**Hybrid Recommender System**

**Yaqi Zhou; Yanan Huo; Li Luo; Kejia Shi**

**March 2016**

**1. Abstract**

With the development of internet and the coming of the age of big data, personalized recommendation services has become a new fashion. We mainly use R to explore and develop the algorithm with package *Recommenderlab*, where the collaborative filtering has been developed maturely, to some extent. The two main parts in this report are exploring the R package to predict the ratings based on the 100 thousand MovieLense data set and combine the information from both users and items to develop a new algorithm. Furthermore, we also managed to connect to the High Throughput Computing center to realize the algorithm on the 20 million MovieLense data set.

**2. Plan**

The first thing we did is to explore the 100 thousand MovieLense data set from website, comparing with the built-in data set in the *Recommenderlab* package and to make sure that the code in R can be generally utilized into both data sets. The data set from website with several files are not completely accurate or concise as the built-in data set, where there are some overlap documented ratings of the same user and the same item and even the same movie has two item ID numbers. To prevent the similar situation happened to the 20 million data set, we developed our own R code to tidy up the data set. Meanwhile, data visualization is a necessary process to express the data structure in a macroscopicalrespective, which are shown in the next part.

Secondly, we looked through several papers about the recommender system to learn the algorithm inside the package and even a book by S.K. Gorakala and Michele Usuelli, 2015. Following these papers and book, we learned the mechanism of the recommendation models and evaluated the predict result. We have plot out the ROC curves and compared them based on one-user prediction.

Moreover, we recognized some limitations of collaborative filterings, such as cold start problem and less information in rating matrix. Thus, we intended to combine the information of users and items to provide more precise judgment for prediction.

Finally, we evaluate the

**3. Result**

1. Data Exploring

In the beginning, we developed our own code to tidy up the 100 thousand data set and explore the nature of it. We have 943 users and 1664 movies the data set, with 99392 ratings. Firstly, similarity between users and between items is the main measurements that collaborative filtering algorithms based on, where it is calculated relied on the similarity of rating. To illustrate, we tabled the similarity matrix among the first five users and items and their image plot, respectively.

**Table 1.1 User Similarity**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| user | 1 | 2 | 3 | 4 | 5 |
| 1 | 0.000 | 0.169 | 0.038 | 0.066 | 0.380 |
| 2 | 0.169 | 0.000 | 0.097 | 0.153 | 0.074 |
| 3 | 0.038 | 0.097 | 0.000 | 0.333 | 0.022 |
| 4 | 0.066 | 0.153 | 0.333 | 0.000 | 0.033 |
| 5 | 0.038 | 0.074 | 0.022 | 0.033 | 0.000 |

*The cell counts in the table are the distances between the two users, where 0.000 means that the two users are exactly the same and 1.000 means that the two users have entirely different ratings based on the items that they have rated.*



**Figure 1.1.1**The deeper the color is, the close that the two users rated.

**Table 1.2 Item Similarity**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| item | Tory Story(1995) | Golden Eye(1995) | Four Rooms(1995) | Get shorty(1995) | Copycat(1995) |
| Tory Story(1995) | 0.000 | 0.402 | 0.330 | 0.455 | 0.288 |
| Golden Eye(1995) | 0.402 | 0.000 | 0.273 | 0.503 | 0.319 |
| Four Rooms(1995) | 0.330 | 0.273 | 0.000 | 0.325 | 0.213 |
| Get shorty(1995) | 0.455 | 0.503 | 0.325 | 0.000 | 0.334 |
| Copycat(1995) | 0.288 | 0.319 | 0.213 | 0.334 | 0.000 |

*The cell counts in the table are the distances between the two items, where 0.000 means that the two items are exactly the same and 1.000 means that the two items have got entirely different ratings.*



**Figure 1.1.2** The similarity among these five movies seems not so close.

Since there are plenty of unrated items, the rating matrix must be a sparse matrix, with few rating values. We got rid of the large amount of missing values and picked the 99392 ratings out to explore the details. We drew a histogram of the ratings without missing data and it tells that most ratings are larger than 2 and 4 is the most common rating.



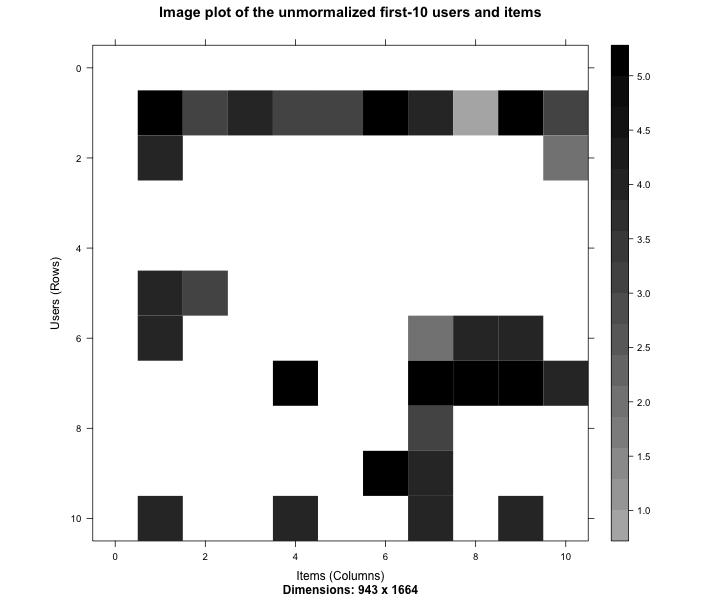
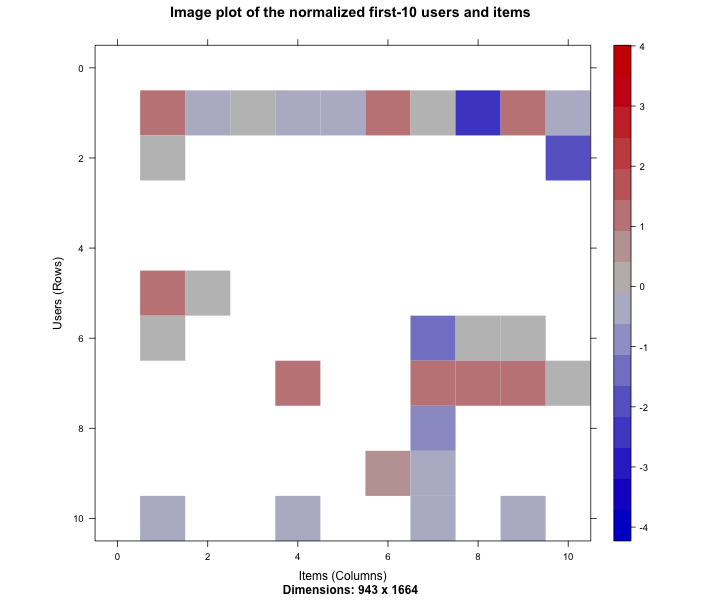
**Figure 1.1.3** Histogram of the ratings without missing values

Merely knowing the rough distribution of ratings are far from knowing well the nature of this data set. From the *Recommenderlab* package, we can learn quickly the number of non-missing values for each column and also the average value within columns. Then, we can sort the Movies by the number of rated, which can be interpreted as the popularity ranking, to some extent. The next plot shows that the number of views of the top-ten movies.



**Figure 1.1.4** Number of views of the top-ten movies

Since that different users have their own judgment criteria, there may be bias within a user and sequentially, removing this effect by normalizing the data is the next step. To normalize the data, we expect the average of each user is 0. To see whether there is bias existing within a user, we plot out the heatmap before and after normalization to indicate the necessity of normalization.



**Figure 1.1.5** The heatmap of the first 10 users over the first 10 items before and after normalization.

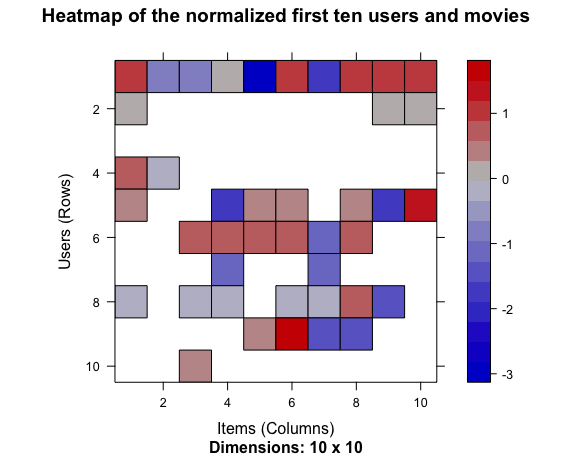
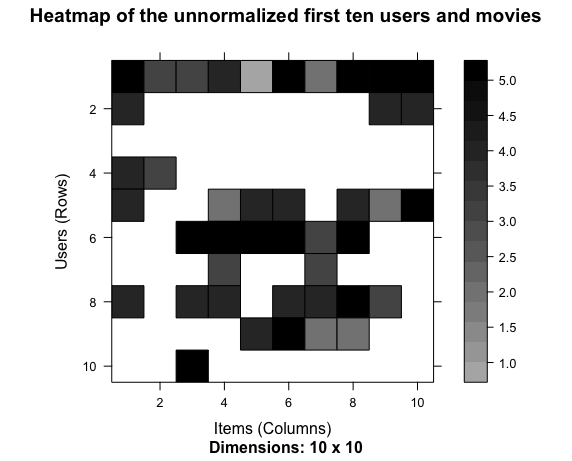
Comparing the two figures above, the 10th user (row) could be the most typical user to explain the importance of normalization, where he gave all the 4.0 rating of the first four items, while after normalization, the 4.0 rating turns out to be -1.0, which is even below his average rating. The similar situation happens through the whole data set, thus we must normalize the data before applying models on it.

1. Data preparation

To prepare the data, we follow the next two steps:

1. Select the relevant data;
2. Normalize the data.

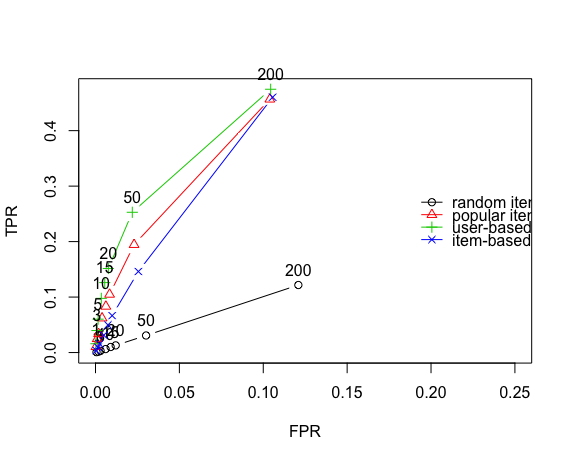
Since plenty users rated only a few movies and lots of movies have been rated only a few times, the situation leads to a sparse rating matrix, and, sequentially, much more bias. These missing value could trigger the less accuracy in predicting their rating. Thus, it is necessary to determine the minimum numbers of users per movie and vice versa. We should do several iterations to leave out the correct value of this particular number. We require that users should rated as least 50 movies and movies should be rated at least 100 times. Then the rating matrix we got here contains 560 users and 332 items with 55298 ratings, which decreases the rating matrix of MovieLens by more than half. Next, we do the normalization and plot out the heatmaps.



**Figure 1.2.1** The heatmaps of the first ten users and movies after filtering the frequency.

1. Model Evaluating

Six models are built in the package, while we focus on these two, User-based Collaborative Filtering and Item-based Collaborative Filtering, with the comparing group, Popular and Random. We choose the first 905 users and all movies to be the training data and the 906th user to be the user that we predict and evaluate. With the particular training data set, we run Recommenderlab to give the recommender lists of the test user, with top-1, 3, 5, 10, 20, 50 and 200 lists. With *k-fold* method and setting *k* to be 4, we plot out the ROC curves as following. To evaluate the effectiveness of the four methods, we can compute the area under the curves, which shows apparently that the UBCF is the best one.



**Figure 1.3.1** ROC curves of the four methods.

On the other hand, the running time of the IBCF lasts more than 50 times than the other methods last. With the performance in the ROC figure, we can conclude that the IBCF may not be the best choice within these four methods.