HarvardX PH125.9x Movielens project

Yebin Kim

2024-12-07

1 Introduction & Overview

This report is for the project submission of the HarvardX course, “PH125.9x: Data Science: Capstone”, especially for the ‘Movielens project.’ For this project, the MovieLens 10M dataset from ‘GroupLens(https://grouplens.org/)’ was used. And in this introduction section, I’m going to describe the dataset and summarize the goal of the project and key steps that were performed.

1.1 Data Exploration

Before starting this project, let’s explore the MovieLens dataset first. As you see the result of the code below, there are 7 columns; movieId, title, year, genres, userId, rating, timestamp.

```r
# Load packages that can be needed in this project
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```r
library(caret)
```

```
##          : lattice
##
##         : 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```r
library(dslabs)

# Load the MovieLens dataset:
data(movielens)

head(movielens)
```

```
##   movieId                                    title year
## 1      31                          Dangerous Minds 1995
## 2    1029                                    Dumbo 1941
## 3    1061                                 Sleepers 1996
## 4    1129                     Escape from New York 1981
## 5    1172 Cinema Paradiso (Nuovo cinema Paradiso) 1989
## 6    1263                         Deer Hunter, The 1978
##                                      genres userId rating  timestamp
## 1                                     Drama      1    2.5 1260759144
## 2 Animation|Children|Drama|Musical      1    3.0 1260759179
## 3                                  Thriller      1    3.0 1260759182
## 4 Action|Adventure|Sci-Fi|Thriller      1    2.0 1260759185
## 5                                     Drama      1    4.0 1260759205
## 6                                 Drama|War      1    2.0 1260759151
```

Each variable/column is defined as follows:

variable/column	description
movieId	an integer that contains the identification number of each movie
title	a character that contains the name of the movie
year	an integer that includes the year of the movie release
genres	a factor that shows the genre of each movie, delimited by the pipe(‘
userId	an integer that is the identification number of each user who made the rating
rating	a numeric which demonstrates the rating of each movie by each user (a multiple of 0.5, from 0.5 to 5.0)
timestamp	an integer that represents the time when the rating was made

And in this data set, comparing the number of rows with that of the unique users or movies, we can see there could be many empty cells of this data set as below.

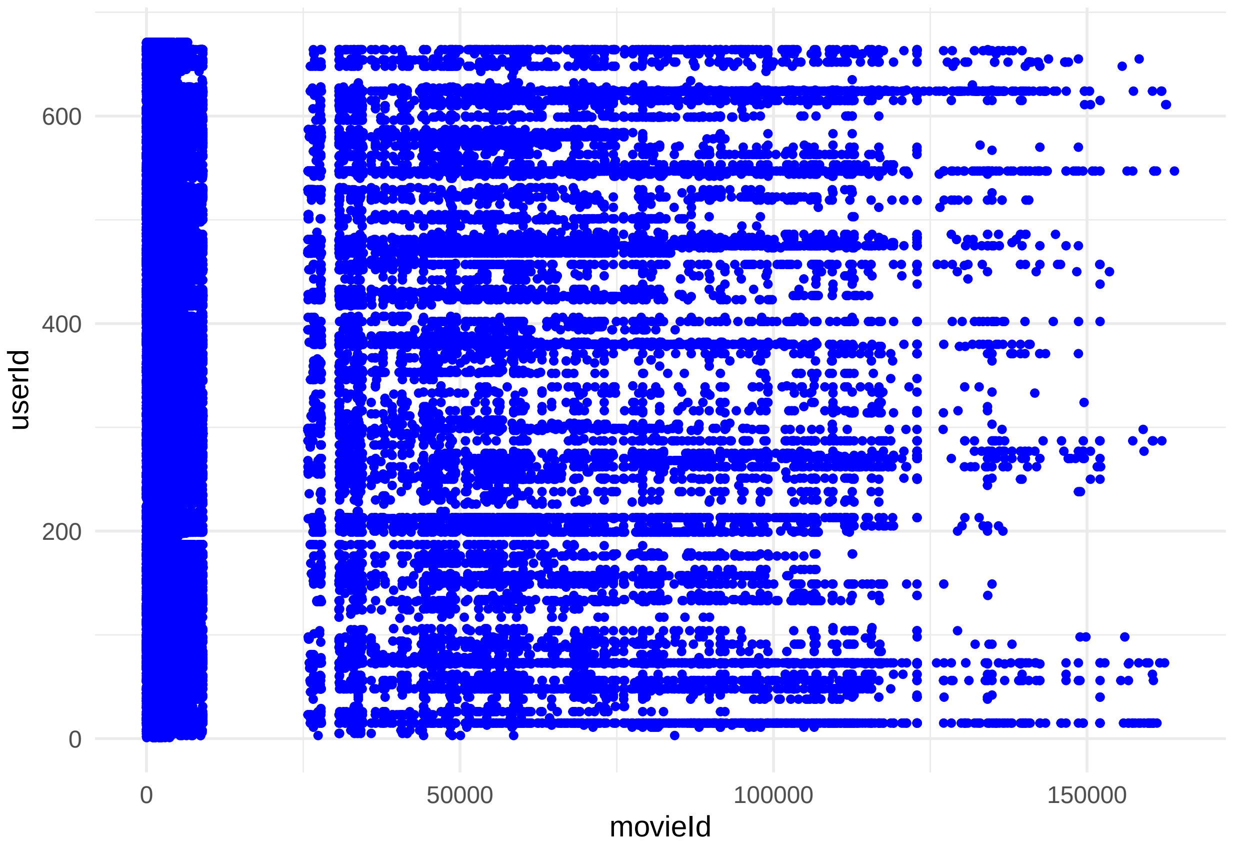
```r
nrow(movielens)
```

```
## [1] 100004
```

```r
movielens %>% summarize(
  n_users = n_distinct(userId),
n_movies = n_distinct(movieId)
)
```

```
##   n_users n_movies
## 1     671     9066
```

```r
ggplot(data=movielens,aes(movieId,userId))+geom_point(color="blue",size=1)+theme_minimal()
```



1.2 Project Objective

The goal of this project is to make a recommendation system using MovieLens dataset. Especially, this recommendation system is evaluated and choosed based on the RMSE(Root Mean Squared Error), and the RMSE of the final model should be lower than 0.86490.

2 Data Analysis

In this chapter, I'm going to explain the process and techniques used, including data cleaning, data exploration and visualization, insights gained, and my modeling approach.

2.1 Dataset Pre-processing

First of all, this is the overview of each dataset using the createDataPartition.

edx(90%)	validation(10%)
edx_train_set(72%)+edx_test_set(18%)	validation(10%)

At first, 10% of the movielens dataset is used as a validation data set, which will be the 'final_holdout_test' set. To make sure not use this dataset to train my algorithm, this validation set should ONLY be used to test my final algorithm at the end of my project with my final model. So it may not be used to test the RMSE of multiple models during model development. Therefore, I have to use 90% of the dataset(a.k.a. edx) to develop my model. So, except the validation dataset, I split the edx dataset into training set('edx_train_set') and test test('edx_test_set'). The test set, 20% of the edx dataset, will be used to test the RMSE. And the training set, 80% of the edx dataset, will be used to train the model. So, I'm going to use only train_set to train my model.

```r
# Validation data set will be 10% of MovieLens data
set.seed(1)
test_index<-createDataPartition(y=movielens$rating, times=1, p=0.1, list=FALSE)
edx<-movielens[-test_index,]
valid<-movielens[test_index,]

# Make sure userId and movieId in validation data set are also in edx set
validation<-valid %>%
  semi_join(edx, by="movieId") %>%
  semi_join(edx, by="userId")

# Add rows removed from validation set back into edx set
removed<-anti_join(valid, validation)
```

```
## Joining with `by = join_by(movieId, title, year, genres, userId, rating,
## timestamp)`
```

```r
edx<-rbind(edx,removed)

# Split edx dataset into training and test set
## Create train set and test set
set.seed(1)
test_index<-createDataPartition(y=edx$rating, times=1, p=0.2, list=FALSE)
edx_train_set<-edx[-test_index,]
edx_test_set<-edx[test_index,]

## Make sure userId and movieId in validation set are also in edx set
edx_test_set <- valid %>%
  semi_join(edx_train_set, by = 'movieId') %>%
  semi_join(edx_train_set, by = 'userId')

## Add rows removed from validation set back into edx set
removed <- anti_join(valid, edx_test_set)
```

```
## Joining with `by = join_by(movieId, title, year, genres, userId, rating,
## timestamp)`
```

```r
edx_train_set <- rbind(edx_train_set, removed)

#The RMSE function that can be used in this project
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}


## Extract the genre in edx&validation datasets
#edx
edx <- edx %>%
  mutate(genres = fct_explicit_na(genres,
                                  na_level = "(no genres listed)")
  ) %>%
  separate_rows(genres, sep = "\\|")
```

```
## Warning: There was 1 warning in `mutate()`.
## i In argument: `genres = fct_explicit_na(genres, na_level = "(no genres
##   listed)")`.
## Caused by warning:
## ! `fct_explicit_na()` was deprecated in forcats 1.0.0.
## i Please use `fct_na_value_to_level()` instead.
```

```
#validation
validation <- validation %>%
  mutate(genres = fct_explicit_na(genres,
                                  na_level = "(no genres listed)")
  ) %>%
  separate_rows(genres,
                sep = "\\|")
```

And in this project, I don't use other variables except for userId, movieId, rating, title, and genres. So, I select necessary columns in each data set.

```
# Select necessary columns in edx&validation dataset
edx <- edx %>% select(userId, movieId, rating, title, genres)
validation <- validation %>% select(userId, movieId, rating, title, genres)
```

2.2 Movie Model

Now, let's start with the prediction model. First, I calculated the average rating of all movies and naive rmse, and then save them as values.

```
# Average of all movies
mu_hat <- mean(edx_train_set$rating)
naive_rmse <- RMSE(edx_test_set$rating, mu_hat)
naive_rmse
```

```
## [1] 1.047613
```

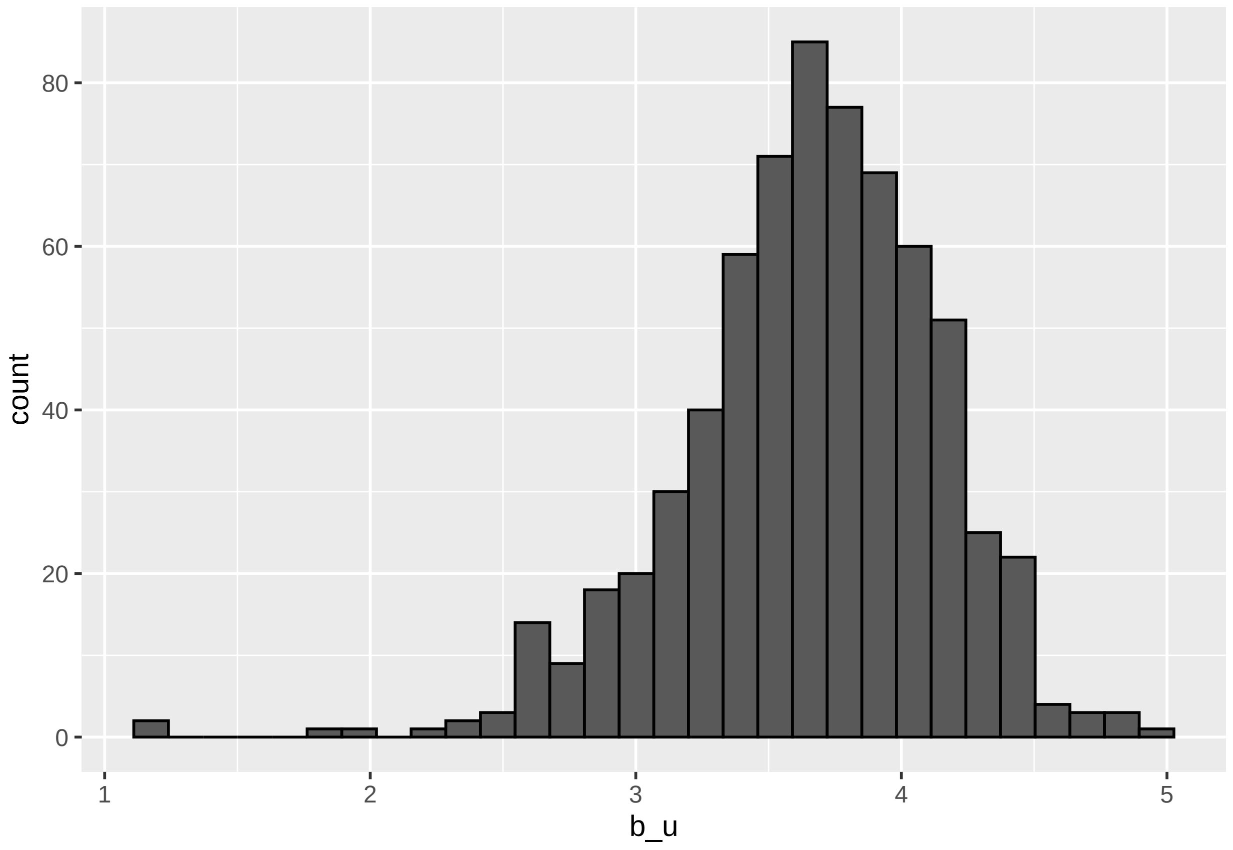
And in this project I won't use 'lm' or 'predict' function. It's because the textbook of our course said that there are thousands of bi as each movie gets one, so the lm() function will be very slow. Therefore I'm not going to run the code lm(), and just use RMSE function to find out the performance of prediction model. This time, I'll make movie based model.

```
# Average by movie
movie_avg <- edx_train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_hat))

# Prediction by movie
predicted_ratings_m<-mu_hat + edx_test_set %>% left_join(movie_avg,by='movieId') %>% pull(b_i)
movie_effect<-RMSE(predicted_ratings_m,edx_test_set$rating)
movie_effect
```

```
## [1] 0.9779385
```

```r
# Average rating for users that have rated over 120 movies
edx_train_set %>%
group_by(userId) %>%
summarize(b_u = mean(rating)) %>%
filter(n()>=120) %>%
ggplot(aes(b_u)) +
geom_histogram(bins = 30, color = "black")
```



As we see, the RMSE became lower than the naive rmse.

2.3 Movie+User Model

Though I have made lower RMSE, the RMSE could not be that low because I only used 'movieId' variable. So, I add the 'userId' variable to make the model more precise.

```r
# Average by movie&user
user_avg <- edx_train_set %>%
  left_join(movie_avg, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_hat-b_i))

# Prediction by movie&user
```

```r
predicted_ratings_m_u<-edx_test_set %>%
  left_join(movie_avg,by='movieId') %>%
  left_join(user_avg, by='userId') %>%
  mutate(p=mu_hat+b_i+b_u) %>% pull(p)
movie_user_effect<-RMSE(predicted_ratings_m_u,edx_test_set$rating)
movie_user_effect
```

```
## [1] 0.9022202
```

Surprisingly, the RMSE goes down when I add the 'userId' variables to the prediction model.

2.4 Movie+User+Genre Model

The 'Movie+User Model' still has to be developed because there are other variables this model hasn't used. So this time, I'm going to use 'genres' variables to make the RMSE of the model lower.

```r
# Average by movie&user&genre
genre_avg <- edx_train_set %>%
  left_join(movie_avg, by='movieId') %>%
  left_join(user_avg,by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu_hat-b_i-b_u))

# Prediction by movie&user&genre
predicted_ratings_m_u_g<-edx_test_set %>%
  left_join(movie_avg,by='movieId') %>%
  left_join(user_avg, by='userId') %>%
  left_join(genre_avg, by='genres') %>%
  mutate(p=mu_hat+b_i+b_u+b_g) %>%
  pull(p)

movie_user_genre_effect<-RMSE(predicted_ratings_m_u_g,edx_test_set$rating)
movie_user_genre_effect
```

```
## [1] 0.9004642
```

2.5 Regularization

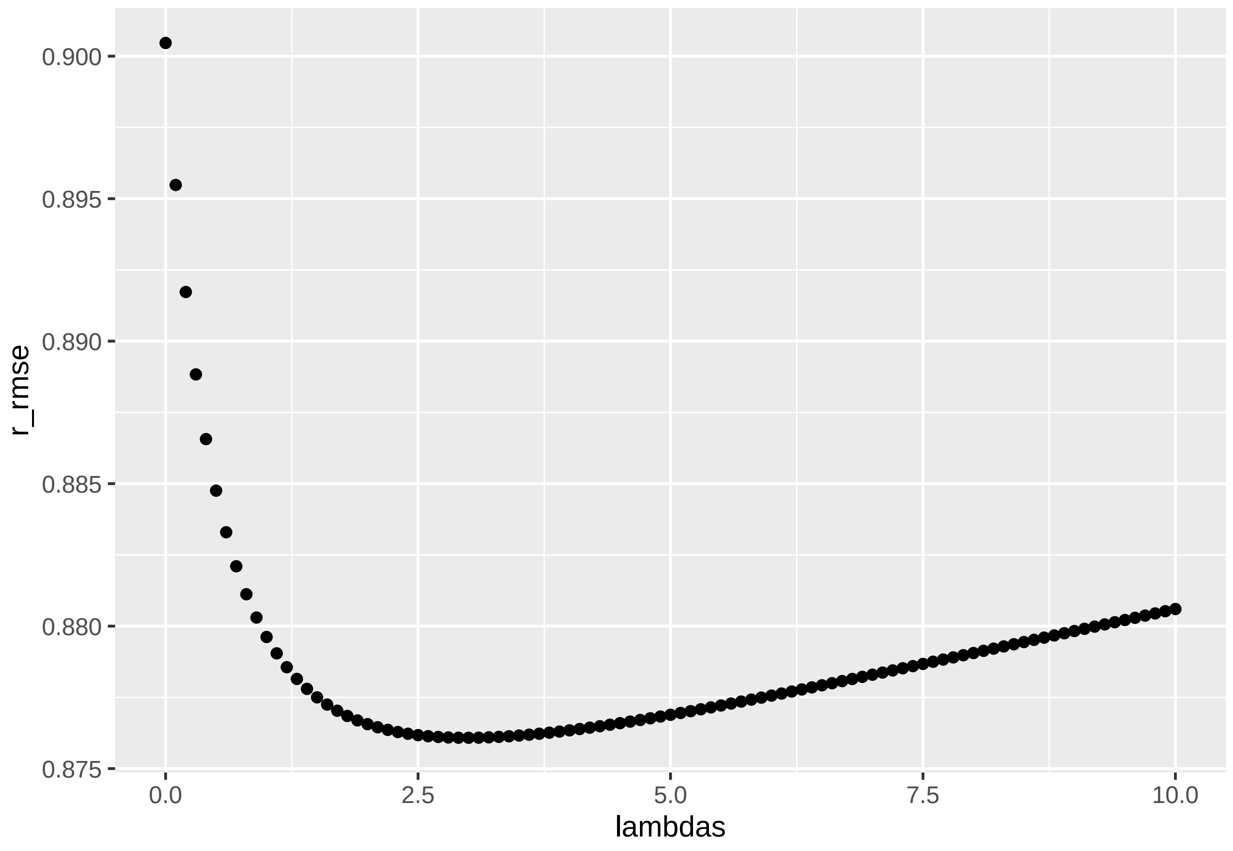
Although the RMSE decreased as we add more variables, there is still another way to make it more lower. Can you guess? That's right. It's 'Regularization'! And in order to do effective regularization, we have to find the optimal lambda to add the penalty.

```r
#Find the optimal lambda for Regularization
lambdas <- seq(0, 10, 0.1)
r_rmse <- sapply(lambdas, function(l){
  b_i_s <- edx_train_set %>%
    group_by(movieId) %>%
    summarize(b_i_s = sum(rating - mu_hat)/(n()+l))
  b_u_s <- edx_train_set %>%
    left_join(b_i_s, by='movieId') %>%
    group_by(userId) %>%
    summarize(b_u_s = sum(rating - b_i_s - mu_hat)/(n()+l))
  b_g_s <- edx_train_set %>%
    left_join(b_i_s, by='movieId') %>%
    left_join(b_u_s, by='userId') %>%
    group_by(genres) %>%
    summarize(b_g_s = sum(rating - b_i_s - b_u_s - mu_hat)/(n()+l))

  predicted_ratings <- edx_test_set %>%
    left_join(b_i_s, by = 'movieId') %>%
    left_join(b_u_s, by = 'userId') %>%
    left_join(b_g_s, by= 'genres') %>%
    mutate(p_r = mu_hat + b_i_s + b_u_s + b_g_s) %>%
    pull(p_r)
  return(RMSE(predicted_ratings, edx_test_set$rating))
})

qplot(lambdas, r_rmse)
```

```
## Warning: `qplot()` was deprecated in ggplot2 3.4.0.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



```
lambda <- lambdas[which.min(r_rmse)]
lambda
```

```
## [1] 3
```

Through the plot above, we can see that the best value of lambda is 3. So I'm going to use this lambda to the regularization. The code as follows:

```
# Regularization
movie_reg_avg <- edx_train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_hat) / (n() + lambda), n_i = n())

user_reg_avg <- edx_train_set %>%
  left_join(movie_reg_avg, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu_hat - b_i) / (n() + lambda), n_u = n())

genre_reg_avg <- edx_train_set %>%
  left_join(movie_reg_avg, by = "movieId") %>%
  left_join(user_reg_avg, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu_hat - b_i - b_u) / (n() + lambda), n_g = n())

predicted_ratings <- edx_test_set %>%
  left_join(movie_reg_avg, by = "movieId") %>%
  left_join(user_reg_avg, by = "userId") %>%
  left_join(genre_reg_avg, by = "genres") %>%
  summarize(pred = mu_hat + b_i + b_u + b_g) %>%
  pull(pred)
```

```
## Warning: Returning more (or less) than 1 row per `summarise()` group was deprecated in
## dplyr 1.1.0.
## i Please use `reframe()` instead.
## i When switching from `summarise()` to `reframe()`, remember that `reframe()`
##   always returns an ungrouped data frame and adjust accordingly.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
movie_user_genre_reg_model <- RMSE(predicted_ratings, edx_test_set$rating)
movie_user_genre_reg_model
```

```
## [1] 0.8760829
```

Can you see the RMSE getting lower? Now let's move on to the final step.

2.6 Final Check

As we made the regularized model, it's time to check of this model by validation data set. You might remember that I created the data partition for 'validation', and never used this in data analysis process including training and testing. Finally, I'm going to use this validation data set to do final check. Let's see how it works.

```r
# Final check with Validation data set
predicted_ratings <- validation %>%
  left_join(movie_reg_avg, by = 'movieId') %>%
  left_join(user_reg_avg, by = 'userId') %>%
  left_join(genre_reg_avg, by = 'genres') %>%
  mutate(pred = mu_hat + b_i + b_u + b_g) %>%
  pull(pred)

valid_predictions <- predicted_ratings[!is.na(predicted_ratings)]
valid_ratings <- validation$rating[!is.na(predicted_ratings)]

final_model <- RMSE(valid_predictions, valid_ratings)
final_model
```

```
## [1] 0.8897014
```

As you see, the RMSE of the final model is 0.8897014.

3. Results

From now on, I'm going to present the modeling results and discuss the model performance that I made. These are the RMSEs of each model calculated by the test set in edx.

Method	RMSE(test set)
Just the average (naive)	1.0476129
Movie Effect Model	0.9779385
Movie + User Effect Model	0.9022202
Movie + User + Genre Effect Model	0.9004642
Movie + User + Genre Effect Model (Regularized)	0.8760829

Among those RMSEs, the regularized one is the lowest. So, finally I apply this model into the final data set, which was defined as 'validation'. And the result is as below.

Method	RMSE(validation)
Movie + User + Genre Effect Model (Regularized)	0.8897014

4. Conclusion

Finally, let me give you a brief summary of the report, and the limitations and future work of my model. According to the objective of this project, the RMSE of the final model should be lower than 0.86490. And finally, the model I made has 0.8897014 as RMSE. So it can be said that this data analysis partially achieved the goal of this project.