Movielens

Massimo Palme

30/5/2020

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.

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.

Code:

first, create the edx and validation sets

################################  
# 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/  
# 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

Then, I defined train and test sets

#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)

Then I defined the RMSE function

#define RMSE function  
RMSE <- function(true\_ratings, predicted\_ratings){  
 sqrt(mean((true\_ratings - predicted\_ratings)^2))  
}

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

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

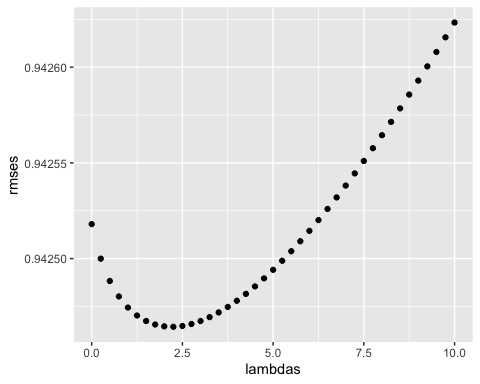
#REGULARIZATION  
# define a cross-validation set for movies  
set.seed(1, sample.kind="Rounding")

## 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 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)  
  
#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

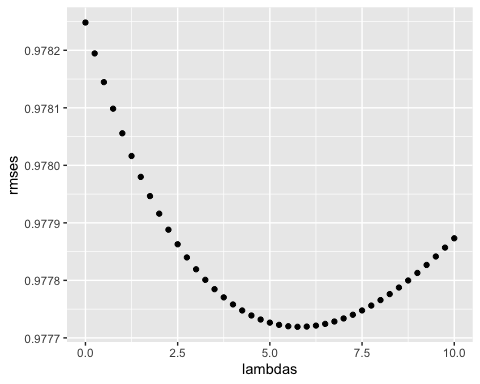
#now repeat for users  
  
  
# cross-validation set   
set.seed(1, sample.kind="Rounding")

## 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

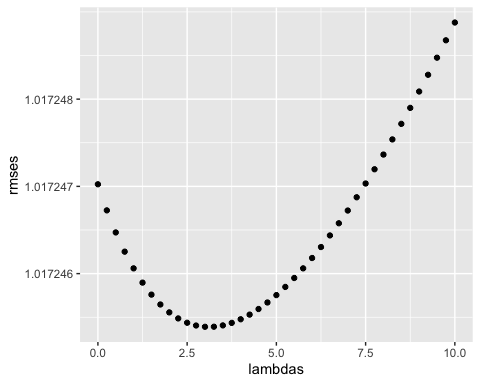
#now for genres  
# cross-validation set   
set.seed(1, sample.kind="Rounding")

## 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,]  
  
# 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.

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.

#to select the best lambda values, perform k-fold cross-validation with k=5  
#first for movie   
set.seed(1, sample.kind="Rounding")

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

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")

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

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.

#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

Results demonstrate that:

1. the movie effect is stronger than user and genre effect. User effect is stronger than genre. Combined model leads to a good result.
2. Regularization permits to obtain a value close to the requested value by the exercise.
3. 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 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.