Movie Recommendation System

Kevin Vu Duc

9/27/2021

1. Introduction

Recommendation system is popular system used in various industries such as retails and entertainment. The system use user's past behavior (purchased products or rating for products) as input to predict the items (or rating for items) that users may have interest in and make recommendation for customers. In this project, based on the ratings which are available to us, we will construct a recommendation system to predict or estimate remaining ratings that users have not given for many movies.

The dataset used for this project is Movielens, which is a part of data used in Netflix challenges in 2006. That contains various information about movies. The data can be downloaded from this link "http://files.grouplens.org/datasets/movielens/ml-10m.zip". Each dataset has six variables, below are descriptions of those variables:

- movieId: Unique ID for the movie.
- title: Movie title (not unique).
- genres: Genres associated with the movie.
- userId: Unique ID for the user.
- rating: A rating between 0 and 5 for the movie.
- timestamp: Date and time the rating was given.

Data is split into two sets, edx dataset as the training set and validation set as test set. The dataset edx contains 9000055. In order to predict missing ratings, we will go through several key steps: selection evaluation method, exploring dataset, identify suitable models, improve model and produce final prediction for ratings. We also point out limitations of project or give suggested methodologies that may apply for future work.

2. Methods and Analysis

2.1 Evaluation metric

There are several common methods to evaluate performance of a model such as Mean Absolute Error, Mean Square Error or Root Mean Squared Error(RMSE). In this project, we will use Root Mean Squared Error as our evaluation metric. The formula to calculate RMSE as below:

$$RMSE = \sqrt{(\frac{1}{N}\sum_{i}(\hat{y}_i - y_i)^2)}$$

where $\hat{y}_i$ is the predicted values, and y_i is the actual values. In R, we can develop a function to calculate RMSE like this:

```
RMSE <- function(actual_y, predicted_y){
  sqrt(mean((actual_y - predicted_y)^2))
                                  }
```

2.2 Data Exploration

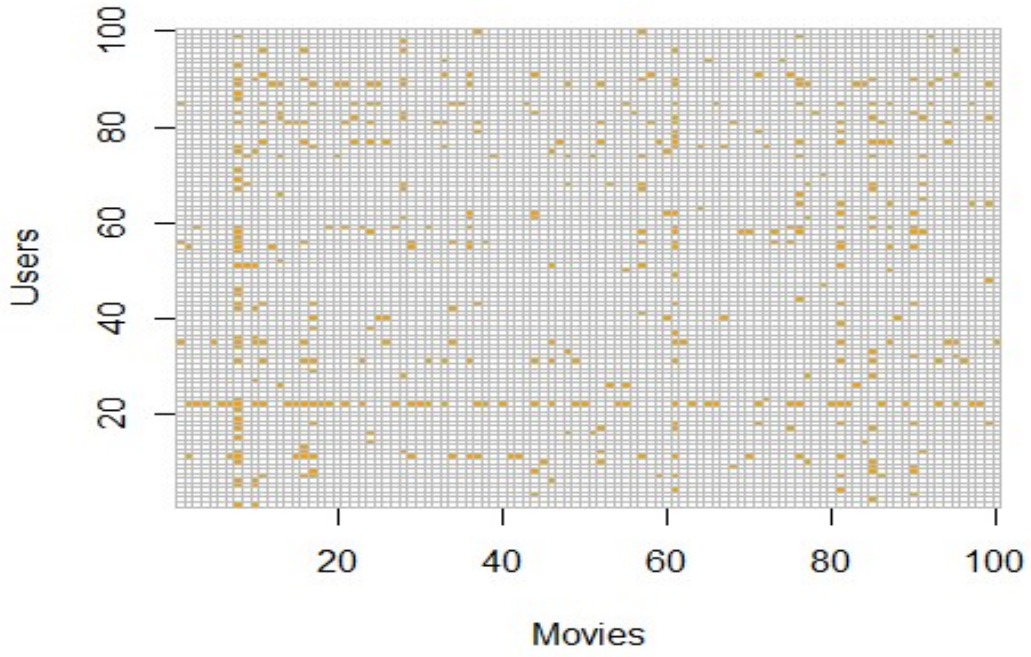
Below are five rows of edx dataset that will give us a general understanding about this data.

```
head(edx) %>% knitr::kable(align = "c")
```

userId	movieId	rating	timestamp	title	genres
1	122	5	838985046	Boomerang (1992)	Comedy|Romance
1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi
1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy

Edx has 9000055 rows and 6 columns. Which contains ratings for 10677 movies given by 69878 users. If all movies were rated by all users, we would get over 746 million ratings. In fact, we have around nine million ratings, which is far less than 746 million. It means that if we present our data in a matrix with users as rows, movies as columns and cells are ratings, we will have vast majority of cells are empty.

We can visualize our data by an image of 100x100 matrix with users as rows and movies as columns to image how much data are missing. It is evident that almost cells are missing and some movies have more ratings than others.



We can explore characteristics of variable in following table. Variables userId, movieId, rating, timestamp is numerical type, while title, genres are character type.

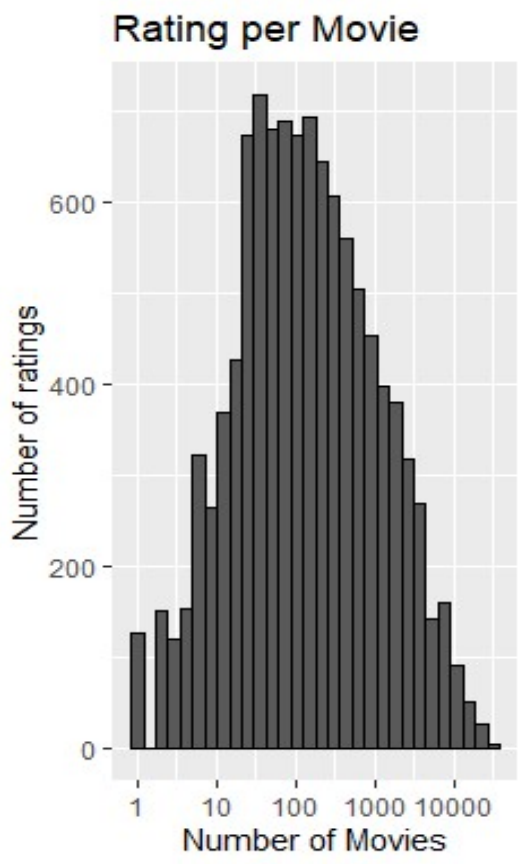
```
str(edx)

Classes 'data.table' and 'data.frame':     9000055 obs. of   6 variables:
   userId          :   int       1   1   1   1   1   1   1   1   1   1   ...
   movieId    :  num    122  185  292  316  329  355  356  362  364  370  ...
   rating          :   num       5   5   5   5   5   5   5   5   5   5   ...
   timestamp:  int    838985046  838983525  838983421  838983392  838983392...
   title     : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)"...
   genres    : chr  "Comedy|Romance" "Action|Crime|Thriller"
```

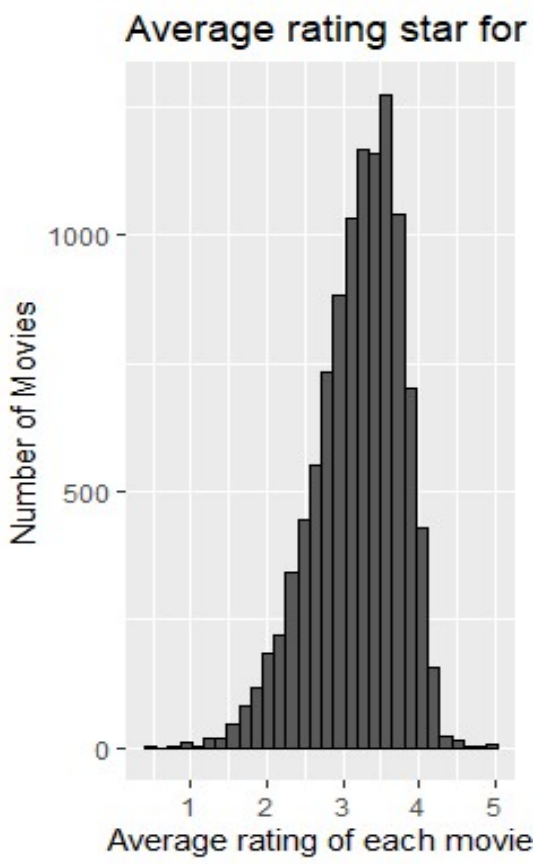
Since ratings are among these values (0.5,1,1.5,2,2.5,3,3.5,4,4.5,5). We can see how ratings are distributed among this set:

```
table(edx$rating)
```

0.5	1	1.5	2	2.5	3	3.5	4	4.5	5
85374	345679	106426	711422	333010	2121240	791624	2588430	526736	1390114

Let's explore how many ratings each movie has. It is evident from the chart that each movie received very different number of ratings. Many got over 400 ratings, while some got less than 50 ratings. The average rating chart shows that the mean values of rating for each movie largely fall between 2.5 star to 4 star and tend to skew to the left.





We provide here the top eight movies which have highest number of ratings, each has around 30000 ratings. And the bottom eight movies with lowest number of ratings, each movie has only rating.

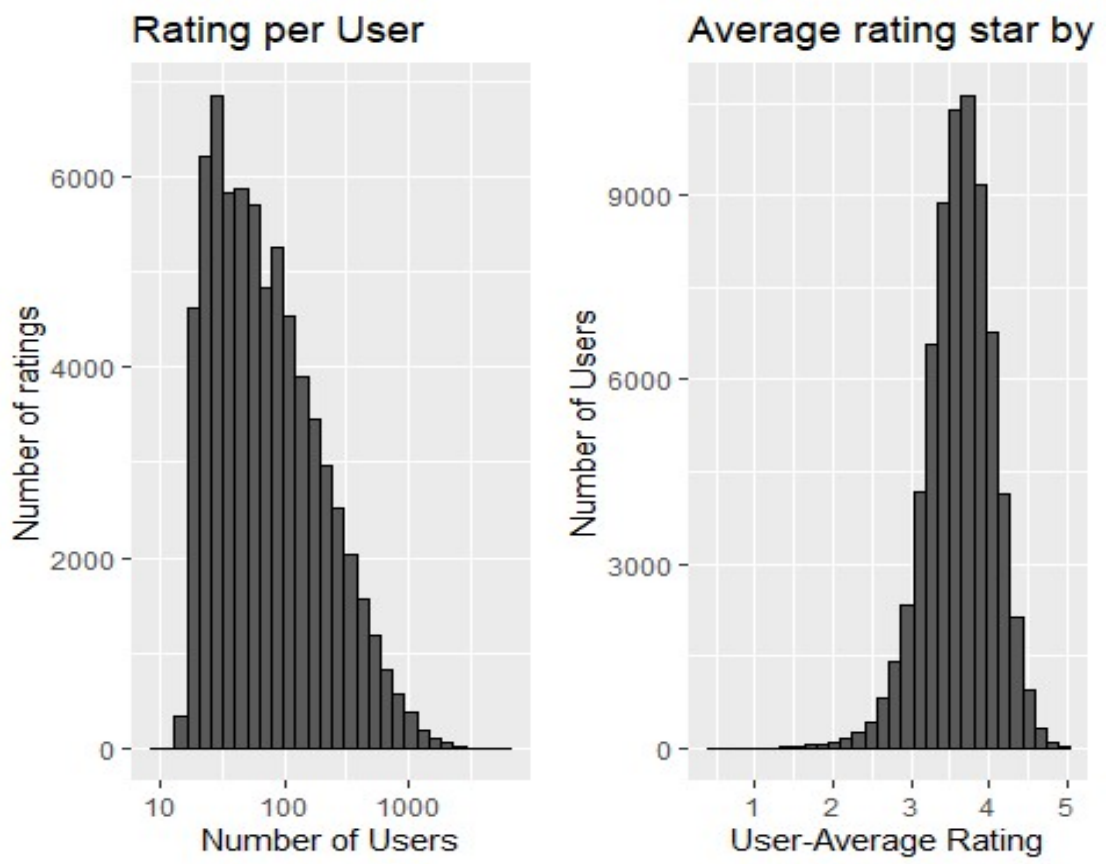
Movies with highest number of rating

movieId	n_rating	movie
296	31362	Pulp Fiction (1994)
356	31079	Forrest Gump (1994)
593	30382	Silence of the Lambs, The (1991)
480	29360	Jurassic Park (1993)
318	28015	Shawshank Redemption, The (1994)
110	26212	Braveheart (1995)
457	25998	Fugitive, The (1993)
589	25984	Terminator 2: Judgment Day (1991)

Movies with lowest number of rating

movieId	n_rating	movie
3191	1	Quarry, The (1998)
3226	1	Hellhounds on My Trail (1999)
3234	1	Train Ride to Hollywood (1978)
3356	1	Condo Painting (2000)
3383	1	Big Fella (1937)
3561	1	Stacy's Knights (1982)
3583	1	Black Tights (1-2-3-4 ou Les Collants noirs) (1960)
4071	1	Dog Run (1996)

Similarity, the chart for number of rating given by each user also indicate that each user gave different number of ratings. Some users gave over 4000 ratings, but some user gave less than 50 ratings.



And here are top 8 users with highest number of rating given and top 8 users who gave lowest number of ratings.

Users gave highest number of ratings

userId	n_rating	movie
59269	6616	Toy Story (1995)
67385	6360	Toy Story (1995)
14463	4648	Toy Story (1995)
68259	4036	Toy Story (1995)
27468	4023	Toy Story (1995)
19635	3771	Toy Story (1995)
3817	3733	Jumanji (1995)
63134	3371	Toy Story (1995)

Users gave lowest number of ratings

userId	n_rating	movie
62516	10	Rob Roy (1995)
22170	12	Toy Story (1995)
15719	13	Usual Suspects, The (1995)
50608	13	Toy Story (1995)
901	14	Shawshank Redemption, The (1994)
1833	14	Clerks (1994)
2476	14	Rising Sun (1993)
5214	14	Pocahontas (1995)

2.3 Modeling Methods:

This section will introduce different models to predict unknown ratings and then evaluation performance of models. Although linear models already introduced in the course, it is better to put them here, so we have some sense about performance between models.

2.3.1 Linear Model with Movie and User Effects

Since users provided more ratings will have higher impact to the prediction than users gave less. And movies have high number of rating also have higher influence than movies have less. Then we should take user effect and movie effect into account when making prediction. Let say, $y_{u,i}$ is the rating for movie i by user u, b_i is the average effect of movie i, b_u is the average user effect of user u. Our model can present as below:

$$y_{u,i} = mu + b_i + b_u + \epsilon_{u,i}$$

term $\epsilon_{u,i}$ is residual or noise, mu is the average of rating over all movies. And we have to find minimum value of following function:

$$\sum_{u,i} (y_{u,i} - mu - b_i - b_u)^2$$

2.3.2 Linear model with regularization

It is similar with linear model introduced in 2.3.1, but will add regularized terms to the loss function to penalize noises, extreme points. The model is no change but the loss function is different.

$$y_{u,i} = mu + b_i + b_u + \epsilon_{u,i}$$

Here is the loss function for regularized method

$$\sum_{u,i} (y_{u,i} - mu - b_i - b_u)^2 + \lambda(\sum_i b_i^2 + \sum_i b_u^2)$$

The value of b_i and b_u that minimize that function are given by:

$$\hat{b}_i = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} (y_{u,i} - mu)$$

$$\hat{b}_u = \frac{1}{\lambda + n_u} \sum_{u=1}^{n_i} (y_{u,i} - mu - b_i)$$

where n_i is number of ratings for movie i, n_u is number of ratings given by user u.

2.3.3 Parallel Matrix factorization:

If we present userId, movieId and ratings into a matrix, then our task become to predict unknown entries in the rating matrix based on observed values as shown in table below. Where question marks indicate unknow ratings.

movie1	movie2	movie3	...	movie_n
user_1	1	1	3 ...	5
user_2	2	??	?? ...	2
user_3	??	3	3 ...	??
user_4	4	5	2 ...	2
user_5	5	2	?? ...	2
...	...	...	... ...	...
user_m ??	1 ?? ...	??		

The technique to solve the problem here is called matrix factorization. The idea is to approximate whole rating matrix R_{mxn} by the product of two matrices of lower dimensions, P_{mxk} and Q_{nxk} that:

$$R \approx PQ'$$

Solution for P and Q is given by solving the optimization problem:

$$min_{P,Q} \sum_{(u,v)} [f(p_u, q_v; r_{u,v}) + \mu_P ||p_u||_1 + \mu_q [[q_v]]_1 + \frac{\lambda_P}{2} ||p_u||_2^2, \frac{\lambda_Q}{2} ||p_q||_2^2]$$

where p_u is the u-th row of P, q_v is the v-th row of Q. f is the loss function, μ_P, μ_Q, λ_P,λ_Q are penalty parameters. Q' is the transpose of matrix Q.

3. Results

In this section, we will implement code for different models. First, We split our dataset 'edx' dataset into 'training_set' and 'testing_set' dataset for evaluation purpose. It depends on models we built, if process involve tuning parameters, we will use 'train_set' and 'test_set' for picking optimal parameters. If model do not require tuning process, we can train model directly on 'edx' dataset and make final prediction on 'validation' set.

3.1 Linear Model with Movie and User Effects

For this model, we will work directly on 'edx' dataset, we calculate average rating over all movies, then compute movie effect, after that we calculate user effect. These calculated values are used to make prediction on 'validation' dataset.

```
mu                              <-                              mean(edx$rating)
b_i          <-          edx          %>%          group_by(movieId)          %>%
  summarise(b_i=mean(rating-mu))

b_u        <-        edx        %>%        left_join(b_i,by="movieId")        %>%
  group_by(userId)                                                          %>%
  summarise(b_u=mean(rating-mu-b_i))
user_movie_pred    <-    validation    %>%    left_join(b_i,by="movieId")    %>%
  left_join(b_u,by="userId")                                                %>%
  mutate(pred=mu+b_i+b_u)                                                   %>%
  .$pred
#contain          results          on          result_rmses          tibble
result_rmses   <-   tibble(Method="Linear   Model   with   Movie+User   Effects",
                     RMSE=RMSE(user_movie_pred,validation$rating))
```

This model give us a root mean square error at 0.86535.

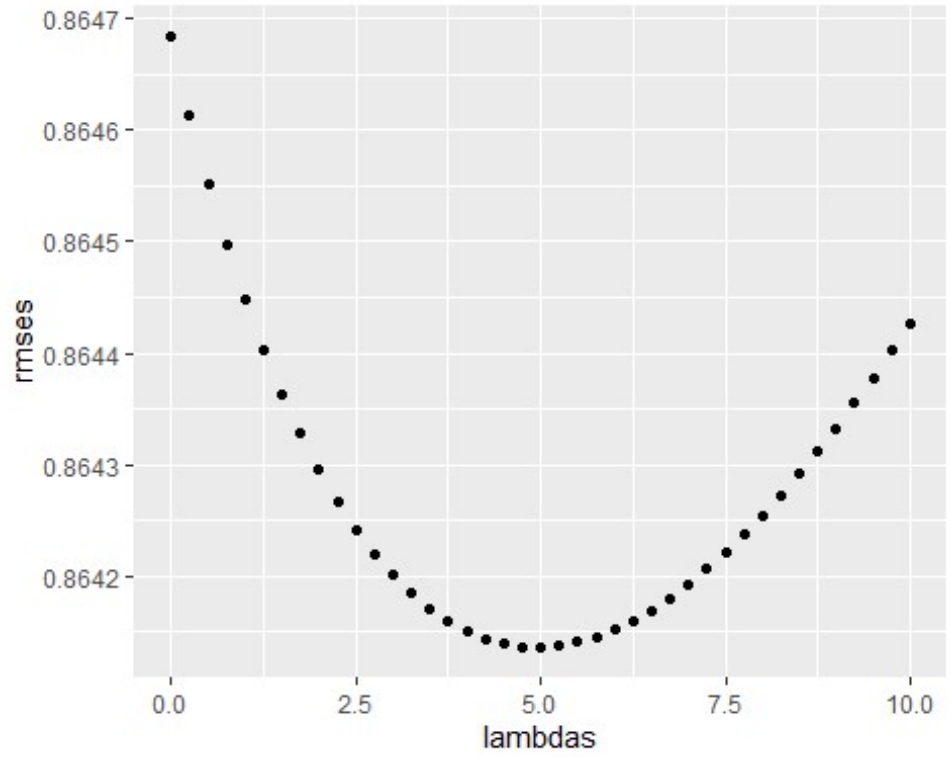
3.2 Regularized Linear Model with movie and user effect

This model, we have to tune λ parameter, then it need to involve 'train_set' and 'test_set' in order to select optimal λ. Below is the code to perform cross validation for λ:

```
mu<-                                                    mean(train_set$rating)
lambdas                              <-                              seq(0,10,0.25)
rmses               <-               sapply(lambdas,                 function(l){
  b_i        <-        train_set        %>%        group_by(movieId)        %>%
    summarise(b_i=sum(rating-mu)/(n()+l))
  b_u      <-      train_set      %>%      left_join(b_i,by="movieId")      %>%
```

```
    group_by(userId)                                                    %>%
    summarise(b_u=sum(rating-mu-b_i)/(n()+l))
  predicted_ratings                <-                test_set          %>%
    left_join(b_i,by="movieId")                                         %>%
    left_join(b_u,by="userId")                                          %>%
    mutate(pred=mu+b_i+b_u)%>%
    .$pred
  return                              (RMSE(predicted_ratings,test_set$rating))
})
```

We can easily see the optimal λ is 5. This correspond with the minimum value of RMSE is 0.86414 on the 'test_set' dataset.



Now, we can make final prediction on 'validation' dataset:

```
l_min                          <-                          lambdas[which.min(rmses)]

b_i        <-        train_set        %>%        group_by(movieId)        %>%
  summarise(b_i=sum(rating-mu)/(n()+l_min))
b_u      <-      train_set      %>%      left_join(b_i,by="movieId")      %>%
  group_by(userId)                                                      %>%
  summarise(b_u=sum(rating-mu-b_i)/(n()+l_min))

reg_user_movie_pred              <-              validation              %>%
  left_join(b_i,by="movieId")                                           %>%
  left_join(b_u,by="userId")                                            %>%
  mutate(pred=mu+b_i+b_u)                                               %>%
```

```
  .$pred
result_rmses                    <-                    bind_rows(result_rmses,
                           tibble(Method="Regularized   Movie+User   Effects",
                           RMSE=RMSE(reg_user_movie_pred,validation$rating)))
result_rmses %>% knitr::kable()
```

Method	RMSE
Linear Model with Movie+User Effects	0.86535
Regularized Movie+User Effects	0.86522

3.3 Parallel Matrix factorization

For this model, we must install and use the package "recosystem". This model supports for different tasks: training, tuning, exporting model (exporting P and Q matrices) and prediction. We must convert our R data into recosystem data type.

The argument, data_memory() means we specify a data set from R objects. There are 6 parameters which are tunable, they are dim, costp_l1, costp_l2, costq_l1, costq_l2 and lrate. Tuning parameters for this model take significant amount of time for a normal computer. So, this project we use only default parameter and then ignore tuning process. Actually, performance with default parameters is good enough to beat the project goal (0.8649). Therefore, we will train the model directly on 'edx_data' set then make final prediction on 'validation' set.

```
#create recosystem model object
model <- recosystem::Reco()
#Training the model
model$train(edx_data)
```

```
iter      tr_rmse     obj
    0       0.9548  1.4607e+007
    1       0.8788  1.3354e+007
    2       0.8550  1.3071e+007
    3       0.8440  1.2935e+007
    4       0.8375  1.2876e+007
    5       0.8323  1.2827e+007
    6       0.8289  1.2796e+007
    7       0.8268  1.2774e+007
    8       0.8253  1.2760e+007
    9       0.8244  1.2754e+007
   10       0.8235  1.2742e+007
   11       0.8229  1.2737e+007
   12       0.8224  1.2731e+007
   13       0.8220  1.2727e+007
   14       0.8216  1.2723e+007
   15       0.8213  1.2717e+007
   16       0.8210  1.2717e+007
   17       0.8207  1.2712e+007
```

```
   18       0.8205  1.2710e+007
   19       0.8203  1.2707e+007

#Make prediction on validation set
reco_pred <- model$predict(validation_data,out_memory())
result_rmses <- bind_rows(result_rmses,
                          tibble(Method="Parallel Matrix factorization",
                          RMSE=RMSE(reco_pred,validation$rating)))
```

Final RMSE for 'parallel matrix factorization' model is 0.83242.

3.4 Summary results

Here is the table to summarize performance of three models constructed above. The performance of linear model is lowest at 0.86535, it is slightly higher target value of 0.849. The regularized model performs slightly better but still higher than our target, its RMSE is at 0.86522. And finally, parallel 'matrix factorization' is the most outstanding model, its RMSE is 0.83242, this is far much lower than our expected value.

```
result_rmses %>% knitr::kable()
```

Method	RMSE
Linear Model with Movie+User Effects	0.86535
Regularized Movie+User Effects	0.86522
Parallel Matrix factorization	0.83242

4. Conclusion

We have gone through different steps to reveal important features of the dataset, and construct three different models to predict unknown ratings for different users and movies. Among that, two models beat our targeted value, the best model perform much better than our expected. However, there are still limitations and potential methods we may not explore in this project.

The major limitation of linear model and regularized model is the movies and users must appear in the training set in order to predict ratings successful. In practice, it may not work in that way. There are still new users rated, new movies are updated. And the system need to be able to predict as well. The 'parallel matrix factorization' method solves that issue but a new challenge happen to this method is the computational cost. If we tune several parameters with normal computer, it may take an hour to complete. Of course, if we take time to tune the model, we are possible to get a RMSE that is lower than 0.8.

There are various algorithms used for developing and testing Recommender system, such as: User-based collaborative filtering(UBCF), Funk SVD(SVDF), SVD with column-mean Imputation (SVD), They are all supported by R package called "recommenderlab". There are potential opportunities to develop good models from such algorithms.