Data Science 'MovieLens' Capstone Project

András Gelencsér

10 November 2019

Overview

This document describes the result of the capstone project for the Harvard Data Science course based on the MovieLens dataset.

The goal of the project is to develop a movie recommendation algorithm based on data science and machine learning techniques to predict the movie ranking by users based on historical movie rating data. The performnce of the algorithm is messed with the root mean standard error (RMSE) value.

The used dataset conatains movie rating data set collected by GroupLens Research. The full MoviLens dataset include more then 20 millions movie ratings for more than 27,000 movies by 138,000 users. The rated movies cover different time periods. The data sets are avaiable on the https://grouplens.org/datasets/movielens/ url.

Because of the size the full dataset only a subset of the data is used for the project. The subset contains 1,000,000 rating entries for the training purpose (training set) and 100,000 rating entries for evaulation of the algorithm performance (test set).

The following steps were done during the project for developing the algorithm:

1. Download the data from the movielens page
2. Analyse the data structure
3. Define the recommendation model based on the result the data analysis
4. Implement and train the algortihm based on the train set
5. Review the algorithm based on the test set

The final recommendation algorithm achied an RMSE value of 0.8000 on the test set.

Methods & analysis

At first we need to prepare the dataset for the analysis. The analysis is based on the reduced movielens dataset including approximatelly 1,000,000 rating data. The following R code downloads the data from the http://grouplens.org site and extract the needed data. To avoid unecessary data trafic, the data will be downloaded only if not done yet.

```
##########################################################
# Create edx set, validation set, and submission file
##########################################################

# 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 ---------------------------------------------------------------------------
```

```
## v ggplot2 3.2.0     v purrr   0.3.2
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   0.8.3     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

```
## -- 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
```

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

#flag for marking file download
file_downloaded <- FALSE

#download the zip file only if not done yet
if (!dir.exists("ml-10M100K")){
     dl <- tempfile()
     download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)
     unzip(dl, "ml-10M100K/ratings.dat")
     unzip(dl, "ml-10M100K/movies.dat")

     #mark the download in the flag
     file_downloaded <- TRUE
}

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

movies <- str_split_fixed(readLines(file("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)
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)
```

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

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

#remove temp file if download done
if (file_downloaded) rm(dl)

#remove temporary variables
rm(ratings, movies, test_index, temp, movielens, removed)
```

Avaiable Data

For the analysis we have data about 10,000 movies and 70,000 users. The movies have the following known data:

- Movie ID
- Movie title
- Introduction year of the movie (can be extraxted from the title field)
- Genres

The ratings have the following known data:

- UserId
- Rating timestamp (from this data we can extract the year and month of the rating)
- Rating value (the rating value is a number between 0.5 and 5)

Unfortunatelly we don‘t have any information (age, gender, country, spoken languages) about the users For the development the recommedation algorithm we can use only the existing data.

Movies

Users

Model

Based on the movie and user data analysis we can develop a prediction model.

If we don’t have any information about the movie and the user in the test set (new movie and new user), we can use the overall avarage rating from the training set rating data. This predection will minimase the RSME value for unknown movies and user.

If we have rating data about the movie to predict we can assume, that the abriviation of the movie from the overall average shows the quality of the movie. We can use this information to tune the prediction value.

Implementation & result

```
#The value range for the possible lambda parameter
lambdas <- seq(0, 10, 0.25)

#Function to calculate the optimum lambda parameter for the movie bias useing two set of data (train an
rmses_for_movie_lambdas <- function(train, test){
```

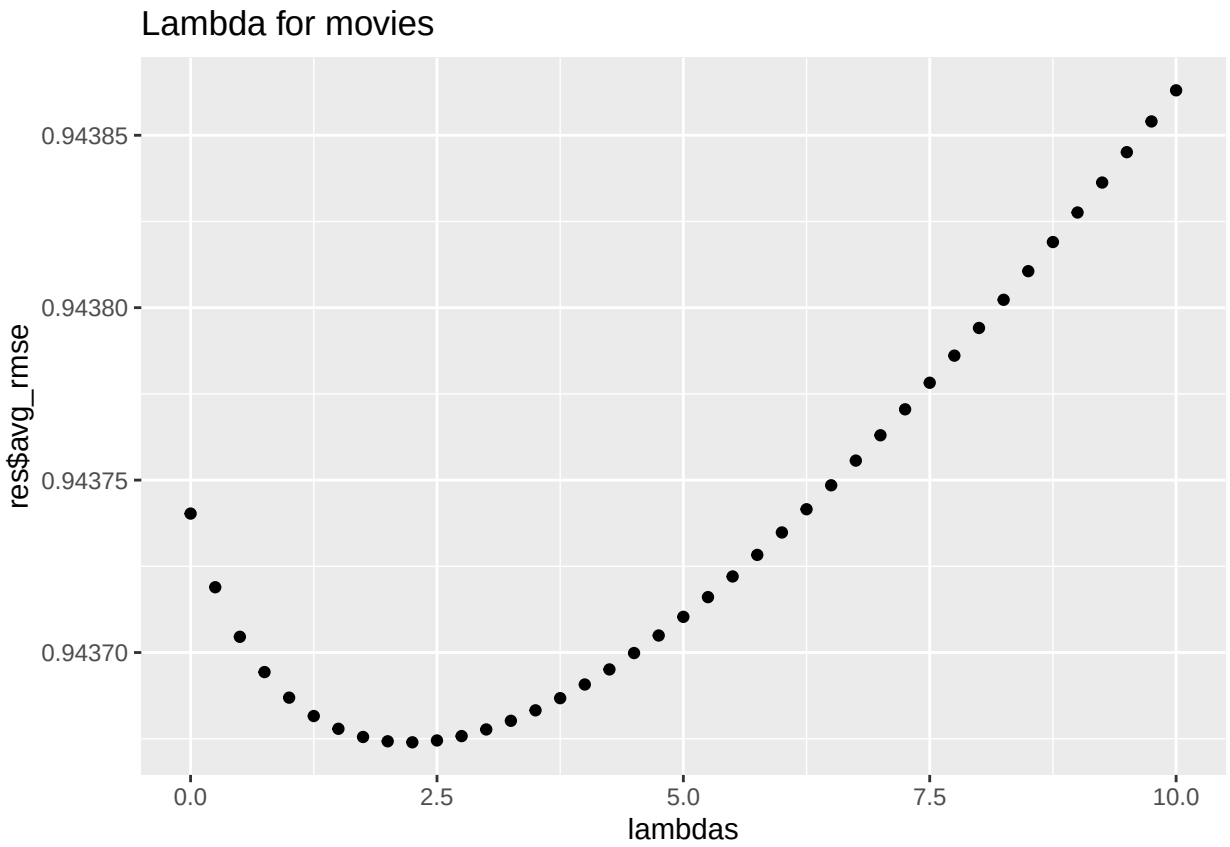
```
  #data frame variable for collecting the result data
  RMSE_all = data.frame(lambda = numeric(), rmse = numeric())
  #overall average of the rating in the train set
  avg <- mean(train$rating)
  #necessary sum and entry count values for the calculation (initialisation)
  movie_sum <- train %>% group_by(movieId) %>% summarise(s=sum(rating-avg), n_i=n())

  #loop for all lambdas in the provided range
  for (l in lambdas){
    #calculating predicted values for the test set using the actual lambda value from the range
    predicted_ratings <- test %>%
      left_join(movie_sum, by='movieId') %>%
      mutate(b_i = s / (n_i + l)) %>%
      mutate(pred = avg + b_i) %>%
      pull(pred)

    #calculate the RMSE value for the prediction
    rmse_akt =RMSE(predicted_ratings, test$rating)

    #add RMSE value to the result data frame
    RMSE_all <- bind_rows(RMSE_all, data.frame(lambda = l, rmse = rmse_akt))
  }
  #return the result data frame
  # the result will contain the RMSE value for all the lambda values from the provided lambda range
  return(RMSE_all)
}

#Function to calculate the optimum lambda parameter for the movie bias useing two set of data (train an
#This function needs the caluclated optimal lambda value for the movie bias
rmses_for_user_lambdas <- function(train, test, movie_lambda){

  #data frame variable for collecting the result data
  RMSE_all = data.frame(lambda = numeric(), rmse = numeric())
  #overall average of the rating in the train set
  avg <- mean(train$rating)
  #the movie bias values for the train set (bias calculated with the provieded optimal lambda value for
  b_movie <- train %>%
    group_by(movieId) %>%
    summarize(b_movie = sum(rating - avg) / (n() + movie_lambda))

  #necessary sum and entry count values for the calculation (initialisation)
  user_sum <- train %>%
    left_join(b_movie, by="movieId") %>%
    group_by(userId) %>%
    summarize(s = sum(rating - b_movie - avg), n_i= n())

  #loop for all lambdas in the provided range
  for (l in lambdas){
    #calculating predicted values for the test set using the actual lambda value from the range
    predicted_ratings <- test %>%
      left_join(b_movie, by='movieId') %>%
      left_join(user_sum, by='userId') %>%
      mutate(b_u = s / (n_i + l)) %>%
```

```
      mutate(pred = avg + b_movie + b_u) %>%
      pull(pred)

    #calculate the RMSE value for the prediction
    rmse_akt =RMSE(predicted_ratings, test$rating)
    #add RMSE value to the result data frame
    RMSE_all <- bind_rows(RMSE_all, data.frame(lambda = l, rmse = rmse_akt))
  }
  #return the result data frame
  # the result will contain the RMSE value for all the lambda values from the provided lambda range
  return(RMSE_all)
}


cross_validation_movie <- function(trainset, lambdas, cv_n){

  data_count = nrow(edx)
  RMSE_all = data.frame(lambda = numeric(), cv = numeric(), rmse = numeric())

  trainset_randomised <- trainset[sample(nrow(trainset)),]

  for (i in c(1:cv_n)){

    part_count = data_count / cv_n
    cat( "Run: ", i, " Test Part: ", str_pad((trunc((i-1) * part_count) + 1),width = 8,side = "left"),
    idx = c(   (trunc((i-1) * part_count) + 1) : trunc(i * part_count) )
    tmp = trainset_randomised[idx,]
    train = trainset_randomised[-idx,]
    test <- tmp %>%
      semi_join(train, by = "movieId") %>%
      semi_join(train, by = "userId")
    removed <- anti_join(tmp, test, by=c("movieId", "userId"))
    train <- rbind(train, removed)
    RMSE_Lambdas <- rmses_for_movie_lambdas(train, test)
    RMSE_all <- bind_rows(RMSE_all,RMSE_Lambdas %>% mutate(cv = i))
    cat("\n")
  }
  return(RMSE_all)
}

cross_validation_user <- function(trainset, lambdas, cv_n, movie_lambda){

  data_count = nrow(edx)
  RMSE_all = data.frame(lambda = numeric(), cv = numeric(), rmse = numeric())

  trainset_randomised <- trainset[sample(nrow(trainset)),]

  for (i in c(1:cv_n)){

    part_count = data_count / cv_n
    cat( "Run: ", i, " Test Part: ", str_pad((trunc((i-1) * part_count) + 1),width = 8,side = "left"),
    idx = c(   (trunc((i-1) * part_count) + 1) : trunc(i * part_count) )
    tmp = trainset_randomised[idx,]
```

```
    train = trainset_randomised[-idx,]
    test <- tmp %>%
      semi_join(train, by = "movieId") %>%
      semi_join(train, by = "userId")
    removed <- anti_join(tmp, test, by=c("movieId", "userId"))
    train <- rbind(train, removed)
    RMSE_Lambdas <- rmses_for_user_lambdas(train, test, movie_lambda)
    RMSE_all <- bind_rows(RMSE_all,RMSE_Lambdas %>% mutate(cv = i))
    cat("\n")
  }
  return(RMSE_all)
}

res <- cross_validation_movie(edx, lambdas,5)
```

```
## Run:  1  Test Part:         1   1800012
##
## Run:  2  Test Part:   1800013   3600024
##
## Run:  3  Test Part:   3600025   5400036
##
## Run:  4  Test Part:   5400037   7200048
##
## Run:  5  Test Part:   7200049   9000061
```

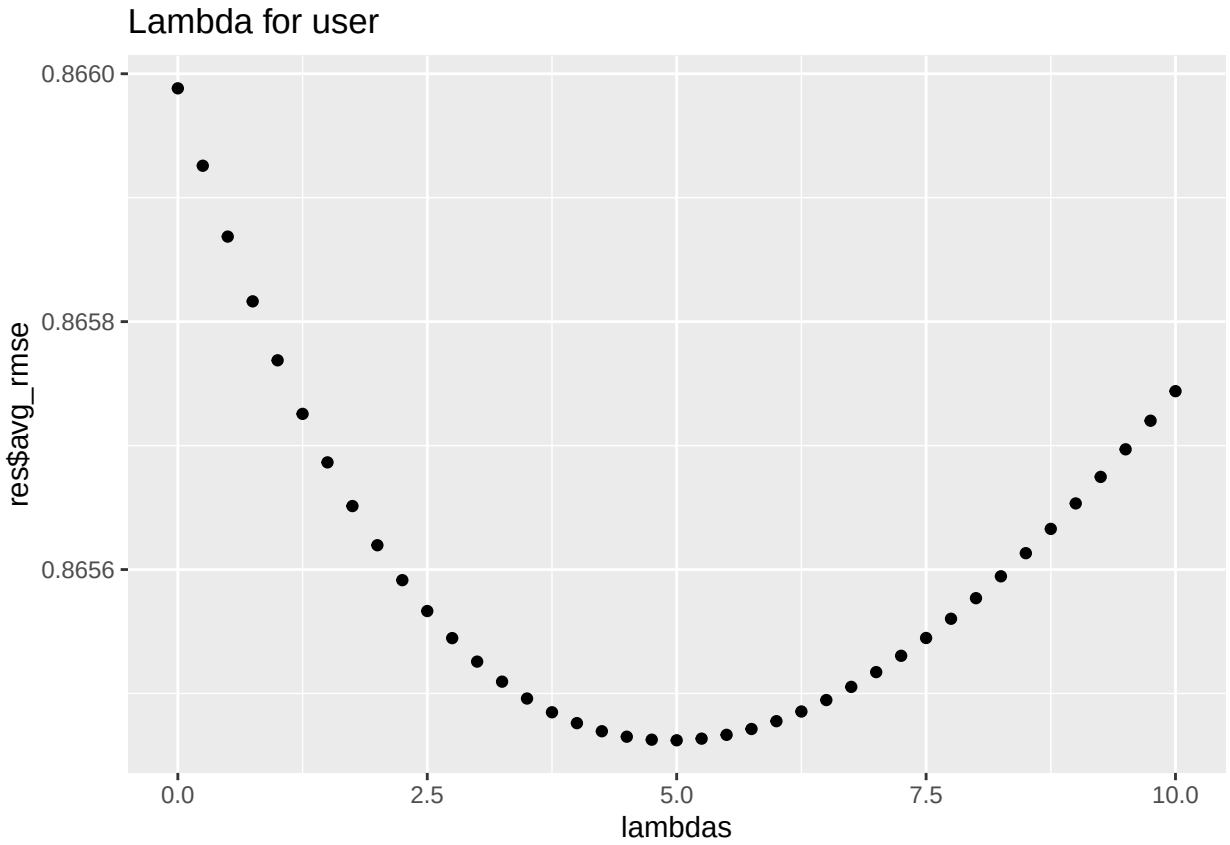
```
res <- res %>% group_by(lambda) %>% summarise(avg_rmse=mean(rmse))
qplot(main=c('Lambda for movies'), lambdas, res$avg_rmse)
```




```
movie_lambda = lambdas[which.min(res$avg_rmse)]

res <- cross_validation_user(edx, lambdas,5, movie_lambda)
```

```
## Run:  1  Test Part:         1   1800012
##
## Run:  2  Test Part:   1800013   3600024
##
## Run:  3  Test Part:   3600025   5400036
##
## Run:  4  Test Part:   5400037   7200048
##
## Run:  5  Test Part:   7200049   9000061
```

```
res <- res %>% group_by(lambda) %>% summarise(avg_rmse=mean(rmse))
qplot(main=c('Lambda for user'), lambdas, res$avg_rmse)
```




```
user_lambda = lambdas[which.min(res$avg_rmse)]
```

Regularisation

Conclusion