MovieLens Project

Juan Du

2022-08-13

1 Introduction

This document is the MovieLens project in the course HarvardX: ph125.9x Data Science: Capstone project.

1.1 The MovieLens Dataset

The version of movieLens 10M dataset used for this project generated by the Grouplens research lab. The rawdata can be found and downloaded here: 10M version of the movieLens dataset

We divide the dataset into two parts: edx dataset and validation dataset. The edx dataset is used to analyze the predictors, find insights and develop the recommendation system; The validation dataset is used to validate effectiveness of our developed recommendation system. We are going to train our algorithms using the edx dataset and predict move ratings in the validation dataset.

1.2 The Goal of the Project

For this project, the aim to create a movie recommendation system to predict movie rating using MovieLens dataset. Our prediction models are scored against the true grade in the form of root mean squared error (RMSE), and the goal is to reduce the error as much as possible. And for movies with a predicted high rating for a given user are then recommendation to that user.

1.3 Key Steps of the Project

This project includes data ingestion, data exploration and analysis, method description, results and conclusion. The prediction algorithms used in this project generally follows the models used in the course[1].

2 Method and Analysis

2.1 Netflix Challenge and RMSE

One of the most famous success story of the recommend system is the Netflix Prize competition launched on October 2006.

The Netflix Prize was an open competition for the best collaborative filtering algorithm to predict user ratings for films, based on previous ratings without any other information about the users or films, i.e. without the users being identified except by numbers assigned for the contest[2].

The Netflix challenge used the typical error loss: they decided on a winner based on the residual mean squared error(RMSE) on the validation set. RMSE is similar to a standard deviation: it is the typical error

we make when predicting a move rating[3]. Submitted predictions are scored against true grades in terms of root mean squared error (RMSE), and the goal is to reduce their error as much as possible.

To achieve the best score in this project, we have to get an RMSE under 0.86490.

2.2 Data Ingestion

```
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 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.9
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     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)
library(ggplot2)
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:data.table':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
library(dslabs)

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

dl <- tempfile()
download.file("https://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")

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)
```

2.3 The Basic Summary of The dataset

```
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1:                Comedy|Romance
## 2:          Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

The edx dataset includes 6 variables: userId, movieId, rating, timestamp, titles and genres.

- userId and movieId are integers
- rating is an integer, are from 1 to 5 stars and including half-star
- timestamp is represneted in second since 1/1/1970 UTC
- genres is character, represents the genre

```
# number of rows and columns are there in the edx dataset
dim(edx)
```

```
## [1] 9000055       6
```

```
# number of rows and columns are there in the validation dataset
dim(validation)
```

```
## [1] 999999      6
```

We can see the edx datset includes total 9000055 observations. The validation dataset includes the same variables, with 999999 observations, which is the 10% of the 10M MovieLens dataset.

Each row represents a rating given by one user to one movie

```
# number of unique users that provided ratings and number of unique movies were rated in edx dataset
edx %>% summarise(n_users = n_distinct(userId),
                  n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   69878    10677
```

There are total 69,878 unique users and total 10,677 unique movies in edx dataset. if every user rates every movie, it should have had 746M observations (69878 x 10677),while our edx dataset has 9M observation, which means this large matrix contains many empty cells.

This machine learning challenge is complicated because each outcome Y has a different set of predictors. If we are predicting the rating for movie i by user u, in principle, all other ratings related to movie i and by user u may be used as predictors, but different users rate different movies and a different number of movies[4].

2.4 Rating Distribution

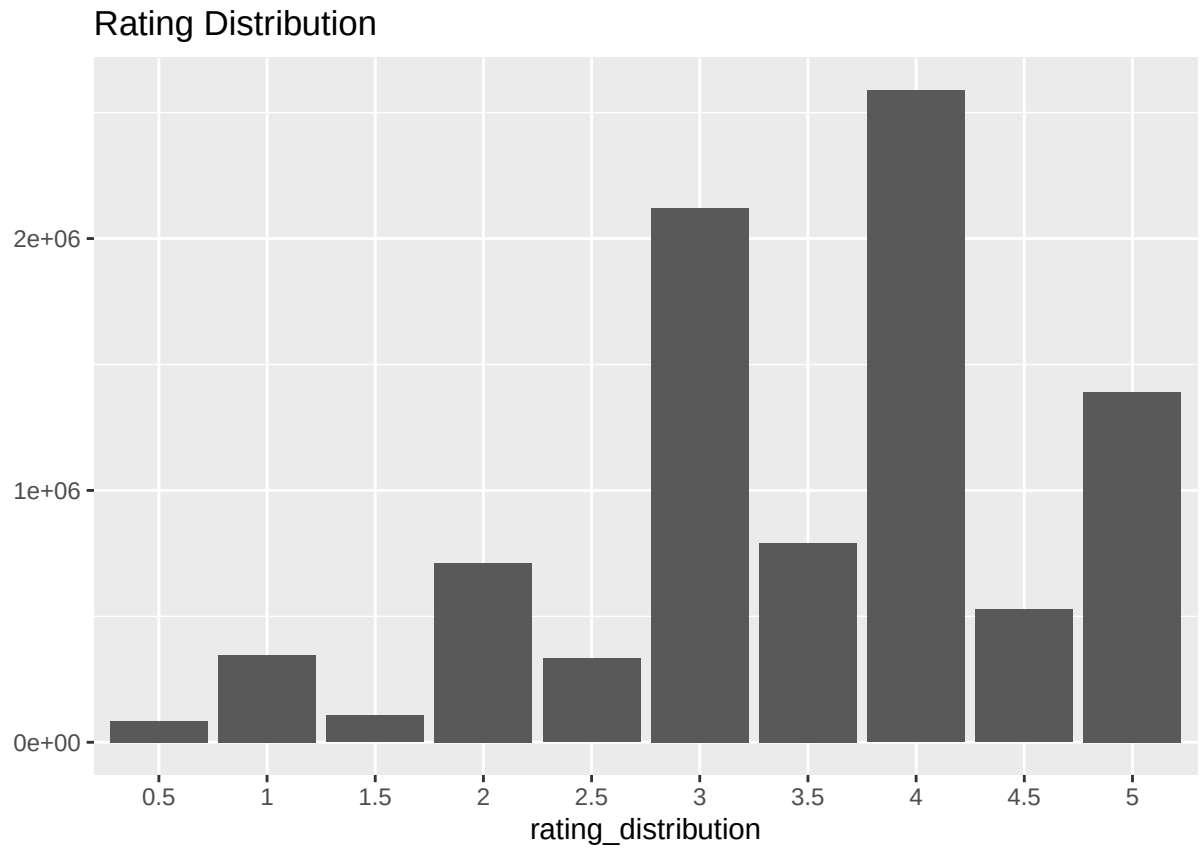
```
# Average rating in edx dataset
mean(edx$rating)
```

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

```
# Rating distribution
rating_distribution <- as.vector(edx$rating)
unique(rating_distribution)
```

```
##  [1] 5.0 3.0 2.0 4.0 4.5 3.5 1.0 1.5 2.5 0.5
```

```
# Visualize rating distribution
rating_distribution <- rating_distribution[rating_distribution !=0]
rating_distribution <- factor(rating_distribution)
qplot(rating_distribution) + ggtitle("Rating Distribution ")
```



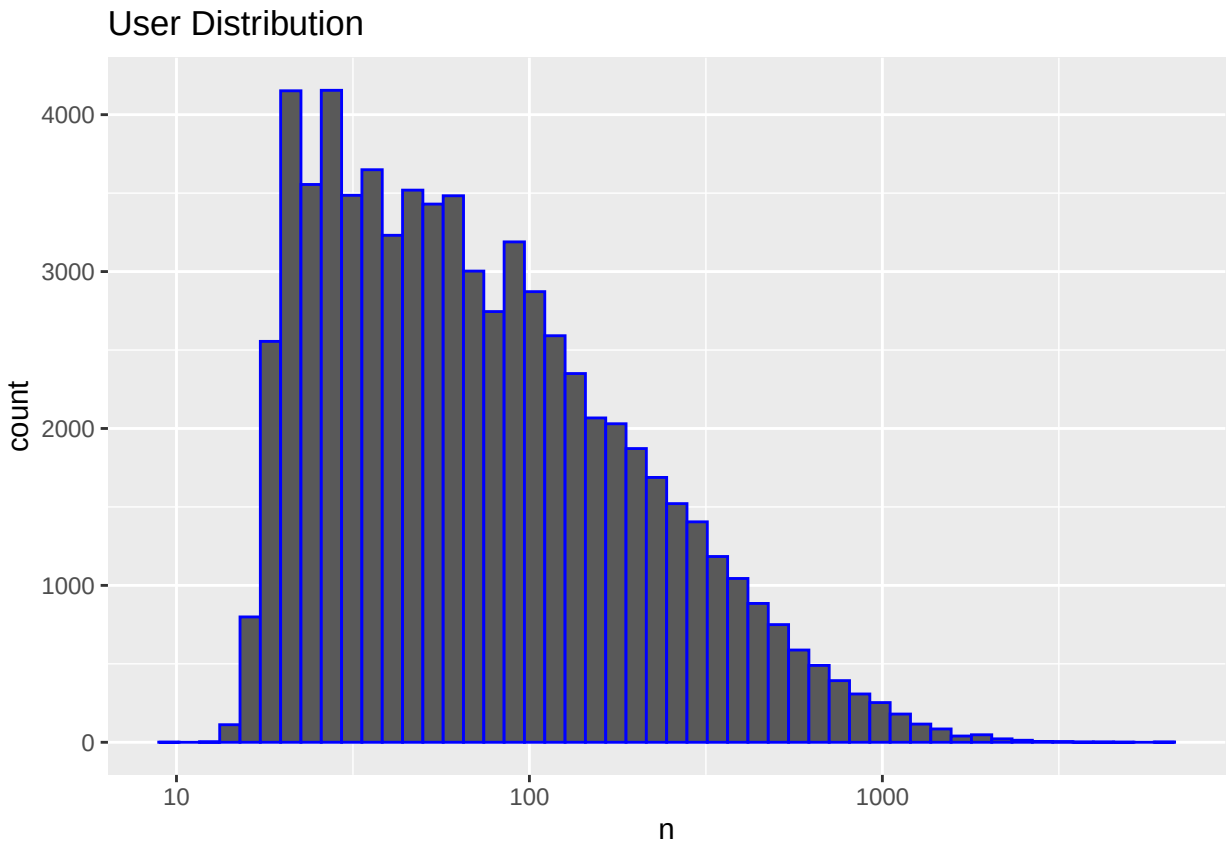
The minimum rating rated to any movie is 0.5 and the maximum rating rated is 5.0.

The summary of rating is as below:

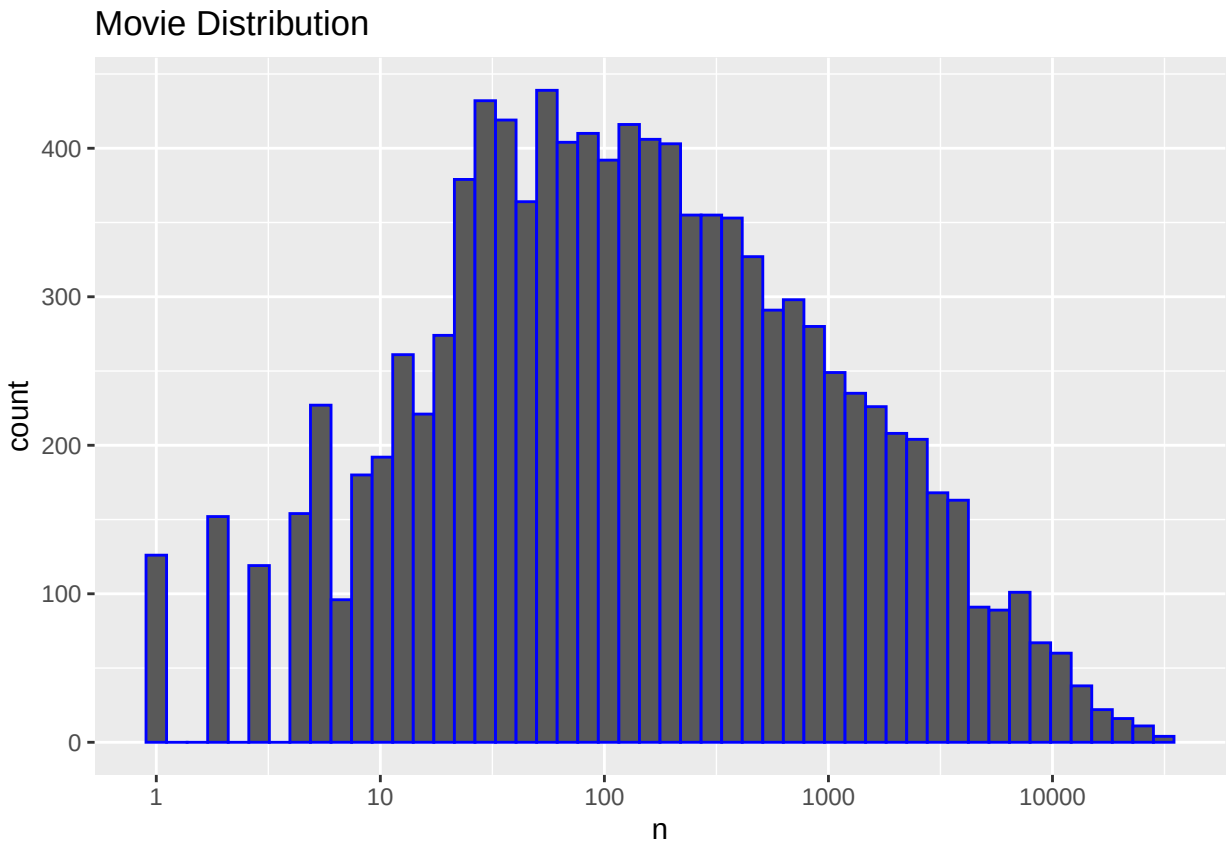
- The average rating in edx dataset is 3.51.
- The distribution shows that users are more likely to rate movie either 4 or 3.
- Whole ratings are more common than half-point ratings

2.5 User Distribution and movie distribution

```
# User distribution
edx %>% count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 50, color="blue") +
  scale_x_log10() +
  ggtitle("User Distribution")
```




```
# Movie Distribution
edx %>% count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 50, color="blue") +
  scale_x_log10()+
  ggtitle("Movie Distribution")
```



The exploration of user and movie distribution shows that some users are more active than others at rating movies; and some movies watched much more than others.

2.6 The concept of Regularization

As a lot of movies are rated by very few users, which can bring a lot of uncertainty for our prediction. In order to reduce the standard error or RMSE, we introduce the concept of regularization. Regularization can allow us to penalize large estimates which using small sample size.

Same problem is a lot of users are only rating few movies.

We will use regularization to constrain the total variability of the effect sizes.

2.7 Model Building Strategy - Method description

We will use the methodologies we learned in the course.

- Using mean only
- Movie Effect
- Movie and user effect model
- Regularization movie-effect model
- Regularization movie and user effect model

```
# Build a function that computes the RMSE for vectors of ratings and their predictors
RMSE <- function(true_ratings, predicted_ratings){
```

```
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

3 Recommendation System Modelling and Results

3.1 Model 1: Just Mean Average Model

First to build the simplest recommendation system: we predict the same rating for all movies.

```
# Calculate Mean for all movie ratings
mu_hat <- mean(edx$rating)
mu_hat
```

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

```
# Predict all unknown ratings with mu_hat and calculate RMSE
mu_rmse <- RMSE(validation$rating, mu_hat)
mu_rmse
```

```
## [1] 1.061202
```

```
# Save the Model 1 results
rmse_results <- data_frame(method = "Just the mean average",
                                    RMSE = mu_rmse)
```

```
## Warning: `data_frame()` was deprecated in tibble 1.1.0.
## Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
```

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

method	RMSE
Just the mean average	1.061202

The RMSE for "Just the mean average" model is 1.06, so we have to improve our model.

3.2 Model 2: Movie-effect Model

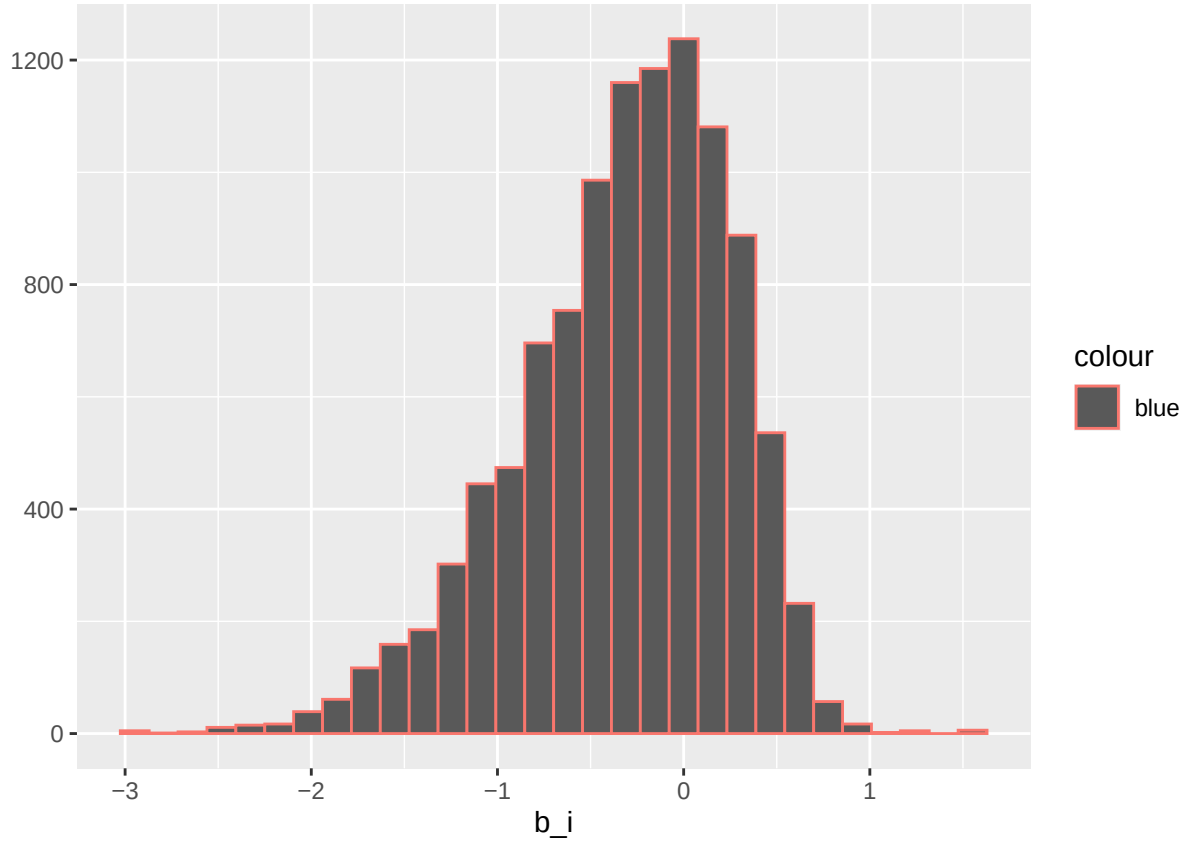
As some movies are generally rated higher than others, so we add the term b_i to represent average ranking bias for movie i.

```
# Calculate Movie-effect b_i
mu <- mean(edx$rating)
movie_avgs <- edx %>%
  group_by(movieId) %>%
```

```
  summarise(b_i = mean(rating - mu))

# Plot movie rating variability
qplot(b_i, data = movie_avgs, bin = 30, color = "blue")
```

```
## Warning: Ignoring unknown parameters: bin
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
# Predict movie-effect model and calculate RMSE
predicted_ratings <- mu + validation %>%
  left_join(movie_avgs, by = 'movieId') %>%
  pull(b_i)
bi_rmse <- RMSE(predicted_ratings, validation$rating)
bi_rmse
```

```
## [1] 0.9439087
```

```
# Save the Model 2 result
rmse_results <- bind_rows(rmse_results, data_frame(method = "Movie-effect model", RMSE =bi_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the mean average	1.0612018
Movie-effect model	0.9439087

The RMSE for movie-effect Model is 0.9439.

3.3 Model 3: Movie and User Effect Model

As some users are very picky and some users are very generous when rank movies, so we add b_u to reflect user-specific effect.

```
# Compute mu and b_i and estimate b_u
user_avgs <- edx %>%
  left_join(movie_avgs, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu -b_i))

#Construct predictors
predicted_ratings <- validation %>%
  left_join(movie_avgs, by = 'movieId') %>%
  left_join(user_avgs, by = 'userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

#calculate RMSE
bu_rmse <- RMSE(predicted_ratings, validation$rating)
bu_rmse
```

```
## [1] 0.8653488
```

```
# Save the model 3 RMSE results
rmse_results <- bind_rows(rmse_results,
                          data_frame(method = "Movie and user effect model",
                                     RMSE = bu_rmse))
rmse_results %>% knitr::kable()
```

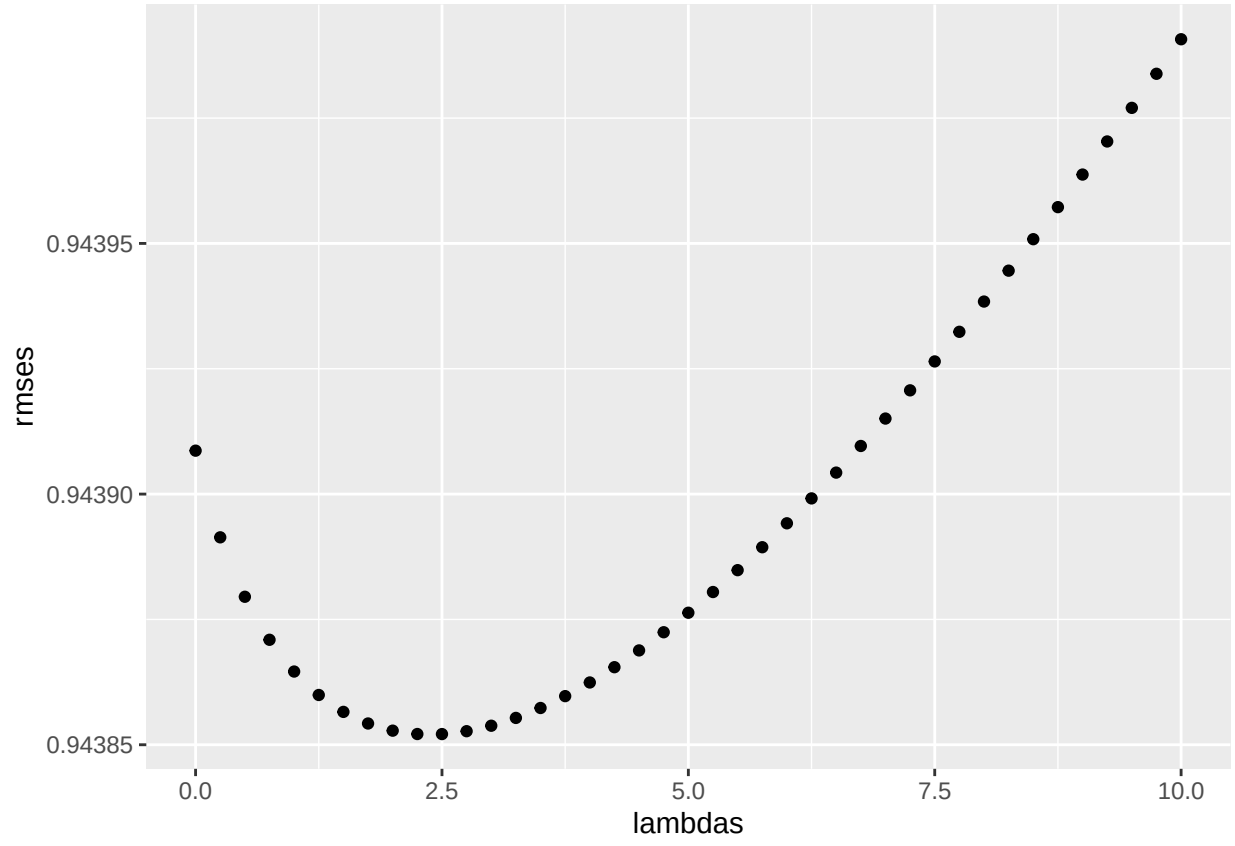
method	RMSE
Just the mean average	1.0612018
Movie-effect model	0.9439087
Movie and user effect model	0.8653488

The RMSE for Movie and user effect model is 0.8653, we get improved result.

3.4 Model 4: Regularized Movie-effect Model

Firstly, let's choose penalty terms for movie-effect model, based on some movies are only rated by few users.

```r
# Calculate optimal lambda
lambdas <- seq(0, 10, 0.25)

mu <- mean(edx$rating)

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

rmses <- sapply(lambdas, function(l){
  predicted_ratings <- validation %>%
    left_join(just_the_sum, by = 'movieId') %>%
    mutate(b_i = s/(n_i +l)) %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)
return(RMSE(predicted_ratings, validation$rating))
  })

qplot(lambdas, rmses)
```




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

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

```r
# Calculate regularized movie-effect RMSE
movie_reg_avgs <- edx %>%
  group_by(movieId) %>%
  summarise(b_i = sum(rating - mu)/ (n() + lambda), n_i=n())

predicted_ratings <- validation %>%
  left_join(movie_reg_avgs, by = 'movieId') %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)

# calculate Model 4 RMSE
movie_reg_rmse <- RMSE(predicted_ratings, validation$rating)

# Save the Model 4 RMSE result
rmse_results <- bind_rows(rmse_results,
                          data_frame(method = "Regularized movie-effect model",
                                     RMSE = movie_reg_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the mean average	1.0612018
Movie-effect model	0.9439087
Movie and user effect model	0.8653488
Regularized movie-effect model	0.9438521

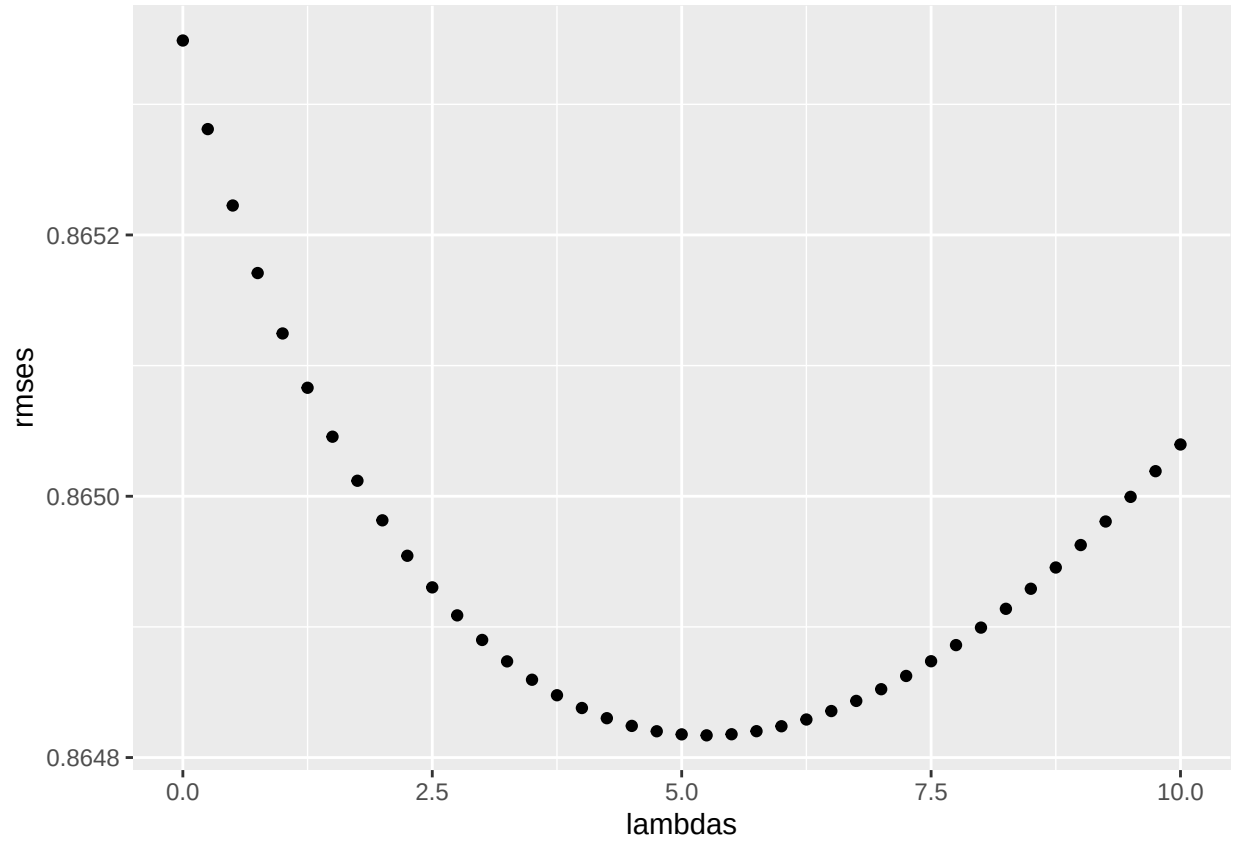
RMSE for the penalized movie-effect model is 0.94, we still need to improve our model

3.5 Model 5: Regularized Movie and User effect Model

Let's use penalty to constrain the cases both users and movies with smaller sizes

```r
# Calculate optimal lambda
lambdas <- seq(0, 10, 0.25)

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

  mu <- mean(edx$rating)

  b_i <- edx %>%
    group_by(movieId) %>%
    summarise(b_i = sum(rating - mu)/ (n() + l))

  b_u <- edx %>%
    left_join(b_i, by = 'movieId') %>%
    group_by(userId) %>%
    summarise(b_u = sum(rating - b_i - mu) / (n() +l ))

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

```
    pull(pred)

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

qplot(lambdas, rmses)
```




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

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

```
# Calculate regularized movie and user effect RMSE

movie_reg_avgs <- edx %>%
  group_by(movieId) %>%
  summarise(b_i = sum(rating - mu) /(n() + lambda), n_i = n())

user_reg_avgs <- edx %>%
  left_join(movie_reg_avgs, by = 'movieId') %>%
  group_by(userId) %>%
  summarise(b_u = sum(rating - b_i - mu)/(n() +lambda), n_u = n())

predicted_ratings <- validation %>%
```

```
  left_join(movie_reg_avgs, by = 'movieId') %>%
  left_join(user_reg_avgs, by = 'userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

# Calculate model 5 RMSE
movie_user_reg_rmse <- RMSE(predicted_ratings, validation$rating)

# Save the result
rmse_results <- bind_rows(rmse_results,
                          data_frame(method = "Regularized movie and user effect model",
                                     RMSE = movie_user_reg_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the mean average	1.0612018
Movie-effect model	0.9439087
Movie and user effect model	0.8653488
Regularized movie-effect model	0.9438521
Regularized movie and user effect model	0.8648170

4 Conclusion

This MovieLens project has tried 5 different recommendation system algorithms to predict move ratings for the 10M MovieLens dataset. The RMSE table shows an improvement of the 5 models over different assumptions, and the Regularized movie and user effect model achieved the best result with the smallest RMSE 0.8648.

The limitations of the project is that it hasn't covered genres-effect and release year-effect analysis.

The future work will focus on more additional baseline predictors analysis such as number of days since the user first rating, number of days since the movie's first rating, number of people rating the movie and the movie's overall rating, etc.

5 References

[1] Rafael A. Irizarry, Introduction to Data Science

[2] Wikipedia, Netflix Prize

[3] Rafael A. Irizarry, Introduction to Data Science, 34.7.3

[4] Rafael A. Irizarry, Introduction to Data Science, 34.7.1

- Reza Hashmemi, https://www.rpubs.com/rezapci/MovieLens