HarvardX Data Science Capstone Project 1

Maggie Yim

8/29/2021

HarvardX Data Science Capstone Project: MovieLens

Section 1. Introduction

In this project, a movie recommendation system will be created using the open-source MovieLens dataset. Movie recommendation system aims to use the past records of movie ratings that users have given to predict their future ratings and make specific recommendations to users using machine learning techniques and algorithms. From a business standpoint, it is profitable as it can enhance user experience by personalizing the platform rather than simply recommending the most popular items.

The entire MovieLens dataset is very large, which includes 27,000,000 ratings and 1,100,000 tag applications applied to 58,000 movies by 280,000 users. Includes tag genome data with 14 million relevance scores across 1,100 tags. You can find the entire latest data set here.Therefore, a small subset of the dataset will be used for this project in order to ease the computation. It is a stable benchmark dataset which includes 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users. You can find the dataset here

To build a movie recommendation system, this project trained a machine learning algorithm considering global average, movie-specific effect, and user-specific effect. In this document, it will separate into three sections. First, it shows the process and techniques used, including data cleaning, data exploration and visualization, insights gained, and the modeling approach. Second, it presents the modeling result and discusses the model performance. After that, it gives a brief summary of the report, its limitations and future work.

Section 2. Dataset

Section 2.1. Data Preparation

Section 2.1.1. Installing Packpage First, certain R packpages, which would be used in data preparation, analysis and visualization, need to be installed in R. Caret package is used, same as other machine learning algorithms using this code:

```
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.1     v dplyr   1.0.5
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1
```

```
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caret

## Loading required package: lattice

##
## Attaching package: 'caret'

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

```
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: data.table

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

## The following object is masked from 'package:purrr':
##
##     transpose
```

```
library(tidyverse)
library(caret)
library(data.table)
```

Section 2.1.2. Create two set of data - Train Set and Validation Set

Then, the MovieLens dataset is split into two sets of data which are training set (90%) and validation set (10%) in order to perform machine learning model later.

While performing machine learning, there are three phases.

1. **Training phase**: Training set is used to train a model.
2. **Validation phase**: Validation set is used to evaluate the model fit on the training set and it is useful to fine-tune the model hyperparameters (i.e. the parameter whose value is used to control machine learning process, for example train-test split ratio, choice of optimization algorithms) and affect the final model during development stage.
3. **Application phase**: Testing set is used to get the prediction result from the freshly-developed model. Since we do not have any reference value in this real-world data, only the result in validation phase is used to speculate the model performance in this project. Sometimes, validation set is treated as testing set but it is not a good practice because testing set is well curated in general. It includes data samples that span different classed the model would encounter when the model is used in the real-world

The following code is used to create training set and validation set.

```
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")

# if using R 4.0 or later:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))

movielens <- left_join(ratings, movies, by = "movieId")

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

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

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

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

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

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

About Dataset Split Ratio

In this project, 90% of dataset went to training and the rest 10% is used as validation set. The decision of dataset split ratio is case by case but it depends on two main things: the total data sample in dataset and the actual model to be trained. If the model has very few hyperparameters and it will be easier to validate and tune the model, the size of validation set can be reduced. In the other hand, if the model has a lot of hyperparameters, it is preferable to have a larger validation set.

Section 2.2 Additional Information of Training Dataset Section 2.2.1 Overview

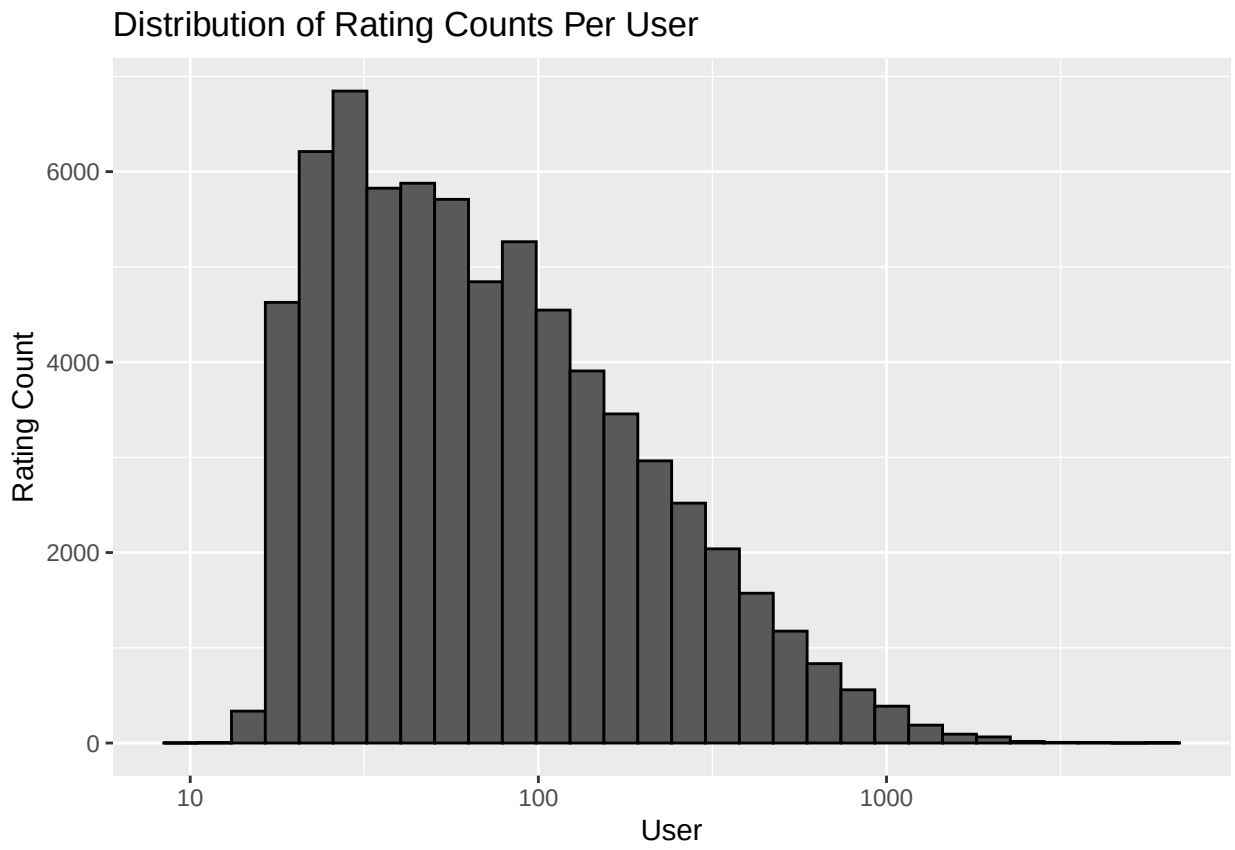
Before developing a movie recommendation system, let's look at some general properties of the data. As mentioned above, a small subset of the open-source MovieLens dataset is used to create a predictive model. The dataset contains 9,000,055 rows and 6 columns, involving 10,677 movies and 69,878 users. Each row represents a rating given by one user to one movie.

```
#Statistics in the edx dataset
nrow<-dim(edx)[1]
ncol<-dim(edx)[2]
nmovie<-n_distinct(edx$movieId)
nuser<-n_distinct(edx$userId)

tibble(Summary = c("Number of Row in Dataset","Number of Column in Dataset","Number of Movies","Number
```

Summary	Count
Number of Row in Dataset	9000055
Number of Column in Dataset	6
Number of Movies	10677
Number of Users	69878

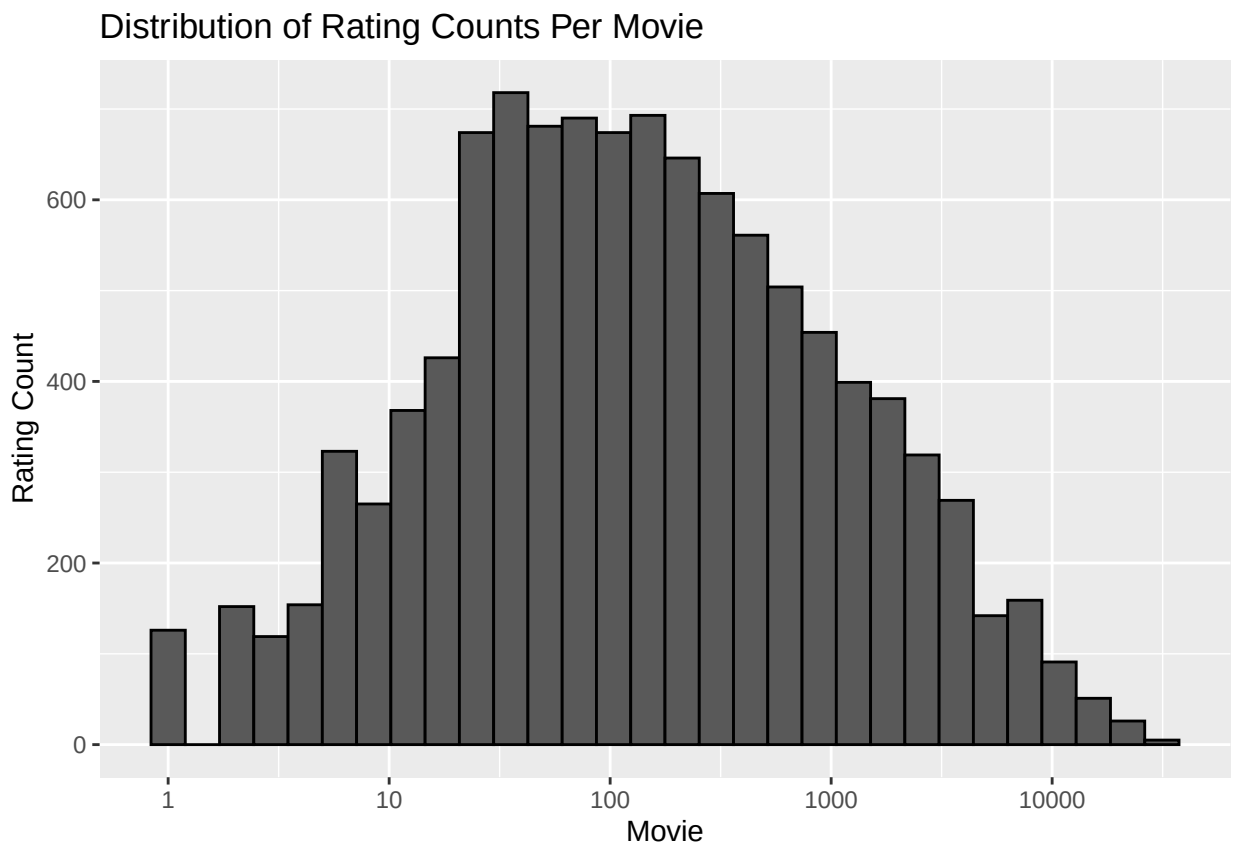
Section 2.2.2. Rating Count by User

Here is the distribution of rating counts per user. Some users are more active than others at rating movies. Notice that some users have rated over 6,000 movies while others have only rated a handful.



Section 2.2.3. Rating Count by Movie

A second observation is that some movies get rated more than others and some even have no rating.



Section 2.2.4. Rating by Movie Genre

From the following table, it shows the number of movie rating for each type of movie genre. Drama movie has the largest number of rating while romance movie has the smallest number of rating.

```
#Number of movie ratings are in each of the following genres in the edx dataset
genres = c("Drama", "Comedy", "Thriller", "Romance")

data_frame(Genre = genres, "Number of Rating" = sapply(genres, function(g) {
    sum(str_detect(edx$genres, g))
}))%>% knitr::kable()
```

```
## Warning: `data_frame()` was deprecated in tibble 1.1.0.
## Please use `tibble()` instead.
```

Genre	Number of Rating
Drama	3910127
Comedy	3540930
Thriller	2325899
Romance	1712100

Section 2.2.5. Movie Rating of Zero

No movies have a rating of 0. Movies are rated from 0.5 to 5.0 in 0.5 increments. The number of 0s can be found using *edx %>% filter(rating == 3) %>% tally()*.

Section 2.2.6. Top Five Rating

Per the following table, rating of 4 is mostly given by user and rating of 2 is the least.

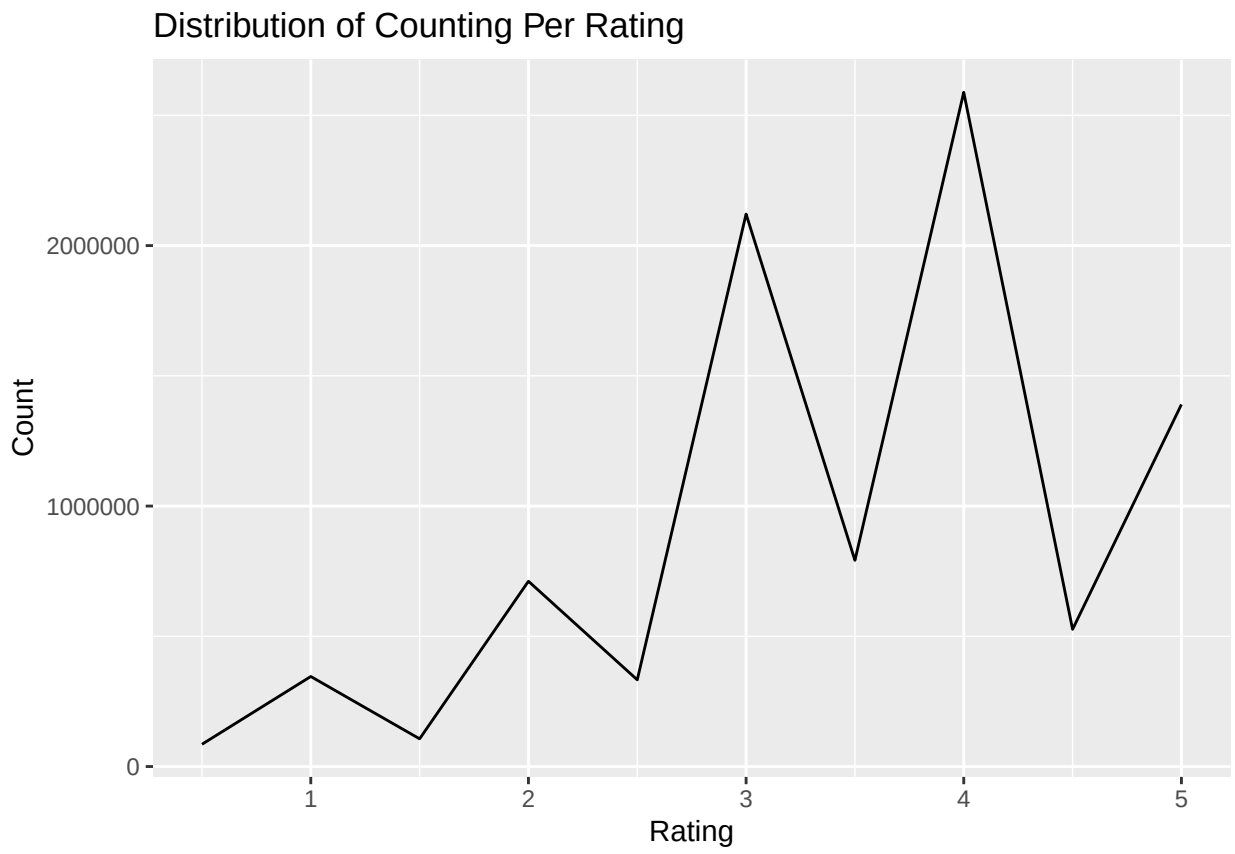
```
#The top five ratings in order from most to least
edx %>%
  group_by(rating) %>%
  summarize(count = n()) %>%
  top_n(5) %>%
  arrange(desc(count))%>%
  knitr::kable()
```

```
## Selecting by count
```

rating	count
4.0	2588430
3.0	2121240
5.0	1390114
3.5	791624
2.0	711422

Section 2.2.7. Half Star Rating

In general, half star ratings are less common than whole star ratings (e.g. there are fewer ratings of 3.5 than there are ratings of 3 or 4)



Section 3. Method

Section 3.1. The Dataset

After separating the dataset into training set and validation set, a machine learning algorithm is developed using the training set (i.e edx set) to predict movie ratings in validation set (i.e. final hold-out test set) as if they were unknown.

From the above section, we know that the dataset consists of 6 columns but only 3 of them are used in developing machine learning algorithm. They are: - userId - movieId - rating

You can see a snapshot of 6 data records below:

```
edx %>%
    select(userId,movieId,rating) %>%
    head() %>%
  knitr::kable()
```

userId	movieId	rating
1	122	5
1	185	5
1	292	5
1	316	5
1	329	5

userId	movieId	rating
1	355	5

Section 3.2. Performance Metrics - Root Mean Squared Error

In order to compare different model or compare model with some baseline, a loss function is needed to quantify how it perform. There are two common methods to measure the accuracy of a recommendation system: Mean Absolute Percentage Error (MAPE) and Root Mean Squared Error (RMSE).

MAPE is the average of the absolute differences between actual observation and prediction in term of percentage, without considering the direction and all individual difference has equal weight.

RMSE is the square root of the average of squared difference between actual observation and prediction. In other words, it measures the average magnitude of error.

In this project, RMSE is used to evaluate the model performance. Although RMSE does not express mean of the error alone and has implication that is difficult to understand or tease out, RMSE has distinct advantage over MAPE, which is avoiding to use absolute value as it is not desirable in most of the mathematical calculations. Moreover, RMSE can penalize large errors more than MAPE.

The following function is created to compute this residual means squared error for a vector of ratings and their corresponding predictors.

```
RMSE <- function(true_rating, predicted_rating){
  sqrt(mean((true_rating - predicted_rating)^2))
}
```

Section 3.3. Designing Movie Recommendation System

Section 3.3.1. Considering average in the model only

First, let's start to build the simplest movie recommendation system by assuming the same rating for all movies regardless of user. In order to minimize the RMSE, the assigned rating is the least squares estimate of the "true" rating for all movies. In this case, it is the mean of all ratings in training set:

```
mu_hat <- mean(edx$rating)
mu_hat
```

```
## [1] 3.512465
```

After that, we can test the model performance using validation set. If all unknown rating in validation set is predicted by the above rating average, we obtain the following RMSE:

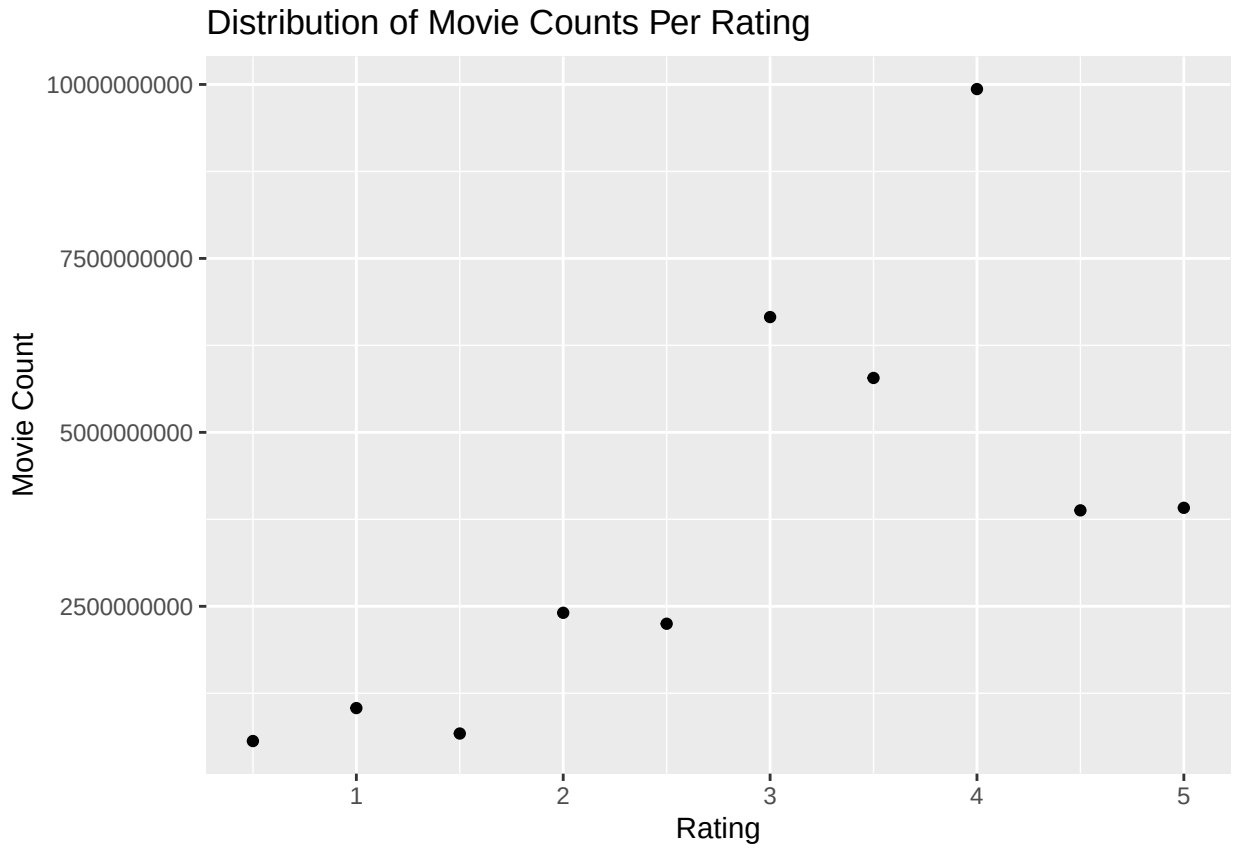
```
naive_rmse <- RMSE(validation$rating, mu_hat)

rmse_results <- data_frame("Predictive Method" = "Just Global Average", RMSE = naive_rmse)
rmse_results %>% knitr::kable()
```

Predictive Method	RMSE
Just Global Average	1.061202

The RMSE of the model including global average of rating is 1.0612018, which is quite high because this model is generalizing for user and not personalizing for the user's preference. It can be improved by adding bias, which is the simplifying assumption to make

Section 3.3.2. Modeling Movie Effects

From the graph below, it shows that each movies are rated differently so some movies are rated higher than others. Therefore, the previous model can be improved by adding additional constant called effects or bias to shorten the difference between the predicted value and the correct value which we are trying to predict.



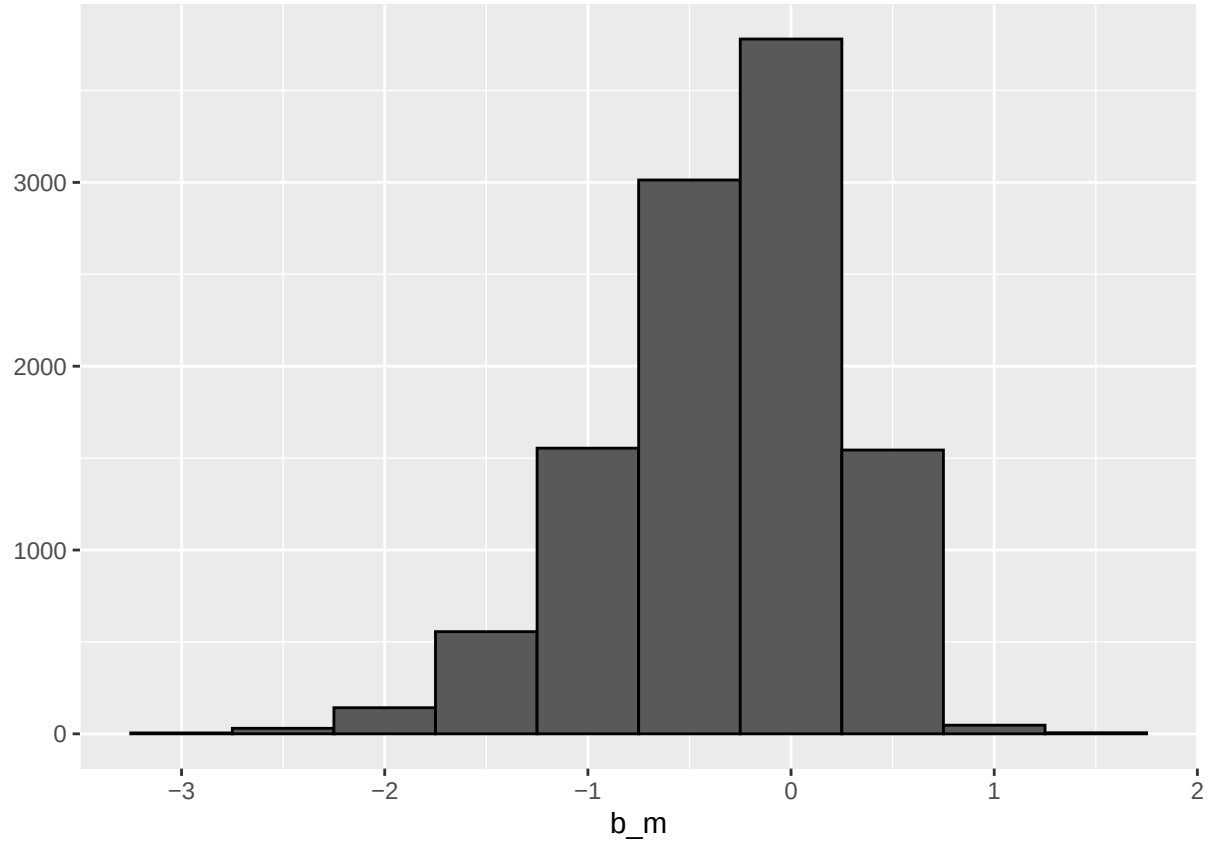
Rating	Number of Movie
0.5	563498937
1.0	1037027919
1.5	671611949
2.0	2406762708
2.5	2249344100
3.0	6657056181
3.5	5782103342
4.0	9933741656
4.5	3878970447
5.0	3915430690

As the least squares estimate bias is just the average of (specific movie's rating - average of all rating), the bias can be computed this way:

```
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_m = mean(rating - mu_hat))
```

From the graph below, it show these bias estimates vary substantially

```
qplot(b_m, data = movie_avgs, bins = 10, color = I("black"))
```



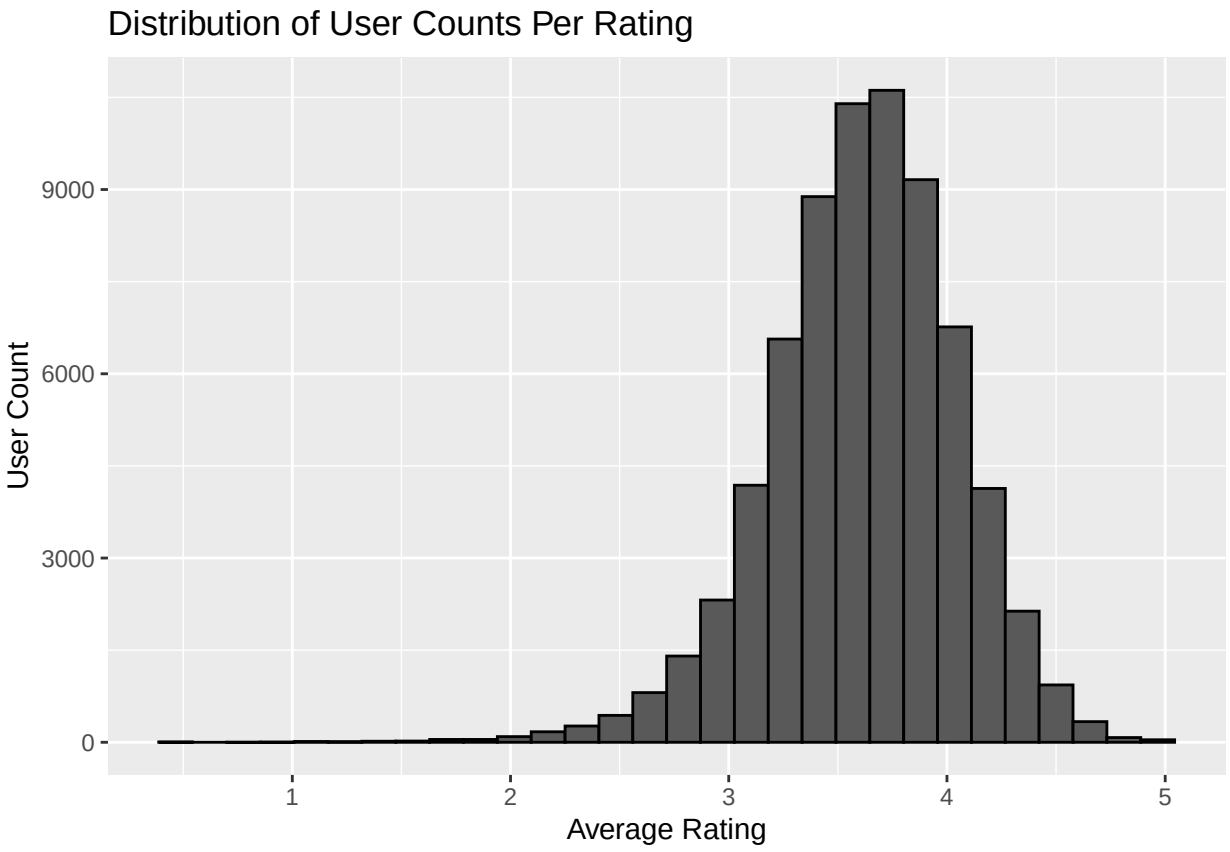
After the movie effect is added into the predictive model, the RMSE is decreased from 1.0612 to 0.9439. In other words, the model can predict better movie rating now.

```
predicted_rating <- mu_hat + validation %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_m

model_1_rmse <- RMSE(predicted_rating, validation$rating)

rmse_results <- bind_rows(rmse_results,
                          data_frame("Predictive Method"="Movie Effect Model",
                                     RMSE = model_1_rmse ))
rmse_results %>% knitr::kable()
```

Predictive Method	RMSE
Just Global Average	1.0612018
Movie Effect Model	0.9439087

Section 3.3.3. Modeling User Effects

From the following graph, it is noted that different user rates the movie differently. There is substantial variability across users as well: some users love the movie very much while some of them are very rigorous. This implies that the previous model can be further enhanced by adding user-specific effect as well.



Similar to movie-specific effect, the least squares estimate bias is just the average of (specific user's rating - average of all rating). Thus, the bias can be computed this way:

```
user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_hat - b_m))
```

After the movie effect and user effect are added into the predictive model, the RMSE is decreased from 0.9439 to 0.8653. The model performance is enhanced.

```
predicted_rating <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu_hat + b_m + b_u) %>%
  .$pred

model_2_rmse <- RMSE(predicted_rating, validation$rating)

rmse_results <- bind_rows(rmse_results,
                          data_frame("Predictive Method"="Movie + User Effects Model",
                                     RMSE = model_2_rmse ))

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

Predictive Method	RMSE
Just Global Average	1.0612018
Movie Effect Model	0.9439087
Movie + User Effects Model	0.8653488

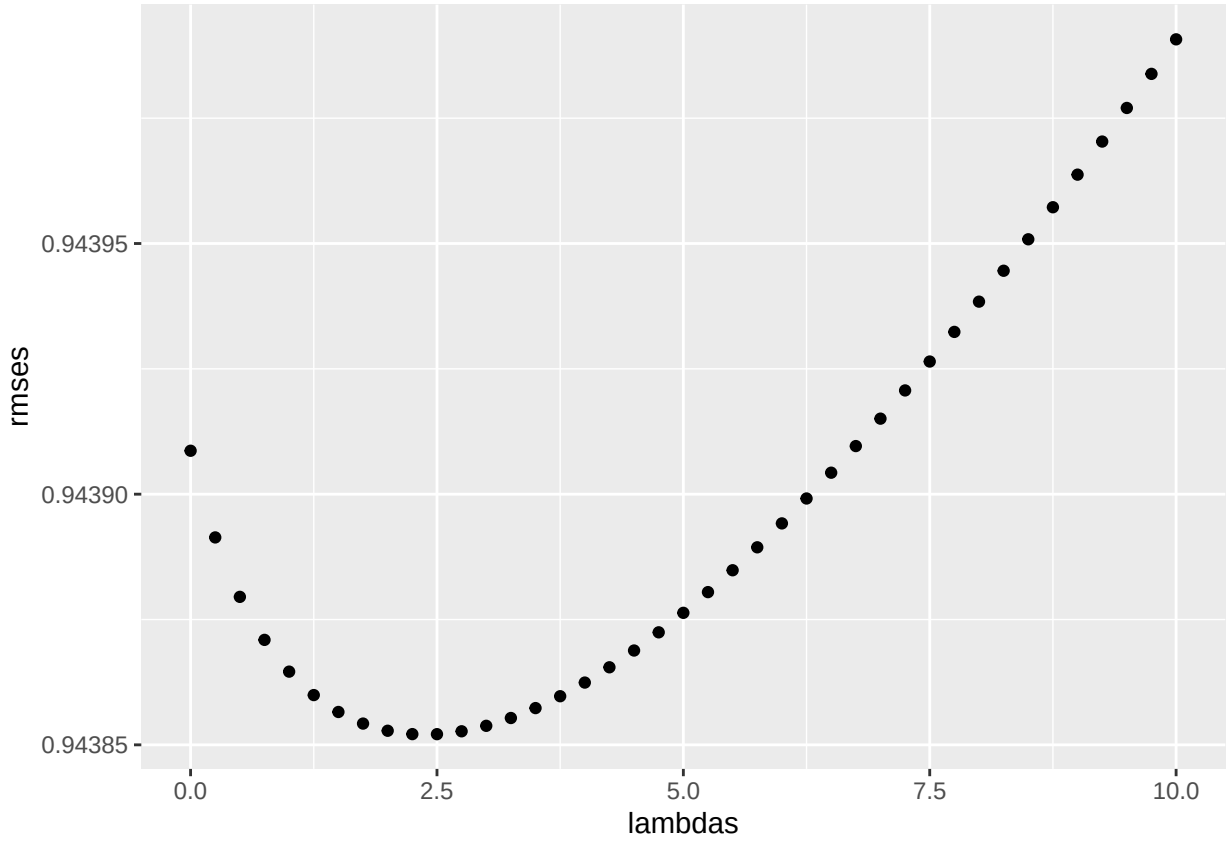
Section 3.3.4. Regularization for Movie Effect in the Model

In the previous section, the improvement in RMSE was still limited. One of the reasons is the large variation between movies. From *Section 2.2.3*, some movies were rated by very few users. There might have more uncertainty on the movie effect and it would cause a larger estimates of movie effect (either positive or negative). Those estimation will probably have a large error during modeling and even increase RMSE. For this, regularization is introduced.

Regularization is a technique used to reduce the error in prediction by adding a penality into the model to improve the function fitting on the training set and avoid overfitting. It penalizes large estimate generated by using the small dataset and control the total variability of the effect, i.e. movie effect in this case. When the movie effect is large, the penalty will also be increased.

Before computing the regularized estimate of movie effect, a penalty tuning parameter, so called lambda is required. In this project, cross validation is used on the training set to select the optimal lambda, which minimizes RMSE the most. From information below, the optimal lambda for the estimate movie effect is 2.5.

```
lambdas <- seq(0, 10, 0.25)

just_the_sum <- edx %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu_hat), n_i = n())

rmses <- sapply(lambdas, function(l){
  predicted_rating <- validation %>%
    left_join(just_the_sum, by='movieId') %>%
    mutate(b_m = s/(n_i+l)) %>%
    mutate(pred = mu_hat + b_m) %>%
    .$pred
  return(RMSE(predicted_rating, validation$rating))
})
qplot(lambdas, rmses)
```




```
lambdas[which.min(rmses)]
```

```
## [1] 2.5
```

To see how the model is improved using regularized estimate of movie effect with lambda = 2.5, the following code is used.The RMSE is dropped to 0.8652.

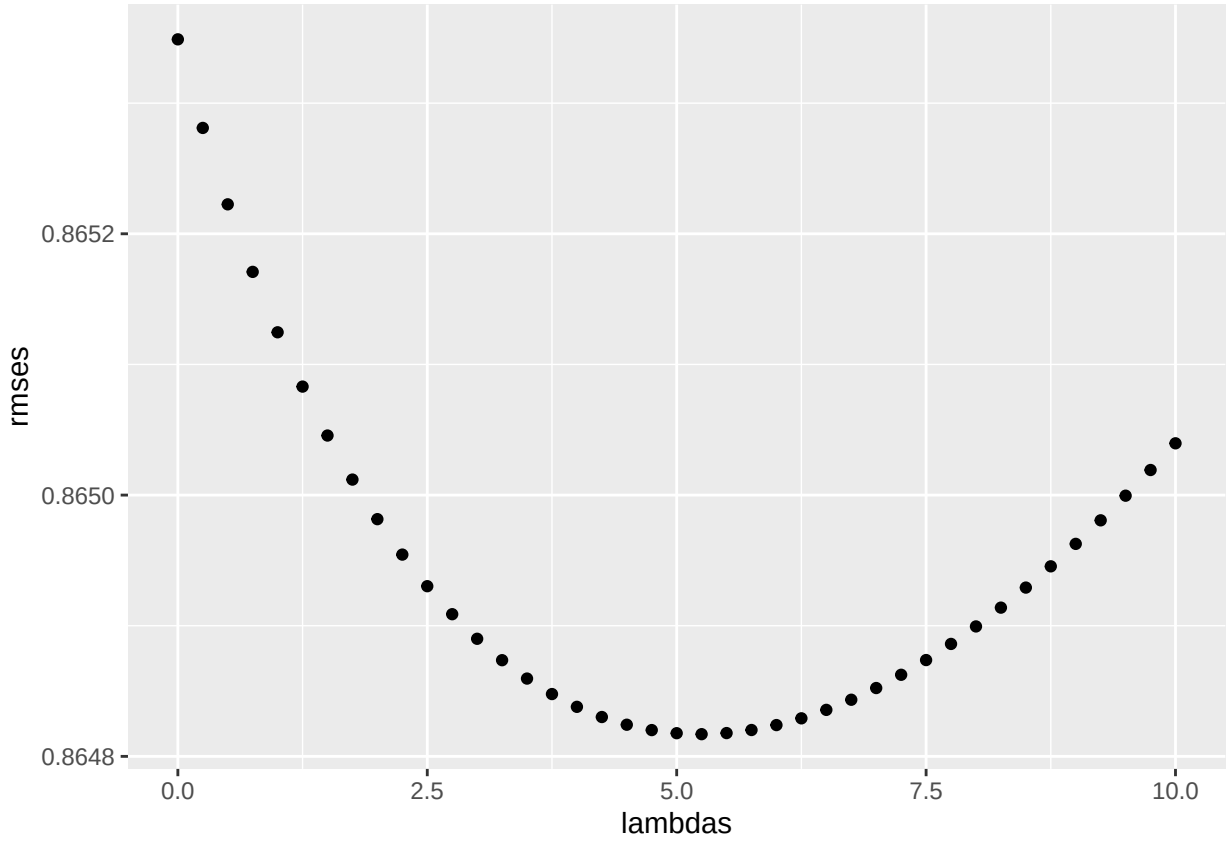
```
lambda <- 2.5
movie_reg_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_m2 = sum(rating - mu_hat)/(n()+lambda), n_i = n())

predicted_rating <- validation %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu_hat + b_u+ b_m2) %>%
  .$pred

model_3_rmse <- RMSE(predicted_rating, validation$rating)

rmse_results <- bind_rows(rmse_results,
                          data_frame("Predictive Method"="Regularized Movie + User Effects Model",
                                     RMSE = model_3_rmse ))

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

Predictive Method	RMSE
Just Global Average	1.0612018
Movie Effect Model	0.9439087
Movie + User Effects Model	0.8653488
Regularized Movie + User Effects Model	0.8652263

Section 3.3.5. Regularization for User Effect in the Model

User effect can be regularized as well because there is large user to user variation (see *Section 2.2.2* for details). Lambda parameter is further selected using the following code. The optimal lambda for the full model is 5.25.

```
lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas, function(l){

  mu_hat <- mean(edx$rating)

  b_m <- edx %>%
    group_by(movieId) %>%
    summarize(b_m = sum(rating - mu_hat)/(n()+l))

  b_u <- edx %>%
    left_join(b_m, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_m - mu_hat)/(n()+l))

  predicted_rating <-
    validation %>%
    left_join(b_m, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu_hat + b_m + b_u) %>%
    .$pred

    return(RMSE(predicted_rating, validation$rating))
})

qplot(lambdas, rmses)
```




```
lambdas[which.min(rmses)]
```

```
## [1] 5.25
```

Finally, the RMSE is 0.8648 according to the code below. It implies that regularization enhanced the performance of predictive model.

```
lambda <- 5.25

movie_reg_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_m2 = sum(rating - mu_hat)/(n()+lambda), n_i = n())

user_reg_avgs <- edx %>%
  left_join(movie_reg_avgs, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u2 = sum(rating - mu_hat-b_m2)/(n()+lambda), n_i = n())

predicted_rating <- validation %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  left_join(user_reg_avgs, by='userId') %>%
  mutate(pred = mu_hat + b_m2 + b_u2) %>%
  .$pred

model_4_rmse <- RMSE(predicted_rating, validation$rating)
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame("Predictive Method"="Regularized Movie + Regularized User Effects M
                                     RMSE = model_4_rmse ))

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

Predictive Method	RMSE
Just Global Average	1.0612018
Movie Effect Model	0.9439087
Movie + User Effects Model	0.8653488
Regularized Movie + User Effects Model	0.8652263
Regularized Movie + Regularized User Effects Model	0.8648170

Section 4. Results

After regularizing movie effect and user effect, the RMSE is decreased from 1.0612 to 0.8648. This indicates that the model can predict better than using the global rating average.

```
data_frame("Final Result" ="Regularized Movie + Regularized User Effects Model",RMSE = model_4_rmse ) %
```

Final Result	RMSE
Regularized Movie + Regularized User Effects Model	0.864817

Section 5. Conclusion

Section 5.1. Summary

In this project, we learned how to build a movie recommendation system by considering global average, movie-movie similarity and user-user similarity. We then used regularization for movie effect and user effect to enhance the system. The RMSE is largely dropped from 1.0612 to 0.8648. It indicates that the recommendation system performs well.

Section 5.2. Limitation

Section 5.2.1. Limited Dataset

As mentioned earlier, only a subset of MovieLens dataset is used to build the movie recommendation system. In order to have more accurate model, full dataset is suggested to be used for modeling.

Section 5.2.2. No Past Data for New Movie/User

Since the above predictive model used the past rating record to predict the user's preference so as to recommend movie for each user, no information can be used if new user and/or new movie is involved.

Section 5.3. Future work

Section 5.3.1. Cross-Validation

In this project, the dataset is only split into training and validation datasets once to build the movie recommendation model. In fact, the data set can be repeatedly split into the above two datasets, so called cross-validation, so as to utilize all significant data for training and get better and more stable prediction.

Section 5.3.2. Additional Variable In order to solve the potential issue mentioned in *Section 5.2.2*, the recommendation sysyem can be improved by considering additional variable during model training. For example, user's information (e.g. address, age, gender, device) can be used for new user. For new movies, the model can inlude the variables such as genre, cast, and crew.