A Study of Recommender Systems with Applications

A PROJECT
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL OF THE
UNIVERSITY OF MINNESOTA
BY

Xiaowen Fang

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

Advisor: Kang James

May, 2014

Acknowledgments

First, I would like to thank the department of Mathematics for providing me this precious studying opportunity. I gratefully thank my academic advisor Dr. Kang James for her consistent supports in study and research. She always tries her best to provide helps and encouragements. I also would like to express my sincerely appreciation to Dr. Barry James and Dr. Xuan Li for being my committee and providing strong supports. I would like to thank Charles Kwong and Yi Xiao for their helps with installing and using computational programs. Last but not least, I would like to thank fellow students in the math department for their helpful discussions.

Abstract

In this project, we studied K-Nearest Neighbor (KNN), asymmetric Singular Value Decomposition (SVD) and Restricted Boltzmann Machine (RBM) methods in recommender systems. We applied these methods to three datasets: movielens100k, Amazon-Meta and R3.Yahoo!Music. By comparing their results, we found that the recommender methods are very much data oriented. Since Amazon-Meta was collected based on friend recommendation settings, we obtained the lowest root mean square error (RMSE) among the three datasets by using the KNN algorithm. The users in the Amazon-Meta dataset are correlated. Therefore RBM has larger RMSE compared to the KNN and asymmetric SVD methods. For R3.Yahoo!Music all three methods give moderate RMSE, but none of the methods give RMSE less than 1. For the movieLens100k data, asymmetric SVD shows the best prediction among the three methods.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In our daily life, we use recommendations quite often. For example, when we choose a restaurant, we may ask friends about their preferences and take their opinions into consideration. Before going to the cinema to watch a movie, often we already have a movie in our mind recommended by friends or recommended by other people's reviews online. When we want to choose a primary physician or a clinic, we may pay attention to the ratings made by other users online or the recommendations from friends. When we buy merchandise through an online store, it is very necessary for us to read reviews from other customers to have information about their quality. In LinkedIn and Facebook, there often appear recommendations about people you may know.

The rapid development of internet techniques has changed our daily lives dramatically. There is too much information nowadays. We need an automatic tool to utilize the most useful information to help us find things we like to have or to purchase without going through all catalogs or all related webpages.

A recommender system is used to predict ratings or preferences so that a few top items or services can be selected and be recommended to potential users. However, good prediction accuracy alone does not guarantee users an effective and satisfying experience. Novelty is another factor for improving the appreciation of users for the recommender system [1].

The concept of a recommender system was invented to harvest the opinions of millions of people online. It was first introduced by Goldberg et al. in 1992 with the name PARC Tapestry system [2]. In this system the idea of collaborative filtering was first formed. Two years later, GroupLens system, introduced by Resnick et al at

University of Minnesota Twin Cities, performed automated collaborative filtering [3]. In those early systems, they used algorithms such as “K-Nearest Neighbor” (KNN) that identified other users with similar tastes and combined their ratings together into a personalized, weighted average. Since then, recommender systems have been successively applied to multiple disciplines, including applications in information retrieval, data mining, business and marketing research as well as in healthcare recommendation. For example, recommender systems have been applied in product recommendation of online shopping in Amazon and Ebay etc. [4]. They have also been used in suggestion of interesting websites such as by yahoo.com, recommendation of music and movie products, e.g. Pandora; social tags, e.g. weibo.com; research articles, search queries, people, restaurants, financial services, insurance etc..

In 2006, Netflix competition was announced and a $1 million Netflix Prize was provided for a 10% improvement in prediction accuracy. Due to the availability of the large Netflix dataset, algorithmic research has received great attention.

Although recommender system has been comprehensively studied for a decade, it is still an active research area in data mining and machine learning field. Every year recommender system is a topic in many conferences, e.g. RecSys, SIGIR, KDD. The study of social-based recommender systems has just started.

In this project, we do not develop our own algorithm. Our main goal is to study the three important recommender methods and apply them to large datasets.

Chapter 2

Overview of Recommendation Approaches

Recommendation approaches are generally divided into two categories: content-based and collaborative filtering approaches.

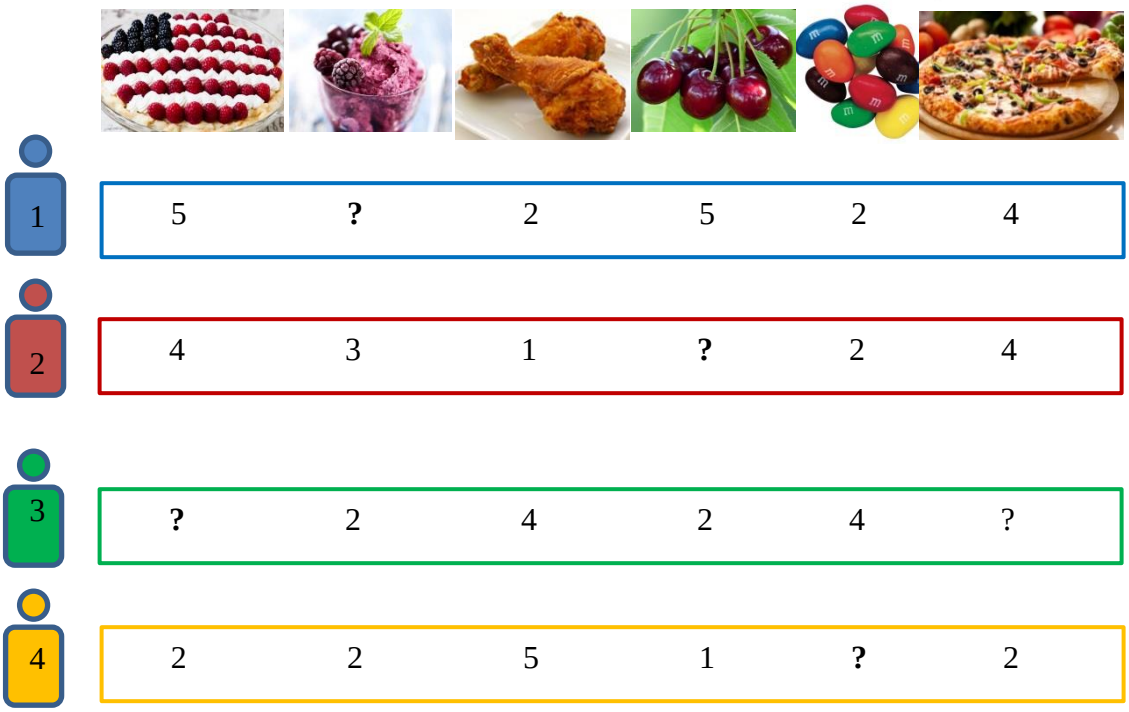
2.1 Content-Based Recommendation Approach

The job of content-based recommender systems is to identify the common characteristics of items which a user has given high ratings. Then it recommends new items which are similar to those a given user has liked in the past. For example, customer u has given a preferred rating toward comedy movies, so it will recommend more comedy movies to this customer. This technique updates the profile of user u whenever user u rates an item i, then the new user profiles can be used to recommend new items to this user.

Content-based recommendation approaches have several advantages when compared to the collaborative filtering approaches, such as: user independence, transparency, capability to handle new items which have not been rated by any users. Nonetheless, they have several shortcomings. Especially they have no inherent methods for finding unexpected items. The users are always recommended similar items to those they already rated with a limited degree of novelty. Meanwhile, they do not have methods for new users. Enough ratings have to be collected before they can be applied.

2.2 Collaborative-Filtering Recommendation Approach

Collaborative filtering approaches utilize the ratings of all users in the system. It believes that like-minded people will rate items similarly or similar items will obtain similar rating [5]. For example, the ratings of user u to some new items (unrated by user u) will be similar to the ratings of user v of the same items (rated by user v) if user u and v have rated other items similarly. In the following example, suppose we have rating data about four users toward six different foods, from left to right: fruit pie, raspberry ice cream, buttermilk fried chicken, cherry, candy, pizza. We want to predict if user 2 likes cherry or not. According to user 2's rating history, we observed that user 1 has very similar preferences to user 2 and user 1 likes cherry. Hence it is very likely that user 2 also likes to eat cherry.



In general, collaborative filtering methods contain two groups: neighborhood and model-based methods. Neighborhood methods use the ratings directly to find similar groups to do weighted prediction for unrated items. While model-based methods use ratings to model the user-item interactions with latent characteristics. Therefore these latent factor models can be used to predict ratings of users for new items.

In this project, we will focus on collaborative filtering approaches and apply three collaborative filtering approaches to the movieLens100k, R3.Yahoo! Music and 10% Amazon-meta dataset. The three approaches are KNN algorithm, Singular Value Decomposition (SVD) algorithm and Restricted Boltzmann Machine (RBM).

Chapter 3

Introduction and Statistics of Datasets

The MovieLens100k dataset was collected by the GroupLens Research Project at the University of Minnesota during the seven-month period from September 19th,1997 to April 22nd, 1998 via the MovieLens web site: http://movielens.umn.edu. This dataset contains 100,000 ratings from 943 users on 1682 movies.

The Amazon-Meta dataset was collected for studying a recommendation referral program run by a large retailer. Each time a customer purchased an item (e.g. book, music, movie etc.), he/she could get 10% credit if he/she send out email recommending the item to friends and some of these friends purchased the same item via a referral link in the email. Among these friends, the first person who did the purchasing also got a 10% discount on the item. In this study, we randomly select 10% of the total ratings. There are 663638 ratings made from 316375 users for 99169 items in our studied dataset.

The R3.Yahoo!Music rating dataset contains ratings for songs from two different groups of users. The first group was rated by 10,000 users during normal interaction with Yahoo! Music services. Each such user gave at least 10 ratings. The second group of ratings was collected during an online survey of randomly selected songs. This dataset contains 54,000 ratings from 5400 users with each user giving exactly 10 ratings. In total there are 15400 users who rated 1000 songs with 311704 ratings.

3.1 Overall Statistics

The overall statistics of the MovieLens100k, 10% of Amazon-Meta and R3.Yahoo!Music datasets are listed in Table 1 and were calculated by using Recommender101 package [6].

For the MovieLens100k dataset in Table 1(a), the sparsity is 6.3%, in other words, 93.7% of values are not available in the user × item matrix. The average rating is 3.53. On average, there are about 106 ratings per user and about 59 ratings per movie. Users who gave less than 20 ratings are excluded from this dataset while the minimum number of ratings obtained by movies is 1. The most ratings given by a user are 737 and the most ratings a movie received are 538. The Gini coefficient of rating is 0.288.

For the 10% Amazon-Meta dataset in Table 1(b), the sparsity is 0.1%, indicating that 99.9% of values in the user × item matrix are missing. The average rating is 4.159, which is very high. On average, there are around 2 ratings per user and ~7 ratings per item. The minimum number of ratings obtained by an item is 1. The minimum number of ratings given by a user is also 1. The most ratings given by a user are 86650 and the most ratings an item received are 521. The Gini coefficient of rating is 0.473.

For the R3.Yahoo!Music dataset in Table 1(c), the sparsity is 2%, indicating that 98% of values are missing in the user × item matrix. The average rating is 2.892. On average, there are around 20 ratings per user and around 312 ratings per movie. The minimum number of ratings given by an user is 10. The minimum number of ratings obtained by a song is 14. The most ratings given by a user are 648 and the most ratings a song received are 5543. The Gini coefficient of rating is 0.185.

Figure 1 shows the counts of each rating. As we could see, for the MovieLens100k dataset the frequency of rating 4 is the highest, as shown in Figure

1(a), while the frequency of rating 5 is the highest for 10% Amazon-Meta dataset in Figure 1(b), which is quite reasonable since recommendations from familiar friends offer more similar tastes than those from random selected people. From the R3.Yahoo!Music dataset plot in Figure 1(c), we could see that the two ends have higher frequencies than the middle, which indicates that people have very intense reactions toward music, either strongly liking or strongly disliking a certain music.

Table 1. Overall Statistics. From left to right, datasets are MovieLens100k, Amazon-meta (10%) and R3.Yahoo!Music Dataset.

Dataset	MovieLens100k	Amazon-Meta	R3.Yahoo!Music
#Users	943	316375	15400
#Items	1682	99169	1000
#Ratings	100000	663638	311704
Sparsity	0.063	0.001	0.02
Global avg	3.53	4.159	2.892
Ratings freqs			
1	6110	53365	97844
2	11370	37378	39652
3	27145	55486	49131
4	34174	121794	48480
5	21201	395615	76597
Gini coefficient	0.288	0.473	0.185
Avg. Ratings per user	106	2	20
Avg. Ratings per item	59	7	312
Min. Ratings per user	20	1	10
Min. Ratings per item	1	1	14
Max. Ratings per user	737	86650	648
Max. Ratings per item	583	521	5543

Figure 2 shows the relationship of the number of users who give ratings to the number of items. As we see, the number of users drops dramatically with increasing number of their rated items. Comparing the three datasets, we also find that the MovieLens100k dataset has a denser user× item rating matrix since they have a larger x-axis scale while in the Amazon-Meta dataset, a majority of users give ratings to less than 5 items. Yahoo Music is between the MovieLens100k and Amazon-Meta dataset.

Figure 3 demonstrates the relationship of the number of items which obtained ratings to the number of users. As we see, the number of items drops dramatically with increasing number of users for both MovieLens100k and 10% of Amazon-Meta dataset. However, for the R3.Yahoo!Music Dataset in Figure 3(c), the number of items first increase with the increasing of the number of users, then slowly drops. For the 10% Amazon-Meta dataset, a few people who gives ratings to more than 10k items, with behavior different from the majority of users, who only give ratings to less than 5 items as demonstrated in Figure 2 (b).

Figure 4 shows the normal Z-score plot. For MovieLense100k in Figure 2(a), the skewness is – 0.51 and kurtosis is 2.59 with sample standard deviation 1.13. Therefore we say the distribution of rating is skewed to high ratings. For 10% Amazon-Meta dataset, since majority rating is located on 5 as shown in Figure 1(b), so the peak around 0 is dominant in Figure 2(b). The skewness is -1.42 and the kurtosis is 3.76. So we would say the 10% of Amazon-Meta dataset is skewed to favorable ratings. For the R3.Yahoo!Music Dataset in Figure 2(c), the skewness is 0.075 and the kurtosis is 1.46. The ratings of the Yahoo Music dataset are quite symmetric.

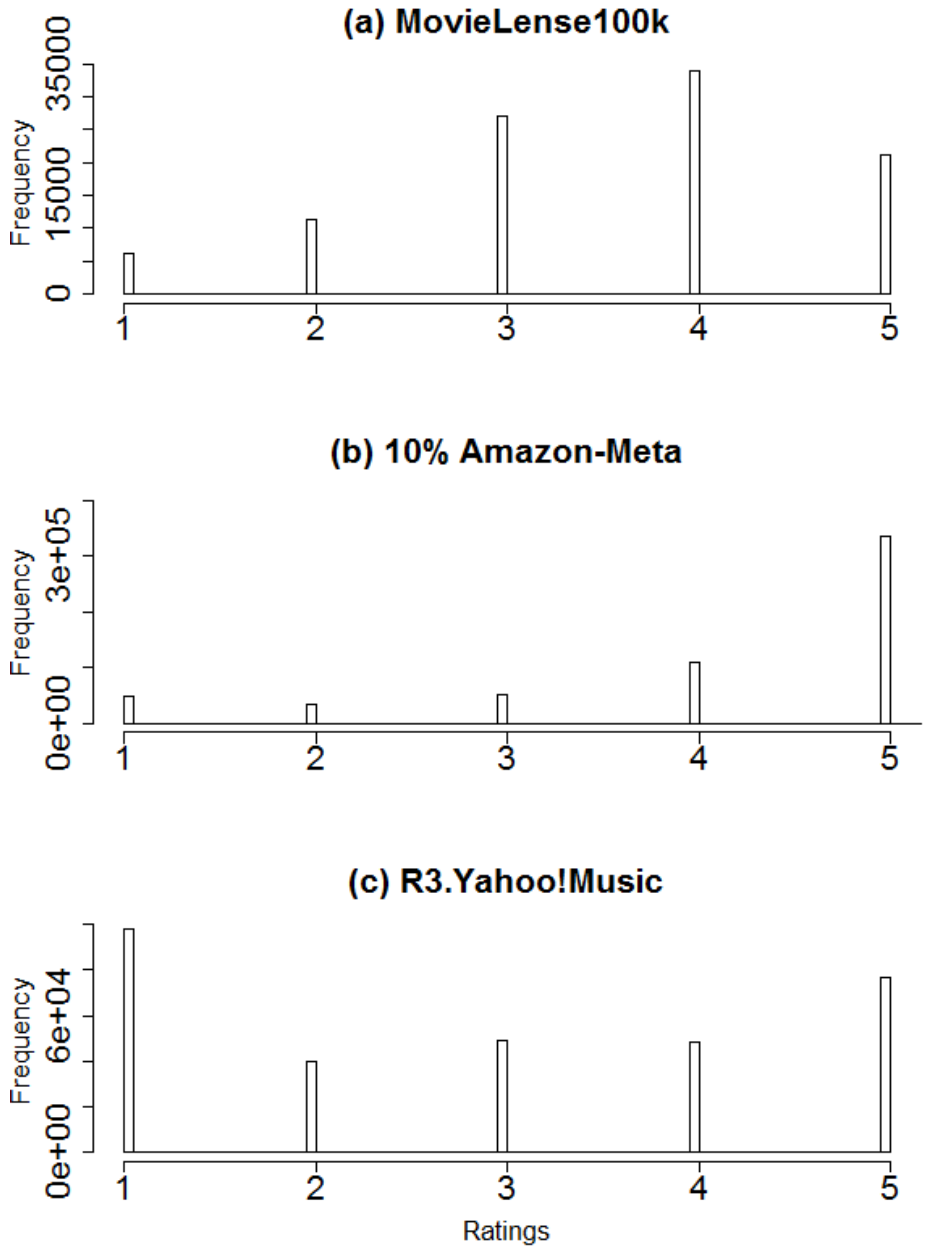


Figure 1. Histogram of counting of each rating (from dislike at rating 1 to strong like at rating 5) for (a) movieLens 100k, (b) 10% Amazon-Meta and (c) R3.Yahoo!Music dataset, respectively.

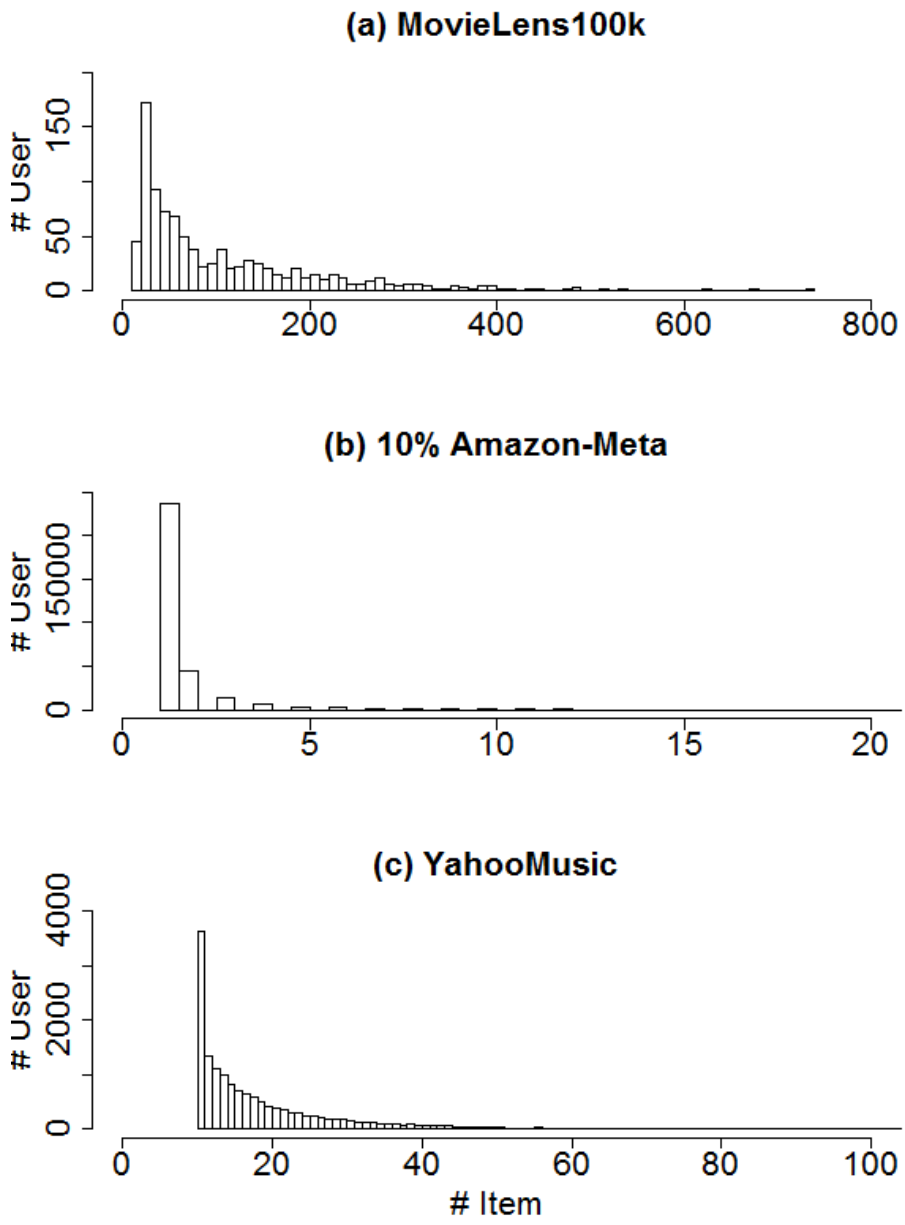


Figure 2. The number of users versus the number of items which the users give rating to in (a) movieLens 100k, (b) 10% Amazon-Meta and (c) R3.Yahoo!Music dataset, respectively.

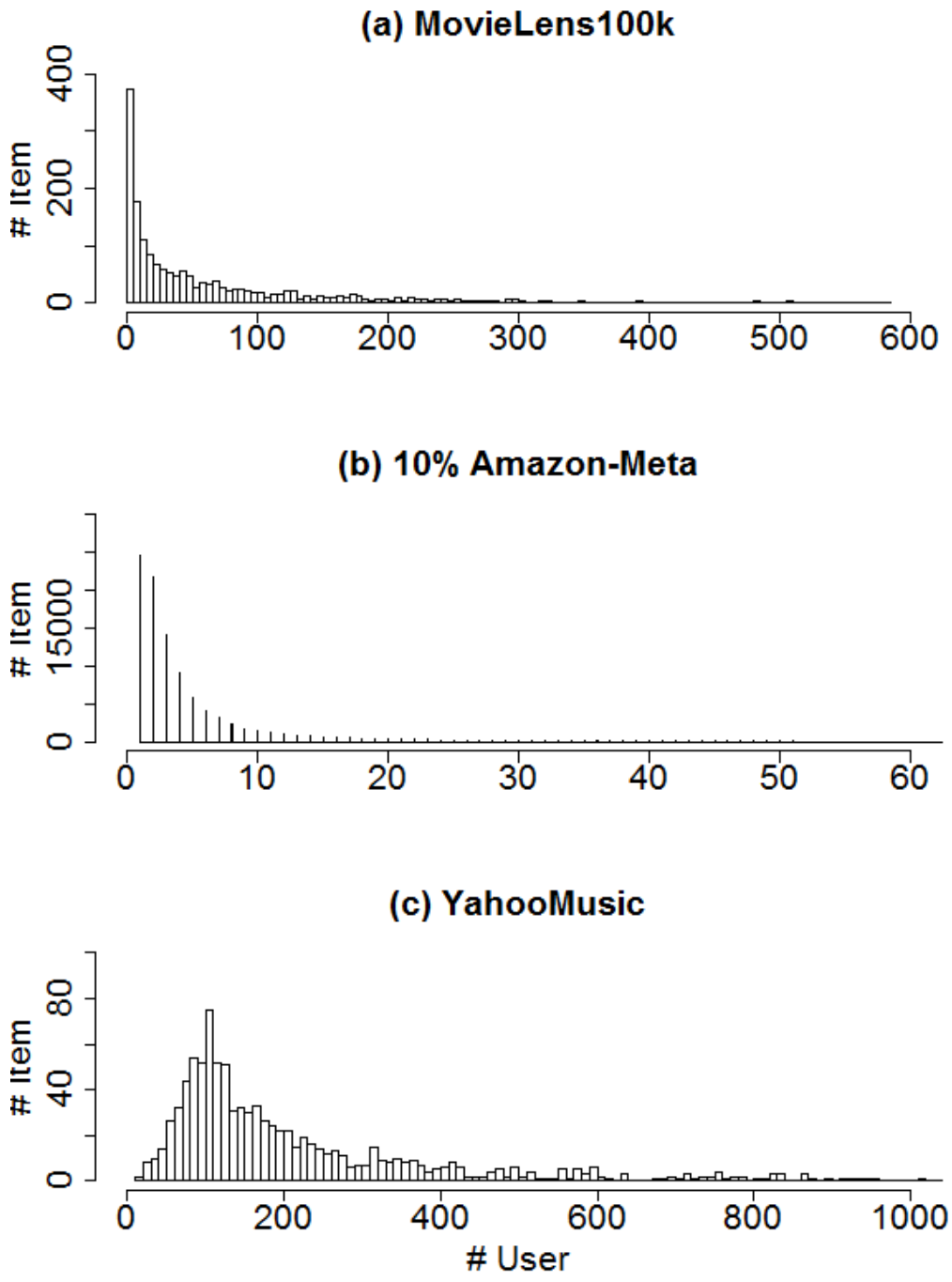


Figure 3.The number of items versus the number of users in (a) movieLens 100k, (b) 10% Amazon-Meta and (c) R3.Yahoo!Music dataset, respectively.

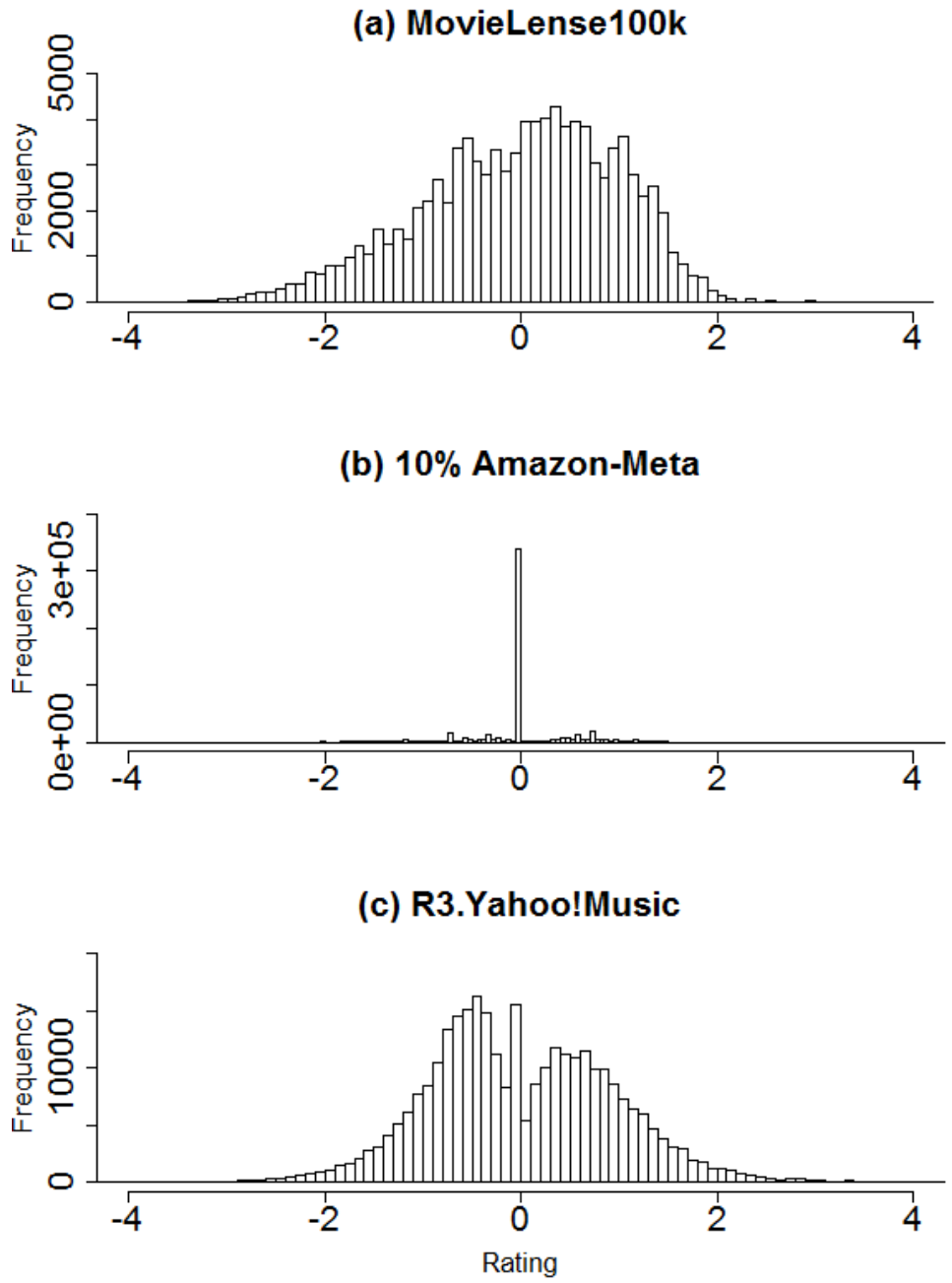


Figure 4. Normal Z-score plot of ratings for (a) movieLens 100k, (b) 10% Amazon-Meta and (c) R3.Yahoo!Music dataset, respectively.

Chapter 4

Baseline Estimates

Some users tend to give higher ratings than others toward all items. Some items have higher chance to obtain higher ratings than other items. A baseline estimate for an unknown rating r_{ui} of user u to item i, is denoted by b_{ui}, and it is defined as [7]:

$$b_{ui} = \mu + b_u + b_i \ . \tag{1}$$

Where μ represents overall average ratings for the whole dataset; u and i represent user and item, respectively. b_u and b_i are the average deviations of user u and item i from the average, respectively. The μ of the movieLens100k, 10% Amazon-Meta and R3.Yahoo!Music datasets are 3.53, 4.16 and 2.89, respectively, as shown in Table 1.

In order to estimate b_u and b_i, the least squares problem can be solved on a training set:

$$\min_{b_u,b_i} \{\sum_{(u,i)\in\mathcal{K}}(r_{ui} - \mu - b_u - b_i)^2 + \lambda_1(\sum_u b_u^2 + \sum_i b_i^2)\}. \tag{2}$$

In equation (2), for a training set $\mathcal{K} = \{(u,i)|r_{ui}\ is\ known\}$ is a set notation which has all known ratings in the past. Here the first term $\sum_{(u,i)\in\mathcal{K}}(r_{ui} - \mu - b_u - b_i)^2$ is to use to find $b_u's$ and $b_i's$ that fit the given rating r_{ui}. The regularizing term $\lambda_1(\sum_u b_u^2 + \sum_i b_i^2)$ is used to avoid overfitting by penalizing the magnitudes of the parameters with a small λ_1. Once we train the data to find estimates of b_u and b_i, we could use them to predict unknown ratings.

In recommender systems, baseline estimates are often used to combine with other models such as neighborhood and SVD methods. This is similar to a regression

problem. Baseline estimates were used as one of the predictors to explain the part of the contribution to the response variable, while the residues need to be modeled by involving other factors such as neighborhood and SVD models so that people can further reduce the residue.

Chapter 5

K-Nearest-Neighborhood Method and Experimental Results

Collaborative filtering relies on a user's history such as previous transactions, product ratings etc.. Collaborative filtering has two major methods, one is the neighborhood methods and another one is latent factor models. Neighborhood methods compute the relationships between items or between users. Both neighborhood and latent factorization address different levels of structure in the data, and none of them is optimal on its own. Neighborhood models are most effective at detecting localized relationships, but cannot capture weak signals. Latent factor models are effective at estimating overall structure of most or all items, but it is poor at detecting strong associations [7].

The Neighborhood method can be used within items, called item-oriented neighborhood methods, or within users, called user-oriented neighborhood methods. All earlier collaborative filtering systems used user-oriented neighborhood methods. They estimate unknown ratings based on recorded ratings of like-minded users. Later the item-oriented approach becomes popular. It is very similar to the user-oriented one by using the known ratings made by the same user on similar items. The item-oriented approach became more in favor because users are familiar with items previously preferred by them, but they do not know those like-minded users.

The neighborhood method can use different similarity measurements. Frequently, it is based on the Pearson correlation coefficient, ρ_{ij}, which measures the tendency of users to rate items i and j similarly. Sometimes, cosine similarity is also

applied. If the data are centered, both the Pearson correlation coefficient and cosine similarity should have the same results.

For the user-oriented approach, we calculate the Pearson correlation coefficient between user u and v as:

$$\rho_{uv} = \frac{\sum_{i \epsilon N}(r_{ui} - r_u)(r_{vi} - r_v)}{\sqrt{\sum_{i \epsilon N}(r_{ui} - r_u)^2 \sum_{i \epsilon N}(r_{vi} - r_v)^2}}, \quad (3)$$

where N is the set of items both user u and v have rated. r_u and r_v are the average ratings of user u and user v for the items in set N.

For the item-oriented approach, we calculate the Pearson correlation coefficient between item i and j similarly as:

$$\rho_{ij} = \frac{\sum_{u,v \epsilon M}(r_{ui} - r_i)(r_{vi} - r_j)}{\sqrt{\sum_{u \epsilon M}(r_{ui} - r_i)^2 \sum_{v \epsilon M}(r_{vi} - r_j)^2}}, \quad (4)$$

where M is the set of users both item i and j are rated by. r_i and r_j are the average ratings of item i and item j for the users in set M.

Since the user-oriented approach is analogous to the item-oriented approach, we will focus on the item-oriented approach. The same principles can be applied to the user-oriented approach.

If we want to predict the rating of user u to the item i, first we calculate similarity between item i (unrated by user u) and many other items, then we look at the items which user u already rated, then among the user u rated items, we choose top K items which have top K similarity, which is the so-called item-oriented K-Nearest

Neighborhood methods. We denote such a K-neighborhood set by $S^k_{(i;u)}$. Here the rating of r_{ui} is unknown, but we estimate it by using the following equation:

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{j \epsilon S^k_{(i;u)}} \rho_{ij(r_{uj} - b_{uj})}}{\sum_{j \epsilon S^k_{(i;u)}} \rho_{ij}}. \quad (5)$$

Root mean squared error (RMSE) is one of the measurements to evaluate the difference between the predicted rating value and the true rating value. RMSE is calculated by:

$$RMSE = \sqrt{\frac{\sum_{(u,i) \in \kappa} (\hat{r}_{ui} - r_{ui})^2}{|\kappa|}}. \quad (6)$$

In equation (6), κ has the same definition as in equation (2). And $|\kappa|$ is the cardinality of set κ.

Table 2 lists the experimental results by using both user-oriented and item-oriented neighborhood approaches, respectively, with k=5, 10, 15 and 20. Here Root Mean Squared Error (RMSE) is used to indicate the quality of predictions. According to Table 2, we have the following observations:

(1) With increasing K, both tables show decreasing RMSE.
(2) Given other conditions fixed, the RMSE from the item-oriented approach is lower than the one in user-oriented approach.

Table 3 summarizes the experimental results of all three datasets. From these results, we see that the K-Nearest Neighborhood method has better RMSE results on the 10% Amazon-Meta dataset. By using Pearson similarity, RMSE is slightly below 0.9. It achieved 0.956 on the MovieLens100k dataset. For the R3.Yahoo!Music dataset, the RMSE achieved by using K-Nearest neighborhood method is 1.224.

Table 2. RMSE of applying user-oriented KNN on the MovieLens100k dataset with a number of neighbors K=5, 10, 15, 20, 30. These mean values and the sample standard deviation are calculated from 10 measurements at each parameter setting.

K	RMSE User-oriented KNN	RMSE Item –oriented KNN
5	1.050 ± 0.005	1.029 ± 0.006
10	0.991 ± 0.006	0.990 ± 0.007
15	0.975 ± 0.005	0.970 ± 0.005
20	0.962 ± 0.006	0.964 ± 0.003
30	0.954 ± 0.005	0.956 ± 0.003

Table 3. RMSE of applying item-oriented KNN on MovieLens100k, 10% Amazon-Meta and R3.Yahoo!Music datasets with a number of neighbors K=30. These mean values and the sample standard deviation are calculated from 5 measurements at each parameter setting except MovieLens100k dataset, which is from Table 2. Item-oriented KNN with Pearson similarity was adopted.

Datasets	RMSE
MovieLens100k	0.956 ± 0.003
10% Amazon-Meta	0.899 ± 0.001
R3.Yahoo!Music	1.228 ± 0.004

Chapter 6

Latent Factor Models and Experimental Results

Latent factor models are used to explain the observed ratings by characterizing both items and users on some latent features which can be inferred from the rating patterns. Latent factor sometimes is also called latent variables that cannot be directly observed. They can be inferred from other observed variables. For example, in the movies recommender system, the latent features could be comedy versus drama, amount of action or children topics etc.. Latent factor models contains examples from neural networks and SVD, etc.. In this Chapter, we will focus on the SVD methods.

6.1 Introduction of SVD

In our three datasets, these user-item rating matrices are rectangular matrices. If A is an arbitrary real valued $m \times n$ matrix, there exists a unique singular value decomposition [8]. Its SVD is given by

$$A = UDV^T \tag{7}$$

where U is an $m \times m$ matrix with m unique orthonormal eigenvectors of AA^T forming its columns, and V is a $n \times n$ matrix with n unique orthonormal eigenvectors of A^TA forming its columns; D is an $m \times n$ matrix with descending ordered nonnegative diagonal entries. And the diagonal values are called singular values of A. It is expressed as

$$\lambda_j = \sqrt{eigen\ value\ (AA^T)} = \sqrt{eigen\ value\ (A^T A)},\ \ j = 1, 2, \ldots \min(m, n) \tag{8}$$

By the Eckart-Young Theorem [9], matrix A can be approximated by

$$A^{(k)} = U^{(k)} D^{(k)} (V^{(k)})^T. \tag{9}$$

In equation (9), $0 < k \leq \min(m, n)$, $U^{(k)}$ and $V^{(k)}$ are matrices formed by the first k columns of matrix U and V, respectively. So the dimension of $U^{(k)}$ and $V^{(k)}$ are $m \times k$ and $k \times n$, respectively. And $D^{(k)}$ is the upper left $k \times k$ block of matrix D. Therefore the matrix A can be approximated as:

$$A^{(k)} = p^T \cdot q \tag{10}$$

In equation (10), the dimension of p is $k \times m$ and the dimension of q is $k \times n$. For user-item rating matrix A, usually, p is called user preference matrix and q is called item aspect matrix. In the movieLens100k case, we assume that for a rating in the rating matrix A, it is viewed as the sum of user's preferences about the various aspects of a movie over a smaller number of parameters k. Therefore, we use K features to describe a movie, and p is with K preferences per user to describe how much the user prefers each aspect of a movie.

As mentioned in the part on baseline estimates, rating can be partially explained by user and item bias, but only baseline estimates are not enough to cover all contributions to the rating prediction, and we also need to involve more models. Here, in order to apply the SVD method to a sparse rating matrix, the first step is to remove the baseline estimator: b_{ui}, so the rating of user u to item i can be estimated as:

$$\hat{r}_{ui} = b_{ui} + {p_u}^T \cdot q_i. \tag{11}$$

In equation (11), $p_u \in \mathbb{R}^K$ is the user u factor vector and $q_i \in \mathbb{R}^k$ is the item i factor vector.

The conventional SVD is undefined when dealing with sparse matrices. Moreover, it is highly prone to overfitting when dealing with a highly sparse matrix. Simon Funk directly model on the observed ratings and avoid overfitting by using a regularized model such as [7, 10]:

$$\min_{p_u,q_i,b_u,b_i}\left(\sum_{(u,i)\in\kappa}(r_{ui} - b_{ui} - {p_u}^T \cdot q_i)^2 + \lambda_3\left(\left|\left|p_u\right|\right|^2 + \left|\left|q_i\right|\right|^2 + b_u^2 + b_i^2\right).\right. \quad (12)$$

A simple gradient descent technique is applied to solve equation (12). So far we have introduced the famous Funk SVD model [10].

Sometime, user may only browser items, they may not buy it or they may buy some products but they may not give explicit ratings. In order to take implicit information into consideration, Koren developed an extension of the Funk SVD method called asymmetric SVD [7]. In our three datasets, whether the user rated the item or not is used as implicit information no matter how they rated them (low or high). Therefore the implicit information matrix is a binary matrix with entry equal to 1 if they rated, 0 otherwise. Asymmetric SVD can be expressed as:

$$\hat{r}_{ui} = b_{ui} + |R(u)|^{-\frac{1}{2}} {q_i}^T\left(\sum_{j\in R(u)}\left(r_{uj} - b_{uj}\right)q_j + \sum_{j\in R(u)} y_j\right). \quad (13)$$

In equation (13), $|R(u)|^{-\frac{1}{2}}\left(\sum_{j\in R(u)}\left(r_{uj} - b_{uj}\right)q_j + \sum_{j\in R(u)} y_j\right)$ is used to replace the user preference vector p_u. $q_i, q_j, y_j \in \mathbb{R}^k$, $q_j, y_j$are the explicit and implicit vector of item j, respectively. $R(u)$ represents the set of items rated by user u. This model brings several benefits compared to the Funk SVD model:

(1) Fewer parameters. Usually the number of users is greater than the number of items.
(2) This new model can handle a new user situation easily as soon as the new user has information input, since there is no user information in the model involved.
(3) Efficient integration of implicit feedback.

Analogy to the situation of conventional SVD dealing with sparse matrices. Stochastic gradient descent and regularity were applied as shown in equation (14):

$$\min_{q_i,q_j,y_j,b_u,b_i}\{\sum_{(u,i)\in\kappa}\left(r_{ui}-\mu-b_u-b_i-|R(u)|^{-\frac{1}{2}}\,{q_i}^T\left(\sum_{j\in R(u)}(r_{uj}-b_{uj})q_j+\sum_{j\in R(u)}y_j\right)\right)^2+\lambda_5(\sum_{j\in R(u)}\left||q_j|\right|^2+\sum_{j\in R(u)}\left||y_j|\right|^2+\left||q_i|\right|^2+b_u^2+b_i^2)\}. \tag{14}$$

6.2 Experimental Results

In order to obtain optimal results, we need to first fine-tune several parameters, including the regularization coefficient λ, the learning rate γ and the number of features k. Table 5 gives detailed information about how we tune the regularization coefficient λ on the MovieLens100k dataset. The results were based on the 7 replicates with the randomly 5-fold cross-validation method.

Table 4 shows that for the MovieLens100k dataset, if we let the regularization coefficient be less than 0.015, both RMSE and variation are large according to the 7 replicates measurements. Figure 5 shows the plot of mean values of RMSE vs. the regularization coefficient. When the regularization coefficient is changed from 0.00001 to 0.015, RMSE drops dramatically then remains stable for a regularization coefficient change from 0.015 to 0.2; after 0.2, RMSE increases slowly with increasing regularization coefficient. In order to maintain the performance, we set λ equal to 0.02. These measurements are done with the learning rating fixed at 0.002 and number of features K=100.

Table 5 lists the RMSE values when we keep changing the learning rates in the columns and the corresponding number of iterations. From the table we can see when the learning rate equals 0.004, the mean of RMSE is the lowest. Therefore we choose 0.004 as MovieLens100k's gradient descent learning rate.

A similar strategy has also been applied to the Yahoo Music and 10% Amazon-Meta Dataset. We picked regularization coefficient=0.02 and learning rate=0.004 for Yahoo Music and regularization coefficient=0.02 and learning rate=0.007 for 10% Amazon-Meta dataset.

Table 6 demonstrates the change in RMSEs of MovieLens100k and R3.Yahoo!Music datasets with the increasing number of features. The two datasets show that the number of features in the asymmetric SVD approach does not affect the RMSE very much. Increasing the number of features also increases the calculation time. Since the 10% Amazon-Meta dataset is quite large, we skipped this step on it.

Table 7 summarizes the RMSE of the three datasets after applying the asymmetric SVD methods at their optimized conditions. From this table we see that

MovieLens100k has the lowest RMSE while both the 10% Amazon-Meta and R3.Yahoo!Music have RMSE greater than 1.

Table 4. RMSE of applying the Asymmetric SVD method on MovieLens100k dataset with fixed number of features equal to 100 and learning rate equals .002. Mean and standard deviation are calculated based on 7 replicates.

Regulation coefficient	RMSE	RMSE	RMSE	RMSE	RMSE	RMSE	RMSE	Mean	Standard Deviation
0.00001	0.938	0.946	0.946	0.942	0.949	0.941	0.941	0.943	0.004
0.0001	1.394	0.942	0.942	0.933	1.408	0.948	0.942	1.073	0.224
0.0005	1.389	0.938	0.938	1.378	1.392	1.371	0.945	1.193	0.236
0.005	1.248	1.197	1.222	0.942	1.225	0.938	0.942	1.102	0.152
0.01	0.939	0.941	0.934	0.932	1.056	0.941	0.938	0.954	0.045
0.015	0.932	0.939	0.938	0.942	0.935	0.939	0.939	0.938	0.003
0.02	0.939	0.948	0.946	0.936	0.941	0.939	0.936	0.941	0.005
0.03	0.936	0.935	0.941	0.946	0.943	0.941	0.941	0.94	0.004
0.035	0.944	0.928	0.941	0.942	0.939	0.94	0.939	0.939	0.005
0.04	0.944	0.936	0.94	0.931	0.94	0.933	0.939	0.938	0.005
0.05	0.943	0.942	0.942	0.941	0.938	0.93	0.941	0.94	0.005
0.06	0.939	0.942	0.935	0.937	0.937	0.942	0.935	0.938	0.003
0.07	0.944	0.935	0.929	0.938	0.941	0.941	0.95	0.94	0.007
0.08	0.948	0.935	0.94	0.945	0.942	0.939	0.942	0.942	0.004
0.09	0.946	0.943	0.943	0.937	0.946	0.932	0.94	0.941	0.005
0.1	0.947	0.936	0.937	0.946	0.939	0.937	0.935	0.94	0.005
0.2	0.941	0.945	0.943	0.947	0.947	0.946	0.947	0.945	0.002
0.3	0.954	0.949	0.954	0.948	0.955	0.947	0.951	0.951	0.003
0.4	0.958	0.953	0.959	0.959	0.955	0.952	0.957	0.956	0.003
0.5	0.958	0.959	0.961	0.96	0.968	0.959	0.959	0.961	0.003
0.6	0.967	0.957	0.965	0.965	0.962	0.96	0.964	0.963	0.003
0.8	0.974	0.977	0.975	0.973	0.975	0.968	0.973	0.974	0.003
1	0.986	0.991	0.981	0.992	0.988	0.987	0.984	0.987	0.004

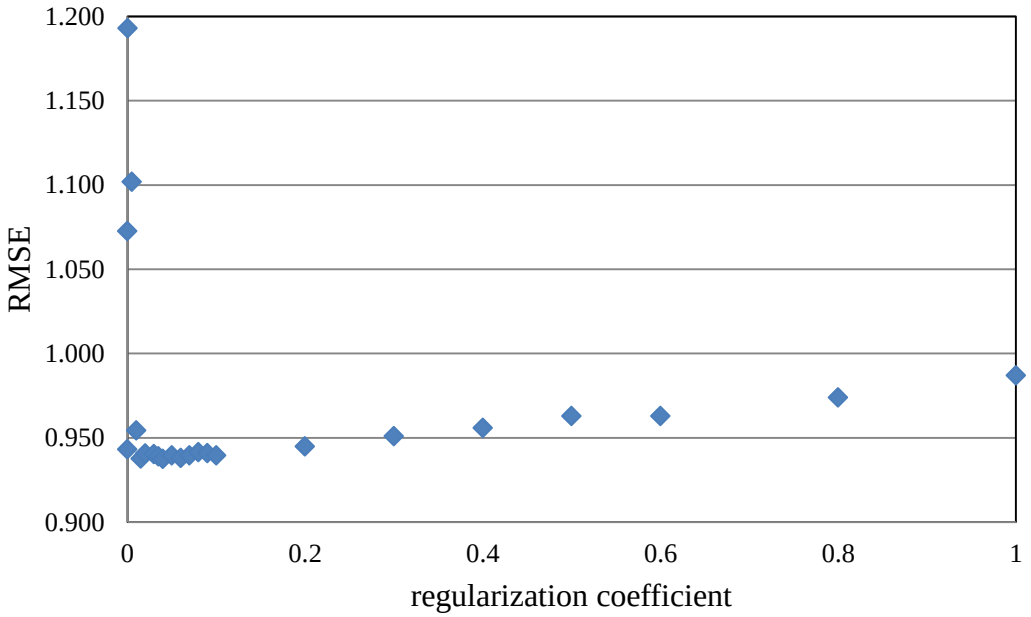


Figure 2. the scatter plot of RMSE vs. regularization coefficient on MovieLens100k dataset.

Table 5. RMSE of tuning learning rating based on the changes of number of iterations on MovieLens100k dataset with regularization coefficient=0.02 and K=100.

Iteration \ Learning Rates	0.0005	0.001	0.004	0.005
100	0.946	0.946	0.939	0.939
150	0.939	0.943	0.94	0.943
500	0.936	0.944	0.936	0.941
250	0.944	0.941	0.945	0.958
500	0.947	0.941	0.943	0.939
Mean	0.942	0.943	0.941	0.944
Standard deviation	0.005	0.002	0.004	0.008

Table 6. Tuning number of features on MovieLens100k and R3.Yahoo!Music dataset with regularization coefficient=0.02 and learning rate=0.004.

K	RMSE MovieLens100k	RMSE R3.Yahoo!Music
50	0.942 ± 0.004	1.221 ± 0.003
100	0.938 ± 0.004	1.219 ± 0.003
200	0.941 ± 0.006	1.218 ± 0.002
300	0.945 ± 0.004	1.218 ± 0.001
400	0.944 ± 0.004	1.218 ± 0.003
500	0.939 ± 0.005	1.217 ± 0.003

Table 7. RMSE of applying the Asymmetric SVD to MovieLens100k, 10% Amazon-Meta and R3.Yahoo!Music datasets. These mean values and the sample standard deviation are calculated from 5 measurements at each parameter setting.

	RMSE
MovieLens100k	0.938 ± 0.004
10% Amazon-Meta	1.079 ± 0.004
R3.Yahoo!Music	1.219 ± 0.003

Chapter 7

Restricted Boltzmann Machine and Experimental Results

The Boltzmann machine is a stochastic neural network. It is used in dimensionality reduction, classification, collaborative filtering, feature learning. In general, a Boltzmann machine is computationally slow. [11]

7.1. Introduction of Restricted Boltzmann Machine (RBM)

Restricted Boltzmann machine (RBM) is a special type of Boltzmann machine that impose restrictions on the network to speed up the learning process. Figure 6 demonstrates schematically the differences between the Boltzmann machine and RBM.

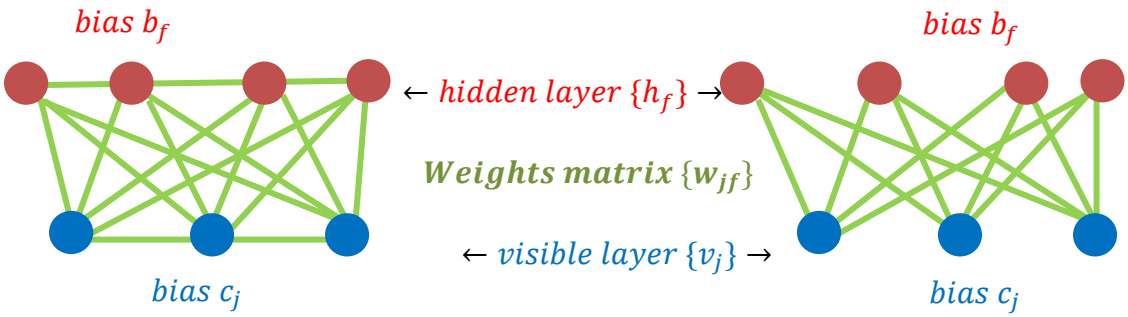


Figure 3. Schema of the Boltzmann machine on left and restricted Boltzmann machine on right. Weight matrix is not associated with users. It is only associated with the connections between the units in the two layers.

There are no connections between units within either layer in RBM. Therefore, the units within the same layer are independent of each other. All units of one layer are connected to all units in another layer. In order to be an RBM, these connections are required to be bidirectional and symmetric. [11] The RBM is used to learn important features of an unknown target distribution based on samples from this target distribution. Training a restricted Boltzmann machine means tuning the parameters such that the probability distribution fits the training data as well as possible.

The standard type of RBM has binary-valued hidden and visible units and consists of a matrix of weights $W = w_{jf}$ associated with the connection of the hidden unit $h_f$and the visible unit v_j. There are also biases c_j for the visible unit and b_f for the hidden unit. The visible units form the first layer to represent the observations and the hidden units constitute the second layers to model the dependencies between the components of observations and they can be viewed as non-linear feature detectors. The connection between v_js and h_fs is called a configuration. Assume there are *N* visible units (in the recommender systems, *N* is the total number of items) and *F* hidden units. The energy [11] of a configuration $(\vec{v}, \vec{h})$ is defined as

$$E(\vec{v}, \vec{h}) = -\sum_{j=1}^{N} c_j v_j - \sum_{f=1}^{F} b_f h_f - \sum_{j=1}^{N} \sum_{f=1}^{F} h_f w_{jf} v_j. \quad (15)$$

In RBM, the probability distributions over hidden and visible vectors are defined in terms of the energy function as:

$$P(\vec{v}, \vec{h}) = \frac{e^{-E(\vec{v},\vec{h})}}{\sum_{\vec{v}',\vec{h}'} e^{-E(\vec{v}',\vec{h}')}} = \frac{e^{-E(\vec{v},\vec{h})}}{Z}. \quad (16)$$

In equation (16), $Z = \sum_{\vec{v}',\vec{h}'} e^{-E(\vec{v}',\vec{h}')}$, called the partition function associated with total energy. And the vector pair $(\vec{v}, \vec{h})$ represents one user's item-rating relationship in our

recommender situation. Since the units in the same layers are independent of each other, the marginal distribution of the visible vector can be expressed as:

$$P(\vec{v}) = \frac{\sum_{\vec{h}} e^{-E(\vec{v},\vec{h})}}{Z} = \frac{1}{Z}\sum_{h_1}\sum_{h_2}\dots\sum_{h_F} e^{\sum_{j=1}^{N} c_j v_j}\prod_{f=1}^{F} e^{h_f\left(b_f+\sum_{j=1}^{N} w_{jf}v_j\right)}$$

$$= \frac{e^{\sum_{j=1}^{N} c_j v_j}}{Z}\sum_{h_1} e^{h_1\left(b_1+\sum_{j=1}^{N} w_{j1}v_j\right)}\sum_{h_2} e^{h_2\left(b_2+\sum_{j=1}^{N} w_{j2}v_j\right)}\dots\sum_{h_F} e^{h_F\left(b_F+\sum_{j=1}^{N} w_{jF}v_j\right)}$$

$$= \frac{e^{\sum_{j=1}^{N} c_j v_j}}{Z}\prod_{f=1}^{F}\sum_{h_f} e^{h_f\left(b_f+\sum_{j=1}^{N} w_{jf}v_j\right)}$$

$$= \frac{1}{Z}\prod_{j=1}^{N} e^{c_j v_j}\prod_{f=1}^{F}\sum_{h_f} e^{h_f\left(b_f+\sum_{j=1}^{N} w_{jf}v_j\right)}. \quad (17)$$

In the restricted Boltzmann machine, connections only exist between layers. The units either in the visible layer or in the hidden layer are independent. The conditional probability of the visible layer and conditional probability of the hidden layer can be written as

$$P\left(\vec{v}|\vec{h}\right) = \prod_{j=1}^{N} P(v_j|\vec{h}) \quad (18)$$

$$P\left(\vec{h}|\vec{v}\right) = \prod_{f=1}^{F} P(h_f|\vec{v}). \quad (19)$$

Both visible units and hidden units are binary-valued units. When they are 1, we say they are in activated status, if they are 0, we assume they are in deactivated status. The individual activation probabilities for the hidden units are:

$$P\left(h_f = 1\middle|\vec{v}\right) = \frac{1}{1+\exp[-(\vec{v}\, w_{*f}+b_f)]} \quad (20)$$

Here is the proof:

$$P(\vec{h}|\vec{v}) = \frac{P(\vec{h},\vec{v})}{\sum_{\vec{h'}} P\left(\vec{h'},\vec{v}\right)} = \frac{\exp\left(\vec{v}^T w\vec{h} + \vec{c}^T\vec{v} + \vec{b}^T\vec{h}\right)}{\sum_{\vec{h'}\in\{0,1\}^F} \exp\left(\vec{v}^T w\vec{h'} + \vec{c}^T\vec{v} + \vec{b}^T\vec{h'}\right)}$$

$$= \frac{\exp\left(\sum_f (\vec{v}^T w_{*f} h_f + b_f h_f\right)}{\sum_{h'_1\in\{0,1\}} \dots \sum_{h'_f\in\{0,1\}} \dots \sum_{h'_F\in\{0,1\}} \prod_f \exp\left(\vec{v}\, w_{*f} h'_f + b_f h'_f\right)}$$

$$= \frac{\prod_f \exp\left(\vec{v}^T w_{*f} h_f + b_f h_f\right)}{\left(\sum_{h'_1\in\{0,1\}} \exp(\vec{v}^T\, w_{*1} h'_1 + b_1 h'_1)\right) \dots \left(\sum_{h'_F\in\{0,1\}} \exp(\vec{v}^T\, w_{*F} h'_F + b_F h'_F)\right)}$$

$$= \frac{\prod_f \exp\left(\vec{v}^T w_{*f} h_f + b_f h_f\right)}{\prod_f \left(1 + \exp\left(\vec{v}^T w_{*f} + b_f\right)\right))}$$

$$= \prod_f \frac{\exp(\vec{v}^T\, w_{*f} h_f + b_f h_f)}{1 + \exp(\vec{v}^T\, w_{*f} + b_f)}.$$

Therefore,

$$P\left(h_f = 1|\vec{v}\right) = \frac{\exp\left(\vec{v}^T\, w_{*f} + b_f\right)}{1 + \exp\left(\vec{v}^T\, w_{*f} + b_f\right)} = \frac{1}{1 + \exp[-\left(\vec{v}^T\, w_{*f} + b_f\right)]}.$$

Similarly, we can get

$$P\left(v_j = 1|\vec{h}\right) = \frac{\exp(w_{j*}\vec{h} + c_j)}{1+\exp(w_{j*}\vec{h} + c_j)} = \frac{1}{1+\exp[-(w_{j*}\vec{h} + c_j)]}. \quad (21)$$

7.2. RBM with Integer Valued Visible Units

Although the standard RBM has only binary units and symmetric connections, it is not required that units to be binary. In order to apply RBM to our three recommender system datasets, the units in the visible layer cannot be only binary values since there are K rates from 1 to K indicating dislike to strongly like. In our case, K=5. So the modified RBM is shown in Figure 7.

Comparing this RBM extension to the standard RBM, the single unit of v_j becomes a vector denoted as $\vec{v_j} = \{v_j^1, v_j^2, \dots, v_j^K\}$, . For our three datasets, K=5. If rating $r_{ij} = k$, then $v_j^k = 1$ and others are zero. Therefore according to equation (20) and (21), we have [11]

$$E(V, \vec{h}) = -\sum_j \sum_k c_j^k v_j^k - \sum_f b_f h_f - \sum_j \sum_f \sum_k h_f w_{jf}^k v_j^k \quad (22)$$

$$P(h_f = 1|V) = \frac{1}{1+\exp[-(b_f + \sum_{j=1}^{F} \sum_{k=1}^{K} w_{jf} v_j^k)]} \quad (23)$$

$$P(v_j^k = 1|\vec{h}) = \frac{\exp(w_{j*}\vec{h} + c_j^k)}{\sum_{k=1}^{K} \exp(w_{j*}\vec{h} + c_j^k)}. \quad (24)$$

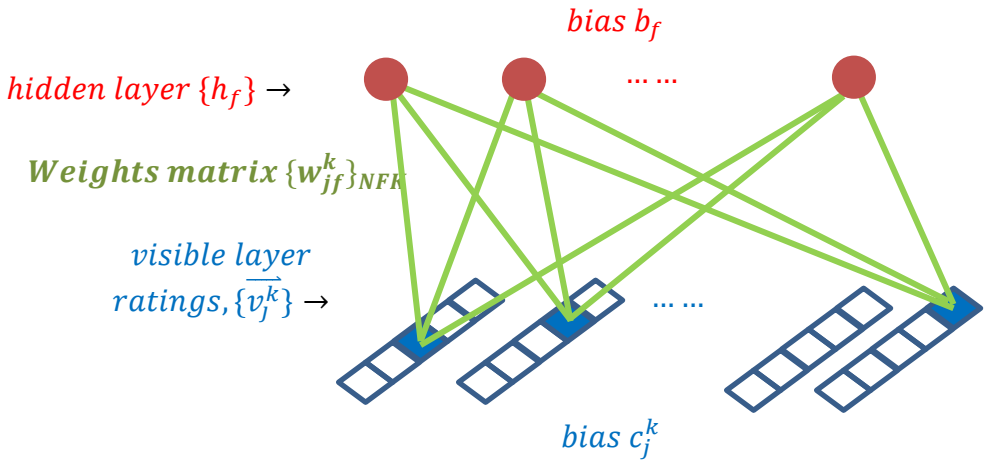


Figure 4. Schema of the Restricted Boltzmann machine for a single user with visible units having 5 binary cells, representing 5 possible ratings. If $\boldsymbol{rating = k} \in \{\mathbf{1}, \dots, \mathbf{5}\}$, then only the $\boldsymbol{k^{th}}$ cell (shadowed cells) has value 1 others are zero. In this model, each of the user's hidden units is connected to every visible unit that represents a rating made by that user. In this model, weight matrix is not associated with users, it is only associated with the connections between the units in the two layers.

In equation (23), the conditional distributions of the i^{th} user's hidden features variables, given that user's observed ratings $v_j^k = 1$, are modeled as conditionally

independent Bernoulli variables. In equation (24), the conditional distribution of the i^{th} user's observed rating v_j^k, given that user's hidden features vector $\vec{h}$, is modeled as a multinomial distribution.

7.3. Learning of RBM

The learning process of RBM is to fit the training dataset by adjusting the parameter θ, which is used to represent the weight matrix and the biases for units in both the visible layer and the hidden layer. [11] We can write the unknown distribution as: $p(V|\theta)$, and the training set as: S. We need to assume the data samples are independent and identically distributed (i.i.d.). The parameter updates are obtained by performing gradient ascent in the log-likelihood as follows:

$$\ln \mathcal{L}_V(\theta) = \ln p_\theta(V) = \ln \frac{1}{Z} \sum_{\vec{h}} e^{-E(V,\vec{h})} = \ln \sum_{\vec{h}} e^{-E(V,\vec{h})} - \ln \sum_{V,\vec{h}} e^{-E(V,\vec{h})} \quad (25)$$

$$\theta^{(t+1)} = \theta^{(t)} + \eta \frac{\partial \ln \mathcal{L}(\theta^{(t)}|V)}{\partial \theta^{(t)}}. \quad (26)$$

In equation (26), η is the gradient learning rate, which is always a small positive number; $\theta^{(t)}$ represents the fitting results after t iterations. Now by the definition of log-likelihood function in equation (25), we can find the gradient of likelihood w.r.t. parameters as

$$\frac{\partial \ln \mathcal{L}_V(\theta)}{\partial \theta} = \frac{\partial}{\partial \theta}\left(\ln \sum_{\vec{h}} e^{-E(V,\vec{h})}\right) - \frac{\partial}{\partial \theta}\left(\ln \sum_{V,\vec{h}} e^{-E(V,\vec{h})}\right)$$

$$= -\frac{1}{\sum_{\vec{h}} e^{-E(V,\vec{h})}} \sum_{\vec{h}} e^{-E(V,\vec{h})} \frac{\partial E(V,\vec{h})}{\partial \theta} + \frac{1}{\sum_{V,\vec{h}} e^{-E(V,\vec{h})}} \sum_{V,\vec{h}} e^{-E(V,\vec{h})} \frac{\partial E(V,\vec{h})}{\partial \theta}$$

$$= -\sum_{\vec{h}} p(\vec{h}|V)\frac{\partial E(V,\vec{h})}{\partial \theta} + \sum_{V,\vec{h}} p(V,\vec{h})\frac{\partial E(V,\vec{h})}{\partial \theta}$$

$$= -<\frac{\partial E(V,\vec{h})}{\partial \theta}>_{p(\vec{h}|V)} + <\frac{\partial E(V,\vec{h})}{\partial \theta}>_{p(V,\vec{h})}$$

$$= -<\frac{\partial E(V,\vec{h})}{\partial \theta}>_{data} + <\frac{\partial E(V,\vec{h})}{\partial \theta}>_{model}. \quad (27)$$

From the above equation, the gradient of the likelihood is the difference between two expected values of the energy function under the model distribution and under the conditional distribution of the hidden variables given the training example.

By equations (22) and (27), we can obtain the gradient of the likelihood w.r.t. weight matrix

$$\frac{\partial \ln \mathcal{L}_V(\theta)}{\partial w_{jf}} = -\sum_{\vec{h}} p(\vec{h}|V)\frac{\partial E(V,\vec{h})}{\partial w_{jf}} + \sum_{V,\vec{h}} p(V,\vec{h})\frac{\partial E(V,\vec{h})}{\partial w_{jf}}$$

$$= \sum_{\vec{h}} p(\vec{h}|V) h_f v_j - \sum_{V} p(V) \sum_{\vec{h}} p(\vec{h}|V)\, h_f v_j$$

$$= p(h_f = 1|V) v_j - \textstyle\sum_V p(V)\, p(h_f = 1|V) v_j\,. \quad (28)$$

For the mean of this derivative over a training set S, we have:

$$\frac{1}{|S|}\sum_{V\in S}\frac{\partial \ln \mathcal{L}_V(\theta)}{\partial w_{jf}} = \frac{1}{|S|}\sum_{V\in S}\left[-\mathrm{E}_{p(\vec{h}|V)}\left[\frac{\partial E(V,\vec{h})}{\partial w_{jf}}\right] + \mathrm{E}_{p(V,\vec{h})}\left[\frac{\partial E(V,\vec{h})}{\partial w_{jf}}\right]\right]$$

$$= \frac{1}{|S|}\sum_{V\in S}\left[\mathrm{E}_{p(\vec{h}|V)}[h_f v_j] - \mathrm{E}_{p(V,\vec{h})}[h_f v_j]\right]$$

$$= < h_f v_j >_{p(\vec{h}|V)q(V)} - < h_f v_j >_{p(V,\vec{h})}. \quad (29)$$

Here $q(V)$ represents the data distribution. We can further simplify equation (29) to:

$$\textstyle\sum_{V\in S}\frac{\partial \ln \mathcal{L}(\theta|V)}{\partial w_{jf}} \propto < h_f v_j >_{data} - < h_f v_j >_{model}. \quad (30)$$

Similarly, we can have the gradient of the likelihood w.r.t. the bias term c_j of the j^{th} visible variable

$$\frac{\partial \ln \mathcal{L}_V(\theta)}{\partial c_j} = v_j - \sum_V p(V) v_j \propto < v_j >_{data} - < v_j >_{model} \quad (31)$$

and w.r.t. the bias term of b_f of the f^{th} hidden variable

$$\frac{\partial \ln \mathcal{L}_V(\theta)}{\partial b_f} = p\left(h_f = 1 \middle| V\right) - \sum_V p(V)\, p\left(h_f = 1 \middle| V\right) = < h_f >_{data} - < h_f >_{model} \quad (32)$$

The second term in equations (30-32) is the sum over all values of the visible variables. In order to avoid the computational burden, contrastive divergence (CD) algorithms have been applied to approximate the RBM log-likelihood gradient if given some training data [11].

7.4. Experimental Results of Applying RBM on the three datasets

In this experiment, the RBM package in Graphchi [12] has been applied to the three datasets. Before the experiment, we applied the RBM package to the sample Netflix datasets: smallnetflix_mm (used as training set) and smallnetflix_mme (used as validation set) to verify the RBM package and we obtained RMSE 0.911 with learning rate 0.015, regularization coefficient 0.01, the number of iterations 30. Table 8 lists the RMSE results for the three datasets. For the MovieLens100k dataset, RMSE 0.972 was obtained with learning rate 0.02 and regularization coefficient 0.02 and the number of hidden units 100; for the 10% Amazon-Meta, we obtained RMSE 1.570 with learning rate 0.007, regularization coefficient 0.02 and 100 hidden units. For the R3.Yahoo!Music, we got RMSE 1.238 with learning rate 0.02, regularization coefficient 0.02 and 100 hidden units.

Table 8. RMSE of applying the restricted Boltzmann machine to MovieLens100k, 10% Amazon-Meta and R3.Yahoo!Music datasets. These mean values and the sample standard deviation are calculated from 5 measurements at each parameter setting.

	RMSE
MovieLens100k	0.972 ± 0.010
10% Amazon-Meta	1.570 ± 0.004
R3.Yahoo!Music	1.193 ± 0.002

Chapter 8

Conclusion

By comparing the RMSE obtained from the three methods on the three datasets, the recommender techniques are very data oriented. Since Amazon-Meta was collected in a friend recommendation setting, the KNN methods obtained the lowest RMSE prediction. Since the users in Amazon-Meta are correlated, they are not i.i.d., therefore RBM produced the highest RMSE among all recommender methods and among all the three datasets. For movie recommendation, asymmetric SVD gives the lowest RMSE, while for the music dataset, RBM has slightly better RMSE.

Table 9. Comparison of the RMSE of applying KNN, asymmetric SVD and RBM to MovieLens100k, 10% Amazon-Meta and R3.Yahoo!Music datasets. These results have been displayed in Table 3, 7, 8 respectively.

	KNN	Asymmetric SVD	RBM
MovieLens100k	0.956 ± 0.003	0.938 ± 0.004	0.972 ± 0.010
10% Amazon-Meta	0.899 ± 0.001	1.079 ± 0.004	1.570 ± 0.004
R3.Yahoo!Music	1.228 ± 0.004	1.219 ± 0.003	1.193 ± 0.002

Appendix

A.

Theorem: *Let A be a complex $m \times n$ matrix. A has a singular value decomposition of the form $A = UDV^T$, where D is a uniquely determined $m \times n$ (real) diagonal matrix, U is an $m \times m$ unitary matrix, and V is an $n \times n$ unitary matrix.*

Proof [8]

(1) Observe $A^T A$ or AA^T are symmetric

(2) According to symmetric matrix properties, $A^T A$ has n pairwise orthogonal eigenvetors. Let $rank(A^T A) = r$, so $r \leq \min(m, n)$. $v_1, \dots, v_r$ be those normalized eigenvectors associated with the nonzero eigenvalues $\lambda_1, \dots, \lambda_r$ of $A^T A$

(3) $A\, A^T\, A\, v_i = A\, \lambda_i\, v_i = \lambda_i\, A\, v_i$, so $A\, v_i$ is an eigenvector of $A\, A^T$ with the same eigenvalue λ_i

(4) $\|Av_i\| = \sqrt{(Av_i)^T Av_i} = \sqrt{v_i{}^T A^T A v_i} = \sqrt{\lambda_i v_i{}^T v_i} = \sqrt{\lambda_i}$

(5) Let $u_i = Av_i/\sqrt{\lambda_i}$, then $u_1, \dots, u_r$are orthonormal vectors, then

$$u_i^T A\, v_j = (Av_i/\sqrt{\lambda_i})^T Av_j = \frac{v_i^T A^T A v_j}{\sqrt{\lambda_i}} = \frac{\lambda_j\, v_i^T\, v_j}{\sqrt{\lambda_i}} = \begin{cases} 0 & if\ i \neq j \\ \sqrt{\lambda_i} & if\ i = j \end{cases}$$

(6) Writing all into matrix format as

$$\begin{bmatrix} u_1^T \\ \vdots \\ u_r^T \end{bmatrix} A[\, v_1^T \cdots v_r^T] = diag(\sqrt{\lambda_1}, \dots, \sqrt{\lambda_r})$$

(7) Define $U = [u_1 \dots u_m]$ and $V = [v_1 \dots v_n]$ are vector, here $u_{r+1}, \dots, u_m$ are pairwise orthogonal and orthogonal to $u_1, \dots, u_r$, and $v_{r+1}, \dots, v_n$ are pairwise orthogonal and orthogonal to $v_1, \dots, v_r$. Therefore, $UU^T = VV^T = I$.

(8) Now we have $U^T AV = \mathrm{D}$, here D is a $m \times n$ matrix with the first r diagonal elements are nonzero $\sqrt{\lambda_1} \geq \sqrt{\lambda_2}$ ... $\geq \sqrt{\lambda_r} \neq 0$ and all other elements are zero.

(9) $\because\ U^T AV = \mathrm{D}$ and $UU^T = VV^T = I\ \therefore\ A = U\mathrm{D}V^T$

B.

Theorem: *Let A be a complex $m \times n$ matrix, the SVD of A is $A = UDV^T$. The best rank-k low rank approximation of A is $A^{(k)} = U^{(k)}D^{(k)}(V^{(k)})^T$, where $D^{(k)}$ is $k \times k$ (real) diagonal matrix of the left-top corner of D, $U^{(k)}$ is an $m \times k$ unitary matrix containing the first k columns of U, and V is an $n \times k$ unitary matrix containing the first k columns of V.*

Proof

http://www.caam.rice.edu/~embree/caam453/lecture18.pdf

(1) From the proof of existence of SVD in Appendix A, we have $A = UDV^T$. $rank(D) = r$. Matrix D has r nonzero diagonal elements in descending order and all other elements of D zero.

(2) $A = UDV^T = \sum_{j=1}^{r} \sqrt{\lambda_j}\, u_j v_j^T$, here $u_j v_j^T$ is rank 1 matrix

(3) Let $A^{(k)} = \sum_{j=1}^{k} \sqrt{\lambda_j}\, u_j v_j^T$, here $k = rank\left(A^{(k)}\right)$ and $k \leq r$

(4) $\therefore A - A^{(k)} = \sum_{j=1}^{r} \sqrt{\lambda_j}\, u_j v_j^T - \sum_{j=1}^{k} \sqrt{\lambda_j}\, u_j v_j^T = \sum_{j=k+1}^{r} \sqrt{\lambda_j}\, u_j v_j^T$

(5) $\therefore \sum_{j=k+1}^{r} \sqrt{\lambda_j}\, u_j v_j^T$ is SVD of $A - A^{(k)}$.

(6) $\therefore \left\|A - A^{(k)}\right\|_2 = \sqrt{\lambda_{k+1}}$

(7) $\forall\, X \in \mathbb{C}^{m \times n}$ with $rank(X) = k$, we need to prove that $\|A - X\|_2 \geq \sqrt{\lambda_{k+1}}$

(8) $\because Ker(X) \subseteq \mathbb{C}^n$ with $\dim\left(Ker(X)\right) = n - k$, also $span\{v_1, \dots, v_{k+1}\} \subseteq \mathbb{C}^n$ and $span\{v_1, \dots, v_{k+1}\}\ has\ at\ most\ k+1$ dimensions.
$\therefore Ker(X) \cap span\{v_1, \dots, v_{k+1}\} \neq \phi$

(9) Let $z = \gamma_1 v_1 + \cdots + \gamma_{k+1} v_{k+1}$ be a unit vector and
$z \in Ker(X) \cap span\{v_1, \dots, v_{k+1}\}$ and $\|z\|_2 = 1$

(10) Since $\|z\|_2 = 1$

$$\left(\sum_{j=1}^{k+1}\gamma_j v_j\right)^{*}\left(\sum_{j=1}^{k+1}\gamma_j v_j\right)=\sum_{j=1}^{k+1}{\gamma_j}^2=1$$

$$\|A-X\|_2 \geq \|(A-X)z\|_2 = \|Az\|_2 = \left\|\sum_{j=1}^{k+1}\sqrt{\lambda_j}\,u_j v_j^T z\right\|_2 = \left\|\sum_{j=1}^{k+1}\sqrt{\lambda_j}\,u_j\gamma_j\right\|_2$$

$$\geq \sqrt{\lambda_{k+1}}\left\|\sum_{j=1}^{k+1}u_j\gamma_j\right\|_2$$

$$\left\|\sum_{j=1}^{k+1}u_j\gamma_j\right\|_2 = (\sum_{j=1}^{k+1}u_j\gamma_j)^T\sum_{j=1}^{k+1}u_j\gamma_j = \sum_{j=1}^{k+1}{\gamma_j}^2=1$$

$$\therefore \|A-X\|_2 \geq \sqrt{\lambda_{k+1}}$$

Therefor $A^{(k)}$is the best rank k approximation to A.

C.

R code for doing 0.8/0.2 random data spliting

```
l1<-read.csv("./filename.csv", header=F)
splitdf<-function(dataframe,seed=NULL){
 if(!is.null(seed)) set.seed(seed)
 index<-1:nrow(dataframe)
 trainindex<-sample(index,trunc(length(index)/5))
 trainset<-dataframe[trainindex, ]
 testset<-dataframe[-trainindex, ]
 list(trainset=trainset, testset=testset)
}
splits<-splitdf(l1,seed=211)
str(splits)
lapply(splits,nrow)
lapply(splits,head)
trainingxw1<-splits$trainset
testingxw1<-splits$testset
write.table(trainingxw1,file="filename1.txt",sep=" ")
write.table(testingxw1,file="filename1.txt",sep=" ")
```

D.

Matlab Code for Converting Dataset into Sparse Matrix Market Format for Graphchi Package

```
%% written by Charles Kwong
%% initialization
clear;
clc;
quote = ""; sep = ','; escape = '\';


 %% process
 tic
 [numbers, text] = swallow_csv('averageAmazon10percent.txt', quote, sep, escape);
 data = numbers(:,1:3);
 [~,~,rowInd] = unique(data(:,1));
 [~,~,colInd] = unique(data(:,2));
 maxRowInd = max(rowInd);
 maxColInd = max(colInd);
 uDataMatrix = sparse(rowInd,colInd,data(:,3),maxRowInd,maxColInd);
 mmwrite('test.txt',uDataMatrix);
 toc
```

E.

Steps for Run KNN and Asymmetric SVD in the Recomender101 Package

(1) First download the Recommender101 from http://ls13-www.cs.uni-dortmund.de/homepage/recommender101/index.shtml

(2) In Eclipse, under ./src folder, find recommender101.properties file, in there, we can setup dataset directory by letting DataLoaderClass= org.recommender101.data.DefaultDataLoader:filename=your_file_directory |sampleNUsers=1000, here you can change sampleNUsers to any number you want to sample.

(3) Let DataSplitterClass=org.recommender101.data.DefaultDataSplitter:nbFolds=5 or any other numbers, here, we run 5-fold cross-validation in random splitting style.

(4) If we run the KNN, we let AlgorithmClasses= org.recommender101.recommender.baseline.NearestNeighbors

(5) For setting up parameters for the KNN experiment, we could find the NearestNeighbors.java file under org.recommender101.recommender.baseline directory, in this file, we could set K as nbNeighbors, itemBased=true or false, useCosineSimilarity=false or true

(6) If we run the Asymmetric SVD, we let AlgorithmClasses= org.recommender101.recommender.extensions.factorizednghbors.FactorizedNeighborhoodRecommender:lambda=0.02|gamma=0.007|iterations=100|nbFactors=500

In here, lambda is the regularization coefficient, gamma is the learning rate, “nbFactors” is the number of features. “Iterations” is the number of steps during stochastic gradient descent.

F.

Code for Run the Restricted Boltzmann Machine Experiments in Graphchi Package

./toolkits/collaborative_filtering/rbm --training=smallnetflix_mm --validation=smallnetflix_mme --minval=1 --maxval=5 --max_iter=6 --quiet=1 –D=100 –rbm_alpha=0.01 rbm_beta=0.02

In this command, D is the number of features in the hidden layer, rbm_alpha is the learning rate for stochastic gradient descent, rbm_beta is the parameter of regularization coefficient.

References

[1] J. Herlocker, J. Konstan, L. Terveen and R. J., "Evaluating Collaborative Filtering Recommender Systems," *ACM Transactions on Information Systems,* pp. 5-53, 2004.

[2] D. Goldberg, D. Nichols, B. M. Oki and D. Terry, "Using collaborative filtering to weave an information Tapestry," *Communications of the ACM,* vol. 35, no. 12, p. 61, 1992.

[3] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom and J. Riedl, "GroupLens: An Open Architecture for Collaborative Filtering of Netnews," in *Computer supported cooperative work*, New York, NY, USA, 1994.

[4] G. Linden, B. Smith and J. York, "Amazon.com Recommendations Item-to-Item Collaborative Filtering," the IEEE Computer Society, 2003.

[5] M. Ekstrand, J. Riedl and J. Konstan, "Collaborative Filtering Recommender Systems," *Foundations and Trends in Human–Computer Interaction,* pp. 81-73, 2010.

[6] D. Jannach, L. Lerche, F. Gedikli and G. Bonnin, "What recommenders recommend - An analysis of accuracy, popularity, and sales diversity effects," in *21st International Conference on User Modeling, Adaptation and Personalization (UMAP 2013)*, Rome, Italy, 2013.

[7] Y. Koren, "Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model," in *KDD'08*, Las Vegas, Nevada, USA, 2008.

[8] B. Holger, "An existence proof for the singular value decomposition," 27 October 2004. [Online].

[9] Y. G. Eckart G., "The approximation of one matrix by another of lower rank," *Psychometrika*, pp. 211-218, 1936.

[10] S. Funk, "Netflix Updata: Try It at Home," 11 December 2006. [Online]. Available: sifter.org/~simon/journal/20061211.html.

[11] A. Fischer and C. Igel, "Training Restricted Boltzmann Machines: An Introduction," *Pattern Recognition,* pp. 25-39, 2014.

[12] D. Bickson, "Disk-based large-scale graph computation," 12 2012. [Online]. Available: http://bickson.blogspot.com/2012/12/collaborative-filtering-with-graphchi.html.

[13] A. Feuerverger, Y. He and S. Khatri, "Statistical Significance of the Netflix Challenge" *Statistical Science,* pp. 202-231, 2012.