Report movieLens Project

Felipe Vergara

1.-Introduction

This is a report of estimating movie ratings using data from the MovieLens dataset to fulfill the requirements of HarvardX Data Science Certificate. Basically the aim is to predict how a user will rate a specific movie according to specific dataset characteristics.The original data was obtained from this website.

It was used the following libraries:

```
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")
if(!require(broom)) install.packages("broom", repos = "http://cran.us.r-project.org")
if(!require(pander)) install.packages("pander", repos = "http://cran.us.r-project.org")
```

```
library(tidyverse)
library(caret)# machine learning procedure
library(data.table)
library (broom)
library(pander)#to create table
```

1.1-Load the training and test data:

The dataset is compounded by ratings of users to movies from a score of 0 (very bad) to 5 (very good). In this case, the data was already sortered and separated previously in the training and test data.

As according to the machine learning methodology, **the training data is for the algorithm creation, and the test data is for assessing our final algorithm**. Generally, during the procedure is used mostly the training data which corresponds to the 90% of the dataset. Therefore, here it is presented briefly the training data which is a large dataset.

Training data dimension:

rows	columns
9000055	6

As a first look, an user can rate different movies, so it was necessary to have a frame of how many users and movies were related.

Dataset example:

Indeed it contains a large number of users and movies as it can be seen in the table below:

User and movies numbers:

userId	movieId	rating	timestamp	title	genres
1	122	5	838985046	Boomerang (1992)	Comedy|Romance
1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi

users	movies
69878	10677

2.-Analysis

After a brief exploration the next step was to explore the data in detail with the purpose to identify the variable distribution, to clean it if it is possible, and finally to decide which method was the most appropiate for the given dataset.

2.1.-Cleaning

The dataset has 6 columns, and from all of them the **timestamp** variable did not look helpful for the estimation, thus it was deleted from the table.

```
edx<-edx%>%select(-timestamp)
```

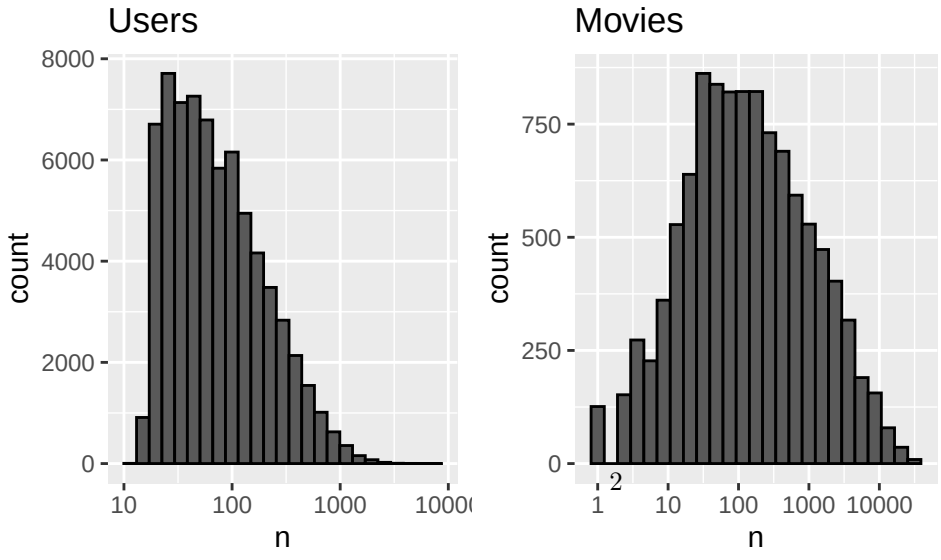
2.2.-Predicting algorithm

According of what it can be seen in the dataset, it was decided to use this approach:

$$Y_{u,i} = \mu + b_i + b_u + \varepsilon_{u,i}$$

Where:
$y_{u,i}$: predicted rate
μ: average of all ratings
b_i: average ranking for movie i or *bias*
b_u: user-specific effect u $\varepsilon_{u,i}$: independent errors sampled taking into account previous parameters.

The formula indicates that the average (μ) of all ratings is the simplest approach to predict a certain rate. However, as the experience says there are movies that are rated higher than others and also there were users who rated extremely positive and negative than others. Although this is complicated to visualize, it can be inferred from the graphs below:



As it can be observed,there are movies that are rated more often that others, and there are users who rated more frequently than others.

Therefore, to integrate the mentioned effect, b_i refers to the average ranking of movie $_i$. On the other side b_u represents the average rate of user $_u$ corrected by b_i.

$$\hat{b}_i = Y_{u,i} - \hat{\mu}$$

$$\hat{b}_u = Y_{u,i} - \hat{\mu} - \hat{b}_i$$

2.3.-Testing algorithm

Finally, the algorithm was tested using the typical error loss expressed as the residual mean squared error (RMSE).

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{u,i}\left(\hat{y}_{u,i} - y_{u,i}\right)^2}$$

It means the typical error that is made when it is predicted a rate. The importance of this formula is to observe the differences between the rate $y_{u,i}$ with the predicted rates $\hat{y}_{u,i}$. Low values mean a better prediction of the model. Values over 1, for example, means that the typical error is larger than 1 score, so it is necessary to get values below 1.

2.4.-Penalizations

Notwithstanding, before testing the algorithm it was decided to apply penalties over average ranking for movie b_i and for the user-specific effect b_u. The idea is based that the *bias* and the *user-specific effect* are affected by the number of observations associated to them. So, at larger observations the penalization is lower.

In this sense, to constraint the variability of the effect sizes, it was considered that for b_i and b_u values having few observations, it was better to be assigned values close or even 0, instead of the provided value. The penalization is represented by λ, which is a tuning parameter, being practically ignored when there were large observations. At a larger λ, shrinks more.

Thus, the minimization is represented in the following way:

$$\frac{1}{N}\sum_{u,i}\left(y_{u,i} - \mu - b_i - b_u\right)^2 + \lambda\left(\sum_i b_i^2 + \sum_u b_u^2\right)$$

where to calculate the penalization to b_i and b_u are expressed like this:

$$\hat{b}_i(\lambda) = \frac{1}{\lambda + n_i}\sum_{u=1}^{n_i}\left(Y_{u,i} - \hat{\mu}\right)$$

$$\hat{b}_u(\lambda) = \frac{1}{\lambda + n_i}\sum_{u=1}^{n_i}\left(Y_{u,i} - \hat{\mu} - \hat{b}_u\right)$$

One important point is that the definition of the λ value had to be calculated in a certain way. For this purpose it was selected the cross-validation method, which **must be applied on the training set**.

2.4.1.-Cross-validation The basic idea of cross-validation is to create an algorithm or to estimate a parameter, in this case λ, and to test it in the training data. Therefore, to be consistent with the machine learning methodology, the training data must be split between **training data** and **validate set data**. Then, to obtain more convinced results, this procedure has to occure several times over the original training data, creating the known *K-folds*.

On the other hand, **the test data, here validation dataset, it is not used at all**.

One important point is to apply the parameter equally across all the *K-folds*, before starting with a new parameter value. So for this project it was tested the algorithm calculating RMSE, but wit several provided parameter values.
This is expressed with this formula:

$$\text{RMSE}(\lambda) = \frac{1}{K}\sum_{b=1}^{K}\text{RMSE}_b(\lambda)$$

In this sense, it was defined to create 3 *K-folds*, due to hardware processing. As a result or estimation is the following:

$$\hat{\text{RMSE}}(\lambda) = \frac{1}{3}\sum_{b=1}^{3}\hat{\text{RMSE}}_b(\lambda)$$

The final step is to select the parameter that minimizes the RMSE (see Results section).

2.5.-Testing calibrated algorithm

Finally, after obtaining the optimal λ, it was added to the algorithm and reapplied the test, but instead of using the training data as in section 2.4.1, it was used the test data (validation dataset).

3.-Results

This section is organized presenting the lambda value selection and the values of the typical error loss.

3.1.-Choosing Lambda

First, it was created the *K-folds* with their corresponding training and validate set. The process was meticulously made, assuring to define properly the training and test set from the training set *edx*.

```
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use 'set.seed(1)'
# to create 3 training partition to identify the appropriate lambda value
edx_test_index <- createDataPartition(y = edx$rating, times = 3, p = 0.1, list = FALSE)
k<-3
#partition of the training dataset (edx) to apply cross-validation
edx_tr_list<-list()
edx_temp_list<-list()
edx_test_list<-list()
removed<-list()
for (i in 1:k){
edx_tr_list[[i]]<- edx[-edx_test_index[,i],]
edx_temp_list[[i]]<- edx[edx_test_index[,i],]
#Make sure userId and movieId in edx validation set are also in edx training set
edx_test_list[[i]] <- as.data.frame(edx_temp_list[i]) %>%
  semi_join(edx_tr_list[[i]], by = "movieId") %>%
```

```
  semi_join(edx_tr_list[[i]], by = "userId")
#Add rows removed from edx validation set back into edx training set
removed[[i]] <- anti_join(as.data.frame(edx_temp_list[i]),as.data.frame(edx_test_list[[i]]))
edx_tr_list[[i]] <- rbind(edx_tr_list[[i]], removed[[i]])
}
#remove temporal unnecessary data
rm(edx_temp_list,removed, edx_temps,control,edx_trainings)
```

Secondly, it was applied the cross validation formula. To simplify the calculations it was tested only 11 λ values, all of them integer values from 0 to 11.

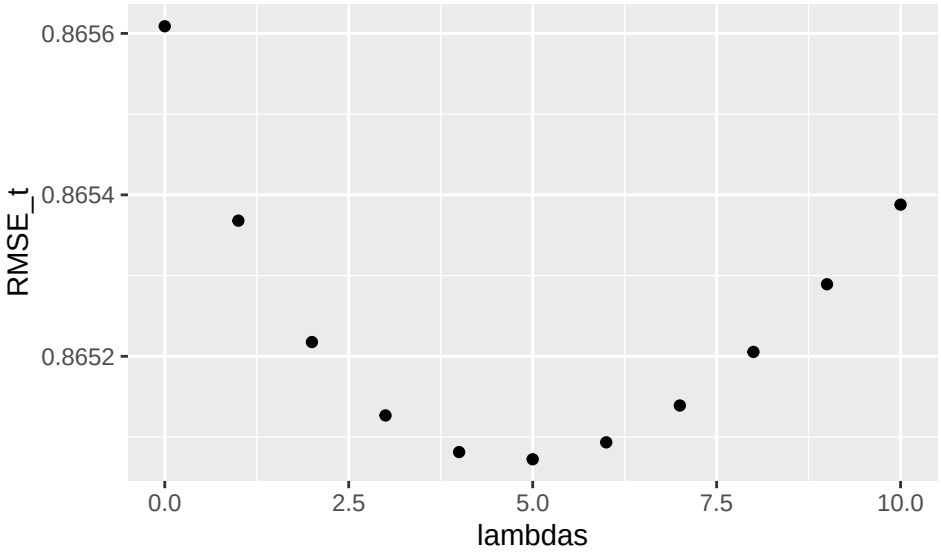
```
#Procedure is to test  possible lambda values,
#then it is created 3x11 possible RMSE values, using loop, sapply
#and matrix formulas.

lambdas <- seq(0, 10, 1)# 11 possible lambda values
rmses<-list()
for (j in 1:3) {
rmses[[j]] <- sapply(lambdas, function(l){
  mu <- mean(edx_tr_list[[j]]$rating)

  b_i <- edx_tr_list[[j]] %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  b_u <- edx_tr_list[[j]] %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  predicted_ratings <-
    edx_test_list[[j]] %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  return(RMSE(predicted_ratings, edx_test_list[[j]]$rating))
})}
```

In this case, the procedure took a couple of minutes since it was creating at least 33 operations (3 *K-folds* x 11 λ values).

The next step was to get the overall RMSE according to each λ value, and then to select the optimal λ.

```
#RMSE calculation from the cross-validation step
RMSE_t<-rowMeans(matrix(cbind(rmses[[1]],rmses[[2]],rmses[[3]]),11,3))
```

The next graph present the λ results:



As it can be seen, the optimal λ from the cross validation is 5

```
lambda <- lambdas[which.min(RMSE_t)]
lambda# optimal lambda
```

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

So the next step was to prove this lambda in the algorithm and to test it with the validation dataset with RMSE.

3.2.-RMSE

The algorithm calculation is the following:

```
##Final calculation
mu <- mean(edx$rating) # average of all ratings
b_i <- edx %>% #average ranking for movie i
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

b_u <- edx %>% #user-specific effect u
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

predicted_ratings <- #Prediction
  validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
```

As it can be observed, it has 4 calculations, the first 3 are the variables $\hat{\mu}$, $\hat{b}_i$ and $\hat{b}_u$, the last one is the prediction $\hat{y}_{u,i}$. Likewise, it was added the lambda value.

Finally, the last step was to test the algorithm.

```
RMSE(predicted_ratings, validation$rating) #Final outcome
```

```
## [1] 0.8648177
```

An acceptable prediction, since is below 1, fulfilling the machine learning requirements.

4.-Conclusions

Summarizing, the obtained result satisfies the expectations of this project. It was applied a recommendation system to predict user rate of different movies from the MovieLens dataset. The approach used 3 variables (average of all rankings, average ranking of a movie and user-specific effect), and added penalizations to the last variables as well to incorporate specific characteristics of the movie streaming field. In the end, the result is trustworthy, having a loss of less than **0.8648177**, fulfilling the requirements of the course.

Even though this project is based in a consistent machine learning approach (recommendation system), there are still data patterns that could be integrated to the algorithm. For example, it seems that rates are related not only to movie genres (drama, comedy, etc) and/or topics (mafia, medieval, etc), but also to actor/actress. Considering them undoubtedly will help to improve the algorithm, being extremely useful to predict rates in the streaming company or even for similar activities.