Exploring graph-based approaches for non-binary rating prediction

CS224W (Analysis of Networks) course project

Anand Dhoot
anandd@stanford.edu

Vivekkumar Patel
vivek14@stanford.edu

Abstract

In this project, we explore graph-based methods to predict the rating that a user would give to a product. This is the central problem in creating a recommendation system and has been studied in great detail. Most standard approaches attempt the binary recommendation problem – estimating if a person would like/dislike an item, or if one person should be recommended as a friend to another on Facebook. This project aims to explore random-walk and other graph based techniques for problems involving non-binary ratings. We suggest a modification to the original Random Walk with Restarts setting and also propose another solution to predict rating using shortest paths in user-product bipartite graphs.

1 Introduction

The main question Recommender systems answer is to predict the rating a user would give or preference that a user would have to an item. There has been a plethora of work in this direction, more so given the widespread use of internet resulting in tons of real-life applications. Rise of large number of social platforms and more features in mobile devices have led to generation of large amounts of data and an increasing desire for users to personalize the content they see. This need to personalize the content offers exciting opportunities to businesses. To attract more users, a business would like to modify a user's content based on her/his preferences, for eg. suggesting friends to a user, or suggesting products or pages of interest. Netflix has claimed that as much as 75% of their traffic is driven from recommendations! [1]

Traditionally, several approaches have been made towards proposing a solution to the question – content-based approaches try to model the similarity between users/items using domain knowledge while Collaborative Filtering technique is a attempt to solve the problem statistically by exclusively using historical rating data. This strategy does not require explicit modeling of contextual information, but rather works by identifying similarities in rating patterns of different users and is known to perform well [2].

There are several types of Collaborative Filtering approaches which have been developed – Matrix-based methods, graph-based techniques such as Random Walks and Link Prediction, etc. The key challenge in the problem is that the data is sparse and there is little likelihood that any random user/product have interacted in the past. However, using collective information of the behavior of all users, there's much more context to the problem, and therefore, increases likelihood of making good recommendations.

Traditional systems typically use binary information (such as like/dislike, upvote/downvote, follow/unfollow) or convert non-binary information into binary to train their models. In addition, they train their models to solve the ranking problem – given a query user, predict the movie/item/product/friend that they are most likely to interact with.

In this project, we work with the Amazon review dataset, and study graph-based recommender systems for the rating task. We explore Random Walk based techniques and propose modifications to them to be able to do better predictions. We show empirically that our modifications perform better than the

base models, and these could potentially be combined with other information to build better models.

2 Previous work

There has been lot of effort on creating recommender systems in the past. As observed in [3] the set of approaches to this problem can broadly classified into a few categories: Collaborative Filtering (using historical interactions between users and items only), Content-based systems (suggestions through user & item attributes only) and hybrid methods. Two important areas in collaborative filtering are *neighborhood methods* and *latent factor models*.

2.1 Matrix Factorization

One latent factor modeling technique is matrix factorization. Koren et al.[4] use this method over the Netflix dataset. They consider feature vectors of size 20-100 for each user and product and predict ratings based on the dot product of these feature vectors. Using gradient descent, they try and refine the feature vectors to minimize the error in predicted ratings.

This method has become very popular recently because it can also be used as an unsupervised learning method for latent variable decomposition and dimensionality reduction [5]. We use this as one of the baselines to compare our model against.

2.2 Similarity based methods

Similarity based methods make use of some similarity measures to make rating or ranking predictions. Millan et al. [6] make use of an asymmetric similarity measure to calculate similarity between two users and form a similarity matrix. Using this matrix, they extract clusters of users through thresholding and then make recommendations based on the users' past behaviors for the target user.

Various similarity measures are used to make predictions. Some of the popular similarity measures are Cosine and Pearson similarity. There are also some graph based similarity measures such as the Jaccard Coefficient and Adamic-Adar measure, which can be used to make predictions.

2.3 Graph-based systems

More recently, with the increase of social networks such as Amazon, Yelp, Pinterest, etc., there has been increased interest in providing recommendations through graph algorithms as opposed to the earlier matrix-based methods (since graphs can potentially capture longer range influences). The most common method to create graph-based recommendation systems is to create a user-item bipartite graph and use it to generate recommendations.

Zhou et al. [7] proposed one of the first algorithms to project this bipartite graph into the user space (therefore capturing user-user similarity from the bipartite graph directly). They propose the creation of a weighted projection in order to capture & model the information in the bipartite graph. Other examples include pin recommendations [8] and topic modeling [9] on Pinterest over the pin-board and pin-topic bipartite networks. Nearly all of the papers talk about projection of unweighted bipartite graphs, which is an easier problem than weighted bipartite graphs.

The recommendation problem can also be modeled as a Link Prediction problem on the bipartite graph. Some examples of such efforts include Link Predictions on the Foursquare network [10], drug-disease network [11]. In these cases as well, the bipartite graphs are unweighted, resulting in lesser complexity than unweighted graphs that would typically be created using typical product-review datasets.

Yet another approach to suggest recommendations using graph-based approaches is by performing random walks over the bipartite graph. Some efforts in this direction include identification attributes corresponding to an object [12] and finding relevant pins on the Pinterest network [13]. Using random walks essentially generates a ranking of the relevant items/objects that a user may like and therefore is a good method to display a collection of objects/items as recommendations from which the user could choose from. We plan to extend this work to the rating prediction problem.

3 Dataset

We work on the Amazon review dataset [14]. This is a real-world dataset of people's reviews to Amazon products along with related metadata such as the exact rating, review text, product description, category information, links to other products (also viewed/also bought). For this project, we work exclusively with the rating information to empirically evaluate the performance of graph-based systems. This work can be extended to include contextual information from the other attributes to improve on the performance of our model.

While the review data is available for all Amazon categories, for this project we focus on a subset of this dataset, split by categories due to limitations of memory & compute. To avoid the cold-start problem, we consider the 5-core versions of the review dataset. This set contains reviews such that every user and item both belong to at least 5 reviews. Ideally we would like to work with categories in which the 5-core forms a large percentage of the total number of reviews to ensure we are less prone to noise.

Category	Total reviews (x 1000)	Reviews in 5-core (x 1000)	fraction of total
Books	22,507	8,898	39.5%
Electronics	7,824	1,689	21.6%
Movies & TV	4,607	1,698	36.8%
CDs & Vinyl	3,749	1,098	29.3%
Clothes, Shoes,..	5,749	277	4.8%

As can be observed from the adjoining table, 'Books' and 'Movies & TV' have the highest fraction. For this project, we report performance on the 'Movies & TV' category (1.7 million reviews in 5-core, 37% of total) for reasons like availability of compute.

3.1 Creating the graph for our models

For using graph-based algorithms recommender systems, a common strategy is to create an item-user undirected bipartite graph. This is done by creating a graph where every item and every user corresponds to a unique node in a graph. An item node and a user node are connected if that user has ever reviewed that particular item. Two item nodes or two user nodes are never connected to each other. A toy example is shown in 1.

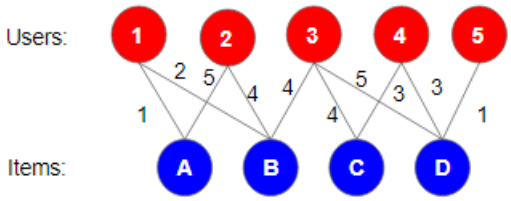

Figure 1: User-Item Bipartite Graph

When working with problems with non-binary ratings, a weighted bipartite graph is often used to capture all the information in the dataset. The weight of every edge is the (normalized) rating given by the user to the item which the edge is connecting.

Detailed statistics of our dataset such as the degree distribution, distribution of ratings, etc are in the 'Experimental results' section.

4 Algorithms/Models

4.1 Random walks with restarts

This is one of the most commonly used algorithms to generate a list of similar recommendations. This was briefly discussed in class as well in the context of finding relevant pins on the Pinterest graph. The pseudo-code used in the algorithm is spelled out in 1.

The essence of this algorithm is similar to the calculation of Personalized PageRank, with the teleport set being the *src* node. The idea behind using Random Walks with Restarts is that the random walk would frequently visit nodes that are 'easily reachable' from the *src* node, and therefore, the visit counts returned by the algorithm could be used to generate a relevance ranking. This project focuses on rating prediction task for reasons described above; and therefore the visit counts are translated into ratings by mapping uniformly on a 1 to 5 scale – the top value corresponds to rating 5 and the last value to 1.

Algorithm 1: The basic algorithm to get visit counts for every node by performing Random Walk with Restarts

Input : A weighted bipartite graph G, number of random walks to perform N, source node to start the random walk from src

Output: Visit counts of every node $n \in G$

```
1  for i ∈ 1, 2, ··· N do
2      Node u = src
3      for s ∈ 1, 2, ··· numSteps do
4          v ← randomNeighbor(u)
5          visitCount[v] + +
6          u ← v
7          if random() < β then
8              break
9          end
10     end
11 end
12 return visitCount
```

Note that this algorithm is easily scalable – every random walk is independent from each other and every call to this function independent of another call. In fact, it can handle dynamic changes in the underlying graph structure as well, with minimal overhead.

4.2 Weighted Random Walks

Since we are working with non-binary ratings in this project, our bipartite graph is weighted. We therefore adapt the pseudo-code in Algorithm 1 to weighted graphs by modifying line 4. Instead of choosing a random neighbor of node u, we do a weighted random sampling between neighbors of u, where the weight corresponds to the edge weight in the bipartite graph.

This ensures that higher ratings create a bias to take that outgoing edge of the respective node, therefore resulting in higher probability of reaching relevant nodes. In other words, in the case when a user rates some product p_1 with higher rating than p_2, the number of times node p_1 is visited would be higher than p_2. Similarly, from every product, it is more likely to walk to a user node who rates that product highly. Therefore, when the visit counts over all product nodes are normalized, the products having a higher 'score' would correspond to those to which the user is likely to give a higher rating. This approach therefore gives an implicit ranking of all the products that a user may 'like' or prefer.

4.3 Incorporating edge weight information in predictions

Observing the visit counts obtained from Weighted Random walks, we observed that most nodes remain unexplored, or have very small visit counts (less that 0.01% for the node with largest visit counts). Therefore, we propose an alternative method for predicting the rating using visit counts. The pseudo-code is explained in Algorithm 2. The idea is to have a more explicit handle on the predicted ratings through the edge weights.

The rationale behind this algorithm is that if a product was reached by a series of low-weight edges in the Random Walk, the rating that gets allotted to the product should also be low (and the other way round as well). Since the final prediction is not based on visit counts, the quality of ratings is the only factor that directs the ratings here, and that may be noisy for nodes which are reached a very small number of times. We call this approach 'Average over Random Walk with Restarts' in the subsequent sections of this paper.

4.4 Average over Shortest Path

Yet another novel approach we implement is to use shortest paths for rating prediction. This banks on the idea of influence flow from one user to another.

The exact algorithm computes the prediction between a user u and a product p by simply finding the average of the average rating over all shortest paths between u and p. The average weight across any one shortest path models the influence of one user flowing to the other and we consider every such shortest path with equal weight to average out the noise.

This modeling of influence flow between two users is typically performed on the user or item projections of the bipartite graph or by creating the user and item

Algorithm 2: Rating prediction using average over edge weights from Random Walk with Restarts

Input : A weighted bipartite graph G, number of random walks to perform N, source node to start the random walk from src

Output: Predicted ratings for every node $n \in G$

```
for n ∈ Nodes(G) do
    sumAvgWts[n] = 0
    visitCount[n] = 0
end
for i ∈ 1, 2, ··· N do
    Node u = src
    sumWalkWeights = 0
    numSteps = 0
    for s ∈ 1, 2, ··· numSteps do
        v ← randomNeighbor(u)

        // update walk specific variables
        sumWalkWeights += weight(u, v)
        numSteps++

        // update global variables
        sumAvgWts[v] +=
         sumWalkWeights/numSteps
        visitCount[v] ++

        u ← v
        if random() < β then
            break
        end
    end
end
for n ∈ Nodes(G) do
    predictions[n] =
     sumAvgWts[n]/visitCount[n]
end
return predictions
```

similarity measures by other techniques. However, most real graphs of this scale have a short diameter (longest shortest path) – for the bipartite graph that we are working with, the diameter is 5.8 (or 3 hops back and forth between items and users). Therefore, finding all all possible shortest paths in such a network becomes computationally tractable. (Shortest paths between any two nodes must have length 3 or 5).

5 Dataset observations & Baselines

5.1 Evaluation strategy

To be able to quantitatively evaluate our results and compare our approaches with baselines and with each other, we split our data into train and evaluation sets.

We then use our train set to create the graph that our algorithms need and predict a rating for every example in the test set. Finally, to compare results, we evaluate our algorithms using a simple RMSE measure against the gold labels.

The other approach we explored and then discarded in favor of rating prediction was to evaluate a relevance ranking (which is a natural output of the Random Walk strategy) due to lack of availability of true ranks – fields in the metadata such as 'related_products', 'also_bought' and 'also_viewed' were not available for all products.

5.2 Statistics of the dataset

First up, we plot some basic statistics of the dataset we're using. The degree distribution of the bipartite graph generated from our dataset is in Figure 2.

It can be observed that the degree distribution follows the expected power-law trend. The initial part of the graph deviates from the power law distribution, because this graph was created by removing the users and products with less than 5 reviews. It is likely that most of those reviews were for high-degree nodes (by the Preferential Attachment argument). Our graph has a total of 1.7 million ratings (edges) between about 125k users and 50k products. This

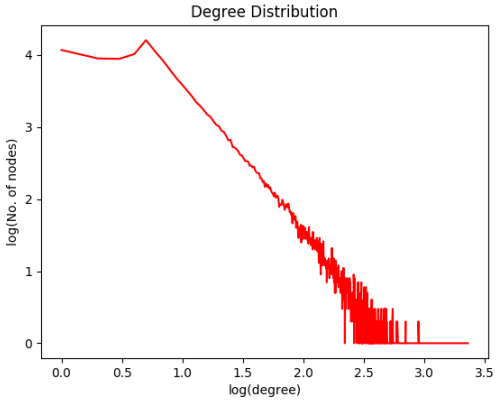

Figure 2: Degree distribution of the User-Item Bipartite Graph

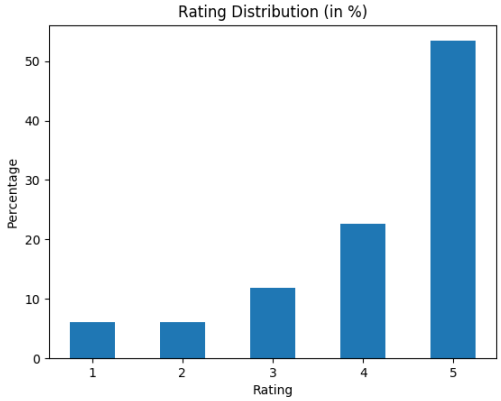

Figure 3: Distribution of ratings in the Movies and TV dataset

corresponds to about 14 reviews by every user and nearly 34 reviews for every product in the dataset on average.

The distribution of the absolute rating value shows some more interesting trends though. The histogram plotted in Figure 3 tells us the high level details of the dataset such as the overall popularity of products, the general rating trend followed by users. We see that more than half of the ratings are 5/5 and ratings of 1 or 2 out of 5 each correspond to less than 8% of the data. In fact, the average rating of all products in our dataset is 4.1.

This observation becomes pretty crucial to explain our results: assuming you predict the average rating for all examples in the test set, the maximum contribution to the RMSE penalty is from the labels 1 and 2. In addition, since the contribution increases with the square of difference, the misclassification cost of 1 as a 4 (=9) is more than two times as large as that of misclassifying 2 as 4 (=4). However, more practically, this problem is important to model, since one would not like to recommend products to a user that they are likely to rate negatively.

5.3 Baselines

5.3.1 Collaborative filtering using Matrix Factorization

This algorithm takes as input the user-item matrix (each row corresponding to a user, each column corresponding to an item), in which an entry may either be blank (if that user has not rated that item) or is equal to the rating. We have a feature vector for every item as well as user, and we use the dot product of these vectors as the predicted rating. The total error is defined as the sum of the squared error for each rating in the user-item matrix. The algorithm starts by randomly initializing the feature vectors of each user and item, and uses gradient descent to iteratively find the correct estimates for entries of these feature vectors. After training the model so that the training error is minimal, we can predict the missing entries by just computing the dot product of the corresponding user and item feature vectors.

We use this method as a baseline to compare against our graph based method. This method also suffers from data sparsity. If enough ratings are not available initially, then the regression may not converge to the required values.

5.3.2 Cosine Similarity

For M users $\{u_1, u_2, u_3..., u_M\}$ and N items $\{i_1, i_2, i_3, ...i_N\}$, a MxN user-item matrix is constructed, with its (a, b)-th entry R_{ab} equal to the rating of the item i_b given by user u_a. If user u_a has not rated item i_b, then we set $R_{ab} = 0$. Let $\mathbf{u}_a$ denote the a-th row of this matrix. We call it the user-vector for user u_a. Similarly, item vector for item i_b, would be the i-th column of the matrix, denoted by $\mathbf{i}_b$. The amount of similarity between users p and q can be calculated as:

$$Sim(u_p, u_q) = \frac{\mathbf{u}_p.\mathbf{u}_q}{||\mathbf{u}_p||.||\mathbf{u}_q||}$$

While this is a simple model of similarity, the key concern is the large dimensionality of the user-vector. However, this forms a good baseline to evaluate our model against.

To generate a rating for a user p and product q, we take a linear combination of the ratings of this product by the similar users. We calculate the predicted rating as follows:

$$\hat{r}_{pq} = \bar{r_p} + \alpha \sum_{j=1}^{n} sim(u_p, u_j)(r_{jq} - \bar{r}_j)$$

where

$$\bar{r}_p = \frac{1}{|I_p|} \sum_{i \in I_p} r_{pi}$$

i.e. $\bar{r}_p$ is the average rating given by user u_p, and α is a normalizing factor such that absolute value of the similarity metrics sum to 1.

5.4 User and item graph projection

Given the user-product ratings, we would like to create a graph where every node corresponds to a either user or item, and two nodes are connected if the two corresponding users (or items) have rated the same product (or have been rated by the same user). The interesting aspect here is to find out the weight that needs to be assigned to every edge when doing the prediction.

We look at the user and item projection graphs to analyze various properties of these graphs such as the connectedness, diameter of the graph, shortest path lengths between any two nodes and the degree distribution. We require that each pair of user and item, which is not connected, have atleast one path in the original bipartite graph and are also not too far from each other (otherwise, the predicted rating will be a random average). Higher the number of shortest paths between a user and item, better is the average predicted. Connectedness and small diameter of these projections will imply more information for us to work with.

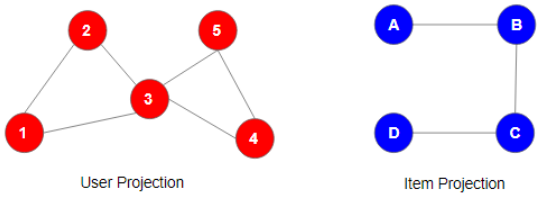

Figure 4: Projections of the Bipartite Graph

5.5 Analysis of Baselines

The performance of matrix factorization deteriorates as the sparseness of the user-item rating matrix increases. Careful tuning of the hyper-parameters such as the learning rate, regularization factor and the number of features to be learnt for each user or item (which we denote here by k), is needed. Decreasing the learning rate leads to better convergence but increases convergence time. Increasing number of features per user or item leads to more flexibility in the modeling but also increases the space and time complexity of this algorithm.

Cosine similarity also suffers in case of data sparsity as it becomes difficult to find similar users or similar items. Due to high sparsity, the similarity scores are very low and this reduced the accuracy of predictions. To circumvent this problem, we experimented using a similarity threshold in which, we consider other items only if their similarity is higher than the specified threshold. This led to another problem. In case of many products, as the similarity scores were already low, using a threshold resulted in no

similar product and hence no prediction was made for that product. On using a similarity threshold of 0.5, this method failed on predicting ratings of 80% of test examples.

A simple random walk can learn a good number of features of a product, even when the data is sparse, due to the graph structure. The score that each node receives in a random walk is dependent on various graph metrics such as the degree of the node, its centrality and number of short paths it has to any other user or item node. These features prove useful in calculating the user-item ratings. For time and space complexity, graph algorithms are generally $O(n+m)$ or $O(|E|)$, where n is the number of users, m is the nummber of items and $|E|$ is the number of edges, which is better than matrix methods.

6 Results and Analysis of our algorithms

6.1 Random walk approaches

The results (RMSE errors from the predictions) of approaches based on Random Walks with Restart(RWR) have been compiled in Table 6.1. We observe a significant difference between the two Random Walk approaches, which we have tried to investigate below.

No.	Method	RMSE
1	Random	2.62
2	Cosine Similarity	1.69
3	Matrix Factorization	1.29
4	Weighted RWR	1.39
5	Average over RWR	1.21

Firstly, we see that the traditional Weighted RWR approach outperforms the Cosine similarity measure & underperforms Matrix Factorization approaches by a small amount. Our hypotheses are:

- Matrix factorization has good performance with non-sparse user-item matrices. We attribute the good performance to the rich information present in the 5-core graph that we're using.
- The regular weighted RWR generates counts of the number of times each node was visited during the Random Walk process. To investigate further, we plotted a graph of sorted visit counts for the entire graph (Figure 5). We notice that most nodes are not visited by the Random Walk at all, or are visited very few times (likely due to the weighted nature of the graph and high centrality of a few nodes) and therefore, this model does not make a good prediction for those examples.
- We believe that one of the reasons for high RMSE is the problem of converting rankings to ratings. We currently use a naive method to do this. The 'Average over RWR' approach is an attempt to fix this conversion from a ranking to give a rating score. Computing an average over all Random Walks that reach the destination node results in more accurate classifications and therefore, a lower loss.

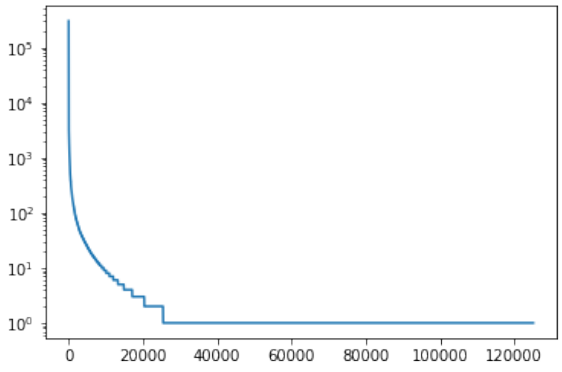

Figure 5: Distribution of Random Walk Visit counts

6.2 The shortest path approach

We observe that the performance of the shortest path approach is the best out of the three categories of models we study.

No.	Method	RMSE Error
1	Matrix Factorization	1.28
2	Average over RWR	1.21
3	Average over Shortest Path	1.05

We attribute the superior performance of this algorithm to the following reasons:

- This algorithm attempts to capture the local interactions between a user and item. A user's rating to a product would be similar to other ratings that this product has received. If a product is highly (or lowly) rated, then a new user would more likely rate the product high (low) as well. Hence, the weight of the edges surrounding this product would be higher (lower). A path that goes from the query user to the query product would incorporate at least one of these high (low) weight paths, and therefore, that gets incorporated in the rating prediction. The same argument can be extended to claim that the approach also takes into account the user's behaviour. If a user generally rates every product with low (high) ratings, then all the paths going from this user to any item would have low (high) averages.
- Considering all shortest paths between a user and product help to capture varied types of interactions through the user(s) and product(s) on the shortest path itself.
- However, we believe that most of the gains on using this approach comes from better identification of products that receive low ratings because now, those low ratings get considered in the average.

7 Conclusion

We present a few graph based algorithms and test their performance on the Movies & TV category of Amazon Review Data. We present analysis of the dataset and compare the performance of our methods against two traditional baselines. We observe that the Average over Shortest Paths method gives the best results. Also, graph methods scale better than other techniques as the size of the dataset increases. By preprocessing certain things and using correct data structures, one can greatly reduce the time complexity of these algorithms, which is very essential while making recommendations in real time.

Our methods are general and do not assume any knowledge of the underlying graph structure, and can be extended to other problems and potentially combined with other with contextual information to give even better results. It would be interesting to compare the performance of these methods with different classes of non-binary data rating data.

References

[1] "Netflix recommendations: Beyond the 5 stars." https://medium.com/netflix-techblog/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429.

[2] X. Su and T. M. Khoshgoftaar, "A survey of collaborative filtering techniques," *Adv. Artificial Intellegence*, vol. 2009, pp. 421425:1–421425:19, 2009.

[3] P. Melville and V. Sindhwani, "Recommender systems," in *Encyclopedia of Machine Learning and Data Mining*, 2010.

[4] C. V. Yehuda Koren, Robert Bell, "Matrix factorization techniques for recommender systems," in *Computer*, 2009.

[5] D. kumar Bokde, S. Girase, and D. Mukhopadhyay, "Role of matrix factorization model in collaborative filtering algorithm: A survey," *CoRR*, vol. abs/1503.07475, 2015.

[6] M. Millan, M. Trujillo, and E. Ortiz, *A Collaborative Recommender System Based on Asymmetric User Similarity*, pp. 663–672. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.

[7] T. Zhou, J. Ren, M. Medo, and Y.-C. Zhang, "Bipartite network projection and personal recommendation.," *Physical review. E, Statistical, nonlinear, and soft matter physics*, vol. 76 4 Pt 2, p. 046115, 2007.

[8] D. Liu, C. Lu, and K. Mohan, "Pinterest analysis and recommendations," 2014.

[9] J. Han, D. Choi, A.-Y. Choi, J. Choi, T. Chung, T. T. Kwon, J.-Y. Rha, and C.-N. Chuah, "Sharing topics in pinterest: Understanding content creation and diffusion behaviors," in *COSN*, 2015.

[10] R. Fortuna and U. Marovt, "Link prediction in foursquare network," *CoRR*, vol. abs/1602.03636, 2016.

[11] E. Gündoan and Buket Kaya, "A link prediction approach for drug recommendation in disease-drug bipartite network," *2017 International Artificial Intelligence and Data Processing Symposium (IDAP)*, pp. 1–4, 2017.

[12] M. A. Yildirim and M. Coscia, "Using random walks to generate associations between objects," in *PloS one*, 2014.

[13] D. C. Liu, S. Rogers, R. Shiau, D. Kislyuk, K. C. Ma, Z. Zhong, J. Liu, and Y. Jing, "Related pins at pinterest: The evolution of a real-world recommender system," in *WWW*, 2017.

[14] R. He and J. McAuley, "Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering," in *WWW*, 2016.