Movielens

Massimo Palme

30/5/2020

1. INTRODUCTION

In this report the "Movielens" data_base is analysed and a machine learning strategy to predict root mean square error (RMSE) for rating is turned out by using different techniques. Data compounds of 10.000.000 movies ratings assigned by users. Information available is: movie Id and title, user Id, rating, genres and timestamp. The goal of the project is to obtain a RMSE < 0.86490. I obtained a RMSE < 0.86450, better than expected. Key points: I considered movie effect, user effect, genres effect and I regularized the matrix with cross validation.

In the following sections, I describe methods, results and conclusions of the project. I inserted the complete R code and some plots and tables. At the end the RMSE is validated and higlighted.

2. METHODS

First I created the edx and validation sets and looked at the dimensions and structure of the edx data set. Then I performed machine learning to obtain RMSE considering movie, user and genre effect. Then I did a regulraziation process. Finally, I improved the regularization by using a k-fold cross-validation.

2.1 First step, create the edx and validation sets and look at edx set (dimensions and head)

Code and outputs:

```
################################
# 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")

## Loading required package: tidyverse

## ── Attaching packages ─────────────────────────────────────────
tidyverse 1.3.0 ──

## ✔ ggplot2 3.3.0     ✔ purrr   0.3.4
## ✔ tibble  3.0.1     ✔ dplyr   0.8.5
```

```
## ✔ tidyr   1.0.2     ✔ stringr 1.4.0
## ✔ readr   1.3.1     ✔ forcats 0.5.0

## ── Conflicts ─────────────────────────────────────────────────
tidyverse_conflicts() ──
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

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

```
## Loading required package: caret
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

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

```
## Loading required package: data.table
```

```
##
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```
## The following object is masked from 'package:purrr':
##
##     transpose
```

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

```
##
## Attaching package: 'pillar'
```

```
## The following object is masked from 'package:dplyr':
##
##     dim_desc
```

```
# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
```

```r
# 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")

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform
'Rounding' sampler
## used

# 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") %>%
  semi_join(edx, by = "genres")

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)

## Joining, by = c("userId", "movieId", "rating", "timestamp", "title",
"genres")

edx <- rbind(edx, removed)
dim(edx)

## [1] 9000055       6
```

```
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
```

2.2 Then, I defined train and test sets

Code:

```
#Define train and test sets
# Test set will be 10% of edx data
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform
'Rounding' sampler
## used
```

```
# if using R 3.5 or earlier, use `set.seed(1)` instead
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1,
list = FALSE)
train <- movielens[-test_index,]
temp <- movielens[test_index,]

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

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, test)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title",
"genres")
```

```
train <- rbind(train, removed)
```

2.3 Then I defined the RMSE function

Code:

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

2.4 First assessment

I starded tryng just the mean rating, then I improved the model by considering the movie effect, the user effect and the genres effect

Code and output (table):

```
#define mean rating
mu_hat <- mean(train$rating)
mu_hat

## [1] 3.512178

#rmse using mean rating
naive_rmse <- RMSE(test$rating, mu_hat)
naive_rmse

## [1] 1.060232

#evaluating movie effect
mu <- mean(train$rating)
movie_avgs <- train %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

predicted_ratings <- mu + test %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
#rmse movie effect
movie_effect_rmse<-RMSE(predicted_ratings, test$rating)
#evaluating users effect
user_avgs <- train %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu))

predicted_ratings <- test %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_u) %>%
  pull(pred)
#rmse users effect
users_effect_rmse<-RMSE(predicted_ratings, test$rating)
#evaluating genre effect
genre_avgs <- train %>%
```

```
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu))

predicted_ratings <- test %>%
  left_join(genre_avgs, by='genres') %>%
  mutate(pred = mu + b_g) %>%
  pull(pred)
#rmse genre effect
genre_effect_rmse<-RMSE(predicted_ratings, test$rating)

#evaluating combined movie, user and genre effects
user_avgs <- train %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

genre_avgs<-train %>%
  left_join(movie_avgs,by='movieId') %>%
  left_join(user_avgs,by='userId') %>%
  group_by(genres)%>%
  summarize(b_g=mean(rating-mu-b_i-b_u))

predicted_ratings <- test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs,by='genres') %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)
#rmse combined
movie_users_genre_effect_rmse<-RMSE(predicted_ratings, test$rating)

#table of results
rmse_results <- tibble(method = c("Just the average","Movie
effect","Users effect","Genre effect","Movie, users and genre effect"),
RMSE =
c(naive_rmse,movie_effect_rmse,users_effect_rmse,genre_effect_rmse,movie_
users_genre_effect_rmse))

rmse_results

## # A tibble: 5 x 2
##   method                          RMSE
##   <chr>                          <dbl>
## 1 Just the average               1.06
## 2 Movie effect                   0.945
## 3 Users effect                   0.978
## 4 Genre effect                   1.02
## 5 Movie, users and genre effect 0.866
```

The combined movie, user and genres effect modeling leads to a RMSE of 0.8656, not bad but could be improved by regularization.

2.5 Regularization

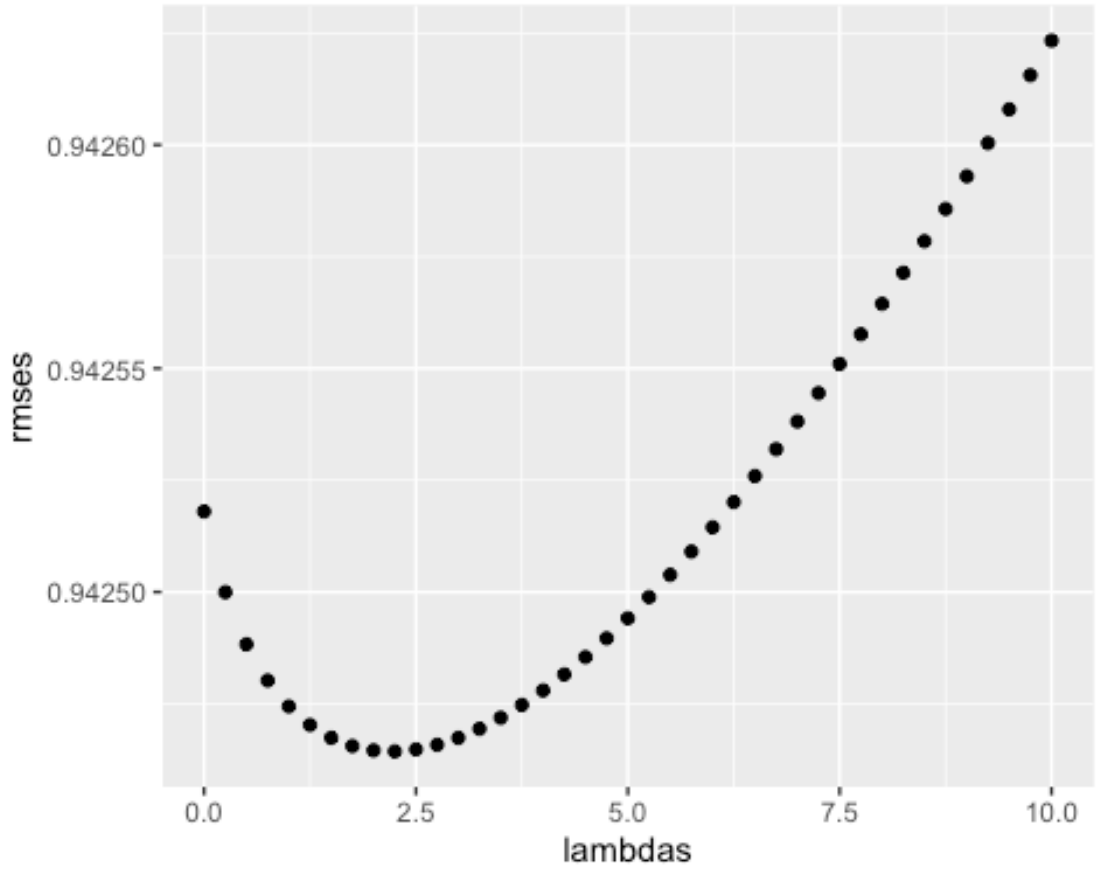
I first tryed a single regularization with a cross-validation test (indipendent from my test set).

2.5.1 Regularization by movie effect

Code and output (figure and lambda value):

```
#REGULARIZATION
# define a cross-validation set for movies
set.seed(1, sample.kind="Rounding")

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform
'Rounding' sampler
## used

# if using R 3.5 or earlier, use `set.seed(1)` instead
test_index <- createDataPartition(y = train$rating, times = 1, p = 0.1,
list = FALSE)
edx_1 <- train[-test_index,]
temp <- train[test_index,]

# Make sure userId and movieId in cross-validation set are also in edx_1
set
cross_validation <- temp %>%
  semi_join(edx_1, by = "movieId") %>%
  semi_join(edx_1, by = "userId") %>%
  semi_join(edx_1, by = "genres")

# Add rows removed from validation set back into edx_1 set
removed <- anti_join(temp, cross_validation)

## Joining, by = c("userId", "movieId", "rating", "timestamp", "title",
"genres")

edx_1 <- rbind(edx_1, removed)

#test values for lambda
lambdas <- seq(0, 10, 0.25)

mu <- mean(edx_1$rating)
just_the_sum <- edx_1 %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())

rmses <- sapply(lambdas, function(l){
  predicted_ratings <- cross_validation %>%
```

```
    left_join(just_the_sum, by='movieId') %>%
    mutate(b_i = s/(n_i+l)) %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)
  return(RMSE(predicted_ratings, cross_validation$rating))
})
qplot(lambdas, rmses)
```




```
lambdas[which.min(rmses)]
```

```
## [1] 2.25
```

Figure shows the lambda that minimizes RMSE for movie regularization

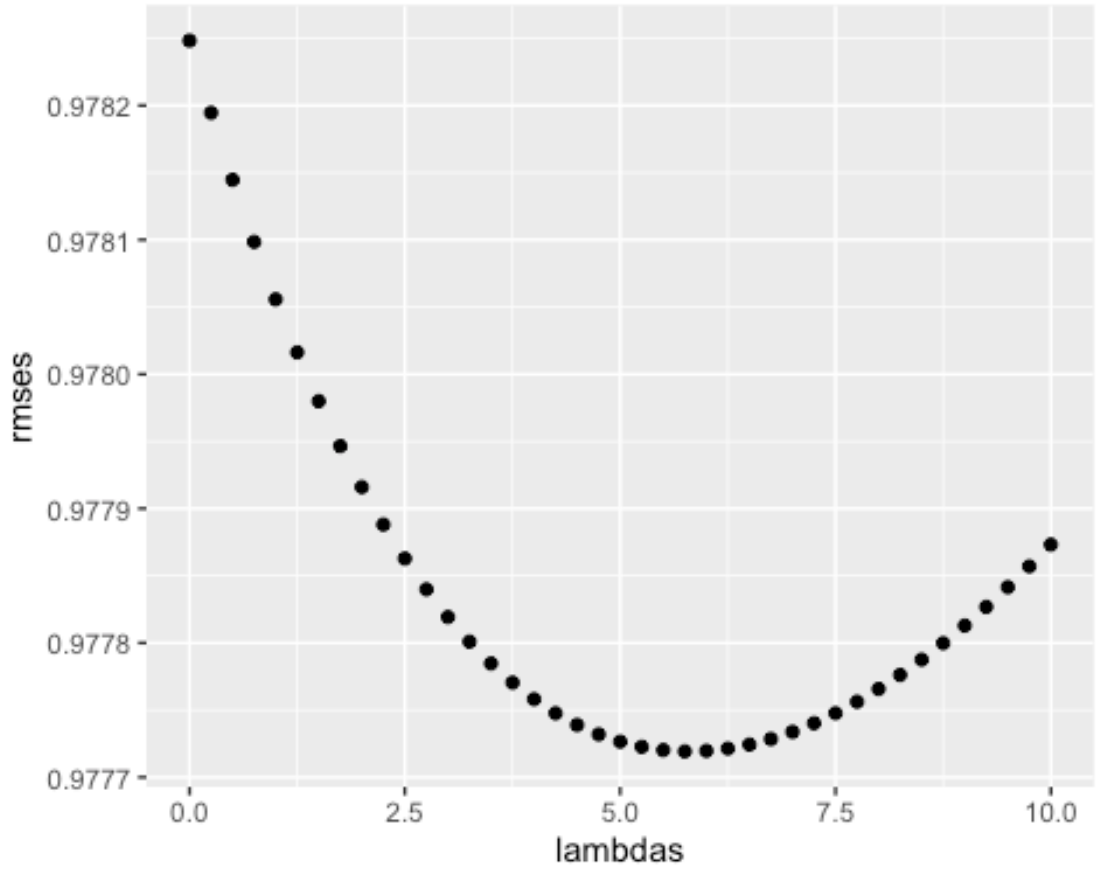
2.5.2 Regularization by user effect

Now I repeated for user effect

Code and output:

```
#now repeat for users


# cross-validation set
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform
'Rounding' sampler
## used

# if using R 3.5 or earlier, use `set.seed(1)` instead
test_index <- createDataPartition(y = train$rating, times = 1, p = 0.1,
list = FALSE)
edx_1 <- train[-test_index,]
temp <- train[test_index,]

# Make sure userId and movieId in validation set are also in edx set
cross_validation <- temp %>%
  semi_join(edx_1, by = "movieId") %>%
  semi_join(edx_1, by = "userId") %>%
  semi_join(edx_1, by ="genres")
# Add rows removed from validation set back into edx set
removed <- anti_join(temp, cross_validation)

## Joining, by = c("userId", "movieId", "rating", "timestamp", "title",
"genres")

edx_1 <- rbind(edx_1, removed)

lambdas <- seq(0, 10, 0.25)

mu <- mean(edx_1$rating)
just_the_sum <- edx_1 %>%
  group_by(userId) %>%
  summarize(s = sum(rating - mu), n_i = n())

rmses <- sapply(lambdas, function(l){
  predicted_ratings <- cross_validation %>%
    left_join(just_the_sum, by='userId') %>%
    mutate(b_u = s/(n_i+l)) %>%
    mutate(pred = mu + b_u) %>%
    pull(pred)
  return(RMSE(predicted_ratings, cross_validation$rating))
})
qplot(lambdas, rmses)
```




```
lambdas[which.min(rmses)]
```

```
## [1] 5.75
```

Figure shows the lambda that minimizes RMSE for users

2.5.3 Regularization by genre effect

Now I repeated for genre

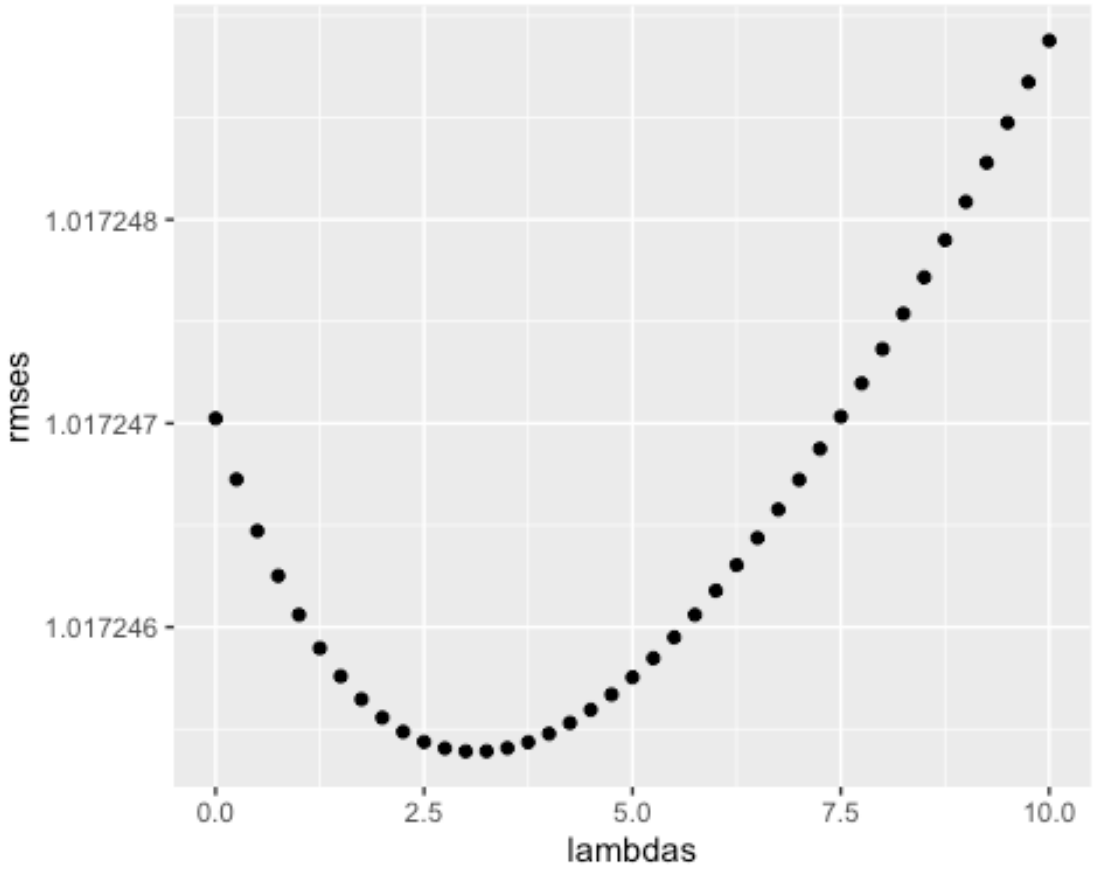
Code and output:

```
#now for genres
# cross-validation set
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform
'Rounding' sampler
## used
```

```
# if using R 3.5 or earlier, use `set.seed(1)` instead
test_index <- createDataPartition(y = train$rating, times = 1, p = 0.1,
list = FALSE)
edx_1 <- train[-test_index,]
temp <- train[test_index,]
```

```r
# Make sure userId and movieId in validation set are also in edx set
cross_validation <- temp %>%
  semi_join(edx_1, by = "movieId") %>%
  semi_join(edx_1, by = "userId") %>%
  semi_join(edx_1, by="genres")

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, cross_validation)

## Joining, by = c("userId", "movieId", "rating", "timestamp", "title",
"genres")

edx_1 <- rbind(edx_1, removed)

lambdas <- seq(0, 10, 0.25)

mu <- mean(edx_1$rating)
just_the_sum <- edx_1 %>%
  group_by(genres) %>%
  summarize(s = sum(rating - mu), n_i = n())

rmses <- sapply(lambdas, function(l){
  predicted_ratings <- cross_validation %>%
    left_join(just_the_sum, by='genres') %>%
    mutate(b_g = s/(n_i+l)) %>%
    mutate(pred = mu + b_g) %>%
    pull(pred)
  return(RMSE(predicted_ratings, cross_validation$rating))
})
qplot(lambdas, rmses)
```




```
lambdas[which.min(rmses)]
```

```
## [1] 3
```

Figure shows the lambda that minimizes RMSE for genres

So, using a singel cross-validation test, we obtain the values of lambda that minimizes RMSE for each effect and a graphical representation of the selection procedure.

Results are lambdas values of 2.25, 5.75, 3 for movie, user and genre effect respectively.

2.6 Regularization with k-fold cross-validation

Now, to better select the lambdas, I did a k-fold cross validation with k=5 for each effect (movie, user and genre). This means, that 5 times for each predictor, a cross-validation set was defined. Then, I selected the average value of lambda for each predictor.

Code and output:

```
#to select the best lambda values, perform k-fold cross-validation with
k=5
```

```r
#first for movie
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform
'Rounding' sampler
## used
```

```r
k<-5

lambda_1<-replicate(k, {
          test_index <- createDataPartition(y = train$rating, times = 1,
p = 0.1, list = FALSE)
          edx_1 <- train[-test_index,]
          temp <- train[test_index,]
          cross_validation <- temp %>%
          semi_join(edx_1, by = "movieId") %>%
          semi_join(edx_1, by = "userId") %>%
          semi_join(edx_1, by="genres")
          removed <- anti_join(temp, cross_validation)
          edx_1 <- rbind(edx_1, removed)
          lambdas <- seq(0, 10, 0.25)
          mu <- mean(edx_1$rating)
          just_the_sum <- edx_1 %>%
          group_by(movieId) %>%
          summarize(s = sum(rating - mu), n_i = n())
          rmses <- sapply(lambdas, function(l){
          predicted_ratings <- cross_validation %>%
          left_join(just_the_sum, by='movieId') %>%
          mutate(b_i = s/(n_i+l)) %>%
          mutate(pred = mu + b_i) %>%
          pull(pred)
          return(RMSE(predicted_ratings, cross_validation$rating))
          })
          qplot(lambdas, rmses)
          lambdas[which.min(rmses)]
})
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title",
"genres")
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title",
"genres")
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title",
"genres")
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title",
"genres")
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title",
"genres")
```

```r
l_1<-mean(lambda_1)
l_1
```

```
## [1] 2.3

#now for users
# cross-validation set
set.seed(1, sample.kind="Rounding")

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform
'Rounding' sampler
## used

# if using R 3.5 or earlier, use `set.seed(1)` instead

k<-5
lambda_2<-replicate(k, {
  test_index <- createDataPartition(y = train$rating, times = 1, p = 0.1,
list = FALSE)
  edx_1 <- train[-test_index,]
  temp <- train[test_index,]
  cross_validation <- temp %>%
  semi_join(edx_1, by = "movieId") %>%
  semi_join(edx_1, by = "userId") %>%
  semi_join(edx_1, by="genres")
  removed <- anti_join(temp, cross_validation)
  edx_1 <- rbind(edx_1, removed)
  lambdas <- seq(0, 10, 0.25)
  mu <- mean(edx_1$rating)
  just_the_sum <- edx_1 %>%
  group_by(userId) %>%
  summarize(s = sum(rating - mu), n_i = n())
  rmses <- sapply(lambdas, function(l){
  predicted_ratings <- cross_validation %>%
    left_join(just_the_sum, by='userId') %>%
    mutate(b_u = s/(n_i+l)) %>%
    mutate(pred = mu + b_u) %>%
    pull(pred)
  return(RMSE(predicted_ratings, cross_validation$rating))
  })
  lambdas[which.min(rmses)]
})

## Joining, by = c("userId", "movieId", "rating", "timestamp", "title",
"genres")
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title",
"genres")
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title",
"genres")
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title",
"genres")
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title",
"genres")
```

```
l_2<-mean(lambda_2)
l_2
```

```
## [1] 5.55
```

```
#then for genres

# cross-validation set
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform
'Rounding' sampler
## used
```

```
# if using R 3.5 or earlier, use `set.seed(1)` instead

k<-5
lambda_3<-replicate(k, {
  test_index <- createDataPartition(y = train$rating, times = 1, p = 0.1,
list = FALSE)
  edx_1 <- train[-test_index,]
  temp <- train[test_index,]
  cross_validation <- temp %>%
  semi_join(edx_1, by = "movieId") %>%
  semi_join(edx_1, by = "userId") %>%
  semi_join(edx_1, by="genres")
  removed <- anti_join(temp, cross_validation)
  edx_1 <- rbind(edx_1, removed)
  lambdas <- seq(0, 10, 0.25)
  mu <- mean(edx_1$rating)
  just_the_sum <- edx_1 %>%
  group_by(genres) %>%
  summarize(s = sum(rating - mu), n_i = n())
  rmses <- sapply(lambdas, function(l){
  predicted_ratings <- cross_validation %>%
    left_join(just_the_sum, by='genres') %>%
    mutate(b_g = s/(n_i+l)) %>%
    mutate(pred = mu + b_g) %>%
    pull(pred)
  return(RMSE(predicted_ratings, cross_validation$rating))
  })
  lambdas[which.min(rmses)]
})
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title",
"genres")
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title",
"genres")
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title",
"genres")
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title",
```

```
"genres")
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title",
"genres")

l_3<-mean(lambda_3)
l_3

## [1] 2.7
```

With k-fold cross-validation the values of lambda are slightly different. I will use these values in final evaluation: 2.3, 5.55 and 2.7 for movie, user and genre effect respectively.

3. RESULTS

Now, we can select the three values of lambda to be used in the final algorithm (results of the k-fold cross-validation), and look to the final results. I used only a single test set.

Code:

```
#now do regularization with the values l_1, l_2, l_3

# values of lambda (movie, user, genre) by k-fold:

l_1

## [1] 2.3

l_2

## [1] 5.55

l_3

## [1] 2.7

mu <- mean(train$rating)

b_i <- train %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+l_1))

b_u <- train %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+l_2))
```

```
b_g <- train %>%
  left_join(b_u, by="userId") %>%
  left_join(b_i, by="movieId")%>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - b_i - b_u - mu)/(n()+l_3))


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

regularized_movie_user_genre_rmse<-RMSE(predicted_ratings, test$rating)

regularized_movie_user_genre_rmse

## [1] 0.8651669

#write results

options(pillar.sigfig=5)

rmse_results <- tibble(method = c("Just the average","Movie
effect","Users effect","Genre effect","Movie, users and genre effect",
"Regularized"), RMSE =
c(naive_rmse,movie_effect_rmse,users_effect_rmse,genre_effect_rmse,movie_
users_genre_effect_rmse,regularized_movie_user_genre_rmse))

rmse_results

## # A tibble: 6 x 2
##   method                          RMSE
##   <chr>                          <dbl>
## 1 Just the average              1.0602
## 2 Movie effect                  0.94457
## 3 Users effect                  0.97777
## 4 Genre effect                  1.0186
## 5 Movie, users and genre effect 0.86564
## 6 Regularized                   0.86517
```

So I obtained a final RMSE value (tested on my test set, not on the validation set) of 0.86517, very close to the required value. I will check it with the validation set, but first I will analyze and discuss briefly the results.

Results demonstrate that:

a) the movie effect is stronger than user and genre effect. User effect is stronger than genre. Combined model leads to a good result.
b) Regularization permits to obtain a value close to the requested value by the exercise.
c) Probably the k-fold cross-validation improved the result just a little, but I did it becouse is a more correct procedure. What I did not was a cross-validation considering the test set, I just used one. But also in the textbook was underlined that in practice this is a common strategy to save computational time.

Now I will check the final value of RMSE with validation set.

```
#now check the best RMSE with validation set

mu <- mean(edx$rating)

b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+l_1))

b_u <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+l_2))

b_g <- edx %>%
  left_join(b_u, by="userId") %>%
  left_join(b_i, by="movieId")%>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - b_i - b_u - mu)/(n()+l_3))


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

regularized_movie_user_genre_rmse_check<-RMSE(predicted_ratings,
validation$rating)

regularized_movie_user_genre_rmse_check

## [1] 0.8644875
```

CONCLUSIONS

The validation test is used to confirm the RSME obtained with the train set. I used this time the complete edx set and the validation. The value is better than required (0.8644875 < 0.86490).

Future possible improvements: an even better result could be obtained by considering the time as a predictor. Other idea could be to look at the number of genres assigned to each movie. It is possible that movies with more genres assigned had higher ratings than movies with just one genre assigned. But I did not check this.