MovieLens Project Sophia Rao

sophia rao

2023-12-04

Introduction

The goal of this project is to create a movie recommendation system using machine learning algorithms. In order to do this, we use the MovieLens 10M dataset. The dataset contains 10,681 movies and 69,878 different users. Each row of the dataset contains the characteristics for a certain movie as well as the rating a specific user gave to the movie. To determine which factors affect movie rating for a given user, we can look at the correlation between the rating a user has given to a specific movie and another characteristic of the movie.

Methods

We download the data and split it into test and training sets. Next, we manipulated the training data into a format that was easier to work with. Most notably, we changed the dates from a Linux format to a human readable format. We created a new column that contained the age of the movie when it was rated by extracting the release year and the rating year and subtracting the two. For the genres, we separated them as most movies belong to multiple genres.

To start, we load the essential packages needed to complete the project.

```
library(dplyr)
library(tidyverse)
library(kableExtra)
library(tidyr)
library(stringr)
library(forcats)
library(ggplot2)
library(caret)
library(Hmisc)
library(data.table)
library(recommenderlab)
library(vioplot)
library(plyr)
library(plotly)
library(hrbrthemes)
library(Metrics)
library(lubridate)
library(recosystem)
```

To start, we downloaded and prepared the data using code provided by the course instructors. It splits the data into training and validation sets. The validation set is 10% of the MovieLens data.

```
############################################################
# Create edx and final_holdout_test sets
############################################################

# Note: this process could take a couple of minutes

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

library(tidyverse)
library(caret)

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

options(timeout = 120)

dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE),
                         stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)

colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

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

# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
```

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

```
# set.seed(1) # if using R 3.5 or earlier
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 final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
```

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

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

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

Before diving in, I view the first few lines to get an overview of the dataset.

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                         genres
## 1                Comedy|Romance
## 2          Action|Crime|Thriller
## 4   Action|Drama|Sci-Fi|Thriller
## 5        Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7        Children|Comedy|Fantasy
```

Next, I converted the time stamp to a human readable date.

```
##   userId movieId rating timestamp                 title release
## 1      1     122      5 838985046              Boomerang    1992
## 2      1     185      5 838983525               Net, The    1995
## 4      1     292      5 838983421               Outbreak    1995
## 5      1     316      5 838983392               Stargate    1994
## 6      1     329      5 838983392 Star Trek: Generations    1994
## 7      1     355      5 838984474       Flintstones, The    1994
##                         genres                date yearOfRating monthOfRating
## 1                Comedy|Romance 1996-08-02 04:24:06         1996            08
## 2          Action|Crime|Thriller 1996-08-02 03:58:45         1996            08
## 4   Action|Drama|Sci-Fi|Thriller 1996-08-02 03:57:01         1996            08
## 5        Action|Adventure|Sci-Fi 1996-08-02 03:56:32         1996            08
## 6 Action|Adventure|Drama|Sci-Fi 1996-08-02 03:56:32         1996            08
## 7        Children|Comedy|Fantasy 1996-08-02 04:14:34         1996            08
```

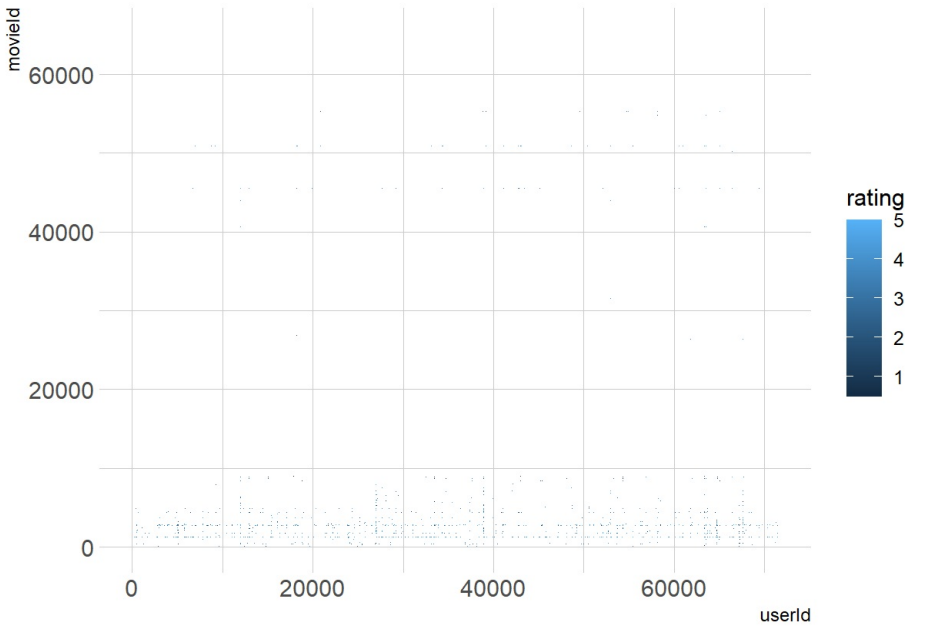
Here we separated the genres.

```
# Extract the genre in edx dataset

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

Here we calculate the “age” of the movie at the time of rating.

```
edx <- edx %>% mutate(yearsSinceRelsease = as.numeric(yearOfRating)-as.numeric(release))
```

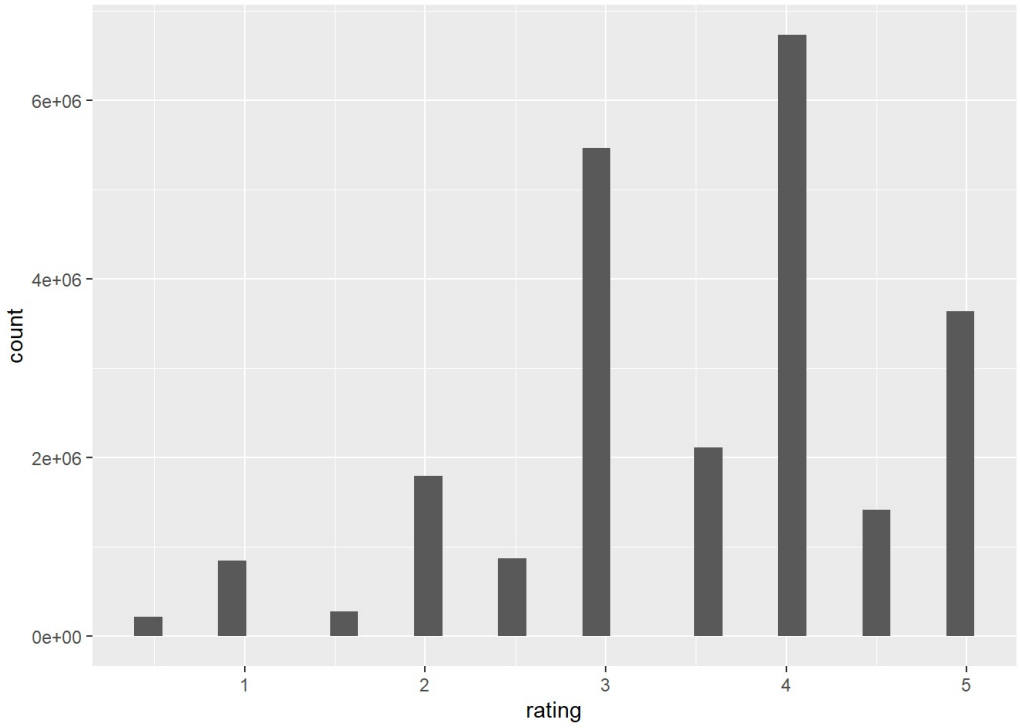
Here we plotted a heat map of the rating each user gave to a specific movie. By looking at this graph, we can easily see gaps in the data. Empty spaces mean that no rating was given for a movie.



By Plotting the frequency of each

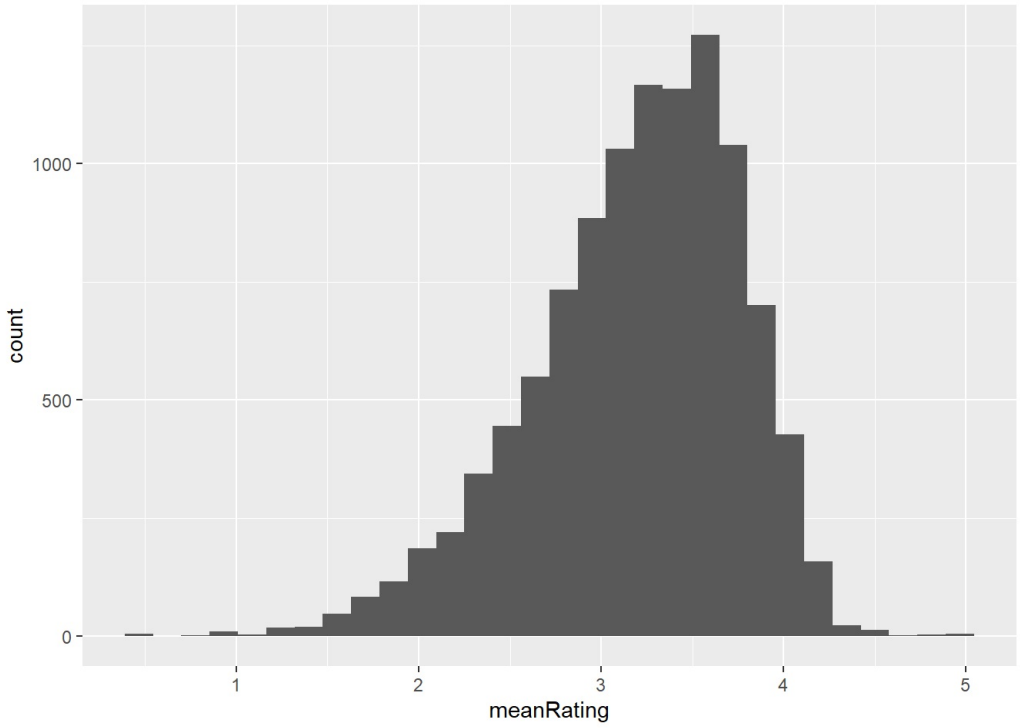
rating, we can clearly see that the data is left-skewed. This means that users tend to give higher ratings. Another takeaway from this graph is that we can see that users give whole star ratings significantly more often than half star ratings.

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



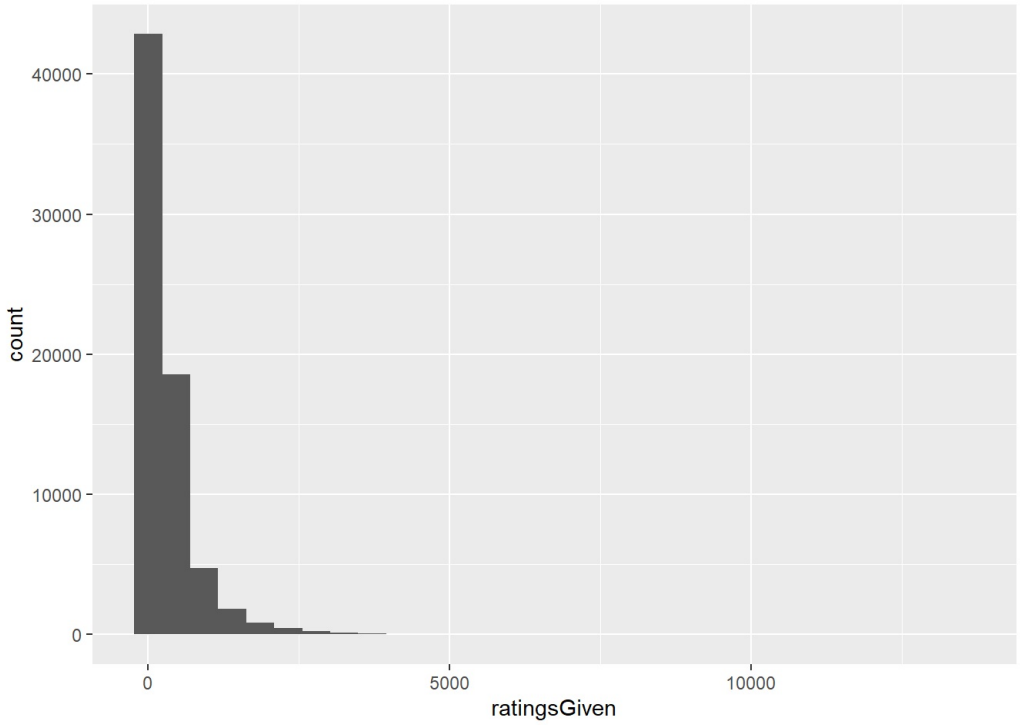
This plot shows the frequency of the average rating per movie. From this, we can see how popular the movies in the data base are with the audience. The higher the frequency of larger average ratings, the more movies offered from the database people like.

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



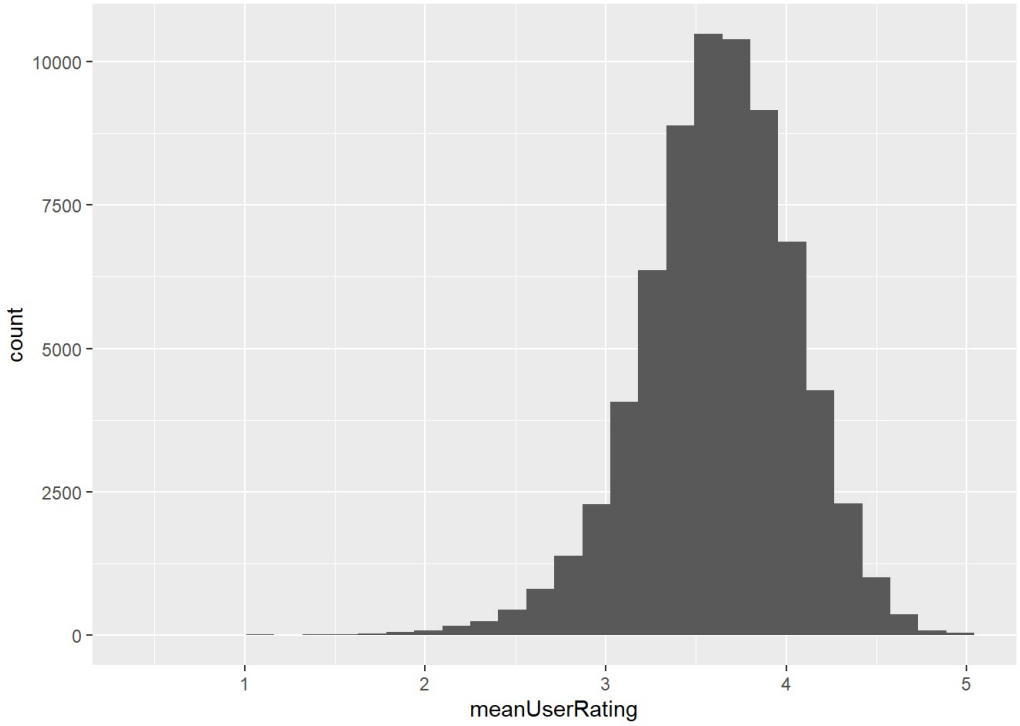
Here we plotted a frequency histogram of the number of ratings given by each user to get an idea of how interactive users are. We notice that most users do not give out very many ratings.

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



In this histogram, we plotted the frequency of mean user ratings. We used this plot to identify how many user's tastes are similar to others based on the mean of their ratings.

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



Results

RMSE stands for Residual Mean Squared Error. It tells you how concentrated the data is around the line of best fit.

```
#RMSE function
RMSE <- function(test, train){
  sqrt(mean((test - train)^2))
}
```

Model 1

In this model we first looked at the overall average of all the ratings.

```
mu = mean(edx$rating)
```

```
naive_rmse <- rmse(final_holdout_test$rating, mu)
naive_rmse
```

```
## [1] 1.061308
```

```
rmse_results <- tibble(Method = "Model 1: Simple overall average model", RMSE = naive_rmse)
rmse_results
```

```
## # A tibble: 1 × 2
##   Method                                RMSE
##   <chr>                                <dbl>
## 1 Model 1: Simple overall average model  1.06
```

Model 2

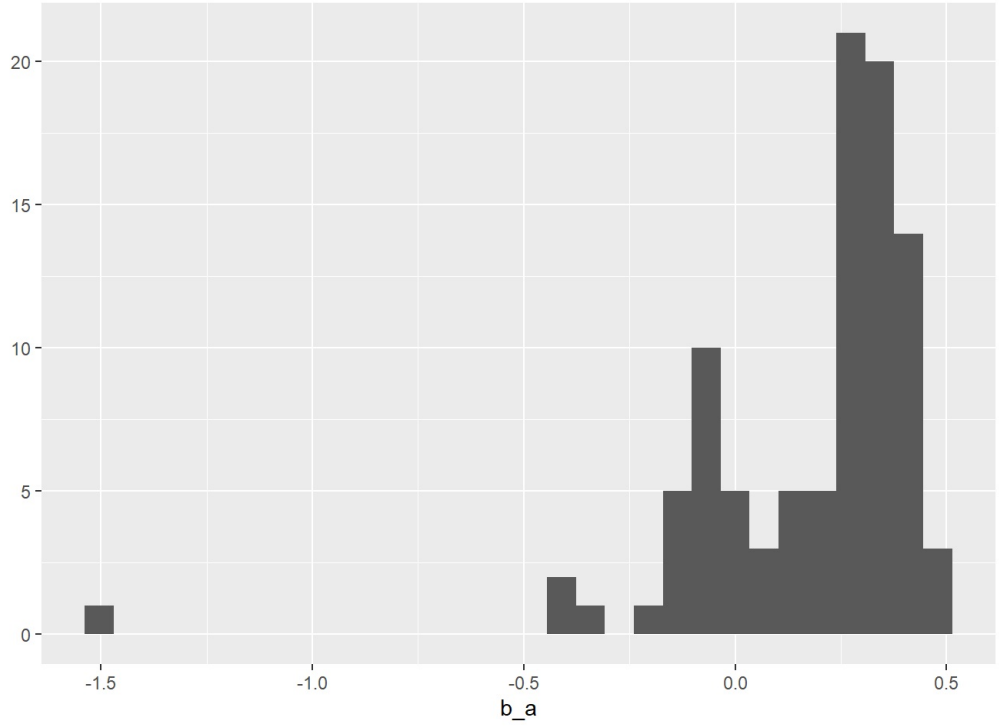
Here we added a bias of age to the model to see if that would better predict the ratings.

```
age_effect<- edx %>%
  group_by(yearsSinceRelsease) %>%
  summarise(b_a = mean(rating)-mu)
head(age_effect)
```

```
## # A tibble: 6 × 2
##   yearsSinceRelsease     b_a
##                <dbl>   <dbl>
## 1                 -2 -1.53
## 2                 -1 -0.417
## 3                  0 -0.0645
## 4                  1 -0.0236
## 5                  2 -0.0278
## 6                  3 -0.0434
```

```
age_effect %>% qplot(b_a, geom ="histogram",  data = .)
```

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



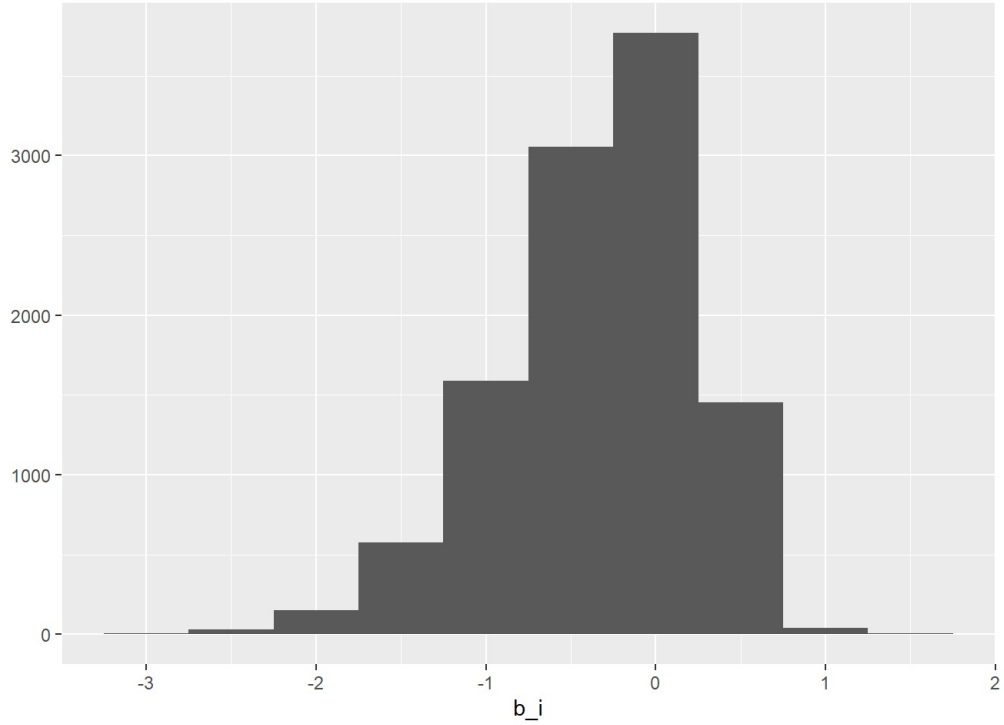

```
model_2_rmse <- RMSE(final_holdout_test$rating,mu) # 1.05239
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method="Age Effect Model",
                                     RMSE = model_2_rmse))
rmse_results
```

```
## # A tibble: 2 × 2
##   Method                                RMSE
##   <chr>                                <dbl>
## 1 Model 1: Simple overall average model  1.06
## 2 Age Effect Model                       1.06
```

Model 3

In this third model, we added the bias of movie. We are saying that for each movie, the average of the ratings on that specific movie will have a difference from the overall average rating of all movies.

```
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarise(b_i = mean(rating - mu))
movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = .)
```




```
head(movie_avgs)
```

```
## # A tibble: 6 × 2
##   movieId    b_i
##     <int>  <dbl>
## 1       1  0.401
## 2       2 -0.322
## 3       3 -0.380
## 4       4 -0.663
## 5       5 -0.458
## 6       6  0.288
```

```
predicted_ratings_3 <- mu + final_holdout_test %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

model_3_rmse <- RMSE(final_holdout_test$rating,na.omit(predicted_ratings_3))
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method="Movie Effect Model",
                                     RMSE = model_3_rmse))
rmse_results
```

```
## # A tibble: 3 × 2
##   Method                                RMSE
##   <chr>                                <dbl>
## 1 Model 1: Simple overall average model 1.06
## 2 Age Effect Model                      1.06
## 3 Movie Effect Model                    0.944
```

Model 4

Then, we added the bias of user (b_u) to the model. Users are the ones rating the movies, so it made sense to think they have an effect on the model.

```
user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarise(b_u = mean(rating - mu - b_i))

#head(user_avgs)

predicted_ratings_4 <- final_holdout_test %>%
  left_join(user_avgs, by='userId') %>%
  left_join(movie_avgs, by='movieId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

model_4_rmse <- RMSE(final_holdout_test$rating,na.omit(predicted_ratings_4))
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method="Movie + User Effects Model",
                                     RMSE = model_4_rmse))
rmse_results
```

```
## # A tibble: 4 × 2
##   Method                                RMSE
##   <chr>                                <dbl>
## 1 Model 1: Simple overall average model 1.06
## 2 Age Effect Model                      1.06
## 3 Movie Effect Model                    0.944
## 4 Movie + User Effects Model            0.867
```

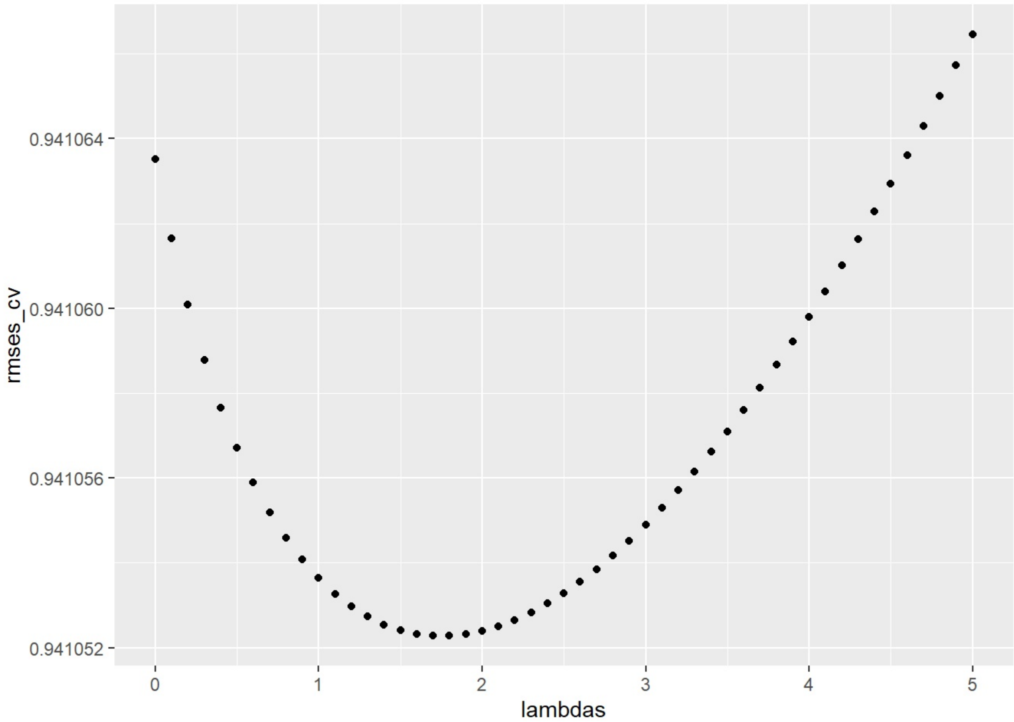
Model 5

Here, we performed regularization. This technique takes into account the movie bias so that the model did not end up over trained.

```
# use 10-fold cross validation to pick a lambda for movie effects regularization
# split the data into 10 parts
set.seed(2019, sample.kind = "Rounding")
cv_splits <- createFolds(edx$rating, k=10, returnTrain =TRUE)

# define a matrix to store the results of cross validation
rmses <- matrix(nrow=10,ncol=51)
lambdas <- seq(0, 5, 0.1)

# perform 10-fold cross validation to determine the optimal lambda
for(k in 1:10) {
  train_set <- edx[cv_splits[[k]],]
  test_set <- edx[-cv_splits[[k]],]

  # Make sure userId and movieId in test set are also in the train set
  test_final <- test_set %>%
    semi_join(train_set, by = "movieId") %>%
    semi_join(train_set, by = "userId")

  # Add rows removed from validation set back into edx set
  removed <- anti_join(test_set, test_final)
  train_final <- rbind(train_set, removed)

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

  rmses[k,] <- sapply(lambdas, function(l){
    predicted_ratings <- test_final %>%
      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, test_final$rating))
  })
}
```

```
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
```

```
rmses_cv <- colMeans(rmses)
qplot(lambdas,rmses_cv)
```




```
lambda <- lambdas[which.min(rmses_cv)]
lambda #2.2
```

```
## [1] 1.8
```

```
mu <- mean(edx$rating)
movie_reg_avgs <- edx %>%
  group_by(movieId) %>%
  summarise(b_i = sum(rating - mu)/(n()+lambda), n_i = n())
predicted_ratings_5 <- final_holdout_test %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)
model_5_rmse <- RMSE(na.omit(predicted_ratings_5), final_holdout_test$rating)   # 0.943852 not too much improved
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method="Regularized Movie Effect Model",
                                     RMSE = model_5_rmse))
rmse_results
```

```
## # A tibble: 5 × 2
##   Method                                RMSE
##   <chr>                                <dbl>
## 1 Model 1: Simple overall average model 1.06
## 2 Age Effect Model                     1.06
## 3 Movie Effect Model                   0.944
## 4 Movie + User Effects Model           0.867
## 5 Regularized Movie Effect Model       0.944
```

Model 6

We performed regularization again. This time we are took into account both the movie and user biases.

```
# define a matrix to store the results of cross validation
lambdas <- seq(0, 8, 0.1)
rmses_2 <- matrix(nrow=10,ncol=length(lambdas))
# perform 10-fold cross validation to determine the optimal lambda
for(k in 1:10) {
  train_set <- edx[cv_splits[[k]],]
  test_set <- edx[-cv_splits[[k]],]

  # Make sure userId and movieId in test set are also in the train set
  test_final <- test_set %>%
    semi_join(train_set, by = "movieId") %>%
    semi_join(train_set, by = "userId")

  # Add rows removed from validation set back into edx set
  removed <- anti_join(test_set, test_final)
  train_final <- rbind(train_set, removed)

  mu <- mean(train_final$rating)

  rmses_2[k,] <- sapply(lambdas, function(l){
    b_i <- train_final %>%
      group_by(movieId) %>%
      summarise(b_i = sum(rating - mu)/(n()+l))
    b_u <- train_final %>%
      left_join(b_i, by="movieId") %>%
      group_by(userId) %>%
      summarise(b_u = sum(rating - b_i - mu)/(n()+l))
    predicted_ratings <-
      test_final %>%
      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, test_final$rating))
  })
}
```

```
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
```

```
rmses_2
```

```
##            [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
##  [1,] 0.8571489 0.8571426 0.8571368 0.8571314 0.8571263 0.8571214 0.8571168
```

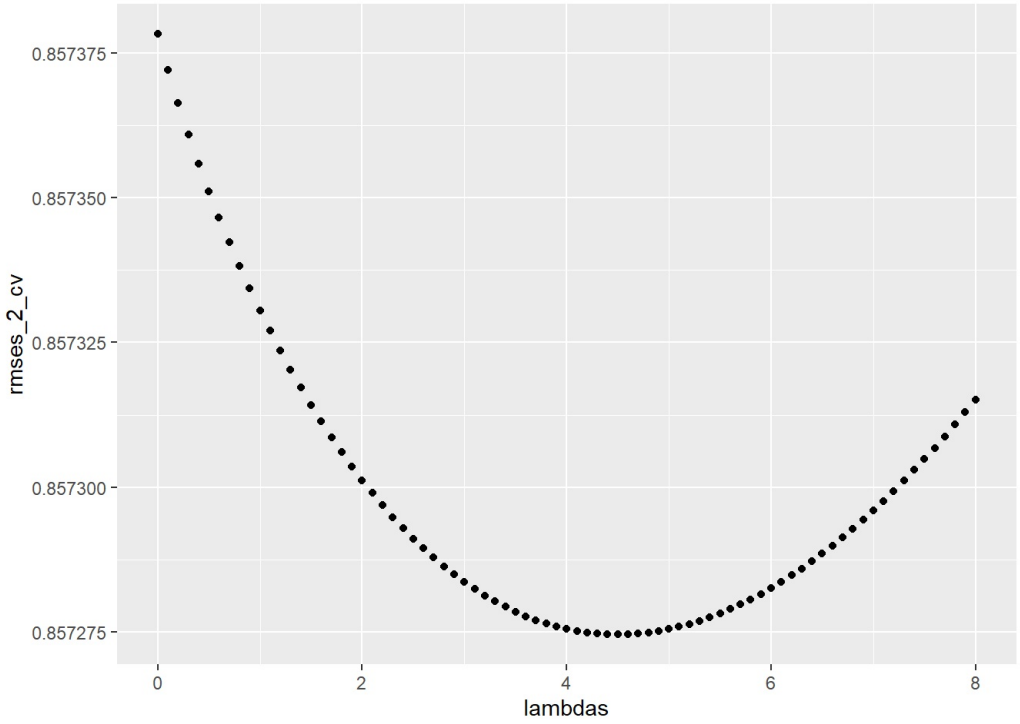
```
##  [2,] 0.8570938 0.8570879 0.8570824 0.8570773 0.8570725 0.8570680 0.8570637
##  [3,] 0.8565986 0.8565932 0.8565880 0.8565832 0.8565785 0.8565741 0.8565698
##  [4,] 0.8575226 0.8575160 0.8575100 0.8575044 0.8574991 0.8574941 0.8574894
##  [5,] 0.8571191 0.8571127 0.8571068 0.8571013 0.8570961 0.8570913 0.8570867
##  [6,] 0.8578513 0.8578452 0.8578395 0.8578341 0.8578291 0.8578242 0.8578196
##  [7,] 0.8574076 0.8574013 0.8573955 0.8573901 0.8573850 0.8573801 0.8573756
##  [8,] 0.8579127 0.8579060 0.8578998 0.8578941 0.8578888 0.8578838 0.8578791
##  [9,] 0.8576301 0.8576242 0.8576187 0.8576135 0.8576086 0.8576040 0.8575996
## [10,] 0.8574979 0.8574914 0.8574856 0.8574802 0.8574751 0.8574704 0.8574660
##           [,8]      [,9]     [,10]     [,11]     [,12]     [,13]     [,14]
##  [1,] 0.8571125 0.8571083 0.8571043 0.8571006 0.8570969 0.8570935 0.8570902
##  [2,] 0.8570596 0.8570557 0.8570520 0.8570485 0.8570451 0.8570419 0.8570388
##  [3,] 0.8565657 0.8565619 0.8565581 0.8565546 0.8565511 0.8565479 0.8565448
##  [4,] 0.8574849 0.8574806 0.8574765 0.8574726 0.8574689 0.8574653 0.8574619
##  [5,] 0.8570823 0.8570781 0.8570741 0.8570703 0.8570667 0.8570632 0.8570598
##  [6,] 0.8578152 0.8578109 0.8578069 0.8578030 0.8577993 0.8577957 0.8577923
##  [7,] 0.8573712 0.8573671 0.8573631 0.8573594 0.8573558 0.8573523 0.8573491
##  [8,] 0.8578746 0.8578704 0.8578664 0.8578626 0.8578590 0.8578555 0.8578522
##  [9,] 0.8575954 0.8575915 0.8575877 0.8575840 0.8575806 0.8575772 0.8575741
## [10,] 0.8574617 0.8574577 0.8574539 0.8574502 0.8574467 0.8574434 0.8574401
##          [,15]     [,16]     [,17]     [,18]     [,19]     [,20]     [,21]
##  [1,] 0.8570871 0.8570841 0.8570812 0.8570785 0.8570759 0.8570735 0.8570711
##  [2,] 0.8570359 0.8570331 0.8570304 0.8570279 0.8570255 0.8570232 0.8570210
##  [3,] 0.8565418 0.8565389 0.8565362 0.8565336 0.8565311 0.8565287 0.8565265
##  [4,] 0.8574586 0.8574554 0.8574524 0.8574496 0.8574468 0.8574442 0.8574417
##  [5,] 0.8570566 0.8570536 0.8570506 0.8570478 0.8570452 0.8570426 0.8570402
##  [6,] 0.8577890 0.8577858 0.8577828 0.8577799 0.8577772 0.8577745 0.8577720
##  [7,] 0.8573459 0.8573429 0.8573401 0.8573373 0.8573347 0.8573323 0.8573299
##  [8,] 0.8578491 0.8578461 0.8578432 0.8578405 0.8578379 0.8578355 0.8578331
##  [9,] 0.8575710 0.8575681 0.8575654 0.8575628 0.8575602 0.8575579 0.8575556
## [10,] 0.8574371 0.8574341 0.8574313 0.8574286 0.8574261 0.8574236 0.8574213
##          [,22]     [,23]     [,24]     [,25]     [,26]     [,27]     [,28]
##  [1,] 0.8570689 0.8570668 0.8570648 0.8570629 0.8570611 0.8570595 0.8570579
##  [2,] 0.8570189 0.8570170 0.8570151 0.8570134 0.8570117 0.8570102 0.8570088
##  [3,] 0.8565244 0.8565224 0.8565205 0.8565187 0.8565170 0.8565154 0.8565139
##  [4,] 0.8574394 0.8574371 0.8574350 0.8574329 0.8574310 0.8574292 0.8574274
##  [5,] 0.8570379 0.8570356 0.8570335 0.8570316 0.8570297 0.8570279 0.8570262
##  [6,] 0.8577696 0.8577673 0.8577651 0.8577631 0.8577611 0.8577592 0.8577575
##  [7,] 0.8573277 0.8573255 0.8573235 0.8573216 0.8573198 0.8573181 0.8573165
##  [8,] 0.8578309 0.8578288 0.8578268 0.8578249 0.8578231 0.8578214 0.8578198
##  [9,] 0.8575534 0.8575514 0.8575494 0.8575476 0.8575459 0.8575442 0.8575427
## [10,] 0.8574191 0.8574169 0.8574149 0.8574130 0.8574112 0.8574095 0.8574079
##          [,29]     [,30]     [,31]     [,32]     [,33]     [,34]     [,35]
##  [1,] 0.8570565 0.8570551 0.8570538 0.8570527 0.8570516 0.8570506 0.8570497
##  [2,] 0.8570074 0.8570062 0.8570050 0.8570040 0.8570030 0.8570021 0.8570013
##  [3,] 0.8565125 0.8565113 0.8565101 0.8565090 0.8565080 0.8565071 0.8565062
##  [4,] 0.8574258 0.8574243 0.8574229 0.8574215 0.8574203 0.8574191 0.8574181
##  [5,] 0.8570246 0.8570231 0.8570217 0.8570204 0.8570192 0.8570181 0.8570171
##  [6,] 0.8577558 0.8577542 0.8577527 0.8577514 0.8577501 0.8577489 0.8577478
##  [7,] 0.8573151 0.8573137 0.8573124 0.8573112 0.8573101 0.8573090 0.8573081
##  [8,] 0.8578183 0.8578169 0.8578157 0.8578145 0.8578134 0.8578124 0.8578114
##  [9,] 0.8575413 0.8575400 0.8575387 0.8575376 0.8575365 0.8575356 0.8575347
## [10,] 0.8574064 0.8574049 0.8574036 0.8574024 0.8574012 0.8574002 0.8573992
##          [,36]     [,37]     [,38]     [,39]     [,40]     [,41]     [,42]
##  [1,] 0.8570489 0.8570482 0.8570476 0.8570471 0.8570466 0.8570462 0.8570459
##  [2,] 0.8570006 0.8570000 0.8569995 0.8569990 0.8569987 0.8569984 0.8569982
##  [3,] 0.8565055 0.8565049 0.8565043 0.8565038 0.8565034 0.8565031 0.8565029
##  [4,] 0.8574171 0.8574162 0.8574154 0.8574147 0.8574141 0.8574136 0.8574131
##  [5,] 0.8570161 0.8570153 0.8570145 0.8570138 0.8570132 0.8570127 0.8570123
##  [6,] 0.8577467 0.8577458 0.8577450 0.8577442 0.8577435 0.8577429 0.8577424
##  [7,] 0.8573073 0.8573065 0.8573058 0.8573053 0.8573048 0.8573043 0.8573040
##  [8,] 0.8578106 0.8578099 0.8578092 0.8578086 0.8578081 0.8578077 0.8578074
##  [9,] 0.8575339 0.8575332 0.8575326 0.8575321 0.8575316 0.8575313 0.8575310
## [10,] 0.8573983 0.8573975 0.8573968 0.8573962 0.8573956 0.8573952 0.8573948
##          [,43]     [,44]     [,45]     [,46]     [,47]     [,48]     [,49]
##  [1,] 0.8570457 0.8570456 0.8570455 0.8570455 0.8570456 0.8570458 0.8570461
##  [2,] 0.8569981 0.8569980 0.8569981 0.8569982 0.8569983 0.8569986 0.8569989
##  [3,] 0.8565028 0.8565027 0.8565027 0.8565028 0.8565030 0.8565032 0.8565035
##  [4,] 0.8574127 0.8574124 0.8574122 0.8574120 0.8574120 0.8574120 0.8574121
##  [5,] 0.8570119 0.8570116 0.8570114 0.8570113 0.8570113 0.8570113 0.8570114
##  [6,] 0.8577420 0.8577416 0.8577413 0.8577411 0.8577410 0.8577409 0.8577409
##  [7,] 0.8573037 0.8573036 0.8573034 0.8573034 0.8573035 0.8573036 0.8573038
##  [8,] 0.8578072 0.8578070 0.8578069 0.8578069 0.8578069 0.8578071 0.8578073
##  [9,] 0.8575308 0.8575307 0.8575306 0.8575306 0.8575307 0.8575309 0.8575312
## [10,] 0.8573945 0.8573942 0.8573941 0.8573940 0.8573940 0.8573941 0.8573942
##          [,50]     [,51]     [,52]     [,53]     [,54]     [,55]     [,56]
##  [1,] 0.8570464 0.8570468 0.8570472 0.8570478 0.8570484 0.8570490 0.8570498
##  [2,] 0.8569993 0.8569998 0.8570003 0.8570009 0.8570016 0.8570024 0.8570032
##  [3,] 0.8565039 0.8565044 0.8565049 0.8565055 0.8565062 0.8565069 0.8565078
```

```
##  [4,] 0.8574122 0.8574124 0.8574127 0.8574131 0.8574135 0.8574140 0.8574146
##  [5,] 0.8570116 0.8570118 0.8570121 0.8570125 0.8570129 0.8570135 0.8570140
##  [6,] 0.8577410 0.8577412 0.8577414 0.8577417 0.8577421 0.8577425 0.8577430
##  [7,] 0.8573040 0.8573044 0.8573048 0.8573053 0.8573058 0.8573064 0.8573071
##  [8,] 0.8578076 0.8578079 0.8578083 0.8578088 0.8578094 0.8578100 0.8578107
##  [9,] 0.8575315 0.8575319 0.8575324 0.8575329 0.8575335 0.8575342 0.8575349
## [10,] 0.8573944 0.8573947 0.8573951 0.8573955 0.8573960 0.8573965 0.8573972
##           [,57]     [,58]     [,59]     [,60]     [,61]     [,62]     [,63]
##  [1,] 0.8570506 0.8570515 0.8570524 0.8570534 0.8570545 0.8570556 0.8570568
##  [2,] 0.8570041 0.8570050 0.8570060 0.8570071 0.8570082 0.8570094 0.8570107
##  [3,] 0.8565086 0.8565096 0.8565106 0.8565117 0.8565128 0.8565140 0.8565153
##  [4,] 0.8574152 0.8574159 0.8574167 0.8574176 0.8574185 0.8574194 0.8574204
##  [5,] 0.8570147 0.8570154 0.8570162 0.8570171 0.8570180 0.8570189 0.8570200
##  [6,] 0.8577436 0.8577443 0.8577450 0.8577457 0.8577466 0.8577475 0.8577484
##  [7,] 0.8573079 0.8573087 0.8573096 0.8573105 0.8573115 0.8573126 0.8573137
##  [8,] 0.8578115 0.8578123 0.8578132 0.8578141 0.8578151 0.8578162 0.8578174
##  [9,] 0.8575357 0.8575366 0.8575375 0.8575385 0.8575396 0.8575407 0.8575419
## [10,] 0.8573979 0.8573986 0.8573994 0.8574003 0.8574013 0.8574023 0.8574034
##           [,64]     [,65]     [,66]     [,67]     [,68]     [,69]     [,70]
##  [1,] 0.8570580 0.8570593 0.8570607 0.8570622 0.8570636 0.8570652 0.8570668
##  [2,] 0.8570120 0.8570134 0.8570148 0.8570163 0.8570179 0.8570195 0.8570212
##  [3,] 0.8565166 0.8565180 0.8565194 0.8565209 0.8565225 0.8565241 0.8565258
##  [4,] 0.8574215 0.8574227 0.8574239 0.8574252 0.8574265 0.8574279 0.8574294
##  [5,] 0.8570211 0.8570223 0.8570235 0.8570248 0.8570261 0.8570275 0.8570290
##  [6,] 0.8577494 0.8577505 0.8577517 0.8577529 0.8577541 0.8577555 0.8577568
##  [7,] 0.8573149 0.8573161 0.8573175 0.8573188 0.8573203 0.8573218 0.8573233
##  [8,] 0.8578186 0.8578198 0.8578212 0.8578226 0.8578240 0.8578255 0.8578271
##  [9,] 0.8575432 0.8575445 0.8575459 0.8575473 0.8575488 0.8575503 0.8575520
## [10,] 0.8574045 0.8574057 0.8574070 0.8574083 0.8574097 0.8574111 0.8574126
##           [,71]     [,72]     [,73]     [,74]     [,75]     [,76]     [,77]
##  [1,] 0.8570685 0.8570702 0.8570720 0.8570738 0.8570757 0.8570776 0.8570796
##  [2,] 0.8570229 0.8570247 0.8570266 0.8570285 0.8570304 0.8570325 0.8570345
##  [3,] 0.8565276 0.8565294 0.8565312 0.8565332 0.8565351 0.8565371 0.8565392
##  [4,] 0.8574309 0.8574324 0.8574340 0.8574357 0.8574375 0.8574393 0.8574411
##  [5,] 0.8570305 0.8570321 0.8570337 0.8570354 0.8570371 0.8570389 0.8570408
##  [6,] 0.8577583 0.8577598 0.8577613 0.8577629 0.8577646 0.8577663 0.8577681
##  [7,] 0.8573249 0.8573266 0.8573283 0.8573301 0.8573319 0.8573338 0.8573357
##  [8,] 0.8578287 0.8578304 0.8578321 0.8578339 0.8578357 0.8578376 0.8578396
##  [9,] 0.8575536 0.8575553 0.8575571 0.8575590 0.8575609 0.8575628 0.8575648
## [10,] 0.8574142 0.8574158 0.8574175 0.8574192 0.8574210 0.8574228 0.8574247
##           [,78]     [,79]     [,80]     [,81]
##  [1,] 0.8570817 0.8570838 0.8570859 0.8570882
##  [2,] 0.8570366 0.8570388 0.8570410 0.8570433
##  [3,] 0.8565414 0.8565435 0.8565458 0.8565481
##  [4,] 0.8574430 0.8574449 0.8574469 0.8574490
##  [5,] 0.8570427 0.8570446 0.8570466 0.8570487
##  [6,] 0.8577699 0.8577718 0.8577737 0.8577757
##  [7,] 0.8573377 0.8573397 0.8573418 0.8573440
##  [8,] 0.8578416 0.8578436 0.8578458 0.8578479
##  [9,] 0.8575669 0.8575690 0.8575711 0.8575733
## [10,] 0.8574266 0.8574286 0.8574307 0.8574328
```

```
rmses_2_cv <- colMeans(rmses_2)
rmses_2_cv
```

```
##  [1] 0.8573783 0.8573720 0.8573663 0.8573609 0.8573559 0.8573511 0.8573466
##  [8] 0.8573423 0.8573382 0.8573343 0.8573306 0.8573270 0.8573236 0.8573203
## [15] 0.8573172 0.8573142 0.8573114 0.8573087 0.8573061 0.8573036 0.8573012
## [22] 0.8572990 0.8572969 0.8572949 0.8572930 0.8572912 0.8572895 0.8572879
## [29] 0.8572864 0.8572850 0.8572837 0.8572825 0.8572813 0.8572803 0.8572794
## [36] 0.8572785 0.8572778 0.8572771 0.8572765 0.8572760 0.8572755 0.8572752
## [43] 0.8572749 0.8572747 0.8572746 0.8572746 0.8572746 0.8572747 0.8572749
## [50] 0.8572752 0.8572755 0.8572759 0.8572764 0.8572769 0.8572775 0.8572782
## [57] 0.8572790 0.8572798 0.8572807 0.8572816 0.8572826 0.8572837 0.8572848
## [64] 0.8572860 0.8572872 0.8572886 0.8572899 0.8572914 0.8572928 0.8572944
## [71] 0.8572960 0.8572977 0.8572994 0.8573012 0.8573030 0.8573049 0.8573068
## [78] 0.8573088 0.8573108 0.8573129 0.8573151
```

```
qplot(lambdas,rmses_2_cv)
```




```
lambda <- lambdas[which.min(rmses_2_cv)]   #4.9
```

```
mu <- mean(edx$rating)
b_i_reg <- edx %>%
    group_by(movieId) %>%
    summarise(b_i = sum(rating - mu)/(n()+lambda))
b_u_reg <- edx %>%
    left_join(b_i_reg, by="movieId") %>%
    group_by(userId) %>%
    summarise(b_u = sum(rating - b_i - mu)/(n()+lambda))
predicted_ratings_6 <-
    final_holdout_test %>%
    left_join(b_i_reg, by = "movieId") %>%
    left_join(b_u_reg, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)
model_6_rmse <- RMSE(na.omit(predicted_ratings_6), final_holdout_test$rating)   # 0.864818
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method="Regularized Movie + User Effect Model",
                                     RMSE = model_6_rmse))
rmse_results
```

```
## # A tibble: 6 × 2
##   Method                                RMSE
##   <chr>                                <dbl>
## 1 Model 1: Simple overall average model 1.06
## 2 Age Effect Model                      1.06
## 3 Movie Effect Model                    0.944
## 4 Movie + User Effects Model            0.867
## 5 Regularized Movie Effect Model        0.944
## 6 Regularized Movie + User Effect Model 0.866
```

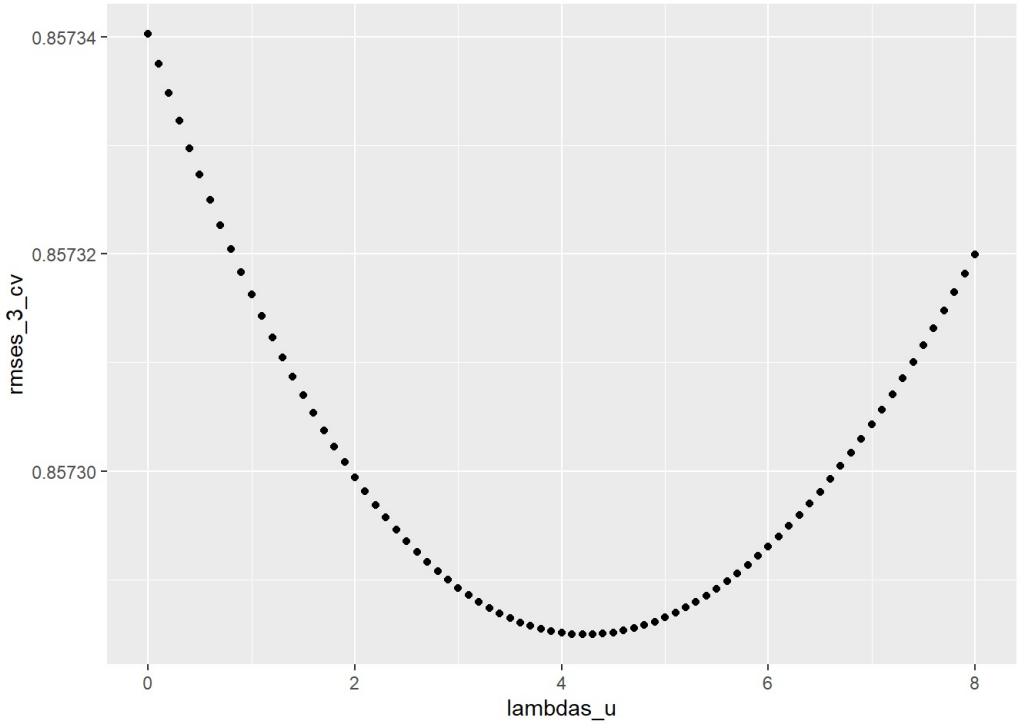
Model 7

For this third round of regularization, we used different lambda values.

```r
# define a matrix to store the results of cross validation
lambda_i <- 2.2
lambdas_u <- seq(0, 8, 0.1)
rmses_3 <- matrix(nrow=10,ncol=length(lambdas_u))

# perform 10-fold cross validation to determine the optimal lambda
for(k in 1:10) {
  train_set <- edx[cv_splits[[k]],]
  test_set <- edx[-cv_splits[[k]],]

  # Make sure userId and movieId in test set are also in the train set
  test_final <- test_set %>%
    semi_join(train_set, by = "movieId") %>%
    semi_join(train_set, by = "userId")

  # Add rows removed from validation set back into edx set
  removed <- anti_join(test_set, test_final)
  train_final <- rbind(train_set, removed)

  mu <- mean(train_final$rating)

  rmses_3[k,] <- sapply(lambdas_u, function(l){
    b_i <- train_final %>%
      group_by(movieId) %>%
      summarise(b_i = sum(rating - mu)/(n()+lambda_i))
    b_u <- train_final %>%
      left_join(b_i, by="movieId") %>%
      group_by(userId) %>%
      summarise(b_u = sum(rating - b_i - mu)/(n()+l))
    predicted_ratings <-
      test_final %>%
      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, test_final$rating))
  })
}
```

```
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
```

```r
#rmses_3
rmses_3_cv <- colMeans(rmses_3)
#rmses_3_cv
qplot(lambdas_u,rmses_3_cv)
```




```
lambda_u <-lambdas_u[which.min(rmses_3_cv)]   #5
lambda_u
```

```
## [1] 4.2
```

```
lambda_i <- 2.2
lambda_u <- 5
mu <- mean(edx$rating)
b_i_reg <- edx %>%
  group_by(movieId) %>%
  summarise(b_i = sum(rating - mu)/(n()+lambda_i))
b_u_reg <- edx %>%
  left_join(b_i_reg, by="movieId") %>%
  group_by(userId) %>%
  summarise(b_u = sum(rating - b_i - mu)/(n()+lambda_u))
predicted_ratings_7 <-
  final_holdout_test %>%
  left_join(b_i_reg, by = "movieId") %>%
  left_join(b_u_reg, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
model_7_rmse <- RMSE(na.omit(predicted_ratings_7), final_holdout_test$rating)   # 0.86485
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method="Regularized Movie + User Effect Model Version 2",
                                     RMSE = model_7_rmse))
rmse_results
```

```
## # A tibble: 7 × 2
##   Method                                          RMSE
##   <chr>                                          <dbl>
## 1 Model 1: Simple overall average model          1.06
## 2 Age Effect Model                               1.06
## 3 Movie Effect Model                             0.944
## 4 Movie + User Effects Model                     0.867
## 5 Regularized Movie Effect Model                 0.944
## 6 Regularized Movie + User Effect Model          0.866
## 7 Regularized Movie + User Effect Model Version 2 0.866
```

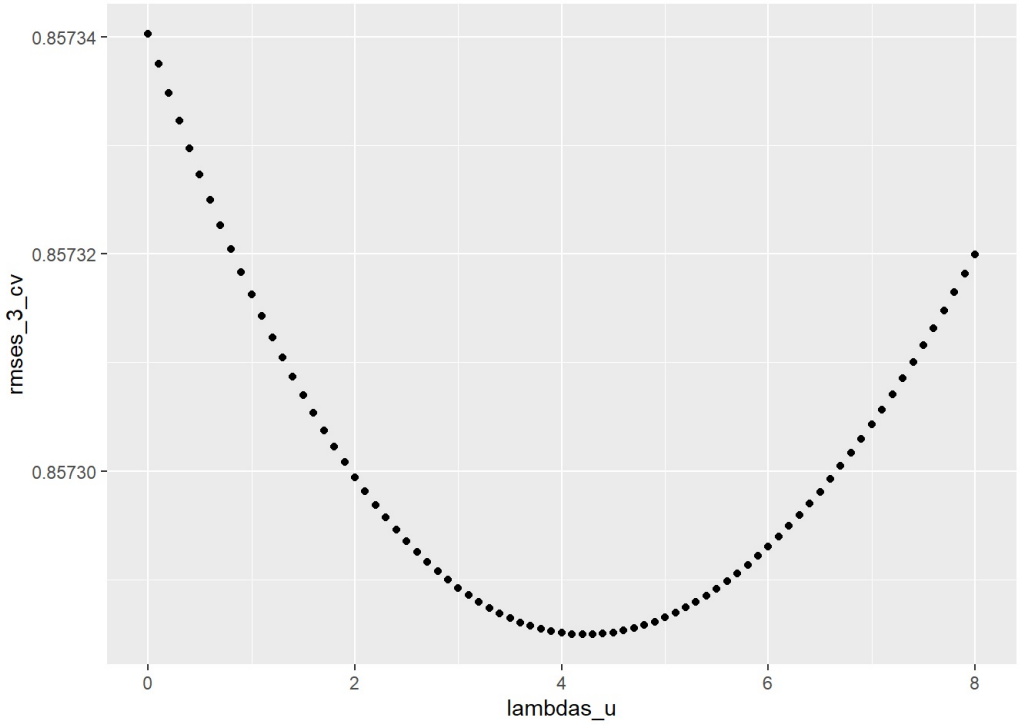
Model 8

In this model, we used cross validation.

```r
# define a matrix to store the results of cross validation
lambda_u <- 5
lambdas_i <- seq(0, 8, 0.1)
rmses_3 <- matrix(nrow=10,ncol=length(lambdas_u))

# perform 10-fold cross validation to determine the optimal lambda
for(k in 1:10) {
  train_set <- edx[cv_splits[[k]],]
  test_set <- edx[-cv_splits[[k]],]

  # Make sure userId and movieId in test set are also in the train set
  test_final <- test_set %>%
    semi_join(train_set, by = "movieId") %>%
    semi_join(train_set, by = "userId")

  # Add rows removed from validation set back into edx set
  removed <- anti_join(test_set, test_final)
  train_final <- rbind(train_set, removed)

  mu <- mean(train_final$rating)

  rmses_3[k,] <- sapply(lambdas_u, function(l){
    b_i <- train_final %>%
      group_by(movieId) %>%
      summarise(b_i = sum(rating - mu)/(n()+lambda_i))
    b_u <- train_final %>%
      left_join(b_i, by="movieId") %>%
      group_by(userId) %>%
      summarise(b_u = sum(rating - b_i - mu)/(n()+l))
    predicted_ratings <-
      test_final %>%
      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, test_final$rating))
  })
}
```

```
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, release,
## genres, date, yearOfRating, monthOfRating, genre, yearsSinceRelsease)`
```

```r
#rmses_3
rmses_3_cv <- colMeans(rmses_3)
#rmses_3_cv
qplot(lambdas_u,rmses_3_cv)
```




```
lambda_i <-lambdas_i[which.min(rmses_3_cv)]   #4.6
lambda_i
```

```
## [1] 4.2
```

```
lambda_i <- 4.6
lambda_u <- 5
mu <- mean(edx$rating)
b_i_reg <- edx %>%
  group_by(movieId) %>%
  summarise(b_i = sum(rating - mu)/(n()+lambda_i))
b_u_reg <- edx %>%
  left_join(b_i_reg, by="movieId") %>%
  group_by(userId) %>%
  summarise(b_u = sum(rating - b_i - mu)/(n()+lambda_u))
predicted_ratings_7 <-
  final_holdout_test %>%
  left_join(b_i_reg, by = "movieId") %>%
  left_join(b_u_reg, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
model_7_rmse <- RMSE(na.omit(predicted_ratings_7), final_holdout_test$rating)   # 0.86485
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method="Regularized Movie + User Effect Model Version 3",
                                     RMSE = model_7_rmse))
rmse_results
```

```
## # A tibble: 8 × 2
##   Method                                          RMSE
##   <chr>                                          <dbl>
## 1 Model 1: Simple overall average model          1.06
## 2 Age Effect Model                               1.06
## 3 Movie Effect Model                             0.944
## 4 Movie + User Effects Model                     0.867
## 5 Regularized Movie Effect Model                 0.944
## 6 Regularized Movie + User Effect Model          0.866
## 7 Regularized Movie + User Effect Model Version 2 0.866
## 8 Regularized Movie + User Effect Model Version 3 0.866
```

Model 9

Here, we used matrix factorization based on the residuals of the baseline model.

```
lambda <- 4.9
mu <- mean(edx$rating)
b_i_reg <- edx %>%
  group_by(movieId) %>%
  summarise(b_i = sum(rating - mu)/(n()+lambda))
b_u_reg <- edx %>%
  left_join(b_i_reg, by="movieId") %>%
  group_by(userId) %>%
  summarise(b_u = sum(rating - b_i - mu)/(n()+lambda))
predicted_ratings_6_edx <-
  edx %>%
  left_join(b_i_reg, by = "movieId") %>%
  left_join(b_u_reg, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
model_6_rmse_edx <- RMSE(na.omit(predicted_ratings_6_edx), edx$rating)
model_6_rmse_edx
```

```
## [1] 0.8540848
```

```
edx_residual <- edx %>%
  left_join(b_i_reg, by = "movieId") %>%
  left_join(b_u_reg, by = "userId") %>%
  mutate(residual = rating - mu - b_i - b_u) %>%
  select(userId, movieId, residual)
head(edx_residual)
```

```
## # A tibble: 6 × 3
##   userId movieId residual
##    <int>   <int>    <dbl>
## 1      1     122   0.634
## 2      1     122   0.634
## 3      1     185   0.364
## 4      1     185   0.364
## 5      1     185   0.364
## 6      1     292   0.0758
```

```
# as matrix
edx_for_mf <- as.matrix(edx_residual)
validation_for_mf <- final_holdout_test %>%
  select(userId, movieId, rating)
validation_for_mf <- as.matrix(validation_for_mf)

# write edx_for_mf and validation_for_mf tables on disk
write.table(edx_for_mf , file = "trainset.txt" , sep = " " , row.names = FALSE, col.names = FALSE)
write.table(validation_for_mf, file = "validset.txt" , sep = " " , row.names = FALSE, col.names = FALSE)

# use data_file() to specify a data set from a file in the hard disk.
set.seed(2019)
train_set <- data_file("trainset.txt")
valid_set <- data_file("validset.txt")

# build a recommender object
r <-Reco()

# tuning training set
opts <- r$tune(train_set, opts = list(dim = c(10, 20, 30), lrate = c(0.1, 0.2),
                                      costp_l1 = 0, costq_l1 = 0,
                                      nthread = 1, niter = 10))
opts
```

```
## $min
## $min$dim
## [1] 20
##
## $min$costp_l1
## [1] 0
##
## $min$costp_l2
## [1] 0.01
##
## $min$costq_l1
## [1] 0
##
## $min$costq_l2
## [1] 0.1
##
## $min$lrate
## [1] 0.1
##
## $min$loss_fun
## [1] 0.8013956
##
##
## $res
##    dim costp_l1 costp_l2 costq_l1 costq_l2 lrate  loss_fun
## 1   10        0     0.01        0     0.01   0.1 0.8220418
## 2   20        0     0.01        0     0.01   0.1 0.8366296
## 3   30        0     0.01        0     0.01   0.1 0.8501177
## 4   10        0     0.10        0     0.01   0.1 0.8132682
## 5   20        0     0.10        0     0.01   0.1 0.8135869
## 6   30        0     0.10        0     0.01   0.1 0.8134812
## 7   10        0     0.01        0     0.10   0.1 0.8051702
## 8   20        0     0.01        0     0.10   0.1 0.8013956
## 9   30        0     0.01        0     0.10   0.1 0.8019726
## 10  10        0     0.10        0     0.10   0.1 0.8152802
## 11  20        0     0.10        0     0.10   0.1 0.8125046
## 12  30        0     0.10        0     0.10   0.1 0.8113999
## 13  10        0     0.01        0     0.01   0.2 0.8272881
## 14  20        0     0.01        0     0.01   0.2 0.8450311
## 15  30        0     0.01        0     0.01   0.2 0.8632647
## 16  10        0     0.10        0     0.01   0.2 0.8152794
## 17  20        0     0.10        0     0.01   0.2 0.8149446
## 18  30        0     0.10        0     0.01   0.2 0.8160064
## 19  10        0     0.01        0     0.10   0.2 0.8098547
## 20  20        0     0.01        0     0.10   0.2 0.8088983
## 21  30        0     0.01        0     0.10   0.2 0.8103678
## 22  10        0     0.10        0     0.10   0.2 0.8148059
## 23  20        0     0.10        0     0.10   0.2 0.8123361
## 24  30        0     0.10        0     0.10   0.2 0.8110877
```

```
# training the recommender model
r$train(train_set, opts = c(opts$min, nthread = 1, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.8274   1.7152e+07
##    1       0.7823   1.5464e+07
##    2       0.7555   1.4810e+07
##    3       0.7367   1.4348e+07
##    4       0.7251   1.4078e+07
##    5       0.7172   1.3898e+07
##    6       0.7117   1.3785e+07
##    7       0.7074   1.3695e+07
##    8       0.7042   1.3633e+07
##    9       0.7014   1.3581e+07
##   10       0.6992   1.3543e+07
##   11       0.6973   1.3504e+07
##   12       0.6958   1.3479e+07
##   13       0.6943   1.3452e+07
##   14       0.6931   1.3426e+07
##   15       0.6921   1.3413e+07
##   16       0.6911   1.3398e+07
##   17       0.6902   1.3375e+07
##   18       0.6895   1.3367e+07
##   19       0.6887   1.3355e+07
```

```
# Making prediction on validation set and calculating RMSE:
pred_file <- tempfile()
r$predict(valid_set, out_file(pred_file))
```

```
## prediction output generated at C:\Users\Zach\AppData\Local\Temp\RtmpGS79q8\file81cc23214b73
```

```
predicted_residuals_mf <- scan(pred_file)
predicted_ratings_mf <- predicted_ratings_6 + predicted_residuals_mf
rmse_mf <- RMSE(na.omit(predicted_ratings_mf),final_holdout_test$rating) # 0.786256
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method="Matrix Factorization",
                                     RMSE = rmse_mf))
rmse_results
```

```
## # A tibble: 9 × 2
##   Method                                          RMSE
##   <chr>                                          <dbl>
## 1 Model 1: Simple overall average model          1.06
## 2 Age Effect Model                               1.06
## 3 Movie Effect Model                             0.944
## 4 Movie + User Effects Model                     0.867
## 5 Regularized Movie Effect Model                 0.944
## 6 Regularized Movie + User Effect Model          0.866
## 7 Regularized Movie + User Effect Model Version 2 0.866
## 8 Regularized Movie + User Effect Model Version 3 0.866
## 9 Matrix Factorization                           0.796
```

Conclusion


```
## # A tibble: 9 × 2
##   Method                                          RMSE
##   <chr>                                          <dbl>
## 1 Model 1: Simple overall average model          1.06
## 2 Age Effect Model                               1.06
## 3 Movie Effect Model                             0.944
## 4 Movie + User Effects Model                     0.867
## 5 Regularized Movie Effect Model                 0.944
## 6 Regularized Movie + User Effect Model          0.866
## 7 Regularized Movie + User Effect Model Version 2 0.866
## 8 Regularized Movie + User Effect Model Version 3 0.866
## 9 Matrix Factorization                           0.796
```


MovieLens is a classical dataset for recommendation systems, and represents a challenge for development of better machine learning algorithms. In this project, the first model only gives an RMSE of 1.0612, and the best baseline model (Model 6: Regularized Movie + User Effect Model) could largely improve it to 0.8648. Furthermore, matrix factorization greatly brought it down to 0.7863. In conclusion, matrix factorization appears to be a very powerful technique for recommendation systems, which usually contains large and sparse datasets, making it hard to make predictions using other machine learning strategies. The effects of age and genres could be further explored to improve the performance of the model.