# Movielens Project Submission

darbaroch

19/5/2021

## 1.1 Executive Summary

This document corresponds to the MovieLens Project Submission for the EDX's PH125.9x course.

The MovieLens data set comprise a large amount of movie ratings issued by users.

The project's task is to develop an algorithm that predicts a rating from 0 to 5 stars given a user, a movie and the time when the rating is performed.The typical error loss is considered and the residual mean squared error (RMSE) is used to evaluate the performance.

RMSE = 0.8649 is considered as a preliminary target as it is the threshold to achieve full marks on the error section.

Training and validation sets were provided within the course.

Data exploration was performed and then several approaches were discussed.

The ultimate algorithm developed considers that the rating to be predicted can be estimated as a sum of different effects to be added to the global mean of all ratings. These effects are: movie effect, user effect and genre effect for each user.

The RMSE achieved on the validation set was 0.847.

## 1.2 Introduction

Training (named "edx") and test (named "validation") sets were provided within the course and can be loaded from (this steps takes several minutes):

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

# 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")

# if using R 3.6 or earlier:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           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)
```

The way these data sets were generated with the code provided in the PH125.9x course assures that every user and every movie in the test set is also present in the training set.

Edx and validation data sets have 9000055 and 999999 observations respectively and 6 variables:

- userId: a unique number for each user.
- movieId: a unique number for each movie.
- title: a character with the complete title. It also includes the year.

- timestamp: the time when the user rated the movie.
- genres: a categorical variable comprising all genres that apply for the movie.
- rating: from 0 to 5 stars, including half stars

A sample of 10 observations is shown here.

```
edx[c(sample(edx$userId, 10)),] %>% as_tibble()
```

```
## # A tibble: 10 x 6
##    userId movieId rating  timestamp title                     genres
##     <int>   <dbl>  <dbl>      <int> <chr>                     <chr>
##  1    433     376    4    913046713 River Wild, The (1994)    Action|Thriller
##  2    348    1278    4   1133745212 Young Frankenstein (1974) Action|Comedy|H~
##  3    391     357    4    834916615 Four Weddings and a Funera~ Comedy|Romance
##  4    182    1914    5    943454181 Smoke Signals (1998)      Comedy|Drama
##  5    393     784    3    857945612 Cable Guy, The (1996)     Comedy|Thriller
##  6    128     232    4    838991959 Eat Drink Man Woman (Yin s~ Comedy|Drama|Ro~
##  7     70    1284    4   1213919429 Big Sleep, The (1946)     Crime|Film-Noir~
##  8    260     508    3    868278487 Philadelphia (1993)       Drama
##  9    546   58803    4.5 1215488723 21 (2008)                Drama
## 10    173     132    3    848491561 Jade (1995)               Thriller
```

Having described the data, the validation set is no longer used until the final RMSE calculation is performed.

# 2.1 Initial data exploration

The training set comprise 10677 different movies and 69878 different users as it can be verified with the following code:

```
edx %>% pull(movieId) %>% unique() %>% length()
```

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

```
edx %>% pull(userId) %>% unique() %>% length()
```

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

Only 4 out of the 6 variables are necessary to uniquely define each observation, as genres and movieId are variables that characterize the movie instead of the observation itself. Each observation can be then defined by: userId, title, rating and timestamp. Although it may be convenient to have all 6 variables in a single data frame, it should be noted that data could be organized in a tidy approach with a data frame comprising these 4 variables, and another data frame with 10677 observations (all the movies) and 3 variables: title, movieId and genres.

The training set's object size is about 900 Mb. Some procedures to be performed on the task can be quite demanding (at least in the computer were this document was developed) so it is considered better to switch to the 2 data frames approach. It saves as much as 150 Mb.

```
movies <- edx %>% select(movieId, title, genres) %>% unique()

edx <- edx %>% select(-title, -genres)
```

Each movie title should have a unique movieId and a unique value for the genres variable related to it. The following code checks if this statement holds or not:

```
movies %>%
  unique() %>%
  group_by(title) %>%
  mutate(id_amount = n()) %>%
  filter(id_amount > 1)
```

```
## # A tibble: 2 x 4
## # Groups:   title [1]
##   movieId title                     genres                          id_amount
##     <dbl> <chr>                     <chr>                               <int>
## 1   34048 War of the Worlds (2005) Action|Adventure|Sci-Fi|Thriller         2
## 2   64997 War of the Worlds (2005) Action                                   2
```

There is only one movie that do not fulfill that criterion. The “War of the Worlds (2005)” title has 2 different movieIds: 34048 and 64997. These 2 movieIds also differ on the genres variable.

The data shows that some users have rated both movieIds, some of them with different ratings.

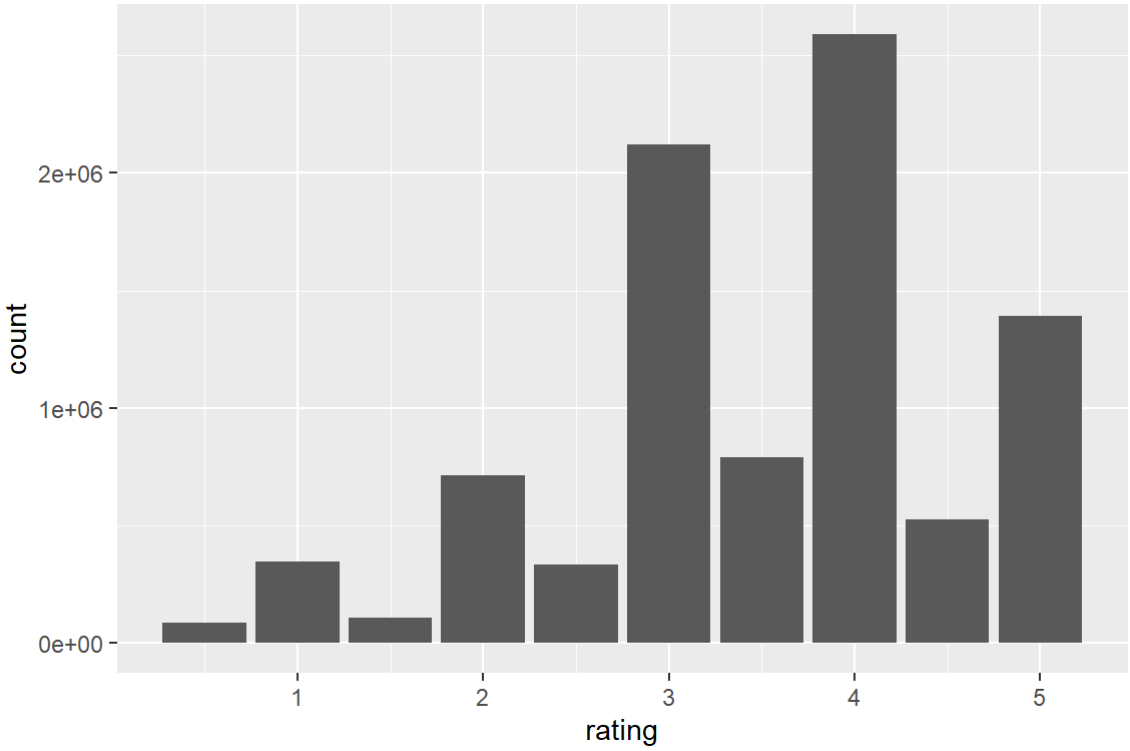
```
edx %>%
  filter(movieId %in% c(34048,64997)) %>%
  group_by(userId) %>%
  mutate(count = n()) %>%
  filter(count >1)
```

```
## # A tibble: 34 x 5
## # Groups:   userId [17]
##    userId movieId rating  timestamp count
##     <int>   <dbl>  <dbl>      <int> <int>
##  1   1018   34048    3   1215615934     2
##  2   1018   64997    3   1230668562     2
##  3  12366   34048    2.5 1218996372     2
##  4  12366   64997    3   1230578605     2
##  5  29892   34048    4.5 1121025521     2
##  6  29892   64997    4   1230604491     2
##  7  30840   34048    1.5 1138124404     2
##  8  30840   64997    1.5 1230945349     2
##  9  40570   34048    0.5 1229468271     2
## 10  40570   64997    1   1231055461     2
## # ... with 24 more rows
```

With the available data it is not possible to assure that it is the same movie. The title could be wrong. It might be the case that one of them is the original version from 1953 and not the 2005 version. It is unknown. Therefore the data will be kept just as it is without any correction. Should the movie be the same, some information is being lost when considering them as if they were 2 different movies.

The ratings go from 0 to 5 stars, and some rating values are much more frequent than others as it can be seen in this plot:

```
edx %>% ggplot(aes(rating)) + geom_bar()
```

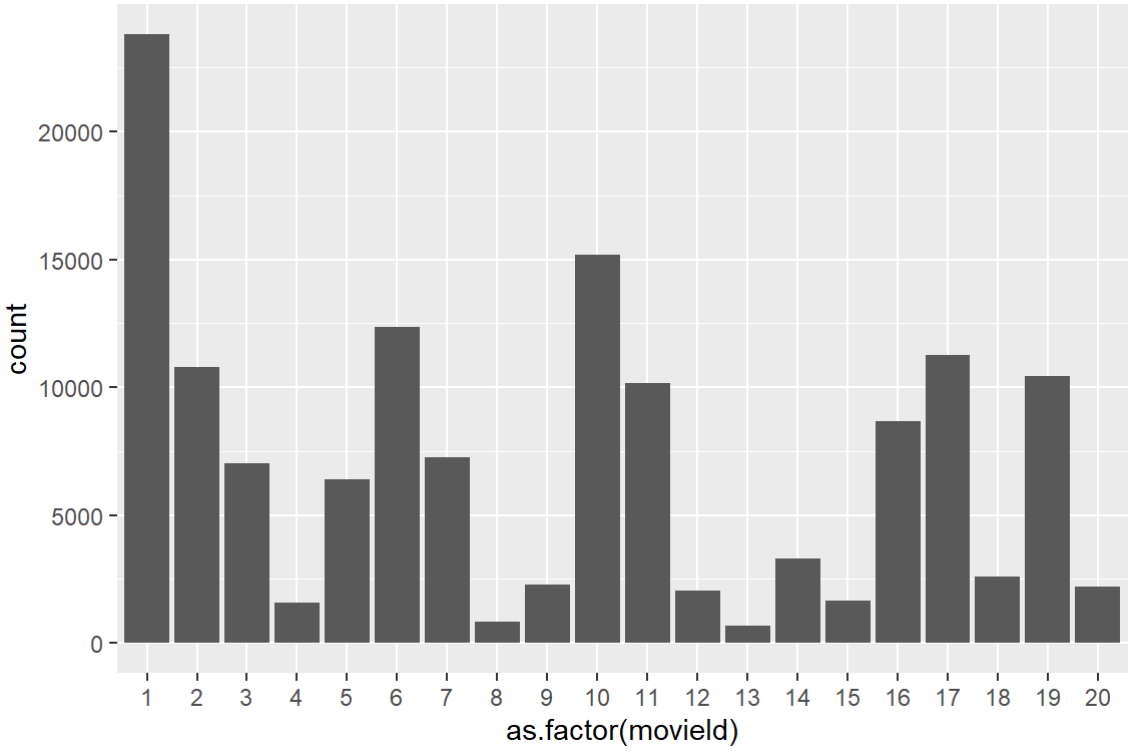



No movie has been rated with zero in the training set.

Rating with half stars is not quite frequent, users as a whole tend to rate using full stars.

Some movies are rated quite frequently while others are not. The following plot shows the count of rating each movie got for a random subset of 20 movies.

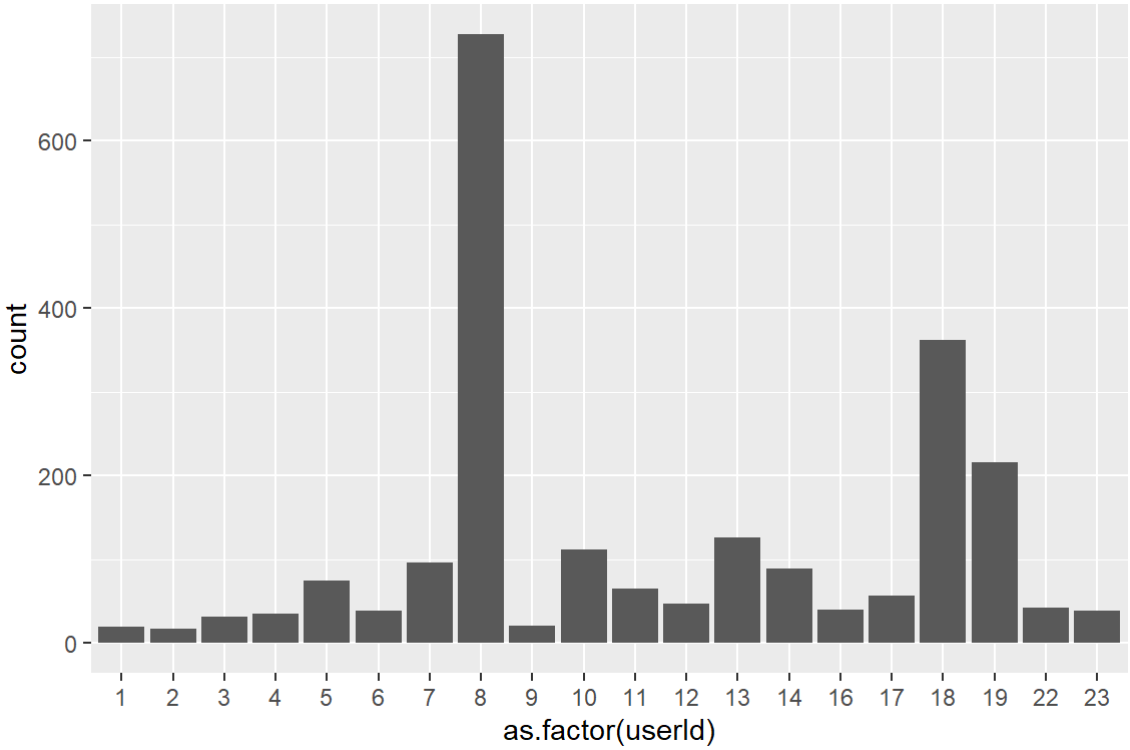
```
edx %>%
  group_by(movieId) %>%
  summarise(count = n()) %>%
  slice(1:20) %>%
  ggplot(aes(x=as.factor(movieId), y = count)) +
  geom_bar(stat = "identity")
```



Some movies received thousands of ratings while others just a few. The data set comprise several movies that were rated just by one user.

A similar behaviour can be seen for users. Some of them rated much more than others.

```
edx %>%
  group_by(userId) %>%
  summarise(count = n()) %>%
  slice(1:20) %>%
  ggplot(aes(x=as.factor(userId), y = count)) +
  geom_bar(stat = "identity")
```




This plot shows subset of 20 users and it can be seen that some of them rated hundreds of movies while others rated just a few.

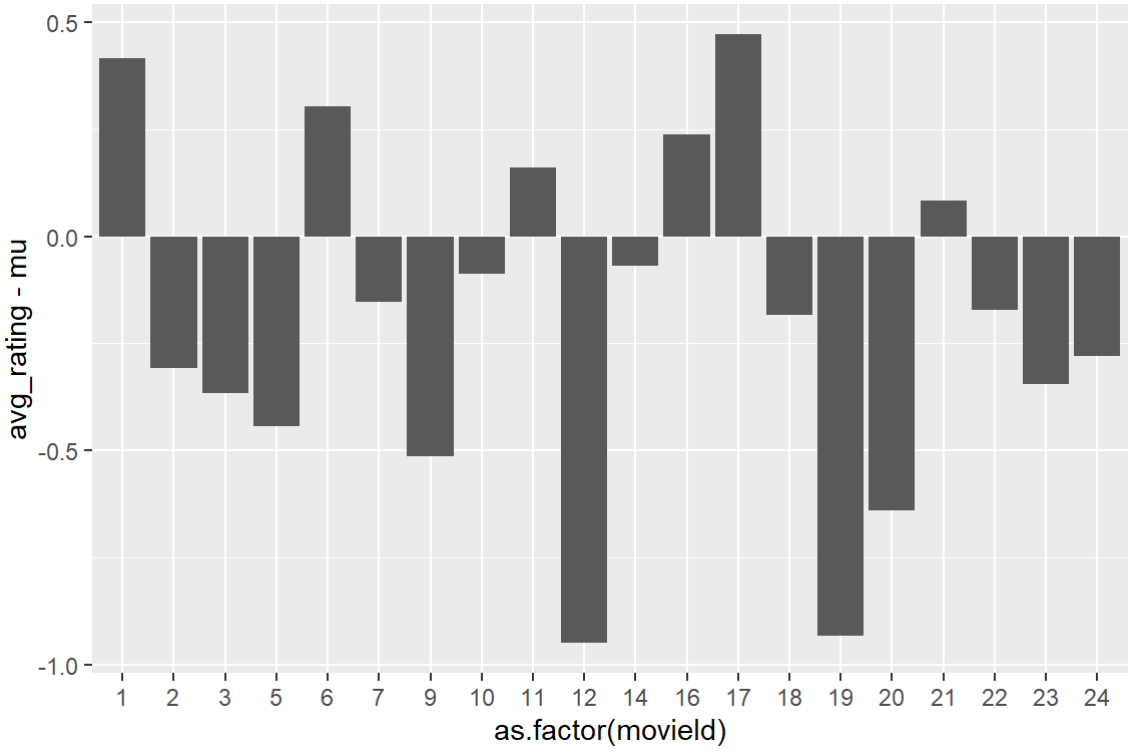
The rating mean for the whole training set is 3.512.

```
mu <- edx %>% pull(rating) %>% mean()
mu
```

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

This is a global mean but it can be seen that different movies have different average ratings. The following plot shows the difference between the movie's average rating and the global mean for a subset of 20 movies. For the sake of argument a filter that keeps movies rated by at least 2000 users is applied to avoid considering averages determined by just a couple of users.

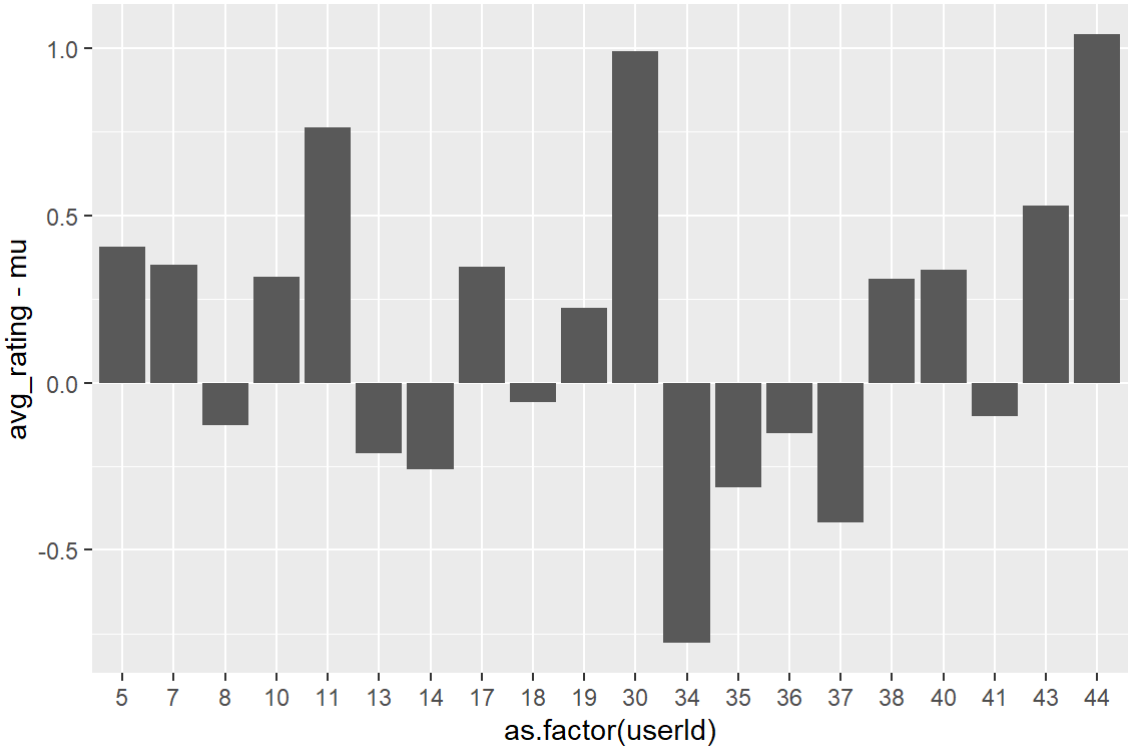
```
edx %>%
  group_by(movieId) %>%
  summarise(avg_rating = mean(rating),
            count = n()) %>%
  filter(count > 2000) %>%
  slice(1:20) %>%
  ggplot(aes(x=as.factor(movieId), y = avg_rating - mu)) +
  geom_bar(stat = "identity")
```




There seems to be significant variability among movies' averages, say a "movie effect" or "movie bias".

A similar behaviour is found for users, some tend to rate higher than others. The following plot shows the difference between average rating and the global mean for 20 users (filtered for users that rated at least 50 movies).

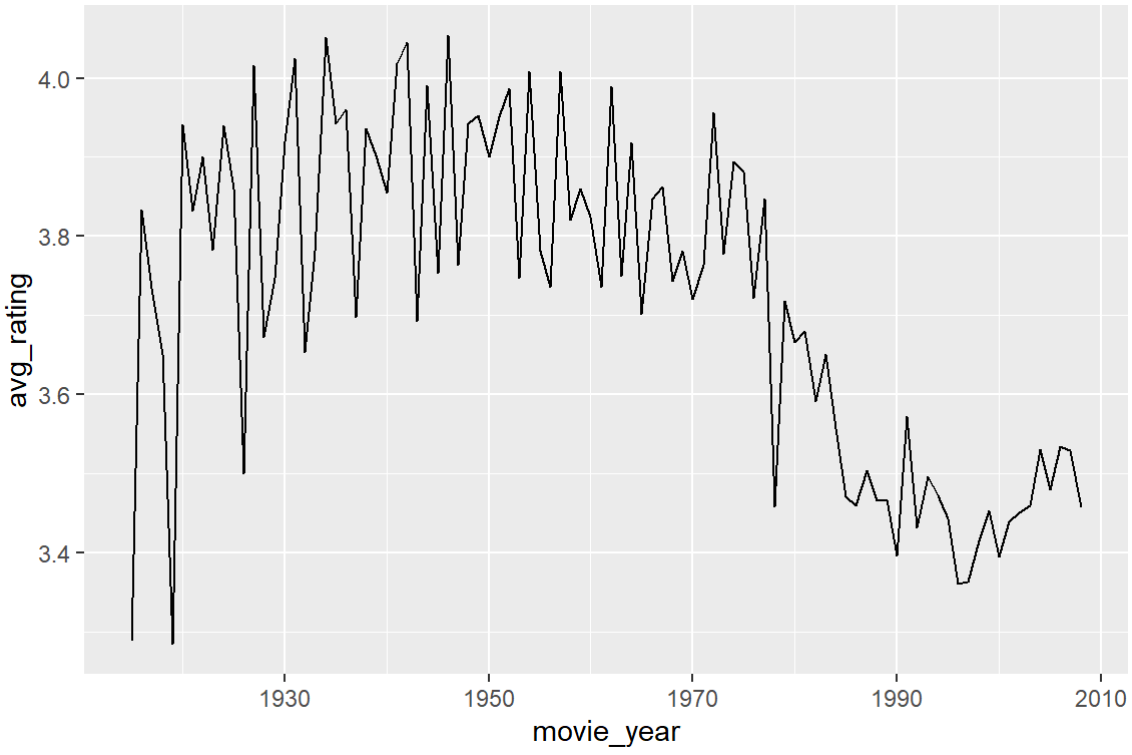
```
edx %>%
  group_by(userId) %>%
  summarise(avg_rating = mean(rating),
            count = n()) %>%
  filter(count > 50) %>%
  slice(1:20) %>%
  ggplot(aes(x=as.factor(userId), y = avg_rating - mu)) +
  geom_bar(stat = "identity")
```




There seems to be significant variability among users' averages, say a "user effect" or "user bias".

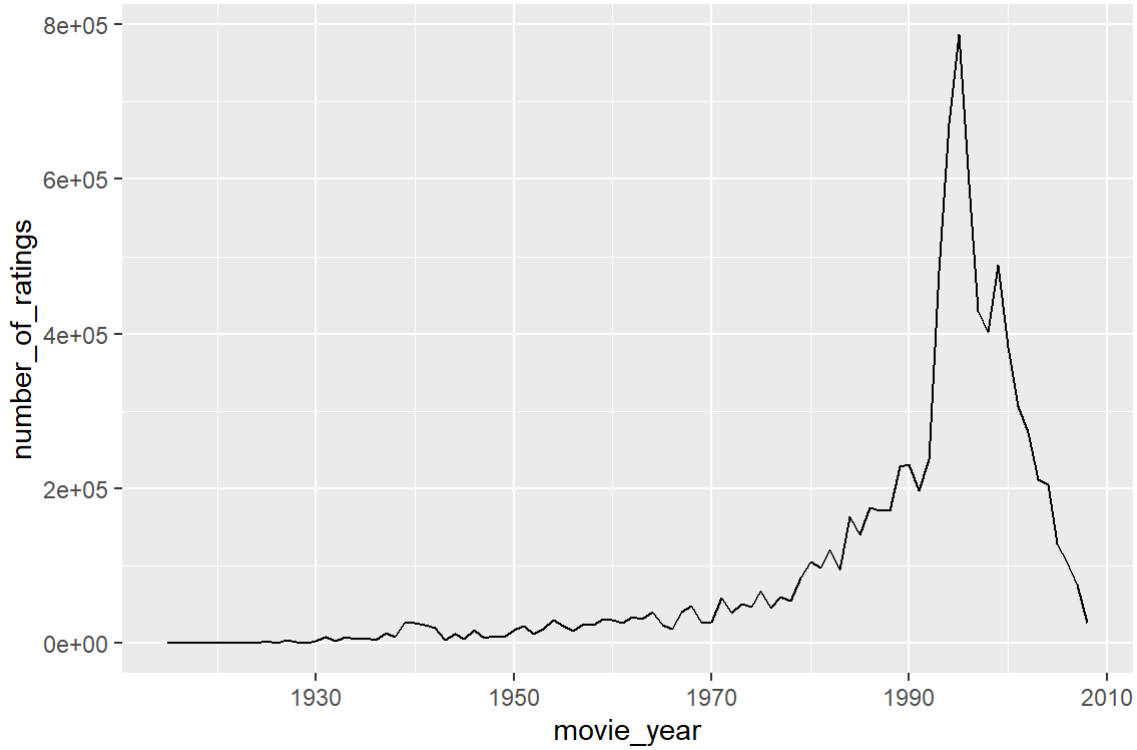
The movie year may be considered as an important feature to explain how users rate movies.

```
movies %>%
  mutate(movie_year = str_remove_all(str_extract(title, pattern = "\\(\\d{4}\\)$"), pattern =
"\\(|\\)")) %>%
  select(movieId, movie_year) %>%
  right_join(edx, by = "movieId") %>%
  group_by(movie_year) %>%
  summarise(avg_rating = mean(rating)) %>%
  mutate(movie_year = as.integer(movie_year)) %>%
  ggplot(aes(x = movie_year, y = avg_rating)) + geom_line()
```



It seems that movies after 1980 have, in average, a lower rating than movies before 1980. But more variability in the average from year to year is also observed and it may be explained because movies before 1980 are much less frequently rated than movies after 1980:

```
movies %>%
  mutate(movie_year = str_remove_all(str_extract(title, pattern = "\\(\\d{4}\\)$"), pattern =
"\\(|\\)")) %>%
  group_by(movie_year) %>%
  select(movieId, movie_year) %>%
  right_join(edx, by = "movieId") %>%
  summarise(number_of_ratings = n()) %>%
  mutate(movie_year = as.integer(movie_year)) %>%
  ggplot(aes(x = movie_year, y = number_of_ratings)) + geom_line()
```




If the movie year is to be taken into account for a model further analysis shall be conducted in order to take into account the difference in the number of ratings.

Ratings seem to have been issued (or loaded into the data set) in different ways. Some users have all their ratings assigned to a single date. For users that rated 10 or more movies on a single date it is possible to assure that those ratings were not issued in the very same day the movie was watched.

```
edx %>%
  as_tibble() %>%
  mutate(timestamp = as.POSIXct(timestamp, origin = "1970-01-01", tz = "UTC")) %>%
  group_by(userId, date = date(timestamp)) %>%
  summarise(count = n()) %>%
  filter(count > 10)
```

```
## # A tibble: 129,133 x 3
## # Groups:   userId [69,730]
##    userId date       count
##     <int> <date>     <int>
##  1      1 1996-08-02    19
##  2      2 1997-07-07    17
##  3      3 2005-12-03    14
##  4      4 1996-10-04    35
##  5      5 1997-03-09    74
##  6      6 2001-09-21    39
##  7      7 2003-04-08    41
##  8      7 2003-04-11    19
##  9      7 2003-05-30    17
## 10      8 2005-03-23    51
## # ... with 129,123 more rows
```

When a user has rated up to 3 movies in a single date, it might be considered that the rating was issued lively, that is: the rating was issued immediately after watching the movie. Should that be the case, lively rating represents a small amount of the data: only 344364 observations out of 9000055 (3.8%) would be considered lively rated.

```
edx %>%
  as_tibble() %>%
  mutate(timestamp = as.POSIXct(timestamp, origin = "1970-01-01", tz = "UTC")) %>%
  group_by(userId, date = date(timestamp)) %>%
  summarise(count = n()) %>%
  mutate(lively_rated = if_else(count < 4, TRUE, FALSE)) %>%
  group_by(lively_rated) %>%
  summarise(total = sum(count))
```

```
## # A tibble: 2 x 2
##   lively_rated   total
##   <lgl>          <int>
## 1 FALSE        8655691
## 2 TRUE          344364
```

It seems that further analysis needs to be conducted to better understand how should the timestamp variable be used. Not having done that, the timestamp variable is removed in order to get a smaller object size.

```
edx <- edx %>% select(-timestamp)
```

# 2.2. Questions and approaches

A priori any observation with all its variables could be useful when trying to predict the rating for another observation.

Having initially explored the data several questions can be posed:

1. How good a naive algorithm that predicts the global mean, 3.512 stars, for every movie and every user at any time would do?
2. How much would that approach improve when complemented with a movie effect and a user effect? Which of them shall be considered first?

3. Do the algorithm improves when considering genre? Shall it be done globally or for each user individually?

These questions are addressed in the following sections while the model is developed.

Despite the choice to look for a model that addresses those questions, it is important to note that many others could be posed, for example:

- Can the timestamp variable be used to explain more variability? Is there a month, season or day effect? Do users tend to rate higher say in May rather than in January? Do they tend to rate differently on business days and weekends? If that does not hold for the users as a whole: is it valid for each user individually?
- How much more variability would the model explain if a combined “year + genre” effect for each user is considered? Say a user likes very much drama movies from the 50s and dislikes comedies from the 90s, while other user only likes action movies from the last 5 years and dislikes anything else. Is the data set large enough to account for this effect without overtraining?
- Is it useful to consider the user’s trend to use or not use half stars when rating?
- Can a similarity approach be considered? For every rating to be predicted, the algorithm could look for the most similar observations, considering which users and which movies are similar to those to be predicted.

# 2.3. Spliting training and test set

For developing the model it is necessary to try some approaches and evaluate how good they are. Therefore the “edx” is split in order to generate training and test sets with a 90%-10% proportion. It must be noted that this is an arbitrary choice. Nevertheless, this split shall be considered just preliminary: once the model is developed a K-fold and cross-validation procedure will be conducted in order to calculate the ultimate parameters, to be sure that the model was not overtrained to this test set and to get the most out of the data.

```
# Test set will be 10% of edx data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train <- edx[-test_index,]
temp <- edx[test_index,]

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

# Add rows removed from test set back into training set
removed <- anti_join(temp, test)
train <- rbind(train, removed) %>% arrange(userId, movieId)

rm(test_index, temp, removed, edx)
```

# 2.4. RMSE function

In this section the RMSE function used to evaluate the performance is defined:

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

# 2.5.1. Naive model with just the global mean as predictor

In this model all predictions are made with the rating mean calculated for the training set.

```
mu <- mean(train$rating) #gets the mean for the training set

predicted_ratings <- test %>%
  mutate(pred = mu) %>% #for each observation on the test set makes the prediction
  pull(pred)

RMSE(test$rating, predicted_ratings)
```

```
## [1] 1.060054
```

# 2.5.2. Movie effect

This model adds a term that considers the average of biases for each movie, relative to the global mean. For each movie, a bias is calculating substracting the global mean to the rating. Then the average of these biases is calculated for each movie independently. In order to deal with movies that have few observations and could have great variability regularization is used. Therefore a term “l_i” is used to penalize these cases. “l_i” is a parameter to be tuned.

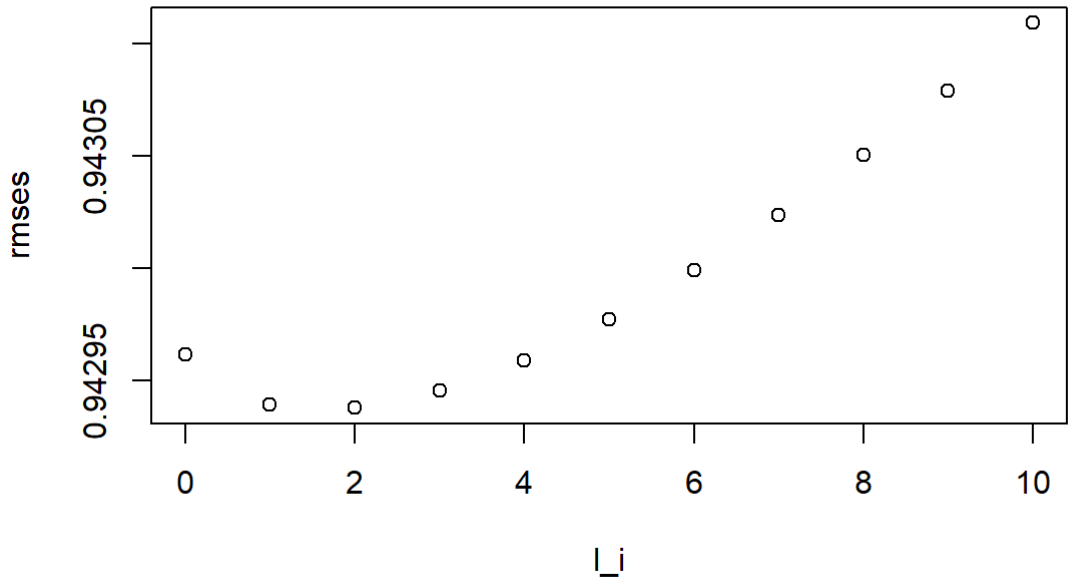
```
l_i <- 5 #term used to penalize movies with few observations

movie_eff <- function(train, test, l_i){
  mu <- mean(train$rating) #gets the mean for the training set

  b_i <- train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l_i)) #get the difference relative to global mean f
or each movie
  predicted_ratings <- test %>%
    left_join(b_i, by = "movieId") %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)

}

predicted_ratings <- movie_eff(train, test, l_i)

RMSE(test$rating, predicted_ratings)
```

```
## [1] 0.9429772
```

Some improvement is identified.

For the tuning of l_i several possibilities are tried and the one that makes the best performance is kept.

```
l_i <- seq(from=0, to = 10, by = 1) #numbers from 1 to 20 will be tried

rmses <- sapply(l_i, function(X){
  predicted_ratings <- movie_eff(train, test, X)

  RMSE(test$rating, predicted_ratings)
})
plot(rmses, x = l_i)
```




l_i = 2 gets the RMSE's minimum value.

# 2.5.3. User effect

Just as for movie effect, the same analysis is conducted for the user effect.

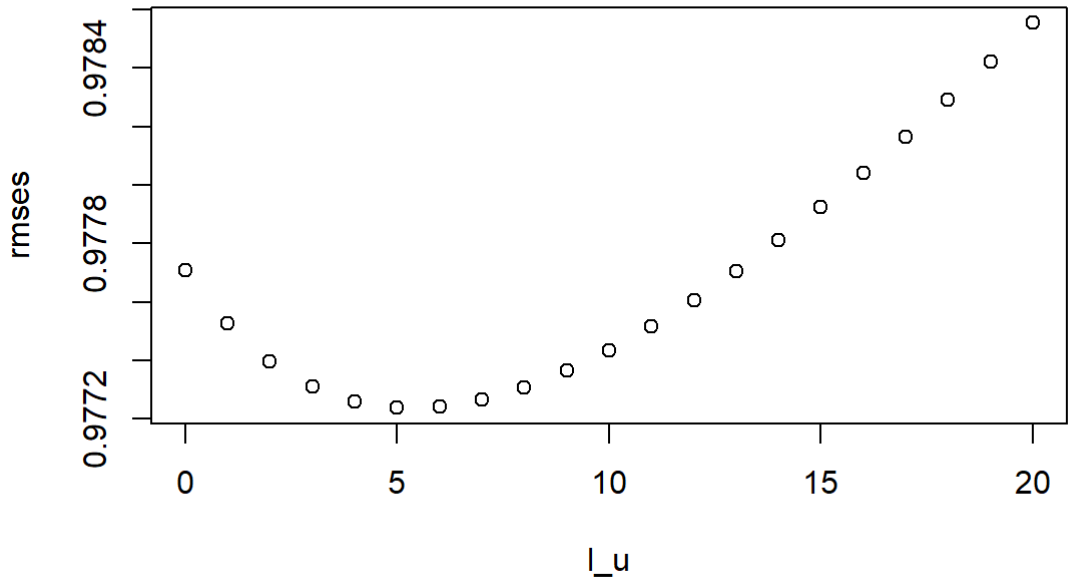
```
l_u <- 5 #term used to penalize users with few observations

user_eff <- function(train, test, l_u){
  b_u <- train %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu)/(n()+l_u)) #get the difference relative to global mean f
or each user

  predicted_ratings <- test %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_u) %>%
    pull(pred)
}

predicted_ratings <- user_eff(train, test, l_u)

RMSE(test$rating, predicted_ratings)
```

```
## [1] 0.9772382
```

Some improvement is found, but it seems that the movie effect explains more variability.

“l_u” is a parameter to be tuned.

```
l_u <- seq(from=0, to = 20, by = 1) #numbers from 1 to 20 will be tried

rmses <- sapply(l_u, function(X){
  predicted_ratings <- user_eff(train, test, X)

  RMSE(test$rating, predicted_ratings)
})
plot(rmses, x = l_u)
```




l_u = 5 gets the RMSE’s minimum value.

# 2.5.4. Both movie and user effects

For the combination of both effects, the term related to the user considers the average of biases in rating, relative to the global mean and movie effect.

```
l_i <- 2 #term used to penalize movies with few observations
l_u <- 5 #term used to penalize movies with few observations

lambdas <- expand.grid(l_i, l_u)
names(lambdas) <- c("li", "lu")

m_and_u_eff <- function(train, test, lambdas, i = 1){
  l_i <- lambdas[i,]$li
  l_u <- lambdas[i,]$lu

  mu <- mean(train$rating)

  b_i <- train %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+l_i)) #get the difference relative to global mean for
each movie

  b_u <- train %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+l_u)) #get the difference relative to global me
an and movie effect for each user

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

}

predicted_ratings <- m_and_u_eff(train, test, lambdas, i = 1)

RMSE(test$rating, predicted_ratings)
```

```
## [1] 0.8641614
```

There is some improvement. If the maximum and minimum of the predicted ratings are looked at it is observed that they are out of range:

```
print(paste0("maximum: ", max(predicted_ratings)))
```

```
## [1] "maximum: 5.87972197621609"
```

```
print(paste0("minimum: ", min(predicted_ratings)))
```

```
## [1] "minimum: -0.485298605051564"
```

When the global mean, the movie effect and user effect got superposed, some predictions got out of range.

It may be considered that there is no point in correcting for this undesirable effect because a rating prediction above 5 stars could be cap to 5, and so it would improve the RMSE result without making any difference in the recommendation system itself. But, as parameters are to be tuned by optimizing the RMSE, it is better to have

it corrected.

So, the algorithm shall have a final calculation step to set the predictions within desired limits: 0.5 - 5 stars. It should be noticed that 0.5 is being used as the lower limit instead of 0, because it was learnt from data exploration that there is no zero rated movie in the training set.

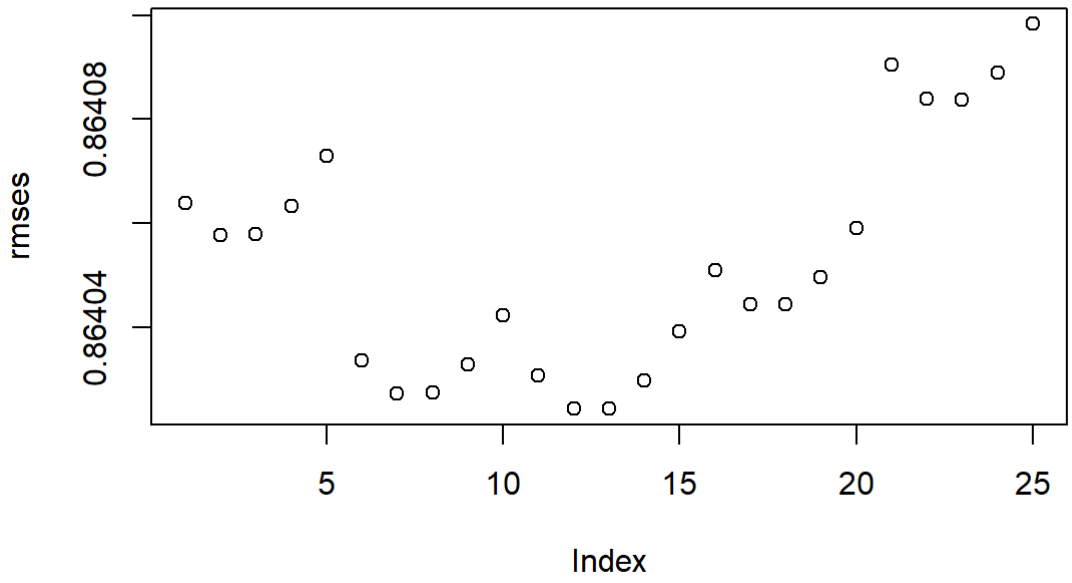
Therefore, the algorithm should look like this:

```
m_and_u_eff <- function(train, test, lambdas, i = 1){
  l_i <- lambdas[i,]$li
  l_u <- lambdas[i,]$lu

  mu <- mean(train$rating)

  b_i <- train %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+l_i)) #get the difference relative to global mean for
each movie

  b_u <- train %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+l_u)) #get the difference relative to global me
an and movie effect for each user

predicted_ratings <- test %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  mutate(pred = if_else(pred > 5, 5, pred)) %>% #if pred is higher than 5 is replaced by 5
  mutate(pred = if_else(pred < 0.5, 0.5, pred)) %>% #if pred is lower than 0.5 is replaced by
0.5
  pull(pred)

RMSE(test$rating, predicted_ratings) #RMSE calculation is the final step because predicted va
lues are no longer used

}
```

Now l_i and l_u can be tuned altogether.

```
l_i <- seq(3, 7, 1) #term used to penalize movies with few observations
l_u <- seq(3, 7, 1) #term used to penalize movies with few observations

lambdas <- expand.grid(l_i, l_u)
names(lambdas) <- c("li", "lu")

rmses <- sapply(1:length(lambdas$li), m_and_u_eff, train = train, test = test, lambdas = lamb
das)
```

```
plot(rmses)
```




The best l_i and l_u are:

```
lambdas %>% mutate(rmse = rmses) %>%
  arrange(rmse) %>%
  slice(1)
```

```
##   li lu      rmse
## 1  4  5 0.8640243
```

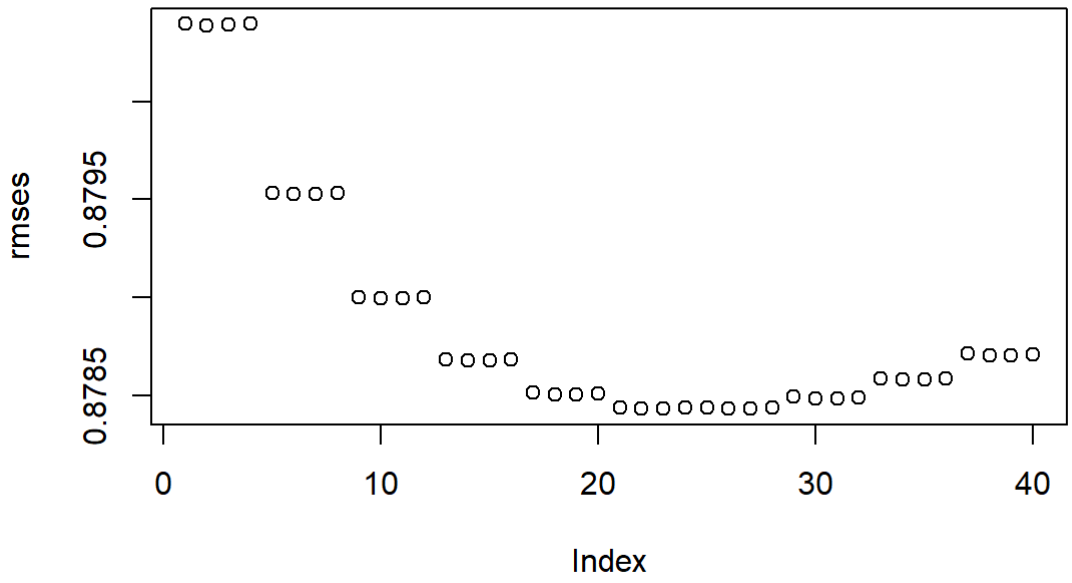
After trying several combinations of l_i and l_u, it was found that the best performance is obtained for l_i = 4 and l_u = 5. Anyhow, the ultimate parameters to be used are to be determined with cross validation.

It should be noticed that the movie effect is being calculated before the user effect and it is not the same doing it the other way round. The choice to do it that way is related to the impact of each effect: it was seen that the movie effect explains more variability than the user effect. Nevertheless, the following code performs the calculation considering the user effect before and only then the movie effect.

```
u_first_and_m_eff <- function(train, test, lambdas, i = 1){
  l_i <- lambdas[i,]$li
  l_u <- lambdas[i,]$lu

  mu <- mean(train$rating)

  b_u <- train %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu)/(n()+l_u)) #get the difference relative to global mean f
or each movie

  b_i <- train %>%
    left_join(b_u, by="userId") %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - b_u - mu)/(n()+l_i)) #get the difference relative to global
 mean and movie effect for each user

  predicted_ratings <- test %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    mutate(pred = if_else(pred > 5, 5, pred)) %>% #if pred is higher than 5 is replaced by 5
    mutate(pred = if_else(pred < 0.5, 0.5, pred)) %>% #if pred is lower than 0.5 is replaced
 by 0.5
    pull(pred)

  RMSE(test$rating, predicted_ratings)
}

l_i <- seq(1, 4, 1) #term used to penalize movies with few observations
l_u <- seq(1, 30, 3) #term used to penalize movies with few observations

lambdas <- expand.grid(l_i, l_u)
names(lambdas) <- c("li", "lu")

rmses <- sapply(1:length(lambdas$li), u_first_and_m_eff, train = train, test = test, lambdas
 = lambdas)

lambdas %>% mutate(rmse = rmses) %>%
  arrange(rmse) %>%
  slice(1)
```

```
##   li lu      rmse
## 1  2 19 0.8784294
```

```
plot(rmses)
```




It is better to compute the movie effect first.

# 2.5.5. Genre effect for users as a whole

Movies are assigned multiple genres. The movie that has the greatest amount of genres assigned to it is “Host, The (Gwoemul) (2006)” with as much as 8 genres.

```
movies %>%
  select(title,genres) %>%
  unique() %>%
  mutate(count = 1 + str_count(genres, pattern = "\\|")) %>%
  arrange(-count) %>%
  slice(1)
```

```
##                       title
## 1 Host, The (Gwoemul) (2006)
##                                                  genres count
## 1 Action|Adventure|Comedy|Drama|Fantasy|Horror|Sci-Fi|Thriller     8
```

In order to evaluate if there is a genre effect it is convenient to separate the genres variable. A new data frame “gen_by_movie” is generated for that purpose.

```
# as separate into 8 columns is being used, all movies with less than 8 genres associated to
 it
# will generate a warning "Expected 8 pieces. Missing pieces filled with NA". It is just ok.
gen_by_movie <- movies %>%
  select(movieId, genres) %>%
  unique() %>%
  separate(genres, into = c("gen1", "gen2", "gen3", "gen4", "gen5", "gen6", "gen7", "gen8"),
 sep = "\\|") %>%
  pivot_longer(cols = -movieId, values_to = "single_genre") %>%
  select(-name) %>%
  filter(!is.na(single_genre)) %>%
  group_by(movieId) %>%
  mutate(genres_number = n()) %>%
  ungroup()
```

There are 19 different genres in the training set.

```
gen_by_movie %>% pull(single_genre) %>% unique()
```

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

There is also one movie “Pull my Daisy (1958)” that has no genres listed.

```
movies %>%
  filter(str_detect(genres, "no genres listed")) %>%
  select(title, movieId, genres) %>%
  unique()
```

```
##                  title movieId             genres
## 1 Pull My Daisy (1958)    8606 (no genres listed)
```

As “Pull My Daisy” is the only movie with “no genres listed”, no correction is performed.

This model considers that after removing the mean, movie bias and user bias the remaining bias is explained by genre. As movies have more than one genre this bias is divided by the amount of genres associated to the movie (it is equally distributed among its genres). Finally the average is calculated for each genre.

```
genre_eff <- function(train, test, lambdas, i = 1){

  l_i <- lambdas[i,]$l_i
  l_u <- lambdas[i,]$l_u
  l_g <- lambdas[i,]$l_g

  mu <- mean(train$rating)

  b_i <- train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l_i)) #get the difference relative to global mean f
or each movie

  b_u <- train %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l_u)) #get the difference relative to global
 mean and movie effect for each user

  b_g <- train %>%
    select(userId, movieId, rating) %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    mutate(b_g = rating - b_u - b_i - mu) %>% #get the difference relative to global mean, mo
vie effect and user effect
    left_join(gen_by_movie, by="movieId") %>%
    mutate(b_g = b_g/genres_number) %>% #divide the bias by the number of genres the movie ha
s
    group_by(single_genre) %>%
    summarize(b_g = sum(b_g)/(n()+l_g)) #get the mean for each genre

  #when summarizing b_g, NAs shall be removed. NA may occur because the movie in the test set
that is going to be predicted may have a genre for which there is no data in the training set
for the user
  predicted_ratings <- test %>%
    select(userId, movieId) %>%
    as_tibble() %>%
    left_join(gen_by_movie, by = "movieId") %>%
    left_join(b_g, by = "single_genre") %>%
    group_by(userId, movieId) %>%
    summarise(b_g = sum(b_g, na.rm = TRUE)) %>% #NA shall be removed
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u + b_g) %>%
    mutate(pred = if_else(pred > 5, 5, pred)) %>%
    mutate(pred = if_else(pred < 0.5, 0.5, pred)) %>%
    pull(pred)

  RMSE(test$rating, predicted_ratings)
}

l_i <- seq(3,7,2)
l_u <- seq(3,7,2)
l_g <- seq(3,7,2) #term used to penalize genres with few observations

lambdas <- expand.grid(l_i, l_u, l_g)
names(lambdas) <- c("l_i", "l_u", "l_g")
```

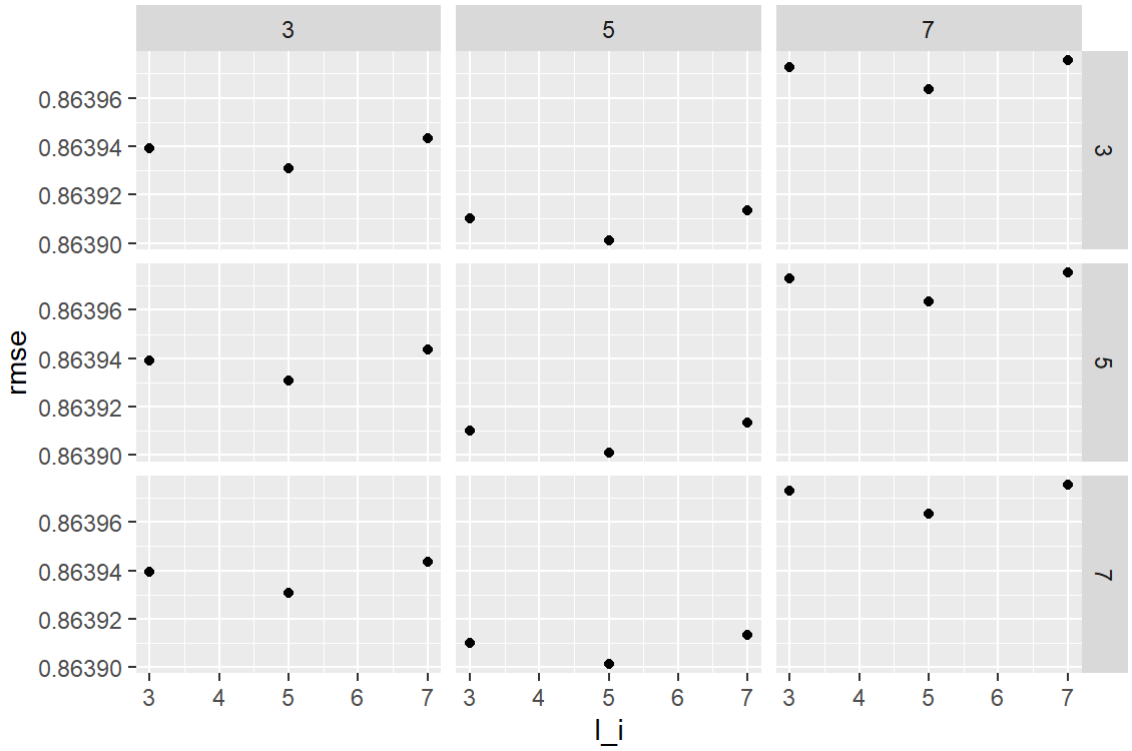
```
rmses <- sapply(1:length(lambdas$l_i), genre_eff, train = train, test = test, lambdas = lambd
as)

lambdas %>% mutate(rmse = rmses) %>%
  arrange(rmse) %>%
  slice(1)
```

```
##   l_i l_u l_g      rmse
## 1   5   5   3 0.8639012
```

Running the algorithm for each set of lambdas took about 30 seconds (in the computer this task was developed). The hardest step is calculating b_g, the table that comprises the genres biases.

In order to make it easy for the reader to run the code, the previous chunk only tries a couple of different lambdas' combinations but many more were tried when developing the algorithm. With these 27 lambda combinations it is possible to see in a plot that the lambdas chosen represent a local minimum. Plot is faceted as there are 3 independent variables (l_i, l_u, l_g). l_g get different rows and l_u get different columns.

```
lambdas %>%
  mutate(rmse = rmses) %>%
  ggplot(aes(x = l_i, y = rmse)) +
  geom_point() +
  facet_grid(l_g ~ l_u)
```




It has not improved much. Genres as a whole does not seem to explain additional variability. There may be some users that prefer comedies while others do not and they may be canceling their effect each other.

It may be better to consider a genre bias for each user.

```
rm(l_g, genre_eff)
```

# 2.5.6. Genre effect for users individually

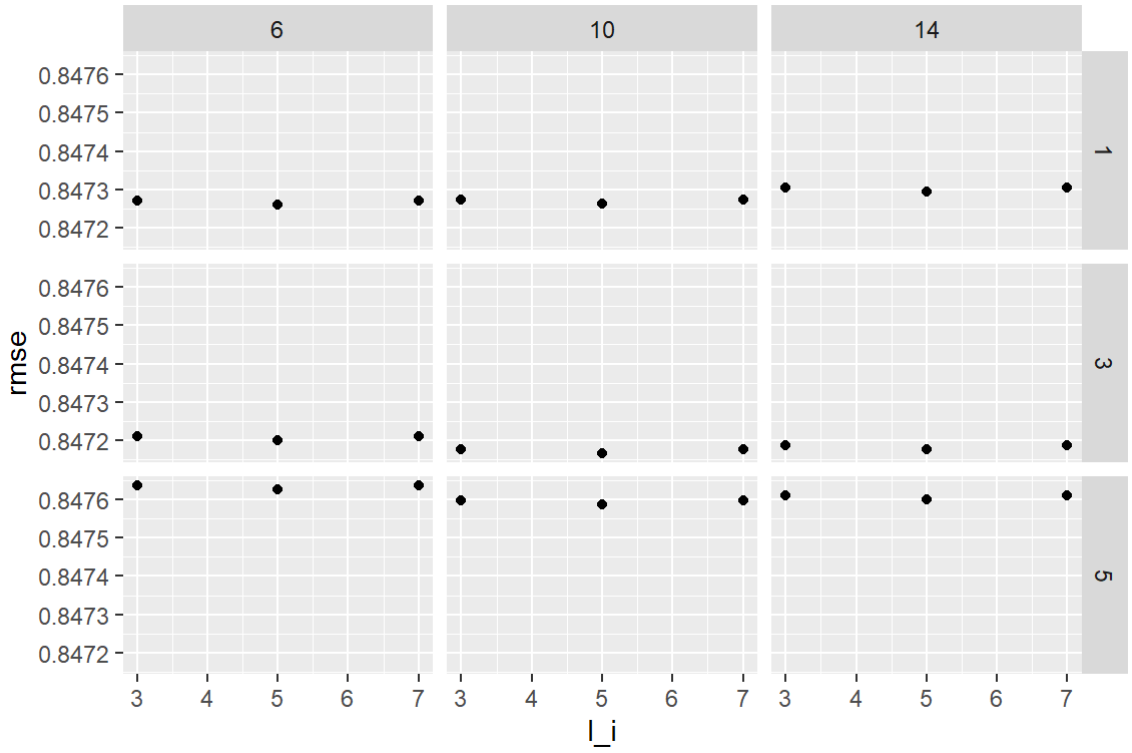
This model's approach is similar to the previous one. It considers that after removing the mean, movie bias and user bias the remaining bias is explained by genre. As movies have more than one genre this bias is divided by the amount of genres associated to the movie (it is equally distributed among genres). But then the average is calculated for each genre and user, rather than just for genre.

```
gen_by_user_eff <- function(train, test, lambdas, i, return_list = FALSE){

  l_i <- lambdas[i,]$l_i
  l_u <- lambdas[i,]$l_u
  l_g_u <- lambdas[i,]$l_g_u

  mu <- mean(train$rating)

  b_i <- train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l_i)) #get the difference relative to global mean f
or each movie

  b_u <- train %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l_u)) #get the difference relative to global
 mean and movie effect for each user

  b_g_u <- train %>%
    select(userId, movieId, rating) %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    mutate(b_g_u = rating - b_u - b_i - mu) %>%
    left_join(gen_by_movie, by="movieId") %>%
    mutate(b_g_u = b_g_u/genres_number) %>%
    group_by(userId, single_genre) %>%
    summarize(b_g_u = sum(b_g_u)/(n()+l_g_u)) %>%
    ungroup()

  predicted_ratings <- test %>%
    select(userId, movieId) %>%
    left_join(gen_by_movie, by = "movieId") %>%
    left_join(b_g_u, by = c("userId", "single_genre")) %>%
    group_by(userId, movieId) %>%
    summarise(b_g_u = sum(b_g_u, na.rm = TRUE)) %>% #ver que hay que sacar los NA
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u + b_g_u) %>%
    mutate(pred = if_else(pred > 5, 5, pred)) %>% #las 2 lineas que siguen son para acotar
    mutate(pred = if_else(pred < 0.5, 0.5, pred)) %>%
    pull(pred)

  rmse <- RMSE(test$rating, predicted_ratings)

  #this "return list" is used to get the predictions both on the training and the test set. I
t was useful to analyze errors during development
  if (return_list == TRUE){
    pred_train <- train %>%
      select(userId, movieId) %>%
      left_join(gen_by_movie, by = "movieId") %>%
      left_join(b_g_u, by = c("userId", "single_genre")) %>%
      group_by(userId, movieId) %>%
      summarise(b_g_u = sum(b_g_u, na.rm = TRUE)) %>% #ver que hay que sacar los NA
      left_join(b_i, by = "movieId") %>%
      left_join(b_u, by = "userId") %>%
      mutate(pred = mu + b_i + b_u + b_g_u) %>%
```

```
      mutate(pred = if_else(pred > 5, 5, pred)) %>% #las 2 lineas que siguen son para acotar
      mutate(pred = if_else(pred < 0.5, 0.5, pred)) %>%
      pull(pred)
    result_list <- list(rmse, predicted_ratings, pred_train)
    names(result_list) <- c("rmse", "pred_test", "pred_train")
    return(result_list)
  } else {
      return(rmse)
  }

}

l_i <- seq(from = 3, to = 7, by = 2)
l_u <- seq(from = 6, to = 14, by = 4)
l_g_u <- seq(from = 1, to = 5, by = 2)

lambdas <- expand.grid(l_i, l_u, l_g_u)
names(lambdas) <- c("l_i", "l_u", "l_g_u")

rmses <- sapply(1:length(lambdas$l_i), gen_by_user_eff, train = train, test = test, lambdas =
lambdas, return_list = FALSE)

lambdas %>% mutate(rmse = rmses) %>%
  arrange(rmse) %>%
  slice(1)
```

```
##   l_i l_u l_g_u      rmse
## 1   5  10     3 0.8471672
```

```
lambdas %>%
  mutate(rmse = rmses) %>%
  ggplot(aes(x = l_i, y = rmse)) +
  geom_point() +
  facet_grid(l_g_u ~ l_u)
```



RMSE = 0.847 is achieved. However parameters have been tuned with the test set and they could have been overtrained to do well on it. The next section use the complete edx data set to perform cross validation.

# 2.7. Final parameter tuning

Cross validation with a K-fold approach is used. 5 folds are chosen.

First, the original edx data set is reconstructed from train and test sets.

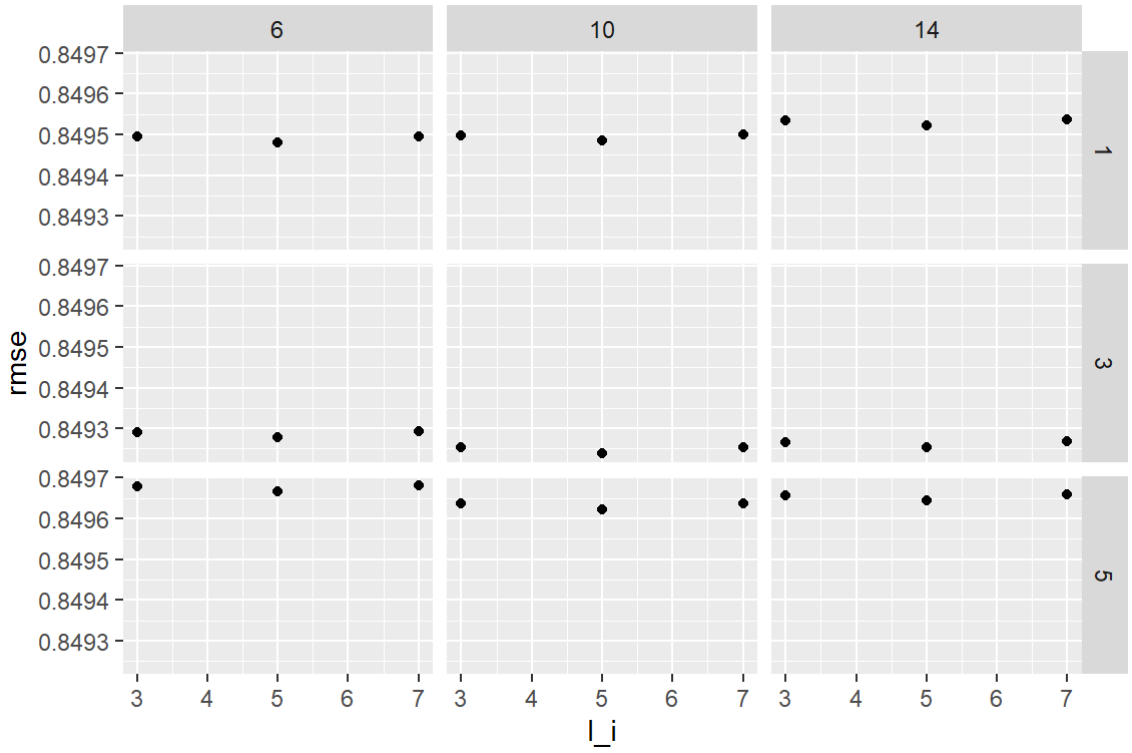
```
edx <- train %>% bind_rows(test) %>% arrange(userId, movieId)
rm(train, test)
```

```
set.seed(1, sample.kind = "Rounding")
indexes <- sample(length(edx$rating))

cross_valid <- function(df, indexes, k, folds, lambdas){
  valid_index <- indexes[(1+length(indexes)/folds*(k-1)):(length(indexes)/folds*k)]

  train_set <- df[-valid_index,]
  temp <- df[valid_index,]

  # Make sure userId and movieId in validation set are also in training set
  valid_set <- temp %>%
    semi_join(train_set, by = "movieId") %>%
    semi_join(train_set, by = "userId") %>%
    arrange(userId, movieId)

  # Add rows removed from validation set back into training set
  removed <- anti_join(temp, valid_set)
  train_set <- rbind(train_set, removed) %>%
    arrange(userId, movieId)

  rm(df, valid_index, temp, removed)
  gc()

  rmses <- sapply(1:length(lambdas$l_i), gen_by_user_eff, train = train_set, test = valid_se
t, lambdas = lambdas, return_list = FALSE)

  rmses <- lambdas %>% mutate(rmse = rmses)
  return(rmses)
}

l_i <- seq(from = 3, to = 7, by = 2)
l_u <- seq(from = 6, to = 14, by = 4)
l_g_u <- seq(from = 1, to = 5, by = 2)

lambdas <- expand.grid(l_i, l_u, l_g_u)
names(lambdas) <- c("l_i", "l_u", "l_g_u")

rmses <- lapply(1:5, cross_valid, df = edx, indexes = indexes, folds = 5, lambdas = lambdas)
```

Running the algorithm for each set of lambdas and each fold took about 30 seconds (in the computer this task was developed). The hardest step is calculating b_g_u, the table that comprises the genres biases for each user. Once parameters have been tuned, this table can be stored and it is not necessary to calculate b_g_u again. Therefore training is time consuming while predicting is not.

In order to make it easy for the reader to run the code, the previous chunk only tries a couple of different lambdas' combinations but many more were tried when developing the algorithm.

```
rmses %>%
  bind_rows() %>%
  group_by(l_i, l_u, l_g_u) %>%
  summarise(rmse = mean(rmse)) %>%
  ggplot(aes(x = l_i, y = rmse)) +
  geom_point() +
  facet_grid(l_g_u ~ l_u)
```




```
rmses %>%
  bind_rows() %>%
  group_by(l_i, l_u, l_g_u) %>%
  summarise(rmse = mean(rmse))%>%
  ungroup() %>%
  arrange(rmse) %>%
  slice(1)
```

```
## # A tibble: 1 x 4
##     l_i   l_u l_g_u  rmse
##   <dbl> <dbl> <dbl> <dbl>
## 1     5    10     3 0.849
```

The best RMSE is 0.849 and is achieved when l_i = 5, l_u = 10 and l_g_u = 3.

# 2.8. Training the model with the entire edx data set

Having completed the parameter tuning, the model can now be trained on the entire edx data set. The output of this step are the b_i, b_u and b_g_u that are ultimately used to make predictions.

```
l_i <- 5
l_u <- 10
l_g_u <- 3

mu <- mean(edx$rating)

b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l_i)) #get the difference relative to global mean f
or each movie

b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l_u)) #get the difference relative to global
 mean and movie effect for each user

b_g_u <- edx %>%
    select(userId, movieId, rating) %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    mutate(b_g_u = rating - b_u - b_i - mu) %>%
    left_join(gen_by_movie, by="movieId") %>%
    mutate(b_g_u = b_g_u/genres_number) %>%
    group_by(userId, single_genre) %>%
    summarize(b_g_u = sum(b_g_u)/(n()+l_g_u)) %>%
    ungroup()
```

Now the model has been trained and predictions can be made.

# 3.1. RMSE calculation on the validation data set

The following chunk calculated the predictions for the validation set. This is the first time the validation set is used for anything.

```
predicted_ratings <- validation %>%
    select(userId, movieId) %>%
    left_join(gen_by_movie, by = "movieId") %>%
    left_join(b_g_u, by = c("userId", "single_genre")) %>%
    group_by(userId, movieId) %>%
    summarise(b_g_u = sum(b_g_u, na.rm = TRUE)) %>% #ver que hay que sacar los NA
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u + b_g_u) %>%
    mutate(pred = if_else(pred > 5, 5, pred)) %>% #las 2 lineas que siguen son para acotar
    mutate(pred = if_else(pred < 0.5, 0.5, pred)) %>%
    pull(pred)
```

With the predicted ratings the RMSE is calculated:

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

```
## [1] 0.84734
```

# 3.2. Comparison to the RMSE achieved in the training data set

The best RMSE obtained with the edx data set was 0.849. The ultimate RMSE obtained with the validation data set is 0.847.

Overtraining to the edx data set is therefore not observed.

Performance on the validation data set seems to be slightly better. This can be caused because the predictions on the validation set were calculated with a model that had been trained with the whole edx data set while the others were trained on subsets comprising 80% each. Of course, it can also be caused by luck. Anyhow the difference is very little.

# 3.3. Discussion on the RMSE as key performance indicator

The goal of this task is to get an RMSE as small as possible, but does it really represent what a recommendation system is meant for?

RMSE penalizes to the same a extent a prediction of 0.5 when the actual rating is 1.5, than it does for a prediction of 3.5 when the actual rating is 4.5.

It could be thought that the most important task of the system is to predict high ratings accurately so that the prediction is used to make a suggestion. Should that be the case, the system shall be able to accurately recognize when a movie is going to be rated with 5 stars for the user, and when the system predicts a 5 star rating it should be very likely that the user is really going to like it that much.

Then high sensitivity and specificity is desired for the highest ratings, say 4.5 and 5 stars, rather than for low ratings. It does not make a great difference if the rating is 0.5 or 1.5, that movie is not going to be recommended for that user. Unlike that, it makes a considerable difference for a rating to be 3.5 or 4.5 if a recommendation is to be made.

If that was to be the case, RMSE should be replaced by another key performance indicator to optimize when developing an algorithm.

# 4. Conclusions

The final model considered 3 biases that added to the rating global mean:

- movie biases
- user biases
- genre per user biases

RMSE = 0.847 was achieved for that model.

The order in which movie and user biases were calculated was considered and it was found that it was better to compute movie bias first.

For the genre effect, a genre bias for users as a whole was for initially considered, but the discarded because better performance was achieved when considering genres for users individually.

Regularization parameters were tuned on the whole edx data set using cross validation and K-fold with 5 folds.

In the questions section other approaches were posed but not addressed in this report and may be considered for future improvement. There is also some discussion about the goodness of using RMSE as key performance indicator regarding the ultimate goal of a recommendation system.