Movie Recomendation System

Jesús Aguerri

11/9/2019

1. INTRODUCTION

This proyect aim to create an algoritm that works as movie recomendation system. Using a machine learning based approach we will try to build and algorihtm that, receiving data as inpout, will can predict movie ratings as ouput. The data that we will use have been provided by the staff of the HervardX: PH125.9x Data Science: Capstone course. This dataset is composed by almost ten millions movies ratings (from 1 to 5 stars) generated by more than sixtynine thousand users. Root Mean Square Error (RMSE) will be the value used for evaluate the algorithms.

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

Our goal will be obtain the minimum RMSE possible. In consecuence, we first will explore the data traying to find questions that could help us to design our models. Later, we will check differnt models, traying to see which one minimize the RMSE.

2. TO GET THE DATA AND TO CREATE SETS

In this proyect we will use the following packages:

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

We will use the MovieLens 10M dataset, which is avaible online in the following links: https://grouplens.org/datasets/movielens/10m/ http://files.grouplens.org/datasets/movielens/ml-10m.zip

To get the data and to create the sets we will use the following code, whixh is provided in the edx capstone project module:

```
# 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")

# 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")
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)
```

This code have created two objects: -edx: the train set -validation: the test set We save the botch objects as rda files, this will be easier to run them later:

```
save(edx, file= "rda/edx.rda")
save(validation, file= "rda/validation.rda")
```

2.1 EXPLORING THE DATA

Using the functions "head()" and "str()" we can see the edx data structure

```
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
```

```
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 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
```

As we see this subset of the data have 6 variables in columns and 9.000.055 observations in rows. Each observations belongs to a rating given by one user to a movie. The variables that are included in the data set are:
-"userId": user identification
-"movieId": movie identification
-"rating": the rate given by a user to a movie
-"timestamp": the date and hour when the rating was done
-"tittle": the tittle and year of the movie
-"genre": the movie´s genre.
Going a bit deeply, we can use summary() to see the structure of each variable

```
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
##
##
##
```

And we also can see the number of unique movies and users in the dataset

```
edx %>% summarise(n_movies = n_distinct(movieId),
                  n_users = n_distinct(userId))
```

```
##   n_movies n_users
## 1    10677   69878
```

3. ANALYSIS

We is going to explore some characteristcis of the data. Our goal is to find bias that could help us to adjust or algorihtm.

3.1. Rating Distribution

Exploring the rating distribution we can see that user gave high ratings more often than low rating, and aslo can see that half ratings are less commom that whole star ratings.

```
edx %>% group_by(rating) %>% summarise(count = n()) %>% arrange(desc(count))
```

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

Using the function "mean()" we can get the mean rating is around 3,5 starts. As we´ll see later, to use the mean to predict rating is the simplest approach posible to predict ratings.

3.2. Ratings per movie

There are movies with a lot of ratings

```
edx %>% group_by(title) %>% summarise(count= n()) %>% arrange(desc(count))
```

```
## # A tibble: 10,676 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
## # ... with 10,666 more rows
```

However, there are movies even with just one rating.

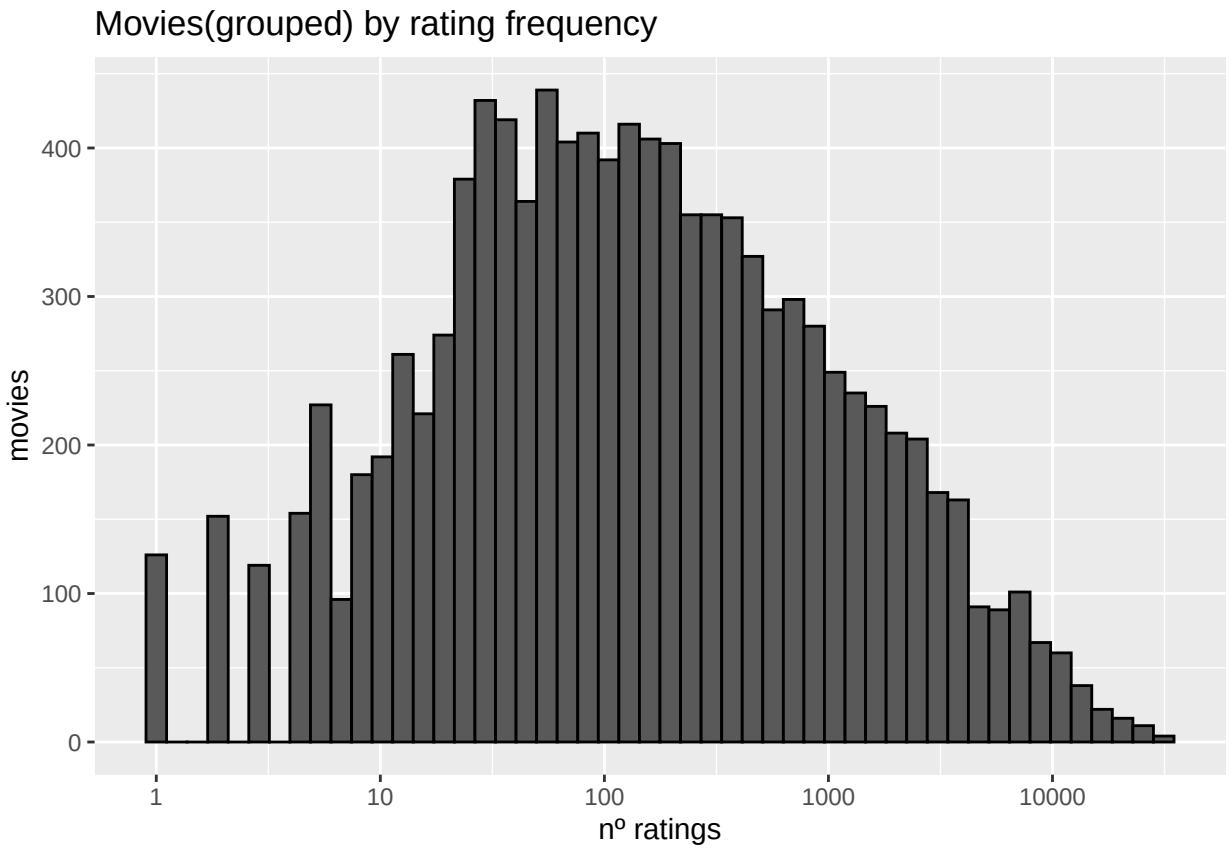
```
edx %>% group_by(title) %>% summarise(count= n()) %>% arrange(count)
```

```
## # A tibble: 10,676 x 2
##    title                                                     count
```

```
##    <chr>                                                                <int>
##  1 1, 2, 3, Sun (Un, deuz, trois, soleil) (1993)                            1
##  2 100 Feet (2008)                                                          1
##  3 4 (2005)                                                                 1
##  4 Accused (Anklaget) (2005)                                                1
##  5 Ace of Hearts (2008)                                                     1
##  6 Ace of Hearts, The (1921)                                                1
##  7 Adios, Sabata (Indio Black, sai che ti dico: Sei un gran figlio d~       1
##  8 Africa addio (1966)                                                      1
##  9 Aleksandra (2007)                                                        1
## 10 Bad Blood (Mauvais sang) (1986)                                          1
## # ... with 10,666 more rows
```

The following plot is usefull to visualize that question

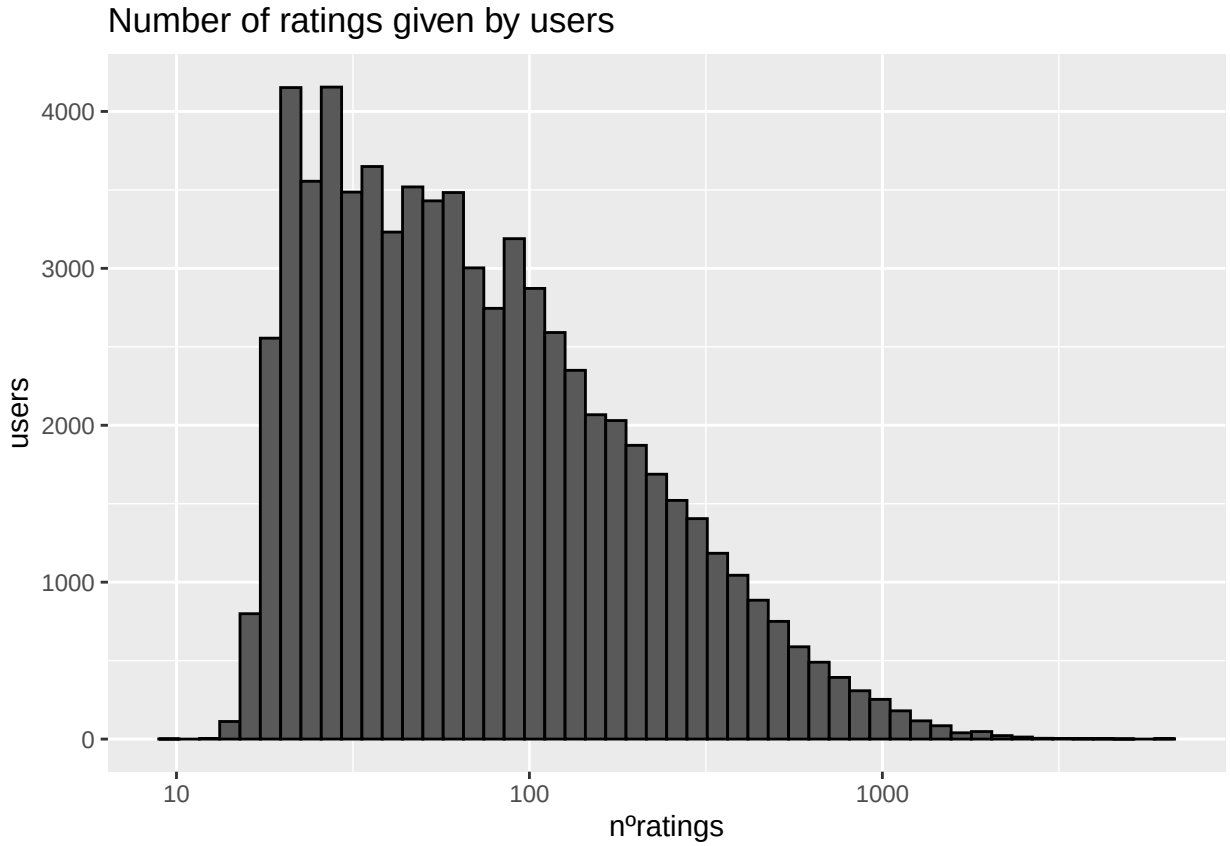
```
edx %>%
  count(movieId) %>%
  ggplot(aes(n))+
  geom_histogram(bins = 50, color = "black") +
  scale_x_log10() +
  ggtitle("Movies(grouped) by rating frequency") +
  xlab("nº ratings")+
  ylab("movies")
```



3.3. Ratings per user

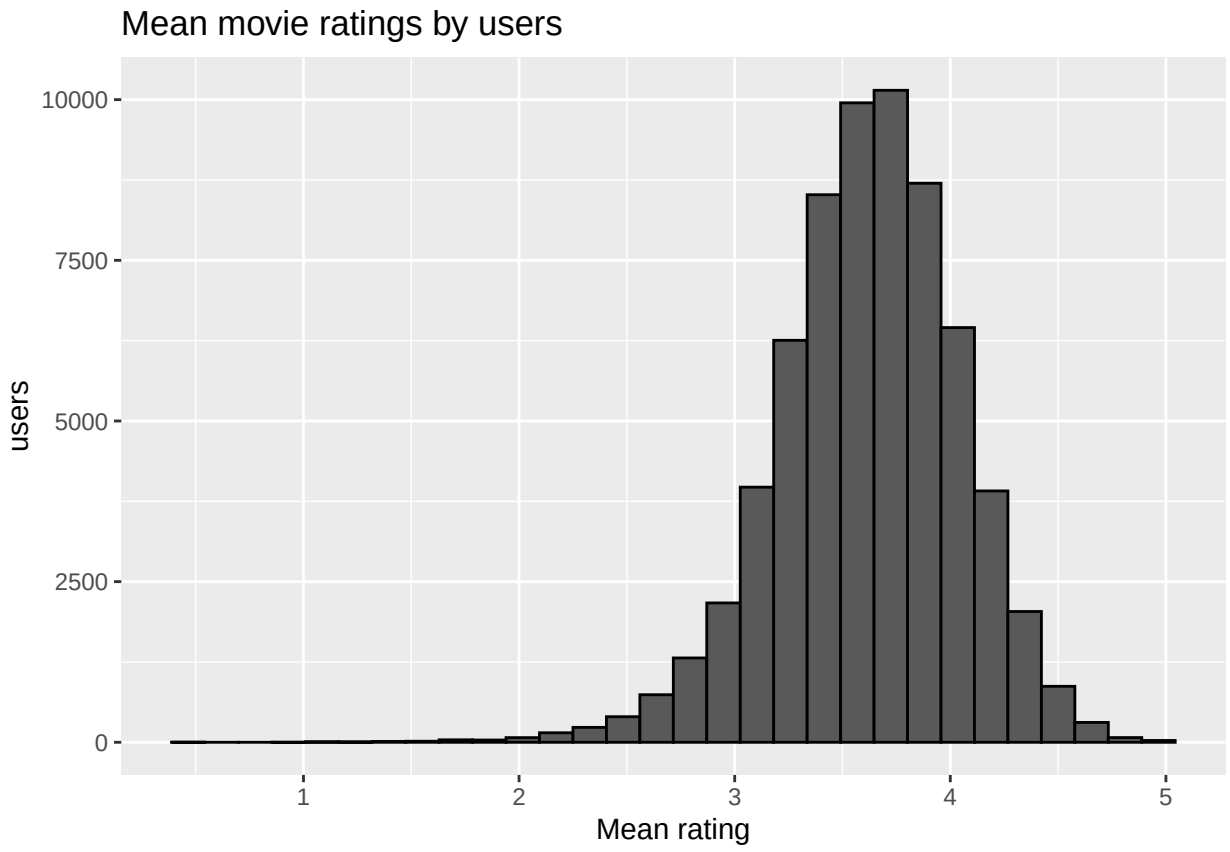
We also can see that there are user that are more actives than others

```
edx %>%
  count(userId) %>%
  ggplot(aes(n))+
  geom_histogram(bins = 50, color = "black")+
  scale_x_log10() +
  ggtitle("Number of ratings given by users")+
  xlab("nºratings")+
  ylab("users")
```



In addition, there are user who give better puntuations and other who are more critical

```
edx %>%
  group_by(userId) %>%
  filter(n() >= 20) %>% #just take user with more tahn 20 ratings
  summarize(b_u = mean(rating)) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black") +
  xlab("Mean rating") +
  ylab("users") +
  ggtitle("Mean movie ratings by users")
```



4. Methods: modeling approach

As we already say we are going to use the Root Mean Square Error (RMSE) as value to evaluate the algorithm performance. Acording to the information that we have extract during the data exploration, we are goint to try to develop diferent algorithms. This algorithms will take into acount the bias that we are detected previously and they will try to minimize the RMSE. The following function will be used to compute the RMSE:

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

4.1. Just the average

The simpliest approach is to use the rating average extracted from our train set and later predict this rating for all the ratings in the test set.

```
mu <- mean(edx$rating)
mu
```

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

```
mu_rmse <- RMSE(validation$rating, mu)
```

This approach predict the same rating for al the movies and variation between movies are explained as random variations represented by $\epsilon_{u,i}$

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

We can see that the RMSE is quite high

```
rmse_results <- tibble(method = "Just Average",
                           RMSE = mu_rmse)
rmse_results %>% knitr::kable()
```

method	RMSE
Just Average	1.061202

4.2. Movie effect model

However not all the movies are rated with the same rates. As we already see we can talk about a movie bias b_i. This parameter and its predictions could be calculated using a liner model, but this operation could send a lot of time because of the size of the data. Nevertheless we can compute this b_i with the following code:

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

Using this parameter we will predict the ratings using the following formula:

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

Our predictions have improved but we already have to take into account other bias.

```
predicted_ratings <- mu + validation %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i
model_2_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie Effect Model",
                                 RMSE = model_2_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Just Average	1.0612018
Movie Effect Model	0.9439087

4.3. Movie effect plus user efect model

As ther are variability across movies, there are variability across users as well. There ara user who hates every movie that they see, and other who loves all the movies. This idea can be included in our model using the parameter b_u, which represents the user bias. This parameter can be calculated with the following code:

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

So we can do our predictions using this formula:

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

To use this formula to predict rating able us to improve our predictions:

```
predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
model_3_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie + User Effects Model",
                                 RMSE = model_3_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Just Average	1.0612018
Movie Effect Model	0.9439087
Movie + User Effects Model	0.8653488

4. Movie and user effects regularizated model

As we have already seen in the data exploration some users create a lot of rating and others just a few. In addition, some movies are rated a lot of times and other less times. This have produced some deviations in our parameters b_u and b_i. However this deviations can be corrected using regularization, a proces which able us to penalize movies and users with a small number of ratings. Regularization requires to choose a tunning parameter named lambda. But we can´t calculate the best lambda using the validation set (the validation set have to be used to test the lambda previously choosen), so are going to create a partition into edx named train set and pick the lambda that minimize the RMSE.
We will calculate lambda using "repeated random sub-sampling validation", also know as Monte Carlo crossvalidation, so we are going to replicate the proces five times. The final lambda will be the average of the 5 lambdas calculated.

```
edx_l <- edx %>% select(userId, movieId, rating)
lambdas <- seq(0, 10, 0.25) # test differnt lambdas
set.seed(1)
# NOTE: This proces could take a few minutes
best_lambda <- replicate (5, simplify = FALSE, {
```

```
  test_index_l <- createDataPartition(edx_l$rating, times = 1, p = .2, list = F)
  # Create the index
  train <- edx[-test_index_l, ] # Create Train set
  test <- edx[test_index_l, ] # Create Test set
  test <- test %>%
    semi_join(train, by = "movieId") %>%# The same movieId and usersId appears in both set
    semi_join(train, by = "userId")
  #Now, we pick the lambda
  rmses_l <- sapply(lambdas, function(l){
    mu <- mean(train$rating)
    b_i <- train %>%
      group_by(movieId) %>%
      summarize(b_i = sum(rating - mu)/(n()+l))
    b_u <- train %>%
      left_join(b_i, by="movieId") %>%
      group_by(userId) %>%
      summarize(b_u = sum(rating - b_i - mu)/(n()+l))
    predicted_ratings <- test %>%
      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$rating))
  })
  lambdas[which.min(rmses_l)]
})
```

We have to choose the lambda that will minimize the RMSE in the validation set, so we will use the mean of the lambdas calculated

```
true_best_lambda <- mean(as.numeric(best_lambda)) #the average of lambdas
true_best_lambda
```

```
## [1] 4.9
```

And finally, we can predict the ratings using the validation set and show the resoults of our final model:

```
rmses_l <- sapply(true_best_lambda, 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 <-
    validation %>%
    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, validation$rating))
})

#Predict and showing the resoults
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Regularized Movie + User Effect Model",
                                 RMSE = min(rmses_l)))
rmse_results %>% knitr::kable()
```

method	RMSE
Just Average	1.0612018
Movie Effect Model	0.9439087
Movie + User Effects Model	0.8653488
Regularized Movie + User Effect Model	0.8648185

5. CONCLUSIONS

Finally we have created a model that can predict movies rating with a RMSE under 0.865, so we can predict movies rating with less than one star of error. It is necessary to highlight that this have been obtained using as variables just the raing, the user and the movie. In the future, other analysis could be done using other varaibles such as the year of the movie or the genre. These analysys could improve the resoults and to get RMSEs even lowers.