Data Science: Capstone - MovieLens Project

Rocío Alvarez Mercé

April 2019

Quick Note

Hello! This is my final project for the MovieLens dataset. It took a little longer than expected because of work issues, so I apologize for not submitting it before. It was written with the idea of presenting it to other people in mind, and let them be able to follow my steps, one by one. As if we were in a class or a little company meeting. That is why I use the pronoun "we" instead of "I" in every part of the report, I guess I'm used to it and I think it gives the readers a better understanding of the analysis and the steps to follow. With this quick note I wanted to emphasize that even if I use "we" in every section, I'm the only author of this report. That being said, hope you enjoy it!

1. Introduction

In this project, I will be creating a movie recommendation system using the MovieLens dataset. First of all, I will go through this dataset, do a little bit of data analysis in order to understand what data we are working with, then apply a machine learning code to predict movie ratings and try to reach a good RMSE (<0.8775) using different models.

The MovieLens dataset (available at http://movielens.org) comes from the GroupLens Research. GroupLens is a research lab in the Department of Computer Science and Engineering at the University of Minnesota, Twin Cities. In this ratings data sets, the information was collected over various periods of time, depending on the size of the set.

2. The datasets: edX Capstone project code

Here is the code provided in the Capstone series to download / generate the datasets and start developing our analysis and the algorithm.

```
############################################################
# Create edx set, validation set, and submission file
############################################################

# 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")

# 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 <- read.table(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))

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

# Validation set will be 10% of MovieLens data

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)
```

Packages just in case

```
library(lubridate)
library(purrr)
library(tidyverse)
library(caret)
```

3. Exploring the data

We are going to explore and analyse the data a little, to see what we are working with, and if we can get some trends or conclusions from thiss data, before rushing to the machine learning algorithm. To start with this section, we are going to follow the quizz questions and then dive into our own personal analysis.

edx DataSet I now have the edx dataframe (df): 6000055 observations of 6 variables. I can check this using:

```
dim(edx)
```

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

Let's take a peek of what this df looks like. What kind of information do we have?

```
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
```

So we have identified 6 columns:

- userId: Identification number for each user. From the MovieLens website we can see that these users were selected at random for inclusion and that their ids have been anonymized.
- movieId: Identification number for each movie. Only movies with at least one rating or tag are included in the dataset.
- rating: Rating (from 0 to 5 stars, with a 0.5 step) given by the user to the movie.
- timestamp: This column represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.
- title: The title of the movie, followed by the year of its premiere.
- genres: One or more movie genres associated with the film.

3.1. Knowing the dataset

Let's do a little more research about the number of users, films and ratings in this dataset. It's always important to know with how much data we are working with, and some parameters to know the amount of data we are analyzing.

How many different movies are in the dataset?

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

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

There are 10677 different movies in our dataframe.

How many different users can we find in our df?

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

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

There are 69878 distinct users in our dataframe.

If we look at the genre column of our dataset, we see that some films have only one category (e.g. "Drama"), but others have more than one (e.g. "Drama" and "Thriller"). These categories are separated by the "|" character. So, how many film categories do we finally have?

```
sep_genres <- edx %>% separate_rows(genres, sep = "\\|") %>%
                    group_by(genres) %>% summarize(count = n()) %>%
                    arrange(desc(count))

sep_genres
```

```
## # A tibble: 20 x 2
##    genres               count
##    <chr>                <int>
##  1 Drama              3910127
##  2 Comedy             3540930
##  3 Action             2560545
##  4 Thriller           2325899
##  5 Adventure          1908892
##  6 Romance            1712100
##  7 Sci-Fi             1341183
##  8 Crime              1327715
##  9 Fantasy             925637
## 10 Children            737994
## 11 Horror              691485
## 12 Mystery             568332
## 13 War                 511147
## 14 Animation           467168
## 15 Musical             433080
## 16 Western             189394
## 17 Film-Noir           118541
## 18 Documentary          93066
## 19 IMAX                 8181
## 20 (no genres listed)      7
```

We can see we have 20 categories (in reality, we have 19 because the 20th one is "no genres listed"):

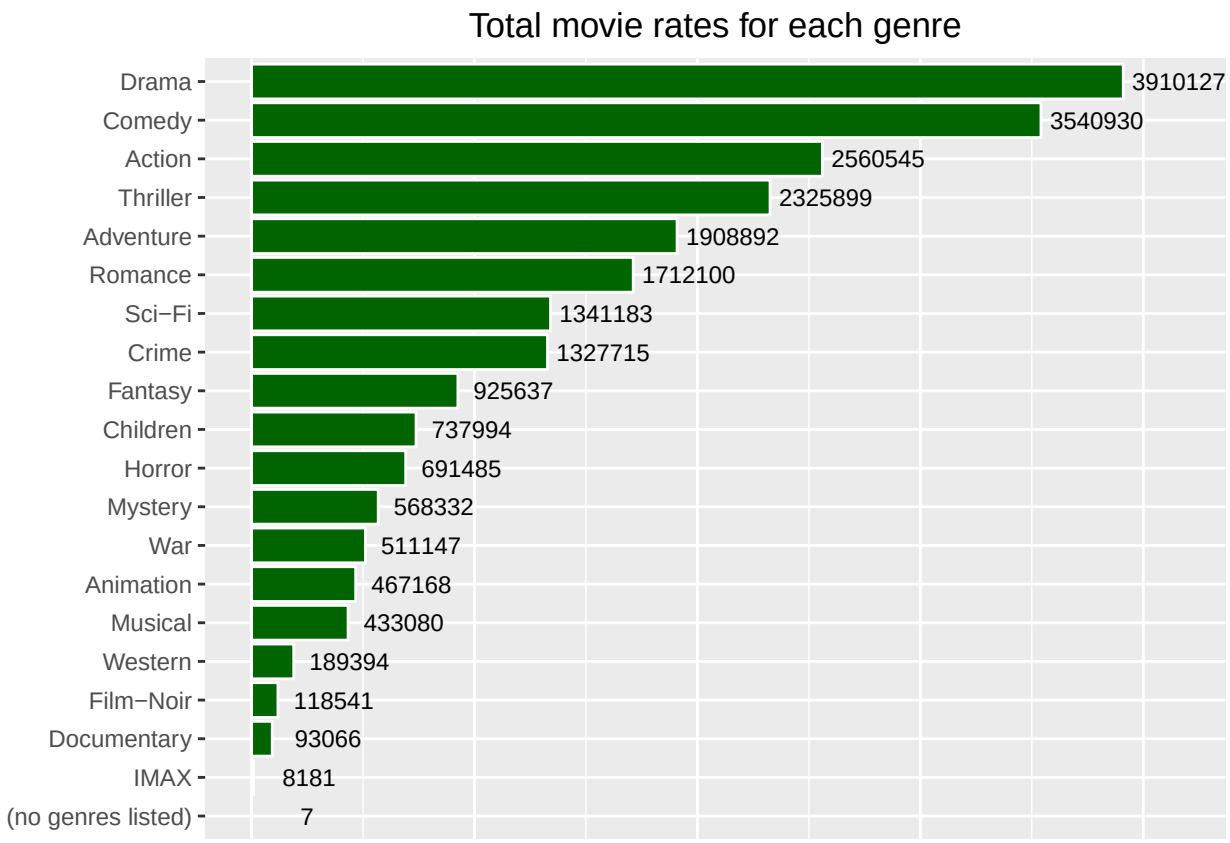
```
sep_genres$genres
```

```
##  [1] "Drama"              "Comedy"             "Action"
##  [4] "Thriller"           "Adventure"          "Romance"
##  [7] "Sci-Fi"             "Crime"              "Fantasy"
## [10] "Children"           "Horror"             "Mystery"
## [13] "War"                "Animation"          "Musical"
## [16] "Western"            "Film-Noir"          "Documentary"
## [19] "IMAX"               "(no genres listed)"
```

Let's visualize this so it's easier to understand:

```
ggplot(sep_genres, aes(x=reorder(genres, count),y = count),
       position = position_stack(reverse = TRUE)) +
      geom_bar(stat = "identity", color = 'white', fill = 'darkgreen') +
```

```
      coord_flip() +
      geom_text(aes(label=sep_genres$count), nudge_y = 250000, size = 3) +
      labs(title = "Total movie rates for each genre") +
      theme(plot.title = element_text(hjust = 0.5), axis.text.x=element_blank(),
            axis.ticks.x=element_blank(), axis.title.x=element_blank(),
            axis.title.y=element_blank())
```



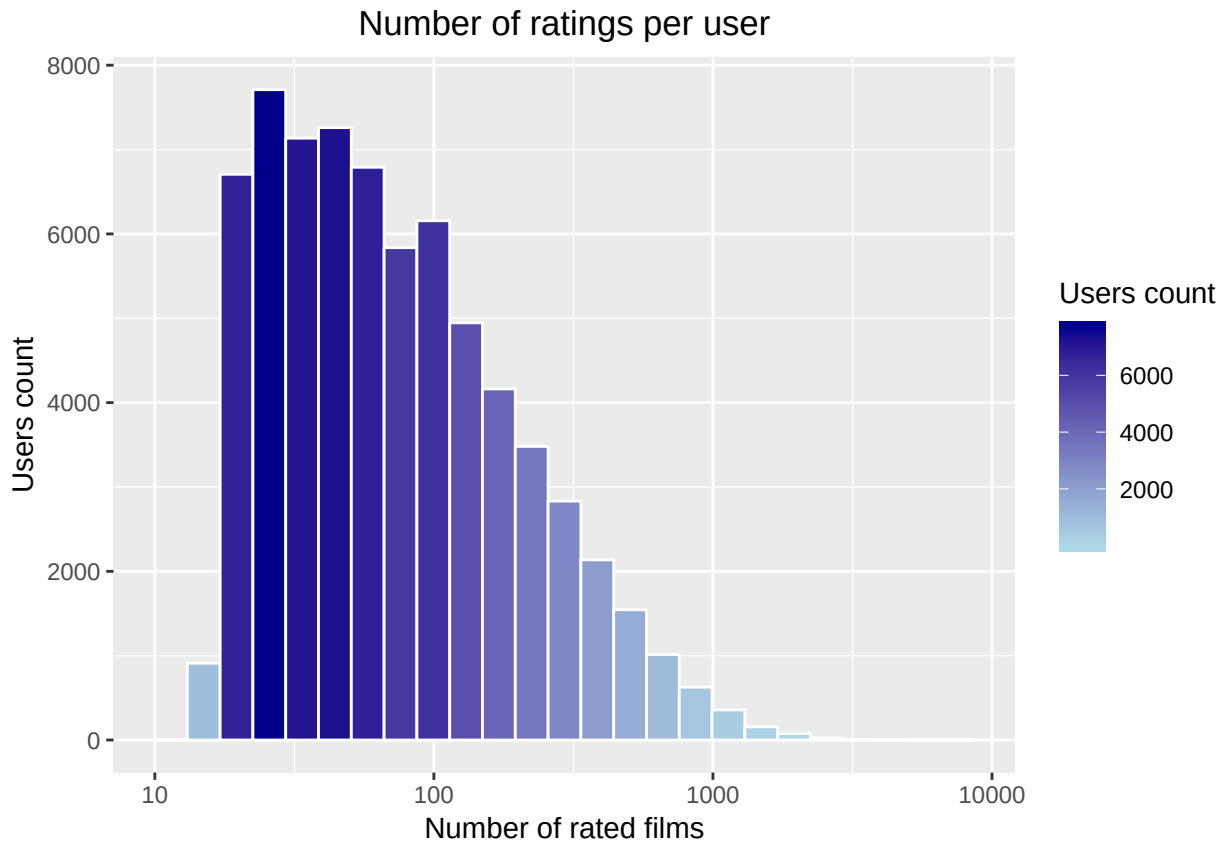
We see that the top 3 genres with more rates are Drama, Comedy and Action. The least rated genres are Film Noir, Documentary and IMAX.

3.2. Movie Ratings

3.2.1 Ratings per user

We want to study the number of ratings per user variable. To achieve this, we can calculate:

```
edx %>% group_by(userId) %>% summarize(n = n()) %>%
        ggplot(aes(n)) +
        geom_histogram(color = "white", aes(fill=..count..), bins = 25) +
        scale_x_log10() +
        labs(title = "Number of ratings per user", x = "Number of rated films",
             y = "Users count") +
        theme(plot.title = element_text(hjust = 0.5)) +
        scale_fill_gradient("Users count", low="lightblue", high="darkblue")
```

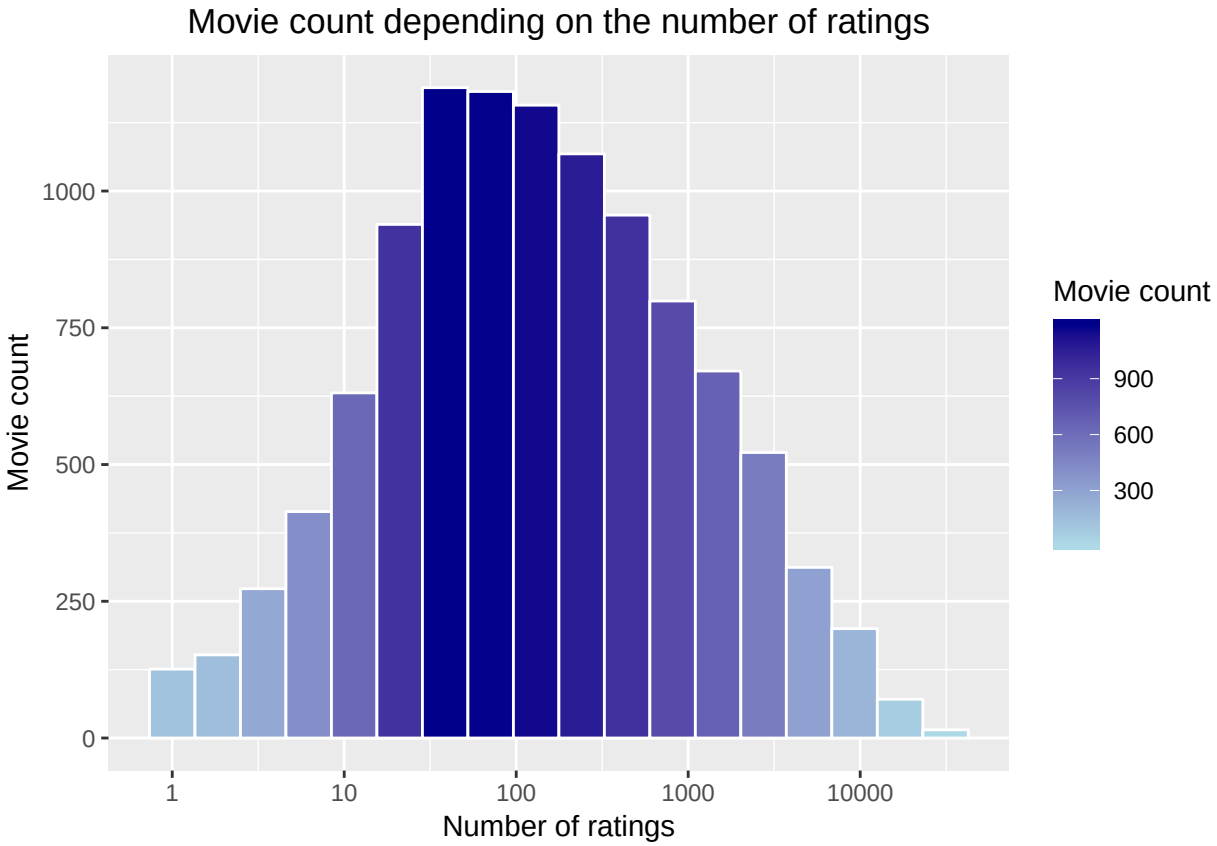


From this graph we can conclude that some users are more active than others at rating movies. We see that the majority of users rate between 30 and 100 films in average. This is a behaviour we should expect, because a small portion of the users will rate more than 1000 films.

3.2.2. Rating distribution on movies

Are all the movies rated the same number of times? We would guess not. To visualize this, we can compute how many movies recieved n quantity of ratings. We can do that with this code:

```
edx %>% count(movieId) %>% arrange(-n) %>% ggplot(aes(n)) + scale_x_log10() +
        geom_histogram(color = "white", aes(fill=..count..), bins = 18) +
        labs(title="Movie count depending on the number of ratings",
             y="Movie count", x="Number of ratings") +
        theme(plot.title = element_text(hjust = 0.5)) +
        scale_fill_gradient("Movie count", low="lightblue", high="darkblue")
```



As we would expect, some movies get rated more than others. Some films are classics, maybe they are older, or have had more marketing or a famous cast working in it, thus have more audience. This probably translates in a bigger number of ratings. Other films (independent movies, low marketing, specialized content) are watched by just a few.

This leads us to ask ourselves, which movies have the greatest number of ratings?

3.2.3. Most rated movies

We group the dataset by title and we count how many times it has been rated. We then print the top 10 rated movies.

```
rated_titles <- edx %>% group_by(title) %>% summarize(count = n()) %>%
                        arrange(desc(count))


head(rated_titles,10)
```

```
## # A tibble: 10 x 2
##    title                                                     count
##    <chr>                                                     <int>
##  1 Pulp Fiction (1994)                                       31362
##  2 Forrest Gump (1994)                                       31079
##  3 Silence of the Lambs, The (1991)                          30382
##  4 Jurassic Park (1993)                                      29360
##  5 Shawshank Redemption, The (1994)                          28015
```

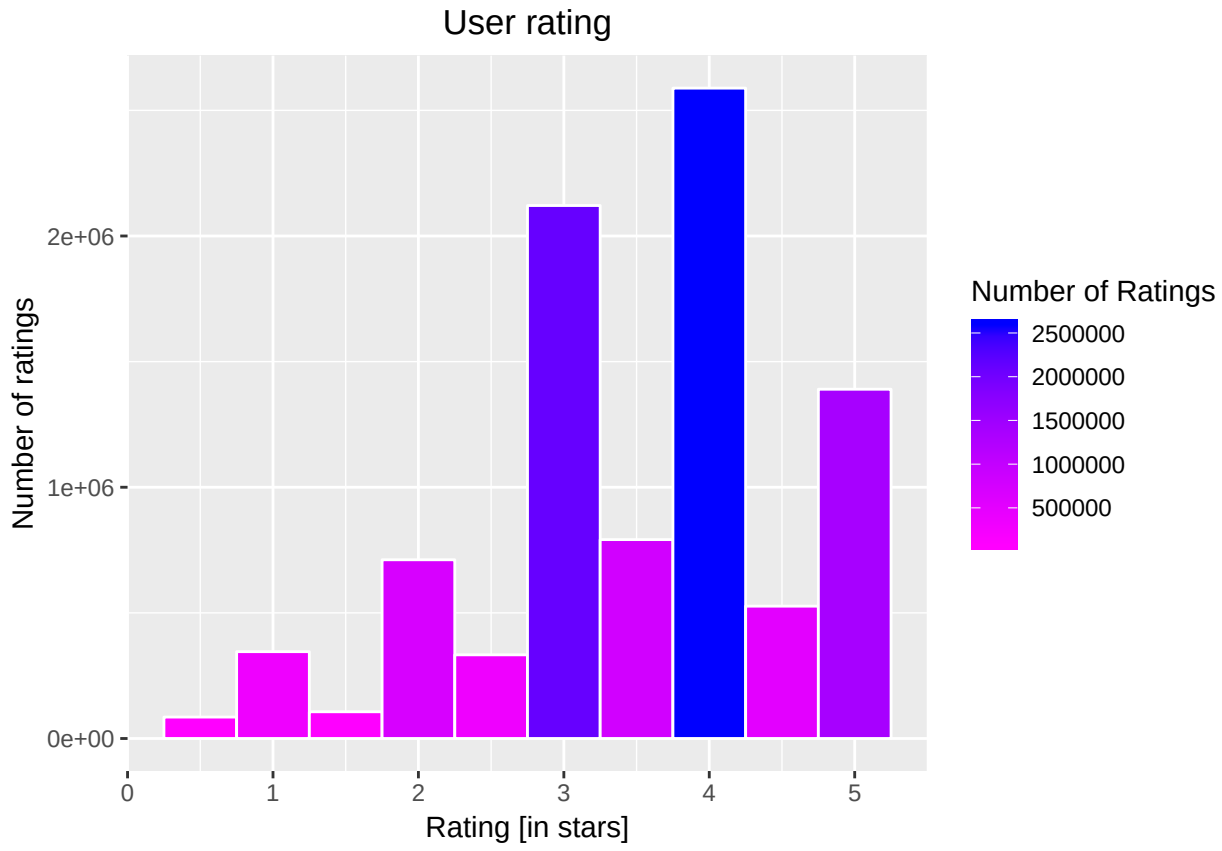
```
##  6 Braveheart (1995)                                              26212
##  7 Fugitive, The (1993)                                           25998
##  8 Terminator 2: Judgment Day (1991)                              25984
##  9 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977) 25672
## 10 Apollo 13 (1995)                                               24284
```

We can see that "Pulp Fiction", "Forrest Gump" and "The Silence of the Lambs" are the three most rated movies in this dataset.

3.2.4. User Rating

How many times did the users give 5 stars to a movie? And how about zero stars? To see the rating distribution of the films given by the users we can plot it using a histogram.

```
edx %>% ggplot(aes(rating)) +
        geom_histogram(binwidth = .5, aes(fill=..count..), color = 'white') +
        labs(title = "User rating", x = "Rating [in stars]", y = "Number of ratings") +
        theme(plot.title = element_text(hjust = 0.5)) +
        scale_fill_gradient("Number of Ratings", low="magenta", high="blue")
```

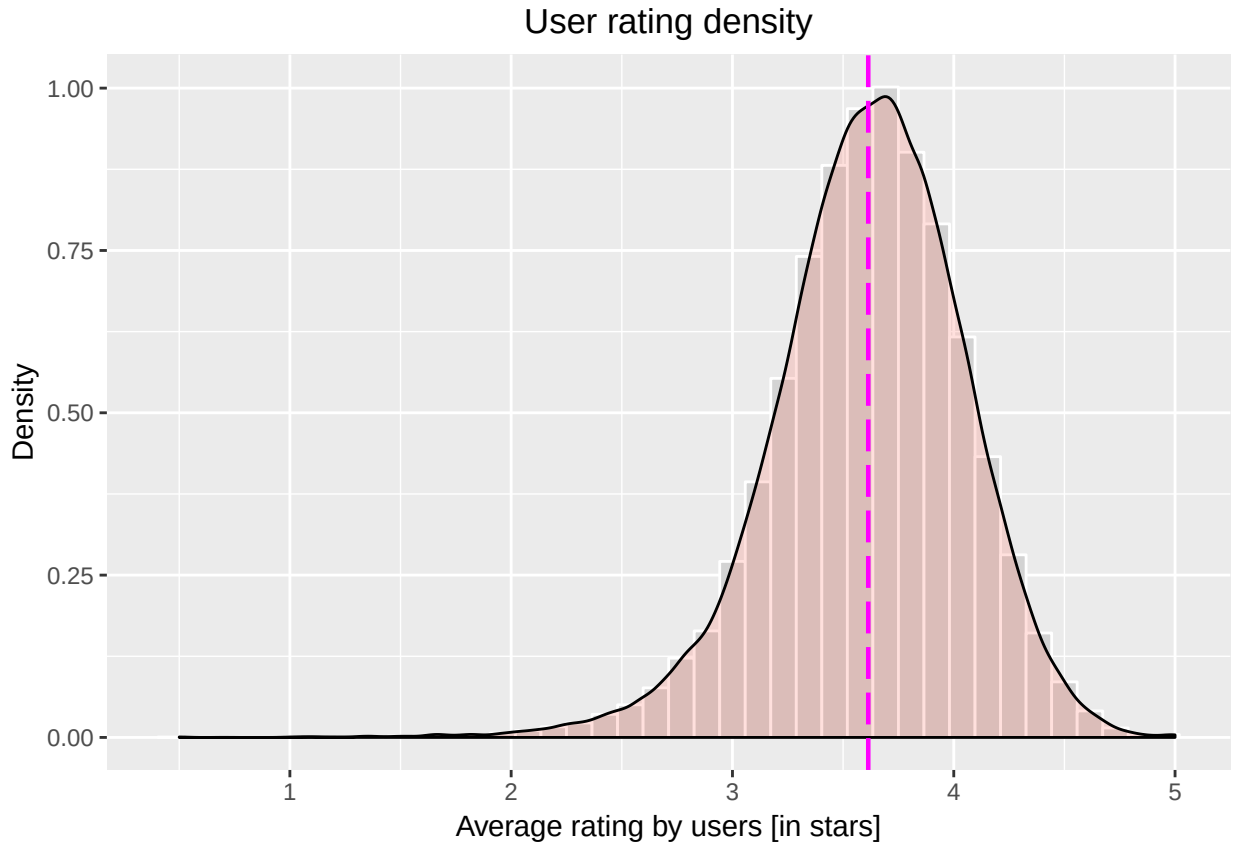


In magenta we can see the lowest user ratings, and in blue the most clicked at. We can say two main things about this histogram: first of all, that movies are mostly well rated, from 3 on, with no movie having zero stars. So our rating scale begins at 0.5 stars. We can see that the five most given ratings in order from most to least are: 4, 3, 5, 3.5 and finally 3 stars. Secondly, we observe that, as asked on the quizz on the last question, half star ratings are less common than whole star ratings (e.g., there are fewer ratings of 3.5 than there are ratings of 3 or 4, etc.).

3.2.5. User rating density

We've seen before that some users, as you would expect, rate more films than others. From that analysis and the previous graph, a question arises: how many stars do users give in average? Are there users that love every movie they see, and some others that give low ratings everytime they punctuate a film?

```
mean_rating <- edx %>% group_by(userId) %>% summarize(avg = mean(rating))
mean_rating %>% ggplot(aes(avg)) +
                geom_histogram(aes(x = avg, y = ..density..), bins = 40,
                               fill = "lightgray", color = "white") +
                geom_density(alpha = .2, fill="#FF6655") +
                labs(title="User rating density",
                    x="Average rating by users [in stars]", y="Density") +
                theme(plot.title = element_text(hjust = 0.5)) +
                geom_vline(aes(xintercept = mean(avg, na.rm = T)),
                           colour = "magenta", linetype ="longdash", size = .8)
```



It seems we have here a normal distribution around 3.5 stars approximatively. Our dataset users have an average rating of:

```
mean(mean_rating$avg)
```

```
## [1] 3.613602
```

We can see the biggest portion of the users give a rating average between 3 and 4.5 stars.

4. Machine Learning

4.1. Creating train and test sets

We need to create our train and test sets out of the edx dataframe. The validation df will be only used to test the performance of the model later. So, a standard way to do this is by randomly splitting the data. We will use the caret package (already loaded) and the function createDataPartition. We chose to select 20% of the data.

```
set.seed(42) #You gotta love this number!
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = F)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]
```

As we did before when creating the validation set (section 2 of the report), we make sure that movieId and userId in the test set are also on the train set:

```
test_set <- test_set %>% semi_join(train_set, by = "movieId") %>%
                         semi_join(train_set, by = "userId")
```

4.2. Model-based approach

For our first approach, we are going to use a model-based approach, serving as a baseline approach for what we want to achieve. This model will assume the same rating for all movies and users, with the differences explained by random variation. The model will look like this:

y_ui = mu + epsilon

Where y_ui is the rating for movie "i" by user "u", mu is the true rating for all movies and users, and epsilon represents the independent errors sampled from the same distribution centered at zero.

We already have our train and test sets (train_set and test_set variables respectively). The estimate that minimizes the RMSE is the least squared estimate of mu. In this case, it will be the average, which we can get with this code:

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

```
## [1] 3.512451
```

Now, to get the baseline approach RMSE, we calculate:

```
RMSE_base <- RMSE(test_set$rating, mu_hat)
RMSE_base
```

```
## [1] 1.059841
```

Our first approach RMSE is 1.0598412 > 1, not a good value. This means we are normally missing the rating value by one or more stars. This was expected, since it's the most basic model and does not take into account a lot of factors. Let's see if we can get a better RMSE with another method.

4.2.1. Method's table

As we will be comparing different methods and approaches, we create a table that's going to store the results we obtain as we go along trying to optimize our RMSE to get it below 0.8775, which is our primary objective in this report. So, to create this table we write:

```
rmse_results <- data_frame(Method = "Model-based Approach", RMSE = RMSE_base)
rmse_results
```

```
## # A tibble: 1 x 2
##   Method                RMSE
##   <chr>                <dbl>
## 1 Model-based Approach  1.06
```

Applying the Model-based approach, we start with this RMSE value, not so good. But we will improve it! Let's try with a different approach.

4.3. Movie Effect Model

We have seen on previous sections on this report that some movies are generally rated higher than others. So, we think we can optimize our previous model (the Model-based approach from section 4.2.) by adding a term b_i that represents the average rating for movie "i". We add that bias to the equation, to get:

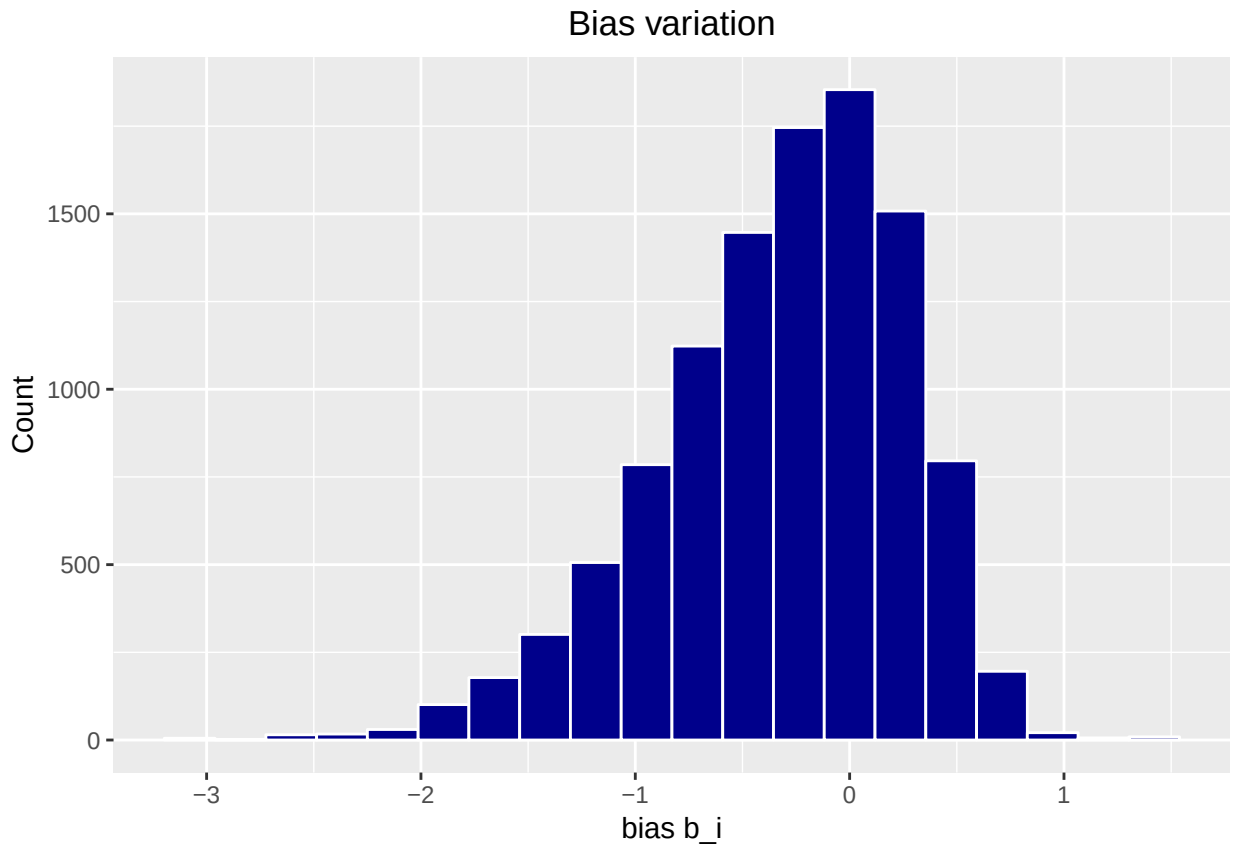
y_ui = mu + b_i + epsilon

For this approach we know that the least squared estimate b_hat_i is the average of y_ui minus the overall mean for each movie "i". We can comput it using this code:

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

How do these estimates vary?

```
movie_avgs %>% ggplot(aes(b_i)) +
               geom_histogram(bins = 20, color = 'white', fill = 'darkblue') +
               labs(title = "Bias variation", x = "bias b_i", y = "Count") +
               theme(plot.title = element_text(hjust = 0.5))
```



We can see the b_i's vary a lot: some movies are good and well rated, some movies are bad and poorly rated. So, let's write our movie effect model and the resulting RMSE:

```
predicted_ratings <- mu + test_set %>% left_join(movie_avgs, by = 'movieId') %>% .$b_i

model_1_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method = "Movie Effect Model", RMSE = model_1_rmse))

rmse_results
```

```
## # A tibble: 2 x 2
##   Method                RMSE
##   <chr>                <dbl>
## 1 Model-based Approach 1.06
## 2 Movie Effect Model   0.943
```

We can see that our RMSE for the Movie effect model now is 0.9427265. It has improved compared to our Model-based approach RMSE, but we can still optimize it. Can a users variable help us improve our estimate?

4.4. Movie + Users Effect Model

We add a new term to our model, taking into account the users effect. We now have:

y_ui = mu + b_i + b_u + epsilon

Where b_u is the user-specific effect. So, if a very strict user that normally rates movies with a low number of stars now rates a great movie with a 3 instead of a 5 for instance, this term will help us balance this effect and improve our predictions.

We will use this code to implement this model and compute our new RMSE:

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

predicted_ratings <- test_set %>% left_join(movie_avgs, by = 'movieId') %>%
                                  left_join(user_avgs, by = 'userId') %>%
                                  mutate(pred = mu + b_i + b_u) %>% .$pred

model_2_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method = "Movie + User Effects Model",
                                     RMSE = model_2_rmse))

rmse_results
```

```
## # A tibble: 3 x 2
##   Method                      RMSE
##   <chr>                      <dbl>
## 1 Model-based Approach       1.06
## 2 Movie Effect Model         0.943
## 3 Movie + User Effects Model 0.843
```

we see that we now obtain a further improvement, a RMSE of 0.842542, better than the ones obtained using the Model-based approach and the Movie Effect Model. What's more, we have obtained a RMSE < 0.8775 in the edx dataset. It's time to try it in our validation set!

4.5. Model application to validation data

Now that we have gone through three different models, from the more basic one to the best one, the "Movie + User Effects Model", adding a different term each time to optimize our predictions, it's time to apply it to the validation data.

```
predicted_validation <- validation %>% left_join(movie_avgs, by = "movieId") %>%
                                       left_join(user_avgs, by = "userId") %>%
                                       mutate(pred = mu + b_i + b_u) %>% .$pred
validation_RMSE <- RMSE(predicted_validation, validation$rating, na.rm = T)
validation_RMSE
```

```
## [1] 0.8820432
```

We create a table with this result, as we did earlier, but now on the validation set:

```
rmse_validation_results <- data_frame(Validation_Method = "Movie + User Effects Model",
                                      RMSE = validation_RMSE)
rmse_validation_results
```

```
## # A tibble: 1 x 2
##   Validation_Method           RMSE
##   <chr>                      <dbl>
## 1 Movie + User Effects Model 0.882
```

We see that for the validation set, the RMSE is bigger than on the test set from the edx dataframe. This was kind of expected, since the validation set has "unseen" data, and we are doing the "real-case scenario" predictions. Unfortunately for us, we haven't still achieved our RMSE < 0.8775 goal. So, how can we further improve this result? Let's see that on the next section of the report.

5. Improving the results: Regularization

Regularization will permit us to penalize large estimates that come from small sample sizes: for instance, larger estimates of b_i when fewer users rate the movie. To achieve this, we will search for a tuning parameter lambda that will help us achieve our RMSE goal. First we will try it on the edx dataframe, to see if it actually optimizes our RMSE. Then, we will finally apply it to the validation set.

5.1. Regularization on the edx dataset

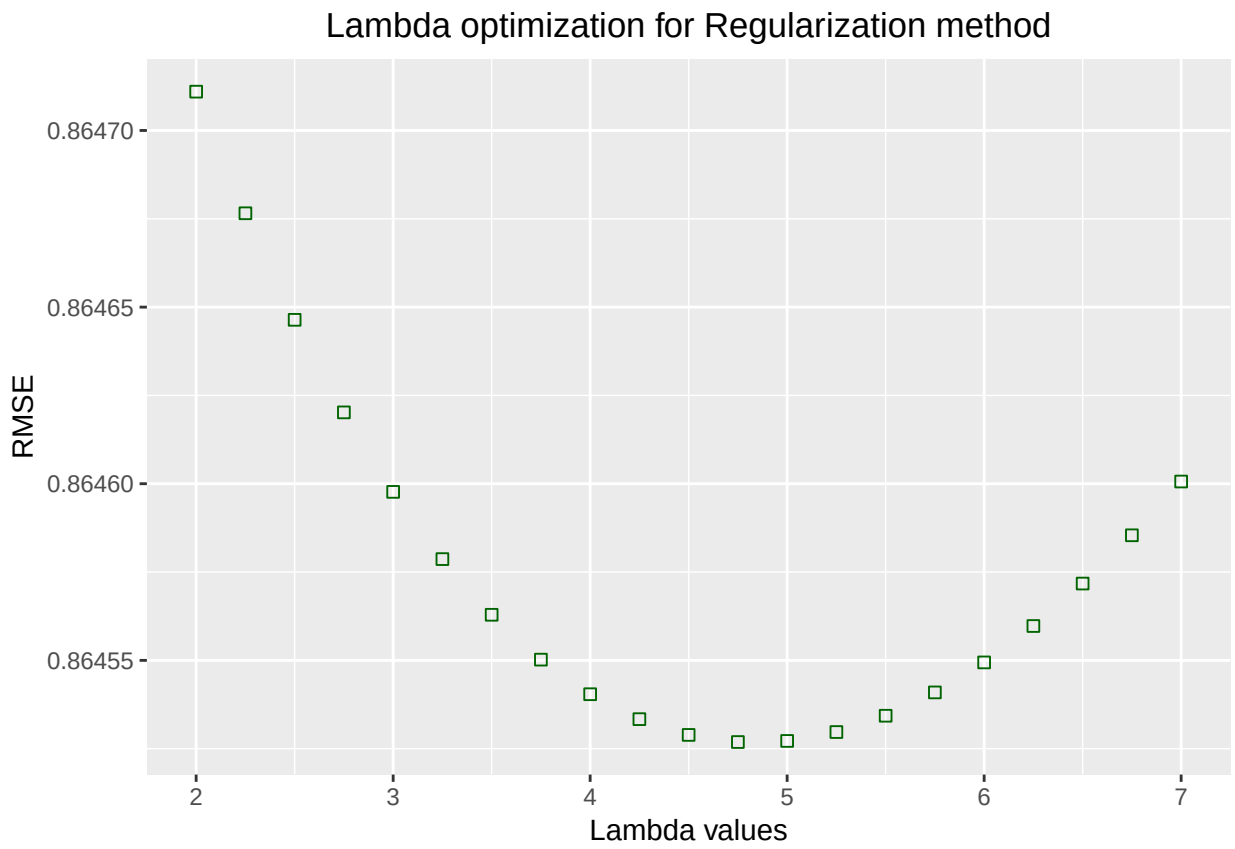
To look for our tuning parameter lambda we use this code:

```
lambdas <- seq(2, 7, 0.25)

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

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

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

  predicted_ratings <- test_set %>% left_join(b_i, by = "movieId") %>%
                                    left_join(b_u, by = "userId") %>%
                                    mutate(pred = mu + b_i + b_u) %>% .$pred

  return(RMSE(predicted_ratings, test_set$rating))
})
```

[Note: we started with a larger interval, from 0 to 10 and then we optimized it to find the minimum so we reduced the computing time]

We create a dataframe with all the lambda values and their respective RMSEs, then we check what lambda corresponds to the minimum value of RMSE.

```
lambdas_df <- data.frame(Lambda_values = lambdas, RMSES = rmses)

ggplot(lambdas_df, aes(x = Lambda_values, y = RMSES)) +
      geom_point(size = 2, shape = 22, color = "darkgreen") +
      labs(title = "Lambda optimization for Regularization method",
           x = "Lambda values", y = "RMSE") +
      theme(plot.title = element_text(hjust = 0.5))
```




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

```
## [1] 4.75
```

The best lambda value in this case is 4.75.

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method =
                                       "Movie + User Effects Model with Regularization",
                                     RMSE = min(rmses)))

rmse_results
```

```
## # A tibble: 4 x 2
##   Method                                          RMSE
##   <chr>                                          <dbl>
## 1 Model-based Approach                           1.06
## 2 Movie Effect Model                             0.943
## 3 Movie + User Effects Model                     0.843
## 4 Movie + User Effects Model with Regularization 0.865
```

We didn't get a better RMSE for this method, maybe it is due to the size of the sample, or the little variation between parameters that it represents. Still, don't lose hope! We will try it on the validation dataset.

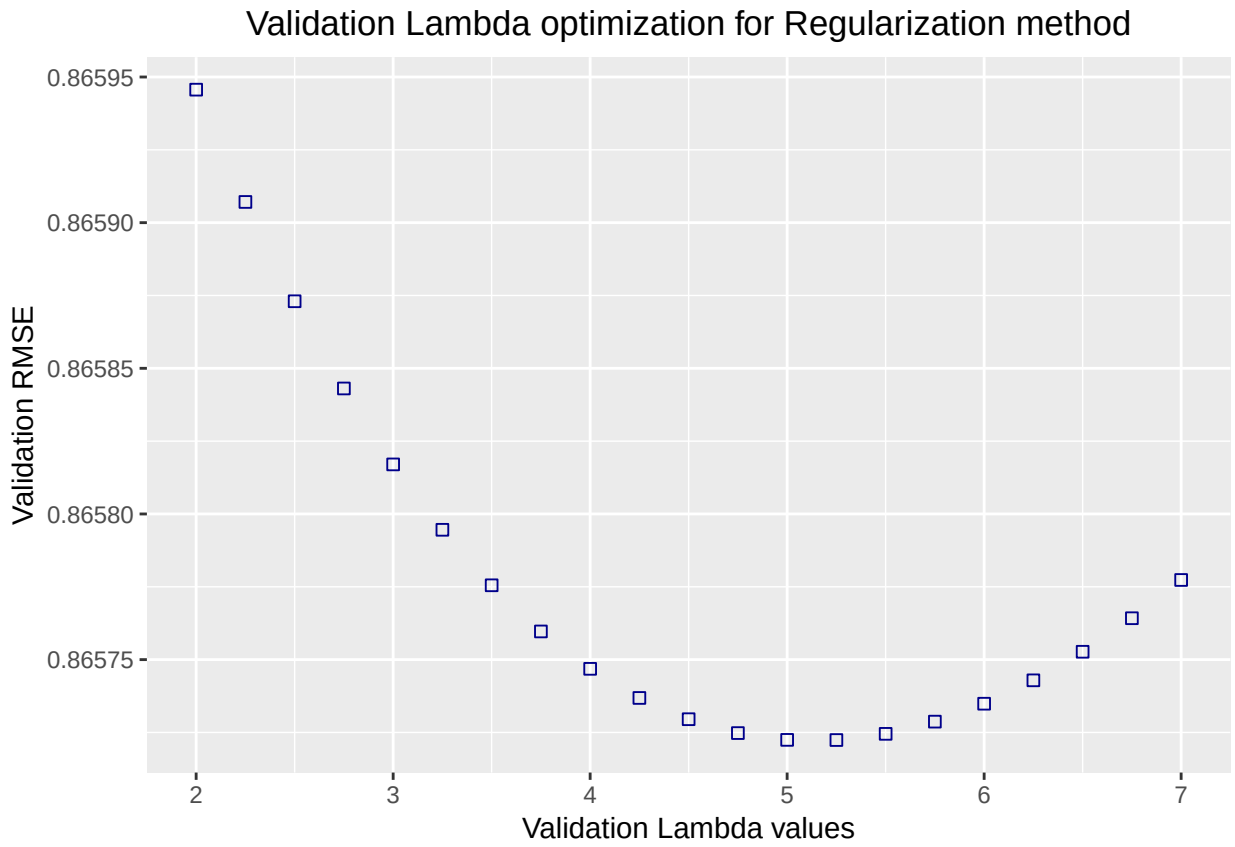
5.2. Regularization on the validation dataset

For this part of the report, we will reproduce what we have achieved in 5.1, but applied to the validation set of data.

```
lambdas_validation <- seq(2, 7, 0.25)

rmses_validation <- sapply(lambdas_validation, function(l){

  mu <- mean(train_set$rating)

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

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

  predicted_validation_ratings <- validation %>% left_join(b_i, by = "movieId") %>%
                                    left_join(b_u, by = "userId") %>%
                                    mutate(pred = mu + b_i + b_u) %>% .$pred

  return(RMSE(predicted_validation_ratings, validation$rating, na.rm = T))
})
```

We create a dataframe with all the lambda values and their respective RMSEs and we plot it, as we did for the edx set.

```
lambdas_validation_df <- data.frame(Validation_Lambda_values = lambdas_validation,
                                    Validation_RMSES = rmses_validation)

ggplot(lambdas_validation_df, aes(x = Validation_Lambda_values, y = Validation_RMSES)) +
      geom_point(size = 2, shape = 22, color = "darkblue") +
      labs(title = "Validation Lambda optimization for Regularization method",
           x = "Validation Lambda values", y = "Validation RMSE") +
      theme(plot.title = element_text(hjust = 0.5))
```




```
opt_lambda_validation <- lambdas_validation[which.min(rmses_validation)]
opt_lambda_validation
```

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

The best lambda value in this case is 5.25.

```
rmse_validation_results <- bind_rows(rmse_validation_results,
                           data_frame(Validation_Method =
                                        "Movie + User Effects Model with  Regularization",
                                      RMSE = min(rmses_validation)))
rmse_validation_results
```

```
## # A tibble: 2 x 2
##   Validation_Method                                RMSE
##   <chr>                                           <dbl>
## 1 Movie + User Effects Model                      0.882
## 2 Movie + User Effects Model with  Regularization 0.866
```

So our final RMSE in the validation set, for the "Movie + User Effects Model with Regularization", is $0.8657224 < 0.8775$, which was our goal from the start.

6. Conclusions and Discussion

The objective of this report was get to know the MovieLens dataset we were provided with, play a little with the variables and the parameters we had to get to know the data, and finally create a Recommendation System using this data.

We went through the data on the first sections, and then tried different approaches to get a low RMSE value, starting from the simplest approach possible: the Model-Based approach, to the Movie + User Effects Model with Regularization method (the most complex one, since it takes into account different parameters).

With the userId and modelId variables, followed by a regularization process, to find the best tuning parameter lambda possible, we arrived to the best RMSE value attained on this report, 0.8657224, thus achieving our goal to fin a RMSE lower than 0.8775.

7. Personal Comments

Some comments I wanted to write about this project: I liked very much doing it, playing with the data and the graphs, to explore and put into this project all the hard work done on the past few months. When I started with this whole Data Science course, I didn't even know how to install Rstudio, and now I'm so proud that I am able to write a whole report, do some analysis on the data and finally create a Recommendation System, and even more, try to improve its RMSE. Thank you all and I hope you enjoyed this report!

GitHub repo: https://github.com/rocioam/Capstone-MovieLens-