CS224W Project Report

Product Recommendation System

Jianfeng Hu
jfhu@stanford.edu

Bo Zhang
bzhang09@stanford.edu

December 10, 2012

1 Introduction

In this paper, we are going to study about recommendation systems. Recommendation systems are typically used by companies, especially e-commerce companies like Amazon.com, to help users discover items they might not have found by themselves and promote sales to potential customers. A good recommendation system can provide customers with the most relevant products. This is a highly-targeted approach which can generate high conversion rate and make it very effective and smooth to do advertisements. So the problem we are trying to study here is that, how to build effective recommendation systems that can predict products that customers like the most and have the most potential to buy.

Based on the research on some existing models and algorithms, we make application-specific improvements on them and then design three new recommendation systems, Item Similarity, Bipartite Projection and Spanning Tree. They can be used to predict the rating for a product that a customer has never reviewed, based on the data of all other users and their ratings in the system. We implement these three algorithms, and then test them on some existing datasets to do comparisons and generate results.

2 Prior Work

There has been a lot of work done in this field. For example, one very popular algorithm is Collaborative Filtering. One type of collaborative filtering is user-based collaborative filtering, which starts by finding a set of customers who have purchased and rated similar items with the target users purchasing history. The algorithm aggregates items from these similar customers, and uses the ratings from other similar users to predict the ratings from this user. Another type of collaborative filtering is item-based collaborative filtering, which was first brought up by Amazon [4] and focuses on finding similar items instead of similar

customers. For each of the users purchased and rated items, the algorithm attempts to find similar items. It then aggregates these similar items and recommends them.

There are also other algorithms that try to exploit graph structures to predict links or ratings. Random walks algorithms [2] could be used in predicting links in complex graphs in a very efficient manner. And also, if we model the user and product graph as a bipartite graph, then it is also feasible to use Bipartite Projection algorithm [5] to calculate the relevance between two customers. So the predicted rating is essentially based on the other relevant customers' ratings. In later sections of this paper, we will introduce three models and algorithms which are derived from the prior work mentioned above with application-specific improvements.

3 Methods

3.1 Baseline

As a first step, we implement a simple random recommend system as our baseline system. This system returns a random rating for each (product, customer) pair. Because the way it works, we expect it to have the worst performance. We will use this baseline system for comparisons with other algorithms.

3.2 Item Similarity

The first recommendation system we build is inspired by Amazons item-based collaborative filtering [4]. In Amazons algorithm, they represent each item with a vector showing who bought/reviewed the item. Similarity between these two products is defined by the cosine of the two vectors. After calculating similarity between all product pairs, we will have an item-item matrix showing the similarity between the items. Finally, the similarities can provide a good reference on some of the other products that a customer would buy.

The original algorithm is used to predict the next product that a customer would buy. To adopt it in our application, which is to predict the rating given by some customer for some product, we create an algorithm that make use of the item-item similarity.

First lets define some terms that we will use later. Let $w(i, j)$ be the similarity between item i and item j; I_u is the set of products customer reviewed, excluding the one we are going to predict with; $rate_u(i)$ is the rate for product i given by customer u. $S(i)$ is the most similar items with item i, including i itself, according to the item-item similarities. Finally, let x be the item that we are trying to predict for customer u.

We calculate a weighted sum for each $j \in S(x)$. For each item i that customer u have a rating, we give them weights using the similarity between i and j:

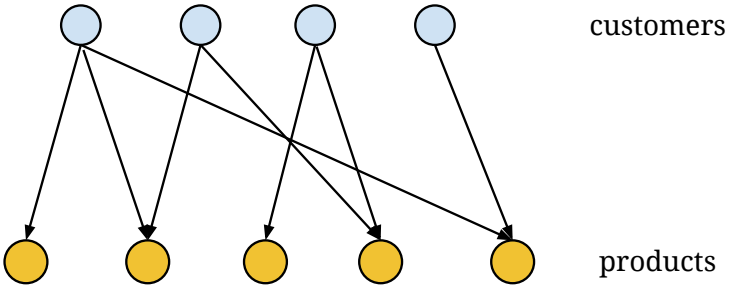

Figure 1: Step 1 Customers to products

$$sum_j = \frac{\sum_{i \in I_u} w(i,j) rate_u(i)}{\sum_{i \in I_u} w(i,j)} \tag{1}$$

This weighted sum basically indicates that, for an item j which is similar to x, what would the rating be for j given all the ratings for $i \in I_u$. Then, we take another weighted sum over each j, where the weights are given by the similarity between j and x. The rating for item x is then given by:

$$rate_u(x) = \frac{\sum_{j \in S(x)} w(x,j) sum_j}{\sum_{j \in S(x)} w(x,j)} \tag{2}$$

3.3 Bipartite Projection

The Amazon's purchase network can be modeled as a bipartite graph. The nodes can be divided into two sets, one set for the customers and the other for the products. Edges exist only between two nodes in different sets. Each edge carries a rating, representing the rating of a product given by a customer.

When doing recommendation to a customer, we learn the similarity between two customers so that we can calculate how much a customer's rating is affecting the other. Besides calculating similarities through collaborative filtering, we can also calculate the user-user coefficient by projecting the bipartite graph to the customer set. The projection will generate a graph only with customer nodes and also show the structure among them.

The most naive way of doing projection is to simply draw an edge between a pair of customers if they have reviews on the same product. However, the resulting unweighted graph may not retain enough information. Zhou et al [5] proposed a method based on resource allocation for calculating weights between a pair of nodes when doing bipartite graph projection.

The resource allocation treats weights as a format of resource. In our case, the weights are the ratings given by the customers. It takes two step. At the first step, the projected set

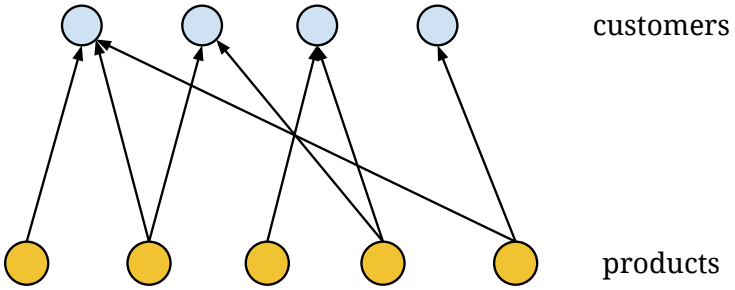

Figure 2: Step 2 Products to customers

(customer set) evenly distribute their ratings to the non-projected set (product set). At the second step, the ratings are projected back to the customer set.

The algorithm is modified from Zhou et als paper [5] to accommodate the 5-star ratings in the dataset. Let customer u_i and product p_l be two nodes in the graph. a_{il} is the rating for product l reviewed by customer i. $k(x)$ is the degree of a node x in the graph. Let w_{ij} be the proportion that customer j would like to distribute its ratings to customer i. From the two-step process, the weight is given by

$$w_{ij} = \frac{1}{k(u_j)} \sum_l \frac{a_{il} a_{jl}}{k(p_l)} \tag{3}$$

Then, an algorithm goes through the weights and build the recommendation list. For each product l, let $\tilde{a}_{il}$ be the predicted rating of that product given by customer i. We sum over for each customer j, for any rating on product l that j would like to contribute to i. In order to get a rating back in the same range, again we calculate the weighted sum of these ratings.

$$\tilde{a}_{il} = \frac{\sum_j w_{ij} a_{jl}}{\sum_j w_{ij}} \tag{4}$$

In practise, the recommendation system calculates the $\tilde{a}$ for all products and recommends the ones with highest score to the customer. For the purpose of testing, we are only using a single score.

3.4 Spanning Tree

We also design an algorithm that makes use of the graph structure. The algorithm explores a new way of defining similarities between products and customers. In this algorithm, we still model the product-customer relationship as a bipartite graph. One set of the node is the product nodes and the other set is the customer nodes. An edge connecting a product to a customer indicates a review of that product. The edge also carries a rating associated with that review.

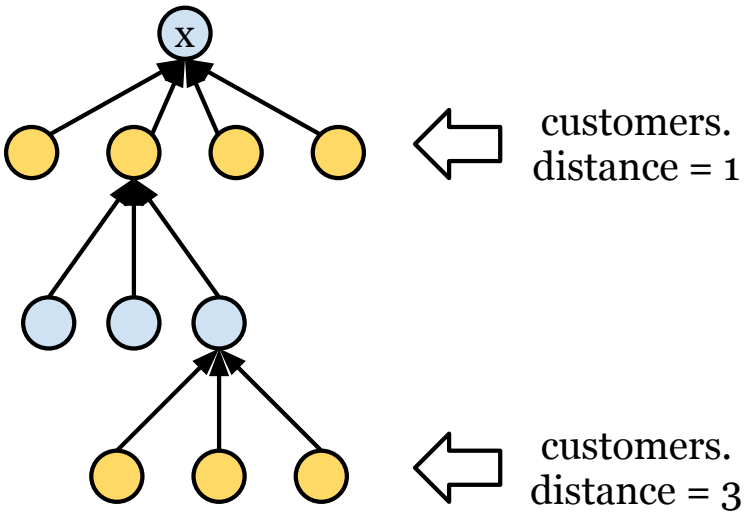

Figure 3: Sample Spanning Tree

Now for example, we want to predict the rating between customer u and product x. The algorithm will create a spanning tree from the graph, rooted at product x. The spanning tree will also have two kinds of node, product node and customer node. Each node may have several children. For a product node, the children are the customers that have reviewed the product; for a customer node, the children are the products that he/she have reviewed. Figure 3 shows an example of such spanning tree. When doing the experiments, we truncate the tree at certain distance because it would be otherwise too big to search through.

Each customer node will have an edge points to its parent, which is always a product. This edge carries the rating that the customer gave to the product. Let $parent(u)$ be the parent product node of some customer u. We also define the distance between product i and customer u — $distance(i, u)$ to be number of edges on the path between i and u. As we can see from Figure 3, the first layer of customers will have distance 1, the next layer of customers will have distance 3, etc. We define the distance because further away the customers from product i the less relevant their ratings will be. We would like to have a weight on their ratings, which is related to distance and decay exponentially, because the number of nodes on the spanning tree would increase exponentially. We defined the weight function to be:

$$w(i, u) = e^{-distance(i,u)} \tag{5}$$

So finally, with U denotes the set of customers, the predicted rate that customer u gives product x would be given by:

$$rate(x) = \frac{\sum_{u \in U} rate_u(parent(u)) \cdot w(x, u)}{\sum_{u \in U} w(x, u)} \tag{6}$$

4 Results and Findings

4.1 Testing Framework

We make use of the "Amazon Product co-purchasing network metadata" [3], a dataset available on the Stanford Large Network Dataset Collection [1]. This dataset contains 0.5 million products with 1.8 million co-purchasing data and 7.8 million user reviews. It provides sufficient information for us to experiment and get meaningful results and conclusion for the project.

We also build a testing framework before implementing all these algorithms. The test framework allows us to to cross-validations in our experiments. We use 10-fold cross validation in our experiments. To be more specific, we divide the customers in our dataset randomly into 10 groups of equal size. Then we pick one of the group, and randomly remove one rating from each customer in the group. The removed ratings become our test dataset. Each data in the test dataset is a triplet of product, customer, and rating. The product and customer are sent to our recommendation system for predicting, and the rating is our ground truth for testing.

Then we test our systems with the test dataset. The score is measured with mean squared error (MSE). After picking the test dataset in turns within the 10 groups, our test system returns an average MSE for the groups. This average MSE serves as our main metrics on how the recommendation system is performing. A lower score means the system is predicting ratings that are closer to what we have in the original dataset.

4.2 Results

4.2.1 All Users

Firstly we do experiments with all users. Table 1 shows the performance of all the algorithms. Different algorithm performs differently.

Item similarity algorithm takes a user's previous ratings into account, together with the similarities between the products. We have to decide the rating for user who has no other reviews solely based on the rates received by similar products. Despite it's the slowest among these algorithms, it gives a better score than the other algorithms.

The spanning tree algorithm predicts rating which is mainly decided by other people's review on the product and similar products. However, this algorithm is not personalized enough. For products with ratings of 1 and 2, it often predicts higher score because most people in the dataset gives higher ratings to products.

The spanning tree can become very large as the distance increases. Its also interesting to see

	Random	Item Similarity	Tree (d=1)	Tree (d=3)	Projection
MSE	5.095	**1.171**	1.483	1.898	1.642

Table 1: MSE for all users

	Item Similarity	Tree (d=1)	Tree (d=3)	Projection
MSE	1.603	1.516	**1.453**	N/A

Table 2: MSE for new users only

how the size of the tree affect the final score. We do experiments with one level of customer (max distance = 1) and two levels of customer (max distance = 3). We do not experiment with bigger tree because the branching factor for the tree is close to 20000, and it would be too slow to run for a large number of test data, but we can see from Table 1 that including more information from the dataset doesn't necessarily help on getting a better result.

The bipartite projection algorithm does not work so well in this case. We notice that for a portion of the users this algorithm cannot predict the rating. We believe its because the users or products don't have enough review data.

4.2.2 New User

We then run some tests with users with no reviews at all. This simulates the scenario where the system is going to predict rating for a new user.
From the results, we can see that the item similarity algorithm does not work as well for new users who do not have any review at all. The reason is that the calculation is closely related to the users previous reviews and when there's no such information.

However, the spanning tree algorithm works better. That's the advantage of this algorithm that it will work even when the user is a new user who has never made a review. In the case where user have no reviews, the predicted ratings will solely based on other people's opinion.

The bipartite partition algorithm in this case doesn't work because there's no product for any new user to project at the first step.

4.2.3 Old User

We also run some tests with users who have at least 10 reviews. We hope it can reveal which algorithm works better when there's sufficient data to analyze. Table 3 shows the results of this experiment.

	Item Similarity	Tree (d=1)	Tree (d=3)	Projection
MSE	1.135	1.246	1.420	**1.129**

Table 3: MSE for old users only

We can see that in this case, bipartite projection gives the best results, followed very close by item similarity algorithm. However, the item similarity algorithm is very computationally expensive, while the bipartite projection gives result within fraction of seconds.

4.3 Conclusion

Recommendation systems help users discover items they might not have found by themselves and promote sales to potential customers, which provide an effective form of targeted marketing by creating a personalized shopping experience for each customer. Lots of companies have such kind of systems, especially for e-commerce companies like Amazon.com, an effective product recommendation system is very essential to their businesses. In this paper, based on the research on some existing models and algorithms, we design three new recommendation systems, Item Similarity, Bipartite Projection and Spanning Tree. They can be used to predict the rating for a product that a customer has never reviewed, based on the data of all other users and their ratings in the system. To examine and compare their effectiveness, we implement these three algorithms and test them on some existing datasets.

In our experiments, we found that, in terms of effectiveness measured with mean squared error (MSE), for all users, Item Similarity has the best result, then followed by Spinning Tree, and Bipartite Projection is the worst. For new users, Spinning Tree has the best result, then followed by Item Similarity, and Bipartite Projection cannot even generate result because of lack of data. For old users, Bipartite Projection has the best result, then followed by Item Similarity, and Spinning Tree is the worst. In terms of computational performance, Bipartite Projection is the fastest algorithm that gives result within fraction of seconds, while Item Similarity can be very computationally expensive.

In the future, we plan to improve the effectiveness and performance by exploring a hybrid system which will apply different algorithms on different user segments. One concrete thought is to use Spinning Tree on new users and use Bipartite Projection on old users. And we also need to experiment on different criteria to decide whether a user is a new user or an old user, and then choose the criterion that has the best result.

We also would like to study how we could control or tweak the outputs of recommendation systems based on application-specific requirements. For example, the company might want to avoid recommending some very popular items to distribute the traffic to other products, or the company would like to promote some newly listed products. In general, it is an promising direction to build recommendation systems that can adapt to more granular and

flexible application-specific requirements.

References

[1] Stanford large network dataset collection. http://snap.stanford.edu/data/index.html.

[2] Lars Backstrom and Jure Leskovec. Supervised random walks: Predicting and recommending links in social networks. *Proceeding of WSDM 2011*, pages 635–644, 2011.

[3] Jure Leskovec, Lada A.Adamic, and Bernardo A. Huberman. The dynamics of viral marketing. *ACM Transactions on the Web (ACMTWEB)*, 1(1), 2007.

[4] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

[5] Tao Zhou, Jie Ren, Matus Medo, and Yi-Cheng Zhang. Bipartite network projection and personal recommendation. *Physical Review E*, page 76, 2007.