Movie Recommendation System

Sergio Walias Sanchez

28/12/2020

1. Introduction

This report is part of the capstone project of HarvardX's Data Science Professional Certificate1 program.

The aim of this project is the development of a recommendation system. These systems are based on a rating scale from 1 to 5, with 1 the lowest rating and 5 the highest rating. Normally, apart from the rating, user and title of movie, more information is added as genre of movie and launching date. The goal of the recommendation systems is to direct users to those films within their preferences. This technique is important for big companies such as Netflix. In 2006 Netflix offered a one-million-dollar prize for the person or group that could improve their recommendation system by at least 10%.

In this capstone project, we develop a movie recommendation system using the MovieLens dataset and applying the information provided during the HarvardX's Data Science Professional Certificate program.

This document is structured as follows. Chapter 1 describes the dataset and summarizes the goal of the project and key steps that were performed. In chapter 2 we explain the process and techniques used, such as data cleaning, data exploration and visualization, any insights gained, and the modeling approach. In chapter 3 we present the modeling results and discuss the model performance. We conclude in chapter 4 with a summary of the report, its limitations and future work.

1.1. MovieLens Dataset

The complete MovieLens dataset consists of 27 million ratings of 58,000 movies by 280,000 users. The research presented in this project is based in a subset of this dataset with 10 million ratings on 10,000 movies by 72,000 users.

[1] https://www.edx.org/professional-certificate/harvardx-data-science

1.2. Model Evaluation

The evaluation of machine learning algorithms consists in comparing the predicted value with the actual outcome. The loss function measures the difference between both values.

The most common loss functions in machine learning are the mean absolute error (MAE), mean squared error (MSE) and root mean squared error (RMSE).

The goal of this project is to create a recommendation system with RMSE lower than 0.8649. There is no specific target for MSE and MAE. So, just the RMSE is used.

1.2.1. Root Mean Squared Error - RMSE

The Root Mean Squared Error, RMSE, is the square root of the MSE. It is the typical metric to evaluate recommendation systems, and is defined by the formula:

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

where N is the number of ratings, $y_{u,i}$ is the rating of movie i by user u and $\hat{y}_{u,i}$ is the prediction of movie i by user u.

The RMSE penalizes large deviations from the mean and is appropriate in cases that small errors are not relevant.

1.3. Process and Workflow

The main steps in a data science project include:

1. Data preparation: download, parse, import and prepare the data to be processed and analyzed.
2. Data exploration and visualization: explore data to understand the features and the relationship between the features and predictors.
3. Data cleaning: the dataset contains unnecessary information that needs to be removed.
4. Data analysis and modeling: create the model using the insights gained during exploration. Also test and validate the model.
5. Communicate: create the report and publish the results.

First, we download the dataset from MovieLens website and split into two subsets used for training and validation. The training subset is called `edX` and the validation subset is called `validation`. The `edx` set is split again into two subsets used for training and testing. When the model reaches the RMSE target in the testing set, we train the `edx` set with the model and use the `validation` set for final validation. We pretend the `validation` set is new data with unknown outcomes.

In the next step we create charts, tables, and statistics summary to understand how the features can impact the outcome. The information and insights obtained during exploration will help to build the machine learning model.

We start building a model which considers the user and movie effects. Then we build a similar model that considers the genre effect, improving the RMSE. Finally, the user and movie effects, receive regularization parameter that penalizes samples with few ratings. Regularization has been introduced also for user, movie and genres effects.

Matrix factorization with Recosystem algorithm is evaluated and provides a better prediction. Although the package Recosystem is not included in[2] program, this is one of the most useful tools for recommendation system and it was interesting to use it.

2. Methods and Analysis

2.1. Data Preparation

In this section we download and prepare the dataset to be used in the analysis. We split the dataset in two parts, the training set called `edX` and the evaluation set called `validation` with 90% and 10% of the original dataset, respectively.

Then, we split the `edx` set in two parts, the `train` set and `test` set with 90% and 10% of `edX` set, respectively. The model is created and trained in the `train` set and tested in the `test` set until the RMSE target is achieved, then finally we train the model again in the entire `edx` set and validate in the `validation` set. The name of this method is *cross-validation*.

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

[2] https://www.edx.org/professional-certificate/harvardx-data-science

```
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(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,]
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)

options(digits=10)
```

The `edx` set is used for training and testing, and the `validation` set is used for final validation to simulate the new data.

Here, we split the edx set in 2 parts: the training set and the test set.

The model building is done in the training set, and the test set is used to test the model. When the model is complete, we use the `validation` set to calculate the final RMSE.
We use the same procedure used to create `edx` and `validation` sets.

The training set will be 90% of edx data and the test set will be the remaining 10%.

```
#DATA PREPARATION
#partition
set.seed(1)
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1,
list = FALSE)
edx_train <- edx[-test_index,]
edx_test_prov <- edx[test_index,]
#assuring userId and movieId in both partitions
edx_test<-
edx_test_prov%>%semi_join(edx_train,by="movieId")%>%semi_join(edx_train,b
y="userId")
train_2<-anti_join (edx_test_prov,edx_test)
edx_train<-rbind(edx_train,train_2)
```

2.2. Data Exploration

```
str(edx)

## Classes 'data.table' and 'data.frame':   9000061 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 231 292 316 329 355 356 362 364 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983392 838983421 838983392
838983392 838984474 838983653 838984885 838983707 ...
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Dumb & Dumber
(1994)" "Outbreak (1995)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Comedy"
"Action|Drama|Sci-Fi|Thriller" ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

The dataset is in tidy format, i.e., each row has one observation and the column names are the features. The `rating` column is the desired outcome. The user information is stored in `userId`; the movie information is both in `movieId` and `title` columns. The rating date is available in `timestamp` measured in seconds since January 1st, 1970. Each movie is tagged with one or more genre in the `genre's` column.

```
head(edx)

##    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
```

2.2.1. Genres

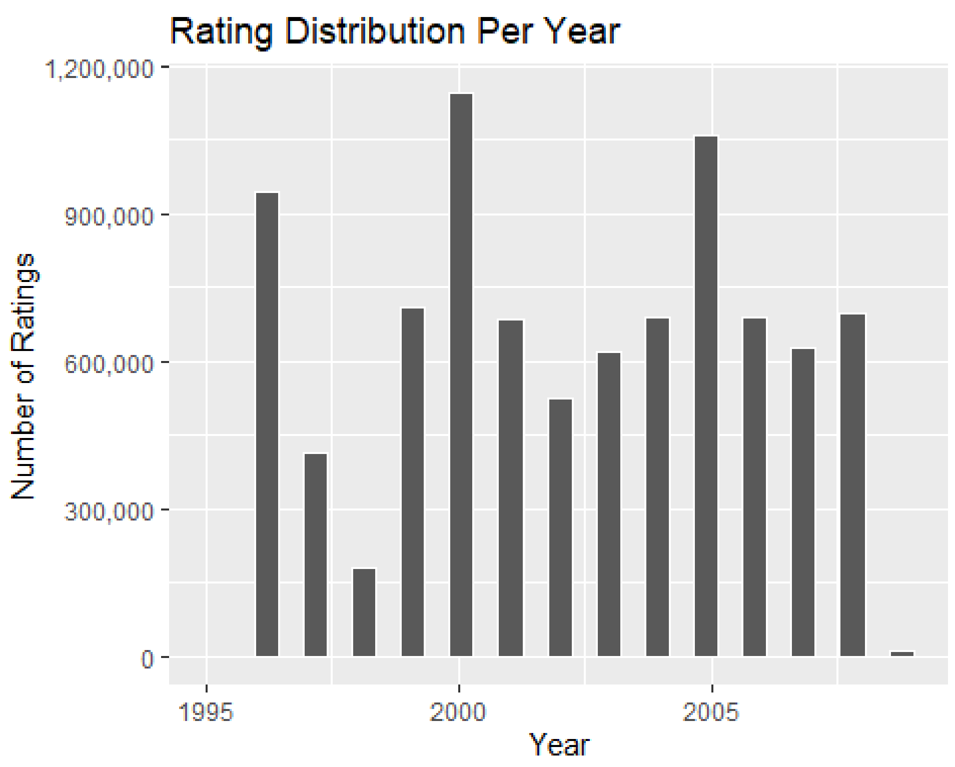
Along with the movie title, MovieLens provides the list of genres for each movie.

```
edx %>% group_by(genres) %>%
  summarise(n=n()) %>% arrange(-n)%>%
  head(10)
```

```
## # A tibble: 10 x 2
##    genres                          n
##    <chr>                       <int>
##  1 Drama                      733353
##  2 Comedy                     700883
##  3 Comedy|Romance             365894
##  4 Comedy|Drama               323518
##  5 Comedy|Drama|Romance       261098
##  6 Drama|Romance              259735
##  7 Action|Adventure|Sci-Fi    220363
##  8 Action|Adventure|Thriller 148933
##  9 Drama|Thriller             145359
## 10 Crime|Drama                137424
```

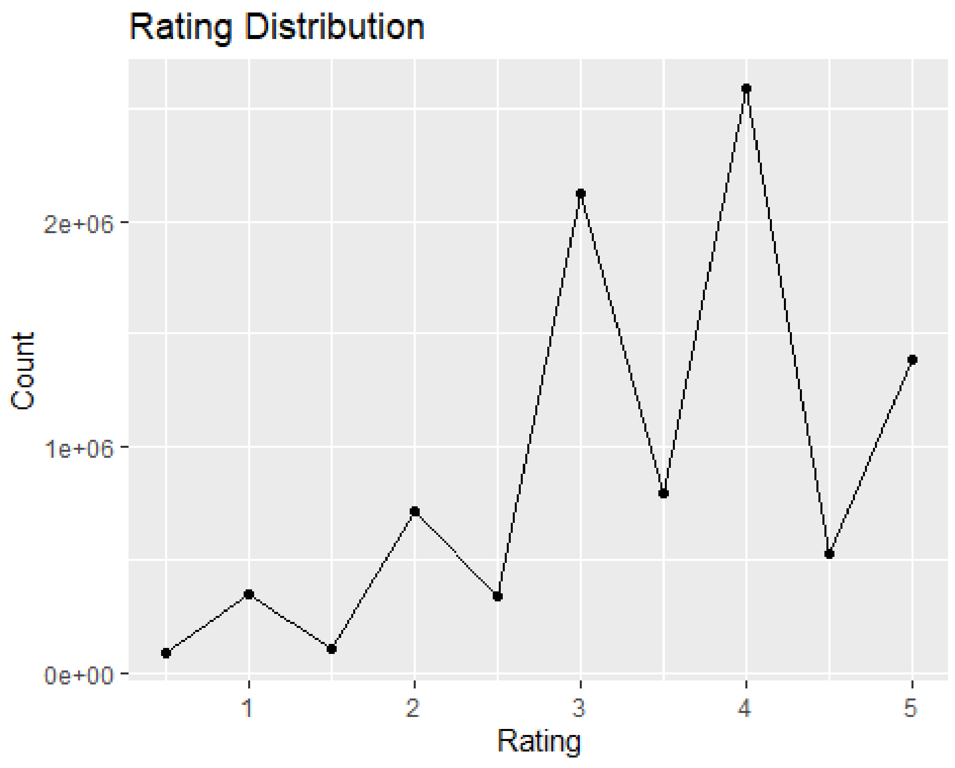
2.2.2. Date

```
edx %>% mutate(year = year(as_datetime(timestamp, origin="1970-01-01")))
%>%
  ggplot(aes(x=year)) +
  geom_histogram(color = "white") +
  ggtitle("Rating Distribution Per Year") +
  xlab("Year") +
  ylab("Number of Ratings") +
  scale_y_continuous(labels = comma)
```



2.2.3. Ratings

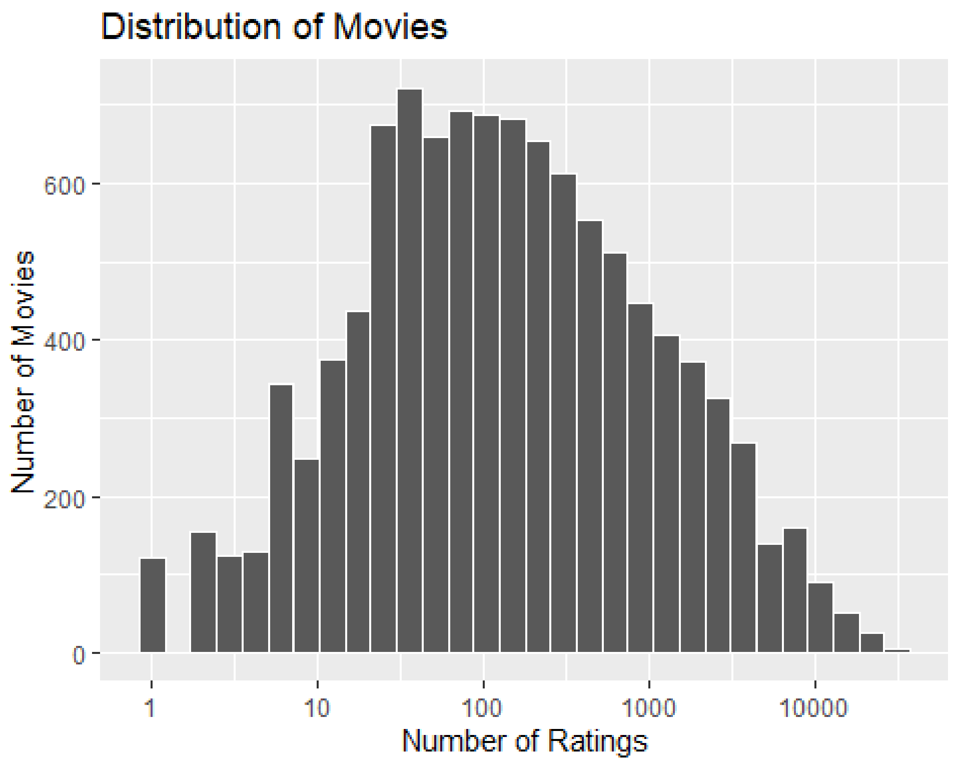
```
edx %>% group_by(rating) %>% summarize(n=n())

## # A tibble: 10 x 2
##    rating       n
##     <dbl>   <int>
##  1    0.5   85420
##  2    1    345935
##  3    1.5  106379
##  4    2    710998
##  5    2.5  332783
##  6    3   2121638
##  7    3.5  792037
##  8    4   2588021
##  9    4.5  526309
## 10    5   1390541

edx %>% group_by(rating) %>%
  summarise(count=n()) %>%
  ggplot(aes(x=rating, y=count)) +
  geom_line() +
  geom_point() +
  ggtitle("Rating Distribution") +
  xlab("Rating") +
  ylab("Count")
```



2.2.4. Movies

There are different movies in the edx set. Some of them are rated more than others, since many movies are watched by few users and blockbusters tend to have more ratings.

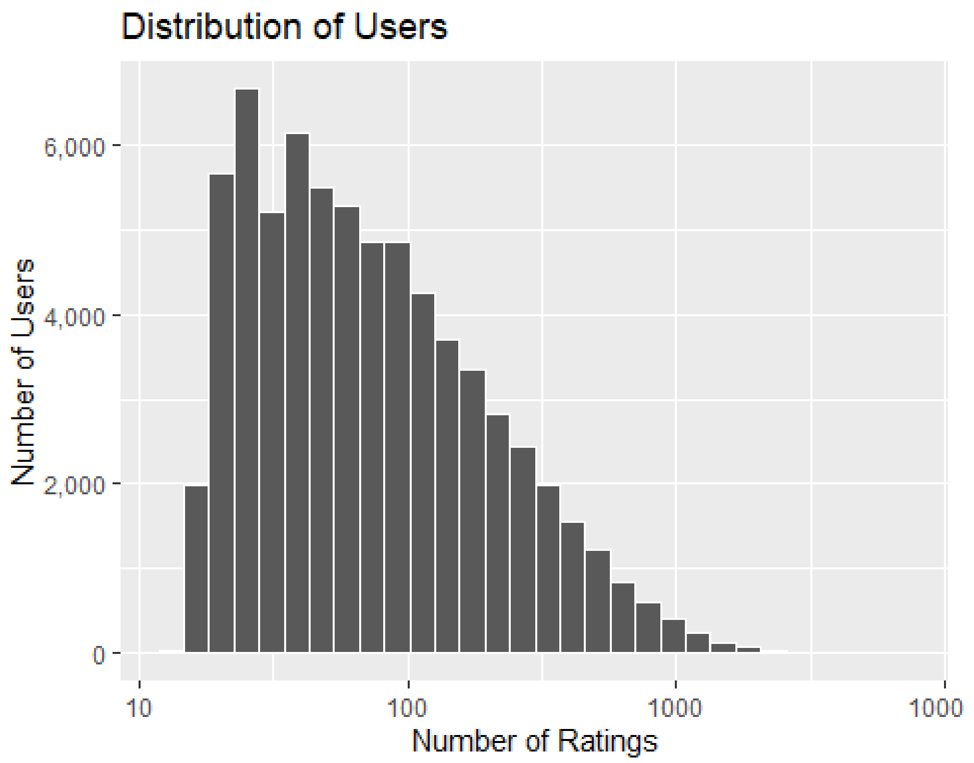
```
edx %>% group_by(movieId) %>%
  summarise(n=n()) %>%
  ggplot(aes(n)) +
  scale_x_log10() +
  geom_histogram(color = "white") +
  ggtitle("Distribution of Movies") +
  xlab("Number of Ratings") +
  ylab("Number of Movies")
```



2.2.5. Users

Most users rate few movies, while a few users rate more than a thousand movies.

```
edx %>% group_by(userId) %>%
  summarise(n=n()) %>%
  ggplot(aes(n)) +
  geom_histogram(color = "white") +
  scale_x_log10() +
  ggtitle("Distribution of Users") +
  xlab("Number of Ratings") +
  ylab("Number of Users") +
  scale_y_continuous(labels = comma)
```



2.3. Data Cleaning

For the following study, timestamp is not used, so this variable will be eliminated from data. Genres will be used with movie, title, rating and user.

```
edx_train <- edx_train %>% select(userId, movieId, rating, genres, title)
edx_test  <- edx_test  %>% select(userId, movieId, rating, genres, title)
```

2.4. Modeling

2.4.1. Linear Models

The first estimation could be the mean of all ratings μ:

$$\hat{Y}_{u,i} = \mu + \epsilon_{i,u}$$

Where $\hat{Y}$ is the predicted rating, μ is the mean of observed data and $\epsilon_{i,u}$ is the error distribution.

Part of the movie-to-movie variability can be explained by the fact that different movies have different rating distribution. Some movies are more popular than others and the public preference varies. This is called movie effect or movie bias, and is expressed as b_i in this formula:

$$\hat{Y}_{u,i} = \mu + b_i + \epsilon_{i,u}$$

The movie effect can be calculated as the mean of the difference between the observed rating y and the mean μ.

$$\hat{b}_i = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{\mu})$$

Like the movie effect, different users have different rating pattern or distribution. This is called user effect or user bias and is expressed in this formula:

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

The prediction model that includes the user effect becomes:

$$\hat{Y}_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

Movies can be grouped into categories or genres, with different distributions. In general, movies in the same genre get similar ratings. The genres effect or bias could be calculated as follows:

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

$$\hat{Y}_{u,i} = \mu + b_i + b_u + b_g + \epsilon_{u,i}$$

In our model, mu is called o_avg, b_i, b_mov, b_u, b_user and b_g, b_genres.

2.4.2. Regularization

The linear model provides a good estimation for the ratings but does not consider that many movies have very few numbers of ratings, and some users rate very few movies. This means that the sample size is very small for these movies and these users. Statistically, this leads to large estimated error.

The estimated value can be improved adding a factor that penalizes small sample sizes and have have little or no impact otherwise. Thus, estimated movie and user effects can be calculated with these formulas:

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

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

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

For values of N smaller than or like λ, $\hat{b}_i$ $\hat{b}_u$ and $\hat{b}_g$ is smaller than the original values, whereas for values of N much larger than λ, $\hat{b}_i$ $\hat{b}_u$ and $\hat{b}_g$ change very little.

An effective method to choose λ that minimizes the RMSE is running simulations with several values of λ.

2.4.3. Matrix Factorization

Matrix factorization is widely used machine learning tool for predicting ratings in recommendation systems. The data can be converted into a matrix such that each user is in a row, each movie is in a column and the rating is in the cell, then the algorithm attempts to fill in the missing values.

The concept is to approximate a large rating matrix $R_{m\times n}$ into the product of two lower dimension matrices $P_{k\times m}$ and $Q_{k\times n}$, such that

The R `recosystem`[3] package provides methods to decompose the rating matrix and estimate the user rating, using parallel matrix factorization.

3. Results

This section presents the code and results of the models.

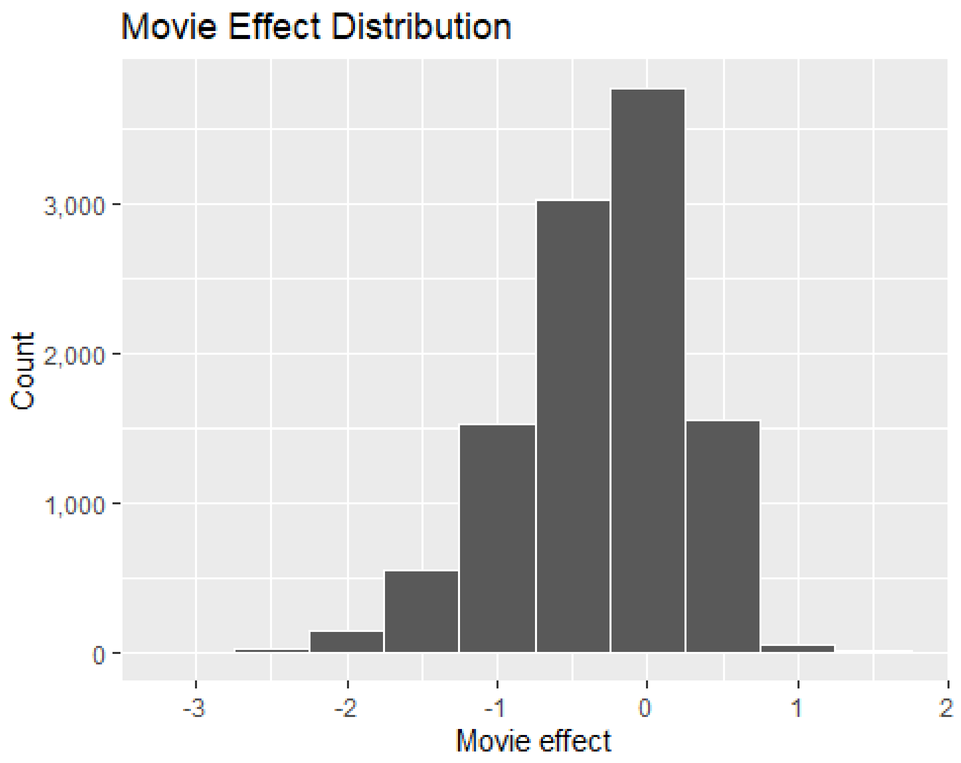
3.1. Linear Models

```
#calculate overall average o_avg
  o_avg<-mean(edx_train$rating)
```

3.1.1. Linear Model 1. Movie and user effect

```
#calculate movie bias
  movie_avg<-edx_train%>%group_by(movieId)%>%summarize(b_mov =
mean(rating - o_avg))
#distribution movie_avg
  movie_avg %>% ggplot(aes(x = b_mov)) +
    geom_histogram(bins=10, col = I("white")) +
    ggtitle("Movie Effect Distribution") +
    xlab("Movie effect") +
    ylab("Count") +
    scale_y_continuous(labels = comma)
```

[3] https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html



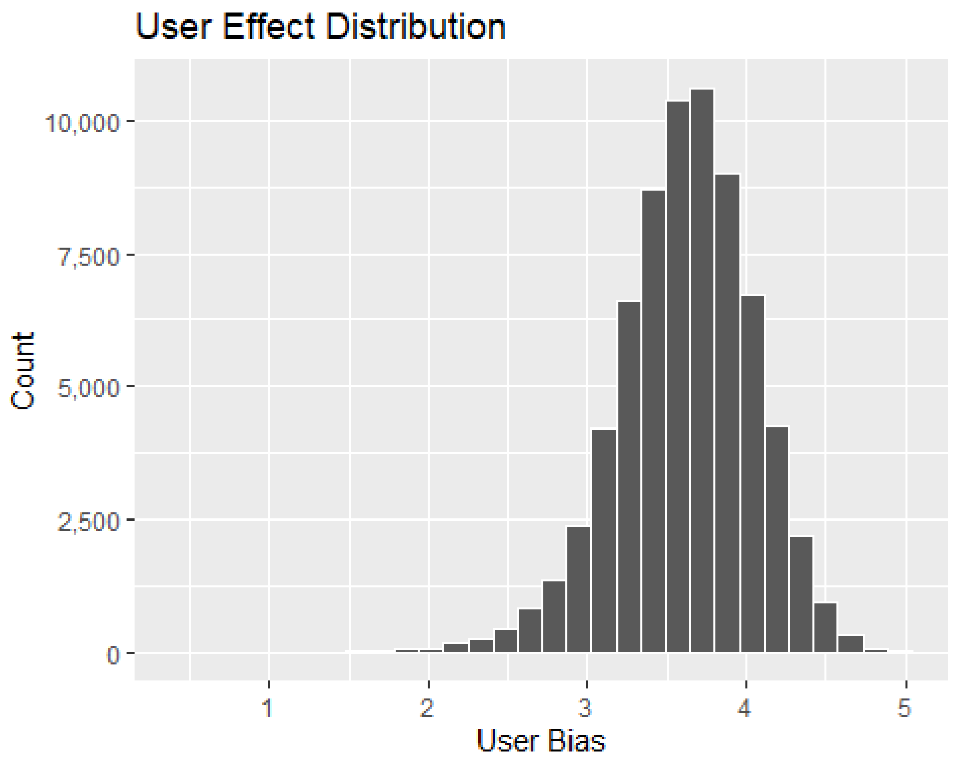
The movie effect is normally left skewed distributed.

b_u is the user effect (bias) for user u.

$$\hat{y}_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

In our model, mu is called o_avg, b_i, b_mov, b_u, b_user and b_g, b_genres.

```
#calculate user bias
  user_avg<-edx_train%>%left_join(movie_avg,
by='movieId')%>%group_by(userId)%>%summarize(b_user = mean(rating - o_avg
- b_mov))
#distribution user_avg
  edx_train %>%
    group_by(userId) %>%
    summarize(b_user = mean(rating)) %>%
    filter(n()>=100) %>%
    ggplot(aes(b_user)) +
    geom_histogram(color = "white") +
    ggtitle("User Effect Distribution") +
    xlab("User Bias") +
    ylab("Count") +
    scale_y_continuous(labels = comma)
```



The user effect is normally distributed.

Predict the rating with mean + bmov + buser

```
#prediction with user and movie bias
  y_hat_bm_bu <- edx_test %>%
    left_join(movie_avg, by='movieId') %>%
    left_join(user_avg, by='userId') %>%
    mutate(pred = o_avg + b_mov + b_user) %>%
    .$pred
```

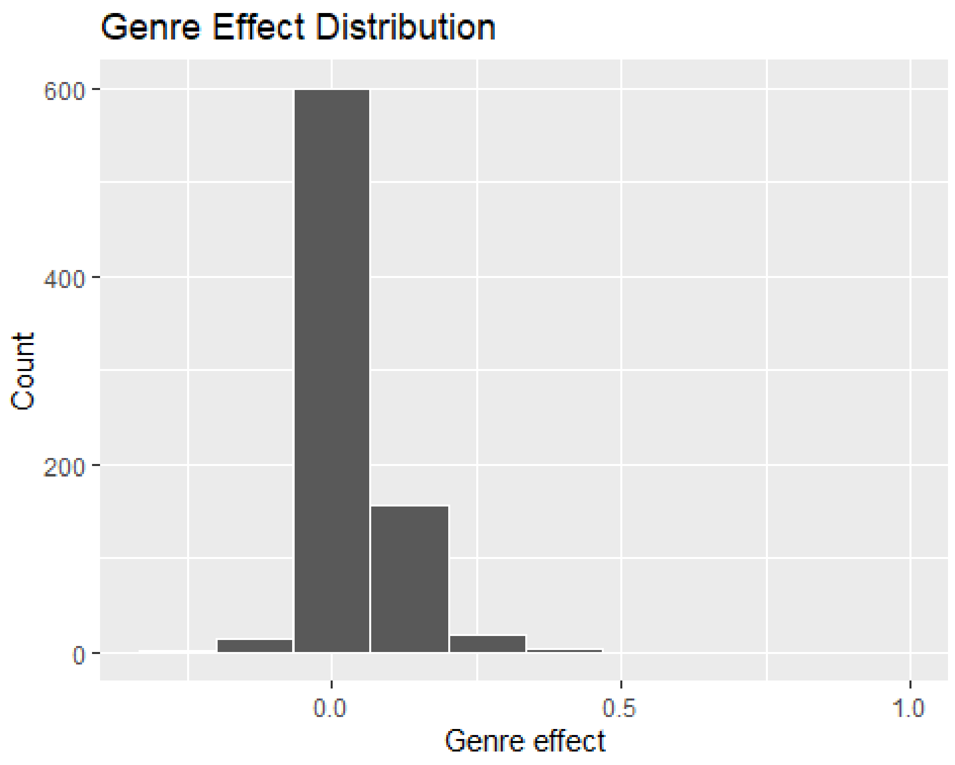
Evaluation of case 1:

```
#RMSE (linear user and movie)
  RMSE_1 <- sqrt(mean((edx_test$rating - y_hat_bm_bu)^2))
  RMSE_1

## [1] 0.864604679

  RMSE_edx_test<-tibble(Method="Lineal mov+us",RMSE=RMSE_1)
```

3.1.2. Linear Model 2. Movie, user and genres effect

```
#calculate genres bias
  genres_avg<-edx_train%>%left_join(movie_avg, by='movieId') %>%
    left_join(user_avg, by='userId') %>%
group_by(genres)%>%summarize(b_genres = mean(rating - o_avg - b_mov-
b_user))
#distribution genres bias
```

```
  genres_avg %>% ggplot(aes(x = b_genres)) +
    geom_histogram(bins=10, col = I("white")) +
    ggtitle("Genre Effect Distribution") +
    xlab("Genre effect") +
    ylab("Count") +
    scale_y_continuous(labels = comma)
```



Predict the rating with mean + bmov + buser + bgenres

```
#prediction with user, movie and genre bias
  y_hat_bm_bu_g <- edx_test %>%
    left_join(movie_avg, by='movieId') %>%
    left_join(user_avg, by='userId') %>%
    left_join(genres_avg, by='genres') %>%
    mutate(pred = o_avg + b_mov + b_user+b_genres) %>%
    .$pred
```

Evaluation of case 2:

```
  RMSE_2 <- sqrt(mean((edx_test$rating - y_hat_bm_bu_g)^2))
  RMSE_2

## [1] 0.8642541637

  RMSE_edx_test<-bind_rows(RMSE_edx_test,tibble(Method="Linear
mov+us+gen",RMSE=RMSE_2))
```

3.2. Regularization

3.2.1. Regularization. Case 3. Movie and user effects.

Now, we regularize the user and movie effects adding a penalty factor λ, which is a tuning parameter. We define several values for λ and use the regularization function to pick the best value that minimizes the RMSE. Firstly, we try with a lambda between 0 and 10 with a step of 0.25. If we do not find the minimum, we would try a difference lambda interval.

```
  lambda<-seq(0,10,0.25)
  RMSE_3<-vector(length=41)
```

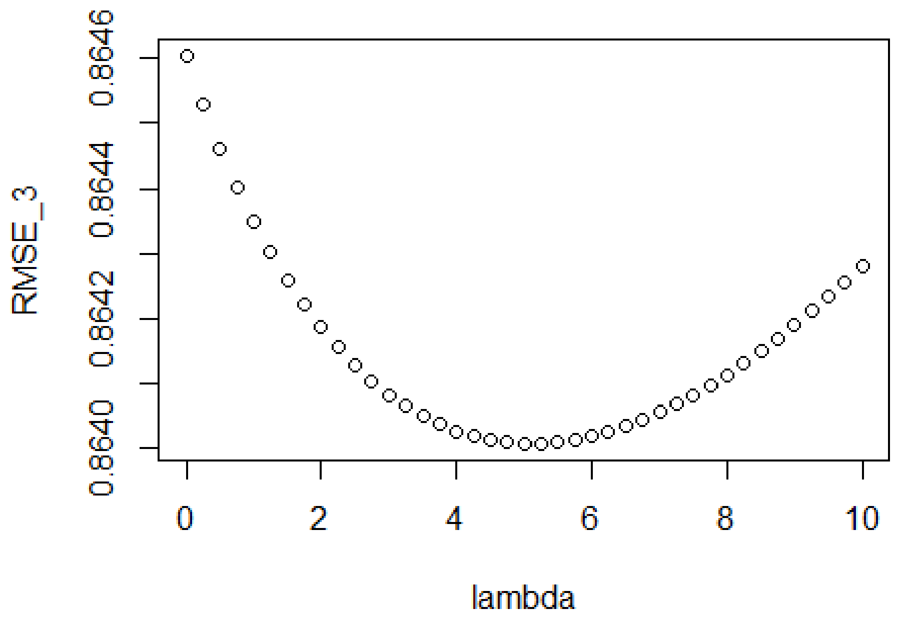
The method consists of the calculation of *bi*, *bu* with a particular *lambda*. Then, with these parameters, prediction will be calculated. With the prediction and the actual ratings, RMSE is calculated. Once the process is repeated for each *lambda*, we pick the *lambda* that suits the minimum RMSE and we calculate with this *lambda* all the parameters. The *lambda* that we pick is called lambda target.

Calculation of RMSE of each *lambda*:

```
 for (i in 1:41){
  #calculate movie bias
  movie_reg<-edx_train%>%group_by(movieId)%>%summarize(b_mov = sum(rating
- o_avg)/(n()+lambda[i]))
  #calculate user bias
  user_reg<-edx_train%>%left_join(movie_reg, by='movieId')%>%
  group_by(userId)%>%summarize(b_user = sum(rating - o_avg -
b_mov)/(n()+lambda[i]))
  #prediction with user and movie bias
  y_hat_reg <- edx_test %>%
    left_join(movie_reg, by='movieId') %>%
    left_join(user_reg, by='userId') %>%
    mutate(pred = o_avg + b_mov + b_user) %>%
    .$pred
  #RMSE (linear user and movie)
  RMSE_3[i] <- sqrt(mean((edx_test$rating - y_hat_reg)^2))
  }
```

Representation of each *lambda* and its RMSE:

```
plot(lambda,RMSE_3)
```



Picking of best *lambda*, *lambda* target:

```
lambda_target<-lambda[which.min(RMSE_3)]
```

Calculation of all parameters with *lambda* target:

```
 #calculate RMSE_3 with lambda_target
  movie_reg<-edx_train%>%group_by(movieId)%>%summarize(b_mov = sum(rating
- o_avg)/(n()+lambda_target))
  user_reg<-edx_train%>%left_join(movie_reg, by='movieId')%>%
  group_by(userId)%>%summarize(b_user = sum(rating - o_avg -
b_mov)/(n()+lambda_target))
  y_hat_reg <- edx_test %>%
    left_join(movie_reg, by='movieId') %>%
    left_join(user_reg, by='userId') %>%
    mutate(pred = o_avg + b_mov + b_user) %>%
    .$pred
```

Evaluation of case 3:

```
  RMSE_3 <- sqrt(mean((edx_test$rating - y_hat_reg)^2))
  (RMSE_3)

## [1] 0.8640060089

  RMSE_edx_test<-bind_rows(RMSE_edx_test,tibble(Method="Regularization
mov+us",RMSE=RMSE_3))
```

3.2.2. Regularization. Case 4. Movie, user and genres effects.

The process is the same as explained in case 3. The only difference is the inclusion of genres effect. Firstly, we try with a lambda between 0 and 10 with a step of 0.25. If we do not find the minimum, we would try a difference lambda interval.

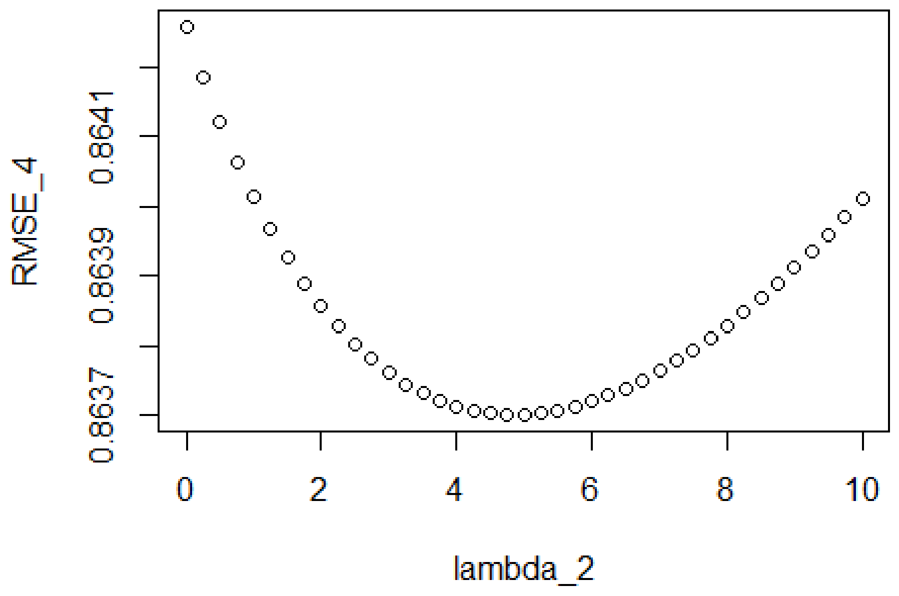
```
  lambda_2<-seq(0,10,0.25)
  RMSE_4<-vector(length=41)
```

Calculation of RMSE of each *lambda*:

```
  for (i in 1:41){
    movie_regg<-edx_train%>%group_by(movieId)%>%summarize(b_mov =
sum(rating - o_avg)/(n()+lambda_2[i]))
    user_regg<-edx_train%>%left_join(movie_reg, by='movieId')%>%
    group_by(userId)%>%summarize(b_user = sum(rating - o_avg -
b_mov)/(n()+lambda_2[i]))

    genres_regg<-edx_train%>%left_join(movie_avg, by='movieId')
%>%left_join(user_avg, by='userId') %>%
    group_by(genres)%>%summarize(b_genres = sum(rating - o_avg - b_mov-
b_user)/(n()+lambda_2[i]))

    y_hat_bm_bu_lg <- edx_test %>%
      left_join(movie_regg, by='movieId') %>%
      left_join(user_regg, by='userId') %>%
      left_join(genres_regg, by='genres') %>%
      mutate(pred = o_avg + b_mov + b_user+b_genres) %>%
      .$pred

    RMSE_4[i] <- sqrt(mean((edx_test$rating - y_hat_bm_bu_lg)^2))}
```

Representation of each *lambda* and its RMSE:

```
plot(lambda_2,RMSE_4)
```



Picking of best *lambda*, *lambda* target:

```
    lambda_target_2<-lambda[which.min(RMSE_4)]
```

Calculation of all parameters with *lambda* target:

```
    #calculate RMSE_4 with lambda_target_2
    movie_regg<-edx_train%>%group_by(movieId)%>%summarize(b_mov =
sum(rating - o_avg)/(n()+lambda_target_2))
    user_regg<-edx_train%>%left_join(movie_reg, by='movieId')%>%
      group_by(userId)%>%summarize(b_user = sum(rating - o_avg -
b_mov)/(n()+lambda_target_2))
    genres_regg<-edx_train%>%left_join(movie_avg, by='movieId')
%>%left_join(user_avg, by='userId') %>%
      group_by(genres)%>%summarize(b_genres = sum(rating - o_avg - b_mov-
b_user)/(n()+lambda_target_2))
    y_hat_regg <- edx_test %>%
      left_join(movie_regg, by='movieId') %>%
      left_join(user_regg, by='userId') %>%
      left_join(genres_regg, by='genres') %>%
      mutate(pred = o_avg + b_mov + b_user+b_genres) %>%
      .$pred
```

Evaluation of case 4:

```
    RMSE_4 <- sqrt(mean((edx_test$rating - y_hat_regg)^2))
    RMSE_4
```

```
## [1] 0.8637010166

    RMSE_edx_test<-bind_rows(RMSE_edx_test,tibble(Method="Regularization
mov+us+gen",RMSE=RMSE_4))
```

3.3. Matrix factorization (case 5)

Matrix factorization approximates a large user-movie matrix into the product of two smaller dimension matrices. Information in the train set is stored in tidy format, with one observation per row, so it needs to be converted to the user-movie matrix before using matrix factorization. This code executes this transformation.

Usage of **`recosystem`**

The usage of **`recosystem`** is quite simple, mainly consisting of the following steps:

1. Create a model object (a Reference Class object in R) by calling `Reco()`.
2. (Optionally) call the `$tune()` method to select best tuning parameters along a set of candidate values.
3. Train the model by calling the `$train()` method. Several parameters can be set inside the function, possibly coming from the result of `$tune()`.
4. (Optionally) export the model via `$output()`, i.e., write the factorization matrices P and Q into files or return them as R objects.
5. Use the `$predict()` method to compute predicted values.

```
set.seed(1)
#recosystem input format:
train_reco <-  with(edx_train, data_memory(user_index = userId,
                                           item_index = movieId,
                                           rating     = rating))
test_reco  <-  with(edx_test,  data_memory(user_index = userId,
                                           item_index = movieId,
                                           rating     = rating))

r <-  recosystem::Reco()
#select best tuning parameters
opts <- r$tune(train_reco, opts = list(dim = c(10, 20, 30),
                                       lrate = c(0.1, 0.2),
                                       costp_l2 = c(0.01, 0.1),
                                       costq_l2 = c(0.01, 0.1),
                                       nthread  = 4, niter = 10))
r$train(train_reco, opts = c(opts$min, nthread = 4, niter = 20))

## iter      tr_rmse          obj
##    0       0.9813  1.1006e+007
##    1       0.8766  8.9991e+006
##    2       0.8436  8.3571e+006
##    3       0.8214  7.9676e+006
##    4       0.8046  7.6961e+006
##    5       0.7918  7.5021e+006
##    6       0.7813  7.3536e+006
##    7       0.7725  7.2358e+006
##    8       0.7649  7.1366e+006
##    9       0.7584  7.0576e+006
##   10       0.7527  6.9934e+006
##   11       0.7477  6.9359e+006
##   12       0.7430  6.8852e+006
##   13       0.7390  6.8399e+006
##   14       0.7354  6.8036e+006
##   15       0.7320  6.7693e+006
##   16       0.7288  6.7370e+006
##   17       0.7260  6.7100e+006
##   18       0.7235  6.6864e+006
##   19       0.7210  6.6628e+006

y_hat_reco <-  r$predict(test_reco, out_memory())
```

Evaluation of case 5:

```
RMSE_5 <- sqrt(mean((edx_test$rating - y_hat_reco)^2))
RMSE_5

## [1] 0.7855914446

RMSE_edx_test<-bind_rows(RMSE_edx_test,tibble(Method="Matrix
factorization",RMSE=RMSE_5))

options(pillar.sigfig=6)
RMSE_edx_test

## # A tibble: 5 x 2
##   Method                           RMSE
##   <chr>                           <dbl>
## 1 Lineal mov+us                0.864605
## 2 Linear mov+us+gen            0.864254
## 3 Regularization mov+us        0.864006
## 4 Regularization mov+us+gen 0.863701
## 5 Matrix factorization         0.785591
```

4. Final Validation

Target RMSE is achieved with regularization and matrix factorization for test set. Finally, this target must be achieved with the validation set. RMSE must remain below the target.

4.1. Validation variables

In this case, mu, bi, bu and bg should be calculated with the whole edx data set with the same procedure as calculated with train set and test set previously Nevertheless, the calculation of *lambda* is not repeated. The *lambda* targets are taken directly from the training process and then the regularized variables of edx data set calculated.

```
#VALIDATION. LINEAL VECTORS.
    o_avg_edx<-mean(edx$rating)
    movie_avg_edx<-edx%>%group_by(movieId)%>%summarize(b_mov_edx =
mean(rating - o_avg_edx))
    user_avg_edx<-
edx%>%left_join(movie_avg_edx,by='movieId')%>%group_by(userId)%>%
      summarize(b_user_edx = mean(rating - o_avg_edx-b_mov_edx))
    genres_avg_edx<-
edx%>%left_join(movie_avg_edx,by='movieId')%>%left_join(user_avg_edx,by='
userId')%>%
      group_by(genres)%>%summarize(b_genres_edx = mean(rating - o_avg_edx
- b_mov_edx-b_user_edx))

#VALIDATION. REGULARIZED VECTORS (lambda_target)
    movie_reg_edx<-edx%>%group_by(movieId)%>%summarize(b_mov_edx =
sum(rating - o_avg_edx)/(n()+lambda_target))
    user_reg_edx<-
edx%>%left_join(movie_reg_edx,by='movieId')%>%group_by(userId)%>%
      summarize(b_user_edx = sum(rating - o_avg_edx-
b_mov_edx)/(n()+lambda_target))

#VALIDATION. REGULARIZED VECTORS (lambda_target_2)
    movie_regg_edx<-edx%>%group_by(movieId)%>%summarize(b_mov_edx =
sum(rating - o_avg_edx)/(n()+lambda_target_2))
    user_regg_edx<-
edx%>%left_join(movie_reg_edx,by='movieId')%>%group_by(userId)%>%
      summarize(b_user_edx = sum(rating - o_avg_edx-
b_mov_edx)/(n()+lambda_target_2))
    genres_regg_edx<-
edx%>%left_join(movie_avg_edx,by='movieId')%>%left_join(user_avg_edx,by='
userId')%>%
      group_by(genres)%>%summarize(b_genres_edx = sum(rating - o_avg_edx
- b_mov_edx-b_user_edx)/(n()+lambda_target_2))
```

4.2. Linear model case 1. Movie and user effect

```
    y_hat_bm_bu_f <- validation %>%
    left_join(movie_avg_edx, by='movieId') %>%
    left_join(user_avg_edx, by='userId') %>%
    mutate(pred = o_avg_edx + b_mov_edx + b_user_edx) %>%
    .$pred
```

RMSE evaluation

```
#RMSE (linear user and movie)
  RMSE_1_v <- sqrt(mean((validation$rating - y_hat_bm_bu_f)^2))
  RMSE_1_v
```

```
## [1] 0.8655329194

  RMSE_validation<-tibble(Method="Lineal mov+us",RMSE=RMSE_1_v)
```

4.3. Linear model case 2. Movie, user and genres effect

```
    y_hat_bm_bu_fg <- validation %>%
    left_join(movie_avg_edx, by='movieId') %>%
    left_join(user_avg_edx, by='userId') %>%
    left_join(genres_avg_edx, by='genres') %>%
    mutate(pred = o_avg_edx + b_mov_edx + b_user_edx+b_genres_edx) %>%
    .$pred
```

RMSE evaluation

```
#RMSE (linear user and movie)
  RMSE_2_v <- sqrt(mean((validation$rating - y_hat_bm_bu_fg)^2))
  RMSE_2_v

## [1] 0.8651946465

  RMSE_validation<-bind_rows(RMSE_validation,tibble(Method="Linear
mov+us+gen",RMSE=RMSE_2_v))
```

4.4. Regularization. Case 3. Movie and user effect

```
  y_hat_bm_bu_reg <- validation %>%
    left_join(movie_reg_edx, by='movieId') %>%
    left_join(user_reg_edx, by='userId') %>%
    mutate(pred = o_avg_edx + b_mov_edx + b_user_edx) %>%
    .$pred
```

RMSE evaluation

```
#RMSE (regularization)
  RMSE_3_v <- sqrt(mean((validation$rating - y_hat_bm_bu_reg)^2))
  RMSE_3_v

## [1] 0.8649886915

  RMSE_validation<-bind_rows(RMSE_validation,tibble(Method="REg
mov+us",RMSE=RMSE_3_v))
```

4.5. Regularization. Case 4. Movie, user and genre effect

```
  y_hat_bm_bu_regg <- validation %>%
    left_join(movie_regg_edx, by='movieId') %>%
    left_join(user_regg_edx, by='userId') %>%
    left_join(genres_regg_edx, by='genres') %>%
    mutate(pred = o_avg_edx + b_mov_edx + b_user_edx+b_genres_edx) %>%
    .$pred
```

RMSE evaluation

```
  RMSE_4_v <- sqrt(mean((validation$rating - y_hat_bm_bu_regg)^2))
  RMSE_4_v

## [1] 0.8646905555

  RMSE_validation<-bind_rows(RMSE_validation,tibble(Method="REg
mov+us+gen",RMSE=RMSE_4_v))
```

4.6. Matrix factorization

```
  set.seed(1)
  edx_reco <-  with(edx, data_memory(user_index = userId,
                                     item_index = movieId,
                                     rating = rating))
  validation_reco  <-  with(validation, data_memory(user_index = userId,
                                                    item_index = movieId,
                                                    rating = rating))
  r <-  recosystem::Reco()
  opts <-  r$tune(edx_reco, opts = list(dim = c(10, 20, 30),
                                        lrate = c(0.1, 0.2),
                                        costp_l2 = c(0.01, 0.1),
                                        costq_l2 = c(0.01, 0.1),
                                        nthread  = 4, niter = 10))

  r$train(edx_reco, opts = c(opts$min, nthread = 4, niter = 20))

## iter      tr_rmse          obj
##    0       0.9703  1.1978e+007
##    1       0.8721  9.8798e+006
##    2       0.8389  9.1782e+006
##    3       0.8165  8.7455e+006
##    4       0.8004  8.4654e+006
##    5       0.7884  8.2657e+006
##    6       0.7787  8.1112e+006
##    7       0.7708  7.9938e+006
##    8       0.7640  7.8982e+006
##    9       0.7581  7.8179e+006
##   10       0.7531  7.7530e+006
##   11       0.7485  7.6945e+006
##   12       0.7444  7.6456e+006
##   13       0.7407  7.6028e+006
##   14       0.7374  7.5648e+006
##   15       0.7343  7.5296e+006
##   16       0.7315  7.4980e+006
##   17       0.7289  7.4724e+006
##   18       0.7265  7.4458e+006
##   19       0.7243  7.4251e+006

  y_hat_final_reco <-  r$predict(validation_reco, out_memory())
```

RMSE evaluation

```
  RMSE_5_v <- sqrt(mean((validation$rating - y_hat_final_reco)^2))

    RMSE_validation<-bind_rows(RMSE_validation,tibble(Method="Matrix
fact",RMSE=RMSE_5_v))
    options(pillar.sigfig=6)
RMSE_validation

## # A tibble: 5 x 2
##   Method                RMSE
##   <chr>                <dbl>
## 1 Lineal mov+us     0.865533
## 2 Linear mov+us+gen 0.865195
## 3 REg mov+us        0.864989
## 4 REg mov+us+gen    0.864691
## 5 Matrix fact       0.783416
```

5. Conclusion

The simplest model (cases 1 and 2) does not achieve the RMSE target, but predictions fall close. (0.8653 and 0.86495). The regularization technique gets 0.864989 and 0.864691 (cases 3 and 4 respectively). RMSE target is achieved with RMSE below 0.8649 in case 4 (regularization and genres effect) and falls close in case 3. Finally, the recosystem package that implements an algorithm that achieved the RMSE of 0.7825, which is clearly below target.

5.1. Best and worst movies. Regularization

5.1.1. Top 10 best movies:

```
validation %>%
  left_join(movie_regg_edx, by='movieId') %>%
  left_join(user_regg_edx, by='userId') %>%
  left_join(genres_regg_edx, by='genres') %>%
  mutate(pred = o_avg_edx + b_mov_edx + b_user_edx+b_genres_edx) %>%
  arrange(-pred) %>%
  group_by(title) %>%
  select(title) %>%
  head(10)

## # A tibble: 10 x 1
## # Groups:   title [8]
##    title
##    <chr>
##  1 Shawshank Redemption, The (1994)
##  2 Godfather, The (1972)
##  3 Goodfellas (1990)
##  4 Godfather, The (1972)
##  5 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)
##  6 Pulp Fiction (1994)
##  7 Blade Runner (1982)
##  8 Schindler's List (1993)
##  9 Annie Hall (1977)
## 10 Shawshank Redemption, The (1994)
```

5.1.2. Top 10 worst movies:

```
validation %>%
  left_join(movie_regg_edx, by='movieId') %>%
  left_join(user_regg_edx, by='userId') %>%
  left_join(genres_regg_edx, by='genres') %>%
  mutate(pred = o_avg_edx + b_mov_edx + b_user_edx+b_genres_edx) %>%
  arrange(pred) %>%
  group_by(title) %>%
  select(title) %>%
  head(10)
```

```
## # A tibble: 10 x 1
## # Groups:   title [9]
##    title
##    <chr>
##  1 Police Academy 5: Assignment: Miami Beach (1988)
##  2 RoboCop 3 (1993)
##  3 Iron Eagle IV (1995)
##  4 Eye of the Beholder (1999)
##  5 Turbo: A Power Rangers Movie (1997)
##  6 Pokemon 4 Ever (a.k.a. PokÃ©mon 4: The Movie) (2002)
##  7 Island of Dr. Moreau, The (1996)
##  8 Airport 1975 (1974)
##  9 Police Academy 5: Assignment: Miami Beach (1988)
## 10 Slaughterhouse 2 (1988)
```

5.2. Best and worst movies. Matrix factorization

5.2.1. Top 10 best movies:

```
tibble(title = validation$title, rating = y_hat_final_reco) %>%
  arrange(-rating) %>%
  group_by(title) %>%
  select(title) %>%
  head(10)
```

```
## # A tibble: 10 x 1
## # Groups:   title [9]
##    title
##    <chr>
##  1 Manufacturing Consent: Noam Chomsky and the Media (1992)
##  2 Year of the Horse (1997)
##  3 Becket (1964)
##  4 Sexual Life of the Belgians, The (La Vie sexuelle des Belges 1950-
1978) (199~
##  5 Cats Don't Dance (1997)
##  6 Passion of the Christ, The (2004)
##  7 Pulp Fiction (1994)
##  8 Shawshank Redemption, The (1994)
##  9 Pulp Fiction (1994)
## 10 Star Wars: Episode V - The Empire Strikes Back (1980)
```

5.2.2. Top 10 worst movies:

```r
tibble(title = validation$title, rating = y_hat_final_reco) %>%
  arrange(rating) %>%
  group_by(title) %>%
  select(title) %>%
  head(10)
```

```
## # A tibble: 10 x 1
## # Groups:   title [10]
##    title
##    <chr>
##  1 Beast of Yucca Flats, The (1961)
##  2 Pearl Harbor (2001)
##  3 Attack of the Crab Monsters (1957)
##  4 Message in a Bottle (1999)
##  5 Texas Chainsaw Massacre, The (1974)
##  6 Norbit (2007)
##  7 Boys, Les (1997)
##  8 Lizzie McGuire Movie, The (2003)
##  9 Rugrats Go Wild! (2003)
## 10 Dirty Love (2005)
```

5.3. Limitations

The ideal recommendation system should consider more information such as timestamp or other aspects. Nevertheless, a common laptop may have problems as the number of variables and rows of ratings increases. Furthermore, the more accurate technique matrix factorization takes many resources from common laptops. Therefore, the algorithms based on recosystem must be run by powerful laptops. The current algorithm is based on these users, movies, genres and ratings. However, new ratings would change the algorithm. The model would be similar but the optimization must be repeated for each method.

5.4. Future work

Apart from matrix factorization (package recosystem) there are techniques such as collaborative filtering that are really used for these challenges.

References

1. Rafael A. Irizarry (2019), Introduction to Data Science: Data Analysis and Prediction Algorithms with R

2. Yixuan Qiu (2017), recosystem: recommendation System Using Parallel Matrix Factorization