edX | Data Science: Capstone - MovieLens Project

Kai Notté

04.08.2019

Contents

1 Introduction

The following documentation describes my solution for the edX Course “PH125.9x Data Science: Capstone” by HarvardX.

The target of this project is to build a recommendation system based on the MovieLens dataset. This dataset includes various user ratings for more than 10,000 movies from the last years. The task is to predict user ratings with machine learning algorhythm which we have learned in previous courses.

To receive most credit points, the course instruction requires a final root mean square error of lower or equal 0.8649.

This documentation will guide through the whole building process: The first chapter provides information about the fundamental data preparation and analysis. In addition, the second chapter contains the method and explanation of the used algorhythm. The third chapter summarizes these findings and provides the final calculation. Finally, the fourth chapter concludes the report and provides some insights into known limitation and possible developtments for the future.

2 Data Creation and Preparation

This chapter includes data preparation and analysis to provide data insights. At first, the raw data will be loaded as provided in the course instruction. This data set will be analysed to find possible issues for the usage in machine learning algorhythm. In addition, the edx data set will be split into train and test data set to follow course’s guidelines. Furthermore, this chapter also provides information about general used functions and libraries.

2.1 Data Creation

The edX HarvardX course PH125.9X provided the following code to create the edx and validation dataset.

```r
# Note: this process could take a couple of minutes

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

# MovieLens 10M dataset: https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
    col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::",
    3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
    title = as.character(title), genres = as.character(genres))

movielens <- left_join(ratings, movies, by = "movieId")

# Validation set will be 10% of MovieLens data

set.seed(1, sample.kind = "Rounding")
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1,
    list = FALSE)
edx <- movielens[-test_index, ]
temp <- movielens[test_index, ]

# Make sure userId and movieId in validation set are also in edx set

validation <- temp %>% semi_join(edx, by = "movieId") %>% semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set

removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

2.2 Data Set Preparation and Analysing

The provided edx dataset is not in a tidy format as you can see below:

```r
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
```

Moreover some general information:

```r
edx %>% summarize(ratings = n(), unique_userIds = n_distinct(userId), unique_movieIds = n_distinct(movi
    )
```

```
##   ratings unique_userIds unique_movieIds
## 1 9000055          69878           10677
```

The edx dataset provided the following information

- `userId` contains unique user identifier.
- `movieId` contains unique movie identifier.
- `rating` represents user's rating for a movie.
- `timestamp` shows the date and time of user's rating in timestamp-format.
- `title` includes the title as well as the publishing year of the rated movie.
- `genres` includes all movie related genres, seperated with the symbol "|".

This leads to the assumption, that some cleaning may be useful for the values title and genres. This cleaning is only required for the edx data set. All movies from the validation data set are also included in the edx data set which results that the extracted information can be joined for both data sets based on the edx data set's extraction. Due to time limitation, this cleaning just will be done if necessary and required by the used algorhythm.

2.3 Create Test and Train Data Sets

According to the course instruction, we are allowed to use the data set `edx` for any training und testing. Moreover, the course instruction does not allow to use the data set `validation` in any train and test phase except for the final validation. Therefore, an additional split of the data set `edx` is required.

Following previous courses, the data set `train` will contain about 80% of the available data. This data set will be used to train any model. To test and evaluate these models, the data set `test` with about 20% of the available data will be used.

Below standing codes split the data set `edx` into both data sets `train` and `test`.

```r
# Split edx data set into test data set (20% of edx data set) and train data
# set (80% of edx data set)
set.seed(1, sample.kind = "Rounding")
```

```r
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
train <- edx[-test_index, ]
test_temp <- edx[test_index, ]

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

# Add rows removed from test data set back into train data set
test_removed <- anti_join(test_temp, test)
train <- rbind(train, test_removed)
```

2.4 Used Libraries and Functions

This section introduces used libraries and general used function.

2.4.1 Used libraries

The course instruction already used the following libraries:

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

In fact, the final algorhythm as well as the analysation do not use any further libraries.

2.4.2 General Used Function

According to the coding method DRY ("Don't repeat yourself"), this chapter introduces function which will be used in this documentation and the final algorhythm.

2.4.2.1 RMSE

The root mean squared error can be calculate with the following function in the whole project.

```r
RMSE <- function(prediction, validation) {
    sqrt(mean((validation - prediction)^2))
}
```

3 Method

The method follows the course instruction from previous courses and Chapter 34 in R. Irizarry's dsbook.

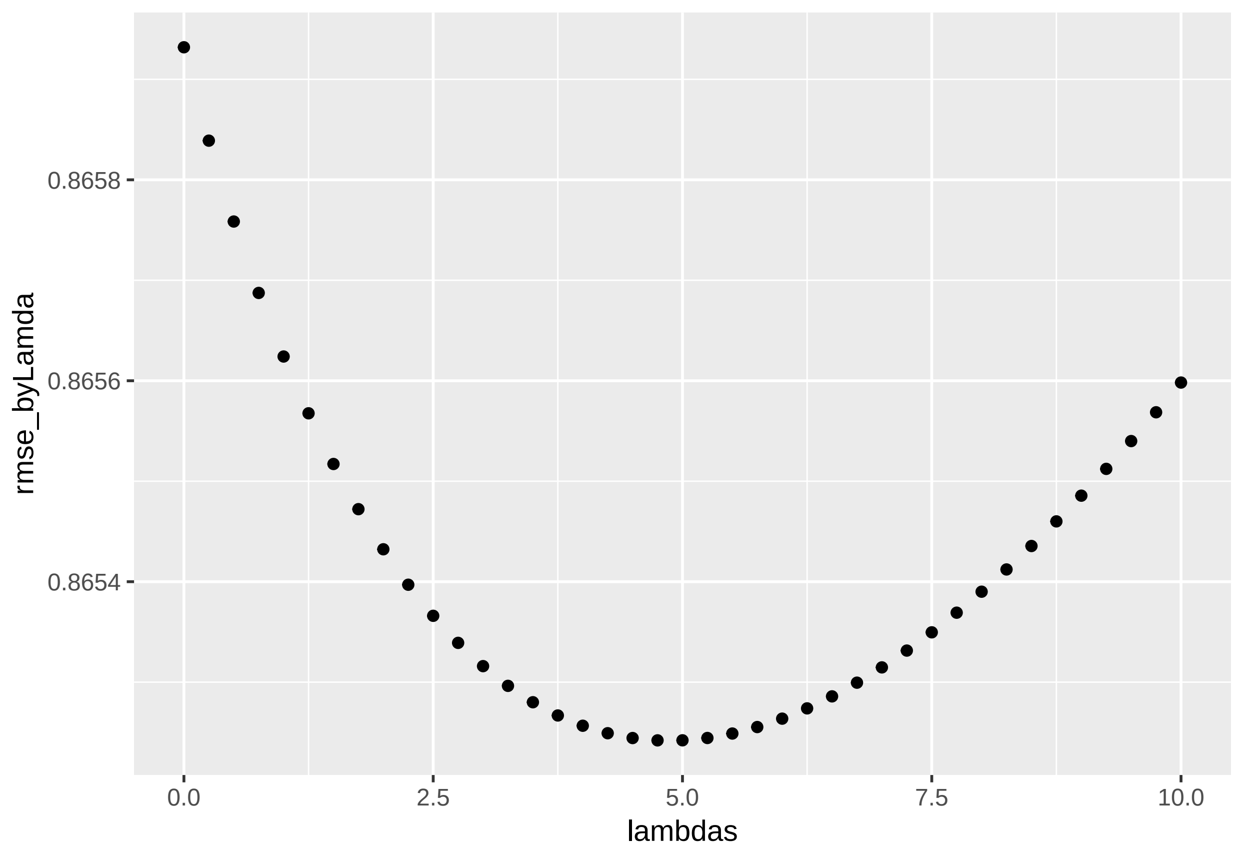
According to the final findings in this Chapter, the used method used movie and user bias in combination with penalized the least. Using this method, some assumptions are required:

- Using the movie bias assumes, that some movies received in average a better rating than others. Therefore, the average of the difference between the average rating `mu` and each movie rating is representing each movie bias `b_movie`.

- The used user bias is like the movie bias assuming, that some users tend to rate movies higher than other users to. Therefore, the average of the difference between the average rating `mu` minus the movie bias `b_movie` and each user rating reults in each user bias `b_user`.
- The regulation by penalizing the last squares adds lambda as a tuning parameter to each bias calculation. Using this lambda reduced the weight of ratings for movies and/or users with just a few ratings. Therefor each bias represents the weighted average by dividing the sum through lambda plus the numbers of ratings instead of the standard average.

The following code calculates the average `mu` and both biases `b_movie` and `b_user` for possible `lambdas` between 0 and 10 in a sequence of 0.25 to optimize `lambda` by reducing the RMSE.

```r
# Calculate mu: average rating of all movies
mu <- mean(train$rating)

# Define the sequenz of lambdas for the regulazation
lambdas <- seq(0, 10, 0.25)

# Iterate for each lambda: Calculate biases and evaluate prediction
rmse_byLamda <- sapply(lambdas, function(lambda) {

    # Calculate the movie bias b_movie on the train data set
    b_movie <- train %>% group_by(movieId) %>% summarize(b_movie = sum(rating -
        mu)/(n() + lambda))

    # Calculate the user bias b_user_l on the train data set
    b_user <- train %>% left_join(b_movie, by = "movieId") %>% group_by(userId) %>%
        summarize(b_user = sum(rating - mu - b_movie)/(n() + lambda))

    # Calculate the predicted values for the test data set
    prediction <- test %>% left_join(b_movie, by = "movieId") %>% left_join(b_user,
        by = "userId") %>% mutate(prediction = mu + b_movie + b_user) %>% pull(prediction)

    # Calculate the RMSE for each lambda
    return(RMSE(prediction, test$rating))
})

# Show RMSE according to lamda
qplot(lambdas, rmse_byLamda)
```



```
# Identify the best lambda with the lowest RMSE
lambda <- lambdas[which.min(rmse_byLamda)]
lambda
```

```
## [1] 4.75
```

```
rmse <- min(rmse_byLamda)
rmse
```

```
## [1] 0.8652421
```

Above codes follows the instructions without any additional modification except the data sets. The lowest scored RMSE is about 0.8652 with the tuning parameter `lambda` of 4.75. This result meets course's requirements, even if it does not within the limits for the best result.

4 Results

The previous chapter results in a RMSE of about 0.8652 with a tuning parameter `lambda` of 4.75 on the edx data set only. For the final result, the whole edx data set will be used to calculate both biases `b_movie` and `b_user` but the tuning parameter `lambda` will stay 4.75. Finally, these biases will be used to calculate the predicted ratings for the validation set which will be evaluate afterwards.

```r
# Calculate the movie bias b_movie on the edx data set with lambda = 4.75
b_movie <- edx %>% group_by(movieId) %>% summarize(b_movie = sum(rating - mu)/(n() +
    lambda))

# Calculate the user bias b_user on the edx data set with lambda = 4.75
b_user <- edx %>% left_join(b_movie, by = "movieId") %>% group_by(userId) %>%
    summarize(b_user = sum(rating - mu - b_movie)/(n() + lambda))

# Calculate the predicted values for the validation data set
prediction <- validation %>% left_join(b_movie, by = "movieId") %>% left_join(b_user,
    by = "userId") %>% mutate(prediction = mu + b_movie + b_user) %>% pull(prediction)

# Calculate the final RMSE
RMSE(prediction, validation$rating)
```

```
## [1] 0.8648201
```

The finally calculated root mean square error is 0.8648 which is close but lower than the target of 0.8649 and lead to the best classification in course's results.

The used method is easy to use and time saving because it is working without any complex machine learning algorhythm like nearest neighbor or similar. Therefor it is usable on machines even with a small processor and RAM even with a big amount of available data lines.

5 Conclusion

The present report introduced one solution to predict user's ratings on movies based on the MovieLens data set. The target was to predict these ratings on a validation data set with a root mean square error equal or below 0.8649. By using movie and user biases which are regularized by penalizing the least squares with a choosen tuning parameter `lambda` of 4.75 the final calculated RMSE results in 0.8648.

The model is designed for fast and time saving calculation but there are limitations on it. The calculations are limited to the existing data set and therefore to past user's ratings. This results in a lack of information to users with no or limited ratings, like new users. In addition, the model is vulnerable to movies with no or limited ratings.

To close movie's lack of information, future models could user similarities of movies as genres or actors (which are available in the MovieLens project).

On user's side, personas like age, sex or given information like interests could provide additional information to identify similar profiles and predict ratings more accurate.