MovieLens project

Andrea Roetti

June 11, 2020

Contents

Introduction

The MovieLens project goal is to build up a prediction system for movie ratings based on a historical series of records. The dataset we will use as base to create our model is called **edx**, while the dataset where we will measure the efficiency of our model, as if we did not know the given ratings, is called **validation**. In order to get started, we first load some libraries that will be useful for the analysis.

```r
library(tidyverse)
library(lattice)
library(caret)
library(lubridate)
```

We therefore read the two datasets that we have previously already downloaded as *.rds* files and saved locally for reasons of size: a file bigger than 100MB is not possible to push to Github, hence not possible to be kept in the project folders synchronised with it. We know that using complete paths rather than relative ones is not recommended, in fact below we also provide, as comment, the path to the datasets as they were in the “data” folder within our working directory.

```r
edx <- readRDS("C:/Users/roetti/Documents/MovieLens/edx.rds")
# edx <- readRDS("data/edx.rds")
validation <- readRDS("C:/Users/roetti/Documents/MovieLens/validation.rds")
# validation <- readRDS("data/validation.rds")
```

To get a first idea of the dataset edx, we here below show the first six records.

```
##   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
```

We see that the dataset has six columns: userId, movieId, rating, timestamp, title and genres. Some of the preliminary and exploratory data analysis include understanding the number of rows, the number of different users, the number of different movies and the average rating.

```r
nrow(edx)
```

```
## [1] 9000055
```

```r
n_distinct(edx$userId)
```

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

```r
n_distinct(edx$movieId)
```

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

```r
mean(edx$rating)
```

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

In our prediction model, the ultimate goal is to best predict the ratings given in the validation dataset, which has the same structure as edx. Our criteria to establish how close our predictions fall to the real ratings is the Root Mean Squared Error, or RMSE, a quantity similar to standard deviation: the lower its amount, the more accurate is the prediction. We therefore define here below the function that calculates the RMSE between two strings containing the same number of evaluations (true vs. predicted). We will try, with our predictions, to minimize the value of this function.

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

Analysis (part 1)

In this part we summarise the key findings and the approaches used in the *Data Science: Machine Learning* course, section 6 “Model Fitting and Recommendation System”. We first create a partition of the edx dataset, based on the rating column, into a train set (80% of records) and a test set (20% of records). For the sake of reproducibility, we set the seed to 1 before doing it.

```r
set.seed(1, sample.kind = "Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1,
                                  p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]
```

We also want to make sure that we don’t include users and movies in the test set that do not appear in the training set, therefore we run the following code.

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

Knowing that the average rating is the constant that minimizes the RMSE, we take, as a first benchmark of our model built on the train set, the average rating in the train set, assigned to *mu*. We then test its effectiveness on the test set and report the result in a table to keep track of our following improvements.

```r
mu <- mean(train_set$rating)
first_rmse <- RMSE(test_set$rating, mu)
rmse_results <- tibble(method = "Just the average", RMSE = first_rmse)
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.059904

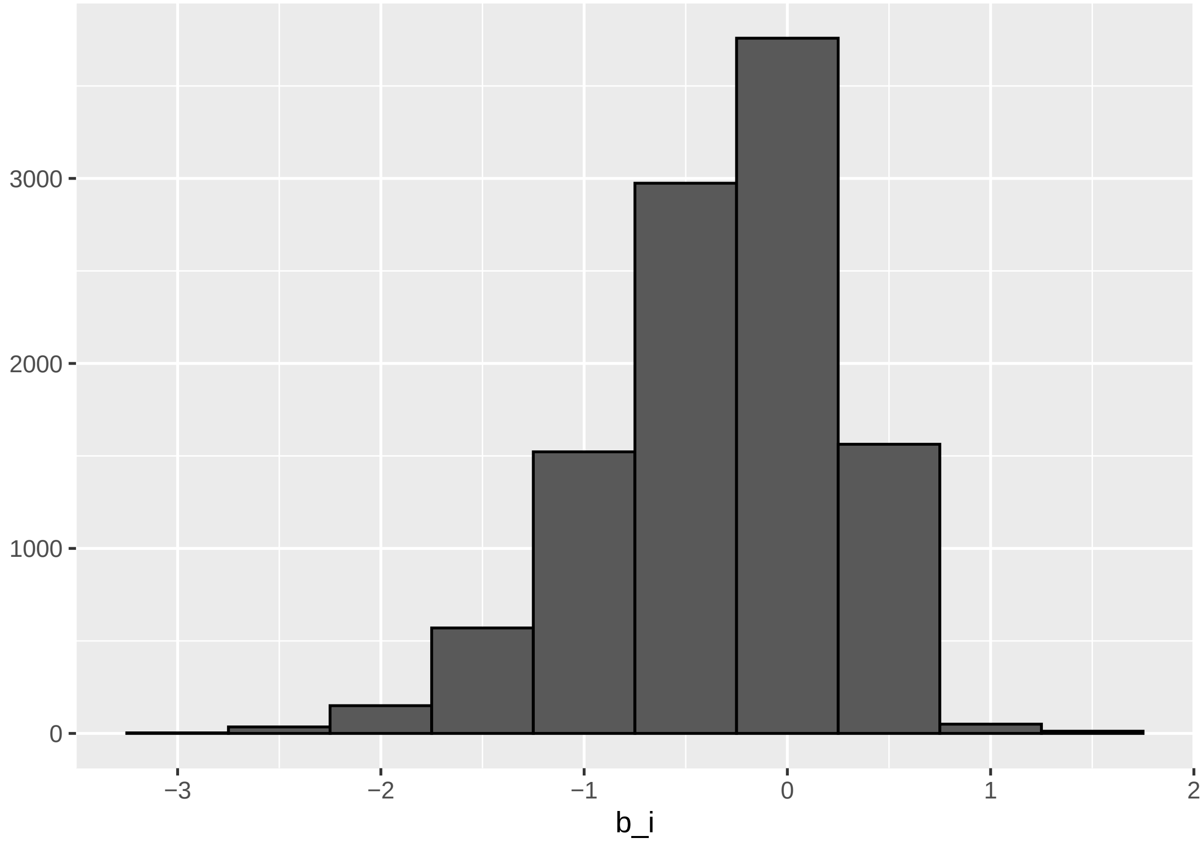
The result, almost 1.06 is quite high.

Movie Effect

To improve the result, we first think that different movies have different average ratings. We call this a bias, namely a factor, the movie ID, that impacts our predicted rating. To check our guess, we subtract the average rating from the movie average, so that our *b_i*, bias factors due to the movie ID, are positive only if the movie has an average rating above the overall average *mu*, negative if below.

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

Then we plot an histogram of the *b_i* distribution.



Our guess is confirmed, we therefore include the factors *b_i* in our prediction system and test again the effectiveness on the test set. We call this predictions *second_pred*, since the first ones simply coincided with *mu* and we report the new RMSE in the same table as before.

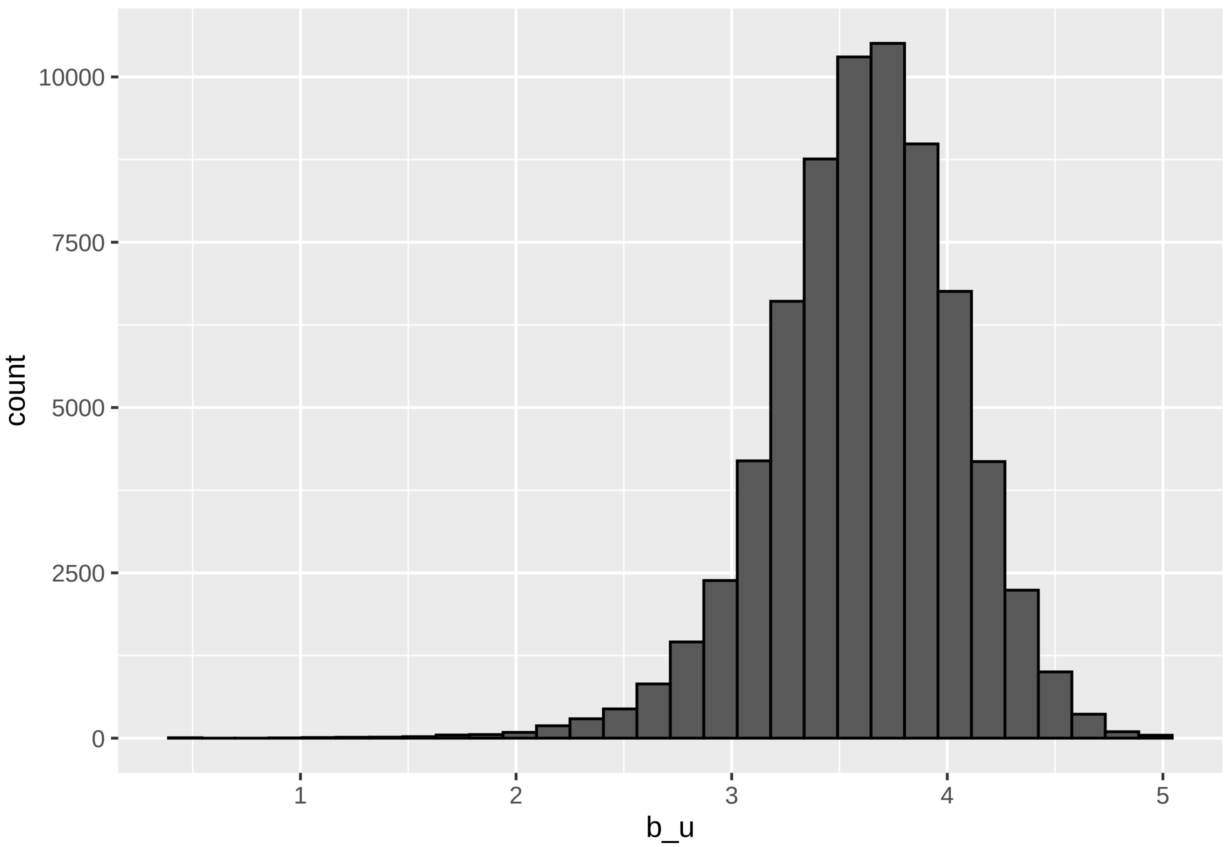
```
second_pred <- test_set %>%
     left_join(b_i, by='movieId') %>%
     mutate(pred = mu + b_i) %>% .$pred
second_rmse <- RMSE(test_set$rating, second_pred)
rmse_results <- bind_rows(rmse_results,
                tibble(method = "Movie Effect Model",
                       RMSE = second_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0599043
Movie Effect Model	0.9437429

The result has quite improved.

User Effect

We suspect that a possible bias could come also from the userId, since different users reasonably give different average ratings. We try to verify this by plotting an histogram of the average ratings for users that have rated at least 100 movies.



The plot confirms our suspect, therefore we include the bias factors *b_u* linked to the userId in our prediction system. As before, we calculate the *b_u* from the training set, we test the model and we update the table with the new RMSE.

```
b_u <- train_set %>%
     left_join(b_i, by = "movieId") %>%
     group_by(userId) %>%
     summarize(b_u = sum(rating - b_i - mu)/(n()))
third_pred <- test_set %>%
     left_join(b_i, by = "movieId") %>%
     left_join(b_u, by = "userId") %>%
     mutate(pred = mu + b_i + b_u) %>% .$pred
third_rmse <- RMSE(test_set$rating, third_pred)
rmse_results <- bind_rows(rmse_results,
                tibble(method = "Movie + User Effects Model",
                       RMSE = third_rmse))
rmse_results %>% knitr::kable()
```

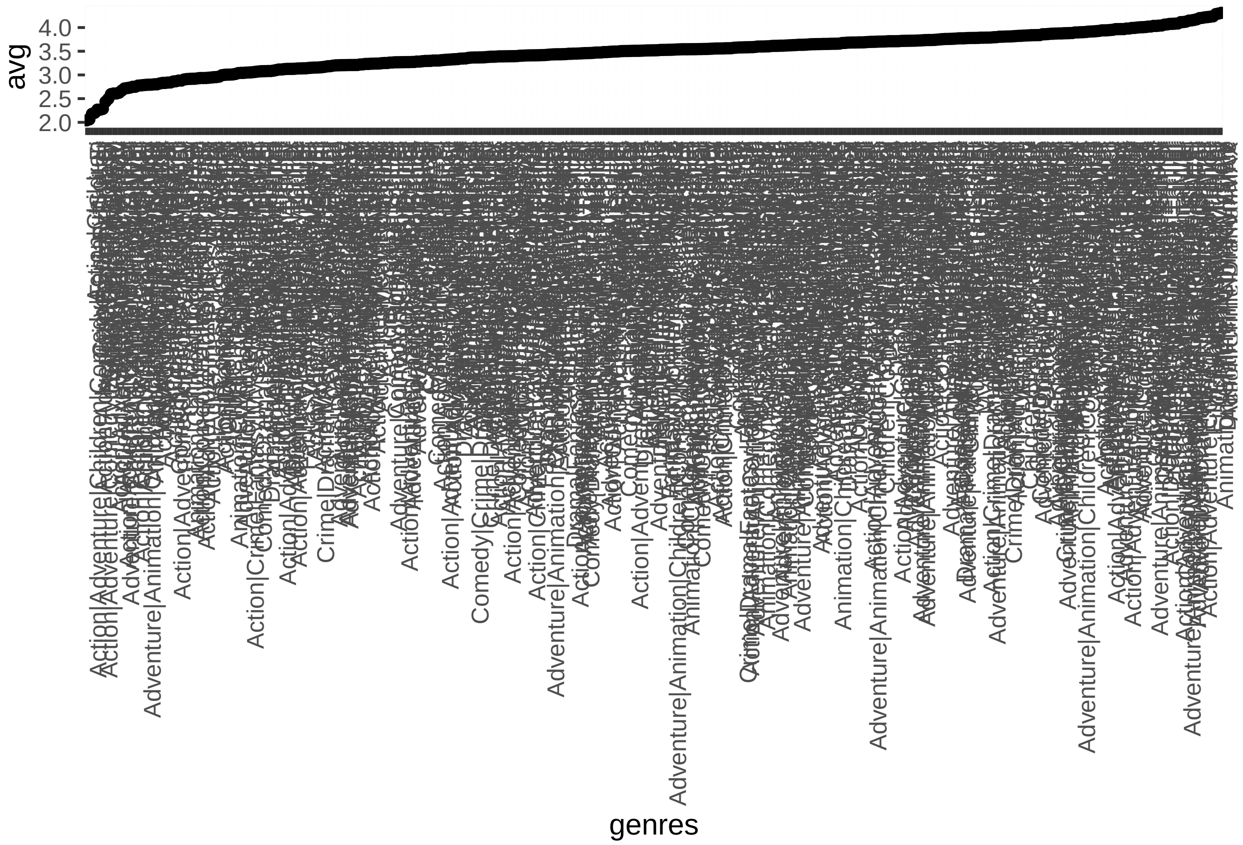
method	RMSE
Just the average	1.0599043
Movie Effect Model	0.9437429
Movie + User Effects Model	0.8659320

Section 6 of the *Data Science: Machine Learning* course continues by mastering the model obtained so far by using a regularization process. Before doing this, we first try to find out other possible biases that can increase even more the accuracy of our predictions.

Analysis (part 2)

Genres Effect

We think that the genres also may represent a bias, since some genres could, on average, be more appreciated than others. We therefore make a plot, to understand if this is true.



Even if for space limitations the genres are not well readable, it looks like also the genres have an impact on the rating. We include the bias *b_g*, calculated on the train set, and then applied on the test set to verify the improvements, if any. We also keep on updating our table of results with the new RMSE, the fourth one.

```r
b_g <- train_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    group_by(genres) %>%
    summarise(b_g = sum(rating - b_i - b_u - mu)/(n()))
fourth_pred <- test_set %>%
    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) %>% .$pred
fourth_rmse <- RMSE(test_set$rating, fourth_pred)
rmse_results <- bind_rows(rmse_results,
                tibble(method = "Movie + User + Genres Effects Model",
                       RMSE = fourth_rmse))
```

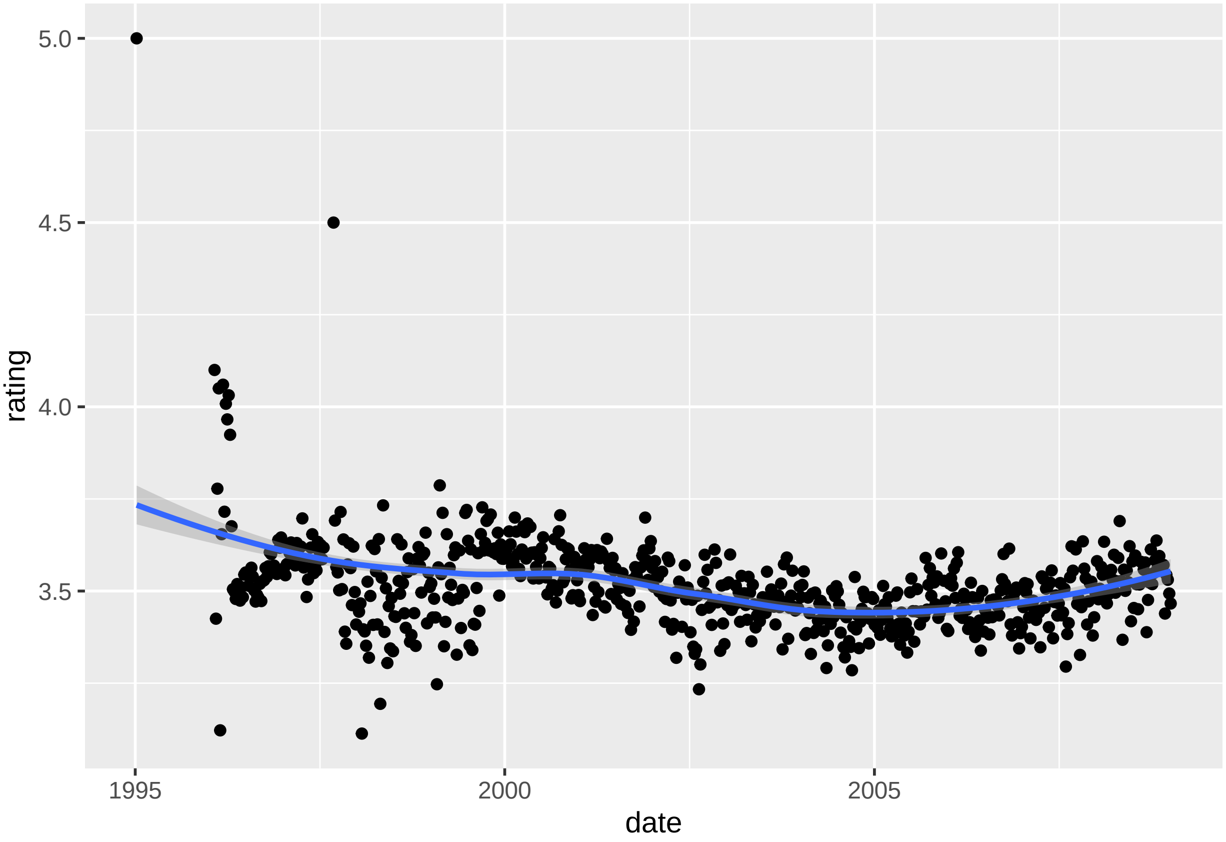
```
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0599043
Movie Effect Model	0.9437429
Movie + User Effects Model	0.8659320
Movie + User + Genres Effects Model	0.8655941

The result has slightly but further improved.

Time Effect

Trying to exploit somehow the timestamp column in the edx dataset, we try to check whether a time effect exists or not. We translate the timestamp into date and then we round the dates by week: the plot we obtain is the following.



We can notice that there is a light but perceivable time effect: we will call this bias *b_d*, since it is due to the date a rating is given, and will include it in our prediction model. As done before, we calculate the factors *b_d* on the training set, we apply the model to the test set and report the RMSE obtained, the fifth one, in the table.

```
b_d <- train_set %>%
    mutate(date=round_date(as_datetime(timestamp), unit = "week")) %>%
    left_join(b_i,by="movieId") %>%
    left_join(b_u,by="userId") %>%
    left_join(b_g,by="genres") %>%
    group_by(date) %>%
    summarise(b_d=sum(rating - b_i - b_u - b_g - mu)/(n()))
fifth_pred <- test_set %>%
    mutate(date=round_date(as_datetime(timestamp), unit = "week")) %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    left_join(b_d, by = "date") %>%
    mutate(pred = mu + b_i + b_u + b_g + b_d) %>% .$pred
fifth_rmse <- RMSE(test_set$rating, fifth_pred)
rmse_results <- bind_rows(rmse_results,
                tibble(method = "Movie + User + Genres +Time Effects Model",
                       RMSE = fifth_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0599043
Movie Effect Model	0.9437429
Movie + User Effects Model	0.8659320
Movie + User + Genres Effects Model	0.8655941
Movie + User + Genres +Time Effects Model	0.8654875

Regularization

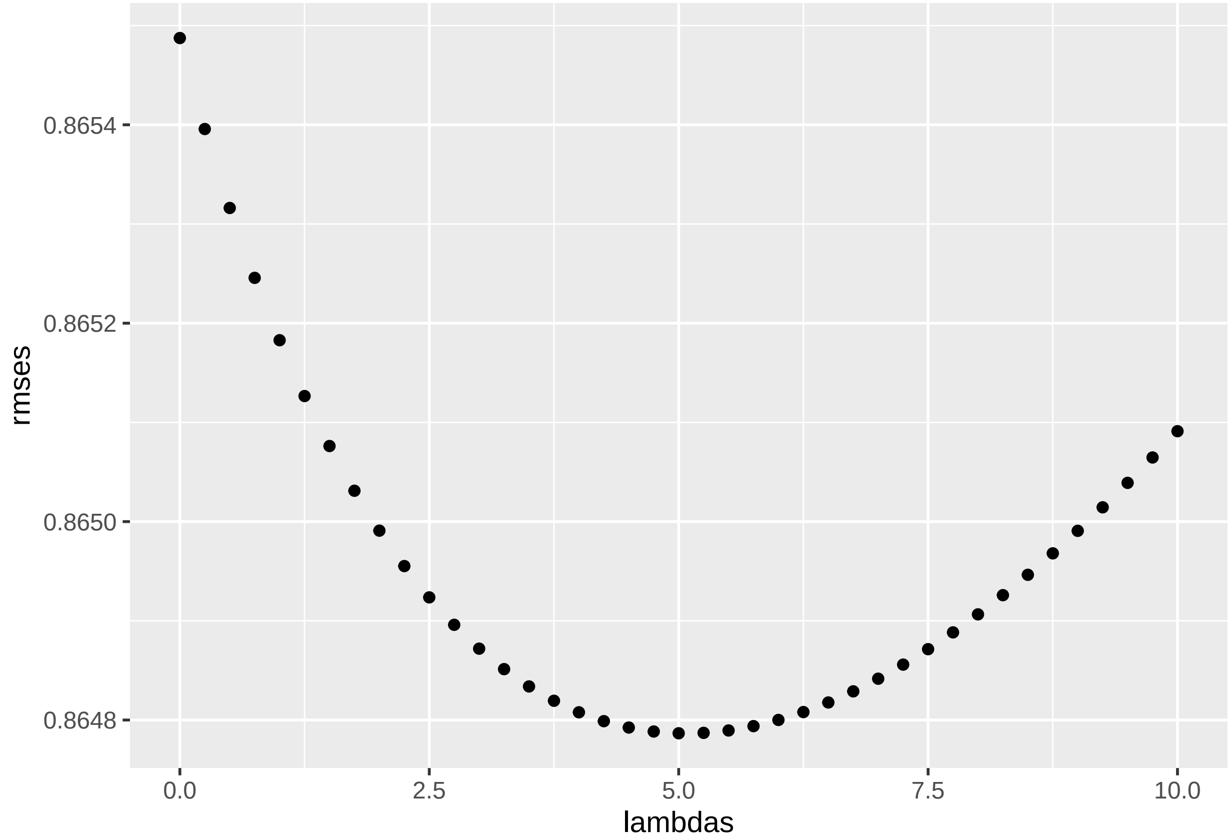
The result improved very slightly, but we can perhaps do better using regularization. We therefore define a sequence of lambdas from 0 to 10, we train our model on each lambda using the train set and we check the effectiveness of each lambda on the test set calculating the RMSE. We finally use the best performing one, updating the table. The following code, if run, may take several minutes.

```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
    mu <- mean(train_set$rating)
    b_i <- train_set %>%
        group_by(movieId) %>%
        summarize(b_i = sum(rating - mu)/(n()+l))
    b_u <- train_set %>%
        left_join(b_i, by = "movieId") %>%
        group_by(userId) %>%
        summarize(b_u = sum(rating - b_i - mu)/(n()+l))
    b_g <- train_set %>%
        left_join(b_i, by = "movieId") %>%
        left_join(b_u, by = "userId") %>%
        group_by(genres) %>%
        summarise(b_g = sum(rating - b_i - b_u - mu)/(n()+l))
    b_d <- train_set %>%
        mutate(date=round_date(as_datetime(timestamp), unit = "week")) %>%
        left_join(b_i,by="movieId") %>%
        left_join(b_u,by="userId") %>%
        left_join(b_g,by="genres") %>%
        group_by(date) %>%
        summarise(b_d=sum(rating - b_i - b_u - b_g - mu)/(n()+l))
    pred <- test_set %>%
        mutate(date=round_date(as_datetime(timestamp), unit = "week")) %>%
        left_join(b_i, by = "movieId") %>%
        left_join(b_u, by = "userId") %>%
        left_join(b_g, by = "genres") %>%
        left_join(b_d, by = "date") %>%
        mutate(pred = mu + b_i + b_u + b_g + b_d) %>% .$pred
    return(RMSE(test_set$rating, pred))})
rmse_results <- bind_rows(rmse_results,
                tibble(method="Regularized Movie + User + Genres +Time Effects Model",
                           RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0599043
Movie Effect Model	0.9437429
Movie + User Effects Model	0.8659320
Movie + User + Genres Effects Model	0.8655941
Movie + User + Genres +Time Effects Model	0.8654875
Regularized Movie + User + Genres +Time Effects Model	0.8647867

The result is pretty satisfying. Here below we can see a plot of lambdas' performances, we then select the best one and we show it.

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



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

```
## [1] 5
```

Results

Once established that our model is complete, we are now training it (namely, defining *mu* and the coefficients *b*) on the whole edx dataset before testing it on the validation set. This way, we use the 100% of dataset at our disposition instead of only the 80% represented by the train set, hoping to further improve the performance.

```r
mu <- mean(edx$rating)
b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+lambda))
b_u <- edx %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))
b_g <- edx %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    group_by(genres) %>%
    summarise(b_g = sum(rating - b_i - b_u - mu)/(n()+lambda))
b_d <- edx %>%
    mutate(date=round_date(as_datetime(timestamp), unit = "week")) %>%
    left_join(b_i,by="movieId") %>%
    left_join(b_u,by="userId") %>%
    left_join(b_g,by="genres") %>%
    group_by(date) %>%
    summarise(b_d=sum(rating - b_i - b_u - b_g - mu)/(n()+lambda))
```

We are now ready to finally test our model on the validation test: we make our prediction and we compute the RMSE. In order to do this, we are going to replace the NAs in our prediction (coming, for instance, because of unknown movies or unknown users) with the average rating.

```r
predicted_ratings <- validation %>%
    mutate(date=round_date(as_datetime(timestamp), unit = "week")) %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    left_join(b_d, by = "date") %>%
    mutate(pred = mu + b_i + b_u + b_g + b_d) %>%
    .$pred %>% replace_na(mu)
RMSE(validation$rating, predicted_ratings)
```

```
## [1] 0.8643062
```

The final RMSE is quite good and even better than the last one obtained on our test set taken as a subset of edx: apparently this confirms that the higher number of records used to train the model has permitted a better performance.

Conclusion

In our strategy, in building the model, we have only used the intuition in order to find out possible biases in the rating system and apply them to our predictions. In the chapter “Analysis (part 1)” we have repeated what was explained in the Section 6 of the *Data Science: Machine Learning*, including in our model the biases related to different movies and different users. In the chapter “Analysis (part 2)” we have then gone beyond, introducing biases related to the genres and the date of the rating. Finally, in the chapter “Results”, before applying our model to the validation set, we have trained our model on the whole edx dataset.

There is certainly room for further improvements, for example using matrix factorization and principal components analysis (PCA) to the residuals of our model. This operation is able to find out factors not always related to intuitive biases, such as subsets of users having similar rating patterns or group of movies of different genres having similar rating patterns. To do this, we would first transform the dataset into a matrix where entrances are represented by userId (rows) and movieId (columns). Since this matrix would be very huge and sparse (namely with a lot of NAs), we would filter only users and movies with a minimum number of ratings to make it manageable and light enough to not make R crash.