MovieLens project_report

Jody Iabichino

15/11/2021

1. Introduction

1.1 Overview

The project that we are describing is based on the dslabs dataset "movielens" that contains information about real-world movie ratings. In this dataset each row represents a unique rating given to a specified movie by a single user. The aim of the project was to build a machine learning algorithm able to predicts the movie ratings with the lowest RMSE achievable. RMSE is a kind of loss function, a typical metric of "goodness of fit", that we had to minimize as much as possible. We can interpret RMSE similar to standard deviation.

The steps followed in this analysis were essentially three. First of all, we built our dataset that after some preliminary analysis (such as data exploration and data cleaning) was splitted into two separate datasets, the "training set" (edx) and the "test set"(temp). Then, a machine learning algorithm was created. In particular, four models were tested, which were very similar to each other and differed only in a few parameters. According to the rules of machine learning, the RMSE on the test set was determined for each of these. In the end, the model with the lowest RMSE was selected.

2. Analysis

2.1 Data exploration

Before starting with the actual data exploration, the zipper file containing all the necessary data was downloaded:

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

The first file was "ratings.dat", which was stored in the "ratings" dataset; the second file was "movies.dat", which was stored in the "movies" dataset.

Of the two datasets, the first few rows were analyzed using the following code:

```
head(ratings)
```

```
##    userId movieId rating timestamp
## 1:      1     122      5 838985046
## 2:      1     185      5 838983525
## 3:      1     231      5 838983392
## 4:      1     292      5 838983421
## 5:      1     316      5 838983392
## 6:      1     329      5 838983392
```

```
head(movies)
```

```
##   movieId                               title
## 1       1                   Toy Story (1995)
## 2       2                     Jumanji (1995)
## 3       3            Grumpier Old Men (1995)
## 4       4           Waiting to Exhale (1995)
## 5       5 Father of the Bride Part II (1995)
## 6       6                         Heat (1995)
##                                        genres
## 1 Adventure|Animation|Children|Comedy|Fantasy
## 2                  Adventure|Children|Fantasy
## 3                              Comedy|Romance
## 4                        Comedy|Drama|Romance
## 5                                      Comedy
## 6                       Action|Crime|Thriller
```

Both datasets had a common key variable, "movieId". This variable was used to create a single dataset "movielens":

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

Analyzing the content of `movielens`, it was possible to find that it had 6 variables and 10000054 rows. Also of this dataset, the first few rows were analyzed to understand the granularity of the data:

```
head(movielens)
```

```
##    userId movieId rating timestamp                          title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     231      5 838983392          Dumb & Dumber (1994)
## 4:      1     292      5 838983421               Outbreak (1995)
## 5:      1     316      5 838983392               Stargate (1994)
## 6:      1     329      5 838983392 Star Trek: Generations (1994)
##                          genres
## 1:               Comedy|Romance
## 2:         Action|Crime|Thriller
## 3:                       Comedy
## 4:  Action|Drama|Sci-Fi|Thriller
## 5:       Action|Adventure|Sci-Fi
## 6: Action|Adventure|Drama|Sci-Fi
```

`Movielens` presents 69878 distinct users. Each user gave one or more ratings to one or more movies and some of these users were more "active" than others. We could see that the top 10 users based on number of ratings gave a total of 47768 ratings:

```
## Selecting by count
```

```
## # A tibble: 10 x 2
##    userId count
##     <int> <int>
```

```
##  1  59269  7359
##  2  67385  7047
##  3  14463  5169
##  4  68259  4483
##  5  27468  4449
##  6   3817  4165
##  7  19635  4165
##  8  63134  3755
##  9  58357  3697
## 10  27584  3479
```

`Movielens` presents also 10677 distincts "movieId". A movieId is a code which identifies each movie. We found that the "title" variable could be used to enrich "movieId" with information. The top 10 movies based on absolute and relative count of ratings are shown here:

```
## # A tibble: 10 x 3
##    movieId_and_title                                                  count count_percentage
##    <chr>                                                              <int>            <dbl>
##  1 296_Pulp Fiction (1994)                                            34864          0.00349
##  2 356_Forrest Gump (1994)                                            34457          0.00345
##  3 593_Silence of the Lambs, The (1991)                               33668          0.00337
##  4 480_Jurassic Park (1993)                                           32631          0.00326
##  5 318_Shawshank Redemption, The (1994)                               31126          0.00311
##  6 110_Braveheart (1995)                                              29154          0.00292
##  7 457_Fugitive, The (1993)                                           28951          0.00290
##  8 589_Terminator 2: Judgment Day (1991)                              28948          0.00289
##  9 260_Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)   28566          0.00286
## 10 150_Apollo 13 (1995)                                               27035          0.00270
```

Each movie was classified by genre and the total number of distinct genres was 797. Genres could be described by a single word or by a combination of words. The top 10 genres in the database can be seen here, each followed by the absolute and relative count of its appearances:

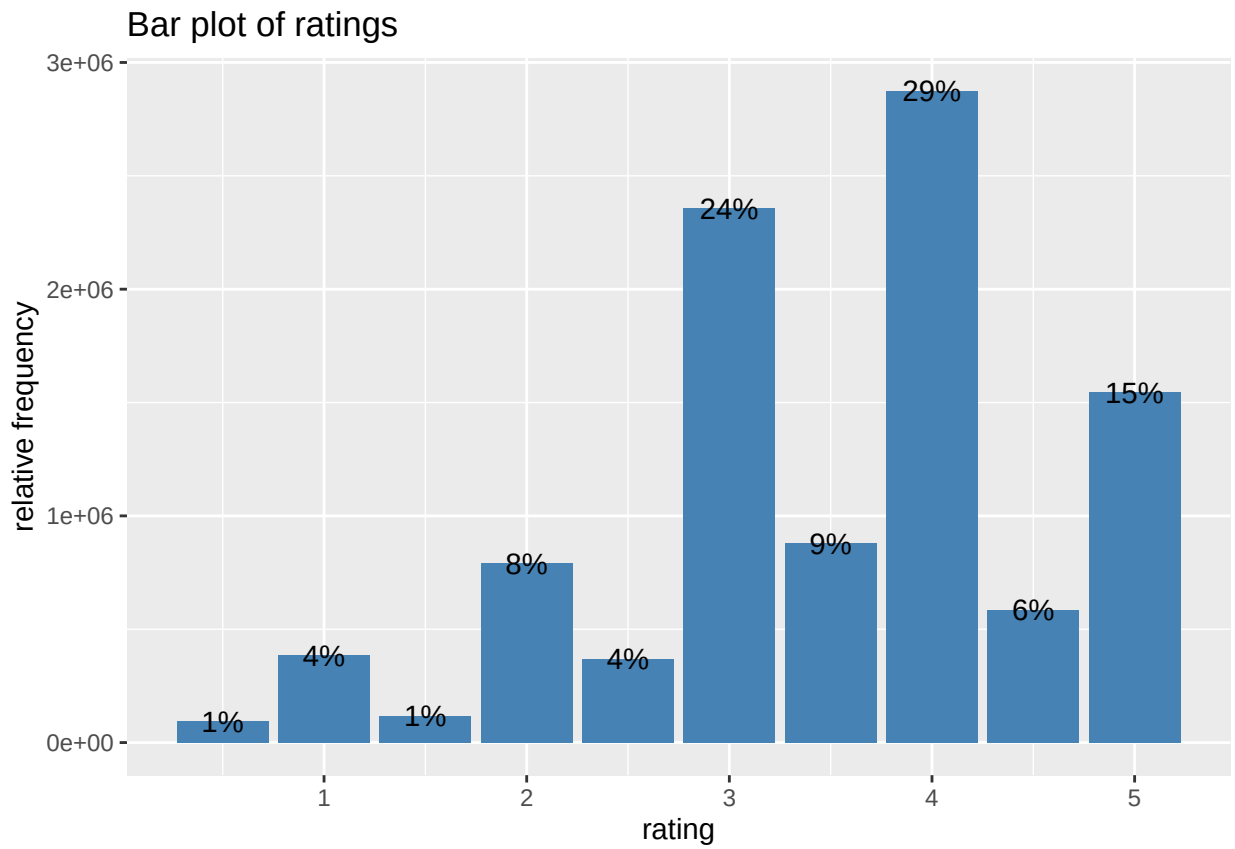
```
## Selecting by count_percentage
```

```
## # A tibble: 10 x 3
##    genres                     count count_percentage
##    <chr>                      <int>            <dbl>
##  1 Drama                     815084           0.0815
##  2 Comedy                    778596           0.0779
##  3 Comedy|Romance            406061           0.0406
##  4 Comedy|Drama              359494           0.0359
##  5 Comedy|Drama|Romance      290231           0.0290
##  6 Drama|Romance             288539           0.0289
##  7 Action|Adventure|Sci-Fi   244586           0.0245
##  8 Action|Adventure|Thriller 165671           0.0166
##  9 Drama|Thriller            161609           0.0162
## 10 Crime|Drama               152827           0.0153
```

Continuing with our analysis, we analized the target variable of the machine learning exercise: the rating. `movielens` had a total of 10 possible ratings, which can be seen here, each followed by the count of appearances:

```
## # A tibble: 10 x 2
##    rating   count
##     <dbl>   <int>
##  1    0.5   94988
##  2    1    384180
##  3    1.5  118278
##  4    2    790306
##  5    2.5  370178
##  6    3   2356676
##  7    3.5  879764
##  8    4   2875850
##  9    4.5  585022
## 10    5   1544812
```

A bar plot helped us to visualize how the ratings were distributed in the dataset:



Looking at the numbers, we found that a half of total ratings were either "3" or "4"; non-integer ratings like "0.5" or "1.5" were not frequently used.

2.2 Data cleaning

After the data exploration, we analized `movielens` to evaluate the quality of the dataset. Indeed, a common task in data analysis is dealing with missing values. In R, missing values are often represented by NA or some other value that represents missing values (i.e. 99). Here we report the results of our analysis:

```
sum(is.na(movielens))
```

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

The results showed that movielens was an extremely clean dataset, with no missing values present.

2.3 Data partition

Concluding data exploration and data cleaning, it was possible to split `movielens` into two distinct databases: the training set, named "edx", and the test set, called "validation", that was the 10% of `movielens` data:

```
library(dslabs)
  library(tidyverse)
  library(caret)
  set.seed(1)
  test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")
removed <- anti_join(temp, validation)
```

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

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

It's important to highlight that through the "semi_join" function, we splitted movielens into two databases so that the users present into the training set were also present in the test set. We created also the RMSE function, useful to estimate the accuracy of our model.

2.4 Modelling approach: the approach presented

After the creation of the training set, "edx", it was possible to start our modelling approach to estimate the variable "rating". During the Harvard course, a step-by-step approach was presented, starting with a very simple model in which the same rating was assumed for all movies and users, with all differences explained by random variation. The model was as follows: y(u,i)= mu + E(u,i)

Where mu was the average rating for all movies and all users (the least square estimate) and E represented the sampled independent errors. To improve the model, the term b(i) was then added to represent the average of the ratings for movie i, but also the term b(u) was added to account for what was the user-specific effect. To still improve the results, it was used regularization. Regularization constrains the total variability of the effect sizes by penalizing large estimates that come from small sample sizes,using a penalty term: lambda, a tuning parameter.

2.5 Modelling approach: our approach

To further improve the model, we decided to make two separate changes. First, we introduced a new parameter b_g, which would take into account the effect of genre. After that, we decided to modify mu. In fact, we started from the idea of not considering the simple average of the whole dataset, but to build a sort of weighted average, which would consider the average of all the ratings for each user, the average of all the ratings for each movie and the average of all the ratings for each genre. To define the weight of each mean in the composition of mu, we used an iterative process. Four models were therefore constructed, each with a different set of weights. We started from a situation in which the weights were the same for all three components of mu (model 1), and then gave greater weight to the average of the user ratings in the subsequent models.

2.5.1 Model 1

As already mentioned, in the first model we started with a situation where the three weights of mu all had the same value. To estimate instead the b's, it was minimized an equation which contained a penalty term: lambda, a tuning parameter. We therefore initially defined the value of lambda and the three weights, and then launched a code which, compared to the one presented in class, in addition to the introduction of the term b_g, had a different way of estimating the mean mu.

```
lambdas <- seq(0, 10, 0.25)

  weight1<-0.33
  weight2<-0.33
  weight3<-0.33

  rmses <- sapply(lambdas, function(l){
    mu1 <- edx %>% group_by(genres) %>%
      summarize(mu1=mean(rating))
    mu2 <- edx %>% group_by(movieId) %>%
      summarize(mu2=mean(rating))
    mu3 <- edx %>% group_by(userId) %>%
      summarize(mu3=mean(rating))

    mu <- edx %>%
      left_join(mu1, by = "genres") %>%
      left_join(mu2, by = "movieId") %>%
      left_join(mu3, by = "userId") %>%
      mutate(mu=weight1*mu1+weight2*mu2+weight3*mu3)%>%
      select(genres, movieId, mu, mu1, mu2, mu3)

    edx2<- data.frame(edx, mu)

    b_i <- edx2 %>%
      group_by(movieId) %>%
      summarize(b_i = sum(rating - mu)/(n()+l))
    b_u <- edx2 %>%
      left_join(b_i, by="movieId") %>%
      group_by(userId) %>%
      summarize(b_u = sum(rating - b_i - mu)/(n()+l))
        b_g <- edx2 %>%
      left_join(b_i, by="movieId") %>%
      left_join(b_u, by="userId") %>%
```

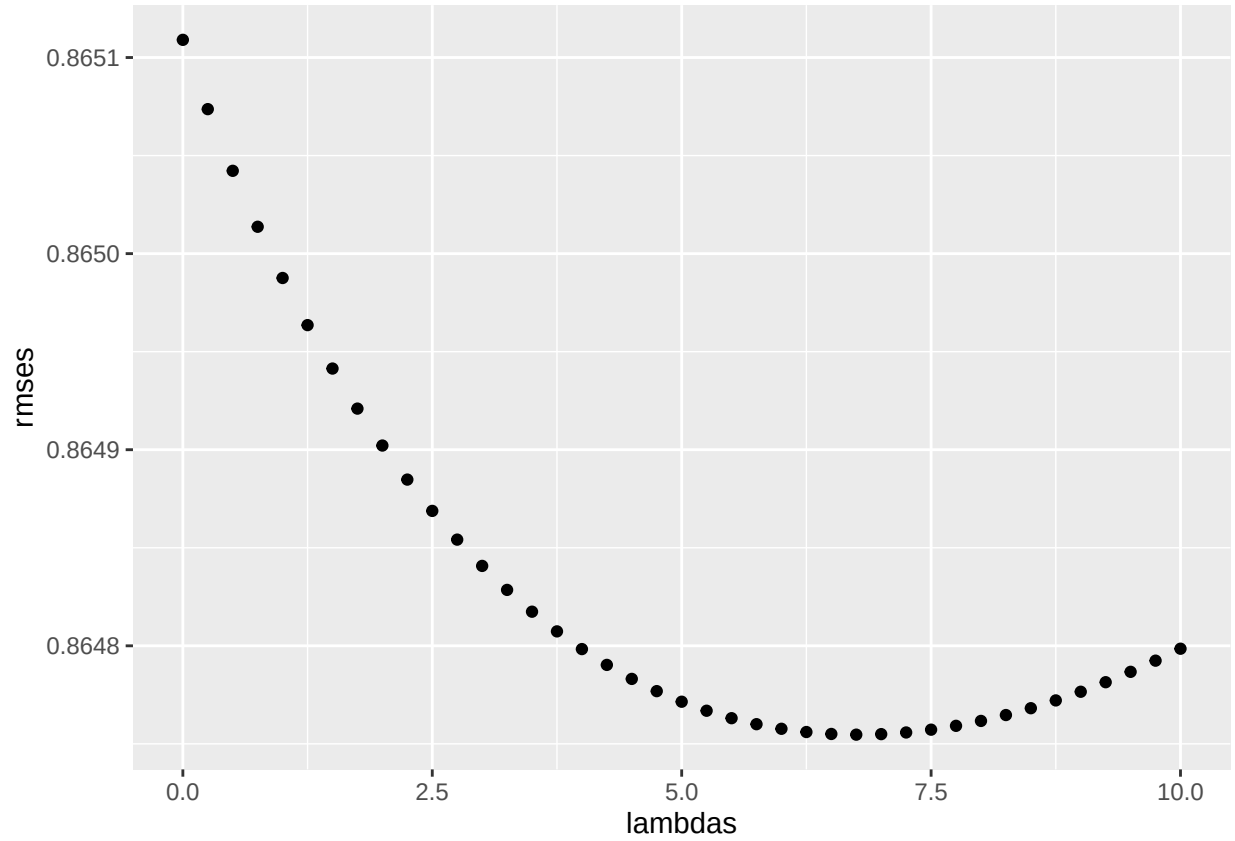
```
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_i - b_u - mu)/(n()+l))

  predicted_ratings <-
    validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    left_join(mu1, by= "genres") %>%
    left_join(mu2, by="movieId")%>%
    left_join(mu3, by="userId")%>%
    mutate(mu_t=weight1*mu1+weight2*mu2+weight3*mu3)

predicted_ratings2<-predicted_ratings %>%
    mutate(pred = mu_t + b_i + b_u + b_g) %>%
    .$pred
return(RMSE(predicted_ratings2, validation$rating))
})
```

At this point, after calculating all possible RMSEs associated with the vector of lambdas, we looked for the lambda value at which the RMSE reaches its minimum value.

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



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

```
## [1] 6.75
```

So the lambda value at which the RMSE is lowest is 6.5. We then relaunched the code by substituting the value 6.5 in place of the lambda vector.

```
lambdas <- 6.5

  weight1<-0.33
  weight2<-0.33
  weight3<-0.33

  rmses <- sapply(lambdas, function(l){
    mu1 <- edx %>% group_by(genres) %>%
      summarize(mu1=mean(rating))
    mu2 <- edx %>% group_by(movieId) %>%
      summarize(mu2=mean(rating))
    mu3 <- edx %>% group_by(userId) %>%
      summarize(mu3=mean(rating))

    mu <- edx %>%
      left_join(mu1, by = "genres") %>%
      left_join(mu2, by = "movieId") %>%
      left_join(mu3, by = "userId") %>%
      mutate(mu=weight1*mu1+weight2*mu2+weight3*mu3)%>%
      select(genres, movieId, mu, mu1, mu2, mu3)

    edx2<- data.frame(edx, mu)

    b_i <- edx2 %>%
      group_by(movieId) %>%
      summarize(b_i = sum(rating - mu)/(n()+l))
    b_u <- edx2 %>%
      left_join(b_i, by="movieId") %>%
      group_by(userId) %>%
      summarize(b_u = sum(rating - b_i - mu)/(n()+l))
        b_g <- edx2 %>%
      left_join(b_i, by="movieId") %>%
      left_join(b_u, by="userId") %>%
      group_by(genres) %>%
      summarize(b_g = sum(rating - b_i - b_u - mu)/(n()+l))

    predicted_ratings <-
      validation %>%
      left_join(b_i, by = "movieId") %>%
      left_join(b_u, by = "userId") %>%
      left_join(b_g, by = "genres") %>%
      left_join(mu1, by= "genres") %>%
      left_join(mu2, by="movieId")%>%
      left_join(mu3, by="userId")%>%
      mutate(mu_t=weight1*mu1+weight2*mu2+weight3*mu3)
```

```
predicted_ratings2<-predicted_ratings %>%
    mutate(pred = mu_t + b_i + b_u + b_g) %>%
    .$pred
return(RMSE(predicted_ratings2, validation$rating))
})
```

In doing so, we obtained an RMSE value of 0.864755.

2.5.2 Other models

In order to make as accurate a choice as possible of the weights constituting the mu average, we built three more models, identical in structure to the first, in which we tried to vary these weights. From model to model, we gave more and more weight to the average of the user's ratings, trying to see what happened to the RMSE. This is the weight structure that was used:

Model	mu1	mu2	mu3
1	0.33	0.33	0.33
2	0.25	0.35	0.40
3	0.20	0.35	0.45
4	0.20	0.30	0.50

For each of the models, the same procedure was therefore carried out as for the first model, by first submitting the entire lambda vector and then searching for the values of the latter at which the RMSE reached its minimum value. Optimal lambda values and relative RMSEs follow:

Model	Optimal_lambda	RMSE
1	6.50	0.8644893
2	6.75	0.8645668
3	6.75	0.8646413
4	6.25	0.8647332

3. Results

The four models analysed produced four RMSE values. The best model, i.e. the one producing the lowest RMSE, was the first.

4. Conclusion

Model 1 produced appreciable results, with a really low RMSE. The idea of modifying the mean in particular seems to have made a positive contribution in terms of results. Clearly, the model could be further refined. Among its limitations, for example, is how to choose the weights that make up mu. Four possible structures were tested in this analysis, but it is that clear that one could optimize the choice through an iterative process leading to the choice of the absolute best weights. Another limitation is that we have not considered principal component analysis, which could lead to an improvement and further lowering of the RMSE. On the other hand, regarding the rating prediction formula, the approach used was definitely a simplified approach, based on the average. The use of more refined prediction models, such as knn or random forest, was initially excluded already during the course lectures due to the high volatility of the y variable, the high number of predictors and the presence of null values. It would be interesting to see if by simplifying the database in some way, more refined rating prediction formulas could be applied, perhaps making use of ensembling to consider multiple methods.