Data Science: Capstone - MovieLENS PROJECT

KLC

April 28, 2020

Objective

The aim of this project is to build a machine learning algorithm to predict ratings. Such kind of algorithm is widely used in for example, Netflix and Amazon, in making recommendation of items to potential customers.

To measure our prediction performance, we use Root Mean Square Error (RMSE) which is defined as the square root of the average of the differences between predicted values and observed values (i.e. the actual value in the testing dataset in our case). RMSE is frequently used to measure the goodness of fit of a model in predicting quantitative data. The smaller an RMSE value, the closer predicted and observed values are. Our aim is to choose a model which yields the lowest RMSE.

The report will first explore the dataset, then analyse several models and compare their performance.

Dataset

Movielens 10M dataset is used for this project. 10% of the data, namely "validation" will be used for validation, while the remaining will be used for training the machine learning algorithm. The following dataset is provided by the edx capstone project.

```
#################################
# Create edx set, validation set
#################################

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

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")
 movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                            title = as.character(title),
                                            genres = as.character(genres))
```

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

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

Data exploration

First, let's grab an overview of the dataset.

```r
#Structure of the dataset
str(edx)
```

```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 8
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|A
```

```r
# First 6 rows and header
head(edx)
```

```
##   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
```

```
# Summary of statitics
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

Each row of the dataset refers to a rating given by a user to a movie. Total number of ratings given is as below:

```
# Number of ratings given
nrow(edx)
```

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

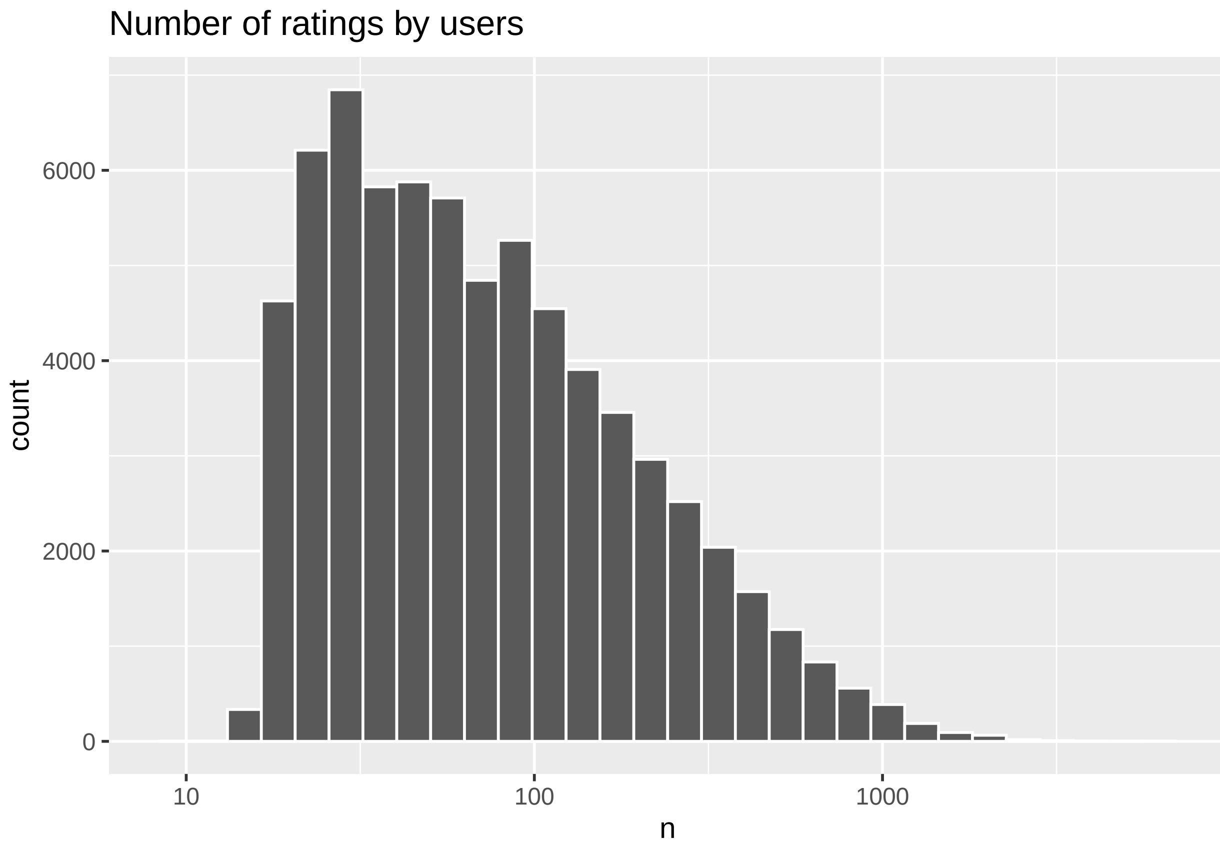
One user can give ratings to multiple movies. Below is the number of users who have given ratings in the dataset.

```
# Total number of users giving ratings
n_distinct(edx$userId)
```

```
## [1] 69878
```

As shown in the plot below, **some users are active in giving ratings while some have given few ratings only**.

```
# Plot of user bias
edx%>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins= 30, color = "white") +
  scale_x_log10() +
  ggtitle("Number of ratings by users")
```



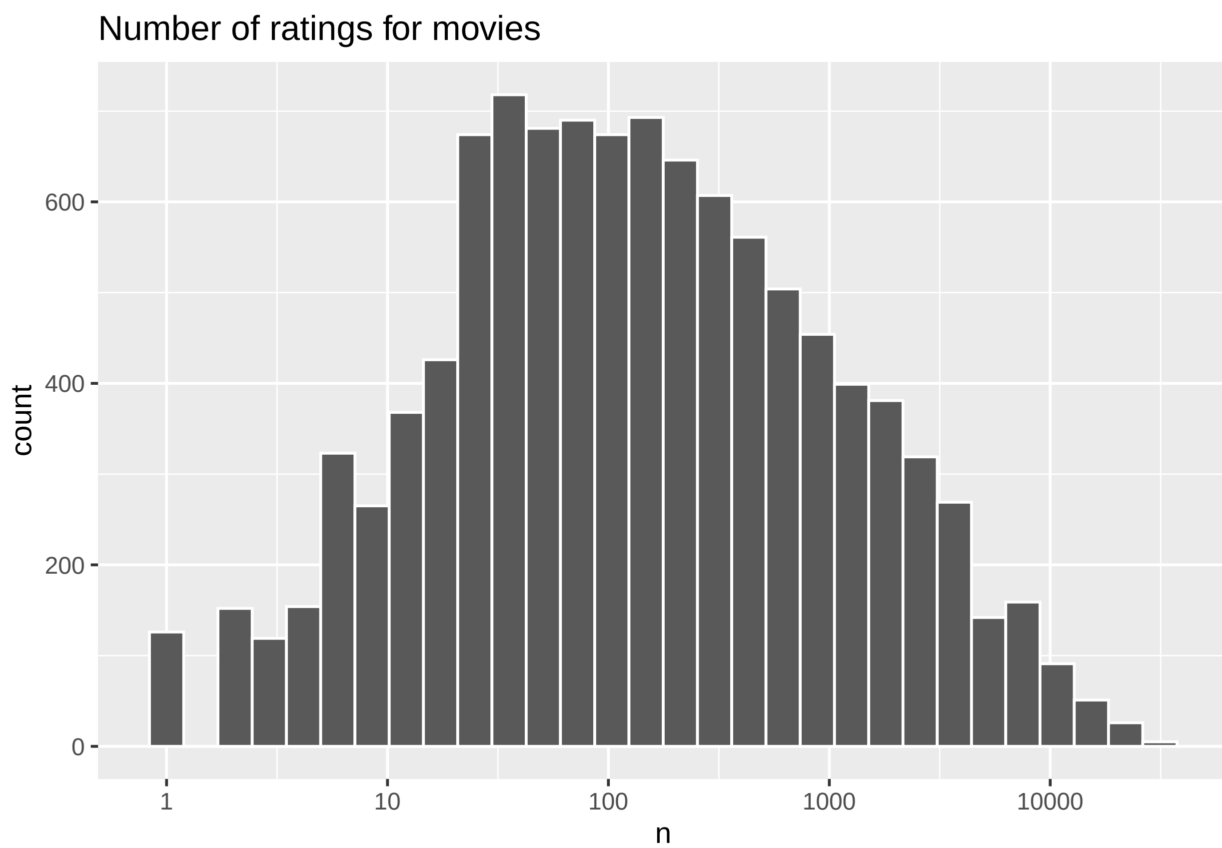
Some popular movies received a lot of ratings, while some movies were only rated by as low as one user. The total number of movies rated is as below:

```
# Total number of movies rated
n_distinct(edx$movieId)
```

```
## [1] 10677
```

As noted through the distribution below, **some movies have been rated more frequent than others**. At the same time, **some movies have been rated very few times**. In fact, 1,139 movies were rated for 10 times or less. This may have adverse impact on the accuracy of our prediction.

```
# Plot of movie bias
edx%>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins= 30, color = "white") +
  scale_x_log10() +
  ggtitle("Number of ratings for movies")
```




```
# Number of movies rated for 10 times or less
edx%>%count(movieId)%>%filter(n<=10)
```

```
## # A tibble: 1,139 x 2
##    movieId     n
##      <dbl> <int>
##  1     109     6
##  2     395     5
##  3     399     6
##  4     403     9
##  5     604     2
##  6     607    10
##  7     642     8
##  8     644     9
##  9     654     9
## 10     739     7
## # ... with 1,129 more rows
```

Data Analysis - Modelling Approach

First, we have to define a function to calculate the RMSE.

```r
# Define RMSE function
RMSE <- function(true_ratings,predicted_ratings){
  sqrt(mean((true_ratings-predicted_ratings)^2))}
```

Simplest model

The simplest model is to assume a same predicted rating for all movies regardless of users, which would be equivalent to the mean of the ratings in the training dataset. We then compute the RMSE with the testing data, and print the result in a table format. This model is denoted as the formula below, with $\epsilon_{u,i}$ as independent error.

$$Y_{u,i} = \mu + \epsilon_{u,i}$$

```r
# Mean of ratings
mu <- mean(edx$rating)
mu
```

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

```r
# Calculate RMSE of this simplest model
simplest_rmse <- RMSE(validation$rating, mu)
simplest_rmse
```

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

```r
# Print the result in a table
rmse_results <- tibble(method = "Using mean rating", RMSE = simplest_rmse)
rmse_results %>% knitr::kable()
```

method	RMSE
Using mean rating	1.061202

To improve the model and get a lower RMSE, we can take into account some insights we gained in the data exploration session above.

Model incorporating the movie effects

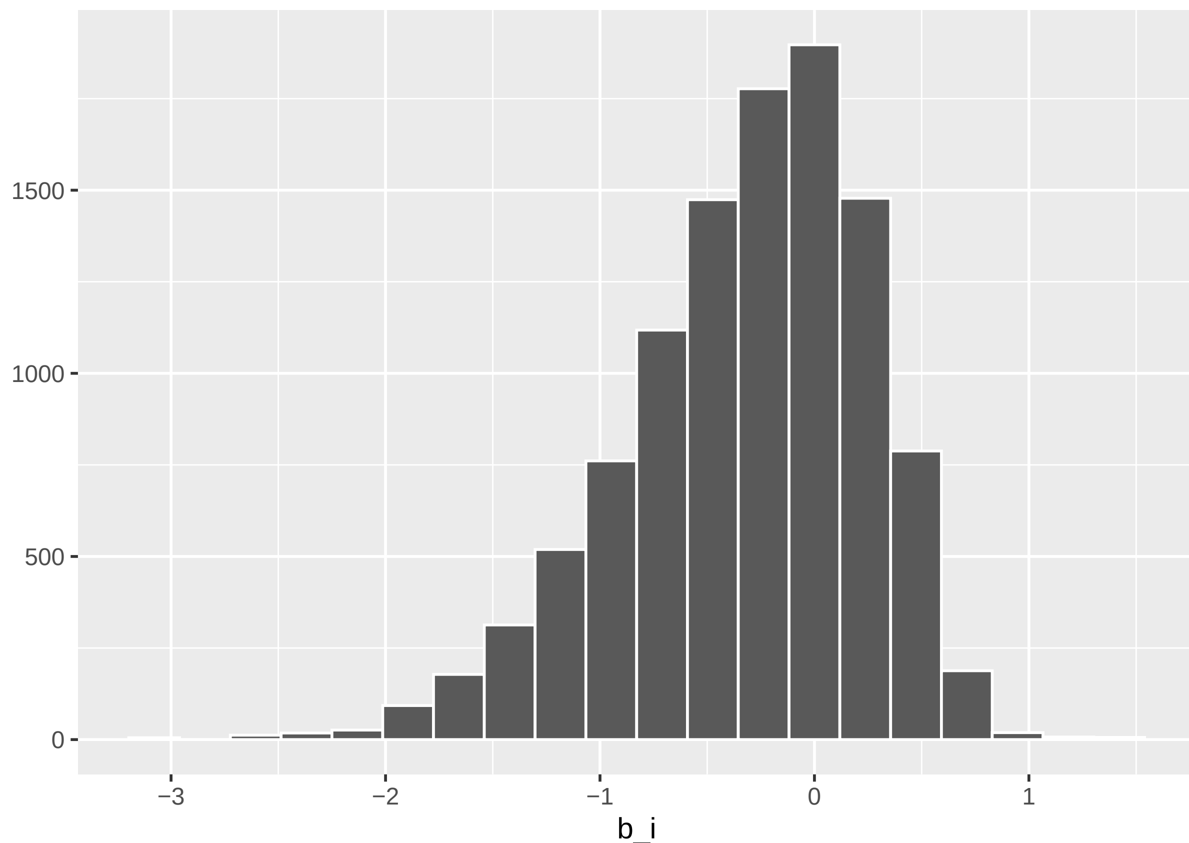
Movies are rated differently. Some movies are generally rated higher than others, as the distribution plot below is skewed towards the right side. We can incorporate this effect by adding a varible b_i as the average ranking for movie i into the previous model:

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

The estimate of b_i can be roughly equivalent to $Y_{u,i} - \mu$ as calculated in the code below:

```r
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

```
movie_avgs %>% qplot(b_i, geom ="histogram", bins = 20, data = ., color = I("white"))
```



To measure how this model is performed, we can calculate its RMSE:

```
# Predict ratings using the test dataset
predicted_ratings_m <- mu + validation %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

# Calculate the RMSE
movie_effects_rmse <- RMSE(validation$rating, predicted_ratings_m)

# Print the result in a table
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie Effect Model",
                                 RMSE = movie_effects_rmse))
rmse_results %>% knitr::kable()
```

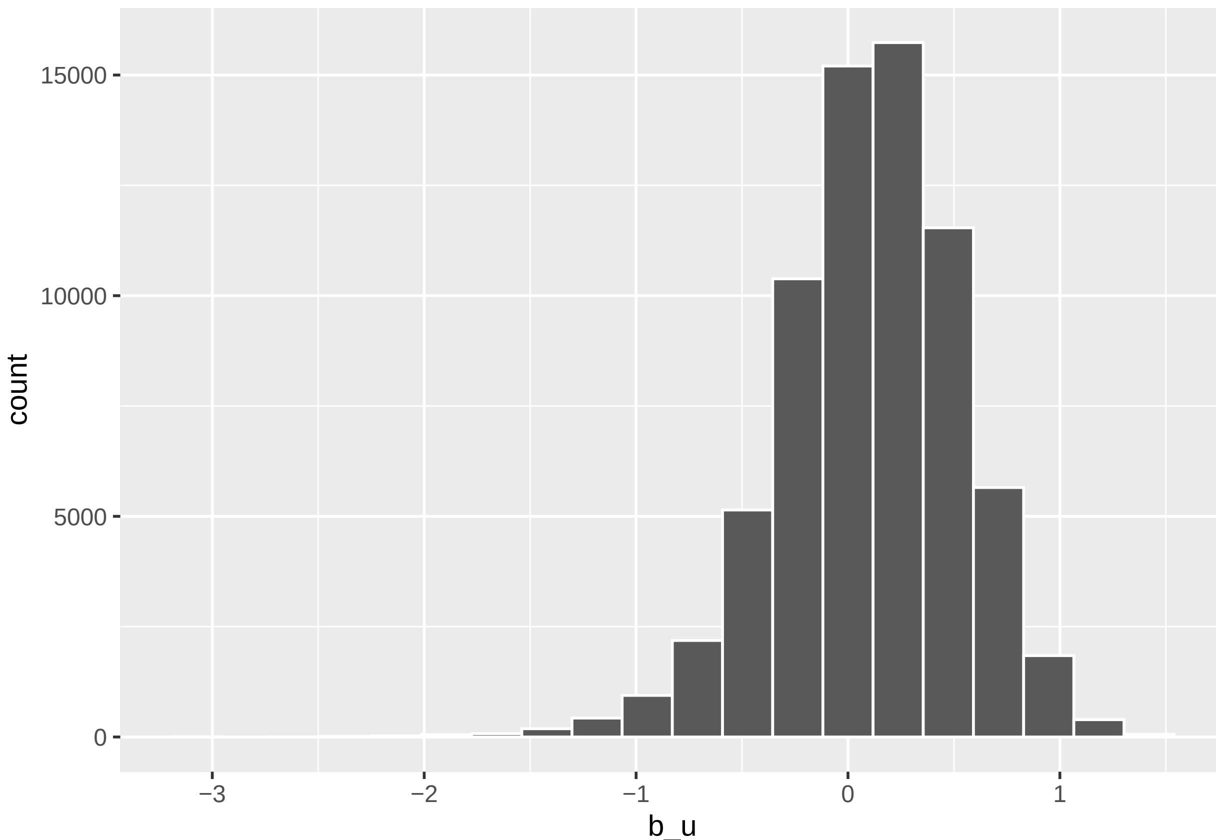
method	RMSE
Using mean rating	1.0612018
Movie Effect Model	0.9439087

We can see an improvement in the model with movie effects, noting a smaller RMSE than the simplest model.

Model incorporating both user and movie effects

As we can see in the plot below, different users rate movies differently. Some tend to rate more positively in general while some could be more picky and rate more negatively.

```
# Plot of average rating for user u
edx %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu)) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 20, color = "white")
```



We can incorporate this effect by adding a varible b_u into previous model:

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

The estimates of b_u can be approximated as $Y_{u,i} - \mu - b_i$ as calculated in the code below:

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

To measure how this model is performed, we can calculate its RMSE:

```r
# Predict ratings using the test dataset
predicted_ratings_m_u <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(prediction = mu + b_i + b_u) %>%
  pull(prediction)

# Calculate the RMSE
user_movie_effects_rmse <- RMSE(validation$rating, predicted_ratings_m_u)

# Print results in a table
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie and User Effects Model",
                                 RMSE = user_movie_effects_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Using mean rating	1.0612018
Movie Effect Model	0.9439087
Movie and User Effects Model	0.8653488

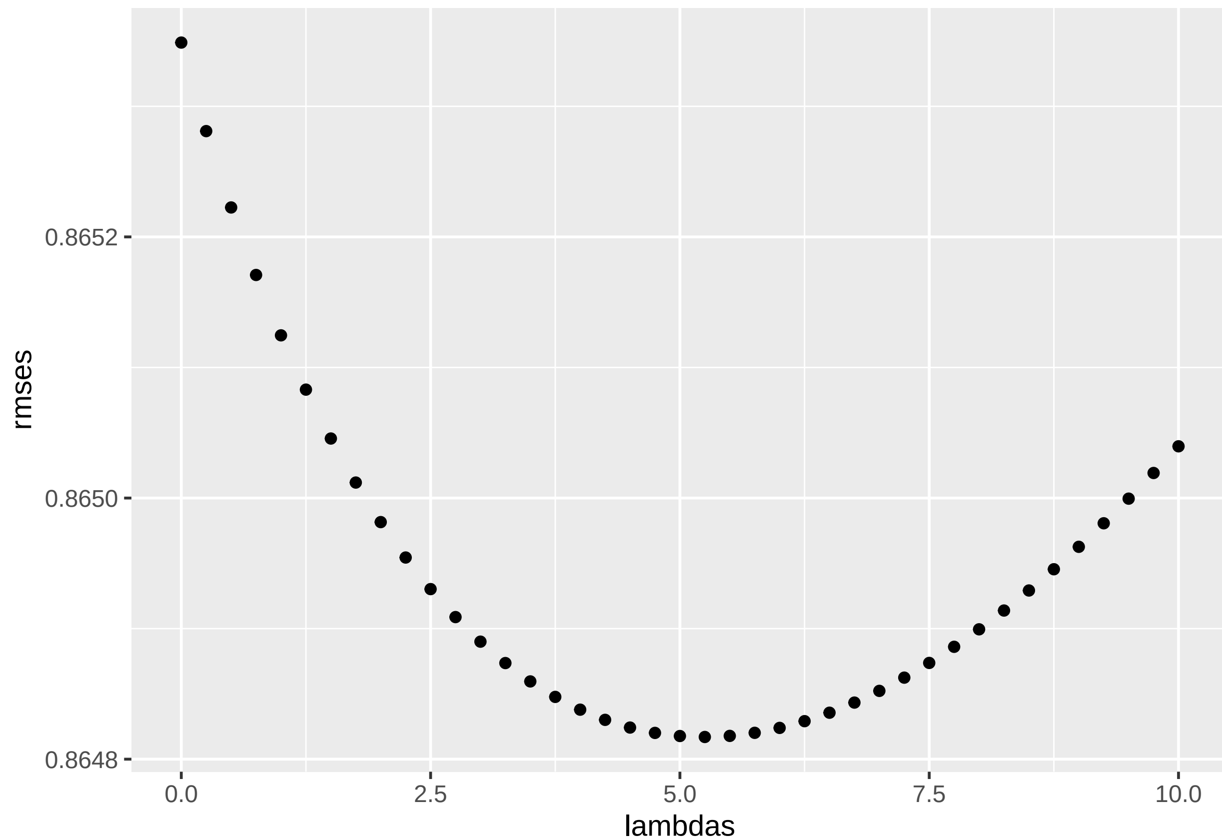
We can see the result has further improved with a smallest RMSE so far.

Regularization Model

As we noted in data exploration session above, some users are much active, while some users rarely rate. Some movies receive much more ratings, while some movies get rated for few times only. These noisy estimates cause more uncertainty and larger error to our prediction, resulting in a higher RMSE. The use of regularization can put a penalty term to give less importance to such effect. We will use cross validation to find a lambda, a tuning parameter, that will yield the smallest RMSE.

```r
# Use cross validation to find a lambda that minimize RMSE
lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas, function(l){
  mu <- mean(edx$rating)

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

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

  predicted_ratings_reg <-
    validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
```

```
    mutate(prediction = mu + b_i + b_u) %>%
    pull(prediction)

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

# Plot RMSE results against lambdas
qplot(lambdas, rmses)
```




```
# Print the lambda that results in the lowest RMSE
lambda <- lambdas[which.min(rmses)]
lambda
```

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

Now, the new RMSE calculated by the regularization model using the optimal lambda will be as below:

```
# Print results in a table
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Regularization Model",
                                 RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

method	RMSE
Using mean rating	1.0612018
Movie Effect Model	0.9439087
Movie and User Effects Model	0.8653488
Regularization Model	0.8648170

Results

The RMSEs for each model discussed above are summarized below:

method	RMSE
Using mean rating	1.0612018
Movie Effect Model	0.9439087
Movie and User Effects Model	0.8653488
Regularization Model	0.8648170

Conclusion

From the table above, we note 11.05% of improvement by incorporating movie effect over the simplest model (using mean rating only). We then see further 8.32% of improvement by taking into account both the user and movie effects. The regularization model giving the lowest RMSE could be viewed as the optimal model for rating prediction. In fact, further improvement may be achieved by examinaing other factors such as timestamp, genres and release year, but hardware limitation, such as insufficient RAM in personal machine, could be a constraint given the long dataset.