MovieLens Recomendation system Project

Matias Garcia Mazzaro

octubre 22, 2020

Contents

The Project

This is a Machine Learning Project named MovieLens, it is a recomendation system of movies based in 10 millons dataset. The sourse of the dataset in this work is provided for Group Lens, for download, press here!!

This work is part of the final project for the Data Science Professional certificate of HarvardX. In October 2006, Netflix offered a challenge to the data science community: improve our recommendation algorithm by 10% and win a million dollars.

The Objetive

We will explore the data and construct a Machine Learning algorithm, describing the complete process, the method to compare diferents models will be trough RMSE (root mean squared error) between the true data and predicted data.

We need create a data set that we will work and a validation data set. But we need to be sure don't use the validation dataset for define the model. For this cuestion, we will create a data partition of the training data for test it and define the model.

We need obtain a RMSE < 0.86490 in the final validation for the total calification.

Chapter 1

Download and construct the dataset

The Netflix data is not publicly available, but the GroupLens research lab generated their own database with over 20 million ratings for over 27,000 movies by more than 138,000 users.

We start with the cration of **edx** and **validation** objects and loading the packages and libraries we will use.

```
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.
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-projec

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

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 3.6 or earlier:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieI
                                           title = as.character(title),
                                           genres = as.character(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)
```

Chapter 2

Analysis

It's important don't use the the **validation** data for train and test, it is because we need a eficient model for the data in the future, the final objetive is recomend movies to users, the true rating will ocurr after our recomendation. The fisrt step, is to take **edx** object and make a data partition **train_set** and **test_set**:

2.1 Data partition

```
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2,
                                  list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]
```

for be sure there are the same users and movies in the test set and training set;

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

2.1.1 Exploring the data before to start

For explore the dataset, we start for know how many diferents movies are in the dataset, and how many diferents users.

```
train_set %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   69878    10641
```

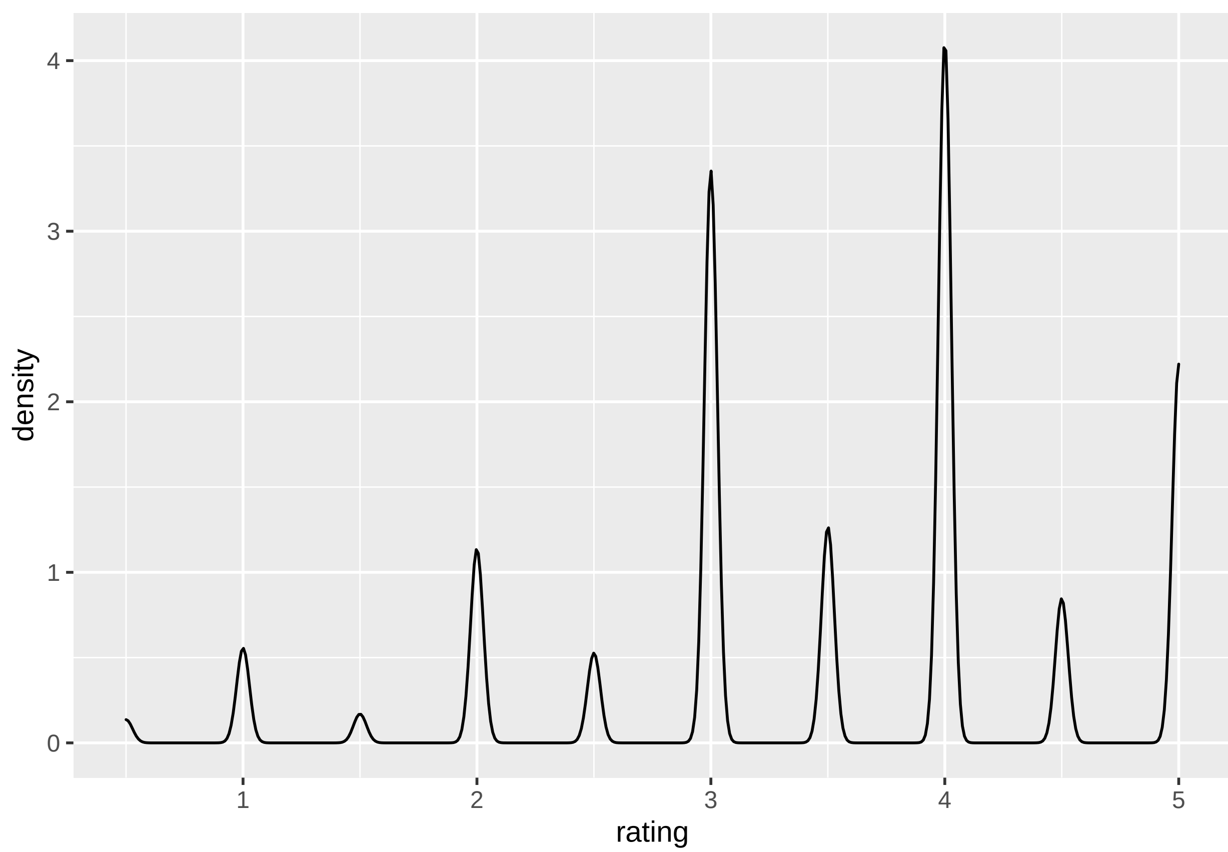
Also we can observe that the "genres" variable have differents asociated genres.

```r
head(train_set$genres)
```

```
## [1] "Action|Crime|Thriller"          "Action|Drama|Sci-Fi|Thriller"
## [3] "Action|Adventure|Sci-Fi"        "Action|Adventure|Drama|Sci-Fi"
## [5] "Children|Comedy|Fantasy"        "Comedy|Drama|Romance|War"
```

We can observe also the distribution density of ratings.

```r
edx %>% group_by(rating) %>% ggplot(aes(rating)) + geom_density()
```



2.2 Creation of the RMSE function

With the porpouse of messure the *root mean squared error* (RMSE), we define;

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

Where **y** is the rating for each movie $\boldsymbol{i}$ by user $\boldsymbol{u}$ and our prediction, and **N** being the number of user/movie combinations and the sum occurring over all these combinations.

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

The RMSE is similarly to a standard deviation: it is the typical error we make when predicting a movie rating. If this number is larger than 1, it means our typical error is larger than one star, which is not good.

2.3 The simplest model

The simplest possible recommendation system is the same rating for all movies regardless of user. A model that assumes the same rating for all movies and users with all the differences explained by random variation would look like this:

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

With ϵ independent errors sampled from the same distribution centered at 0 and μ the "true" rating for all movies.

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

```
## [1] 3.512574
```

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

```
## [1] 1.060704
```

We save the result in a new object for compare results later.

```
rmse_results <- tibble(method = "Just the average", RMSE = naive_rmse)
```

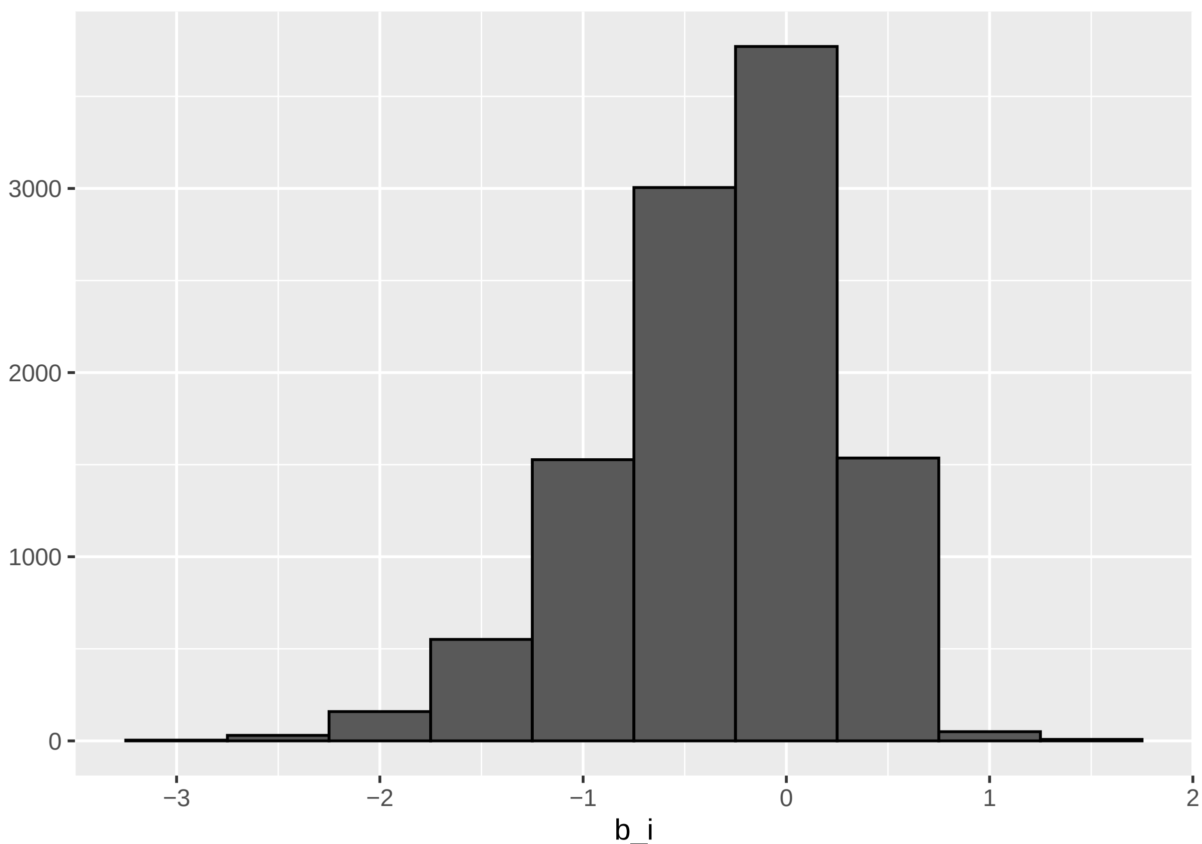
2.4 *Modeling movie effect*

For model the *movie effect*, which implies, a *bias* added to the before model who move the mean for the mean of movie, we write;

$$y_{u,i} = \mu + bi + \epsilon_{u,i}$$

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

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

The distribution of bias is;



We predict and test;

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

```r
movie_effect <- RMSE(predicted_ratings, test_set$rating)
movie_effect
```

```
## [1] 0.9437144
```

We can see, the model including movie effect is better.

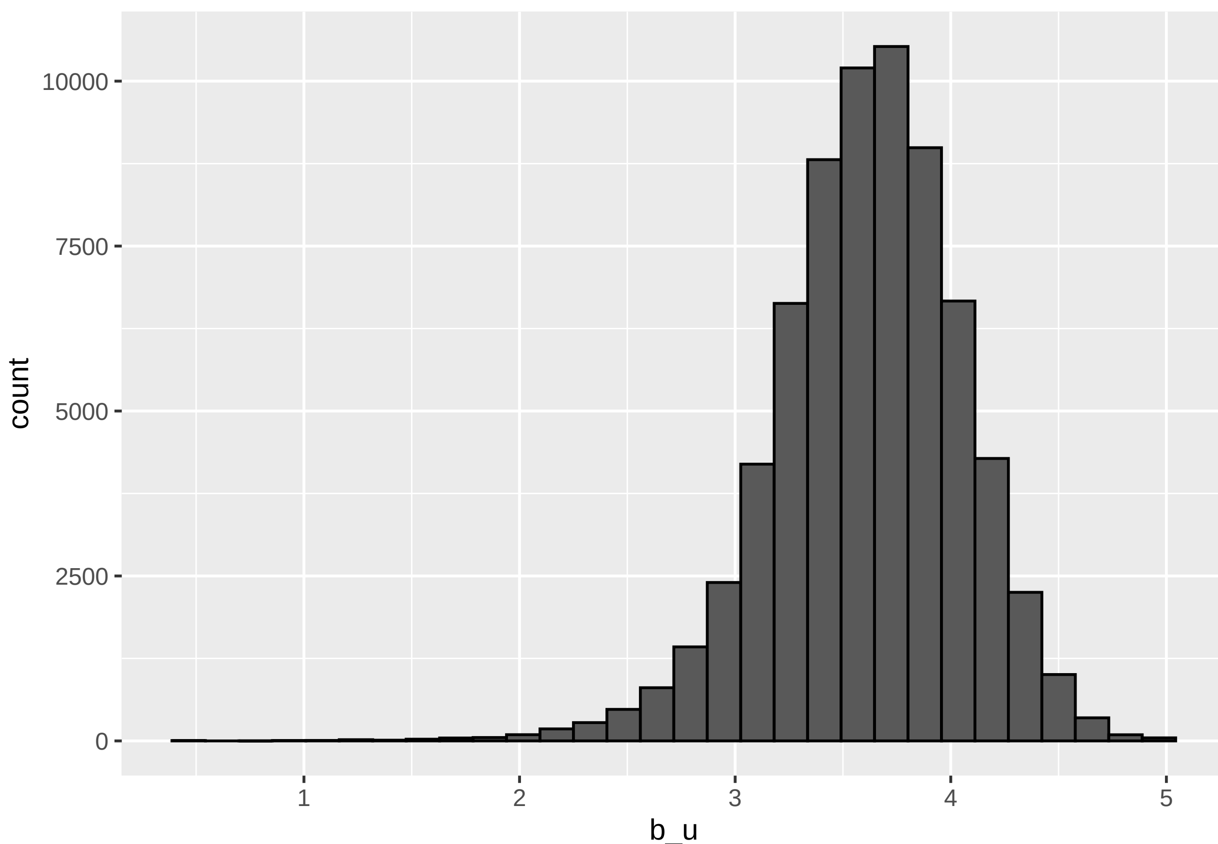
2.5 *Modeling user effect*

We add to the before model the user effect:

$$y_{u,i} = \mu + bi + bu + \epsilon_{u,i}$$

We observe the distribution for diferents users;

```
## `summarise()` ungrouping output (override with `.groups` argument)
```



We calculate de average for users;

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

And predict movie effect adding user effect.

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

user_effect <- RMSE(predicted_ratings, test_set$rating)
user_effect
```

```
## [1] 0.8661625
```

Sustancially diferent with others models.

2.6 Regularization models

If we explore the data for observe mistakes, we can see easily, there are movies with a fews ratings. Regularization permits us to penalize large estimates that are formed using small sample sizes.

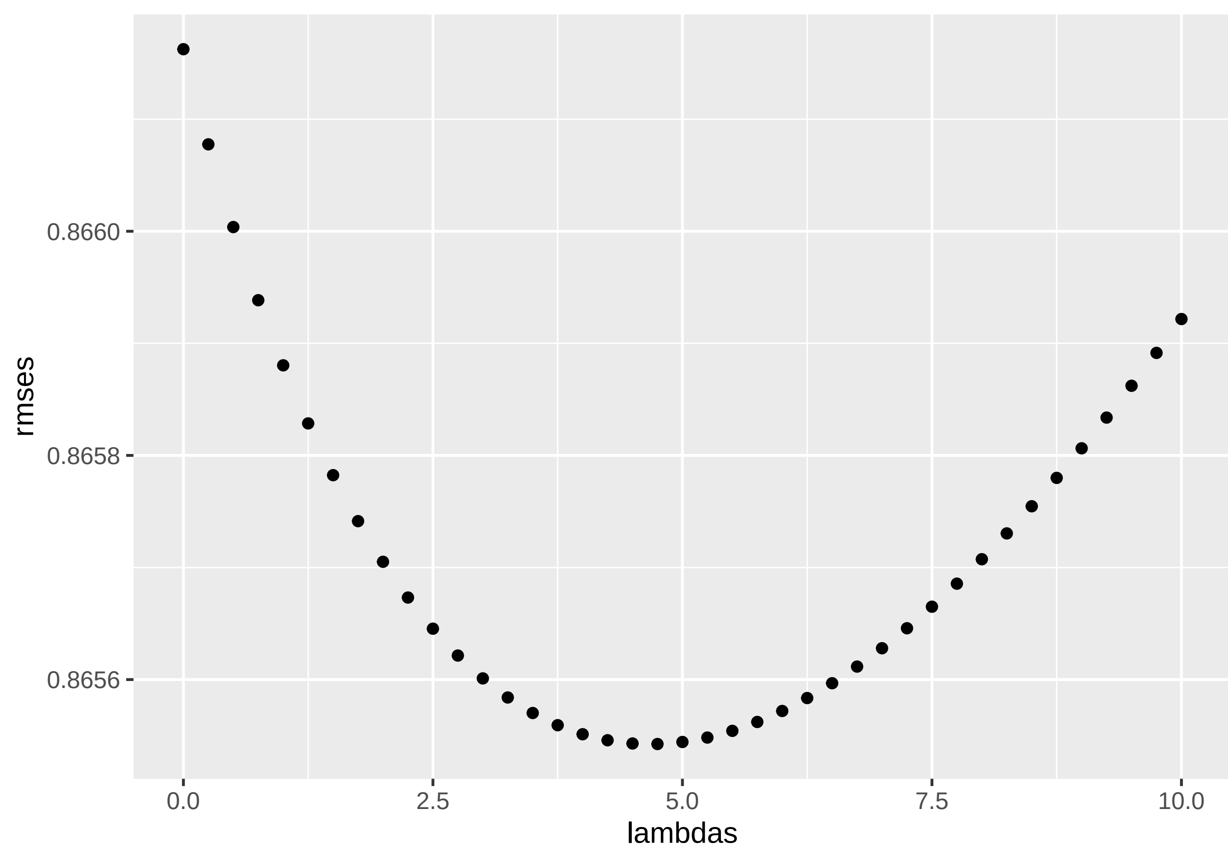
For that we test which Lambda is the optimal ones that aport the minimal RMSE;

```r
lambdas <- seq(0, 10, 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) %>%
    pull(pred)

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

We plot the Lambdas & the rmses for see how this affect, then we print the Lambda for the minimum RMSE.

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




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

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

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

The optimal Lambda is 4.75 .

```r
min(rmses)
```

```
## [1] 0.8655425
```

And the rmse for Lambda 4.75 is 0.86524.

So, now we have this differents RMSEs

```
## # A tibble: 4 x 2
##   method                                RMSE
##   <chr>                                 <chr>
## 1 Just the average                      1.06070
## 2 Movie effect Model                    0.94371
## 3 Movie + User effect Model             0.86616
## 4 Regularized Movie + User effect Model 0.86554
```

Chapter 3

Improve it.

If we want to do this better, we need to think about the data. We observe, the date where the movie was filmed and the date where was ranked for each users. Another information we can consider is the gendres of the movies. For all this we need to process original data.

3.1 Data transformation

3.1.1 Dates transformation

If we look inside the data, we can observe all the titles of the films have the year were was released.

```
head(train_set$title)
```

```
## [1] "Net, The (1995)"              "Outbreak (1995)"
## [3] "Stargate (1994)"              "Star Trek: Generations (1994)"
## [5] "Flintstones, The (1994)"      "Forrest Gump (1994)"
```

The first step is to add a column with the extraction of this year and count the oldness of the film, and then check it;

```
year_movie <-as.numeric(str_sub(train_set$title, start = -5, end = -2))
train_set <- train_set %>% mutate(year = year_movie)

year_movie <-as.numeric(str_sub(test_set$title, start = -5, end = -2))
test_set <- test_set %>% mutate(year = year_movie)

year_movie <-as.numeric(str_sub(validation$title, start = -5, end = -2))
validation <- validation %>% mutate(year = year_movie)
```

```r
year_movie <-as.numeric(str_sub(edx$title, start = -5, end = -2))
edx <- edx %>% mutate(year = year_movie)

validation <- validation %>% mutate(Age = 2020 - year)
train_set <- train_set %>% mutate(Age = 2020 - year)
test_set <- test_set %>% mutate(Age = 2020 - year)
edx <- edx %>% mutate(Age = 2020 - year)

head(train_set)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     185      5 838983525                Net, The (1995)
## 2:      1     292      5 838983421                Outbreak (1995)
## 3:      1     316      5 838983392                Stargate (1994)
## 4:      1     329      5 838983392 Star Trek: Generations (1994)
## 5:      1     355      5 838984474        Flintstones, The (1994)
## 6:      1     356      5 838983653            Forrest Gump (1994)
##                           genres year Age
## 1:         Action|Crime|Thriller 1995  25
## 2:  Action|Drama|Sci-Fi|Thriller 1995  25
## 3:       Action|Adventure|Sci-Fi 1994  26
## 4: Action|Adventure|Drama|Sci-Fi 1994  26
## 5:       Children|Comedy|Fantasy 1994  26
## 6:      Comedy|Drama|Romance|War 1994  26
```

Also we need to convert the timestamp in a year of rating.

```r
class(validation$timestamp)
```

```
## [1] "integer"
```

```r
validation <- validation %>% mutate(timestamp = as.Date.POSIXct(timestamp))
validation <- validation %>% mutate(year_rating = year(timestamp))

train_set <- train_set %>% mutate(timestamp = as.Date.POSIXct(timestamp))
train_set <- train_set %>% mutate(year_rating = year(timestamp))

test_set <- test_set %>% mutate(timestamp = as.Date.POSIXct(timestamp))
test_set <- test_set %>% mutate(year_rating = year(timestamp))

edx <- edx %>% mutate(timestamp = as.Date.POSIXct(timestamp))
edx <- edx %>% mutate(year_rating = year(timestamp))

head(validation)
```

```
##    userId movieId rating  timestamp
## 1:      1     231      5 1996-08-02
## 2:      1     480      5 1996-08-02
## 3:      1     586      5 1996-08-02
## 4:      2     151      3 1997-07-07
## 5:      2     858      2 1997-07-07
## 6:      2    1544      3 1997-07-07
##                                                    title
## 1:                                    Dumb & Dumber (1994)
## 2:                                    Jurassic Park (1993)
## 3:                                       Home Alone (1990)
## 4:                                          Rob Roy (1995)
## 5:                                   Godfather, The (1972)
## 6: Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
##                                   genres year Age year_rating
## 1:                                Comedy 1994  26        1996
## 2:        Action|Adventure|Sci-Fi|Thriller 1993  27        1996
## 3:                        Children|Comedy 1990  30        1996
## 4:                 Action|Drama|Romance|War 1995  25        1997
## 5:                            Crime|Drama 1972  48        1997
## 6: Action|Adventure|Horror|Sci-Fi|Thriller 1997  23        1997
```

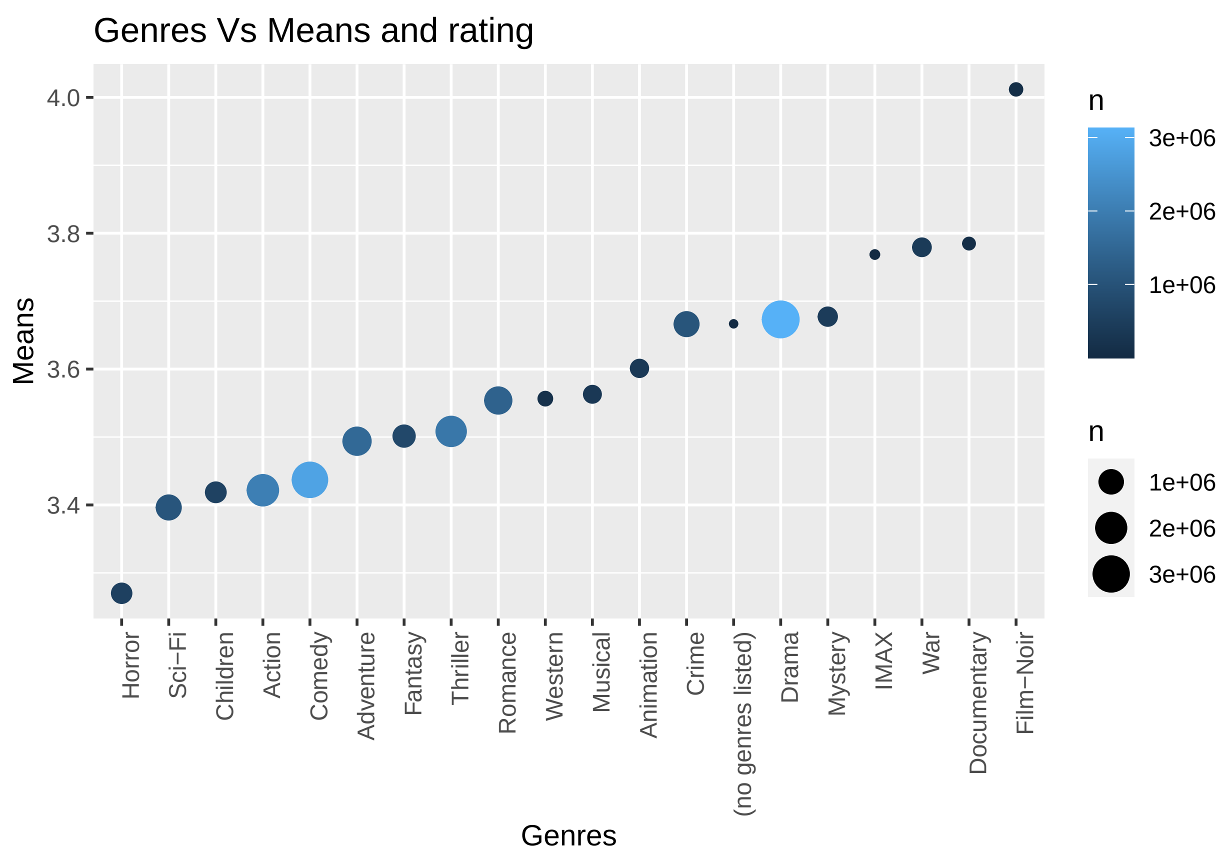
3.1.2 Genres transformation

We start we separate the genres and looks;

```r
genres <- train_set %>% separate_rows(genres, sep = "\\|")

table_genres <- genres %>% group_by(genres) %>% summarize(n = n(), mean = mean(rating))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
table_genres %>% mutate(genres = reorder(genres, mean)) %>% ggplot(aes(genres,mean,size
```



We can observe how diferents genres affect the mean ratings.

```
table_genres %>% mutate(dif = mean - mean(mean))
```

```
## # A tibble: 20 x 4
##    genres                 n  mean     dif
##    <chr>              <int> <dbl>   <dbl>
##  1 Drama            3127327  3.67  0.0857
##  2 Comedy           2832664  3.44 -0.150
##  3 Action           2048512  3.42 -0.166
##  4 Thriller         1861564  3.51 -0.0792
##  5 Adventure        1526571  3.49 -0.0936
##  6 Romance          1370133  3.55 -0.0337
##  7 Sci-Fi           1072759  3.40 -0.191
##  8 Crime            1062930  3.67  0.0788
##  9 Fantasy           740721  3.50 -0.0861
## 10 Children          590400  3.42 -0.169
## 11 Horror            553645  3.27 -0.318
## 12 Mystery           455346  3.68  0.0898
## 13 War               408941  3.78  0.192
## 14 Animation         373847  3.60  0.0136
```

```
## 15 Musical             346386  3.56 -0.0246
## 16 Western             151313  3.56 -0.0310
## 17 Film-Noir            95136  4.01  0.424
## 18 Documentary          74503  3.78  0.197
## 19 IMAX                  6549  3.77  0.181
## 20 (no genres listed)       6  3.67  0.0792
```

3.2 Adding the Oldness Movie effect

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

b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+4.75))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
b_u <- train_set %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+4.75))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
b_age <- train_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by ="userId") %>%
  group_by(Age) %>%
  summarize(b_age = mean(rating - b_i - b_u - mu)/(n()+4.75))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
predicted_ratings <-
  test_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_age, by = "Age") %>%
  mutate(pred = mu + b_i + b_u + b_age) %>%
  pull(pred)

regu_user_movie_age <- RMSE(predicted_ratings, test_set$rating)
```

and compare the results with the before models;

```
rmse_results <- tibble(method = c("Just the average", "Movie effect Model", "Movie + Use
rmse_results
```

```
## # A tibble: 5 x 2
##   method                                        RMSE
##   <chr>                                         <chr>
## 1 Just the average                              1.06070
## 2 Movie effect Model                            0.94371
## 3 Movie + User effect Model                     0.86616
## 4 Regularized Movie + User effect Model         0.86554
## 5 Regularized Movie + User + Age effect Model 0.86554
```

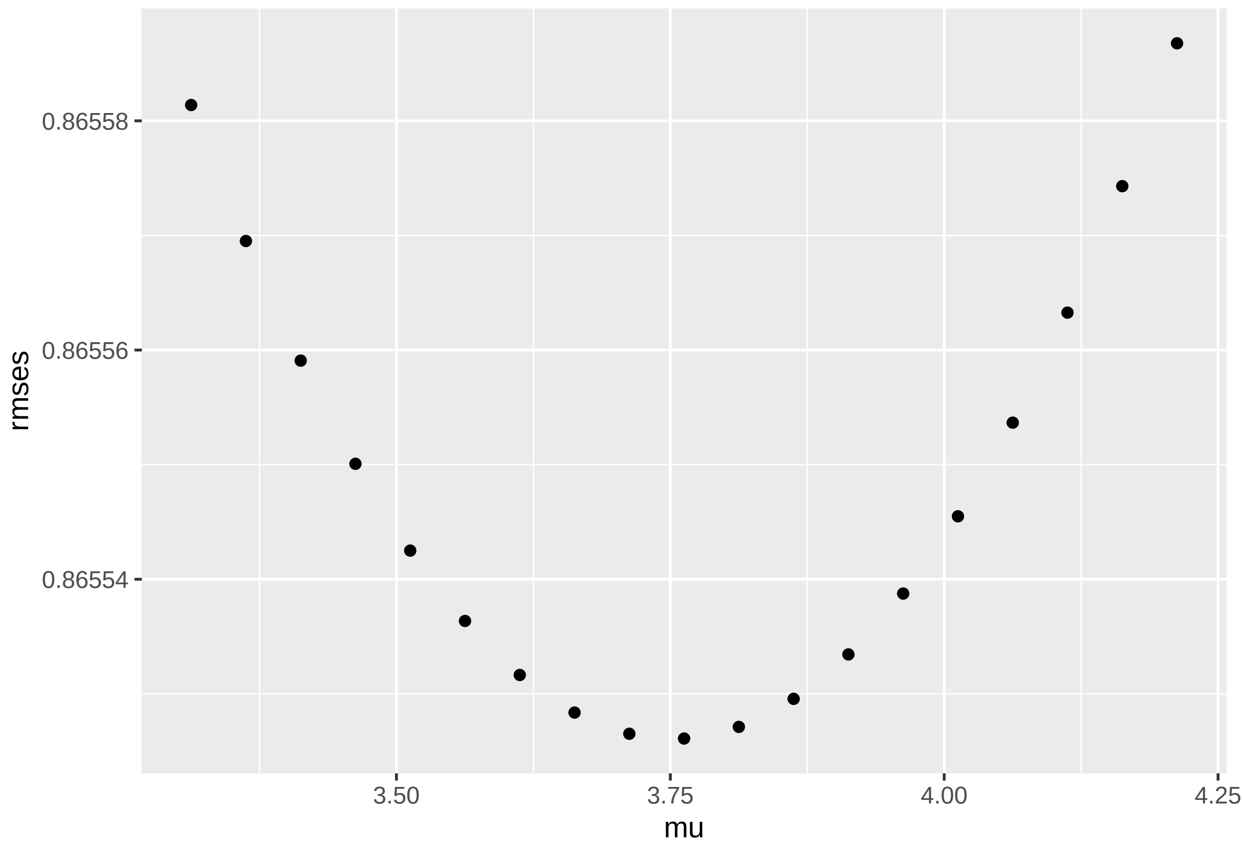
There is not significant results by oldness movie.

3.3 Moving the average in a function

Like we optimize the parameter "Lambda", we will move the mean near the average μ, is posible there a value who report a better result for the RMSE. We plot the μ vs. rmses.

```
mu <- mean(train_set$rating)+seq(-0.2, 0.7, 0.05)

rmses <- sapply(mu, function(mu){

  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+4.75))

  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+4.75))
  b_age <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by ="userId") %>%
    group_by(Age) %>%
    summarize(b_age = mean(rating - b_i - b_u - mu)/(n()+4.75))

  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
```

```
    left_join(b_u, by = "userId") %>%
    left_join(b_age, by = "Age") %>%
    mutate(pred = mu + b_i + b_u + b_age) %>%
    pull(pred)

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

qplot(mu, rmses)
```



We can see there are a optimal μ over the average.

```
mu <- mu[which.min(rmses)]
mu
```

```
## [1] 3.762574
```

```
regu_user_movie_age_mu <- min(rmses)
```

We add this change in our table.

```r
rmse_results <- tibble(method = c("Just the average", "Movie effect Model", "Movie + Use
rmse_results
```

```
## # A tibble: 6 x 2
##   method                                     RMSE
##   <chr>                                      <chr>
## 1 Just the average                           1.06070
## 2 Movie effect Model                         0.94371
## 3 Movie + User effect Model                  0.86616
## 4 Regularized Movie + User effect Model      0.86554
## 5 Regularized Movie + User + Age effect Model    0.86554
## 6 Reg. Movie + User + Age effect Model (best mu) 0.86553
```

The new RMSE is better, but in not suficient.

3.4 Add the genre to the model

We will try modeling also the genre, but, instead for each individual genres, for the combination of them. We can see there are near 800 diferent combinations.

```r
train_set %>% group_by(genres) %>% summarize(b_gen = mu-mean(rating))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 794 x 2
##    genres                                            b_gen
##    <chr>                                             <dbl>
##  1 (no genres listed)                               0.0959
##  2 Action                                           0.826
##  3 Action|Adventure                                 0.103
##  4 Action|Adventure|Animation|Children|Comedy      -0.202
##  5 Action|Adventure|Animation|Children|Comedy|Fantasy  0.763
##  6 Action|Adventure|Animation|Children|Comedy|IMAX     0.526
##  7 Action|Adventure|Animation|Children|Comedy|Sci-Fi   0.694
##  8 Action|Adventure|Animation|Children|Fantasy      1.06
##  9 Action|Adventure|Animation|Children|Sci-Fi       0.785
## 10 Action|Adventure|Animation|Comedy|Drama          0.248
## # ... with 784 more rows
```

We training;

```r
mu <- mean(train_set$rating)

b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+4.75))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
b_u <- train_set %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+4.75))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
b_age <- train_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by ="userId") %>%
  group_by(Age) %>%
  summarize(b_age = sum(rating - b_i - b_u - mu)/(n()+4.75))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
b_gen <- train_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by ="userId") %>%
  left_join(b_age, by = "Age") %>%
  group_by(genres) %>%
  summarize(b_gen = sum(rating - b_i - b_u - b_age - mu)/(n()+4.75))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
predicted_ratings <- test_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_age, by = "Age") %>%
  left_join(b_gen, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + b_age) %>%
  pull(pred)

regu_user_movie_age_gen <- RMSE(predicted_ratings, test_set$rating)
```

We explore the results.

```
rmse_results <- tibble(method = c("Just the average", "Movie effect Model", "Movie + Use
rmse_results
```

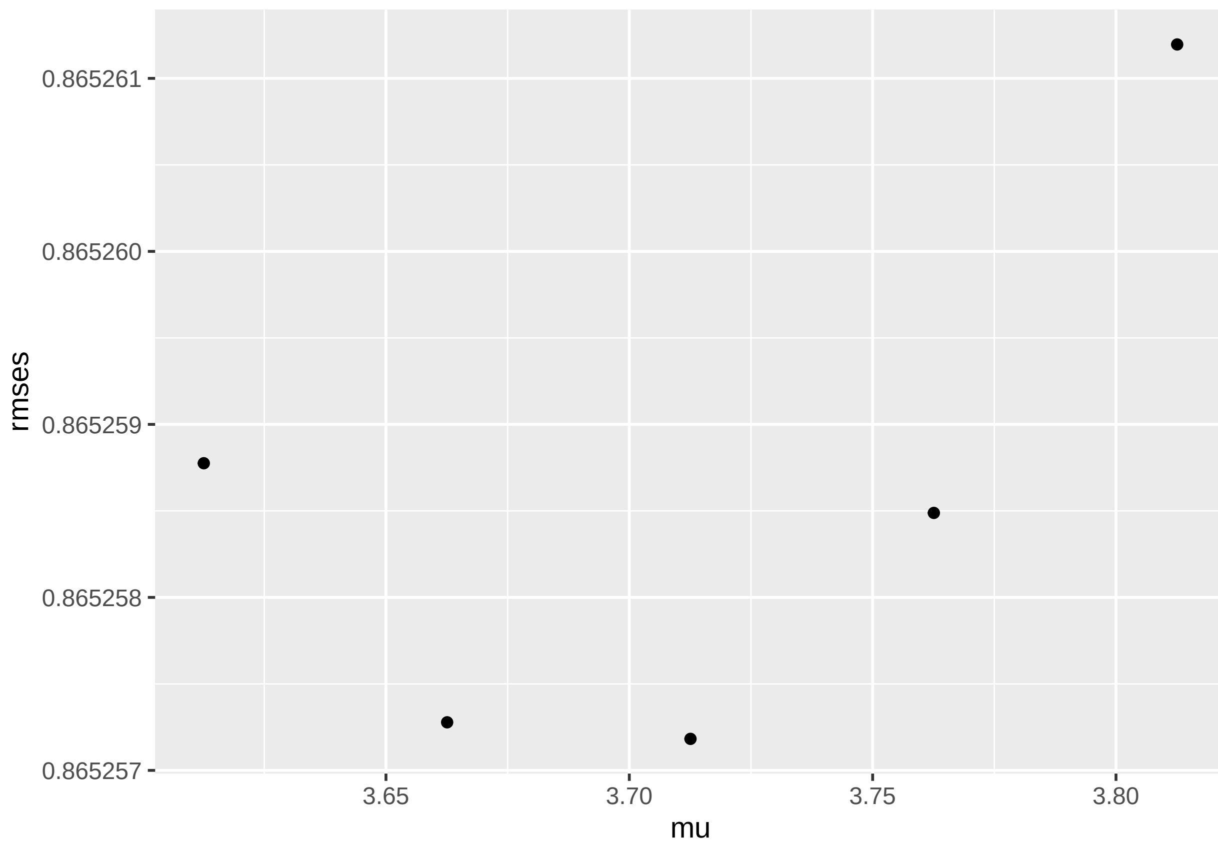
```
## # A tibble: 6 x 2
##   method                                       RMSE
##   <chr>                                        <chr>
## 1 Just the average                             1.06070
## 2 Movie effect Model                           0.94371
## 3 Movie + User effect Model                    0.86616
## 4 Regularized Movie + User effect Model        0.86554
## 5 Regularized Movie + User + Age effect Model  0.86554
## 6 Reg. Movie + User + Age + Genre effect Model 0.86527
```

Now we obtain a significant result.

We fit again the μ parameter for more presicion.

```
mu <- mean(train_set$rating)+seq(0.1, 0.3, 0.05)


rmses <- sapply(mu, function(mu){

  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+4.75))

  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+4.75))

  b_age <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by ="userId") %>%
    group_by(Age) %>%
    summarize(b_age = sum(rating - b_i - b_u - mu)/(n()+4.75))

  b_gen <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by ="userId") %>%
    left_join(b_age, by = "Age") %>%
    group_by(genres) %>%
```

```r
    summarize(b_gen = sum(rating - b_i - b_u - b_age - mu)/(n()+4.75))

  predicted_ratings <- test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_age, by = "Age") %>%
    left_join(b_gen, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_age) %>%
    pull(pred)

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

qplot(mu, rmses)
```




```r
mu[which.min(rmses)]
```

```
## [1] 3.712574
```

```r
regu_user_movie_age_gen_mu <- min(rmses)

rmse_results <- tibble(method = c("Just the average", "Movie effect Model", "Movie + Use
rmse_results
```

```
## # A tibble: 7 x 2
##   method                                            RMSE
##   <chr>                                             <chr>
## 1 Just the average                                  1.06070
## 2 Movie effect Model                                0.94371
## 3 Movie + User effect Model                         0.86616
## 4 Regularized Movie + User effect Model             0.86554
## 5 Regularized Movie + User + Age effect Model       0.86554
## 6 Reg. Movie + User + Age + Genre effect Model      0.86527
## 7 Reg. Movie + User + Age + Genre effect Model (mu opt) 0.86526
```

Chapter 4

Validation

For test our best model, the last one, we need to process now with the validation set. For train our model we will use the complete training set "edx". Like edx have more data, can be posible a improvement respect our last training.

We use our optimized parameters.

```
mu <- 3.712482

b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+4.75))

b_u <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+4.75))

b_age <- edx %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by ="userId") %>%
  group_by(Age) %>%
  summarize(b_age = sum(rating - b_i - b_u - mu)/(n()+4.75))

b_gen <- edx %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by ="userId") %>%
  left_join(b_age, by = "Age") %>%
  group_by(genres) %>%
  summarize(b_gen = sum(rating - b_i - b_u - b_age - mu)/(n()+4.75))

predicted_ratings <- validation %>%
```

```r
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_age, by = "Age") %>%
  left_join(b_gen, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + b_age) %>%
  pull(pred)

validation_rmse <- RMSE(predicted_ratings, validation$rating)
validation_rmse
```

```
## [1] 0.8645024
```

We obtain a RMSE of our prediction and true data, of 0.8645024.

```r
rmse_results <- tibble(method = c("Just the average", "Movie effect Model", "Movie + Use
rmse_results
```

```
## # A tibble: 8 x 2
##   method                                                RMSE
##   <chr>                                                 <chr>
## 1 Just the average                                      1.06070
## 2 Movie effect Model                                    0.94371
## 3 Movie + User effect Model                             0.86616
## 4 Regularized Movie + User effect Model                 0.86554
## 5 Regularized Movie + User + Age effect Model           0.86554
## 6 Reg. Movie + User + Age + Genre effect Model          0.86527
## 7 Reg. Movie + User + Age + Genre effect Model (mu opt) 0.86526
## 8 final model validation                                0.86450
```

Results

In the final validation, were obtain a RMSE of **0.86450** , it is better of the objetive (0.86490).

Conlusions

In the Netflix challenge the winning score was RMSE=0.8712. Data exploration is a process, we start with a very simple model and advance in more complex models trought we were knowing the data.

I obtain the objetive RMSE with a computational power of a desktop computer, and always is posible move the average in the base of the model for optimize, denote a coarse fit. Its posible improve it, ensemble models, and moving parameters, also using models like Random Forest, Matriz Factorization, SVD and PCA.