Movie Recommendation System

Guido Retegui

17/1/2022

Introduction

Recommendation systems are very common nowadays since they provide numerous benefits for the communities. Users of a service, product or website, among others, have the possibility to give recommendations on specific items. It's very common for companies that sell many products to many customers, such as Amazon, to use a recommendation system. That means that they allow their customers to rate their products, and in doing so, they are able to collect massive datasets that can then be used to make predictions. A very common example of this type of prediction is what rating a particular user will give to a specific item. This allows the company to identify items that are likely to receive a high rating from a particular user and then recommended it to that user in order to increase potential sales.

Another example is Netflix which uses a recommendation system to predict how many stars a user will give to a specific movie. One star suggests it is not a good movie, whereas five stars suggest it is an excellent movie.

For this particular Project, we will construct a movie recommendation system taking the "MovieLens" Dataset, created by GroupLens, from University of Minnesota. The complete MovieLens dataset consists of 27 million ratings of 58,000 movies by 280,000 users. However, due to the scope of this project, we are going to use a simpler versión with 10 million ratings (https://grouplens.org/datasets/movielens/10m/).

Analysis and Methods

Data preparation

In this section the dataset is prepared to make the analysis. Taking the MovieLens dataset, it is split in two parts: the edx set, that is going to be used for training and testing, and the validation set, used to validate the effectiveness of our developed recommendation system. This is the code provided:

```r
##########################################################
# Create edx set, validation set (final hold-out test 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")

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


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

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)`
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

Before we start, it is necessary to explore the data contained in the edx set. We can analyze the structure of the edx dataset with this function:

```r
str(edx)
```

```
## Classes 'data.table' and '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 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Ac
##  - attr(*, ".internal.selfref")=<externalptr>
```

We can see the first entries of the edx 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
```

And, to know how many rows and columns there are in the edx dataset:

```
dim(edx)
```

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

We can then conclude that the edx file contains 9000055 rows and 6 columns.

Those 6 columns are:

1- UserId: integer that identifies a particular user.

2- MovieId: numeric that identifies a movie.

3- Rating: numeric (from 0.5 to 5) that is used to score a movie, made on 5-star scale (whole and half-star ratings).

4- Timestamp: integer that is represented in seconds since 1/1/1970 UTC.

5- Title: character, it represents the movie's name and the year it was released.

6- Genres: character, it represents the category of the Movie.

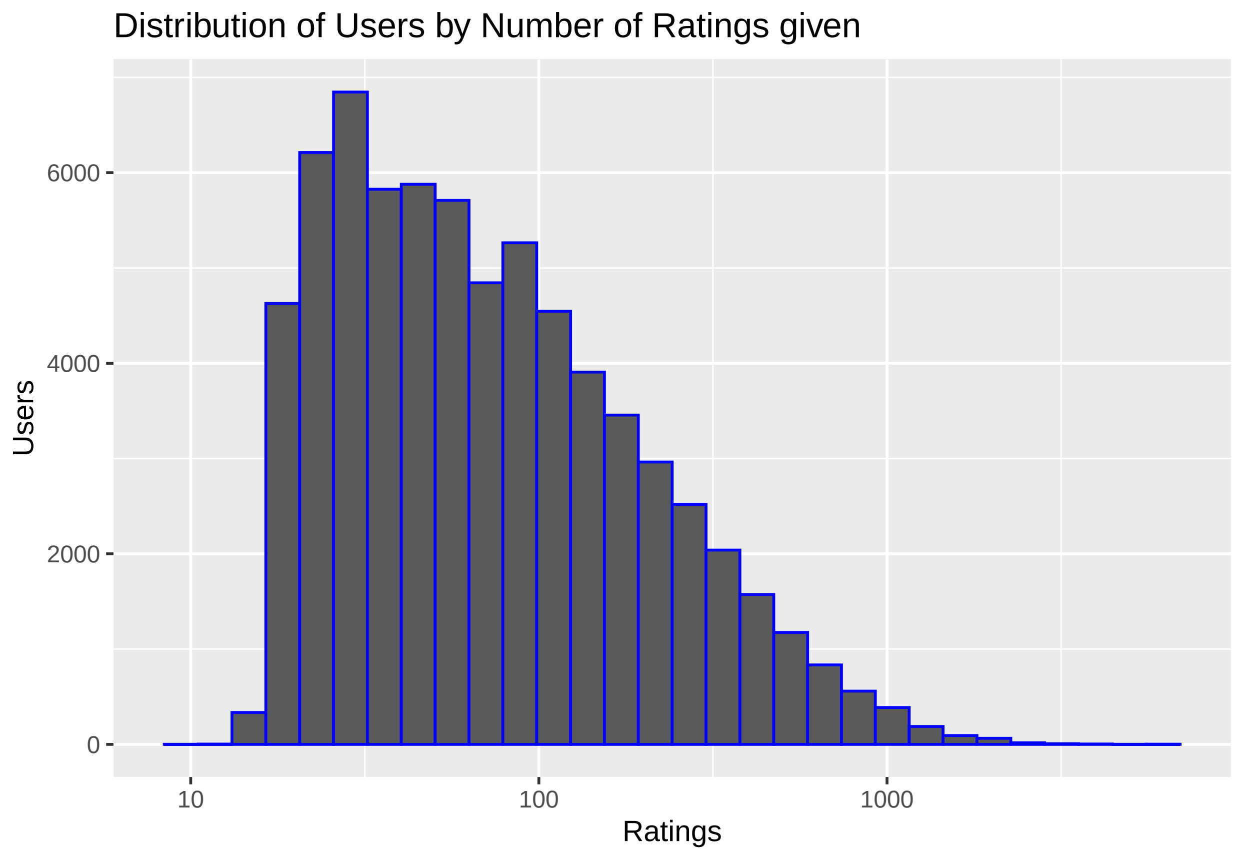
Users

How many distinct users does the dataset have?

```
n_distinct(edx$userId)
```

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

```r
edx %>% group_by(userId) %>%  ## Graph of a distribution of users by number of ratings given
  summarise(n=n()) %>%
  ggplot(aes(n)) +
  geom_histogram(color = "blue") +
  ggtitle("Distribution of Users by Number of Ratings given") +
  xlab("Ratings") +
  ylab("Users")  +
  scale_x_log10()
```



The graph shows the distribution of users by number of ratings given. We can easily see that most of them gave less than a hundred ratings, and just a few gave more than 500 ratings.

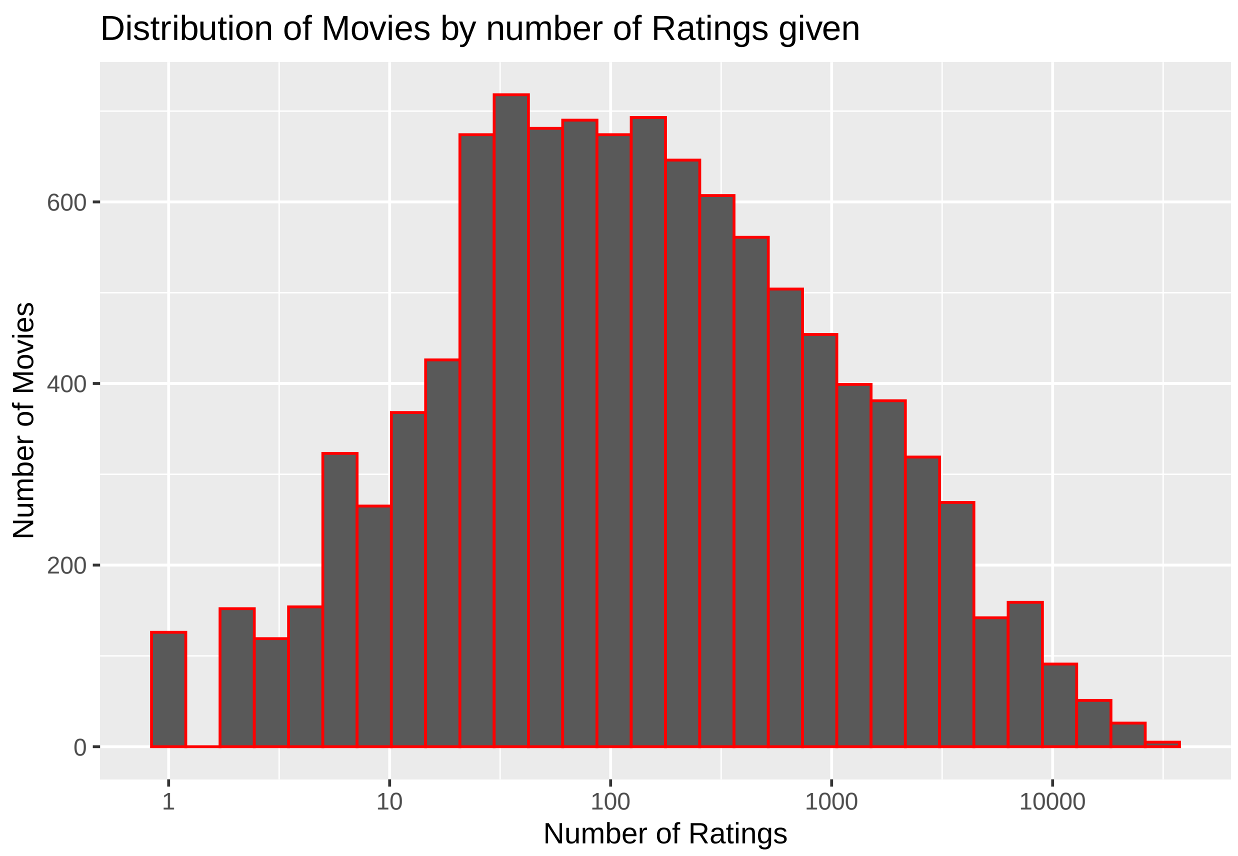
Movies

How many distinct movies does the edx dataset have?

```r
n_distinct(edx$movieId)
```

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

```r
edx %>% group_by(movieId) %>%
  summarise(n=n()) %>%
  ggplot(aes(n)) +
  geom_histogram(color = "red") +
  ggtitle("Distribution of Movies by number of Ratings given") +
  xlab("Number of Ratings") +
  ylab("Number of Movies") +
  scale_x_log10()
```



In this graph we can see the distribution of Movies by number of Ratings. Around 10% of movies have less than 10 ratings.

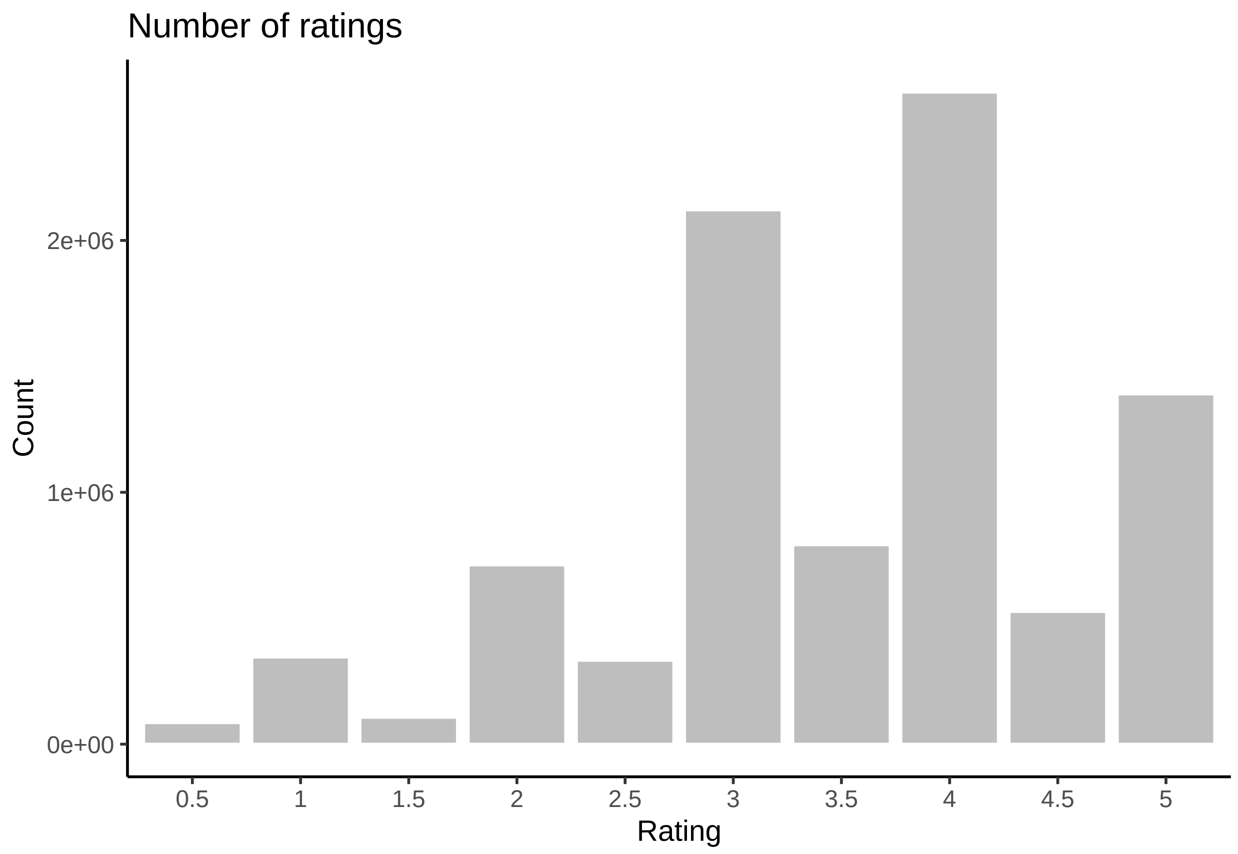
Ratings

The rating of a movie shows (from 0.5 to 5, totaling 10 possible values) the score given by a user. Let's see the rating distribution:

```
ratings_distribution <- edx %>% group_by(rating) %>% summarise(ratings_d_sum = n())
ratings_distribution
```

```
## # A tibble: 10 x 2
##    rating ratings_d_sum
##     <dbl>         <int>
##  1    0.5         85374
##  2    1          345679
##  3    1.5        106426
##  4    2          711422
##  5    2.5        333010
##  6    3          2121240
##  7    3.5         791624
##  8    4          2588430
##  9    4.5         526736
## 10    5          1390114
```

```
ratings_distribution %>% mutate(rating = factor(rating)) %>%
  ggplot(aes(rating, ratings_d_sum)) +
  geom_col(fill = "grey", color = "white") +
  theme_classic() +
  labs(x = "Rating", y = "Count",
       title = "Number of ratings")
```



We conclude half-star ratings are less common than whole star ratings. The mean of ratings is near 3.5 and the most popular rating given is 4 stars (2,588,430 times, representing more than the 28% of the total).

Genres

As we saw before, the edx dataset provides the genre category for every movie. This is a quick review of the first six combinations of these movie categories and how many times they appear:

```
edx %>% group_by(genres) %>%
  summarise(n=n()) %>% head()
```

```
## # A tibble: 6 x 2
##   genres                                                  n
##   <chr>                                               <int>
## 1 (no genres listed)                                      7
## 2 Action                                              24482
## 3 Action|Adventure                                    68688
## 4 Action|Adventure|Animation|Children|Comedy           7467
## 5 Action|Adventure|Animation|Children|Comedy|Fantasy    187
## 6 Action|Adventure|Animation|Children|Comedy|IMAX        66
```

How many distinct combinations of genres are in the edx dataset?

```
n_distinct(edx$genres)
```

```
## [1] 797
```

Model Performance Evaluation

Root Mean Squared Error - RMSE

The evaluation will consist in comparing the predicted value with the actual outcome. To compare this relationship, we need a loss function: The Root Mean Squared Error. To make an understandable example, when the user consistently selects the predicted movie, the error is equal to zero and the algorithm is perfect. The RMSE is defined by the formula:

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})^2}$$

In the general formula, N is the number of observations; $y_{u,i}$ is the observation i,u; $\hat{y}_{u,i}$ is the estimated value of the observation i,u.

In this case, we will use N as the number of ratings, $y_{u,i}$ as the rating of movie i by user u and $\hat{y}_{u,i}$ as the prediction of movie i by user u.

```
RMSE <- function(true_ratings, pred_ratings){
  sqrt(mean((true_ratings - pred_ratings)^2))
}
```

Our goal is to reduce the Root Mean Squared Error below 0.8649, (RMSE < 0.8649).

Evaluated Models

Regression Models

Starting for the simplest model, we can assume that every movie has the same rating and a difference explained by random variation (or error):

$$\hat{Y}_{i,u} = \mu + \epsilon_{i,u}$$

Where $\hat{Y}_{i,u}$ is the prediction of movie i rated by user u, μ is the mean of observed data and $\epsilon_{i,u}$ is the error of movie i rated by user u.

If we focus on the random term $\epsilon_{i,u}$, we can also assume that variability is explained by the fact that every single movie has its own rating distribution. We can improve our previous model by adding the term b_i, to represent the average ranking for movie i. The second model, with the movie effects is then:

$$\hat{Y}_{i,u} = \mu + b_i + \epsilon_{i,u}$$

We can also notice that some users are more active than others. We can also see that some have different rating patterns. To make our last model more intuitive with this idea, we can add the term b_u, representing the user specific effect. The formula will be:

$$\hat{Y}_{i,u} = \mu + b_i + b_u + \epsilon_{i,u}$$

Regularization

As we have previously seen, many movies have a few ratings, and some users rate only a few movies. The sample size of these movies and users are very small. To solve this problem, we are going to add a penalty for large values of b_i and b_u:

$$\frac{1}{N}\sum_{u,i}(y_{u,i} - \mu - b_i - b_u)^2 + \lambda(\sum_i b_i^2 + \sum_u b_u^2)$$

where the regularization term $\lambda(\sum_i b_i^2 + \sum_u b_u^2)$ penalizes the magnitudes of these parameters, and the first term $\frac{1}{N}\sum_{u,i}(y_{u,i} - \mu - b_i - b_u)^2$, it is just the sum of the squared errors.

Using calculus we can actually show that the values of b_i and b_u that minimize this equation are:

$$\hat{b}_i = \frac{1}{n_i + \lambda}\sum_{u=1}^{n_i}(Y_{u,i} - \hat{\mu})$$

$$\hat{b}_u = \frac{1}{n_i + \lambda}\sum_{u=1}^{n_i}(Y_{u,i} - \hat{b}_i - \hat{\mu})$$

where n_i is the number of ratings made for movie i. This approach consists in ignoring the penalty when the sample size n_i is very large, since $n_i + \lambda \approx n_i$. We are going to simulate several values of λ to choose the best option to minimize RMSE.

Results

Regression Models

```
#movie effect

# average of all ratings of the edx dataset
mu <- mean(edx$rating)

# calculate b_i on the training set
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(bi = mean(rating - mu))
```

```
# predicted ratings
pred_ratings_bi <- mu + validation %>%
  left_join(movie_avgs, by='movieId') %>%
  .$bi

#movie + user effect

#b_u using the training set
user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(bu = mean(rating - mu - bi))

#predicted ratings
pred_ratings_bu <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + bi + bu) %>%
  .$pred
```

```
rmse1 <- RMSE(validation$rating,pred_ratings_bi)
rmse1
```

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

```
rmse2 <- RMSE(validation$rating,pred_ratings_bu)
rmse2
```

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

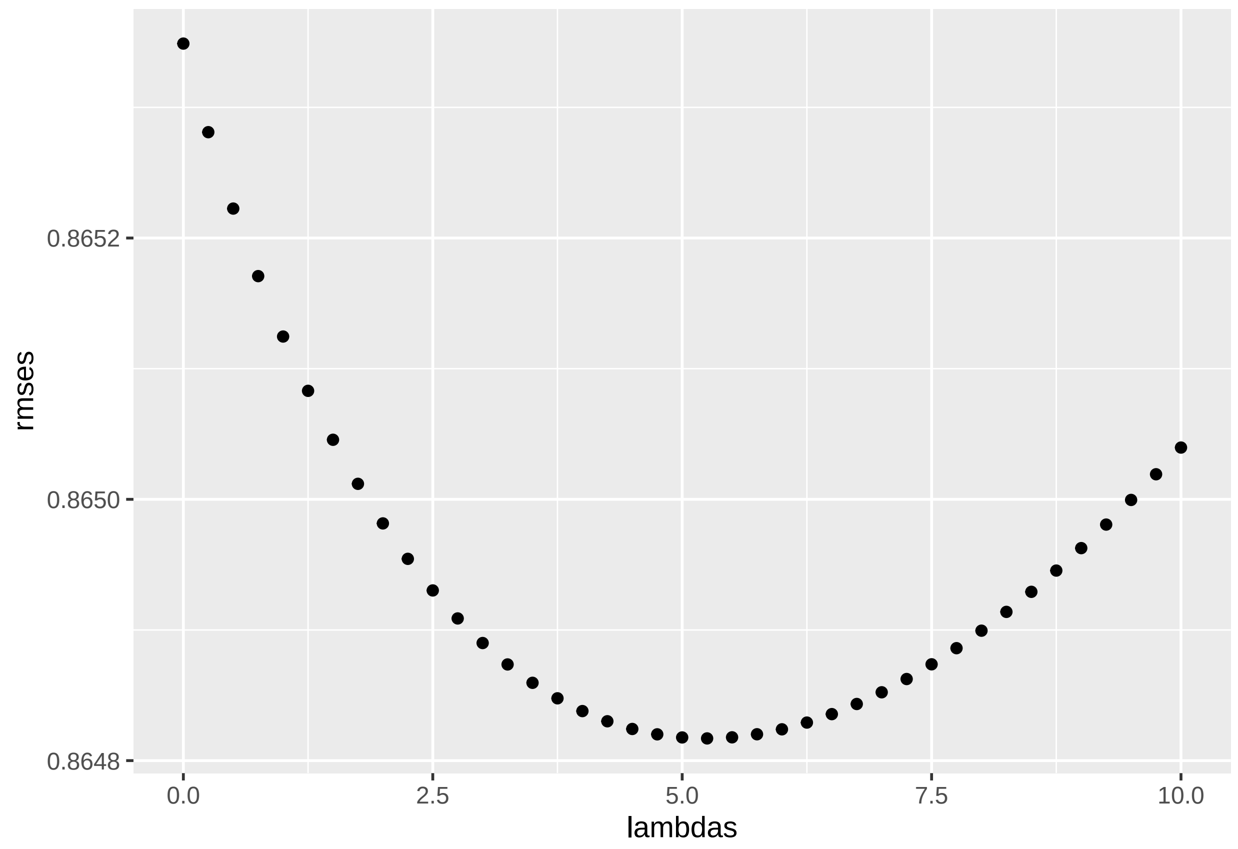
We observe how the last RMSE decreased about 8% with respect to the first RMSE, by adding the user effect. The next step is to perform regularization as it was explained in the methods section.

Regularization

```
# Regularization

# Lambda is a tuning parameter. We can use cross-validation to choose it.

lambdas <- seq(0, 10, 0.25)

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

  mu_reg <- mean(edx$rating)

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

  bu_reg <- edx %>%
    left_join(bi_reg, by="movieId") %>%
    group_by(userId) %>%
    summarize(bu_reg = sum(rating - bi_reg - mu_reg)/(n()+l))

  pred_ratings_biu <-
    validation %>%
    left_join(bi_reg, by = "movieId") %>%
    left_join(bu_reg, by = "userId") %>%
    mutate(pred = mu_reg + bi_reg + bu_reg) %>%
    .$pred

  return(RMSE(validation$rating,pred_ratings_biu))
})
```

Here is the graph that shows the sequence of λ.

```
qplot(lambdas, rmses)
```



We can easily see how many of them accomplished with the condition RMSE < 0.8649. But which of them has the minimum value of RMSE?

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

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

And what is that value of RMSE?

```r
rmse3 <- min(rmses)
rmse3
```

```
## [1] 0.864817
```

Conclusion

With the code provided we split the MovieLens Dataset in two parts, the test and training part, and the validation part. With a very simple review of the Edx dataset, we analyse how the variables interact with each other using some graphs to make the study more understandable. Before we started, we defined the models to use, with their respective features, differences and improvements.

According to the first model, the result gave us a RMSE equal to 0.9439087. To make a better performance, we add the user effects, achieving a RMSE equal to 0.8653488 (an improve of about 8%).

Finally, using the regularization model, we added a penalty for particular cases of movie and users with a few number of ratings, making them less relevant for the evaluation. This gave us a final result of a RMSE equal to 0.864817, achieving the goal.