MovieLens HarvardX PH125.9x Capstone Project

Jordi Toneu

27/4/2020

1 Overview

1.1 Introduction

In 1997, the GroupLens Research, a research lab in the Department of Computer Science and Engineering at the university of Minnesota, created the **MovieLens** project. MovieLens is a recommender system with a virtual community that recommends movies for its users to watch, based on their film preferences using collaborative filtering of members movie ratings and movie reviews. The original dataset contains about 20 million ratings for about 27000 movies and more than 138000 users

1.2 The target

The target of this project is to create a machine learning algorithm to predict movie ratings considering different approaches
We will work with two small subsets of the original MovieLens data set with millions of ratings. The **edx** set will be used to develop our algorithm model and the **validation** set will be used for the final test of the developed algorithm, using the **RMSE** (Residual Mean Squared Error) criteria to estimate how close the predictions are from the true values.

The obtained RMSE with the validation set should be less than **0.86490**

1.3 Download the Movielens Dataset

This is the code to download the edx and validation dataset

```
################################
# 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, sample.kind="Rounding")
# if using R 3.5 or earlier, use `set.seed(1)` instead
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)
```

2. Data exploration, data preparation, analysis and methods

2.1 Data exploration

We have a look at the first rows of the edx dataset to get familiar with the contained information

```
#######################
# Dataset exploration #
#######################

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
```

Each row represents a rating given by one user to one movie

This is the structure of the edx data set

```
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
```

We can see that the dataset contains 9000055 observations of 6 variables

userID: a number that identifies each single user
movieID: a number that identifies each single movie
rating: a number that goes from 0.5 up tp 5 for movie rating
timestamp: a number that identifies review date and time
title: a string that contains the movie title and the movie release year
genres: a string that contains the different genres of a movie

This is the summary of the dataset

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

2.2 Data preparation

We first create the train and the test set to develope and evaluate the different model approaches

The test set will be 20% of the edx set

```
####################
# Data preparation #
####################

# Create the train and test set
# The test set is 20% of the edx set
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
temp <- edx[test_index,]

# Make sure userId and movieId in test set are also in train set
test_set <- temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

We will split and extract information contained in existing variables. We will also create new variables with calculated estimations

We start by converting the **timestamp** variable class and extracting the movie review year in the train, test and validation dataset

```
# Conversion of timestamp class and extraction of movie review year
train_set$timestamp <- as.POSIXct(train_set$timestamp, origin="1970-01-01") %>%
  year()

test_set$timestamp <- as.POSIXct(test_set$timestamp, origin="1970-01-01") %>%
  year()

validation$timestamp <- as.POSIXct(validation$timestamp, origin="1970-01-01") %>%
  year()
```

The movie release year is contained in the **title** variable. We first detect and later extract the movie release year of the string that contains this information and generate a new column called **releaseyear** in the train, test and validation set that will only contain the movie release year

```
# Detection and extraction of release year
train_set$releaseyear <- train_set$title %>%
  str_extract("\\(\\d{4}\\)") %>%
  str_extract("\\d{4}") %>%
  as.numeric()

test_set$releaseyear <- test_set$title %>%
  str_extract("\\(\\d{4}\\)") %>%
  str_extract("\\d{4}") %>%
  as.numeric()

validation$releaseyear <- validation$title %>%
  str_extract("\\(\\d{4}\\)") %>%
  str_extract("\\d{4}") %>%
  as.numeric()
```

Now we generate the variable **MovieAge** that contains the age of the movie when the review was shared

```
# Computation of movie age when sharing the review
train_set$MovieAge <- train_set$timestamp- train_set$releaseyear
test_set$MovieAge <- test_set$timestamp- test_set$releaseyear
validation$MovieAge <- validation$timestamp- validation$releaseyear
```

We can observe that some release years and review years do not match as the review takes place before the release year

```
# Detection of review year errors
head(train_set %>% filter (MovieAge < 0)%>% select(timestamp,title,releaseyear, MovieAge))
```

```
##   timestamp                   title releaseyear MovieAge
## 1      1996 Dangerous Ground (1997)        1997       -1
## 2      1996       Relic, The (1997)        1997       -1
## 3      1996 Dangerous Ground (1997)        1997       -1
## 4      1998   One Man's Hero (1999)        1999       -1
## 5      1999       Yards, The (2000)        2000       -1
## 6      1996       Nightwatch (1997)        1997       -1
```

```
# Showing the number of errors in the train set
train_set %>%
  filter (MovieAge < 0)%>%
  select(timestamp,title,releaseyear, MovieAge) %>%
  summarize ("Number of errors in the train set" = n())
```

```
##   Number of errors in the train set
## 1                               135
```

We set release-review time difference to zero when we detect an error and later check that the errors have been removed

```
# Set time difference to 0 when an error is detected
train_set$MovieAge <- ifelse (train_set$MovieAge <0,0, train_set$MovieAge)
test_set$MovieAge <- ifelse (test_set$MovieAge <0,0, test_set$MovieAge)
validation$MovieAge <- ifelse (validation$MovieAge <0,0, validation$MovieAge)

# Showing there are no more errors
train_set %>% filter (MovieAge < 0)%>%
  select(timestamp,title,releaseyear, MovieAge) %>%
  summarize ("Train set number of errors" = n())
```

```
##   Train set number of errors
## 1                          0
```

```
test_set %>% filter (MovieAge < 0)%>%
  select(timestamp,title,releaseyear, MovieAge) %>%
  summarize ("Test set number of errors" = n())
```

```
##   Test set number of errors
## 1                         0
```

```
validation %>%
  filter (MovieAge < 0)%>%
  select(timestamp,title,releaseyear, MovieAge) %>%
  summarize ("Validation set number of errors" = n())
```

```
##   Validation set number of errors
## 1                               0
```

Each movie can be classified under different type of genre. This information is contained in the **genres** variable. We will separate the type of genre of each movie of the train, test and validation dataset

The train set is a large set. Computing the train set with the **separate_rows** function consumes big memory resources and can generate a problem with memory limitation. To avoid this problem, we will divide the train set in 2 parts, A and B, separate the type of genre of each subset and later join A and B subsets together in order to have one single train set

This is how the train set looks like before genres separation. We can see 7200043 observations

```
# Genres separation
# Structure of the train_set
str (train_set)
```

```
## 'data.frame':    7200043 obs. of  8 variables:
##  $ userId     : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId    : num  185 292 316 329 355 356 362 364 370 377 ...
##  $ rating     : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp  : int  1996 1996 1996 1996 1996 1996 1996 1996 1996 1996 ...
##  $ title      : chr  "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" "Star Trek: Generations (
##  $ genres     : chr  "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Adventure|Sci-Fi
##  $ releaseyear: num  1995 1995 1994 1994 1994 ...
##  $ MovieAge   : num  1 1 2 2 2 2 2 2 2 2 ...
```

We divide the train set in two subsets to reduce memory usage when separating rows

```
# We divide the train set in two subsets to reduce memory usage when separating rows
train_set_A <-train_set [1:3500000,]

train_set_B <-train_set [3500001:7200043,]

train_set_A <- train_set_A %>% separate_rows(genres, sep = "\\|")

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

After splitting the genres, we join again the two subsets to have one only train set

```
# We join the subsets to have one only train set
train_set <- rbind(train_set_A, train_set_B)

rm (train_set_A, train_set_B)
```

This is how the train set looks like after splitting the genres

```
head (train_set)
```

```
##       userId movieId rating timestamp           title   genres releaseyear
## 2...1      1     185      5      1996 Net, The (1995)   Action        1995
## 2...2      1     185      5      1996 Net, The (1995)    Crime        1995
## 2...3      1     185      5      1996 Net, The (1995) Thriller        1995
## 4...4      1     292      5      1996 Outbreak (1995)   Action        1995
## 4...5      1     292      5      1996 Outbreak (1995)    Drama        1995
## 4...6      1     292      5      1996 Outbreak (1995)   Sci-Fi        1995
##       MovieAge
## 2...1        1
## 2...2        1
## 2...3        1
## 4...4        1
## 4...5        1
## 4...6        1
```

Now we separate the genres of the test and validation set

```
# Genres separation in the test set
test_set <- test_set %>% separate_rows(genres, sep = "\\|")

# Genres separation in the validation set
validation <- validation %>% separate_rows(genres, sep = "\\|")
```

This is the RMSE formula we will use to calculate how close our predictions are from the true values

```
# The RMSE formula
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings-predicted_ratings)^2, na.rm=TRUE))}
```

2.3 Data analysis and methods

2.3.1 Average movie rating model

The first approach is a naive model where we just consider the average movie rating for our predictions, a model that assumes the same rating for all movies and all users

We compute the average movie rating on the train set and compute the RMSE on the test set

```
##############################
# Data analysis and methods #
##############################

# Average movie rating method #

# Computation of average movie rating
mu <- mean(train_set$rating)

# Computation of RMSE
avg_rmse <- RMSE(mu, test_set$rating)
```

We generate a results table to see the RMSE improvements based on the addition of different models

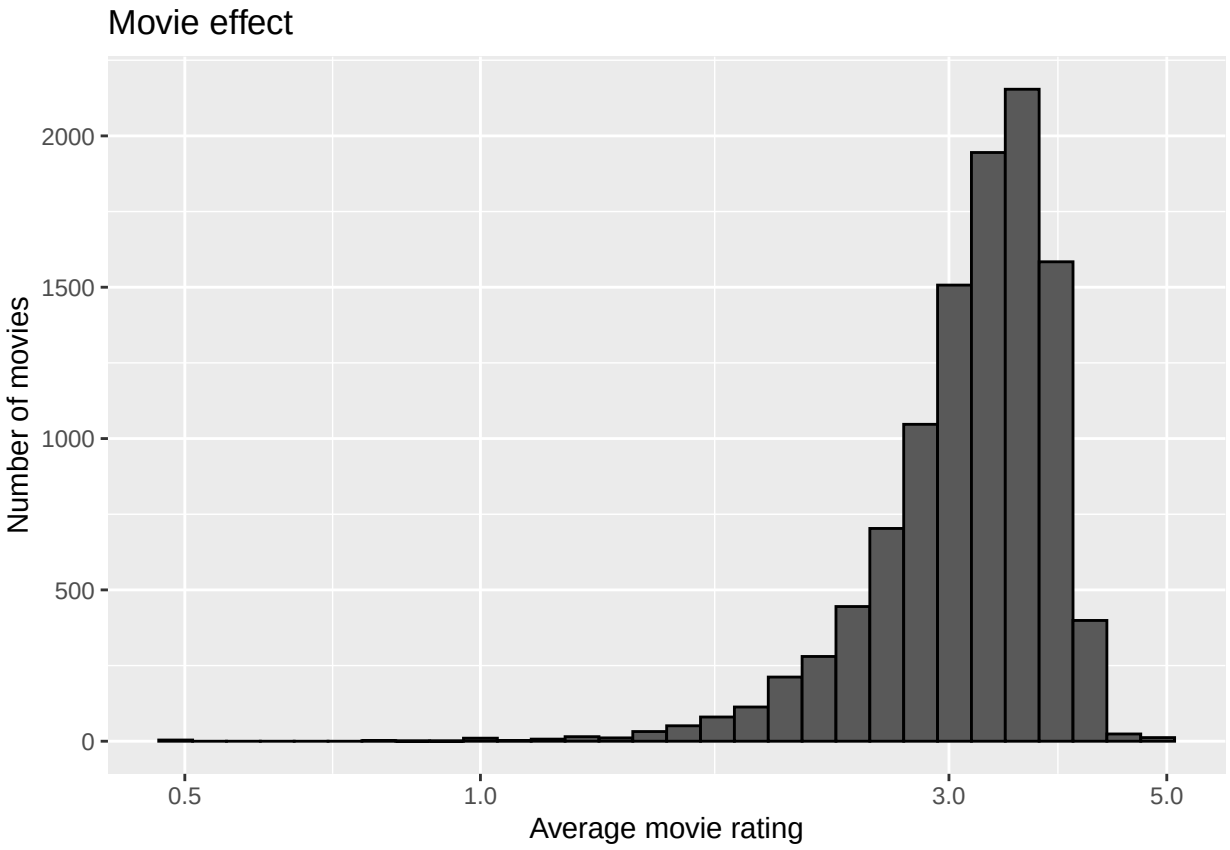
```
## Warning: `data_frame()` is deprecated as of tibble 1.1.0.
## Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

method	RMSE
Average movie rating model	1.052399

We obtain a RMSE of about 1.05, still quite far away from our target. That means that we have to improve our model

2.3.2 Movie rating effect model

In this graph we can observe than some movies are rated better than others



In the histogram we can see the rating distribution and observe that the most frequent rating is around 3.5, which is the overall average movie rating

In our next model will take into consideration this movie effect

```r
# Movie effect model #
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# Computation of prediction
predicted_ratings <- test_set %>%
  left_join(b_i, by = "movieId") %>%
  mutate(pred = mu + b_i) %>%
  .$pred

# Computation of prediction
predicted_ratings <- test_set %>%
  left_join(b_i, by = "movieId") %>%
  mutate(pred = mu + b_i) %>%
  .$pred

# Computation of RMSE
b_i_rmse <- RMSE(predicted_ratings, test_set$rating)

# Showing the table with RMSE obtained with movie effect
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Effect Model",
                                     RMSE = b_i_rmse ))

rmse_results %>% knitr::kable()
```

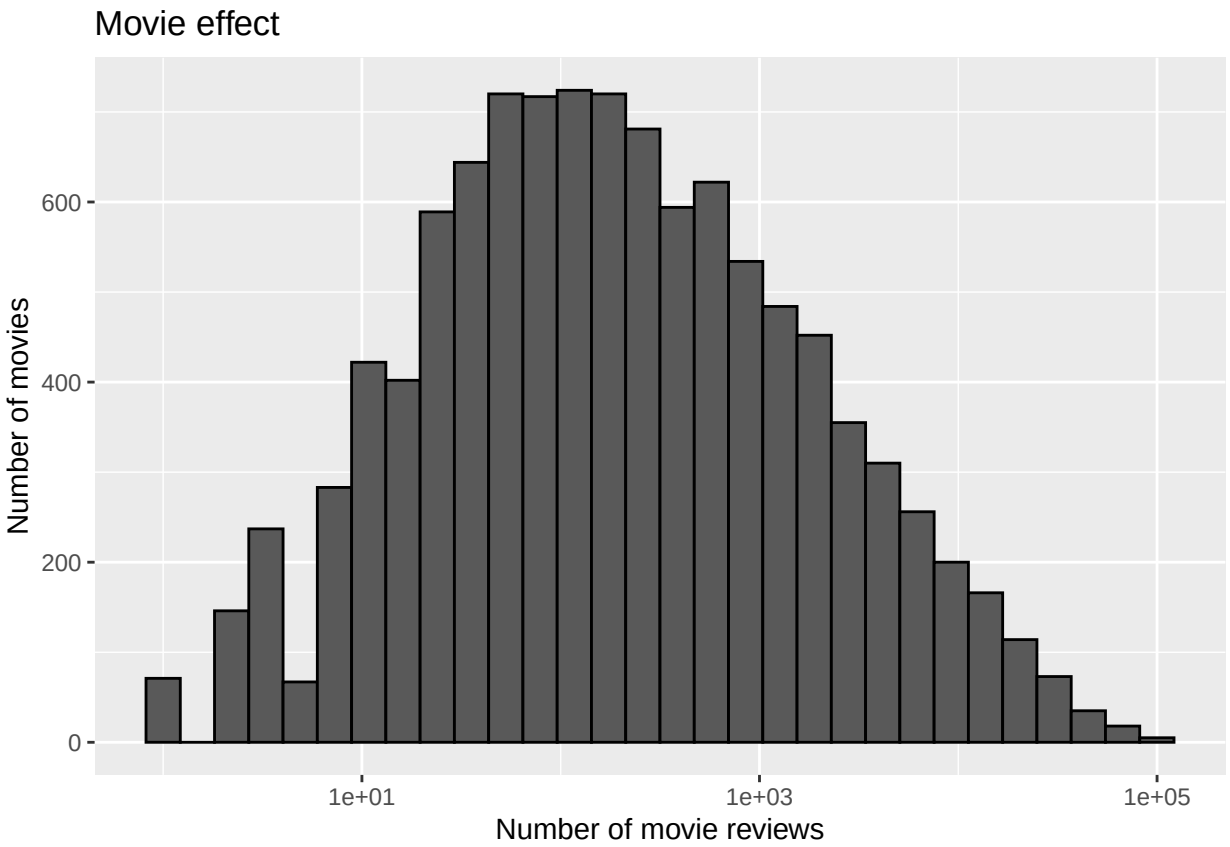
method	RMSE
Average movie rating model	1.0523988
Movie Effect Model	0.9415029

We can see that by considering this movie effect, the RMSE improves

2.3.3 Regularized movie rating effect model

As we can expect, not all movies have the same number of reviews

In this histogram we can see that some movies have more reviews than others



We will now introduce the regularized movie effect model, a model that gives a different weight to the movie rating depending on the number of reviews. So, rating of the movies with less reviews will be less significant than the movies with more reviews. This concept is called regularization and we use a constant variable called lambda, which we simplify with **l** that shrinks or expands the weight of the movie rating according as per the number of reviews

At the end of the research task, we will carry out a method to tune the best model parameters and lambda will be adjusted then. We only try to see if the RMSE is reduced with this model, not to look for the lowest RMSE yet

Just to start, we will set the lambda value to 1.5

```
# Regularized movie effect model
# Lambda value is set to 1.5 and the optimal value will be calculated at the end
l <- 1.5

# Regularized movie effect model
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+l))

# Computation of prediction
 predicted_ratings <- test_set %>%
  left_join(b_i, by = "movieId") %>%
  mutate(pred = mu + b_i) %>%
  .$pred
```

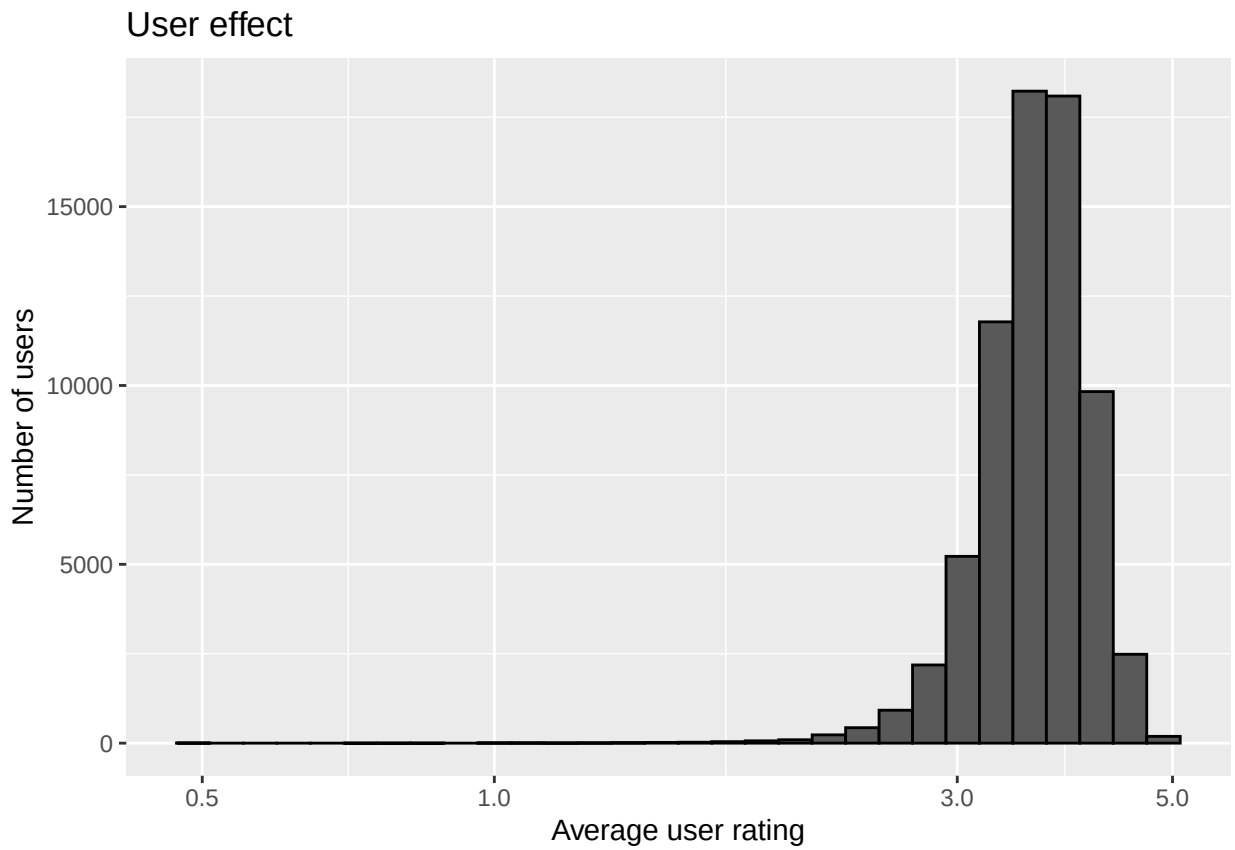
```r
# Computation of RMSE
b_i_rmse <- RMSE(predicted_ratings, test_set$rating)

# Showing the table with RMSE obtained with regulized movie effect model
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie Effect Model",
                                     RMSE = b_i_rmse ))

rmse_results %>% knitr::kable()
```

method	RMSE
Average movie rating model	1.0523988
Movie Effect Model	0.9415029
Regularized Movie Effect Model	0.9414840

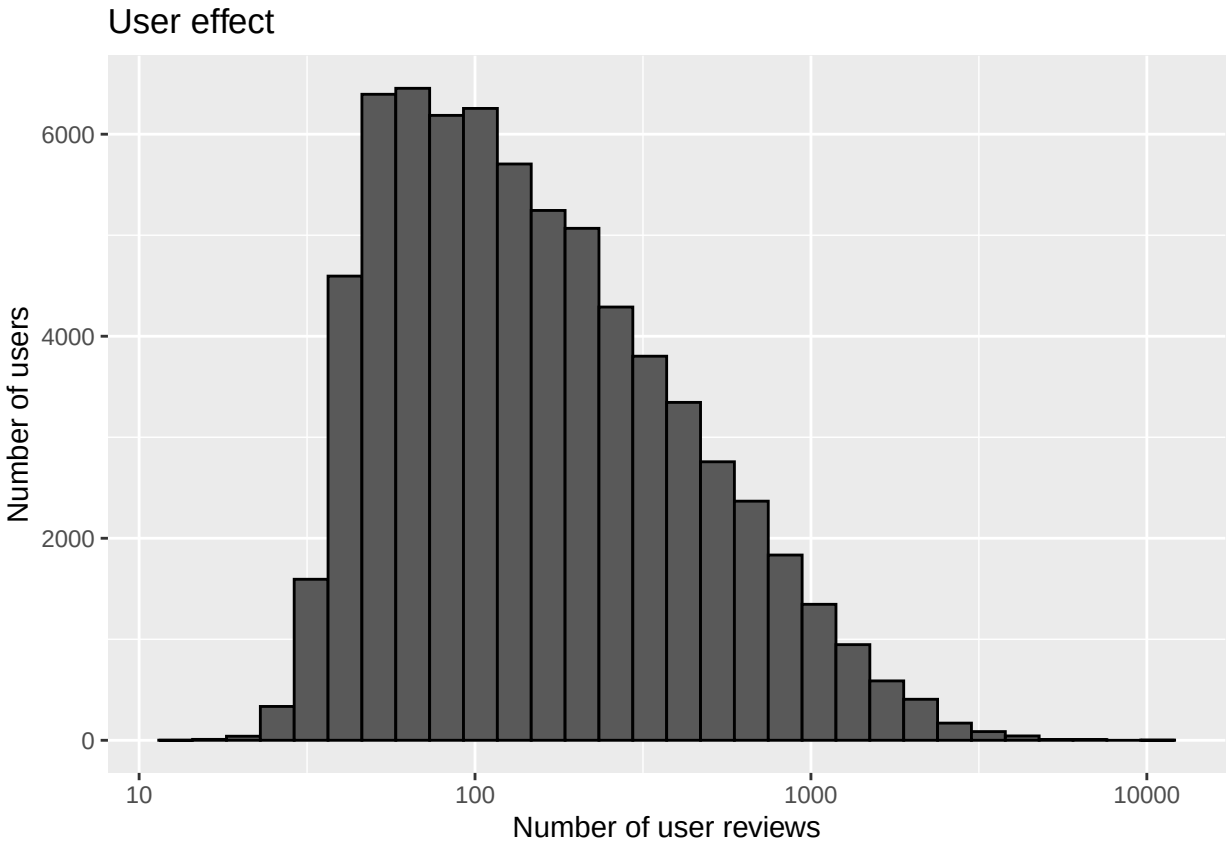
We can see that the movie regularization method has a positive effect on the RMSE

2.3.3 Regularized User rating effect model

Everyone has a different point of view, different opinion and trend. In the next histogram we can see that some users normally rate higher than others and vice versa



And we can also see that some users are more active than others sending reviews



The next model will consider the user effect and apply regularization as some users are more active than others

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

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

# Computation of RMSE
 b_u_rmse <- RMSE(predicted_ratings, test_set$rating)

# Showing the table with RMSE obtained with regulized user effect model
 rmse_results <- bind_rows(rmse_results,
                           data_frame(method="Regularized Movie + User Effect Model",
                                      RMSE = b_u_rmse ))

rmse_results %>% knitr::kable()
```

method	RMSE
Average movie rating model	1.0523988
Movie Effect Model	0.9415029
Regularized Movie Effect Model	0.9414840
Regularized Movie + User Effect Model	0.8644347

We are going into the right direction. The RMSE is getting lower, despite we have not tunned and optimized the model yet

2.3.3 Regularized movie age effect

The oldest movies in the dataset are dated in 1915

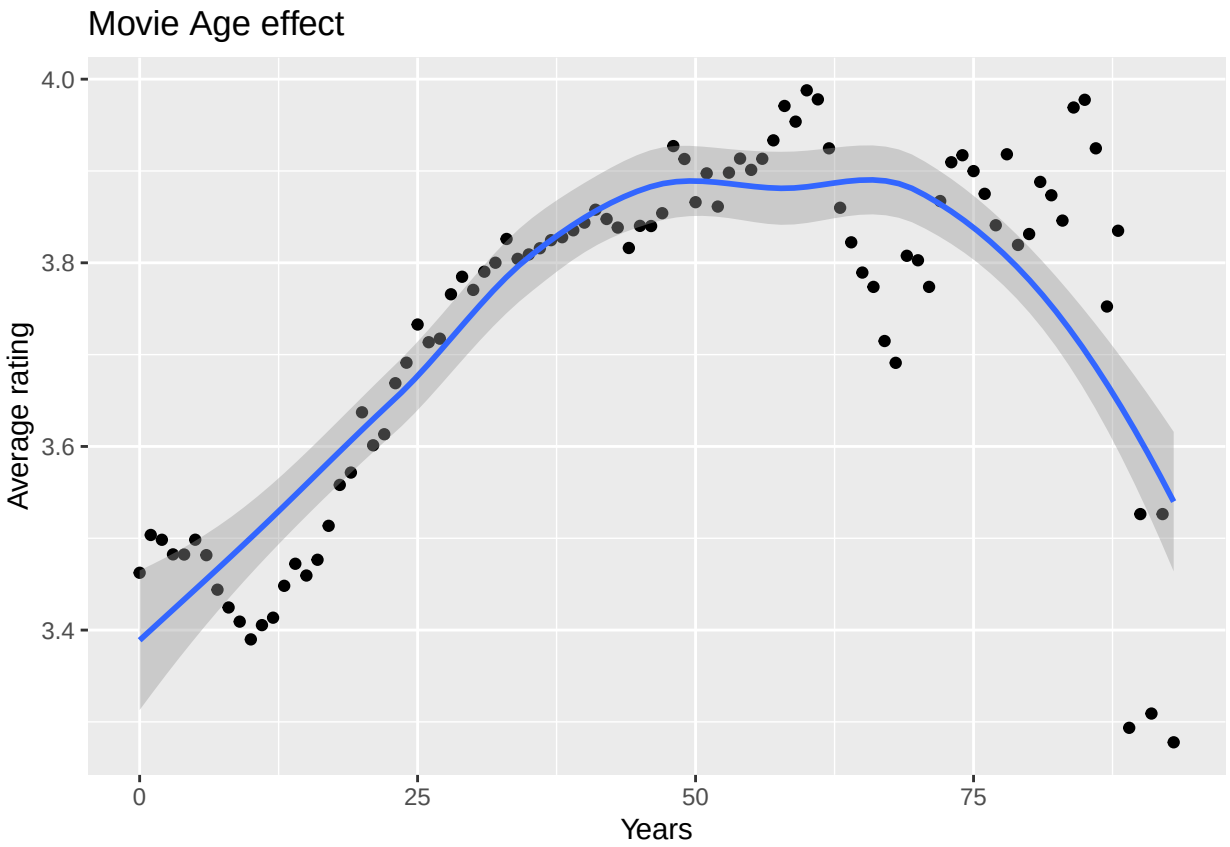
```
##                                 title
## 1        Birth of a Nation, The (1915)
## 2                  Intolerance (1916)
## 3  20,000 Leagues Under the Sea (1916)
## 4               Immigrant, The (1917)
## 5 Father Sergius (Otets Sergiy) (1917)
## 6                Dog's Life, A (1918)
```

The first movie review is dated in 1995

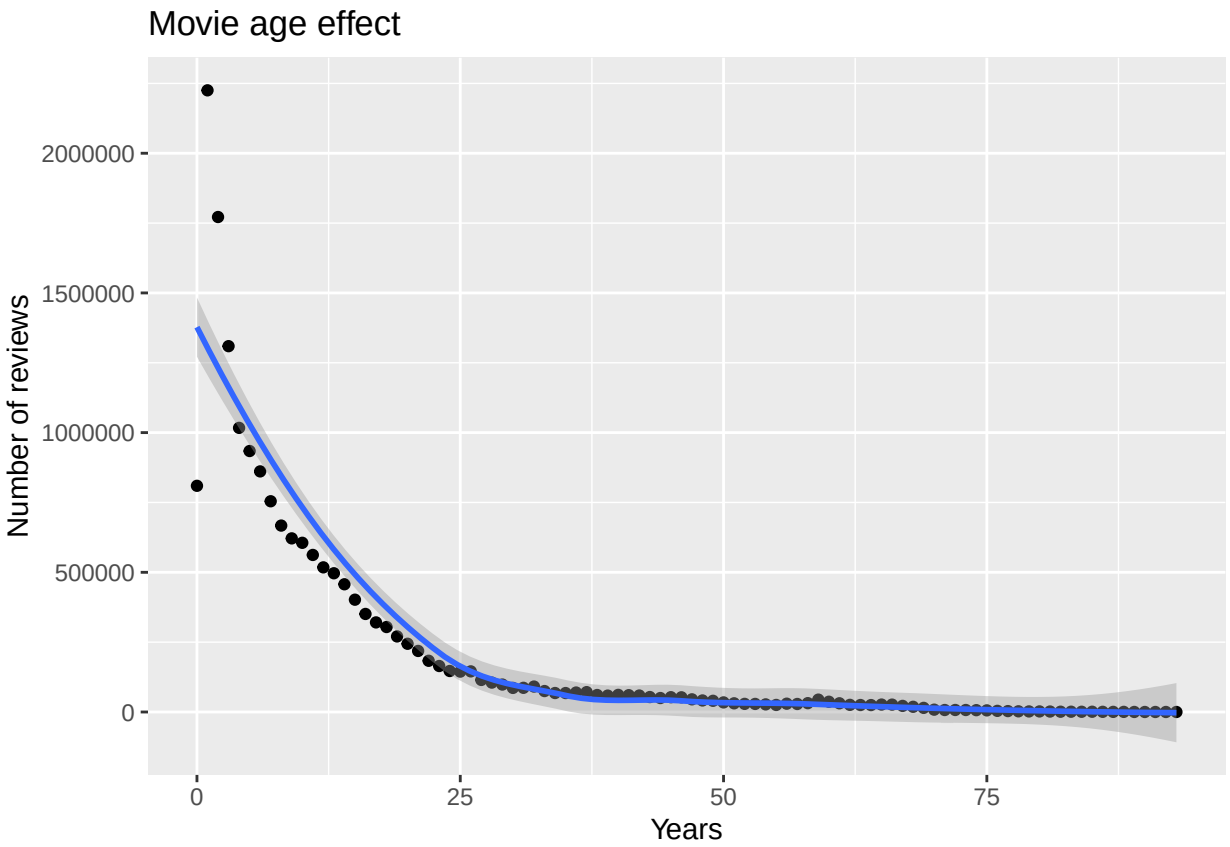
```
## # A tibble: 6 x 3
## # Groups:   movieId [6]
##   movieId title                          timestamp
##     <dbl> <chr>                              <int>
## 1    1079 Fish Called Wanda, A (1988)         1995
## 2     185 Net, The (1995)                     1996
## 3     292 Outbreak (1995)                     1996
## 4     316 Stargate (1994)                     1996
## 5     329 Star Trek: Generations (1994)       1996
## 6     355 Flintstones, The (1994)             1996
```

User rating also depends on the age of the movie at the time to write the review

In the next graph we can see the relation of the movie age and rating



In this graph we see the number of reviews and age of the movie



We can observe that best ratings are obtained when the movies are between 45 and 75 years old, but if we have a look at the distribution of reviews, we can see than very few rated movies are in the range of 45-75 years old. Most rated movies are less than 25 years old.

That means that this effect will also need to be rgularized

```
# Regularized Movie age effect model #
b_ma <- train_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by = "userId") %>%
  group_by(MovieAge)%>%
  summarize(b_ma = sum(rating - b_u - b_i - mu)/(n()+l))

# Computation of prediction
predicted_ratings <-
  test_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_ma, by = "MovieAge") %>%
    mutate(pred = mu + b_i + b_u + b_ma) %>%
  .$pred

# Computation of RMSE
b_ma_rmse <- RMSE(predicted_ratings, test_set$rating)

# Showing the table with RMSE obtained with regularized movie + user + genre + movie age effect model
rmse_results <- bind_rows(rmse_results,
```

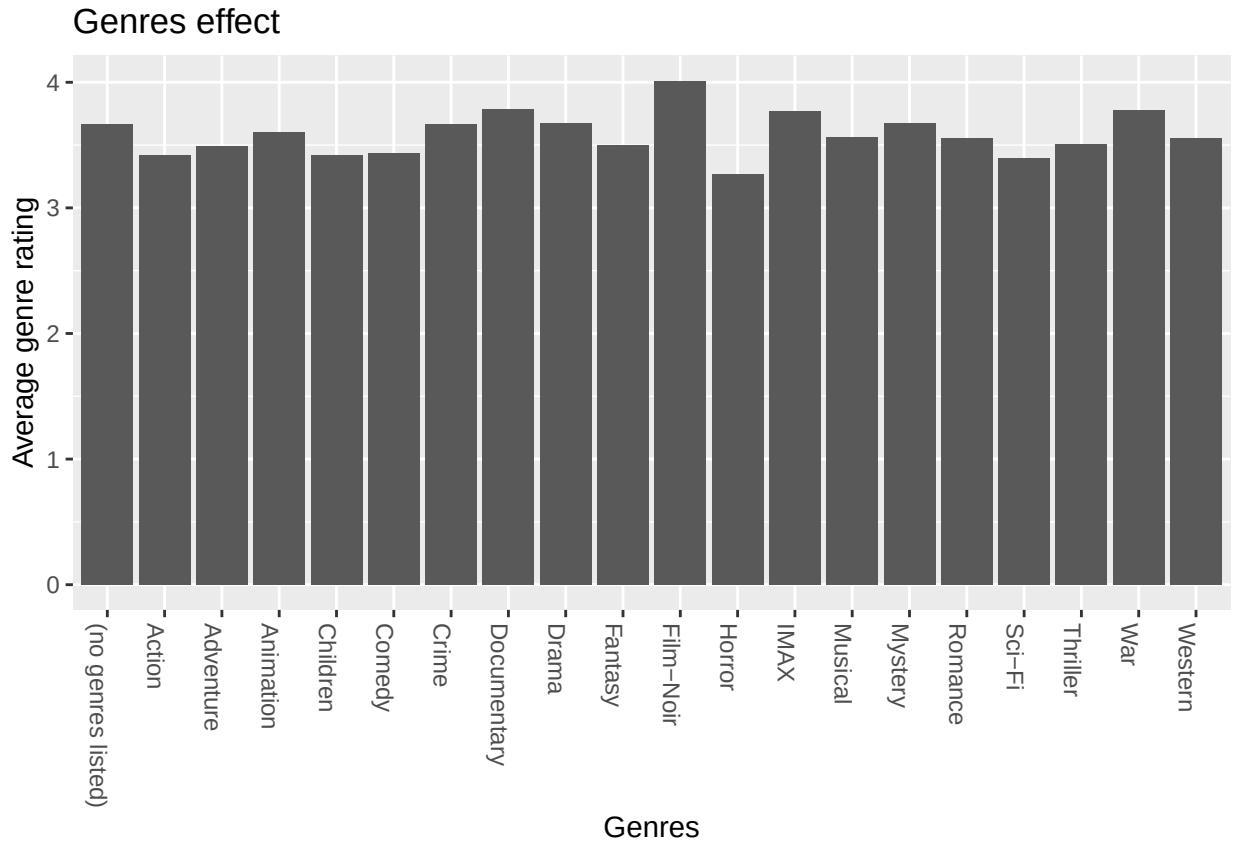
```
                          data_frame(method="Regularized Movie + User + Movie age Effect Model",
                                     RMSE = b_ma_rmse ))

rmse_results %>% knitr::kable()
```

method	RMSE
Average movie rating model	1.0523988
Movie Effect Model	0.9415029
Regularized Movie Effect Model	0.9414840
Regularized Movie + User Effect Model	0.8644347
Regularized Movie + User + Movie age Effect Model	0.8638712

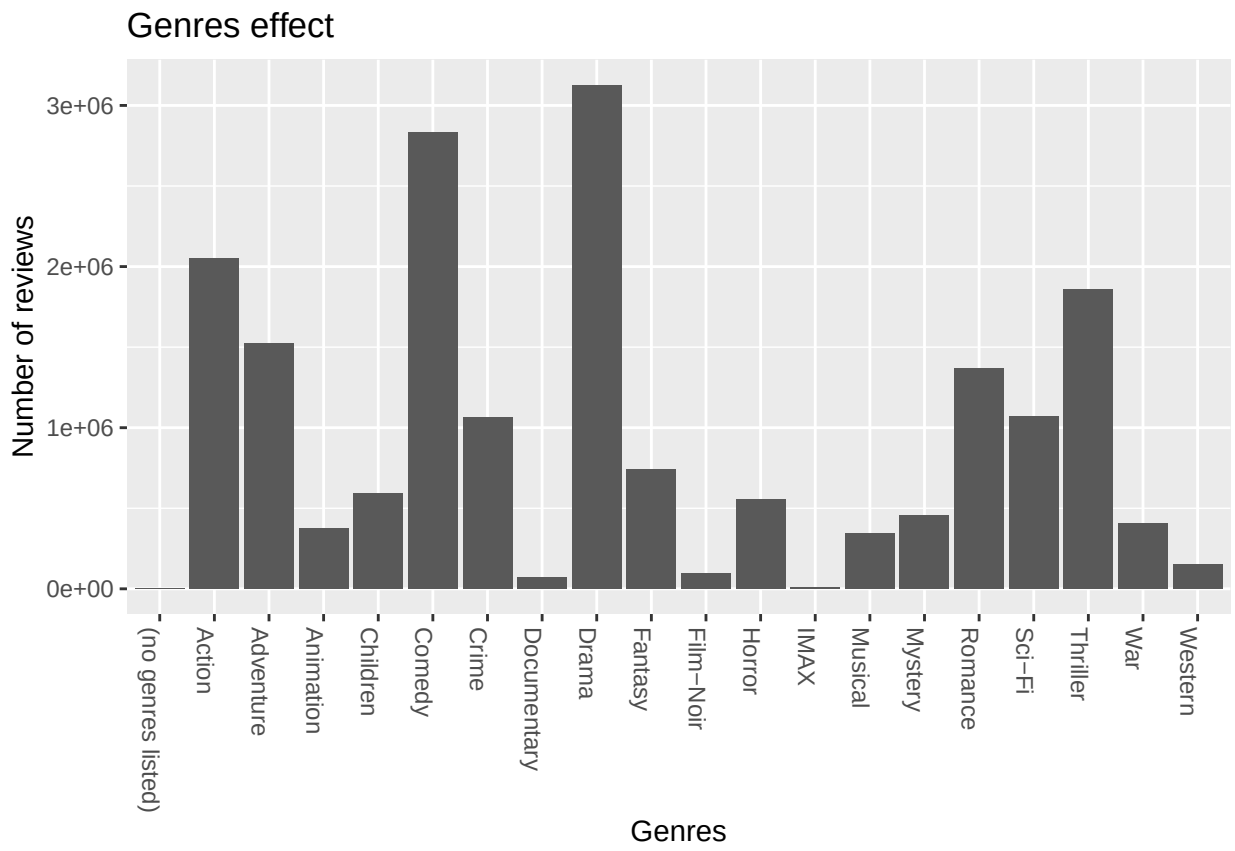
Again, the RMSE is getting lower

2.3.4 Regularized genre effect

We can imagine that some genres are higher rated than others. We can observe this in the following bar plot



There are some genres more popular than others and so, the number of reviews per genre should be different. We can see that in this graph



We can see that genre type with the highest number of reviews are Drama, Comedy, Action while the number of reviwes of "no genres listed", Imax, Documentary and Film-noir genres is very low

Again, we have to apply regularization on the genre effect

This is the model

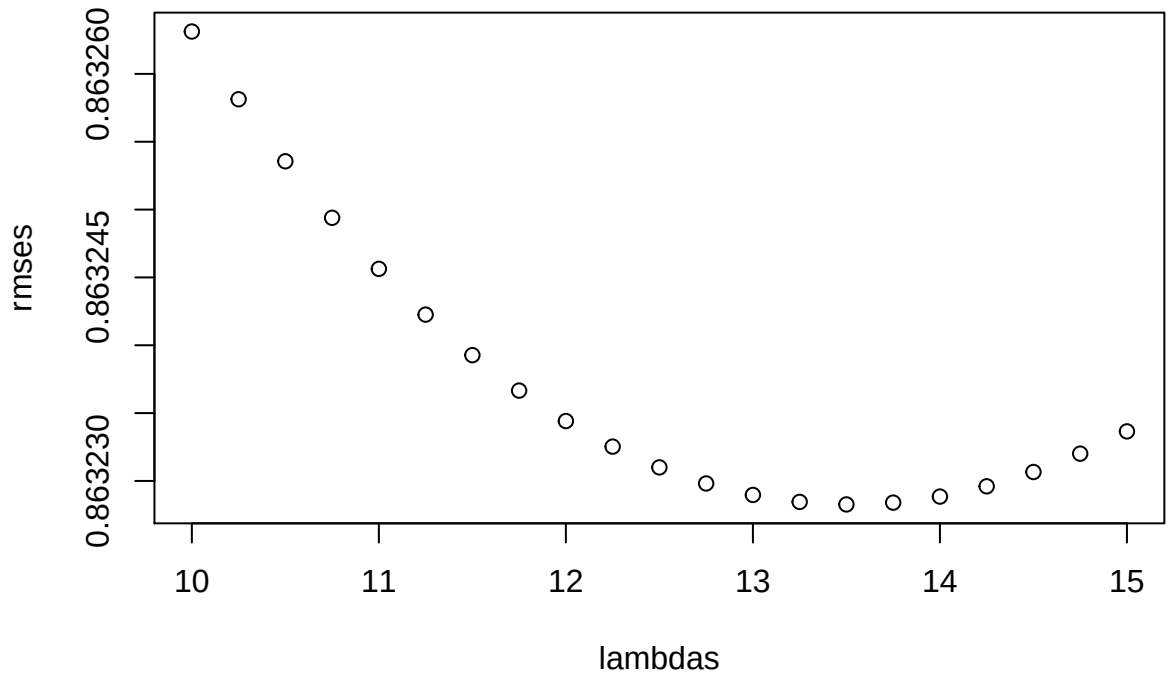
```
# Regularized genre effect model
b_g <- train_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_ma, by = "MovieAge") %>%
  group_by(genres)%>%
  summarize(b_g = sum(rating - b_u - b_i - b_ma - mu)/(n()+1))

# Computation of prediction
predicted_ratings <-
  test_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_ma, by = "MovieAge") %>%
  left_join(b_g, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + b_ma + b_g) %>%
  .$pred

# Computation of RMSE
b_g_rmse <- RMSE(predicted_ratings, test_set$rating)
```

```
# Showing the table with RMSE obtained with regularized movie + user + movie age + genre effect model
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie + User+ Movie age + Genre Effect Model",
                                     RMSE = b_g_rmse ))

rmse_results %>% knitr::kable()
```

method	RMSE
Average movie rating model	1.0523988
Movie Effect Model	0.9415029
Regularized Movie Effect Model	0.9414840
Regularized Movie + User Effect Model	0.8644347
Regularized Movie + User + Movie age Effect Model	0.8638712
Regularized Movie + User+ Movie age + Genre Effect Model	0.8637747

We can see the that RMSE is better than our target of 0.86490, but that happens with the test set. We have no idea yet about the result with the validation set

We will now tune our final model and use cross-validation to select the best lamda value




```
## [1] 13.5
```

The best lambda value is 13.5

We set "l" to 13.5 to tune our final model. We check the RMSE for the last time with the test set

```
# Final model #
# Setting lambda value in the final model
l <- 13.5

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

# Computation of prediction
  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_ma, by = "MovieAge") %>%
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_ma + b_g) %>%
    .$pred

# Computation of RMSE
tuned_rmse <- RMSE(predicted_ratings, test_set$rating)

# Showing the table with RMSE obtained with tuned and regularized movie + user + movie age + genre effe
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Final Model tuned",
                                     RMSE = tuned_rmse ))

rmse_results %>% knitr::kable()
```

method	RMSE
Average movie rating model	1.0523988
Movie Effect Model	0.9415029
Regularized Movie Effect Model	0.9414840
Regularized Movie + User Effect Model	0.8644347
Regularized Movie + User + Movie age Effect Model	0.8638712
Regularized Movie + User+ Movie age + Genre Effect Model	0.8637747

method	RMSE
Final Model tuned	0.8632283

After tunning our model, we can see the best possible RMSE obtained with the test set

3. Results

Now we have to check the RMSE with the validation set

```
############
# Results #
############

# Computation of prediction with VALIDATION set
predicted_ratings <-
  validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_ma, by = "MovieAge") %>%
  mutate(pred = mu + b_i + b_u + b_ma + b_g) %>%
  .$pred

# RMSE with VALIDATION set
validation_rmse <- RMSE(predicted_ratings, validation$rating)

rmse_results <- data_frame(method = "Final model with VALIDATION set", RMSE = validation_rmse)

rmse_results %>% knitr::kable()
```

method	RMSE
Final model with VALIDATION set	0.8629929

The RMSE obtained with the validation set is **0.8629929**

4. Conclusions

4.1 Results

We have obtained a RMSE with the validation set of **0.8629929**, less than our target of a RMSE of **0.86490**

We can see that the RMSE results improve every time we consider the effect of a new bias

Regularization is also an interesting technique to reduce the RMSE

The final model is a valid model because it predicts results on the validation set with a RMSE lower than the traget

As expected, the results with the validation set are not the same as with the test set but we obtained valid results with the two different sets

4.2 Limitations

The number of existing information in the dataset is limited. More accurate predictions could be obtained by increasing the numbers of predictors collecting more information like actor names, movie duration, user education level, user country of residence, user age, user sex and more information

4.3 Future work

The benefit of some other techniques like Singular Value Decomposition (SVD) and others can be studied in the future in order to check if the predictive algorithm model can be improved