HarvardX : MovieLens Project

Jordan Porcu

Contents

Introduction

The project

This document is done within the framework of the *HarvardX PH125.9x Data Science : Capstone* certification. This first project is an imposed subject : Rating prediction for movies.

The data set we are going to use is also already given.

As the subject implies, the main goal of this project is to create a machine learning algorithm able to predict the rating of a movie, based on different variables. We will do so using RStudio, and R language. The second aim of the project is evaluating and minimizing the error on the predictions with the Root Mean Squared Error, also known as RMSE.

Data set creation

This part is already given in the project instructions. It consists of downloading a .zip folder on the Internet, then extracting the .dat file and applying some modifications so we have our clean data set. It can be done with this piece of code :

```r
# Loading the packages we will need during the project
library(tidyverse)
library(caret)
library(data.table)

# Creating a temporary file that will be the downloaded folder
dl <- tempfile()

# Assigning the URL outcome to "dl"
download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

# Creating a dataframe from "dl" using columns such as
# "userId","movieId","rating" and "timestamp" in the "ratings.dat" file
# Warning : this might take several minutes to achieve
ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

# Creating a table from "dl" with datas that are located in the "movies.dat" file
movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)

# Assigning names to "movies" columns
colnames(movies) <- c("movieId", "title", "genres")

# Converting "movies" in a dataframe with right columns names and types
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))

# Joining "movies" and "ratings" with "movieId" as the common colmun
movielens <- left_join(ratings, movies, by = "movieId")

# Setting the seed to 1 for randomness reproductibility
set.seed(1)

# Creating a list of indexes to split the "movielens" data set in train and test parts
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
```

```r
# "edx" will be our training set (90% of the original data set)
edx <- movielens[-test_index,]

# "temp" be our test set (10% of the original data set)
temp <- movielens[test_index,]

# "validation" is our new test set, where all the datas are also in "edx"
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Adding to "edx" what was removed from the "temp" set
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

# Removing variables we wont use anymore
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

After this operation, we have all we need to fulfill our goals.

- **edx** will be the data set we train the model with.
- **validation** will be the data set we test the model on.

RMSE

The Root Mean Squared Error is an often used measure of the error on a prediction. It is defined by the square rooted mean of the sum of the squared errors. The lower RMSE is, the better the model is. Mathematically, it is written like this :

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

Therefore, we create a function to display the RMSE of a prediction with this piece of code :

```r
rmse <- function(prediction,test) {sqrt(mean((test-prediction)^2))}
```

In brief, we now have a data set, split in a training set (edx) and a test set (validation). Moreover, we have a function that displays the RMSE of any prediction we make, with any model we want. We can now proceed to model building, and achieve the goals we set before.

Analysis and model building

Data Analysis

Now that we have our data sets, we can explore it to anticipate the method we will use. Here is a first look at the head of the data set

```
## Number of rows :  9000061
```

```
## Number of columns : 6
```

userId	movieId	rating	timestamp	title	genres
1	122	5	838985046	Boomerang (1992)	Comedy|Romance
1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
1	231	5	838983392	Dumb & Dumber (1994)	Comedy
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi

As we see, the final data set is composed of 9 millions of data and 6 variables :

- *userId* : The ID number of the user who rated the movie.
- *movieID* : The ID number that refers to the rated movie.
- *rating* : The rating the movie received by the user. This is what we will predict.
- *timestamp* : The timestamp of when the rating was done.
- *title* : The title of the movie corresponding to the ID.
- *genres* : The different genres of the movie.

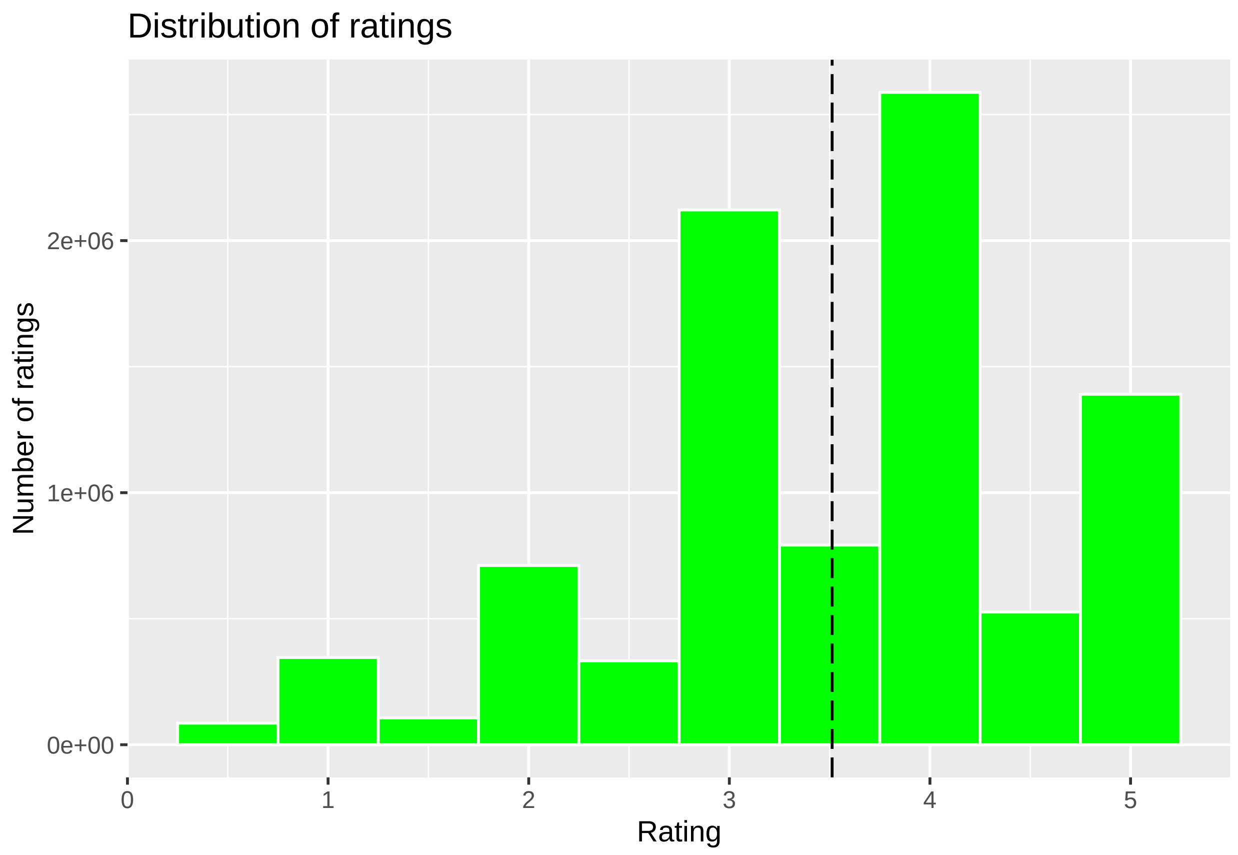
Before we begin, we want to make sure the data set is clear. This is why we compute the number of NaN values in the data set. The output must be 0 if we want to state that we use a clear set.

```
sum(is.na(edx))
```

```
## [1] 0
```

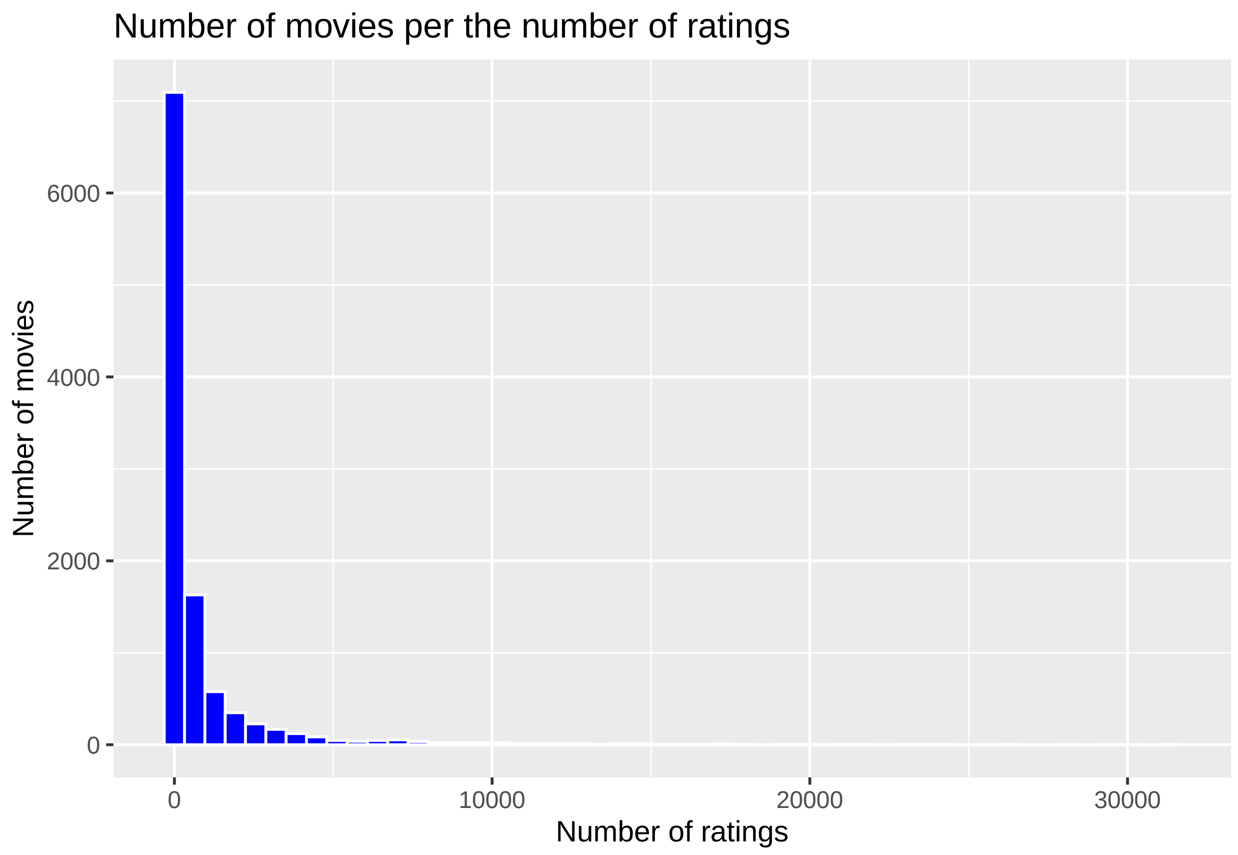
Yes, it is. Now we know our data set is clean, so we can display some useful statistics and analysis.

Here is a first plot of the ratings distribution.



The dashed line represents the mean of ratings, μ. As we can see, ratings tends to be between 3 and 5. In fact, the mean is 3.51 (μ). It's an important first step to have an intuition about the prediction model, but we will tell more later.

Next, we will study how many ratings the movies receive. To do so, we compute a plot of the number of movies per number of ratings :

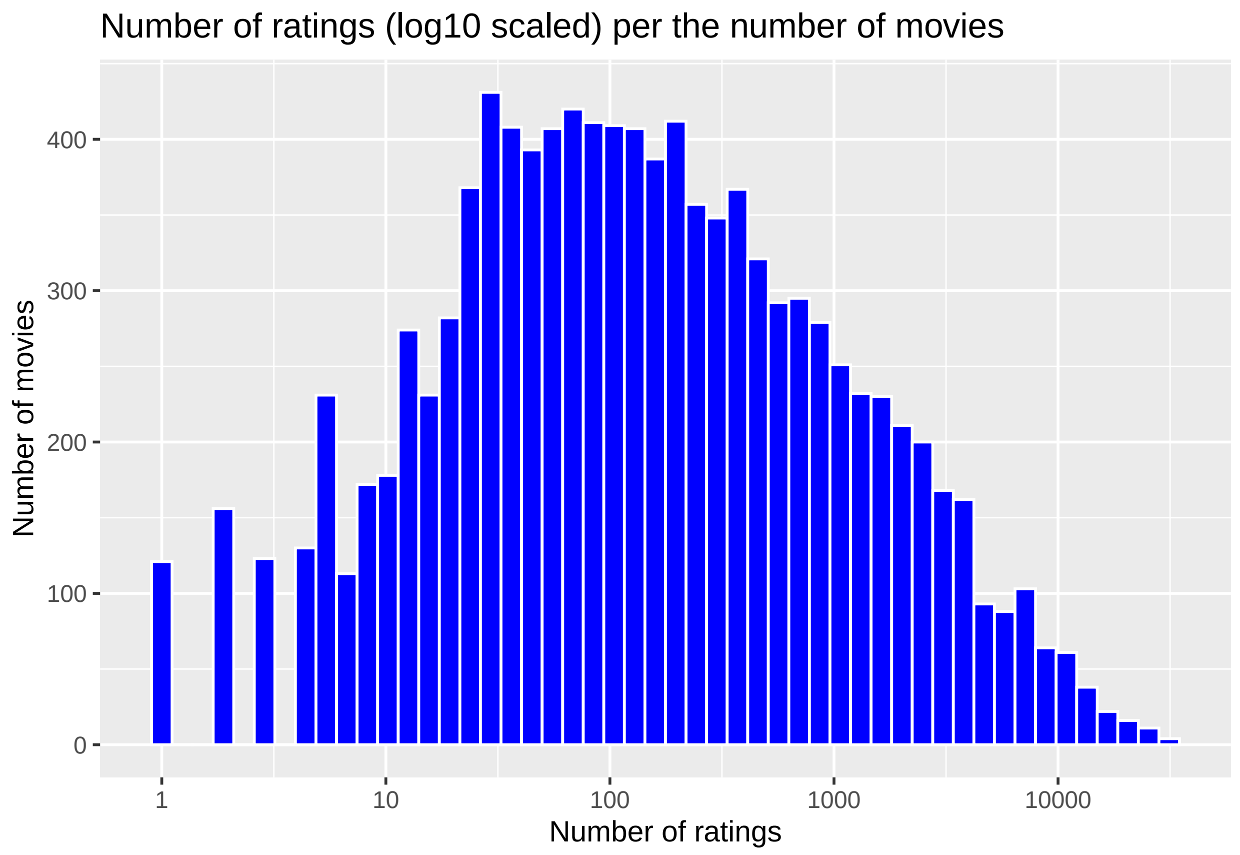


We clearly see that this plot is not usable because of a high peak near 0, then a fast decreasing. To understand why this happens, we need to have a look to some statistics about the number of ratings movies received :

```
edx %>%
  count(movieId) %>%
  summary()
```

```
##     movieId            n
##  Min.   :    1   Min.   :    1.0
##  1st Qu.: 2754   1st Qu.:   30.0
##  Median : 5434   Median :  122.0
##  Mean   :13105   Mean   :  842.9
##  3rd Qu.: 8710   3rd Qu.:  563.0
##  Max.   :65133   Max.   :31336.0
```

We have our answer : the maximum of the number of ratings is over 30,000 but the third quartile is 563, which means that 75% of the values are under 563. It is clear now that those extreme values (the maximum) causes this plot issue. A solution to enhance the plot is to use log10(x) instead of x (for x axis) :



This is better as it clearly allows us to see the distribution. Most of the movies received between 50 an 500 ratings.

Model building

Among the variables we have at disposal, we can ask the question that will rule the model building : what variable may have an impact on the rating ?

Let's try to get an logical thinking about the different variables :

- *userId* : any user will give ratings based on his movie tastes, so we can imagine that the Id of the user we are studying the predicted rating can have an impact on this prediction.
- *movieId* : some movies are more famous than other, some are more appreciated than other. So, we can state that the predicted rating will be affected by the movie id.
- *timestamp* : it could have been really interesting to examine if the rating is correlated to the time between the movie release date and the rating date. Unfortunately, we only have the date of the rating, so we won't be able to use this variable.
- *title* : this variable is clearly used in its ID form, which is way easier to analyse.
- *genres* : in this case, it is interesting because we don't have one genre per movie, but genres groups. Therefore, we can imagine that some genres combinations are more suitable for all than others. This is why we will also see if the genres combination has an impact of predicted rating.

In conclusion, three variables are interesting to study rating prediction. The method is to analyze the RMSE for each model. The first one will be a naive intuition : the predicted rating is merely the rating mean. From this, we can plot biases for each variable we want to study, and then adapt the model from this to optimize the variable selection. Then, we will look for a way to optimize this model.

Naive model

The first model, as previously said, is simply a model that gives the rating mean value to the predicted rating. The mathematical expression for such a model is written as :

$$\hat{y}_{i,j} = \mu + \epsilon_{i,j}$$

with :

- $\hat{y}_{i,j}$: the predicted rating
- μ : the mean value of training set ratings (defined by `mu <- mean(edx$rating)`)
- $\epsilon_{i,j}$: the error on each rating value (difference between predicted and mean)

As seen above, the mean is ~3.51. We can now compute the RMSE of this model :

```
rmse_naive <- rmse(mu,validation$rating)
rmse_naive
```

```
## [1] 1.060651
```

According to the project instruction, this RMSE seems a bit high. This was predictable since this model, called the "naive" model, is meant to have a first view at what the model will look like in the end, but it is not optimized at all.

Movie effect

Intuitively, the first variable we need to study is the movie ID. Why ? Because we need to consider the ratings for each movie to predict future ratings. The model, by his mathematical expression, looks pretty close to the naive one :

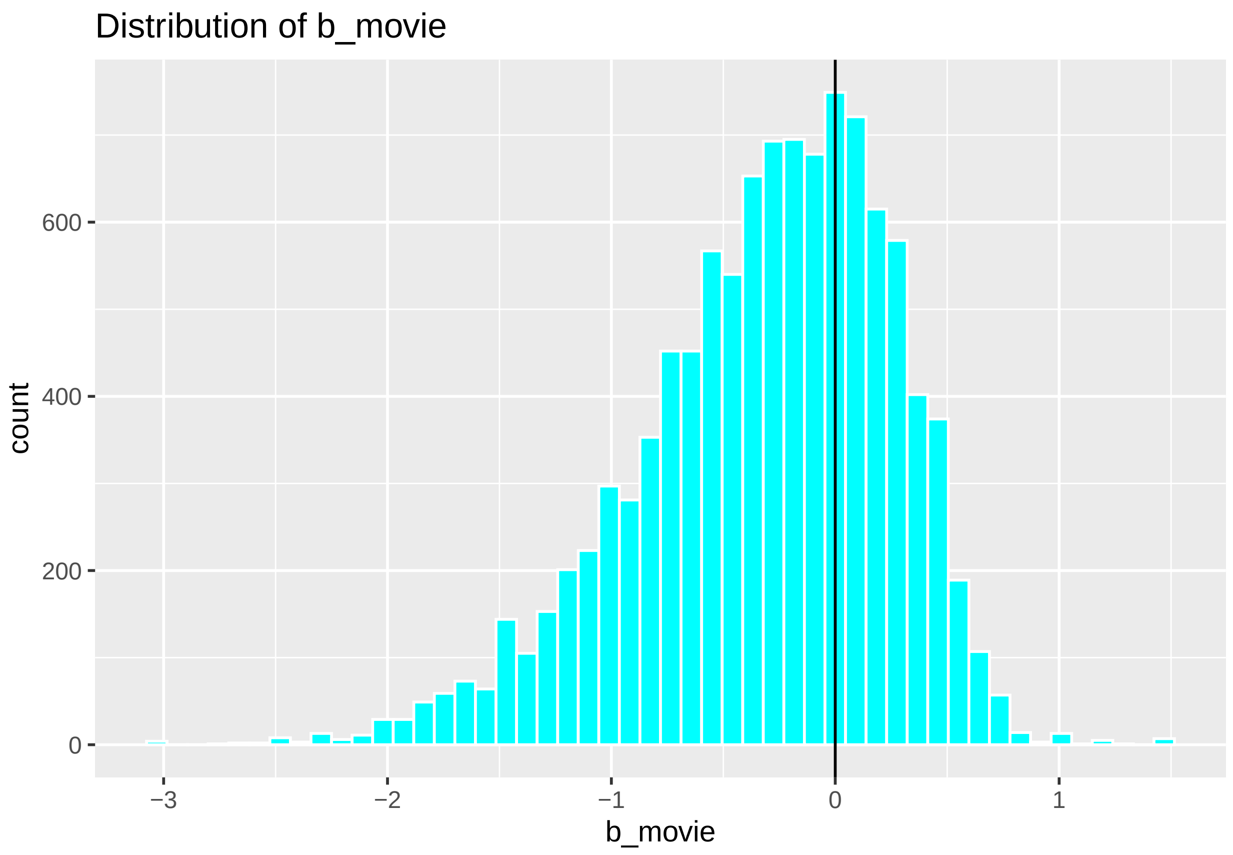
$$\hat{y}_{i,j} = \mu + \beta_{movie} + \epsilon_{i,j}$$

with β_{movie} : the bias induced by the movieId variable

We create the β_{movie} vector with this piece of code :

```
b_movie <- edx %>%
  group_by(movieId) %>%
  summarize(b_movie = mean(rating - mu))
```

Since β_{movie} is the difference between the ratings (grouped by movieId) and μ, we can show the distribution of β_{movie} to see that it has, indeed, an impact on the rating prediction (we will do the same method for next studied variables) :



Now, the method to get the RMSE is a little more complex than before :

```
predict_movie <- mu + validation %>%
  left_join(b_movie, by='movieId') %>%
  pull(b_movie)

rmse_movie <- rmse(predict_movie,validation$rating)
rmse_movie
```

```
## [1] 0.9437046
```

This RMSE is better, but it is not as good as we want it. However, the model building is not over yet.

Movie & user effect

Now that we grouped by movieId, let's see how it evolves with a user effect. The mathematical expression becomes :

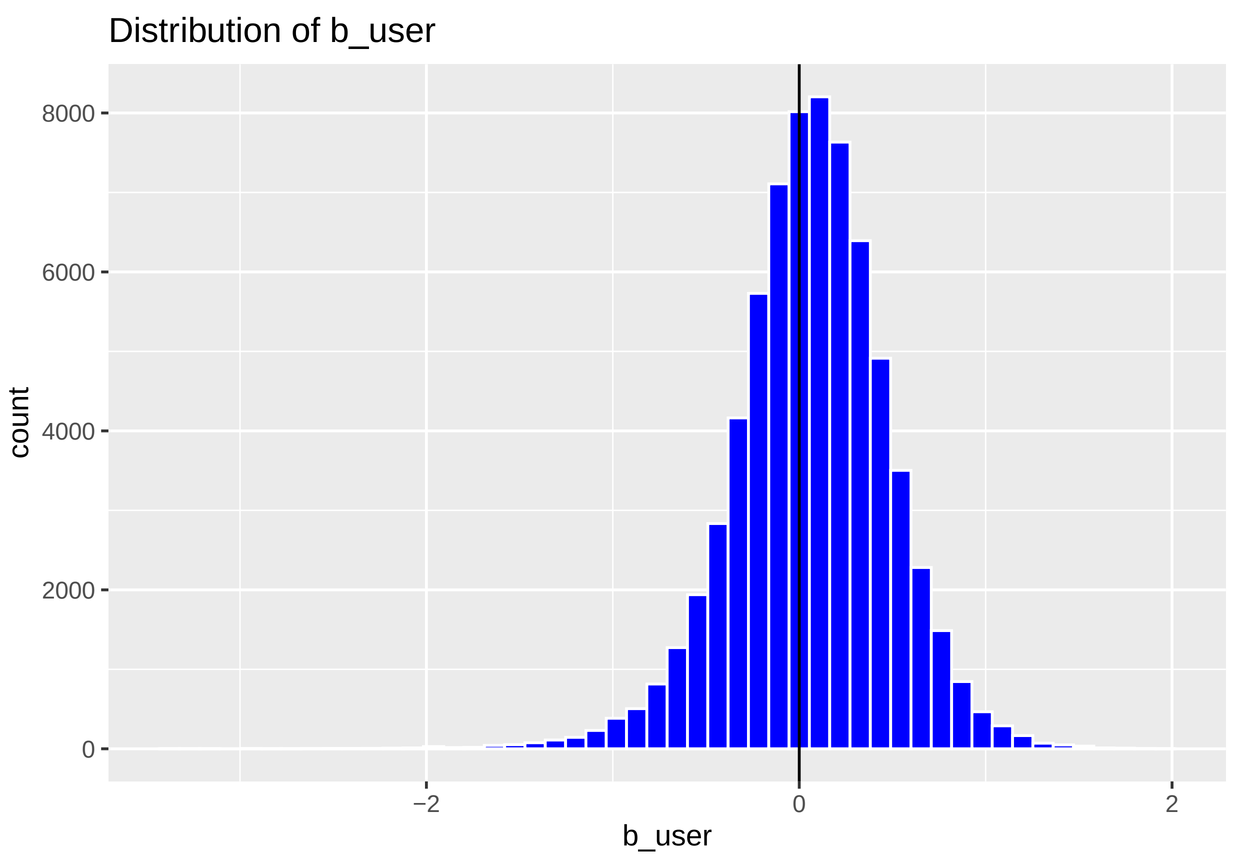
$$\hat{y}_{i,j} = \mu + \beta_{movie} + \beta_{user} + \epsilon_{i,j}$$

with β_{user} : the bias induced by the userId variable

We create β_{user} vector with this code :

```
b_user <- edx %>%
  left_join(b_movie, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_user = mean(rating - mu - b_movie))
```

The distribution of β_{user} can be plotted like this :



We observe as plot that looks close to the movie effect one. The main difference is a smaller standard deviation.

Now we can compute the RMSE for this new model :

```
pred_user <- validation %>%
  left_join(b_movie,by="movieId") %>%
  left_join(b_user,by="userId") %>%
  summarize(pred = mu+b_user+b_movie) %>%
  pull(pred)

rmse_user <- rmse(pred_user,validation$rating)
rmse_user
```

```
## [1] 0.8655329
```

We are coming close to a high quality. Now, to finish the variable selection part, we will add the genres effect.

Movie & user & genres effect

Genres are displayed as groups. So we have two choices :

1. studying genres as genres combinations, so there will be a difference between "Action" and "Action|Thriller" for example. We will have then a lot of different unique genres (to be understood as unique genres combinations)
2. split the "genres" column in columns with unique genres as names, and a value of 1 if the movie is from this genre, and 0 if not. We would have a new data frame with a lot of columns. The model would be very complex, which is good but with our method, it would cause a high risk of **overfitting**, and we do not want to let it happens.

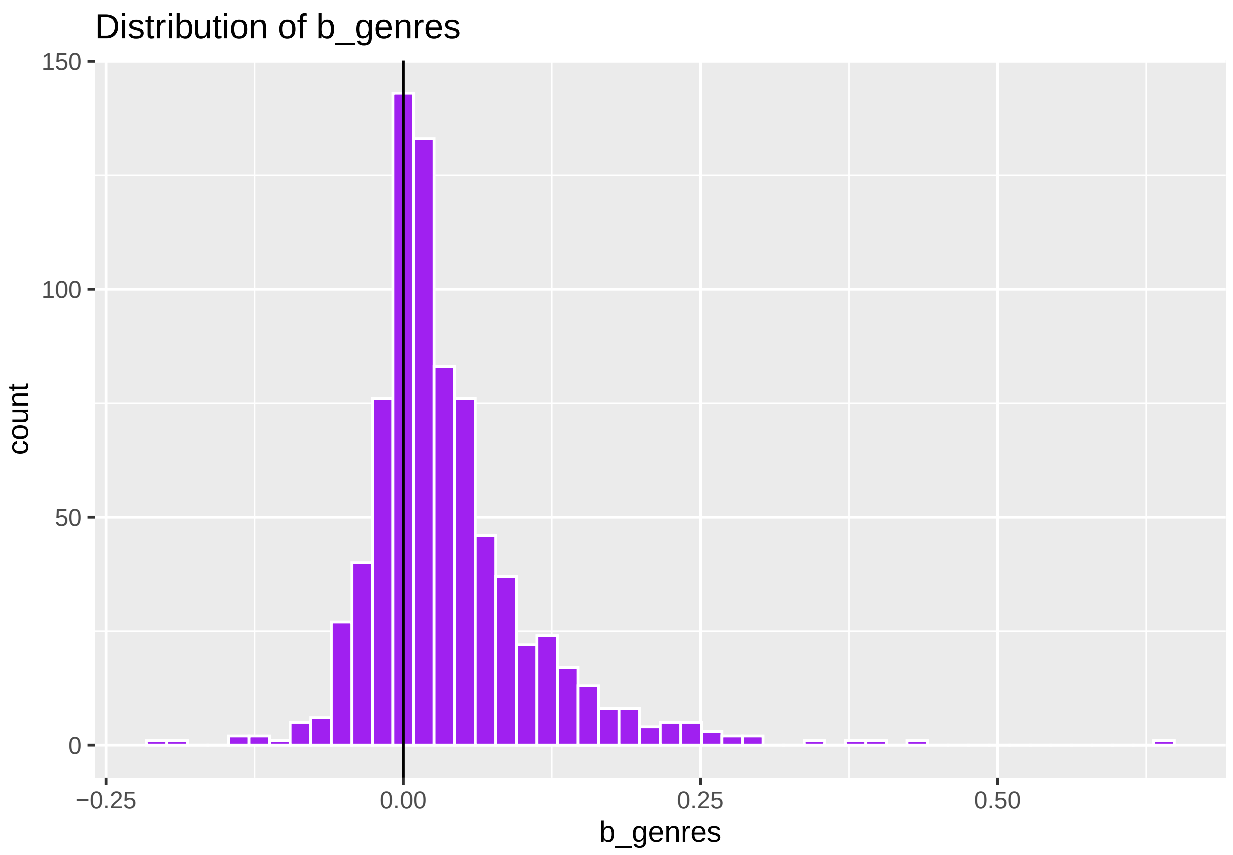
This is why we will choose the first option : treat genres as genres combinations. The mathematical expression for this model is :

$$\hat{y}_{i,j} = \mu + \beta_{movie} + \beta_{user} + \beta_{genres} + \epsilon_{i,j}$$

We create β_{user} vector with this code :

```
b_genres <- edx %>%
  left_join(b_movie, by='movieId') %>%
  group_by(userId) %>%
  left_join(b_user,by="userId") %>%
  group_by(genres) %>%
  summarize(b_genres = mean(rating - mu - b_movie - b_user))
```

The distribution of β_{genres} can be plotted like this :



Again, β_{genres} is distributed around zero, in the same way as β_{movies} and β_{user}, with an even smaller standard deviation. By the way, we can confirm that the standard deviation of biases decrease as we build the model with this code :

```
biases <- c("Movie effect","Movie + user effect", "Movie + user + genres effect")
sd <- c(sd(b_movie$b_movie),sd(b_user$b_user),sd(b_genres$b_genres))
sd_df <- tibble(Bias = biases, SD = sd) %>%
  knitr::kable()

sd_df
```

Bias	SD
Movie effect	0.5711631
Movie + user effect	0.4117163
Movie + user + genres effect	0.0721715

Now we can compute the RMSE for this model :

```
pred_genres <- validation %>%
  left_join(b_movie,by="movieId") %>%
  left_join(b_user,by="userId") %>%
  left_join(b_genres, by="genres") %>%
  summarize(pred=mu+b_movie+b_user+b_genres) %>%
  pull(pred)

rmse_genres <- rmse(pred_genres,validation$rating)
rmse_genres
```

```
## [1] 0.8651946
```

Again, we have a much better RMSE. We considered all the variables we could exploit. The final task is to find a way to optimize this model to use all of its potential.

Final model optimization

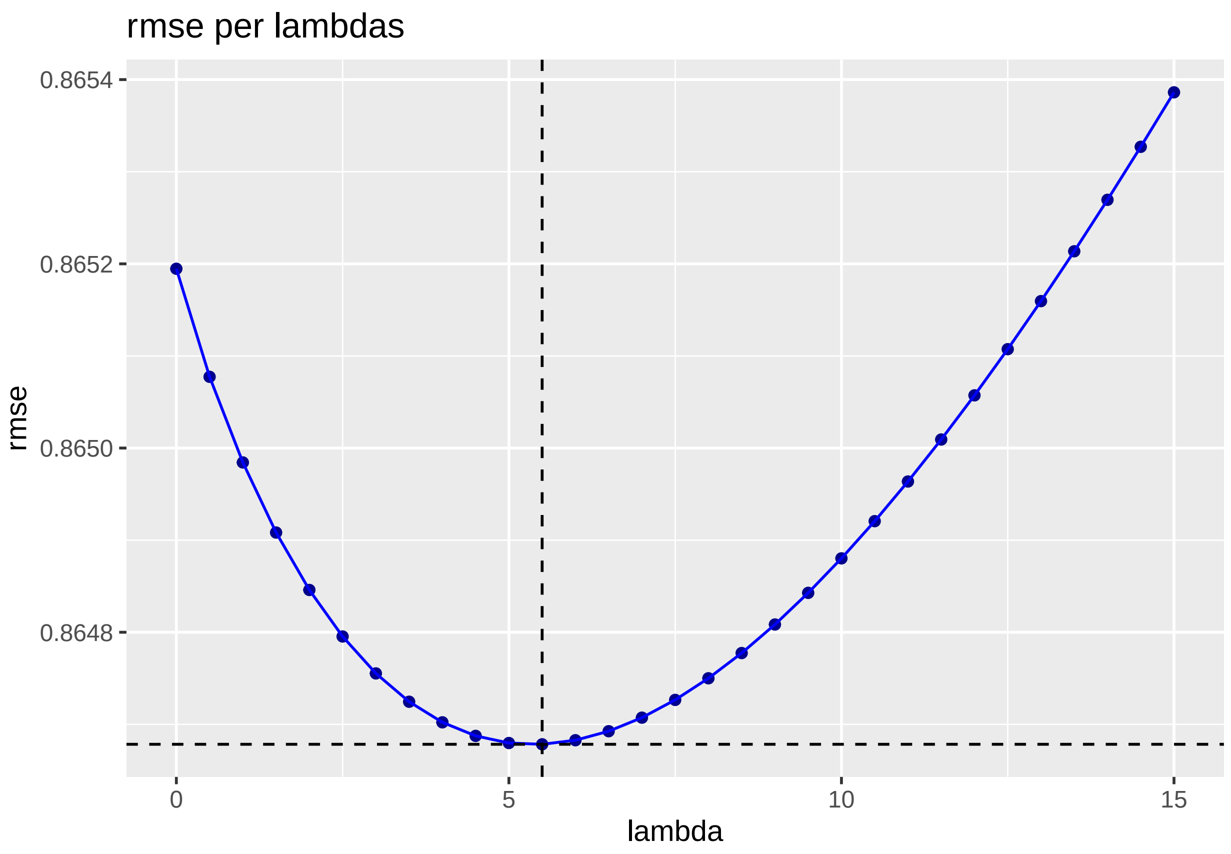
In the data analysis part, we saw that the number of ratings per movies is not normally distributed. What does it means ? It means that a lot of movies have a small number of ratings. How does it skew our model ? A small number of ratings implies a large ratings interval, which cause the model to consider thoses movies as important as movies with a large number of ratings.

How can we optimize this ? We can use a method called **regularization**. It consists of adding a λ factor to reduce the weight of small number of ratings movies.

To find this λ factor, we recreate our model with a defined list of lambdas we test, and we use the one that minimize the RMSE of our model. To do so, we use this code (which explains clearly how we use λ) :

```
list_of_lambdas <- seq(0,15,0.5)

rmses_lambdas <- sapply(list_of_lambdas,function(lambda){

  # same as previous model building but mean() divides the sum by n(), so we we divide by
  # lambda +n() instead to add the lambda weight

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

  b_user <- edx %>%
    left_join(b_movie, by='movieId') %>%
    group_by(userId) %>%
    summarize(b_user = sum(rating - mu - b_movie)/(lambda+n()))

  b_genres <- edx %>%
    left_join(b_movie, by='movieId') %>%
    group_by(userId) %>%
    left_join(b_user,by="userId") %>%
    group_by(genres) %>%
    summarize(b_genres = sum(rating - mu - b_movie - b_user)/(lambda+n()))

  prediction <- validation %>%
    left_join(b_movie,by="movieId") %>%
    left_join(b_user,by="userId") %>%
    left_join(b_genres, by="genres") %>%
    summarize(pred=mu+b_movie+b_user+b_genres) %>%
    pull(pred)

  return(rmse(prediction,validation$rating))
})
```

We can now create a plot to find the λ that minimize the rmse of our model :



And we can find it with :

```
best_lambda <- list_of_lambdas[which.min(rmses_lambdas)]
best_rmse <- rmse_lambda_df %>%
  filter(lambda==best_lambda) %>%
  pull(rmse)

final_model_result <- tibble("Best lambda"=best_lambda, "Corresponding rmse"=best_rmse) %>%
  knitr::kable()

final_model_result
```

Best lambda	Corresponding rmse
5.5	0.8646784

Results

Finally, we can display all of our results, including the final model one, in a data frame :

Model	RMSE
Naive	1.0606506
Movie only	0.9437046
Movie and user	0.8655329
Movie, user and genres	0.8651946
Optimized final model	0.8646784

As we can see 0.8646784 (best_rmse) $<$ 0.86490 (project instructions) so we can state that this model is correct to predict ratings. The main reason we can't get below this range of rmse is the lack of variables, and accuracy through the ones we had.

Conclusion

The machine learning model we created over this report is considering three main variables :

- **movieId** : the predicted rating is impacted by the movie feature
- **userId** : after using the movie feature, the model uses the user id to predict the rating according to the movie and user features
- **genres** : finally, the model also uses genres combination feature to predict the rating

In a last part, we optimized the model we found and we obtained the final RMSE :

$$rmse_{final} = 0.8646784$$