Movielens_analysis_report

Emmanuelle RASSEK
January 27th,2019

```

# Report

# Movielens recommendation system

## 1. Executive summary

Recommendation system is one of the most famous machine learning models. Nowadays, many companies use it to improve customer journeys thanks to relevant products recommandations. The success of Netflix, for instance, is - among other things - based on its strong recommendation system.

For this project, we will be creating a movie recommendation system using the MovieLens dataset, collected by GroupLens Research.

Our objective is to predict - in the most accurate way possible - the movie ratings of the users thanks to the development of a machine learning algorithm.

In order to do so, we will use the 10M100K version of the MovieLens dataset, included in the dslabs package, and divide the dataset into two subsets:

- a training subset to train our algorithm, called “edx”
- a validation subset to predict the movie ratings, called “validation”

In the first part of the report, we will use techniques such as data cleaning and exploration to have an overview of the dataset. Then, we will train five algorithms in order to find a model with the best possible accuracy (RMSE). Finally, we will explain the results and conclude. All the analysis will be made through R studio and the following packages: dplyr, tidyverse and caret.

## 2. Analysis: data description, preparation, exploration and visualization

### 2.A. Data description and cleaning

In this section we will take a first look at our MovieLens dataset. We will also perform necessary cleaning and transformations so that the data better suits our needs.

**Data description**

Our dataset contains 10,000,054 rows, 10,677 movies and 69,878 MovieLens users.
```

It is composed of six features: genre(s), userID, movieID, timestamp,title and rating. The classes of the following features have been re-defined:

- movieId: numeric
- genre: character
- title: character

As mentioned in the introduction, we divided our dataset into two subsets:

- a training subset called "edx". It is composed of **9,000,047** rows and represents 90% of MovieLens data.
- a validation subset called "validation". It is composed of **999,999** rows and it represents 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,]
```

Data preparation

In this project, we have selected the following features to train our algorithms: "rating", "movieId" and "userId". Let's analyse the structure of these columns and check if some data cleaning is necessary.

First, let's analyse the structure of the **"rating"** feature:

```
summary(movielens$rating)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.500   3.000   4.000   3.512   4.000   5.000
```

It seems that there is no missing data (NA).

Now, let's analyse the structure of the **"movieId"** feature:

```
summary(movielens$movieId)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       1     648    1834    4120    3624   65133
```

It seems that there is no missing data (NA) neither.

Finally, let's analyse the structure of the **"userId"** feature:

```
summary(movielens$userId)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       1   18123   35741   35870   53608   71567
```

It seems that there is no missing data (NA) neither.

As we will use a data partition to train our algorithms, let's 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")
```

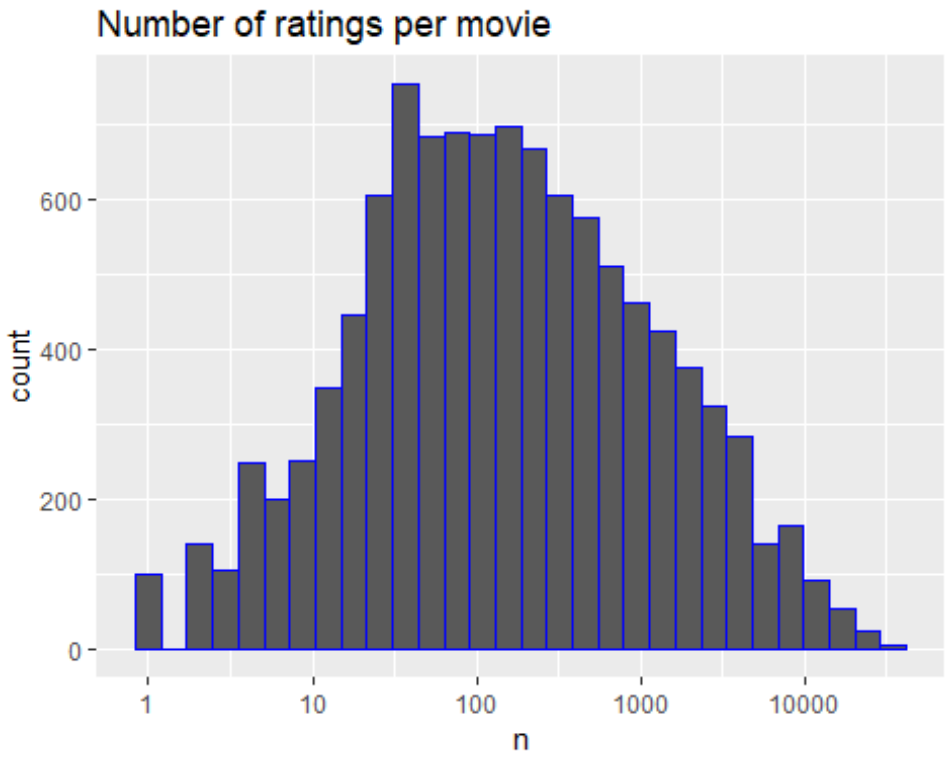
Now let's add rows removed from validation set back into edx set:

```
removed <- anti_join(temp, validation)

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

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

2.B. Data exploration and visualization

Movie ratings: how frequently are movies rated?

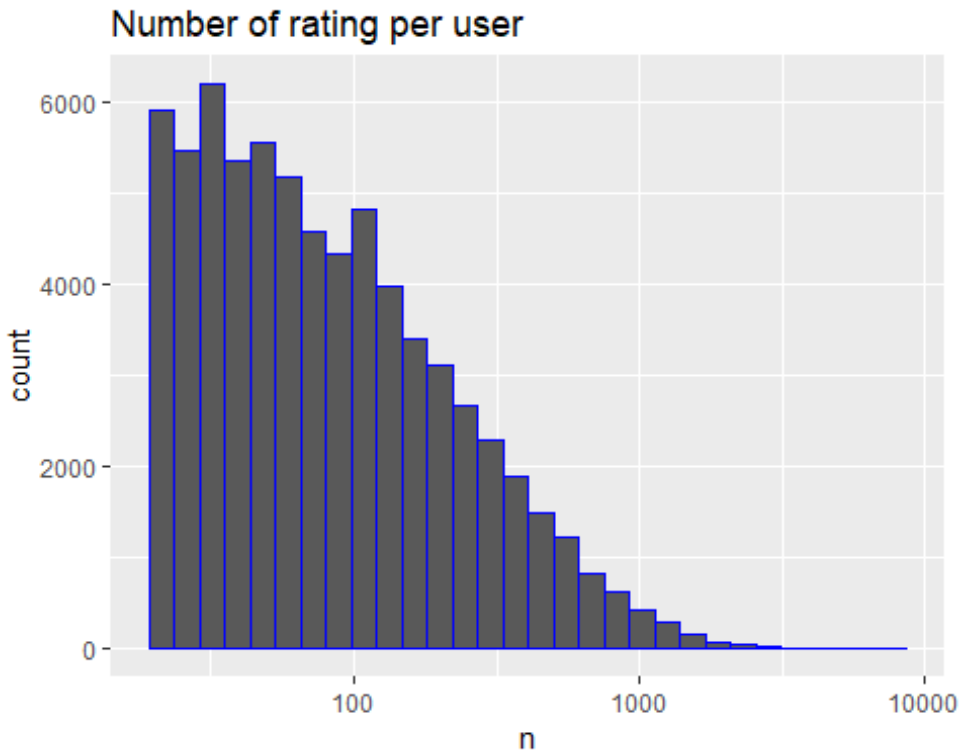
```
movielens %>%
count(movieId) %>%
ggplot(aes(n)) +
geom_histogram(bins = 30, color = "blue") +
scale_x_log10() +
ggtitle("Number of ratings per movie")
```



It appears that some movies are much more rated than others.

User ratings: how often do the users rate movies ?

```
movielens %>%
count(userId) %>%
ggplot(aes(n)) +
geom_histogram(bins = 30, color = "blue") +
scale_x_log10() +
ggtitle("Number of rating per user")
```



We observe that some users rate movies much more often than others.

Thanks to the preparation and exploration of the dataset, we are now ready to train several algorithms.

2.C. Modelling approach

The objective is to define a method that will allow us to train several algorithms in order to identify the best one.

We will proceed in four steps:

- training of the algorithms on the training set:“edx”
- predictions on the validation set:“validation”
- comparision of the RMSE
- conclusion

Please find below the five algorithms we are going to train and test:

a. Model-based approach

b. Content-based approach

c. User-based approach

d. Regularized content-based approach

e. Regularized content-based approach + user-based approach

3. Results of the analysis

Data partition

In order to reach our goal, we will use our two MovieLens dataset subsets:

- the training subset to train our algorithm, called "edx" (90% of MovieLens data)
- the validation subset to predict the movie ratings, called"validation" (10% of MovieLens data)

We are now ready to start the development of our predictive models. As we will go along, we will compare the 5 approaches mentioned above.

a. Model-based approach

In this first approach, we consider the same ratings for all movies and users. We will use a Naive Bayes model :

Yu,i=mu +epsilon_u,i

mu: average of the "true" rating for all users

epsilon_u,i: independent errors sampled from the same distribution centered at 0

```
mu_hat <- mean(edx$rating)
mu_hat

## [1] 3.512465
```

Result:

```
rmse_1 <- RMSE(validation$rating, mu_hat)
rmse_1

## [1] 1.061202
```

b. Content-based approach

In this second approach, we will evaluate the role of "movie effect" on the rating predictions. Some movies get higher ratings than others and this should be included in our model.

Let's add a term to our previous model:

Yu,i=mu+bi+epsilon_u,i

bi: movie effect

Training of the model:

```
mu <- mean(edx$rating)
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

Predictions:

```
predicted_ratings <- mu + validation %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i
```

Result:

```
rmse_2 <- RMSE(predicted_ratings, validation$rating)
rmse_2

## [1] 0.9439087
```

c. User effect-based model

In this third approach, we will evaluate the role of "user effect" on the rating predictions. If we compute the average rating of the users, we find out that there is variability in the way users give ratings:

- some are generous and their average rating is high
- some have an opposite profile and their average rating is low

Let's include this user variability effect in our model by adding a new term:

Yu,i=mu+bi+bu+epsilon_u,i

Bu: user effect

Training of the model:

```
user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
```

Predictions:

```
predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
```

```
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

rmse_3 <- RMSE(predicted_ratings, validation$rating)
rmse_3

## [1] 0.8653488
```

d. Regularized content-based approach: penalized least squares

Penalized least squares estimates provide a way to balance fitting the data closely and avoiding excessive roughness or rapid variation. A penalized least squares estimate is a surface that minimizes the penalized least squares over the class of all surfaces satisfying sufficient regularity conditions.

Let's try this method to improve our algorithm.

In order to estimate bi, we need to minimize this equation:

1N_somme_u,i(yu,i _ mu _ bi)2 + somme_i b2i

1N_somme_u,i(yu,i _ mu _ bi)2 : least square

lambda somme(bi)^2: penalty that gets larger when many bi are large

We can now show that the values of bi that minimize this equation are:

bi(lambda) = 1lambda+ ni somme_u=1(Yu,mu)

where ni is the number of ratings made for movie i.

So when ni is very large, then lambda is effectively ignored since ni+lambda almost equals ni. However, when ni is small, then the estimate bi(lambda) is shrunken towards 0. The larger lambda, the more we shrink.

Let's compute these regularized estimates of bi using lambda=3.

Training of the model:

```
lambda <- 3
mu <- mean(edx$rating)
movie_reg_avgs <- edx%>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())
```

Predictions:

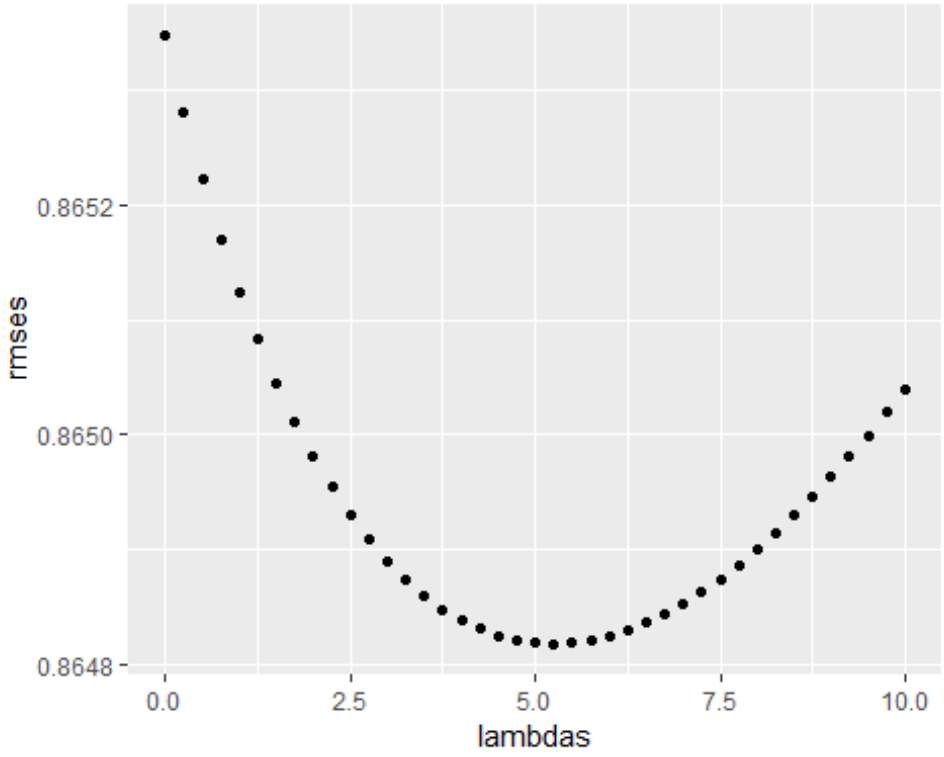
```
predicted_ratings <- validation %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  mutate(pred = mu + b_i) %>%
  .$pred
#Result :
```

```
rmse_4 <- RMSE(predicted_ratings, validation$rating)
rmse_4

## [1] 0.9438538
```

e. Regularized content-based approach + User effect Model

As the best RMSE we got until now is the one from model 3, let's try to combine it with movie regularization. Note that lambda is a tuning parameter. Let's use cross-validation to choose it.

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

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

  mu <- mean(edx$rating)

  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

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

  predicted_ratings <-
    validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred

    return(RMSE(predicted_ratings, validation$rating))
})

qplot(lambdas, rmses)
```




```
rmse_5 <- min(rmses)

lambdas[which.min(rmses)]

## [1] 5.25
```

The lambda value which minimises the rmse is 5.25.

4. Conclusion

Comparison of models performances

We use the Root Mean Squared Error to evaluate the performance of the models.

```
#Model 1
rmse_1

## [1] 1.061202

#Model 2
rmse_2

## [1] 0.9439087

#Model 3
rmse_3

## [1] 0.8653488
```

```
#Model 4
rmse_4

## [1] 0.9438538

#Model 5
rmse_5

## [1] 0.864817
```

Selection of the best model

The model which minimises the RMSE is Model 5: Regularized Movie + User Effect Model.

Model 5: Regularized content-based approach + User effect Model.

To conclude, we can say that in our approach the best model is Model 5.