A Two-Step Approach to Weighted Bipartite Link Recommendations

Nathan Ma
Adlai E. Stevenson High School Lincolnshire, Illinois, 60069, USA

***Abstract*—Many real-world person-person or person-product relationships can be modeled graphically. Specifically, bipartite graphs are especially useful when modeling scenarios involving two disjoint groups. As a result, existing papers have utilized bipartite graphs to address the classical link recommendation problem. Applying the principle of bipartite graphs, this research presents a modified approach to this problem which employs a two-step algorithm for making recommendations that accounts for the frequency and similarity between common edges. Implemented in Python, the new approach was tested using bipartite data from Epinions and Movielens data sources. The findings showed that it improved the baseline results, performing within an estimated error of 14 percent. This two-step algorithm produced promising findings, and can be refined to generate recommendations with even greater accuracy.**

***Keywords*—Bipartite graph; weighted graph; link prediction; two-step algorithm; information retrieval**

I. Introduction

Bipartite graphs [1] are graphs that can be split into two sets of vertices such that, within each set, there are no edges. Edges only exist between the vertices of opposite sets. See Fig. 1, for an example of a bipartite graph. Within a weighted bipartite graph, each edge is also given a weight. Weighted bipartite graphs [2] can be used to model a multitude of real-world relationships such as the product-customer relationship. In this scenario, the weighted edges represent how a customer has rated a product.

This research focused on examining the link prediction problem in the context of weighted bipartite graphs. Though not solely limited to weighted bipartite graphs, the link prediction problem [3] takes in an arbitrary graph network and predicts the possibility of an edge between unpaired vertices. When applied to weighted bipartite graphs, the link prediction problem predicts the weight between unpaired vertices based on the input graph network.

Weighted link prediction has broad applications. For example, it is sometimes analogous to forecasting how customers will rate products based on a network of customer-product ratings. Many platforms, such as Amazon or YouTube, likely employ some form of weighted link prediction algorithm when recommending items to users. The applicability of link prediction can be extended beyond user-item recommendations to image recommendation systems as well [4].

In this paper, we propose an algorithm that performs weighted link recommendation in two steps, which differs from existing multi-step algorithms that utilize techniques derived from the predominant Graph Convolution Networks [5] and Graph Neural Networks [6]. In our two-step approach, the confidence in a potential link occurring between two vertices based on common neighbors was first evaluated. Then, weights were determined based on the information derived from these common neighbors. This algorithm was finally tested on existing datasets that could be modeled using weighted bipartite graphs.

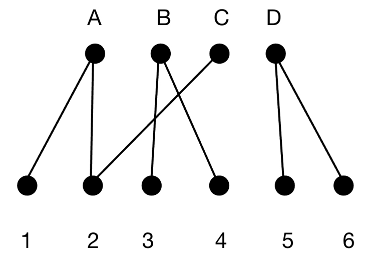

Fig. 1. Example of an unweighted bipartite graph

II. Related Work

Given numerous real-world applications, prior papers have examined different approaches to link prediction using either weighted [7] or unweighted [8] bipartite graphs.

One approach to unweighted link prediction is to specifically predict internal links [9]. Internal links are defined as an unpaired top and bottom node in which a hypothetical edge between the nodes does not change the bipartite projected graph. Only internal links may be considered for potential edges because being "internally linked" means that two vertices already share some degree of common neighbors. In turn, this indicates that the two vertices are more likely to have an edge develop between them. Whether an internal link would become an actual edge may be determined from several proposed weighting functions that account for factors such as common neighbors and the overall number of neighbors. (See Fig. 2, for an example of an internal link.)

One idea used in weighted link prediction is the Pearson coefficient [10]. The Pearson coefficient does not predict edges, but determines a similarity between two nodes within the same disjoint bipartite set. Similarity is determined by first finding the set of common neighbors between the two nodes. For each of the two nodes, the weight of the edge with the common neighbor is then subtracted from the average weight of all edges extending from that node. These resulting values are multiplied, and this process is repeated for all common neighbors. A positive final value corresponds to a positive similarity and vice versa.

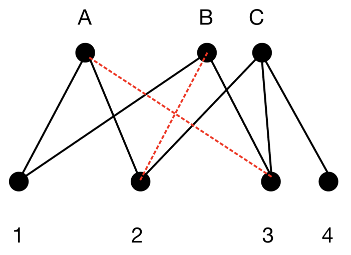

Fig. 2. Example of internal links within a bipartite graph (dotted red edges)

It is also important to note the difference between link prediction and link recommendation. Whereas link prediction involves finding any arbitrary connections that will occur in the future, link recommendation is more specific to the weighted bipartite graph. Instead of dealing with arbitrary connections, link recommendation identifies the most relevant connections between unpaired top and bottom nodes and calculates a weight for a hypothetical edge.

A classical approach specific to link recommendation has been collaborative filtering [11]. The principle underlying collaborative filtering is very similar to that of the Pearson coefficient; if Person A and B rate a product similarly, then Person A should rate another product more similarly to Person B than a randomly chosen person. Thus, collaborative filtering leverages the power of people with mutual interests to generate recommendations.

Collaborative filtering may be applied in many different ways such as: the use of neural networks [12] to establish deeper analysis between the user-user and user-item connections, hybrids between memory and matrix-based ideas [13], and the incorporation of time as a relevant factor in the decision making process [14] [15]. Though collaborative filtering is not the only approach to link recommendation [16], the role of similarity nevertheless plays a vital role.

Traditionally, key factors that affect recommendation algorithms include the size and density of the dataset [17]. This paper builds upon the previous work of both past link prediction and recommendation models to create an algorithm suited for recommendation.

III. Methodology

This recommendation algorithm is comprised of two steps. First, pairs of currently nonadjacent top and bottom nodes were identified for making a prediction. Second, predictions were made after selecting the pairs. Notably, a recommendation could not be made in the case of insufficient data connecting a top node and a bottom node.

Intuitively, the top nodes can be thought of as products and bottom nodes as customers. For ease of understanding, the terms "bottom node" or "customer" and "top node" or "product" are used interchangeably in this paper. Thus, the weights between the top and bottom nodes represent the ratings a customer has assigned to a product.

For each candidate pairing, let t denote the number of neighbors of the original top node, excluding the original bottom node. Next, for each of these neighbors, let n define the number of neighbors that have at least one common top neighbor with the original bottom node. Interpreting $\frac{n}{t}$, a value close to 1 indicates that most of the customers who already purchased the candidate product have purchased items the candidate customer has already purchased. The inverse is also true. Thus, a threshold value for $\frac{n}{t}$ was defined such that all values below were classified as "insufficient" for recommendation, and all values above were classified as "sufficient" for recommendation.

This threshold is expected to vary for every dataset. Generally speaking, the threshold should be lower in datasets with a sparser distribution of edges and greater in more compact datasets. Additionally, datasets with many edges may be "sparse" if they also have many nodes.

Thus, the threshold can be given by the equation $\frac{9}{10} - \frac{4}{x+y}$, where x represents the average amount of edges for a bottom node, and y represents the average amount of edges for a top node.

This threshold was set to have a maximum value of $\frac{9}{10}$. Thus, for larger values of $x+y$, the candidate bottom node must have a common neighbor with at least 90% of candidate-top adjacent bottom nodes. The term $-\frac{4}{x+y}$ serves to compensate for smaller datasets with fewer existing edges. The constant 4 was specifically chosen to accommodate smaller datasets such as the Epinions dataset. After performing some initial testing, the constant values 1, 2, and 3 all returned few or no recommendations because the resultant threshold was too high. The constant value 4 was the first number that consistently produced recommendations.

However, this does not prevent cases where the product and customer have sufficient similarity, but have insufficient data to ascertain that level of sufficiency. For example, if the candidate top node has only one bottom neighbor besides the candidate bottom node, and the candidate bottom node has a common top neighbor with the one candidate top-adjacent bottom node, then $\frac{n}{t} = 1$. This will always yield a value above the maximum threshold set; however, this cannot be definitively determined based on a single data point. Therefore, we also required that the original bottom node and bottom nodes adjacent to the original top node have a number of common neighbors at least equal to the average number of edges of a top node.

Once the unpaired nodes with insufficient data were filtered out, we predicted the expected weight between the candidate top node and the candidate bottom node. This was achieved by returning to the bottom nodes that were neighbors of the candidate top node. For the sake of clarity, these bottom nodes were labeled as candidate top adjacent. We defined the *similarity* of one of these bottom nodes and the candidate bottom node by how closely the common neighbors were rated. In other words, the *similarity* represented how closely the preferences of two customers matched based on the products both had bought. Naturally, a higher similarity means that two customers have similar preferences, which in turn, affects the prediction positively.

Similarity was determined by comparing all products rated by both customers. For each common product, the algorithm computed the difference between the ratings assigned by the

$$\begin{cases} (min(r_1, r_2) - a) \cdot (2 - |r_2 - r_1|) & r_1 \geq a, r_2 \geq a \\ (a - max(r_2, r_1)) \cdot (2 - |r_2 - r_1|) & r_1 \leq a, r_2 \leq a \\ 2 - |r_2 - r_1| & r_1 \geq a, r_2 \leq a \\ 2 - |r_2 - r_1| & r_1 \leq a, r_2 \geq a \end{cases}$$

Fig. 3. Piecewise function for temporary similarity

candidate bottom and the overall average rating assigned by all of its customers. The process was repeated for the other bottom node. The differences were compared, and a temporary similarity for each common top node between the candidate top adjacent bottom node and the candidate bottom node was calculated. This was based on the absolute difference and whether the signs were the same. The piecewise function in Fig. 3 depicts the exact method employed for calculating temporary similarity (r_1 and r_2 denote the two differences; a denotes the average rating of the common top node).

After iterating through each of the common products, the temporary similarity values were averaged to determine the overall similarity between the candidate bottom node and the candidate top-neighboring bottom node.

For each bottom node neighboring the candidate top node, the algorithm computed similarity with the candidate bottom node using the aforementioned algorithm. The predicted rating p was determined by the following formula:

k represents the number of top-adjacent bottom nodes that share at least one common neighbor with the candidate bottom node. s_i represents the similarity score for the ith product-adjacent bottom node. r_i is the analogue for rating.

$$p = \frac{\Sigma_{n=1}^{k} s_i \cdot r_i}{\Sigma_{n=1}^{k} s_i}$$

IV. Results

The algorithm was implemented in Python and tested using two datasets, which were obtained from the Epinions website [18] and the MovieLens website [19], respectively. Both datasets can be modeled using weighted bipartite graphs with weights ranging between 1 and 5. Users in the Epinions dataset rated products, while users in the Movielens dataset rated movies. In both datasets, a 1 represented an utterly unsatisfactory response whereas a 5 represented an extremely satisfactory response.

After reading in the dataset, 80% of the given edges were used to train the algorithm (which provided the averages used in later calculations for similarity) and the remaining 20% of the edges were set aside to test the algorithm.

The algorithm was tested by first running through 20% of edges initially set aside. The first part of the algorithm determined whether the edge had sufficient data to make a prediction. If the edge had sufficient data, the second part of the algorithm was implemented to obtain a predicted weight.

The predicted weight was measured against the actual weight and the percent error was determined. To determine

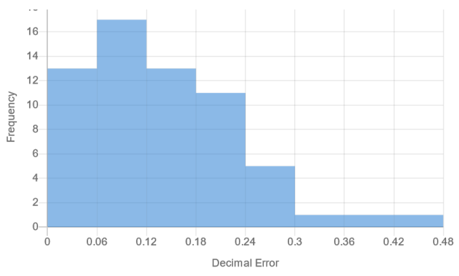

Fig. 4. Histogram of individual error of epinions recommendations

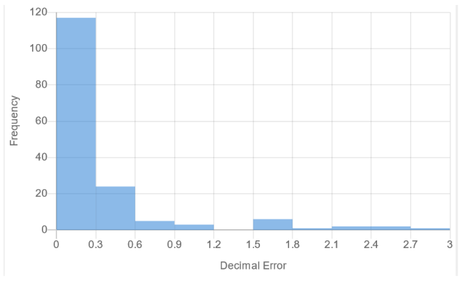

Fig. 5. Histogram of individual error of movielens recommendations

the overall percent error of the algorithm, the average of all individual percent error values for each prediction was used.

The performance of the algorithm across both datasets is detailed in Fig. 4 and 5 and summarized in Table 1.

TABLE I. Percent Error of Epinions and Movielens Dataset

Dataset	Percent Error
Epinions	13.8%
Movielens	14.4%

Comparing Fig. 4 and 5, we noted that the error in predictions from the Epinions dataset was approximately unimodal, centered between 6 and 18 percent. On the contrary, the error in predictions from the Movielens dataset was much more right-skewed with no clear center value exhibited. The overall percent error was therefore calculated using the mean value in the Epinions dataset and the median in the Movielens dataset.

Considering the two datasets, the overall percent error was approximately 14% which improved upon the 20% error of existing algorithms [20]. This attests to the strength of our two-step approach and encourages further exploration into similar link prediction algorithms. However, though this is a promising finding, it must be noted that each recommendation was “screened”; i.e., a recommendation was only made if sufficient data existed in the first place.

V. Discussion

The two-step algorithm performed fairly well; however, outlier recommendations were notable, particularly with respect to the Movielens dataset. Specifically, the Movielens recommendations generated error values up to 300%. Further analysis revealed that though both datasets employed a similar 1-5 rating system and modelled a type of product-customer relationship, the most pronounced difference was that the

Movielens dataset contained nearly 10 times the number of entries than the Epinions dataset. This size difference may have led to greater diversity in opinions, which in turn may necessitate a different approach to the parameters.

With respect to the size of datasets, this algorithm runs in $O(n^3)$ time. Therefore, testing the algorithm on datasets containing more than approximately $100,000$ entries is infeasible given the unrealistic run time required. A significant number of existing datasets contain millions of entries, which limits the scope of datasets available for testing. However, given the size of the Epinions and Movielens datasets, both datasets may provide solid representations of a real-world product-customer relationship. However, the high time complexity is ultimately a limitation compared to other methods which were able to operate on much larger datasets.

An alternative method for calculating $\frac{n}{t}$ may also be considered. In the current algorithm, n was defined as the number of candidate bottom nodes and top-adjacent bottom nodes that share at least one common top node. However, instead of a single common top node, the two bottom nodes may be required to share at least two common top nodes, three common top nodes, or even more common top nodes.

However, it must be noted that adjusting this parameter would likely necessitate changing the $\frac{n}{t}$ threshold. Requiring a greater number of common top nodes would decrease the value of n and warrant lowering the $\frac{n}{t}$ threshold correspondingly. This makes sense as while there might be fewer pairs of sufficient bottom nodes, there would ultimately be a relatively equivalent amount of data to work with as the number of common top nodes was increased.

Additionally, other conditions can be deliberated on in which the $\frac{n}{t}$ value may be deemed sufficient enough to generate a prediction. Examining the threshold, potential edges were sometimes rejected due to insufficient data. Although the $\frac{n}{t}$ value was high enough, the value t might have been too low to instill confidence in making a prediction. However, instead of rejecting those edges outright, the threshold might have been adjusted so that it increased as the number of top-adjacent nodes decreased.

Furthermore, different functions could be explored that determine similarity between the candidate bottom node and a top-adjacent bottom node. In a manner similar to how the Pearson coefficient was calculated, the piecewise function in Fig. 3 rewards the bottom nodes for rating their common top nodes similarly. It does this by taking into account the differences between how the two bottom nodes rated the common top nodes, and how they typically rated a top node. Matching signs (positive or negative) would contribute more positively towards similarity. One could also explore the effect of slightly modifying the constants of the functions.

Finally, this result continues to highlight the value of intuitive implementation within the context of this problem [21]. This follows a recent trend within the documentation of the link prediction problem of simplifying the problem [22]. Whereas previous papers used more involved methods such as neural networks or time series to generate predictions, this algorithm obtains strong results with two intuitive steps: search and calculation.

VI. Conclusion

In this paper, we presented a two-step approach to the classical link recommendation problem. Extending ideas from previous papers focused on sign prediction or unweighted link prediction, the algorithm utilized the properties of the weighted bipartite graph to determine two parameters: 1) the most relevant edges that could appear based on pre-existing edges, and 2) the weights of the edges based on the idea of *similarity*.

In the first step, the algorithm considered whether the candidate bottom node had common neighbors with bottom nodes adjacent to the candidate top node. The algorithm proceeded to compared this number against the total number of bottom nodes adjacent to the candidate top node. The algorithm then further determined whether any edges considered sufficient were a result of low sample size. The exact tuning parameters were determined by the density and size of the datasets.

In the second step, the idea of *similarity* was used. Similarity between the candidate bottom node and candidate top adjacent bottom nodes was calculated by taking into account how closely common neighbors were rated. The weight of the edge between the candidate bottom node and the candidate top was more heavily influenced by higher similarity pairings and less influenced by lower similarity pairings.

This algorithm was tested using the Movielens and Epinions datasets, and it performed well compared with existing algorithms. Further work will refine and improve the results of the proposed algorithm. In particular, future work can explore means to optimize its performance on larger datasets as well as how to fit the constant terms better for each particular datasets.

Acknowledgment

I would like to thank Dr. Zhiliang Xu, professor of Applied and Computational Mathematics and Statistics at the University of Notre Dame, for introducing me to this topic and providing invaluable guidance and support during the research.

References

[1] R. C. Read and R. J. Wilson. *An Atlas of Graphs (Mathematics)*. Clarendon Press, 2005.

[2] J. A. Bondy and U. S. R. Murty. *Graph Theory With Applications*. Great Britain: Macmillan Press, 1976.

[3] C. C. Aggarwal, ed. *Social Network Data Analytics*. New York, NY: Springer, 2011.

[4] K. Kobysheva, N. Voinova, I. Nikiforova. "Hybrid image recommendation algorithm combining content and collaborative filtering approaches". *Procedia Computer Science*. 2021.

[5] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, M. Wang. "LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation". *SIGIR '20: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2020.

[6] H. Zhu et al. "Bilinear Graph Neural Network with Neighbor Interactions" *International Joint Conferences on Artificial Intelligence*. 2020.

[7] P. Bedi, A. Gautam, S. Bansal, D. Bhatia. "Weighted Bipartite Graph Model for Recommender System Using Entropy Based Similarity Measure". *The International Symposium on Intelligent Systems Technologies and Applications*. 2018

[8] P. Kumar, D. Sharma. "A potential energy and mutual information based link prediction approach for bipartite networks". *Scientific Reports*. 2020.

[9] O. Allali, C. Magnien, and M. Latapy. “Link prediction in bipartite graphs using internal links and weighted projection”. *2011 IEEE Conference on Computer Communications Workshops* (INFOCOM WKSHPS). 2011.

[10] N. A. Rahman. *A Course in Theoretical Statistics*. Charles Griffin and Company, 1968.

[11] Wikipedia, ”Collaborative filtering”.
https://en.wikipedia.org/wiki/Collaborative_filtering, 2022.

[12] X. He et al. “Neural Collaborative Filtering”. *WWW ’17: Proceedings of the 26th International Conference on World Wide Web*. Apr. 2017.

[13] R. Zhang et al. “Collaborative Filtering for Recommender Systems”. *2014 Second International Conference on Advanced Cloud and Big Data*. 2014, pp. 301–308.

[14] K. Sun, T. Qian, T. Chen, Y. Liang. Where to Go Next: Modeling Long- and Short-Term User Preferences for Point-of-Interest Recommendation. *Proceedings of the AAAI Conference on Artificial Intelligence*. 2020.

[15] Z. Yu, J. Lian, A. Mahmoody, G. Liu, X. Xie. ”Adaptive User Modeling with Long and Short-Term Preferences for Personalized Recommendation”. *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*. 2020. pp 4213-4219.

[16] H. Ghaleb and M. Abdullah-Al-Wadud. “An Enhanced Similarity Measure for Collaborative Filtering-based Recommender Systems”. *2019 International Conference on Sustainable Technologies for Industry 4.0 (STI)*. 2019, pp. 1–4.

[17] Z. S. Patrous and S. Najafi. “Evaluating Prediction Accuracy for Collaborative Filtering Algorithms in Recommender Systems”. PhD thesis. Stockholm, Sweden: KTH Royal Institute of Technology, 2016.

[18] P. Massa. http://www.trustlet.org/downloaded_epinions.html. 2003.

[19] F. M. Harper and J. A. Konstan. “The MovieLens Datasets: History and Context”. *ACM Transactions on Interactive Intelligent Systems 5.4*. 2015.

[20] K. Goldberg et al. “Eigentaste: A Constant Time Collaborative Filtering Algorithm”. *Information Retrieval 4*, 2004.

[21] H. Fu, P. Poirson, K. S. Lee, C. Wang. ”Revisiting Neighborhood-based Link Prediction for Collaborative Filtering”. *WWW ’22: Companion Proceedings of the Web Conference 2022*. 2022. pp. 1009-1018.

[22] S. Rendle, W. Krichene, L. Zhang, and J. Anderson. ”Neural Collaborative Filtering vs. Matrix Factorization Revisited”. *Fourteenth ACM Conference on Recommender Systems*. 2020. pp 240–248.