HardvardX EDX Capstone - The MovieLens Project

Thiago do Couto

10/06/2019

This stands for a capstone project for EDX HarvardX Data Science Certification Program.

0 - The objective of this project is to predict movie rating.

We’ll use a huge list of movies and its features to predict the movie rating.

1 - Load required libraries

```
# Load required libraries
library(tidyverse)
```

```
## ── Attaching packages ──────────────────────────────────────────
─ tidyverse 1.2.1 ──
```

```
## ✔ ggplot2 3.1.1     ✔ purrr   0.3.2
## ✔ tibble  2.1.3     ✔ dplyr   0.8.1
## ✔ tidyr   0.8.3     ✔ stringr 1.4.0
## ✔ readr   1.3.1     ✔ forcats 0.4.0
```

```
## ── Conflicts ──────────────────────────────────────────── tidy
verse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
```

```
library(caret)
```

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

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

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

2 - Load pre-processed RDS

The standard ML-10M file was freezing RStudio.

```
# Load pre-processed RDS
edx <- readRDS("edx.rds")
validation <- readRDS("validation.rds")
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
```

3 - Define workspace vars - the “ratings” target and the “genres” correlated variable

```
# Define working variables
rating_var <- edx$rating
genres_var <- edx$genres
```

4 - Quiz time

4.1 - How many rows and columns are there in the EDX dataset?

```
# 1 - How many rows and columns are there in the EDX dataset?
data_observations <- nrow(edx)
data_observations # 9000055
```

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

4.2 - How many zeros and three were given in the EDX dataset?

```
# 2 - How many zeros and three were given in the EDX dataset?
sum(edx$rating == 0) # 0
```

```
## [1] 0
```

4.3 - How many different movies are in the EDX dataset?

```
# 3 - How many different movies are in the EDX dataset?
edx %>% summarize(dif_movies = n_distinct(movieId)) # 10677
```

```
##   dif_movies
## 1      10677
```

4.4 - How many different users are in the EDX dataset?

```
# 4 - How many different users are in the EDX dataset?
edx %>% summarize(dif_users = n_distinct(userId)) #
```

```
##   dif_users
## 1     69878
```

5 - How many movie ratings are in each of the following genres in the EDX dataset?

```
# 5 - How many movie ratings are in each of the following genres in the EDX dataset?
drama_filter <- edx %>% filter(str_detect(genres, "Drama")) %>% nrow()
comedy_filter <- edx %>% filter(str_detect(genres, "Comedy")) %>% nrow()
thriller_filter <- edx %>% filter(str_detect(genres, "Thriller")) %>% nrow()
romance_filter <- edx %>% filter(str_detect(genres, "Romance")) %>% nrow()

print('Drama:')
```

```
## [1] "Drama:"
```

```
drama_filter # 3910127
```

```
## [1] 3910127
```

```
print('Comedy:')
```

```
## [1] "Comedy:"
```

```
comedy_filter # 3540930
```

```
## [1] 3540930
```

```
print('Thriller:')
```

```
## [1] "Thriller:"
```

```
thriller_filter # 2325899
```

```
## [1] 2325899
```

```
print('Romance:')
```

```
## [1] "Romance:"
```

```
romance_filter # 1712100
```

```
## [1] 1712100
```

4.6 - Which movie has the greatest number of ratings?

```
# 6 - Which movie has the greatest number of ratings?
greatest_ratings <- edx %>% group_by(title) %>% summarize(number = n()) %>% arrange(d
esc(number))
greatest_ratings[1,1] # Pulp Fiction (1994)
```

```
## # A tibble: 1 x 1
##   title
##   <chr>
## 1 Pulp Fiction (1994)
```

4.7 - What are the five most given ratings in order from most to least?

```
# 7 - What are the five most given ratings in order from most to least?
most_given_ratings <- edx %>% group_by(rating) %>% summarize(number = n()) %>% arrang
e(desc(number))
head(most_given_ratings$rating, 5) # 4.0 3.0 5.0 3.5 2.0
```

```
## [1] 4.0 3.0 5.0 3.5 2.0
```

4.8 - True or False: In general, half star ratings are less common than whole star ratings?

```
# 8 - True or False: In general, half star ratings are less common than whole star ra
tings?
#table(edx$rating)
half_stars <- edx %>% filter(str_detect(rating, ".5")) %>% nrow()
whole_stars <- data_observations - half_stars
print(half_stars < whole_stars) # True
```

```
## [1] TRUE
```

5 - RMSE - Evaluating the model

5.1 - Now we will look forward the Root Mean Squared Error.

```
# RMSE
RMSE <- function(actual, predicted){
  sqrt(mean((actual - predicted)^2))
}
```

5.2 - Setting the Lambda range to test through the loop function.

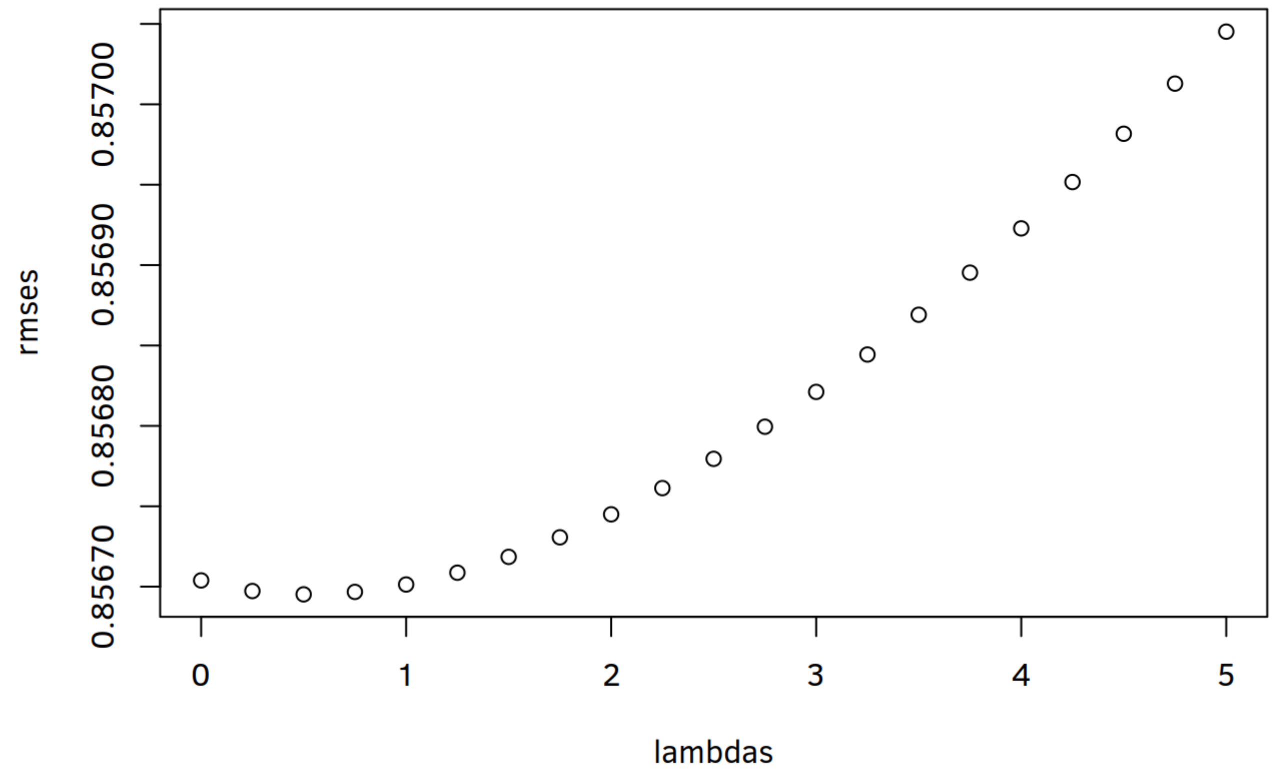
```
lambdas <- seq(0, 5, 0.25)
```

5.3 - RMSEs loop function to test the optimal Lambda value.

```
rmses <- sapply(lambdas, function(l) {

  # Assess the Mu average from the training set ratings
  mu <- mean(edx$rating)

  # Assess the average by movie only - penalize movies with few ratings
  moa_balance <- edx %>%
    group_by(movieId) %>%
    summarize(moa_balance = sum(rating - mu) / (n()+l))

  # Assess the average by movie and user - penalize movies with few ratings
  mua_balance <- edx %>%
    left_join(moa_balance, by = "movieId") %>%
    group_by(userId) %>%
    summarize(mua_balance = sum(rating - moa_balance - mu) / (n()+l))
  predicted_ratings <- edx %>%
    left_join(moa_balance, by = "movieId") %>%
    left_join(mua_balance, by = "userId") %>%
    mutate(col_pred = mu + moa_balance + mua_balance) %>%
    .$col_pred

  return(RMSE(predicted_ratings, edx$rating))
})
```

5.4 - Now that we have tested and store the Lambdas RMSEs, we'll plot RMSE against the Lambda value to see the lowest value.

```
plot(lambdas, rmses)
```



We see that the optimal Lambda stands between 0 and 1, something about 0.5.

5.5 - Let’s see if we deduced right:

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

print('Minimal RMSE:')
```

```
## [1] "Minimal RMSE:"
```

```
min(rmses)
```

```
## [1] 0.8566952
```

```
print('Minimal Lambda:')
```

```
## [1] "Minimal Lambda:"
```

```
min_lambda
```

```
## [1] 0.5
```

As expected, the optimal Lambda value that stands for the lowest RMSE is 0.5.

5.6 - Now we’ll run the prediction function using the optimal Lambda value.

```
y_hat <- sapply(min_lambda, function(l) {

  # Assess the Mu average from the training set ratings
  mu <- mean(edx$rating)

  # Adequate Movie Only Average to min Lambda
  moa_balance <- edx %>%
    group_by(movieId) %>%
    summarize(moa_balance = sum(rating - mu) / (n()+l))

  # Adequate Movie and User Average to min Lambda
  mua_balance <- edx %>%
    left_join(moa_balance, by = "movieId") %>%
    group_by(userId) %>%
    summarize(mua_balance = sum(rating - moa_balance - mu) / (n()+l))

  # Assess the prediction over the validation set
  predicted_ratings <-
    validation %>%
    left_join(moa_balance, by = "movieId") %>%
    left_join(mua_balance, by = "userId") %>%
    mutate(col_pred = mu + moa_balance + mua_balance) %>%
    .$col_pred
    return(predicted_ratings)
})
```

Fine!

6 - Save file to CSV

```
# Save file to CSV
write.csv(validation %>% select(userId, movieId) %>% mutate(rating = y_hat), "submiss
ion.csv", na = "", row.names = FALSE)
```