Data Science: Capstone Movielens Project

David Sanchez Plana

18 junio, 2020

Contents

Executive summary **3**

1 Introduction **3**

1.1 **Input Data** 3

1.2 **Data Exploration and Visualization** 5

2 Methods and Analysis **12**

2.1 **First method: Regularized User and Movie effect** 13

2.2 **Second method: Matrix factorization** 14

3 Results **14**

3.1 **Method 1 results** 14

3.2 **Method 2 results** 16

4 Conclusions **18**

4.1 **Limitations** 19

4.2 **Other possible models / Future work** 19

References **22**

Executive summary

This is a project report for the *Data Science: Capstone* course. It is asked to apply machine learning basis to a provided dataset, with the purpose of predict movie ratings and get a root mean square error (from now on RMSE) as low as possible. To make this report, *R Markdown* guide has been followed.

Applying matrix factorization we achieve a RMSE lower than **0.8**.

1 Introduction

Nowadays, most large companies focus much of their effort on anticipating two important aspects: what the customer wants and what the demand will be. This is a differentiating factor of great added value since it allows you to adjust your productivity and promote customer loyalty.

That is why in recent years, concepts such as machine learning and recommendation systems are current issues and a factor to consider.

An example of this, exposed in previous courses, was the contest that Netflix carried out in order to improve its movie recommendation system. Taking this premise, a predictive system will be created to give automatic movie ratings, and its accuracy and correct operation will be verified.

1.1 Input Data

On one hand, there is the **edx** data set. It will be divided into two data sets, training set and test set, to design and test the algorithm.

On the other hand, there is the **validation** data set. It will be used for evaluating the error of your final algorithm.

The following provided code generates the datasets.

```
#################################
# Create edx set, validation set
#################################

# Note: this process could take a couple of minutes
if(!require(dplyr)) install.packages("dplyr",
                                     repos = "http://cran.us.r-project.org")
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(dslabs)) install.packages("dslabs",
                                     repos = "http://cran.us.r-project.org")
```

```r
if(!require(lubridate)) install.packages("lubridate",
                                         repos = "http://cran.us.r-project.org")
if(!require(ggplot2)) install.packages("ggplot2",
                                         repos = "http://cran.us.r-project.org")
if(!require(recommenderlab)) install.packages("recommenderlab",
                                         repos = "http://cran.us.r-project.org")
if(!require(recosystem)) install.packages("recosystem",
                                         repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table",
                                         repos = "http://cran.us.r-project.org")
if(!require(stringr)) install.packages("stringr",
                                         repos = "http://cran.us.r-project.org")

# Load the libraries
library(dplyr)
library(tidyverse)
library(caret)
library(dslabs)
library(lubridate)
library(ggplot2)
library(recommenderlab)
library(recosystem)
library(data.table)
library(stringr)


# 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 <- fread(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(movieId),
                                             title = as.character(title),
                                             genres = as.character(genres))

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

```r
# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding")
# if using R 3.5 or earlier, use `set.seed(1)` instead
test_index <- createDataPartition(y = movielens$rating,
                                  times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
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)
```

This piece of code that is provided to us generates concise datasets, so there would be no need to apply **data cleaning**.

Once we have the data we must split edx data into training and test sets, in order to validate the models. 80% of the edx data will make up the training set while remaining 20% will form the test set.

```r
#generate training and test sets
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2,
                                  list = FALSE)
train_set <- edx[-test_index,]
temp <- edx[test_index,]
```

Finally, we need to make sure that users and movies in test set are also in train set, like we did with validation and edx.

```r
test_set <- temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)
rm(test_index, temp, removed)
```

1.2 Data Exploration and Visualization

With RStudio we can see the dimensions and the different variables that are in both data sets. However, we can explore the data as follows:

- *edx* fields (wich are the same as in validation dataset):

```r
head(edx)
```

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

- Number of ratings and fields in *edx* and *validation* datasets:

```r
dim(edx)
```

```
## [1] 9000055       6
```

```r
dim(validation)
```

```
## [1] 999999      6
```

- Number of different movies in *edx* and *validation* datasets:

```r
n_distinct(edx$movieId)
```

```
## [1] 10677
```

```r
n_distinct(validation$movieId)
```

```
## [1] 9809
```

- Number of different users in *edx* and *validation* datasets:

```
n_distinct(edx$userId)
```

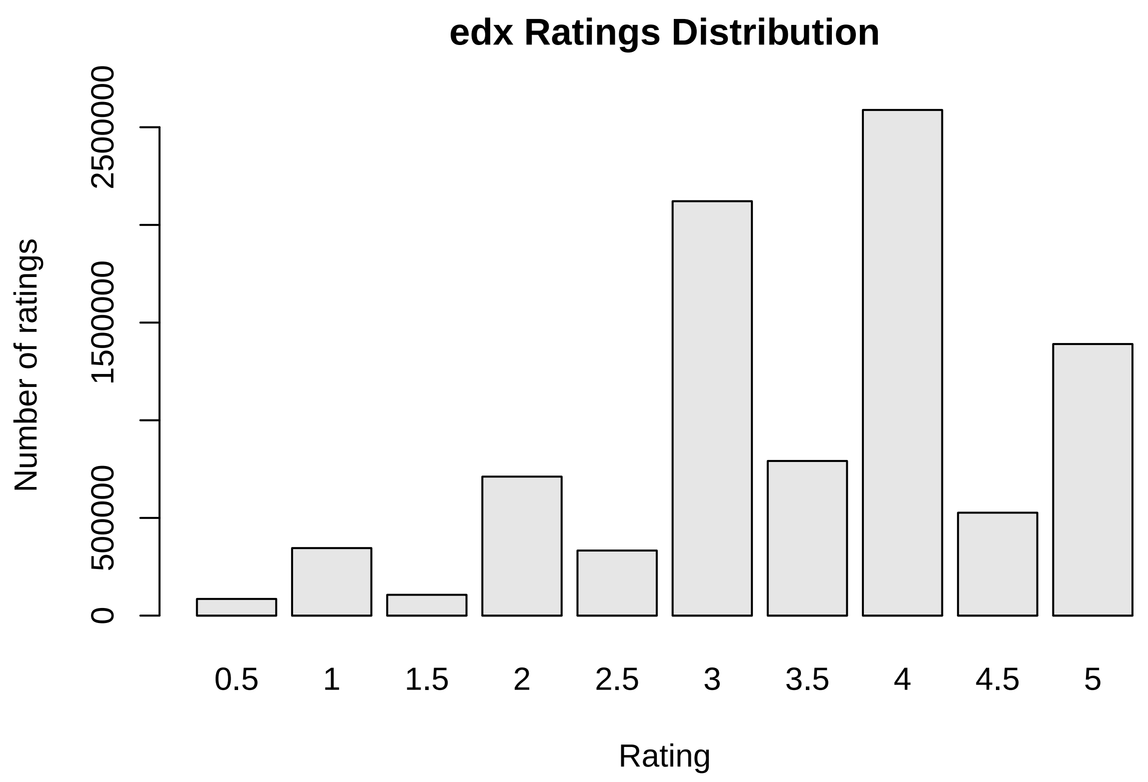
```
## [1] 69878
```

```
n_distinct(validation$userId)
```

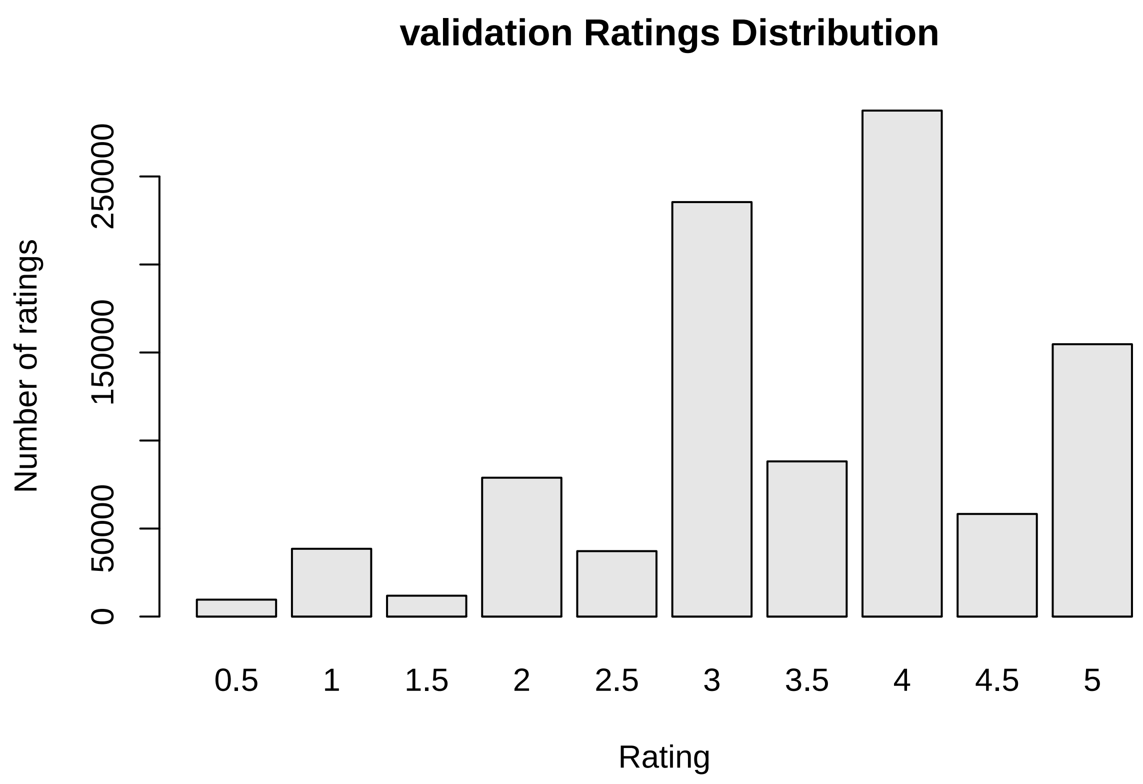
```
## [1] 68534
```

- Distribution of ratings in *edx* and *validation* datasets:

```
ratings_dis_edx<-edx %>% group_by(rating) %>% summarize(count = n())

barplot(t(as.matrix(ratings_dis_edx)),
        beside=FALSE,main="edx Ratings Distribution",
        xlab="Rating",
        ylab="Number of ratings",
        axisnames = TRUE,
        names.arg=ratings_dis_edx$rating
)
```



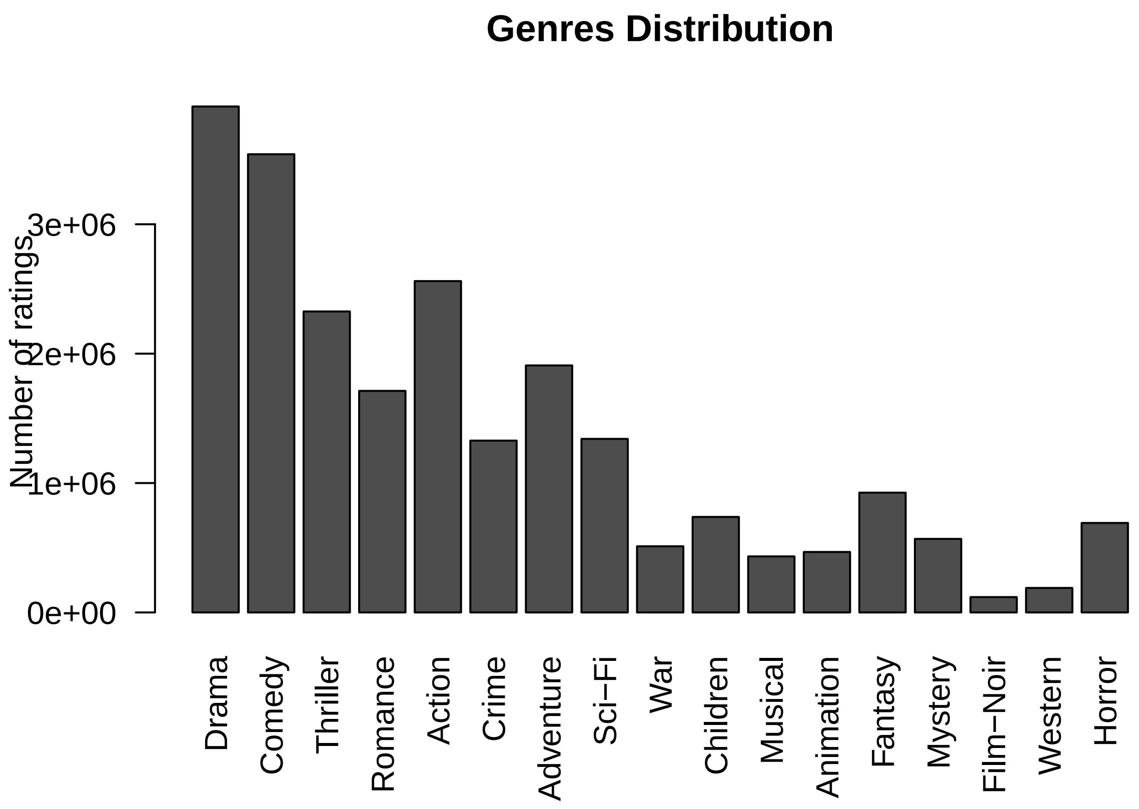
```
ratings_dis_val<-validation %>% group_by(rating) %>% summarize(count = n())

barplot(t(as.matrix(ratings_dis_val)),
        beside=FALSE,main=" validation Ratings Distribution",
        xlab="Rating",
        ylab="Number of ratings",
        axisnames = TRUE,
        names.arg=ratings_dis_val$rating
)
```



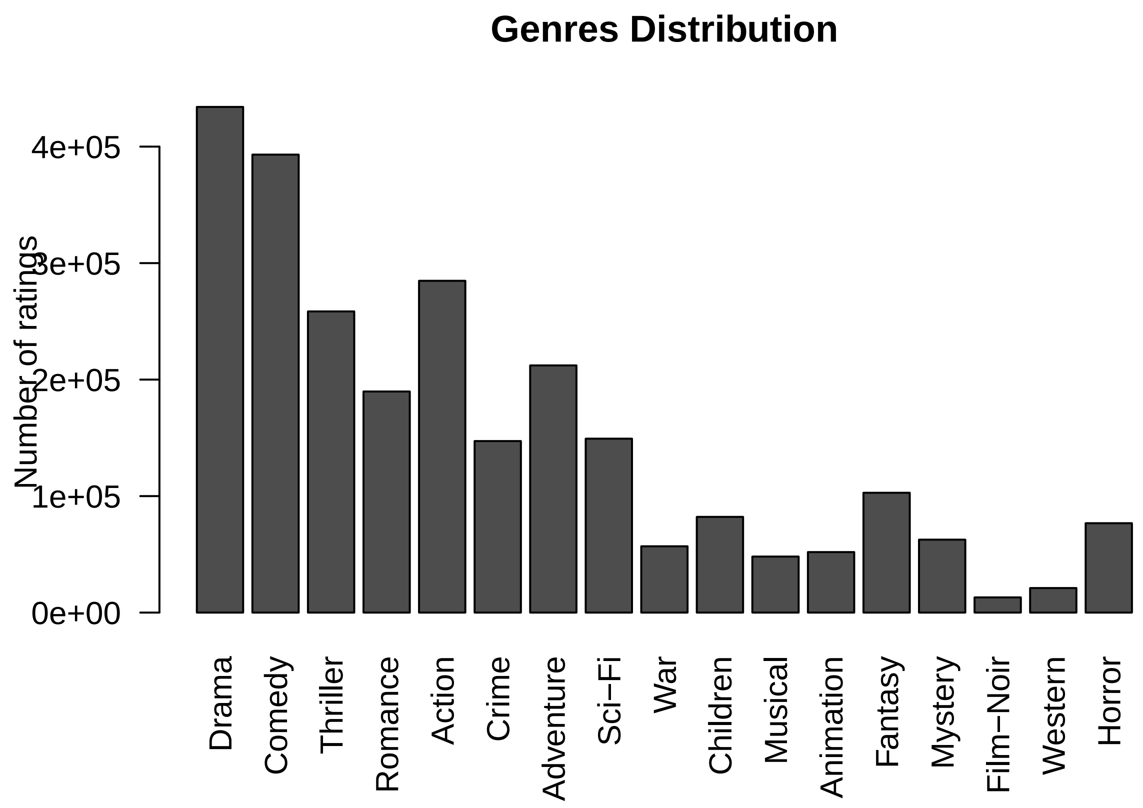
As we can see, users usually give round rates.

- Distribution of genres in *edx* and *validation* datasets:

```
p_gen = c("Drama", "Comedy", "Thriller", "Romance", "Action", "Crime",
          "Adventure", "Sci-Fi", "War", "Children", "Musical", "Animation",
          "Fantasy", "Mystery", "Film-Noir",
          "Western", "Horror")
genres_edx<-sapply(p_gen, function(g) {
  sum(str_detect(edx$genres, g))
})
genres_edx_m<-as.matrix(genres_edx)
barplot(t(genres_edx_m), beside=FALSE,main="Genres Distribution",
        ylab="Number of ratings",
        axisnames = TRUE,
        las=2
)
```



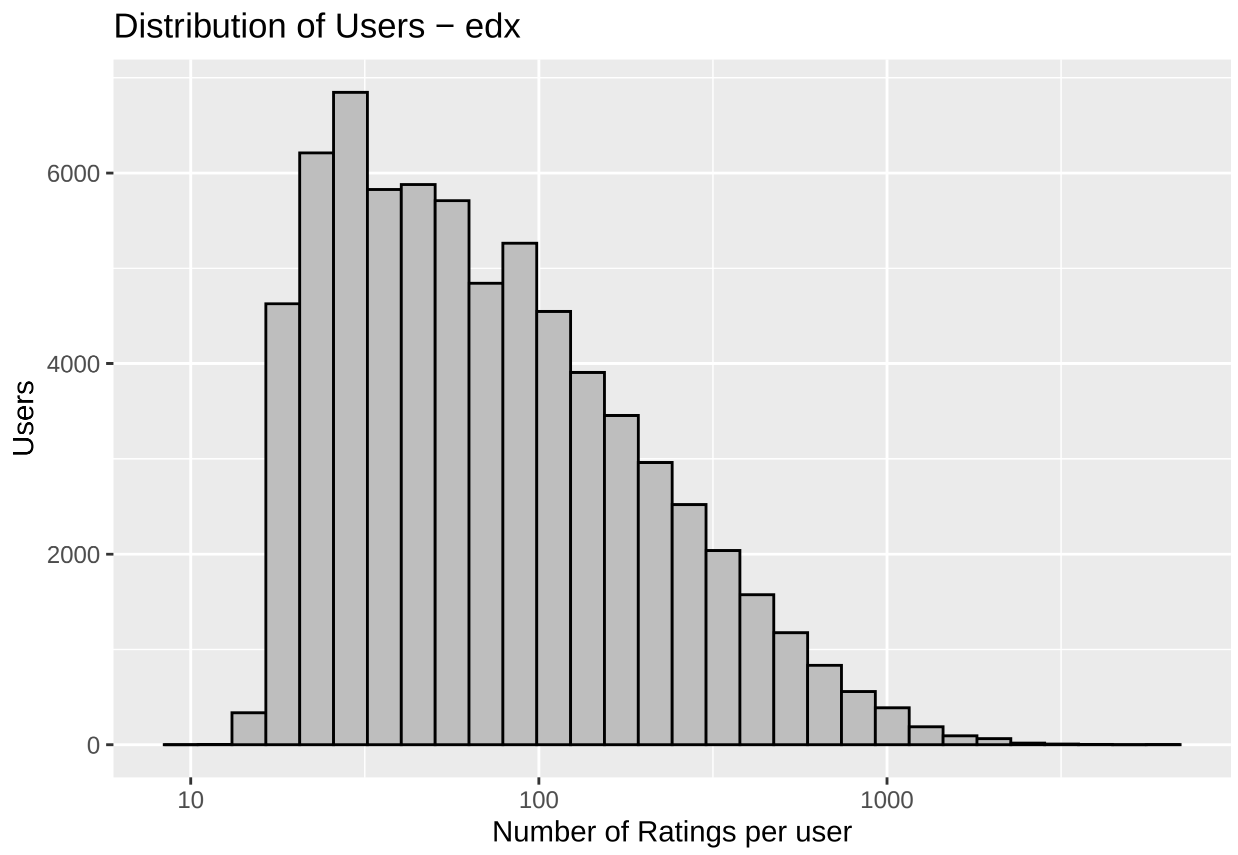
```
genres_val<-sapply(p_gen, function(g) {
  sum(str_detect(validation$genres, g))
})
genres_val_m<-as.matrix(genres_val)
barplot(t(genres_val_m), beside=FALSE,main="Genres Distribution",
        ylab="Number of ratings",
        axisnames = TRUE,
        las=2
)
```



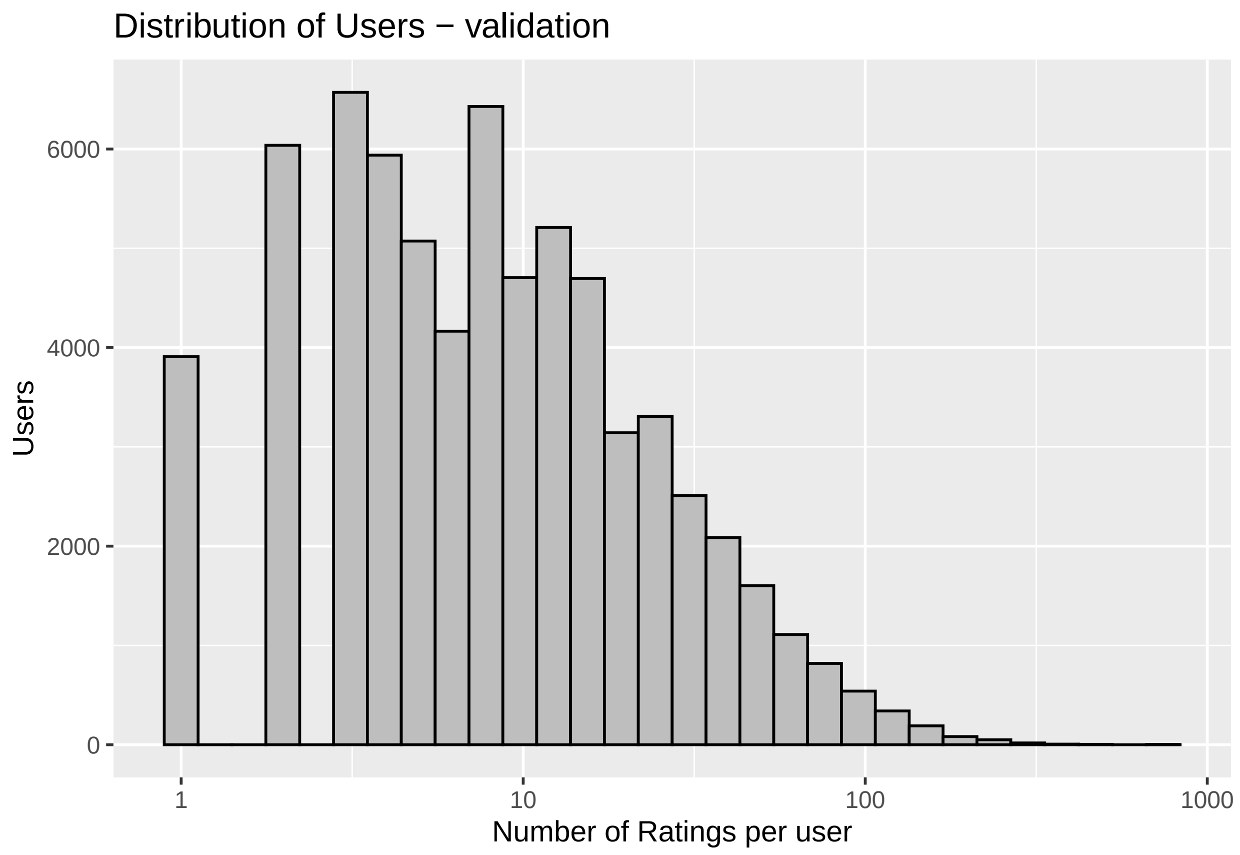
We can see that some genres are more rated than others.

- Distribution of user ratings in *edx* and *validation* datasets:

```
edx %>% group_by(userId) %>%
  summarise(n=n()) %>%
  ggplot(aes(n)) +
  geom_histogram(colour = "black", fill = "grey") +
  scale_x_log10() +
  ggtitle("Distribution of Users - edx") +
  xlab("Number of Ratings per user") +
  ylab("Users")
```



```
validation %>% group_by(userId) %>%
  summarise(n=n()) %>%
  ggplot(aes(n)) +
  geom_histogram(colour = "black", fill = "grey") +
  scale_x_log10() +
  ggtitle("Distribution of Users - validation") +
  xlab("Number of Ratings per user") +
  ylab("Users")
```



We can see that the majority of the users rate less than 100 times. After 100 ratings, there are fewer and fewer users rating a higher number of movies.

2 Methods and Analysis

To evaluate how well the predictive models work, **RMSE** will be calculated, as it is commonly used in predictive models and forecasting.

RMSE is the standard deviation of the residuals and follows the next formula:

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

Being:
N = number of ratings
u = users u
i = movie i
$\hat{y}_{u,i}$ = predicted rating for user u and movie i
$y_{u,i}$ = real rating for user u and movie i

Here is the code:

```r
#Residual Means Squared Errorfunction
RMSE<-function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Once defined how we are going to measure the error, we must define the methods to estimate the ratings.

We are going to try two different methods:
- Regularized User and Movie effect
- Matrix factorization

2.1 First method: Regularized User and Movie effect

We all know that there exist good and bad movies, just as we know that there are very critical people who tend to rate tougher, while other people enjoy any movie.This fact is the basis of the first method proposed.

This method has been studied in a previous course and provides a good approximation because it takes into account both user and movie rating effects.

It also improves results constrainning the total variability of the effect sizes by penalizing large estimates that come from small sample sizes. In other words, we penalize users and movies that have few ratings. This is what we call **regularization**.

The formula of the predicted ratings using this approach is:

$$\hat{y}_{u,i} = \hat{\mu} + \hat{b}_i + \hat{b}_u$$

Where:

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

$$\hat{b}_u = \frac{1}{n_u + \lambda} \sum_{i=1}^{n_u} (y_{u,i} - \hat{b}_i - \hat{\mu})$$

And being:
$\hat{\mu}$ = average of the training set ratings
n_i = number of ratings of the movie i
n_u = number of ratings of the user u
$\hat{b}_i$ = movie ratio, how good a movie is compared to the average (possitive: good movie, negative: bad movie)
$\hat{b}_u$ = user ratio, how demanding a user is (possitive: indulgent, negative: demanding)
λ = regularization tuning parameter that penalizes users and movies that have few ratings

2.2 Second method: Matrix factorization

Traveling through space and visiting different planets is the dream of many people. It is not surprising that this type of people give very high ratings to movies like Star Wars or Star Trek, while they give mediocre ratings to other types of movies, such as for example musical movies. How can we take into account this patterns?

This is where the concept of matrix factorization comes in. First we have to convert the data into a matrix of residuals $r_{u,i}$, where each user gets a row, each movie gets a column and each component is as follows:

$$r_{u,i} = y_{u,i} - \hat{b}_i - \hat{b}_u$$

The second step is to factorize $r_{u,i}$, dividing it into two smaller vectors, p and q, that explain the variance.

The formula of the predicted ratings using this approach is:

$$\hat{y}_{u,i} = \hat{\mu} + \hat{b}_i + \hat{b}_u + \sum_{u,i}(p_u * q_i)$$

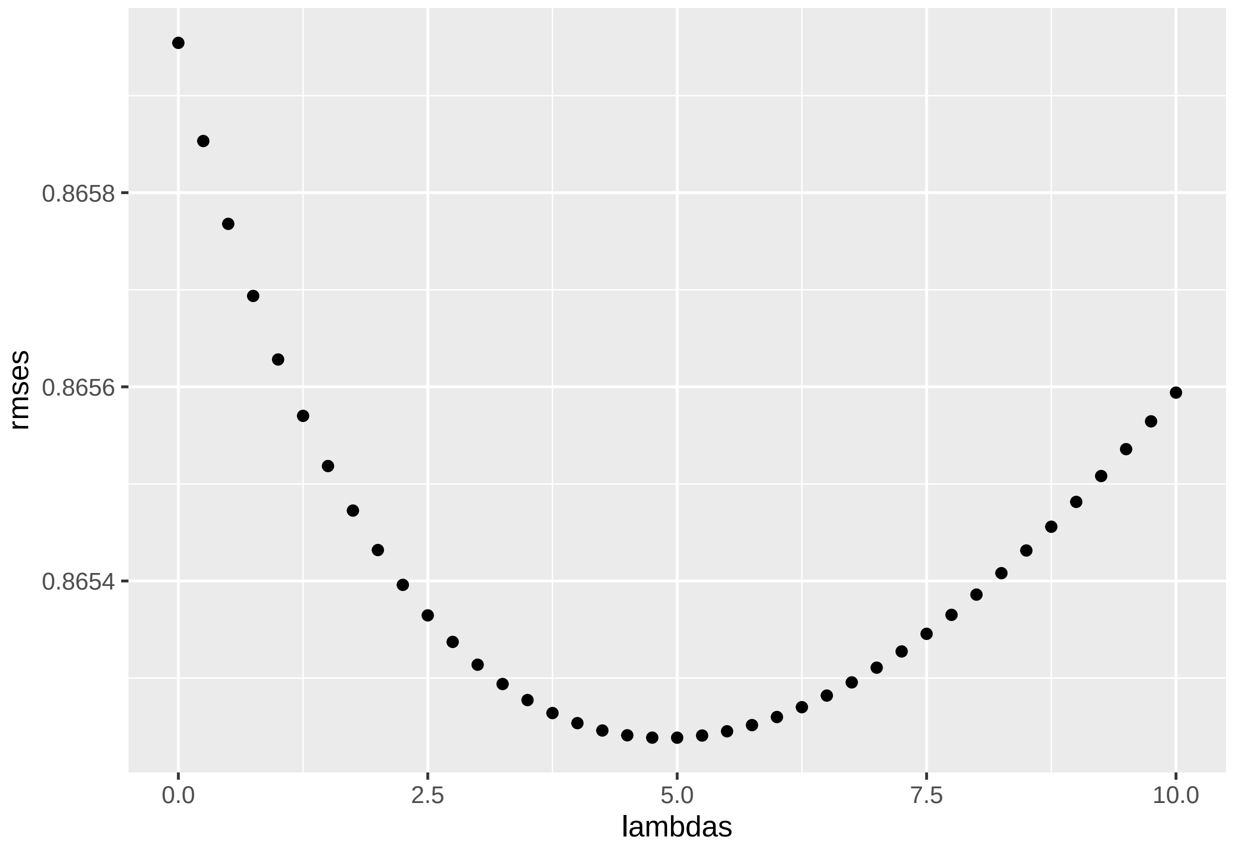
3 Results

This section, like the previous one, is made up of the two methods. Here we can find not only the results but the code.

3.1 Method 1 results

This code runs the regularization model, tunning lambda to optimize the RMSE.

```r
#Calculte RMSE with best lambda
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  b_g <- train_set %>%
    left_join(b_u, by="userId") %>%
    group_by(genres) %>%
    summarize(b_g = mean(rating)/(n()+l))
  predicted_data <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_g)
  return(RMSE(predicted_data$pred, test_set$rating))
})
qplot(lambdas, rmses)
```



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

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

```
model_1_rmse<-min(rmses)
rmse_results <- data_frame(method = "Regularized Movie + User Effect Model",
                           RMSE = model_1_rmse)
```

The RMES using regularization is **0.8652386**.

3.2 Method 2 results

This code runs the matriz factorization model. It implementes the recosystem package, as this package works quite well with dataframes.

The main functions of this package (here is the Recosystem manual) will be explained below for a better understanding of the code:
- data_memory(): specifies the source of data in the recommender system
- Reco(): returns an object of class "RecoSys" equipped with methods train(), tune(), output() and predict(), which describe the typical process of building and tuning model, exporting factorization matrices, and predicting results
- $tune(): uses cross validation to tune the model parameters
- $train(): read from a training data source and creates a model file
- $predict(): predicts the unknown entries in the rating matrix

```r
set.seed(1, sample.kind = "Rounding")

# Create the object of class "RecoSys"
recosystem_data <-  Reco()

# Specify train and test sets as the source of data
train_recosystem <-  with(train_set, data_memory(user_index = userId,
                                                  item_index = movieId,
                                                  rating     = rating))
test_recosystem  <-  with(test_set,  data_memory(user_index = userId,
                                                  item_index = movieId,
                                                  rating     = rating))

# Tune the parameters
tune_parameters <- recosystem_data$tune(train_recosystem, opts = list())

# Train the algorithm with optimal parameters
recosystem_data$train(train_recosystem, opts = c(tune_parameters$min))
```

```
## iter      tr_rmse          obj
##    0       0.9876   9.9450e+06
##    1       0.8807   8.0962e+06
##    2       0.8509   7.5535e+06
##    3       0.8313   7.2165e+06
##    4       0.8169   6.9852e+06
##    5       0.8055   6.8133e+06
##    6       0.7962   6.6812e+06
##    7       0.7883   6.5788e+06
##    8       0.7814   6.4906e+06
##    9       0.7755   6.4172e+06
##   10       0.7704   6.3572e+06
##   11       0.7657   6.3023e+06
##   12      0.7617   6.2581e+06
##   13      0.7580   6.2192e+06
##   14      0.7547   6.1808e+06
##   15      0.7519   6.1536e+06
##   16      0.7492   6.1215e+06
##   17      0.7469   6.0991e+06
##   18      0.7447   6.0745e+06
##   19      0.7428   6.0562e+06
```

```
predicted_values2 <-  recosystem_data$predict(test_recosystem, out_memory())

# Calculate RMSE
model_2_rmse <- RMSE(predicted_values2, test_set$rating)

rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Matrix Factorization",
                                     RMSE = model_2_rmse))
```

The RMES using matrix factorization is **0.7957658**.

Here is the comparison:

```
rmse_results %>% knitr::kable()
```

method	RMSE
Regularized Movie + User Effect Model	0.8652386
Matrix Factorization	0.7957658

We can see that the second method is much better, so the final step is applying it to the validation set:

```
set.seed(1, sample.kind = "Rounding")

# Create the object of class "RecoSys"
recosystem_data_v <-  Reco()

# Specify train and test sets as the source of data
train_recosystem_v <-  with(edx, data_memory(user_index = userId,
                                             item_index = movieId,
                                             rating     = rating))
test_recosystem_v  <-  with(validation,  data_memory(user_index = userId,
                                                     item_index = movieId,
                                                     rating     = rating))


# Tune the parameters
```

```r
tune_parameters_v <- recosystem_data_v$tune(train_recosystem_v, opts = list())

# Train the algorithm with optimal parameters
recosystem_data_v$train(train_recosystem_v, opts = c(tune_parameters_v$min))
```

```
## iter      tr_rmse          obj
##    0       0.9676   1.1918e+07
##    1       0.8751   9.9242e+06
##    2       0.8433   9.2286e+06
##    3       0.8244   8.8385e+06
##    4       0.8104   8.5756e+06
##    5       0.7994   8.3783e+06
##    6       0.7905   8.2306e+06
##    7       0.7830   8.1150e+06
##    8       0.7765   8.0174e+06
##    9       0.7710   7.9361e+06
##   10       0.7664   7.8703e+06
##   11       0.7624   7.8129e+06
##   12       0.7590   7.7676e+06
##   13       0.7561   7.7285e+06
##   14       0.7534   7.6895e+06
##   15       0.7512   7.6637e+06
##   16       0.7491   7.6320e+06
##   17       0.7473   7.6106e+06
##   18       0.7456   7.5865e+06
##   19       0.7440   7.5693e+06
```

```r
predicted_values_v <-  recosystem_data_v$predict(test_recosystem_v, out_memory())

# Calculate RMSE
model_3_rmse <- RMSE(predicted_values_v, validation$rating)
```

The RMES using matrix factorization in the validation set is **0.7882084**.

4 Conclusions

After having tried both models, we have opted for the second one (matrix factorization). Although regularization has not given bad results, it does not consider other factors than the user and the identification of the film.

Choosing matrix factorization is a better option since, as explained above, it takes into account other types of relationships between the data.

4.1 Limitations

The main limitation of this project is undoubtedly the computer itself. The computational burden of handling such a large volume of data makes it not feasible to test other types of models. Converting this Rmd file to PDF took more than 12 hours.

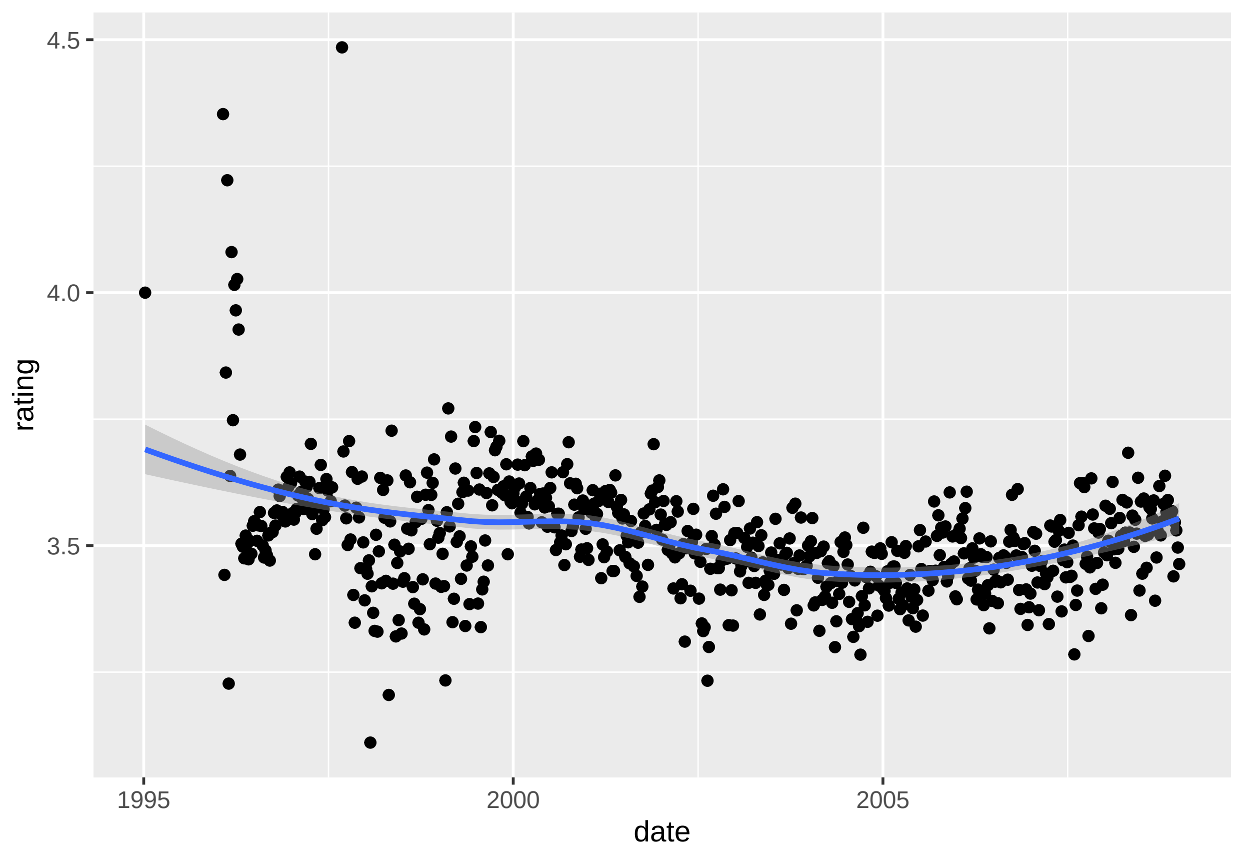
Also the data that we work with is nominal qualitative (userId, MovieId, genres...). No mathematical computations can be carried out so certain models can not be tested.

4.2 Other possible models / Future work

Other models that could have been studied in this project will simply be mentioned in this section.

- Time effect model:
  If we consider the plot resulting from the following code, we can see that the is some evidence of a time effect on average rating.

```
edx <- mutate(edx, date = as_datetime(timestamp))
edx %>% mutate(date = round_date(date, unit = "week")) %>%
    group_by(date) %>%
    summarize(rating = mean(rating)) %>%
    ggplot(aes(date, rating)) +
    geom_point() +
    geom_smooth()
```



The formula of the predicted ratings using this approach would be something like this:

$$\hat{y}_{u,i} = \hat{\mu} + \hat{b}_i + \hat{b}_u + f(d_{u,i})$$

Being:
$f(d_{u,i})$ = smooth function of the day for user's u rating of movie i

- Genre effect model:
  If we consider the tables resulting from the following code, we can see that there are some genres that have lower average rating than others.

```
edx %>% group_by(genres) %>%
  summarise(avg=mean(rating)) %>%
  arrange(-avg) %>%
  head(10)
```

```
## # A tibble: 10 x 2
##    genres                                    avg
##    <chr>                                   <dbl>
##  1 Animation|IMAX|Sci-Fi                    4.71
##  2 Drama|Film-Noir|Romance                  4.30
##  3 Action|Crime|Drama|IMAX                    4.30
##  4 Animation|Children|Comedy|Crime            4.28
##  5 Film-Noir|Mystery                          4.24
##  6 Crime|Film-Noir|Mystery                    4.22
##  7 Film-Noir|Romance|Thriller                 4.22
##  8 Crime|Film-Noir|Thriller                   4.21
##  9 Crime|Mystery|Thriller                     4.20
## 10 Action|Adventure|Comedy|Fantasy|Romance  4.20
```

```
edx %>% group_by(genres) %>%
  summarise(avg=mean(rating)) %>%
  arrange(avg) %>%
  head(10)
```

```
## # A tibble: 10 x 2
##    genres                                         avg
##    <chr>                                        <dbl>
##  1 Documentary|Horror                            1.45
##  2 Action|Animation|Comedy|Horror                1.5
##  3 Action|Horror|Mystery|Thriller                1.61
##  4 Comedy|Film-Noir|Thriller                     1.64
##  5 Action|Drama|Horror|Sci-Fi                    1.75
##  6 Adventure|Drama|Horror|Sci-Fi|Thriller        1.75
##  7 Action|Adventure|Drama|Fantasy|Sci-Fi         1.90
##  8 Action|Children|Comedy                        1.91
##  9 Action|Adventure|Children                     1.92
## 10 Adventure|Animation|Children|Fantasy|Sci-Fi  1.92
```

As there are many combinations of genres, this tables only show the tenth best rated sub-genres and the tenth worst rated sub-genres respectively, but this is useful to have a general image that there are better and worse valued genres and in fact, a genre effect.

The formula of the predicted ratings if we take into account the genres (and the user and movie effect) would be something like this:

$$\hat{y}_{u,i} = \hat{\mu} + \hat{b}_i + \hat{b}_u + \sum_{k=1}^{K} x_{u,i}^k \beta_k(x)$$

Being:
$g_{u,i}$ = genre for user's u rating of movie i
$x_{u,i}^k = 1$ if $g_{u,i}$ is genre k

- Best matrix factorization tuning parameters:
  As we have seen previously, the function $tune from the Recosystem package allows us to modify the model parameters. A good option is to optimize them in order to get the best predicted values.

References

All references are included in the report through hyperlinks.