing we removed the review text from all the files and then created a cleaned up reviews.json. We adopted this approach with the assumption that we would not be making use of review content to do a prediction. The cleaned up reviews file had the following fields
  1. Ratings
  2. Author name or User name
  3. Author Location
  4. Date of Review
  5. HotelID
  6. Hotel Location
- Using the reviews.json created in the above step, we then created two more files one for users/authors and the other for hotels. The users.json had the following fields:
  1. AuthorID (a unique ID assigned by us)
  2. ReviewCount (aggregation of all the reviews written by this author identified by Name/ID)
  3. Last Review Date(date of the last review written by this author/user)
  4. UserLocation
- The hotels.json file on the other hand had the following fields:
  1. HotelID
  2. ReviewCount (aggregation of all the reviews written for this Hotel identified by ID)
  3. HotelLocation
- At the end of this processing, the count of hotels and users are given below:

Hotels count	11797
Users count	781237

Table 1: Hotels and users count

- These two files users.json and hotels.json would then help us model the bipartite graph as we would have all the users on one side of the graph and we would have all the hotels on the other side of the graph and an edge goes between user and hotel if a user has written a review for that hotel.

- As stated earlier that prediction will be done based on the snapshots of the network or graph created at different times. We used t(training) as Feb 15, 2011 meaning all data before it and t'(test) as Feb 15, 2012 meaning all data between t and t' to create train and test data respectively. To get the required graph in terms of nodes and edges,we had to do further processing.

- From the hotels and users file, we picked out the IDs and then assigned a unique ID for each of the HotelID and UserID which is a running sequence so that there is better clarity when we look at the graph data file. Once IDs were created we then created the graph.txt file for both test and train data which will contain all the edges in the current dataset represented by nodes on either side.

- Once we created graph.txt to get statistics about the graph, we used some of the built in methods from SNAP.py package and also referred to code blocks online to get the following details:

Property	**Value**
Number of nodes	793034
Number of edges	1088791
IsConnected	No
Average Degree of a node	1.372944

Table 2: Statistics on the graph.txt

8.4 Creating matrix from input data

We also wanted to experiment with supervised learning methods for link prediction and hence we had to create a matrix file so that it could be used by our classifier to predict the output. During the creation of matrix, we did a further filtering to ensure data is not overly skewed and it is at an acceptable level. This idea was obtained from one of the projects of the Stanford Data Mining Class handled by Jure Leskovec. The following filtering was done:

1. Only taking users who have been active in the last six months. To do this we made use of the Last Review Date field which we had included while creating users.json.

2. Only taking hotels which are at a distance 3 or 3 hop distance from current user as candidate edges. To compute the hop distance we made use of methods in SNAP.py package

For creating matrix we again made use of graph.txt as the input and also took data snapshots at two different times t and t' to create train and test matrix.

The way users review hotels evolves over time and so test and training may have huge differences but we assume that those differences would be small enough that supervised learning would still be effective. We try to see if that is the case. Furthermore we felt it was justified in the sense even in real world scenario this is how data would probably end up being because we would have information only about the previous links.

9 Similarity Measures for Link Prediction in Bipartite Graphs

- Similarity metrics can be used to compute the score of each candidate edge or potential links which might appear in future.

- Metrics used in [4] compare the neighbors of two nodes in the edge. But to make it for bipartite graphs we have to modify it. Because of the property of bipartite graphs nodes on the same side or neighbors of nodes on opposite sides do not intersect. Hence we would want to model them in such a way that we have both sets of nodes to contain the same type of node i.e users/ hotels.

- To do this, we choose two sets S(x) and S(y) for our similarity metrics given nodes(x,y) on opposite sides of a graph. We define S(x) and S(y) in a way which is similar to how we used hop distances for creating matrix.

  - S(x) represents nodes which are 2 hops away from x. Effectively this produces nodes on same side of the network as x because traveling two hops would be

equivalent to going to x's neighbor and then coming back again to the same side of x.

  - S(y) represents y's neighbors. These represent nodes on opposite side of y, which is nothing but same side of x, because we have modeled the graph as bipartite in this case.

- To compute 2 hop distance we used SNAP package's GetNodesAtHop() method to get our required values. Also SNAP modules other methods for graph processing proved useful. We found this package to be useful when we were stumbling across different data sets in our initial search for project data sets.

- Both the sets from the description above will comprise of node's on the side of x. This enables us to use them to compute distance metrics. We used two such metrics in our project:

9.1 Jaccard Similarity

Jaccard Similarity is one of the widely and most commonly used similarity metrics. Jaccard similarity for our bipartite case is defined by:
$\frac{|S(x) \cap S(y)|}{|S(x) \cup S(y)|}$

Jaccard similarity has the advantage that it is relatively easier to implement and also it is computationally less expensive than say cosine similarity for the same graph.

9.2 Common Neighbors

Common neighbors is a metric which is commonly used in graph networks where similarity could be measured in terms of the neighbors. For the bipartite case, it is simply defined as :
$|S(x) \cap S(y)|$

9.3 Note

- For the similarity measures we tried implementing using both the nodes as S(x) in the sense, we had S(x) as users and then calculated all the hotel's neighbors and we also considered S(x) as hotels and then calculated all user's neighbors for S(y)

9.4 Intuition behind similarity metrics and link prediction

- Computing similarity metrics over these set of nodes is effective for link prediction because S(x) returns nodes which are similar to x since they are 2 hops away and on same side of x.

- In other words users who review hotels that you do are likely to be similar to you. It is also reasonable to further assume if lot of them review the same hotel and there is a large overlap between them and s(x) then you are also likely to review that hotel in future. This in a way helps us to uncover the new links which might appear in the graph in future which is our primary motive.

- As stated in the first, this is an assumption which we make and think is reasonable although it has been pointed out that this isn't so straightforward. But the bigger picture from this is, we learned how to modify existing similarity metrics to be applied to a bipartite case, which was our primary goals.

10 Random walk approach to Link Prediction

In this section we describe how random walks can be used for link prediction in bipartite graphs.

10.1 Definition

A random walk is a finite Markov chain that moves through a graph G=(V,E)

10.2 Ideas used from the class/course material

- Suppose each edge/link in the given bipartite graph is assigned a weight w(u,v). Then based on the random surfer which we used for PageRank implementation, each edge (u,v) can also be assigned a transition probability $M_{u,v} = w(u,v)/d(u)$ where d(u)= $\sum_{(u,v)\varepsilon E} w(u,v)$ is the

weighted degree of u or the number of outlinks from u.

- These transition probabilities can be used in random traversal of the graph and when we run this process for a long time, it would eventually point to which nodes we might tend to visit. This in effect would tell us which hotel a user would likely review because an edge/link would be created based on the random walk's completion.

- Convergence in this case would be slightly different from page rank case and also the transition probabilities at a stationary state would be given by $p_s = p_s M$

10.3 Description and Implementation

- We used random walks to produce a score for each candidate edge (u,v). We scored the edge with the stationary distribution probability of reaching v when running a random walk from u.

- To assign the weight to edges stated in the previous subsection, we found there were supervised methods of learning the weights and then applying them or using heuristics to decide the weight.

- We preferred the approach which was simpler to us which was using heuristically chosen edge weights. In this case higher weights should be assigned to links which are likely to be more relevant. Relevance in our case would be the fact that the reviews written recently would tend to be a good reflection than a review which was written several years ago (for most cases and it is an assumption we make). This goes back to the point that as years go by people evolve and hotels evolve too and so the recently written reviews are likely to give a more accurate picture.

- To take that into consideration we give each edge a weight of w_{uv} =1/(c+a_{uv}) where $a_{uv}$is the age or the difference between current date and date of last review written by the user and c is a constant.

- We made use of methods available in networkx package in python for constructing adjacency matrix and scipy and numpy packages for other processing.

- We used sc.parallelize() to create RDDs and used simple map reduce operations similar to our block matrix vector multiplications to calculate the probabilities and wrote the results to a json file.

11 Low rank approximation technique to Link Prediction

- As stated earlier in the context of a adjacency matrix, link prediction can be viewed as predicting new entries in the matrix of the graph. Again there might be issues with this approach given sparsity of the graph and the fact that absence of an entry in the matrix might not necessarily be interpreted as 0 in the boolean case.

- We weren't entirely sure of how the results would turn out but wanted to see what would be the results given this generic assumption and hence proceeded with it.

- A commonly used technique for analyzing large matrices is to compute a low rank approximation for the given matrix. Different techniques are available for decomposition and we tried two of it:

11.1 Singular Value Decomposition

We tried to implement SVD for our given graph and then take the diagonal values to predict the potential links. The feedback given to us was to first try and do a SVD in say MATLAB and plot the diagnol entries of S. We tried that, but the size of the graph was so huge (1169456 x2) that a full adjacency matrix would exceed the maximum array size preference in MATLAB.

To get an idea of the plot, we took the first 30000 entries and loaded into MATLAB and computed SVD just to get a rough idea of the plot. This might not be the correct approach but we wanted to make

sure we did try the methods suggested in the feedback. The plot of diagonal of the S matrix is shown below:

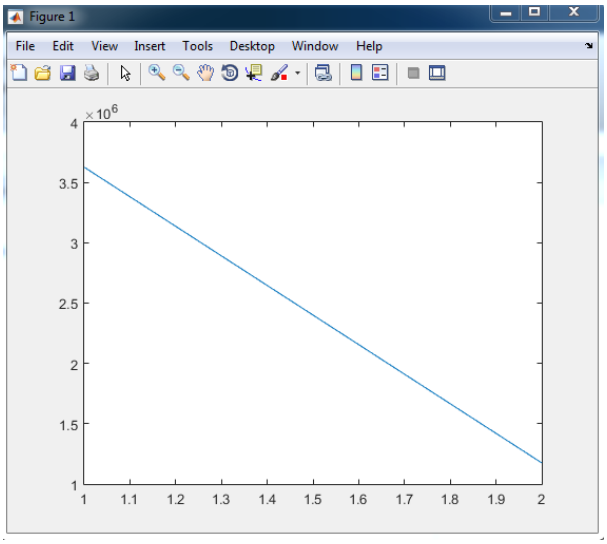

Figure 2: Plot of diagonal of S matrix(30,000 entries)

- The plot doesn't have the typical curve like the one which we have for collaborative filtering and hence might not be a good method to apply. Our data in graph.txt is in such a way that it probably makes this approach not suitable for this data.
- Most of the diagonal entries were zero. This might either be due to the bipartite nature of the graph since edges can't go between nodes on the same side of the graph or it could be that this approach is not correct for this dataset. We weren't entirely sure and from the feedback that we got, we felt this was probably not the correct approach for this dataset.
- But one of the reasons why we wanted to explore this approach was the fact for bipartite graphs, the adjacency matrices is symmetric and it would help with the processing where instead of going for the full adjacency matrix we could have used the sparse matrix for SVD computation.
- We even tried experimenting with SVD in python using linalg package with the above approach and setting a value for k but we don't publish them in final results.
- We also tried using the code for UV decomposition using Stochastic Gradient Descent available on the course website, but we weren't able to comprehend and didn't have much clarity with that.

12 Supervised Binary Classifier approach to Link Prediction

- Our interest in Machine Learning wanted us to explore some Machine Learning technique which could be applied to our problem and from several works in [1, 5, 7] we got an understanding of how binary classification could be applied to link prediction.
- We had initially proposed to experiment with the xgboost algorithm as we felt it could potentially take advantage of parallel processing but its adaptation wasn't easier.

12.1 Gradient Boosting Classifier

- Instead we went for a slightly easier approach of using Gradient Boosting Trees (GBT). We chose this one, as considerable work had been done using SVM and Logistic Regression in this area. Also we were familiar with Decision Trees and Ensembles from our Machine Learning class and also had implemented variants of them for that course.
- GBT is essentially an ensemble technique which uses smaller decision trees. The classification problem for our case was given two nodes will the classifier predict a yes or no on whether a link will occur between the nodes in future.
- Other primary reason why we explored with different methods was the fact that, those methods or measures could be used as features for supervised learning.
- In addition to using those measures as features we also tried adding other features in terms of

degree of user, degree of hotels and the rating because we felt we had less number of features. But they did not change the results significantly.

- We used scikit package's methods to help out with some of the implementation.

12.2 Random Forest Classifier

- Out of curiosity we also tried some more libraries from the scikit package and once such classifier we used was Random Forest Classifier.

- Random forests is a notion of the general technique of random decision forests that are an ensemble learning method for classification, regression and other tasks.

- Random Decision Forest is very effective in eliminating noise in the model input data. Given a long list of input variables and a potentially sparse data set, it is very likely that any predictive model will discover spurious relationships between those inputs and the chosen target variable. This results in overfitting and the model does not generalize well enough to future input it has not seen.

  Because Random Forest builds many trees using a subset of the available input variables and their values, it inherently contains some underlying decision trees that omit the noise generating variable/feature(s). In the end, when it is time to generate a prediction a vote among all the underlying trees takes place and the majority prediction value wins.

13 Results and Evaluation

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

  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.

  We used the scikit metrics module's *roc_auc_score* measure to calculate this accuracy.

- Overall, Gradient Boosting classifier and Random Forest Classifier proved to be the most effective technique.

- Common-neighbor similarity performed better than Jaccard as expected. Among both user and hotel, the user-centric one proved superior, perhaps because most users have reviewed a very small number of hotels, so looking at their neighbors provides little information compared to looking at nodes at distance 2 from a user.

- With all the techniques, having a good accuracy 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.

Property	**Accuracy**
Jaccard for Users	0.439
Jaccard for Hotels	0.411
Common-Neighbors for Users	0.461
Common-Neighbors for Hotels	0.487
Random Walks	0.563
Gradient Boosting	0.613
Random Forest Classifier	0.598

Table 3: Results

14 Conclusion

- For this project we took bipartite graphs and tried to implement a variety of link prediction techniques to them. We found the commonly used link prediction similarity metrics could also be effective on bipartite networks if they were modified to be applied over a larger set of close nodes than just neighbors.

- The supervised classifiers were by far the best method in terms of accuracy, meaning it is the strongest at predicting overall what new edges will be in the network in the future.

- We had issues with trying out low rank approximation techniques like SVD or UV decomposition for the given data set and hence have not included it in the results table.
- One more thing we could have tried is the supervised random walks, which wasn't looked at due to lack of bandwidth and as found out by Lars Backstrom , Jure Leskovec in [1] it is also a very good technique in link prediction.

15 Difficulties faced and Lessons learned

1. One major takeaway from this project was the fact a good data set or a simpler problem to solve using different techniques is definitely more practical and helpful than having a difficult problem at hand. But at the same time having to work with this data set which we chose taught us a few things
2. We still believe that the results may not be a great reflection for this data set of ours but as we had mentioned in the beginning these are potential techniques/methods which could be applied to any problem or network which can be modeled as bipartite graphs and it would most probably work well for most cases
3. Selection of data sets play a very important role and is equally important as selecting the problem to tackle.
4. We faced difficulties particularly with Decomposition techniques and its interpretation of results for this data set.
5. Jaccard similarity measures provides a good baseline measure.
6. Although we did not do a lot of supervised learning methods, it points towards the direction that it can do better for the case of link prediction provided we have all the required inputs.
7. Running the code takes a lot of time especially the similarity metrics for a large graph. There are ways in which it could be optimized and they could provide slightly better running times.

16 Future Work

- One definite extension would be to perform supervised random walks where edge weights are learned by unsupervised methods and then given as inputs to the supervised methods. That could potentially improve the results.
- SVD did not work correctly but we still feel matrix sketching approaches would be able to do a good job and if we had a good data set or more regularized matrix we could potentially see the utility of low rank approximation methods.
- Using a better data set to perform our calculations.

References

[1] Lars Backstrom , Jure Leskovec, Supervised random walks: predicting and recommending links in social networks, Proceedings of the fourth ACM international conference on Web search and data mining, February 09-12, 2011, Hong Kong, China.

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

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

[4] Liben-Nowell, David and Kleinberg, Jon, The link-prediction problem for social networks,Journal of the American Society for Information Science and Technology,http://dx.doi.org/10.1002/asi.20591,2007.

[5] 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

[6] O. Allali, C. Magnien and M. Latapy , ”Link prediction in bipartite graphs using internal links and weighted projection” , NetSciCom , pp.953 -958

[7] Xin Li , Hsinchun Chen, Recommendation as link prediction: a graph kernel-based machine learning approach, Proceedings of the 9th ACM/IEEE-CS joint conference on Digital libraries, June 15-19, 2009, Austin, TX, USA

[8] Zan Huang , Xin Li , Hsinchun Chen, Link prediction approach to collaborative filtering, Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries, June 07-11, 2005, Denver, CO, USA

[9] https://www3.nd.edu/ dial/papers/ICDM12b.pdf

[10] 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.