MovieLens Project Report

Introduction

The MovieLens database was inspired by the Netflix challenge, which began on October 2006[1]. The competition aimed to improve the accuracy of Netflix's recommendation system[1]. To qualify for the price, the winning team had to obtain an outcome at least 10 percent better than that of Netflix[2].

The 10M version of the MovieLens database is very simple in its structure, yet it offers a challenge by its very large size:

```
> head(edx)
  userId movieId rating timestamp                         title                        genres
1      1     122      5 838985046              Boomerang (1992)               Comedy|Romance
2      1     185      5 838983525               Net, The (1995)        Action|Crime|Thriller
4      1     292      5 838983421               Outbreak (1995)  Action|Drama|Sci-Fi|Thriller
5      1     316      5 838983392               Stargate (1994)       Action|Adventure|Sci-Fi
6      1     329      5 838983392 Star Trek: Generations (1994) Action|Adventure|Drama|Sci-Fi
7      1     355      5 838984474       Flintstones, The (1994)       Children|Comedy|Fantasy
> nrow(edx)
[1] 9000055
```

Figure 1 : First lines of the Database

This database is not even half as large as the one provided by Netflix, which contained over 20million ratings[2]. But even a 9 million-rows database is heavy and uses quite a lot of memory, nearly 1GB:

```
> object.size(edx)
936968296 bytes
```

Figure 2 : Size of the Database in byte

Never in the HarvardX Data Science course did we train algorithms on so large datasets. And even when taking a smaller data set in the course, Professor Irizarry warned about the inefficiency of doing a simple linear regression to implement a recommendation system:

"However, note that because there are thousands of b's, each movie gets one parameter, one estimate. So the lm function will be very slow here. So we don't recommend running the code we just showed[3]."

[1] https://bits.blogs.nytimes.com/2009/09/21/netflix-awards-1-million-prize-and-starts-a-new-contest/
[2] https://courses.edx.org/courses/course-v1:HarvardX+PH125.8x+2T2018/courseware/, video „Recommendation Systems"
[3] https://courses.edx.org/courses/course-v1:HarvardX+PH125.8x+2T2018/courseware/, video „Building the Recommendation System"

In the following, we will see that indeed, standard machine learning algorithm are not suitable to train our model. It is necessary to follow an approach closer to what is done in the “Recommendation Systems” Chapter in order to obtain a decent RMSE.

The goal of this project is to build a prediction model that minimizes the RMSE on the validation set.

Major Steps

The major steps that were followed to complete this project are the following:

1. Data Visualization: Exploration of the Database to understand what are the key descriptors that will allow us to train the model
2. Data Wrangling: Even though the data is in tidy format already, some reshaping was performed in order to use the most relevant predictors
3. Modeling: Step-by-Step, the algorithm was built seeking for RMSE improvement.
4. Conclusion: The report is concluded with the final obtained model

Methodology and Analysis section

In this section, the conception of the model is explained in detail. First of all, we try to understand what could be the most relevant predictors. Intuitively and as seen in the course, the movie ID and user ID must have an effect on the ratings: Some movies are better than other and users rate differently. The rating average distribution by user and by movie show that this is clearly the case:

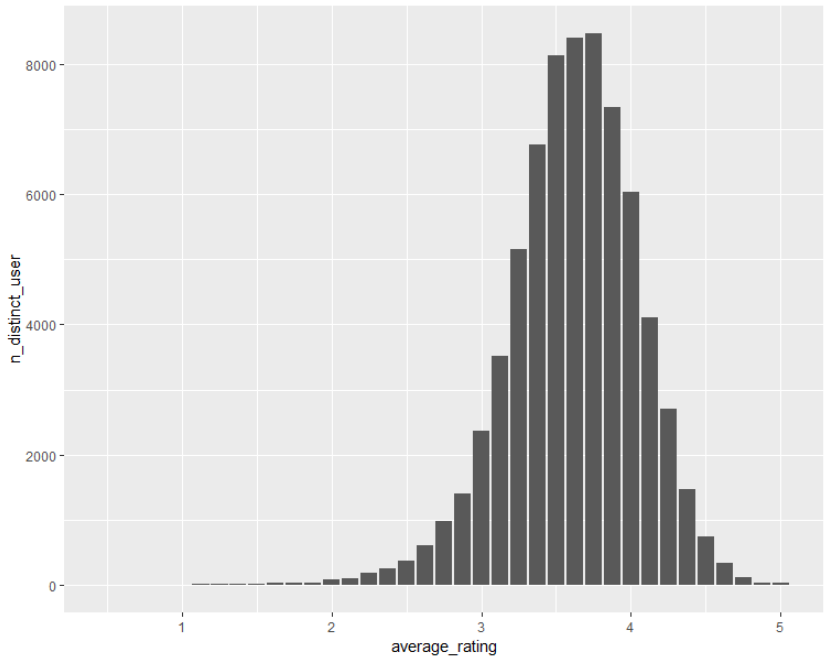

Figure 3 : Rating average distribution by userId

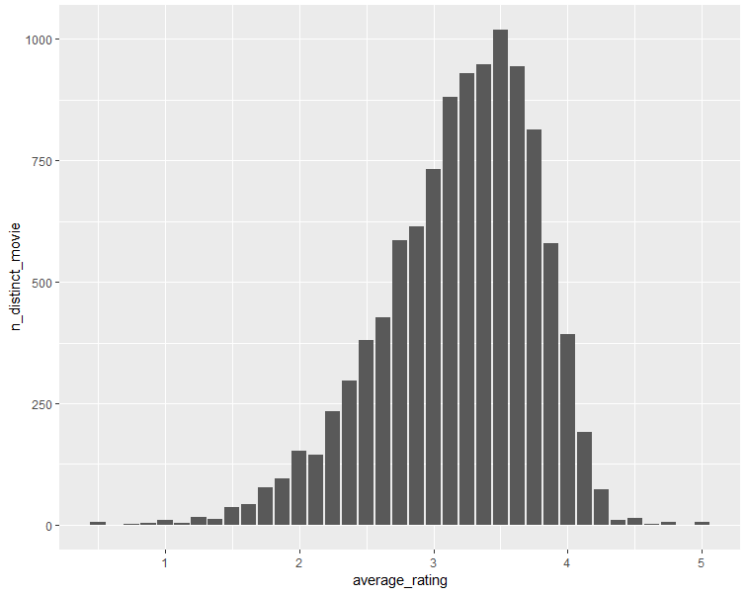

Figure 4 : Rating average distribution by movieId

Until now, the data shown in the charts was directly available from the database. At this stage, reshaping the data is necessary to continue the analysis. First, the year of the movie must also have an impact on the outcome. The year can be extracted from the title of the movie and put in a new column 'year' in a new database 'edx_new':

```
81    #Extraction of the the year from the title
82      edx_new <- edx %>% mutate(year=substr(edx$title, nchar(edx$title)-4, nchar(edx$title)-1))
```

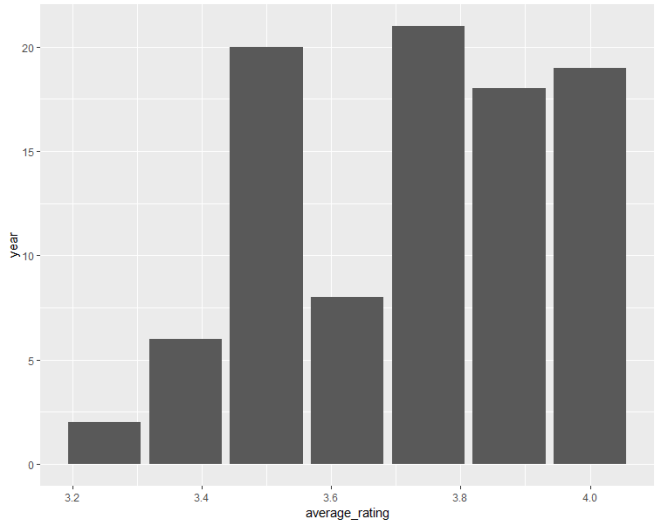

Figure 5: Code to add the year to the database and rating average distribution by year

This distribution happens to be much less significant than the ones of the movies and users.

Intuitively, the genre of the movie must also have an impact. Again, this is confirmed by the below plots:

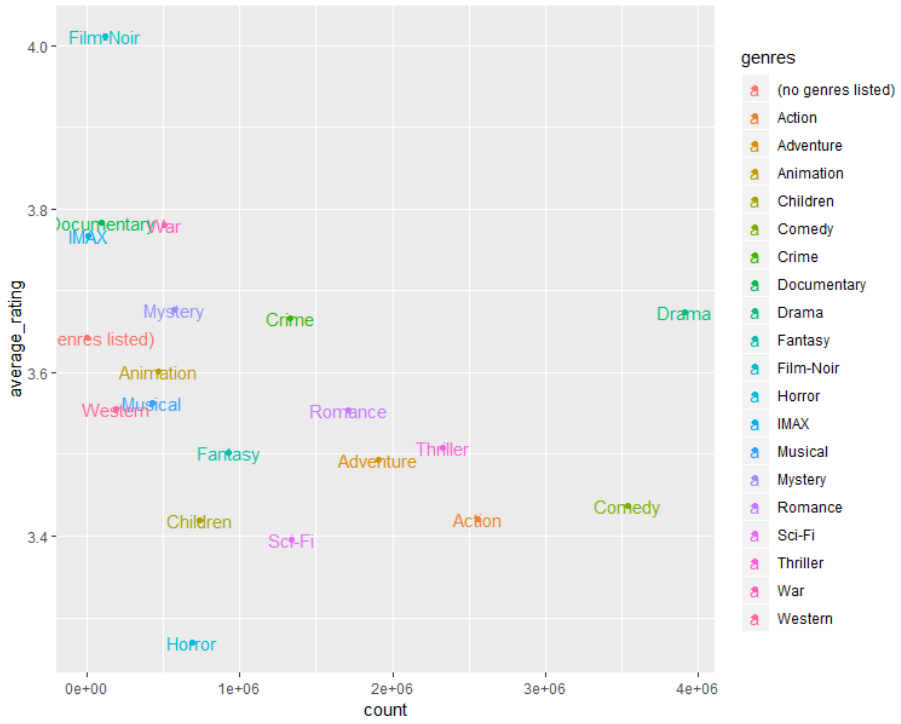

Figure 6 : Average rating and number of movies by genre

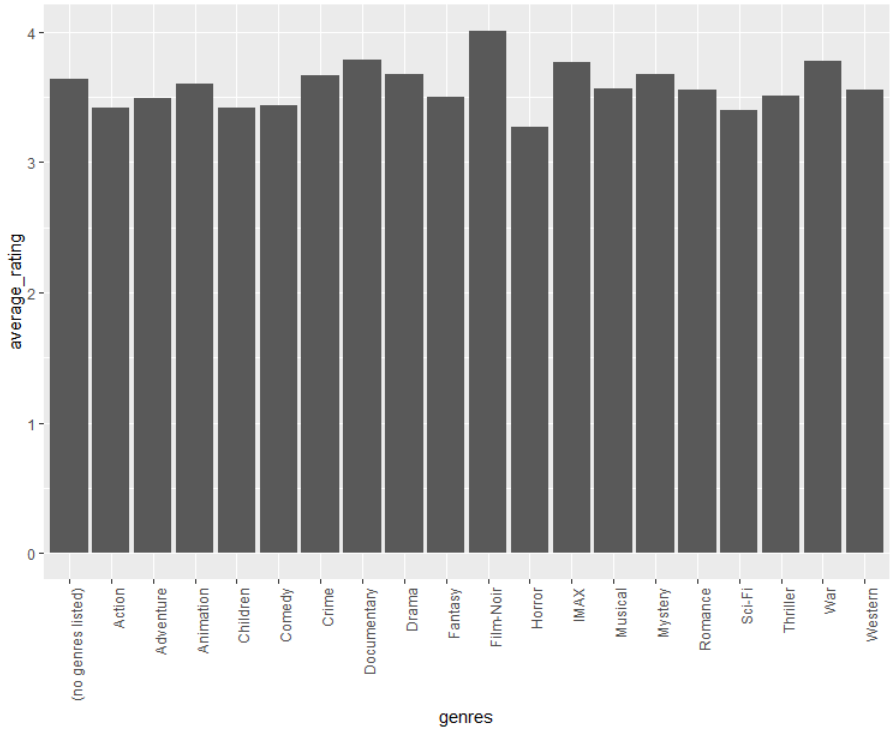

Figure 7 : Average rating by genre

But here again, the distribution is not as spread as for the movie and user effect. This is also logical statistically as we are averaging on groups containing much more ratings, meaning we are closer from the global average.

From this first analysis, it is decided to build successive models that take into account the movie effect, the user effect, the year effect and part of the genre effect. As already introduced in the above, it is necessary to reshape the data in order to take the year and genre into account. The extraction of the year is quite straight away (see Figure 5). For the genre, the problem we have is that in our database, there are nearly 800 genres as several genres are reported for each movie. Therefore, it is decided to modify the database putting the genres in column and, for each movie, assigning a 1 if the movie contains that genre and a 0 otherwise. On my computer, this operation is too heavy to be performed at once and it was necessary to do it on half the database first and the other half and then to bind both again. I finally decided to do it in in four times to lower the chances that the code would crash on someone else's computer. The following lines of code were used:

```
#Reshaping the genre
edx_new_1<- edx[1:2000000,] %>% separate_rows(genres, sep = "\\|") %>% mutate(temp_col=1) %>% spread(genres, temp_col)
edx_new_1[is.na(edx_new_1)] <- 0
```

Figure 8 : Code to reshape the genres

We make sure the database was not modified in the operation, this is the case:

```
> identical(edx$userId, edx_new$userId)
[1] TRUE
>     identical(edx$movieId, edx_new$movieId)
[1] TRUE
>     identical(edx$rating, edx_new$rating)
[1] TRUE
>     identical(edx$timestamp, edx_new$timestamp)
[1] TRUE
>     identical(edx$title, edx_new$title)
[1] TRUE
```

Figure 9: Checking that the new database is identic to the original one

After checking that all userID and movieID that are in the validation set are also in the training set and renaming some names that contain hyphen, the database is ready.

First of all, we quickly check that the standard machine learning regression functions are not appropriate on this dataset. Then, we follow a different approach, close from what is presented in the section “Recommendation systems” of the lecture “Machine Learning”, starting from the global average and taking successively different effects into account, namely the movie effect, the user effect, the year effect and some of the genres effect. Finally, the model is optimized using regularization and rounded to the nearest half.

Results

In this section, we detail the different models that were tested and we report the RMSE that are obtained.

First of all, we compute the RMSE obtained for the global average:

```
> RMSE_Global_Average
[1] 1.061202
```

Figure 10: RMSE global average

Then, we still make a brief try on standard machine learning algorithm.

In spite of Professor Irizarry's warnings, a linear model could still be fitted when using 1 or 2 predictors. However, the result obtain are almost as bad as with the global average, which is not surprising as a linear model is not appropriate for this database.

With knn and rpart, the fit generates an error even with one single predictor.

We are now switching to the method that is expected to deliver the best results. We start from the global average, and, for each movie, we calculate the mean gap to the average. This time, the obtained result is quite promising:

```
#Movie effect
  movie_avgs <- edx_new %>% group_by(movieId) %>% summarize(b_i=mean(rating-Global_Average))
  y_hat_movie_Avg <- Global_Average + validation_new %>% left_join(movie_avgs, by = 'movieId') %>% .$b_i
  RMSE_movie_effect <- RMSE(y_hat_movie_Avg,validation_new$rating)
```

```
> RMSE(y_hat_movie_Avg,validation_new$rating)
[1] 0.9439087
```

Figure 11: RMSE movie effect

After the movie effect, we add a user effect to the model, as from the data visualization, this is the second predictor that has the most spread, significant distribution (after the movieId that was already used). We see that this additional predictor improves the model significantly again:

```
> RMSE_movie_user_effect
[1] 0.8653488
```

Figure 12: RMSE movie and user effect

At this stage, we note that we are already below the targeted RMSE value of 0.875. However, we keep seeking for improvement and we add up the year effect to the model. It is not expected to bring an improvement as high as what the two first predictors did. However, the RMSE value drops again:

```
> RMSE_movie_user_year_effect
[1] 0.8650043
```

Figure 13: RMSE movie, user and year effect

We now test the genre effect. From the database structure that was chosen, the way to work this out is to pick the genres that seem most relevant to be added to the model, and then to see if they bring improvement. From the Figure 6, we first try to add a "Drama effect" as this is the genre that contains the highest number of movies and the gap from the average is significant. The below table confirms that the fact of belonging to the Drama's genre or not has some relevance, even if it is slight:

	Drama	average
	<dbl>	<dbl>
1	0	3.39
2	1	3.67

Figure 14: Average rating for movie belonging/not belonging to the drama genre

Indeed, the model is very, very slightly improved again:

```
> RMSE_movie_user_year_drama_effect
[1] 0.8649613
```

Figure 15: RMSE movie, user, year and drama effect

At this point, we see that adding a Drama effect to the model only brought a very thin improvement of the RMSE. It seems clear that adding up another genre would not improve the result significantly. Instead, we try to use regularization to improve the model a little bit more, which it does indeed:

```
>       rmses[,which.min(rmses[2,])]
[1] 5.0000000 0.8644773
```

Figure 16: RMSE with regularization

Finally, up to know, the result have not be rounded. To be compatible with the output data, result must be rounded to the nearest half. Doing this the final RMSE is obtained, above the decimal value but still below 0.8775:

```
> RMSE_Rounded
[1] 0.8764057
```

Figure 17: RMSE rounded

The below table is a recap of all the RMSE that were obtained along the way, the final RMSE being at the bottom:

method	RMSE
Global Average	1.0612018
LM with movie effect	1.0611815
LM with movie and user effect	1.0611768
Movie Effect Model	0.9439087
Movie & User Effect Model	0.8653488
Movie, User & Year Effect Model	0.8650043
Movie, User, Year and Drama Effect Model	0.8649613
Regularized Model	0.8644773
Final Model	0.8764057

Figure 18: Models tested and associated RMSE

Conclusion

In this project, a recommendation system has been created, without using the typical machine learning algorithm like lm, glm, knn, rpart or random forest, as they were not appropriate for our database. Visualizing the data to find the columns of the database that would be the best candidates to be relevant predictors, a model has been built, adding successive layers to a base given by the global average rating. Finally, regularization was performed and values were rounded to the nearest half to be in the same format that the ratings, giving a final RMSE value of 0.8764057, below the target of 0.8775.

The biggest difficulty that was encountered was to deal with the size of the database. Very often, message errors showed up, similar to this one:

```
Error: cannot allocate vector of size 723.6 Gb
```

Figure 19: Error message example

Even with the final script, this situation can happen, forcing the user to close R and R studio and to rerun all the code again.

In order to improve the result further than what was done in the project, Matrix Factorization could be used. So far, predictors were added up to the model, however, the information of the correlation between predictors was not used. In the video "Matrix Factorization" of the "Machine Learning" lecture, it is shown that the users that like gangster movies usually dislike romantic comedies[4]. To take such effects into account, we can expect that a very powerful computer is necessary as some more basic analysis already caused many problems.

[4] https://courses.edx.org/courses/course-v1:HarvardX+PH125.8x+2T2018/courseware/, video "Matrix Factorization"