Capstone - MovieLens

Pablo Pino Mejias

5 de febrero de 2019

1. Executive summary

This project try to generate a model with enough predictive power to know the rating that a user will give to a movie.

The original project (the most famous at least) that tried to achieve that goal was the **The Netflix Prize** (october 2006). This project was an open competition to predict user ratings for films, based on previous ratings without any other information about the users or films. The goal was to make the company's recommendation engine 10% more accurate.

This document contains an exploratory analysis section in which some characteristics of the data set are shown. This section will also explain the process, techniques and methods that were used to handle the data and to create the predicive model.

The next section shows the results of the previous process and then, the conclusions of the project are given.

Code provided by the edx staff to download an create edx dataset.

```
#Create test and validation sets

# Create edx set, validation set, and submission file

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")

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

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

ratings <- read.table(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                      col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))

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

# Validation set will be 10% of MovieLens data

set.seed(1)
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
```

```r
temp <- movielens[test_index,]

# 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")

# Add rows removed from validation set back into edx set

removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

```r
library(caret)
library(anytime)
library(tidyverse)
library(lubridate)
```

1.1. The dataset

The data set contains 9000055 observations of 6 variables.

- `userId`: Unique identification number given to each user. `numeric` variable
- `movieId`: Unique identification number given to each movie. `numeric` variable.
- `timestamp`: Code that contains date and time in what the rating was given by the user to the specific movie. `integer` variable.
- `title`: Title of the movie. `character` variable.
- `genres`: Motion-picture category associated to the film. `character` variable.
- `rating`: Rating given by the user to the movie. From 0 to 5 *stars* in steps of 0.5. `numeric` variable.

2. Analysis Section

2.1. Data formats.

The `userId` and `movieId` variables are `numeric` columns in the original data set. However it doesn't make sense. The `userId`=2 is not two times the `userId`=1 and the same effect happens with the `movieId` variable. These characteristics are just *labels*, therefore they will be converted to `factor` type to be useful.

Both `movieId` and `title` variables give us the same exact information. They are the **unique identification code** to each film. We could say that these pair of varaibles have 100% of correlation! Only the `movieId` colum will remain. It will be a `factor` too. It optimizes the memory (RAM) usege.

The `timestamp` variable is converted to `POSIXct` type, to be handle correctly as a `date` vector. The year is extracted to the `year` column and the `timestamp` column is dropped.

```r
edx$userId <- as.factor(edx$userId) # Convert `userId` to `factor`.
edx$movieId <- as.factor(edx$movieId) # Convert `movieId` to `factor`.
edx$genres <- as.factor(edx$genres) # Convert `genres` to `factor`.
edx$timestamp <- as.POSIXct(edx$timestamp, origin = "1970-01-01") # Convert `timestamp to `POSIXct`.
```

```r
edx <- edx %>% # It extracts the release year of the movie and creates `year` column.
  mutate(title = str_trim(title)) %>%
  extract(title, c("title_tmp", "year"),
          regex = "^(.*) \\(([0-9 \\-]*)\\)$",
          remove = F) %>%
  mutate(year = if_else(str_length(year) > 4,
                        as.integer(str_split(year, "-",
                                             simplify = T)[1]),
                        as.integer(year))) %>%
  mutate(title = if_else(is.na(title_tmp), title, title_tmp)) %>%
  select(-title_tmp)  %>%
  mutate(genres = if_else(genres == "(no genres listed)",
                          `is.na<-`(genres), genres))

edx <- edx %>% mutate(year_rate = year(timestamp))
  # It extracts the year that the rate was given by the user.

edx <- edx %>% select(-title, -timestamp) # Drop `title` & `timestamp` columns.
edx$genres <- as.factor(edx$genres)
```

```r
head(edx)
```

```
##   userId movieId rating year                        genres year_rate
## 1      1     122      5 1992                Comedy|Romance      1996
## 2      1     185      5 1995         Action|Crime|Thriller      1996
## 3      1     292      5 1995  Action|Drama|Sci-Fi|Thriller      1996
## 4      1     316      5 1994       Action|Adventure|Sci-Fi      1996
## 5      1     329      5 1994 Action|Adventure|Drama|Sci-Fi      1996
## 6      1     355      5 1994       Children|Comedy|Fantasy      1996
```

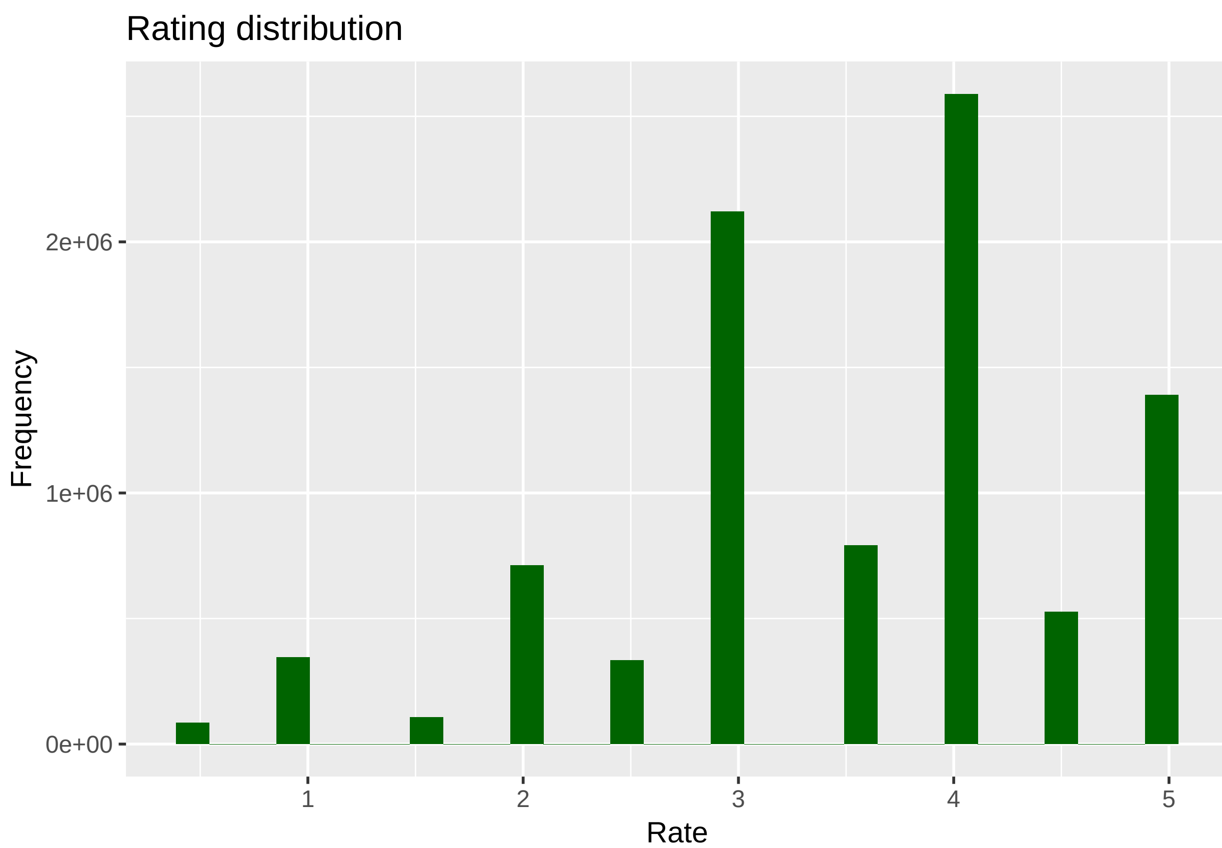
2.2. Exploratory Analysis.

The target is to create a model capable of predicting the variable `rating`.

```r
summary(edx$rating)
```

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

```r
edx %>% ggplot(aes(rating)) +
            geom_histogram(fill = "darkgreen") +
            labs(title = "Rating distribution",
                 x = "Rate",
                 y = "Frequency")
```



We can see that our `rating` variable has a left-skewed distribution. It's interesting that there are more *good* ratings than *bad* ratings. That could be explain by the fact that people want to recommend a film when they liked it, but we could just suppose that, because of we don't have the data here to prove it.

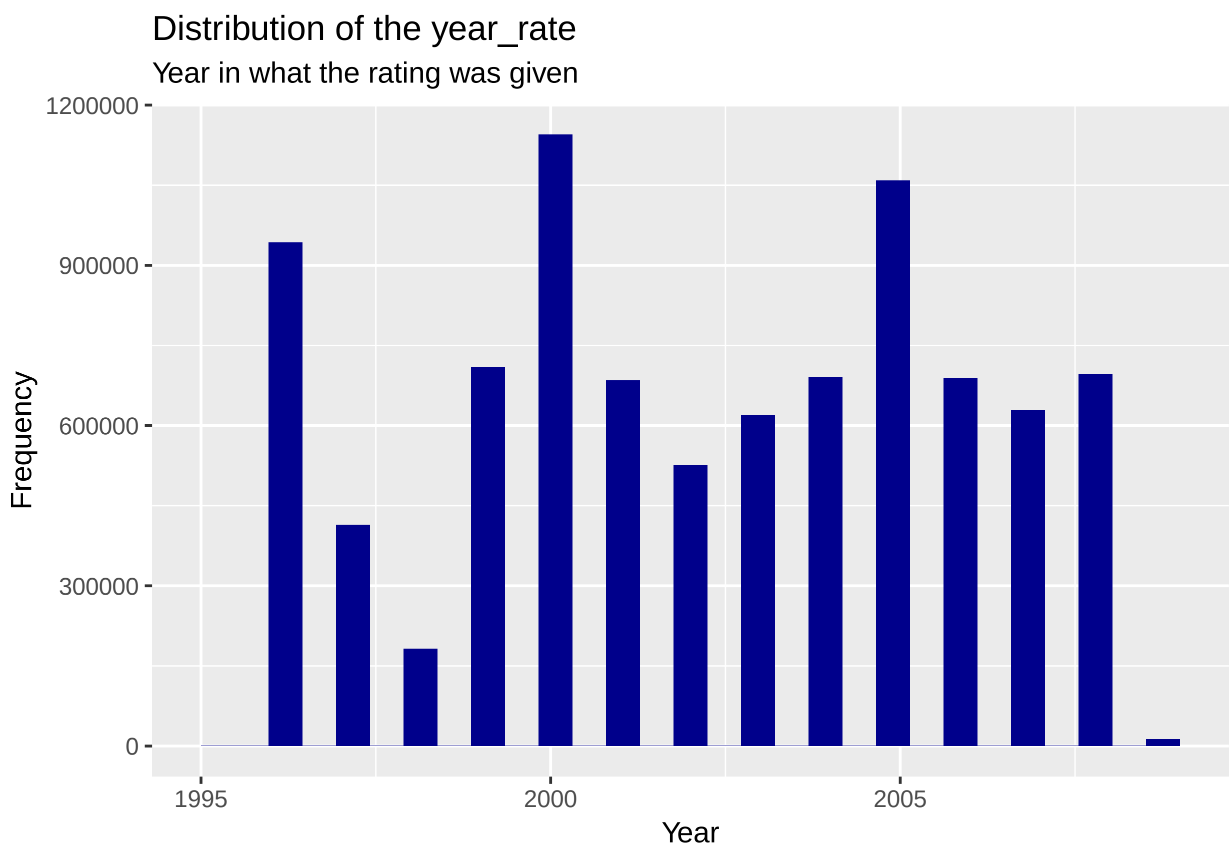
What about the users and movies in the data?

```
edx %>%
  summarize(Unique_Users = n_distinct(userId),
            Unique_Movies = n_distinct(movieId),
            Unique_Genres = n_distinct(genres))
```

```
##   Unique_Users Unique_Movies Unique_Genres
## 1        69878         10677           797
```

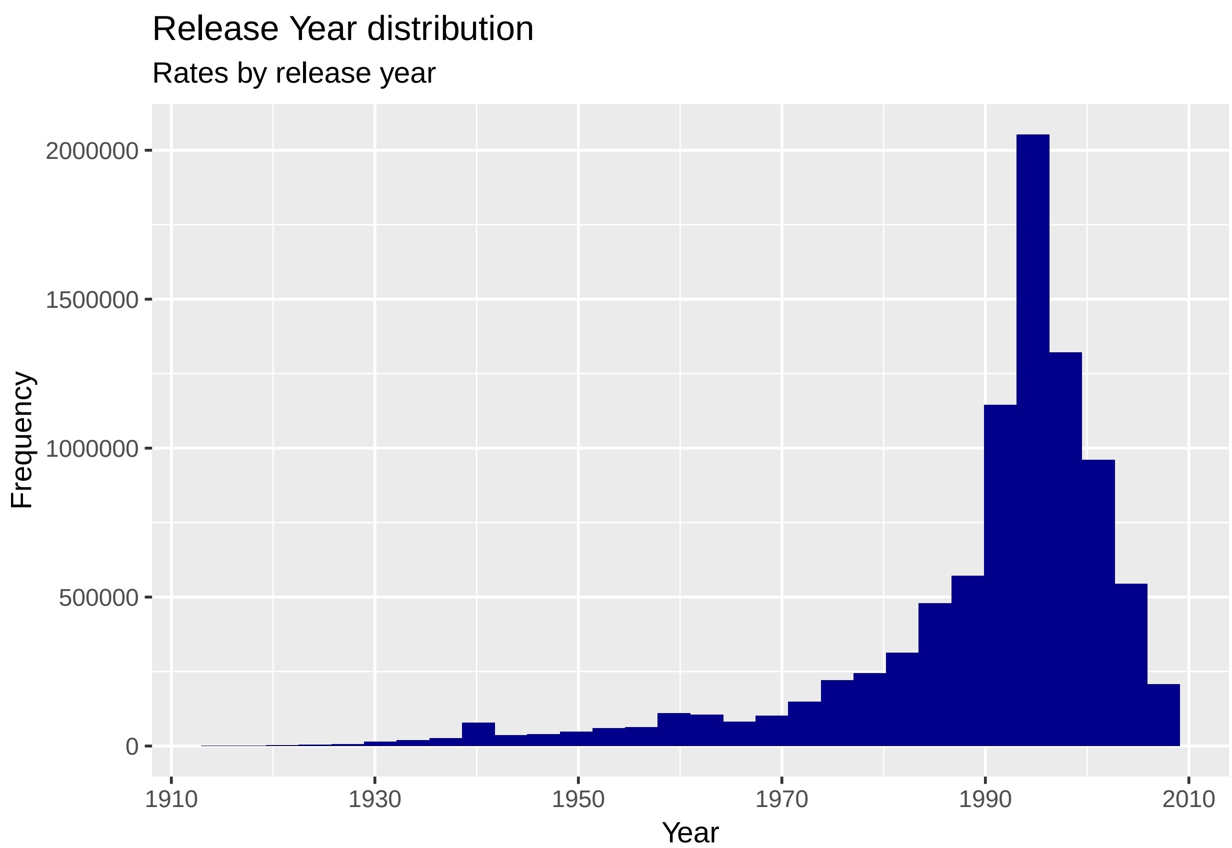
We observe that there are 69878 unique users given ratings to 10677 different films. It's good to remember that the *unique genres* were counted as `factor` with no previous separation so, `Drama` and `Comedy|Drama` are counted as 2 different genres.

```
edx %>% ggplot(aes(year_rate)) +
  geom_histogram(fill = "darkblue") +
  labs(title = "Distribution of the year_rate",
       subtitle = "Year in what the rating was given",
       x = "Year",
       y = "Frequency")
```



We can observe that the years 1998 and 2006 have fewer observations. The quantity of ratings given by year is irregular. That could affect the performance of the model.

```
edx %>% ggplot(aes(year)) +
  geom_histogram(fill = "darkblue") +
  labs(title = "Release Year distribution",
       subtitle = "Rates by release year",
       x = "Year",
       y = "Frequency")
```



The frequency of ratings by *release year* of the films has a clear left skewed distribution. The most of those year are between 1990 and 2009.

```
g <- edx %>%
  select(genres, rating) %>%
  group_by(genres) %>%
  summarize(mean = mean(rating), median = median(rating), n = n()) %>%
  arrange(desc(mean)) %>%
  head(20)

print(g)
```

```
## # A tibble: 20 x 4
##    genres                                     mean median     n
##    <fct>                                     <dbl>  <dbl> <int>
##  1 Animation|IMAX|Sci-Fi                      4.71    5       7
##  2 Drama|Film-Noir|Romance                    4.30    4.5  2989
##  3 Action|Crime|Drama|IMAX                    4.30    4.5  2353
##  4 Animation|Children|Comedy|Crime            4.28    4.5  7167
##  5 Film-Noir|Mystery                          4.24    4    5988
##  6 Crime|Film-Noir|Mystery                    4.22    4    4029
##  7 Film-Noir|Romance|Thriller                 4.22    4    2453
##  8 Crime|Film-Noir|Thriller                   4.21    4    4844
##  9 Crime|Mystery|Thriller                     4.20    4   26892
## 10 Action|Adventure|Comedy|Fantasy|Romance    4.20    4   14809
## 11 Crime|Thriller|War                         4.17    4    4595
## 12 Film-Noir|Mystery|Thriller                 4.16    4    4011
```

```
## 13 Adventure|Drama|Film-Noir|Sci-Fi|Thriller           4.15     4   13957
## 14 Adventure|Animation|Children|Comedy|Sci-Fi           4.15     4    3529
## 15 Adventure|Comedy|Romance|War                         4.13     4    5223
## 16 Adventure|Animation|Children|Comedy|Romance|Sci-Fi  4.12     4    1254
## 17 Comedy|Drama|Romance|Sci-Fi                          4.12     4    7593
## 18 Action|Crime|Drama|Film-Noir|Mystery                 4.12     4    1103
## 19 Animation|Drama|War                                  4.11     4    1039
## 20 Action|Drama|Thriller|War                            4.11     4     480
```

The top 10 **genres** listed above have the highest **mean**. The first place "Animation|IMAX|Sci-Fi" is clearly not significant, because of the only 7 observations. This *genre* will be eliminated below. **Drama** is present in the 2° and 3° place.

```
g[2:nrow(g), ] %>%
  separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  head(20)
```

```
## # A tibble: 20 x 4
## # Groups:   genres [11]
##    genres     mean median     n
##    <chr>     <dbl>  <dbl> <int>
##  1 Drama      4.30    4.5  2989
##  2 Film-Noir  4.30    4.5  2989
##  3 Romance    4.30    4.5  2989
##  4 Action     4.30    4.5  2353
##  5 Crime      4.30    4.5  2353
##  6 Drama      4.30    4.5  2353
##  7 IMAX       4.30    4.5  2353
##  8 Animation  4.28    4.5  7167
##  9 Children   4.28    4.5  7167
## 10 Comedy     4.28    4.5  7167
## 11 Crime      4.28    4.5  7167
## 12 Film-Noir  4.24    4     5988
## 13 Mystery    4.24    4     5988
## 14 Crime      4.22    4     4029
## 15 Film-Noir  4.22    4     4029
## 16 Mystery    4.22    4     4029
## 17 Film-Noir  4.22    4     2453
## 18 Romance    4.22    4     2453
## 19 Thriller   4.22    4     2453
## 20 Crime      4.21    4     4844
```

The genres associated with the highest mean are "Drama", "Film-Noir" and "Romance". The **difference** with the *second section* (mean = 4.28) in that ranking is almost zero.

```
edx %>%
  select(genres, rating) %>%
  group_by(genres) %>%
  summarize(mean = mean(rating), median = median(rating), n = n()) %>%
  arrange(desc(mean)) %>%
  separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  head(20) %>%
  group_by(genres) %>%
  summarise(appearances = sum(n)) %>%
```

```r
  arrange(desc(appearances)) %>%
  head(10)
```

```
## # A tibble: 10 x 2
##    genres    appearances
##    <chr>           <int>
##  1 Film-Noir       15459
##  2 Crime           13549
##  3 Mystery         10017
##  4 Animation        7174
##  5 Children         7167
##  6 Comedy           7167
##  7 Drama            5342
##  8 Romance          2989
##  9 IMAX             2360
## 10 Action           2353
```

In the complete `edx` dataset the genres **Film-Noir**, **Crime** and **Mystery** are the top 3 in appearences. Justn in the 5° and 6° place some *happy* genre appears!

3. The Model

3.1. Train and Test set

First at all, the `train` and `test` set are created.

```r
edx <- edx %>% select(userId, movieId, rating)

test_index <- createDataPartition(edx$rating, times = 1, p = .2, list = F)
    # Create the index

  train <- edx[-test_index, ] # Create Train set
  test <- edx[test_index, ] # Create Test set
  test <- test %>% # The same movieId and usersId appears in both set. (Not the same cases)
    semi_join(train, by = "movieId") %>%
    semi_join(train, by = "userId")
```

```r
dim(train)
```

```
## [1] 7200043       3
```

```r
dim(test)
```

```
## [1] 1799983       3
```

3.2 Baseline *model*

The most basic model is generated when we are just considering the most common rating from the *train* set to be predicted into the test set. This is the **baseline** model.

```r
mu_hat <- mean(train$rating) # Mean accross all movies.
RMSE_baseline <- RMSE(test$rating, mu_hat) # RMSE in test set.
RMSE_baseline
```

```
## [1] 1.060096
```

Now, we have the RMSE to be *beaten* by our model.

```r
rmse_table <- data_frame(Method = "Baseline", RMSE = RMSE_baseline)
rmse_table %>% knitr::kable(caption = "RMSEs")
```

Table 1: RMSEs

Method	RMSE
Baseline	1.060096

We can observe that the RMSE of the most basic model is 1.0600964. It's bigger than 1! In this context, this is a very bad model.

3.3 User and Movie effect Model

The next step is going to try to get a new model with a better RMSE.

We are considering the *user effect* (u_i) and the *movie effect* (m_i) as predictors. Therefore, we are generating the next model to predict `rating` ($\hat{y}_i$):

$$\hat{y}_i = u_i + m_i + \varepsilon$$

```r
mu <- mean(train$rating)

movie_avgs <- train %>%
  group_by(movieId) %>%
  summarize(m_i = mean(rating - mu))

user_avgs <- test %>%
  left_join(movie_avgs, by = "movieId") %>%
  group_by(userId) %>%
  summarize(u_i = mean(rating - mu - m_i))

predicted_ratings <- test %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  mutate(pred = mu + m_i + u_i) %>% .$pred

model_RMSE <- RMSE(predicted_ratings, test$rating)
model_RMSE
```

```
## [1] 0.843668
```

```r
rmse_table <- rbind(rmse_table,
                    data_frame(Method = "User & Movie Effect", RMSE = model_RMSE))

rmse_table %>% knitr::kable(caption = "RMSEs")
```

Table 2: RMSEs

Method	RMSE
Baseline	1.060096
User & Movie Effect	0.843668

We've got obtained a better RMSE. Now it is time to make predictions on unseeing data.

3.4 User and Movie effect Model on *validation* data

First at all, the *validation* data set needs to be handled the same as the *train* data set was handled.

```
validation <- validation %>% select(userId, movieId, rating)

validation$userId <- as.factor(validation$userId)
validation$movieId <- as.factor(validation$movieId)

validation <- validation[complete.cases(validation), ]
```

Now, we are ready to make predictions.

```
predicted_val <- validation %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  mutate(pred = mu + m_i + u_i) %>% .$pred

val_RMSE <- RMSE(predicted_val, validation$rating, na.rm = T)
val_RMSE
```

```
## [1] 0.8817798
```

```
rmse_table_val <- data_frame(Method = "User & Movie Effect on validation", RMSE = val_RMSE)
rmse_table_val %>% knitr::kable(caption = "RMSEs on validation data set")
```

Table 3: RMSEs on validation data set

Method	RMSE
User & Movie Effect on validation	0.8817798

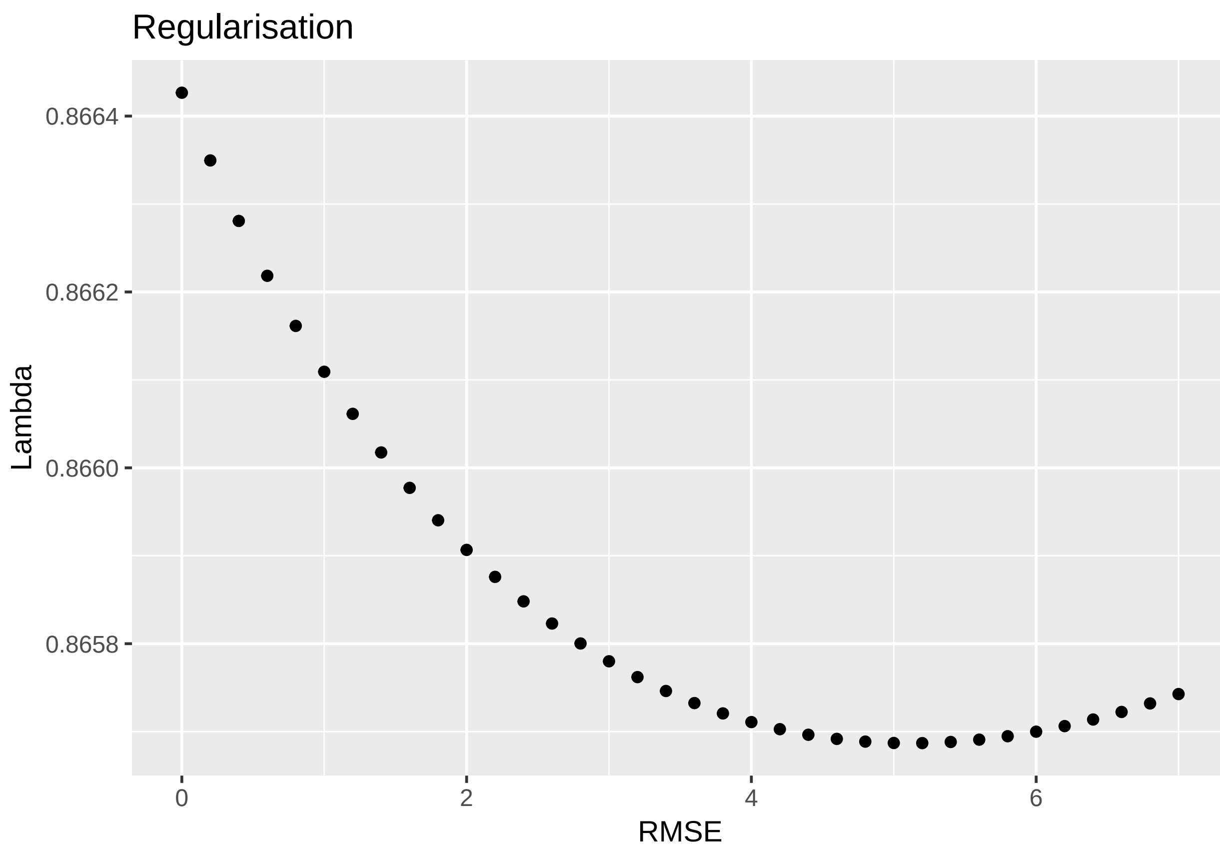
We can see above that this RMSE is higher than the RMSE on the test set. This is highly probable, given that this is unseeing data. The good thing is that the difference is just 0.0381118. Now, let's see if *regularisation* give us better results.

3.5. Regularisation

The regularisation process will evaluate different values for λ, delivering to us the corresponding RMSE.

```
lambda_values <- seq(0, 7, .2)

RMSE_function_reg <- sapply(lambda_values, function(l){

  mu <- mean(train$rating)

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

  u_i <- train %>%
    left_join(m_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(u_i = sum(rating - m_i - mu)/(n()+l))

  predicted_ratings <- test %>%
    left_join(m_i, by = "movieId") %>%
```

```r
    left_join(u_i, by = "userId") %>%
    mutate(pred = mu + m_i + u_i) %>% .$pred

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

qplot(lambda_values, RMSE_function_reg,
      main = "Regularisation",
      xlab = "RMSE", ylab = "Lambda") # lambda vs RMSE
```




```r
lambda_opt <- lambda_values[which.min(RMSE_function_reg)]
lambda_opt # Lambda which minimizes RMSE
```

```
## [1] 5.2
```

```r
rmse_table <- rbind(rmse_table,
                    data_frame(Method = "User & Movie Effect Regularisation",
                               RMSE = min(RMSE_function_reg)))
rmse_table %>% knitr::kable(caption = "RMSEs")
```

Table 4: RMSEs

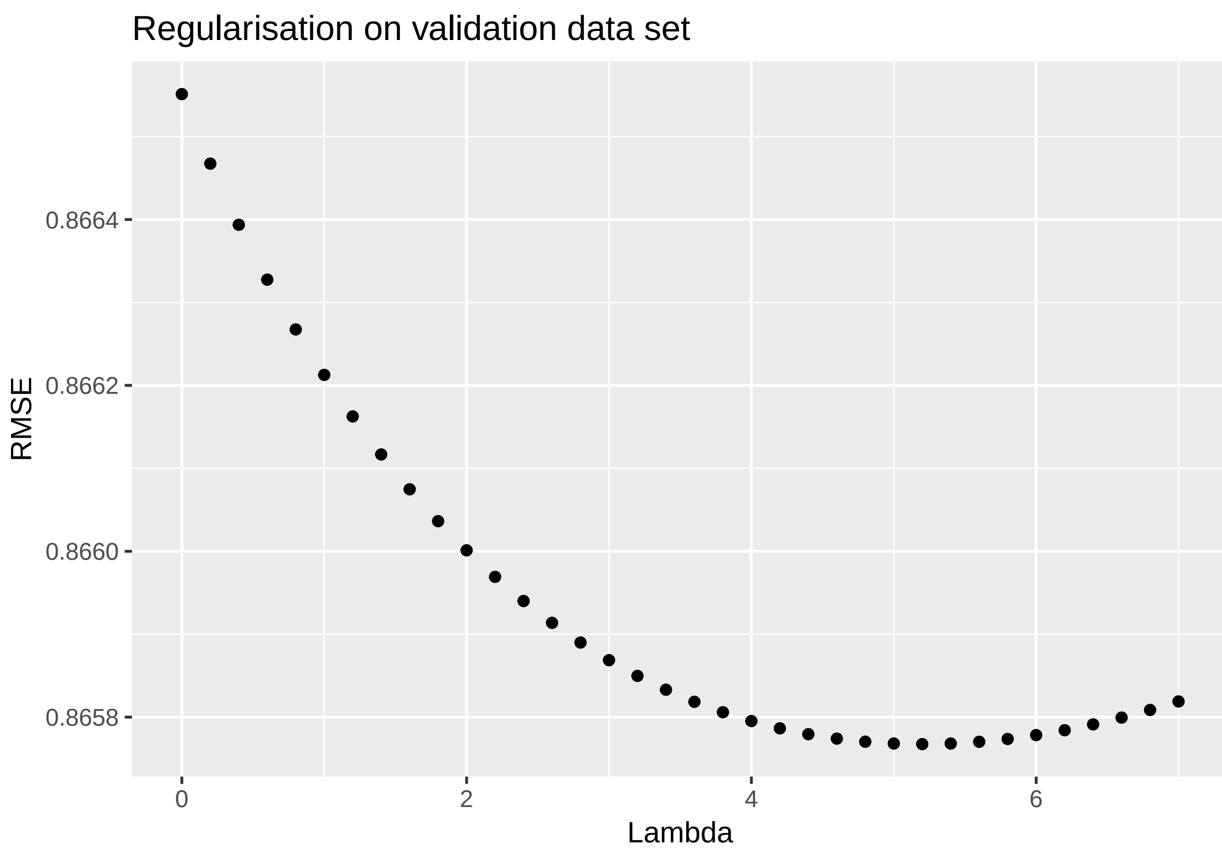
Method	RMSE
Baseline	1.060096
User & Movie Effect	0.843668
User & Movie Effect Regularisation	0.865687

The *regularisation* give as a higher RMSE than the first "User & Movie Effect" model. This is unexpected

3.6 Regularisation on *validation* data set

It is time to see what is the performance of the *regularisation* on the validation data set.

```
RMSE_function_val_reg <- sapply(lambda_values, function(l){

  mu <- mean(train$rating)

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

  u_i <- train %>%
    left_join(m_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(u_i = sum(rating - m_i - mu)/(n()+l))

  predicted_val_reg <- validation %>%
    left_join(m_i, by = "movieId") %>%
    left_join(u_i, by = "userId") %>%
    mutate(pred = mu + m_i + u_i) %>% .$pred

  return(RMSE(predicted_val_reg, validation$rating, na.rm = T))
})

qplot(lambda_values, RMSE_function_val_reg,
      main = "Regularisation on validation data set",
      xlab = "Lambda", ylab = "RMSE")
```




```r
lambda_opt_reg <- lambda_values[which.min(RMSE_function_val_reg)]
lambda_opt_reg # Lambda which minimizes RMSE
```

```
## [1] 5.2
```

```r
min_rmse <- min(RMSE_function_val_reg) # Best RMSE
min_rmse
```

```
## [1] 0.8657675
```

```r
rmse_table_val <- rbind(rmse_table_val,
                    data_frame(Method = "User & Movie Effect Reg. on validation",
                               RMSE = min(RMSE_function_val_reg)))
rmse_table_val %>% knitr::kable(caption = "RMSEs on validation data set")
```

Table 5: RMSEs on validation data set

Method	RMSE
User & Movie Effect on validation	0.8817798
User & Movie Effect Reg. on validation	0.8657675

4. Results

```r
rbind(rmse_table, rmse_table_val) %>% knitr::kable(caption = "RMSEs Summary")
```

Table 6: RMSEs Summary

Method	RMSE
Baseline	1.0600964
User & Movie Effect	0.8436680
User & Movie Effect Regularisation	0.8656870
User & Movie Effect on validation	0.8817798
User & Movie Effect Reg. on validation	0.8657675

We can observe that the better RMSE is obtained from the *User & Movie Effect* model. However, this RMSE *only* obtained on the *test* set. When we move to the *validation* data set, we obtain the worse RMSE (ignoring the baseline).

Considering that we must trust more in the performance of the model when we predict from unseeing data, we can say that the RMSE that results from the *User & Movie Effect with Regularisation on validation* (the last line in the table above) is our definitive model. This RMSE is obtained when $\lambda = 5$ which permit us to achieve **RMSE equal to 0.8657675.**

5. Conclusion

The variables `userId` and `movieId` have sufficient predictive power to permit us to predict how a user will rate a movie. This tell us that we could make better recommendations about movie to specific users of the streaming service. Therefore, the user could decide to spend more time using the service.

The RMSE equal to 0.8657675 is pretty acceptable considering that we have few predictors, but both *User* and *Movie* effects are power enough to predict the `rating` that will be given to a movie, by a specific user.

Final thoughts

The objective of data science is to transform data into information and using machine learning and, if it is possible, to find the best model to predict what we desire to predict. There are a lot of variables that could help us to achieve this. One of those variables is the hardware (capacity) of our machine.

In my specific situation, I always try differents models as *cart*, *random forest*,*gbm*, *neural network* and others. This time it wasn't possible because of the amount of data. 10 millions of rows! It wasn't even possible to sparse the `genres` column!

One option is to "cut" the data frame, but not all the cases will be *trained*.

It's a shame that the **technical capacity** of the machines of the students wasn't considering at the planning phase of this final project. The method used in this project was the only one that was possible to run without crashing my RAM. From the forums of the course, we know that there a lot oof students dealing with the same *issue*.

It's a shame, because the process of a data science project is really beatiful.

Happy learning!

My "movieLens Github repository" is **in this link**