# Movie Recommendation System Report in R

Inga Aritenco

02/09/2021

## INTRODUCTION

Present days data science became a powerful tool for understanding the modern dataistic[1] world. The most popular method for data analysis is machine learning, which is a subfield of artificial intelligence. Machine learning algorithms handles the data into research-intelligible manner to see the real picture of a problem, whether it be speech recognition technology, spam and malware detectors, or movie recommendation systems.

Recommender systems are widely used in electronic commerce. The online streaming services like Amazon Prime, YouTube and Netflix apply statistical techniques to make the proper recommendation to customer's needs, based on their preferences, likes and interactions with a website or even other users.

The innovative development for business and internet technologies in the field of recommender systems started in 2006, when Netflix launched a contest for 1 million dollar prize[2]. The challenge was accepted in September 2009 by „BellKor's Pragmatic Chaos" team which successfully improved the predictions of movie recommendation system by 10% and their reached value of RMSE was 0,8554.

This report uses a 10M version of the MovieLens dataset, created by GroupLens[3] research lab in the Department of Computer Science and Engineering at the University of Minnesota, specializing in recommender systems.

The goal of this project is to train a machine learning algorithm using the inputs in one subset to predict movie ratings in the validation set. The dataset will be separated to train (edx) and test (validation) sets at the start of the data analysis. Root Mean Square Error (RMSE) will be used for computation of predictions with the true values in the final test set (validation). The edx data should be split into separate training and test sets to design and test the chosen algorithm. The final result employs recosystem package using parallel matrix factorization and includes RMSE.

The structure of analysis consists of the following parts:

1. Introduction
2. Analysis | Data Preparation | Data Overview | Visualization
3. Methods and Data Models | Results
4. Conclusion

## ANALYSIS

### Data Preparation

To start working with the MovieLens[4] dataset download the required packages. (If you are an advanced user in R and RStudio, the libraries might be pre-installed. If it is, simply use them.)

[1]https://en.wikipedia.org/wiki/Dataism
[2]https://www.netflixprize.com/index.html
[3]https://grouplens.org/about/what-is-grouplens/
[4]https://grouplens.org/datasets/movielens/10m/

When the process is done, download the dataset and split it into two subsets: `edx`(training set) and `validation`(testing set). Validation set should have 10% of the MovieLens data, and at the same time edx set should be 90%.

Afterwards, divide the `edx` data into `training`(90%) and `test`(10%) sets.

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

# If the packages are already pre-installed, simply use them without downloading
library(tidyverse)
library(caret)
library(data.table)
library(DescTools)
library(lubridate)
library(recosystem)
library(ggthemes)
library(scales)

# 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")

# R 4.0 or later:
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")

# 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)`
```

```r
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)
```

## Data Overview

Firstly, check the structure of the loaded 10M version of MovieLens dataset, released in 2009. It consists of 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users.

`edx` set consists of 9,000,055 rows, meanwhile `validation` set has 999,999 rows.

As shown below, there is 6 columns in both datasets. The most necessary columns in our exploration are the next:

`userId` which contains the number for each user,

`movieId` which contains the number for each film, and also

`title` of the movies, and their `genres`.

Consequently, were obtained the unique quantity of `users`(69,878) and `movies`(10,677).

Were detected a top 10 most rated movies which are cinematographic classics.

```r
# First look at the edx dataset
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1:                Comedy|Romance
## 2:         Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

```r
# The number of rows and columns in the edx (training) set
dim(edx)
```

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

```r
# The number of movies in edx dataset
edx %>% group_by(movieId) %>% summarize(count = n())
```

```
## # A tibble: 10,677 x 2
##    movieId count
##      <dbl> <int>
##  1       1 23790
##  2       2 10779
##  3       3  7028
##  4       4  1577
##  5       5  6400
##  6       6 12346
##  7       7  7259
##  8       8   821
##  9       9  2278
## 10      10 15187
## # ... with 10,667 more rows
```

```r
# The number of different users in the edx dataset
edx %>% group_by(userId) %>% summarize(count = n())
```

```
## # A tibble: 69,878 x 2
##    userId count
##     <int> <int>
##  1      1    19
##  2      2    17
##  3      3    31
##  4      4    35
##  5      5    74
##  6      6    39
##  7      7    96
##  8      8   727
##  9      9    21
## 10     10   112
## # ... with 69,868 more rows
```

```r
# Unique users and movies
edx %>% summarize(n_users = n_distinct(userId), n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   69878    10677
```

```r
# Movies that have the most ratings
edx %>% group_by(title) %>%
  summarize(n_ratings = n()) %>%
  arrange(desc(n_ratings))
```

```
## # A tibble: 10,676 x 2
##    title                                                          n_ratings
##    <chr>                                                              <int>
##  1 Pulp Fiction (1994)                                                31362
##  2 Forrest Gump (1994)                                                31079
##  3 Silence of the Lambs, The (1991)                                   30382
##  4 Jurassic Park (1993)                                               29360
##  5 Shawshank Redemption, The (1994)                                   28015
##  6 Braveheart (1995)                                                  26212
##  7 Fugitive, The (1993)                                         25998
##  8 Terminator 2: Judgment Day (1991)                            25984
##  9 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)     25672
## 10 Apollo 13 (1995)                                             24284
## # ... with 10,666 more rows
```

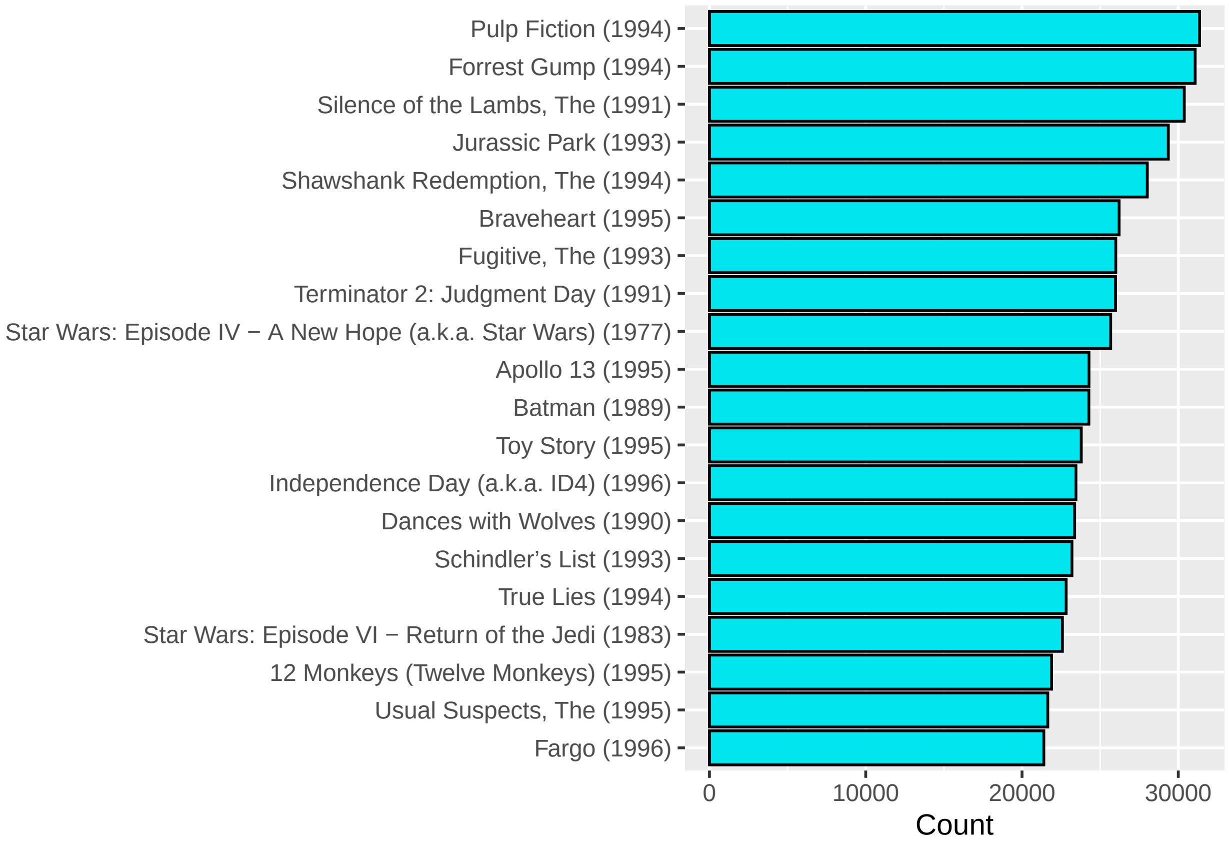
```r
# Validation dataset observations by column
glimpse(validation)
```

```
## Rows: 999,999
## Columns: 6
## $ userId    <int> 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, ~
## $ movieId   <dbl> 231, 480, 586, 151, 858, 1544, 590, 4995, 34, 432, 434, 85, ~
## $ rating    <dbl> 5.0, 5.0, 5.0, 3.0, 2.0, 3.0, 3.5, 4.5, 5.0, 3.0, 3.0, 3.0, ~
## $ timestamp <int> 838983392, 838983653, 838984068, 868246450, 868245645, 86824~
## $ title     <chr> "Dumb & Dumber (1994)", "Jurassic Park (1993)", "Home Alone ~
## $ genres    <chr> "Comedy", "Action|Adventure|Sci-Fi|Thriller", "Children|Come~
```

## Visualization

The plot shows the movies with the most ratings. It clearly illustrates that the most rated film is `Pulp Fiction`(1994).

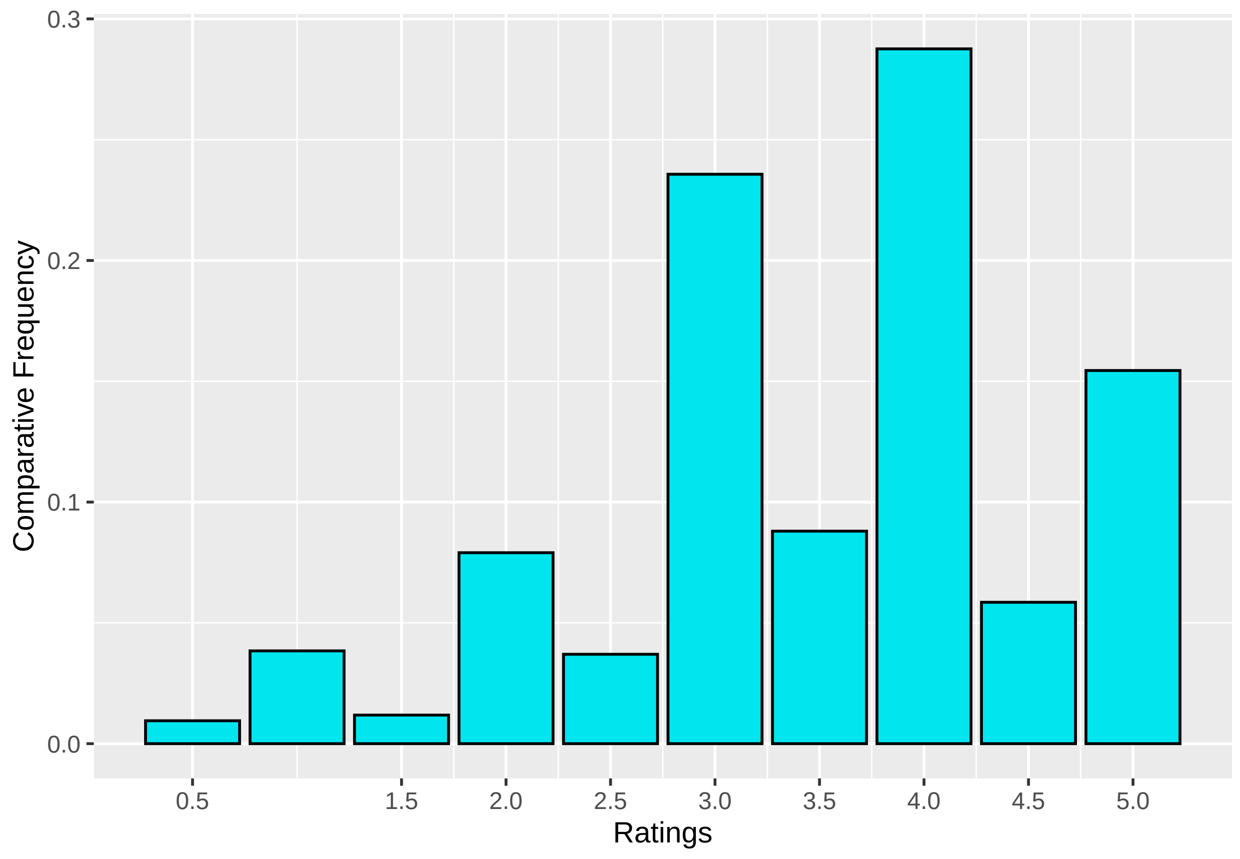
```r
# The most popular movies in the dataset
edx %>%
  group_by(title) %>%
  summarize(count = n()) %>%
  arrange(-count) %>%
  top_n(20, count) %>%
  ggplot(aes(count, reorder(title, count))) +
  geom_bar(color = "black", fill = "turquoise2", stat = "identity") +
  xlab("Count") +
  ylab(NULL) +
  theme_grey()
```

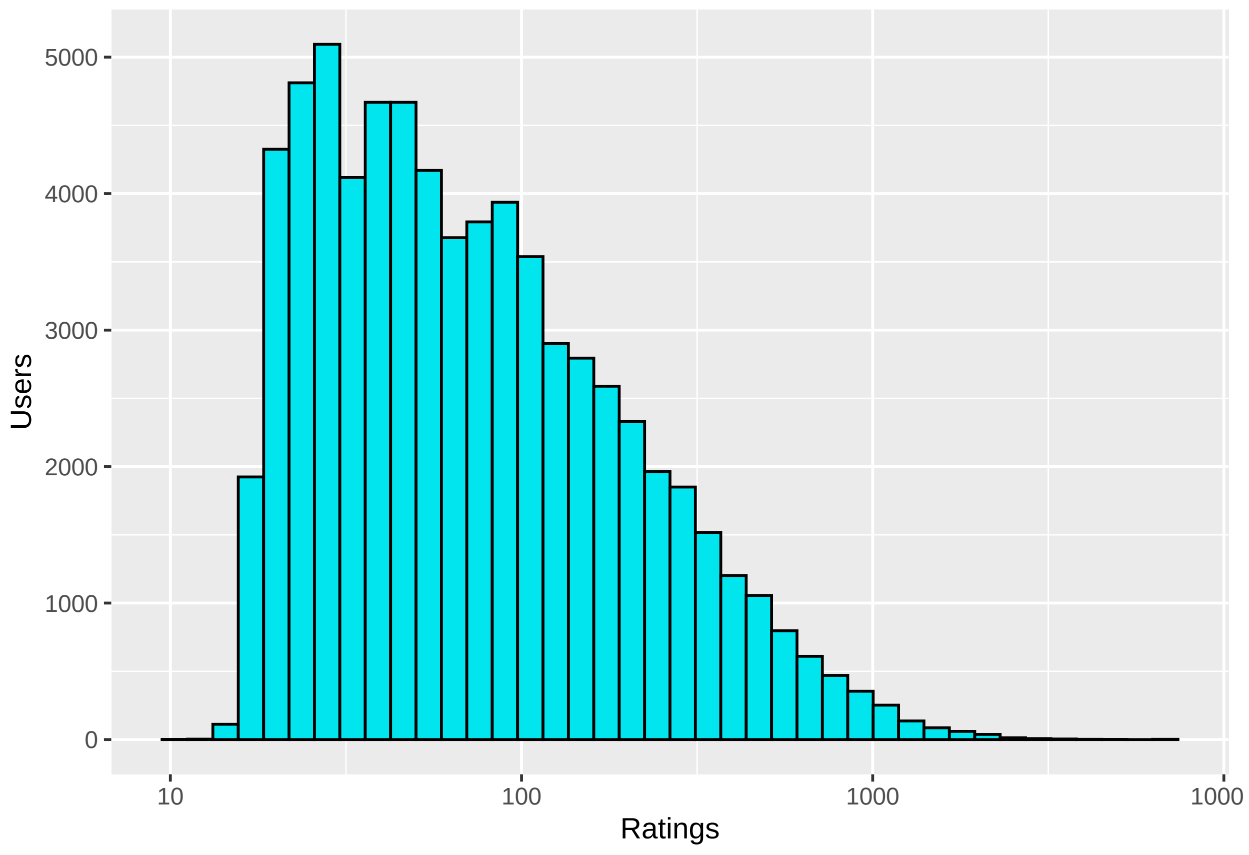


Every movie has its own rating selected by any user. The value of rating starts from 0.5 to 5.

Here is the code provided and the following chart. The plot shows that the most common rating is 4.

```r
# The distribution of movie ratings with range from 0.5 to 5
edx %>% ggplot(aes(rating, y = ..prop..)) +
  geom_bar(color = "black", fill = "turquoise2") +
  labs(x = "Ratings", y = "Comparative Frequency") +
  scale_x_continuous(breaks = c(0.5, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5)) +
  theme_grey()
```


0.3
0.2
0.1
0.0
Comparative Frequency
0.5
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Ratings


5000
4000
3000
2000
1000
0
Users
10
100
1000
1000
Ratings

This histogram shows the right distribution of ratings versus users.

## METHODS AND DATA MODELS

Before to start modeling a movie recommendation system, here is a brief explanation of the methods used in this report.

#### Loss Function

The `loss function` measures the difference between the compared values and the predicted values with the actual outcome. The best choice in this analysis is to use RMSE, which stands for `Root Mean Squared Error`. This particular metric was used in the solution of the Netflix Prize challenge.

Here is the formula:

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

$N$ is defined as the number of `user` and `movie` combinations;

$y_{u,i}$ is defined as the rating for `movie` $i$ and `user` $u$;

$\hat{y}_{u,i}$ is a prediction for `movie` $i$ by `user` $u$;

### Model 1: Average of all ratings

A model used here implement the average value of all ratings for every movie and every user.

The primitive model of the simplest movie recommendation system employs to an equation:

$$Y_{u,i} = \mu + \epsilon_{u,i}$$

where

$Y_{u,i}$ is the predicted rating of `user` $u$ and `movie` $i$;

$\mu$ is the average of the two above-mentioned values;

$\epsilon_{u,i}$ independent errors sampled from the same distribution centered at 0.

Average outcome of all ratings is 3,512

```
# Compute the overall average rating using the training set `edx`
mu <- mean(edx$rating)
mu
```

```
## [1] 3.512465
```

Next, compute a naive RMSE of all unknown ratings with $\mu$. Construct a table with the first method.

```
naive_rmse <- RMSE(validation$rating, mu)
results <- tibble(Method = "Model 1: Overall Average", RMSE = naive_rmse)
results %>% knitr::kable()
```

| Method | RMSE |
|---|---|
| Model 1: Overall Average | 1.061202 |

Obtained result does not correspond to the desired prediction. The RMSE is to high, therefore it is naive.

Our score should be 0,864

## Model 2: Movie Effects

To improve the first model, add the term(`bias`) $b_i$ which represents average ranking for movie $i$. Here is the updated formula:

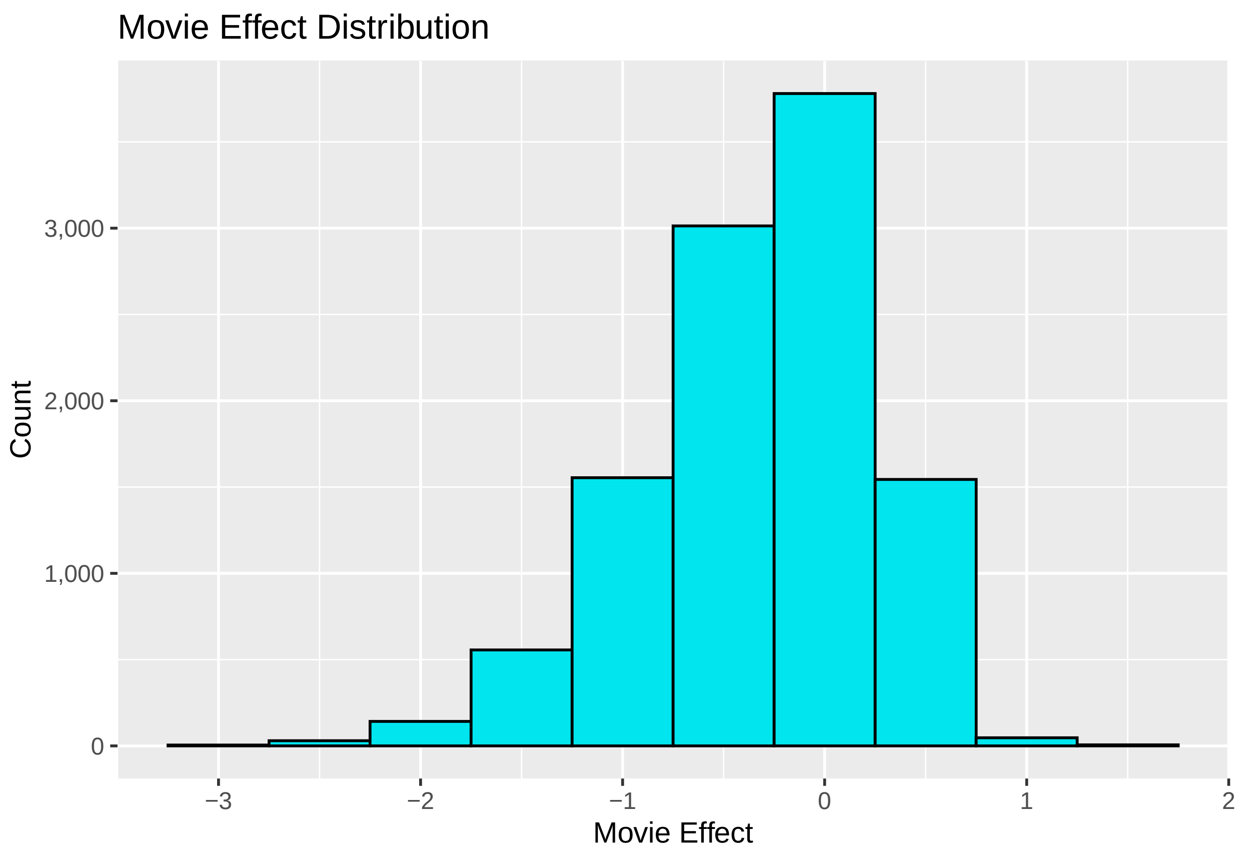
$$Yu,i = \mu + b_i + \epsilon_{u,i}$$

The following code explains the equation. Updated RMSE indicates a little improvement for our model.

```
# Represent average ranking for movie i, by adding term b_i (bias)
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# Predict the ratings with movie bias
predicted_ratings <- mu + validation %>%
  left_join(b_i, by = "movieId") %>%
  pull(b_i)

# Calculate RMSE of movie ranking effect
movie_bs_rmse <- RMSE(predicted_ratings, validation$rating)

# Show results in a tibble
movie_bs_rmse <- RMSE(predicted_ratings, validation$rating)
results <- bind_rows(results, tibble(Method = "Model 2: Average + Movie Effect",
                                     RMSE = movie_bs_rmse))
results %>% knitr::kable()
```

| Method | RMSE |
|---|---|
| Model 1: Overall Average | 1.0612018 |
| Model 2: Average + Movie Effect | 0.9439087 |

The following histogram shows that the movie effect distribution is normally skewed.



### Model 3: Movie and User Effects

See the interconnection between `movie`$(b_i)$ and `user`$(b_u)$ effects. Here was taken into account a fact that some people can like almost all movies, rather some individuals can have a dissenting opinion.

Adding user effect to the equation:

$$Yu, i = \mu + b_i + b_u + \epsilon_{u,i}$$

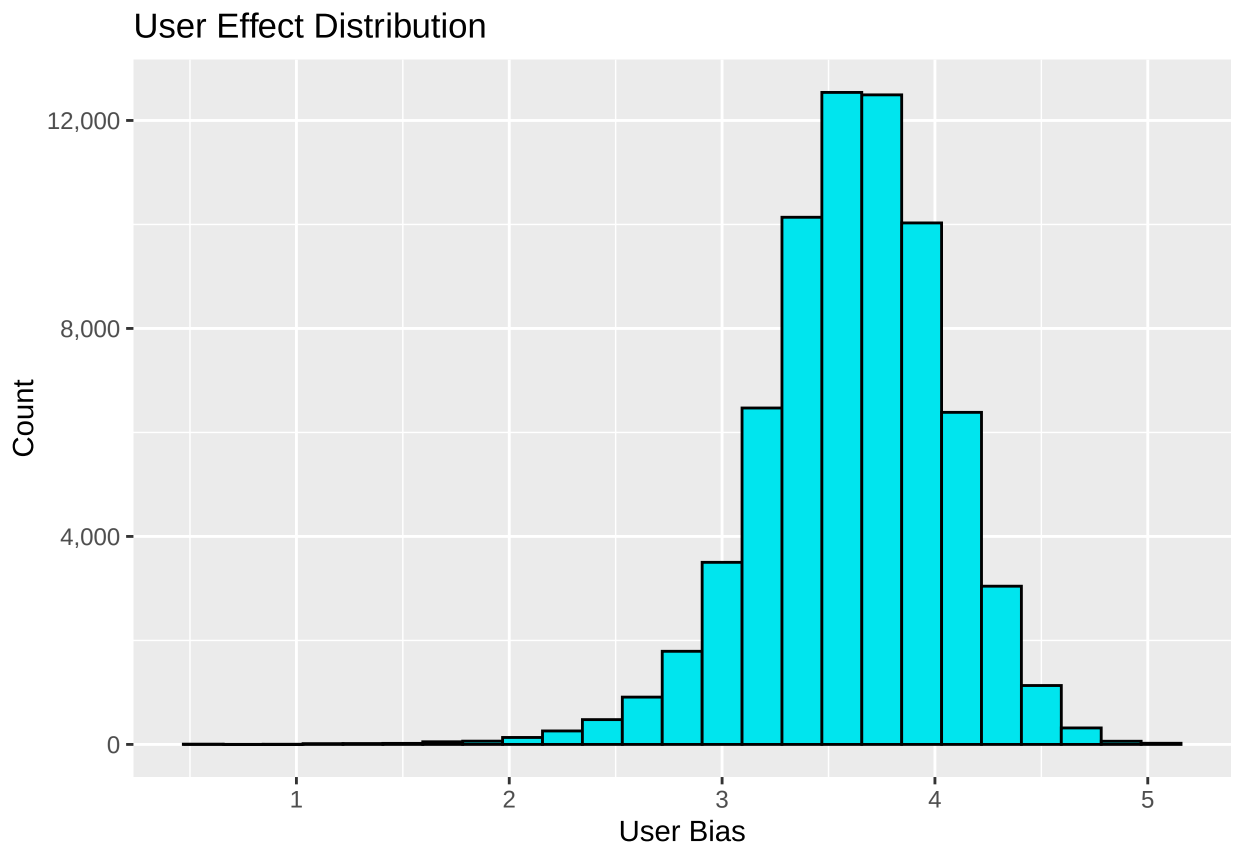
```r
# Compute user bs(bias) effect, b_u
b_u <- edx %>%
  left_join(b_i, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

# Predict new ratings with movie and user bs(bias)
predicted_ratings <- validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
```

Updated RMSE is 0,865. We are closer to reach the goal.

```r
# Calculate RMSE of movie and user ranking effect
user_bs_rmse <- RMSE(predicted_ratings, validation$rating)
results <- bind_rows(results, tibble(Method = "Model 3: Mean + Movie Bias + User Effect",
                                     RMSE = user_bs_rmse))
results %>% knitr::kable()
```

| Method | RMSE |
|---|---|
| Model 1: Overall Average | 1.0612018 |
| Model 2: Average + Movie Effect | 0.9439087 |
| Model 3: Mean + Movie Bias + User Effect | 0.8653488 |

Here is visualization of the user effect distribution.



## Model 4: Regularized Movie and User Effects

The next step is a `regularization` method. This form of regression avoids unwanted errors and model overfitting in the predictions.

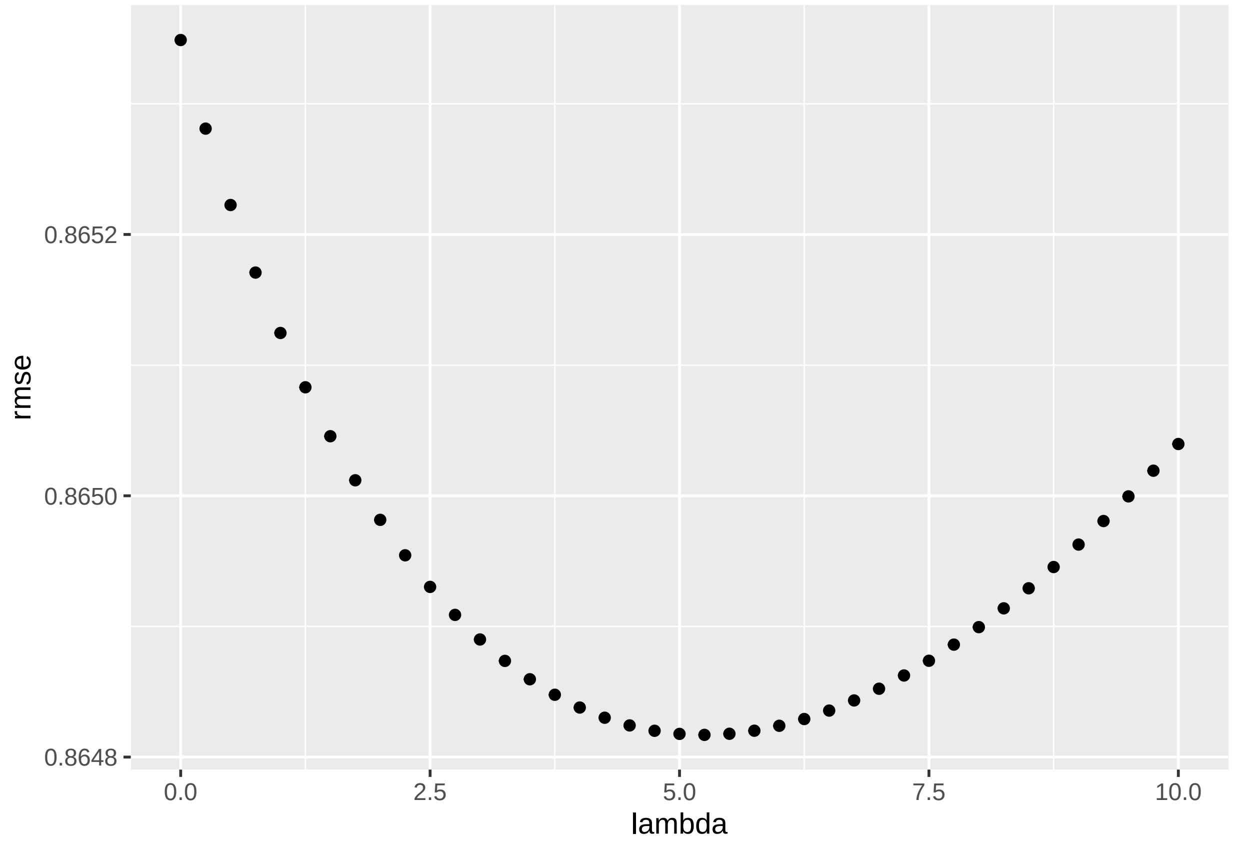
To compute regularized estimates of $b_i$ and $b_u$ use a new tuning parameter $\lambda$, which is a penalty factor.

Follow the equation:

$$\frac{1}{N}\sum_{u,i}(Y_{u,i}-\mu-b_i-b_u)^2+\lambda(\sum_i b_i^2+\sum_u b_u^2)$$

```r
# Determine the most appropriate lambda defined by sequence
lambda <- seq(from = 0, to = 10, by = 0.25)

## Repeat the previous actions, but with regularization
## Receive RMSE output of each lambda

rmse <- sapply(lambda, function(l){
  # Calculate average rating on training set
  mu <- mean(edx$rating)
  # Compute regularized movie bs term
  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  # Compute regularize user bs term
  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  # Compute predictions on validation set as per the terms of reference
  predicted_ratings <- validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)
  # Output RMSE of the predictions
  return(RMSE(predicted_ratings, validation$rating))
})
```

Quick plot of RMSE in relation to $\lambda$.



The minimum value of RMSE is

```
## [1] 0.864817
```

Find the best value of lambda to the linear model with regularized `movie` and `user` effects.

```r
min_lambda <- lambda[which.min(rmse)]
```

The minimum lambda is:

```
## [1] 5.25
```

Let's have a look at the predictions output:

```r
RMSE(predicted_ratings, validation$rating)
```

```
## [1] 0.8653488
```

```r
results <- bind_rows(results, tibble(Method = "Model 4: Regularized Movie and User Effects",
                                     RMSE = min(rmse)))
results %>% knitr::kable()
```

| Method | RMSE |
|---|---:|
| Model 1: Overall Average | 1.0612018 |
| Model 2: Average + Movie Effect | 0.9439087 |
| Model 3: Mean + Movie Bias + User Effect | 0.8653488 |
| Model 4: Regularized Movie and User Effects | 0.8648170 |

Regularization method slightly improved our model. But we already reached our goal. The value of RMSE is 0,864

## Model 5: Parallel Matrix Factorization with `recosystem` package

As stated at the beginning of our analysis, the purpose of this report is to get as close as possible to the Netflix Prize challenge results. More precisely, to reach the target value of RMSE to 0,8554

Thereby, parallel matrix factorization should be used with `recosystem`[5] library.

Matrix factorization is a simple embedding model. The main task of recommender system is to predict unknown entries in the rating matrix based on observed values. The concept is to approximate a whole rating matrix $R_{m \times n}$ (where $_m$ is the number of users; $_n$ is the number of movies;) into the product of two matrices of lower dimensions $P_{k \times m}$ and $Q_{k \times n}$, such that

$$R \approx P'Q$$

The script shown below describes how to use the `recosystem` package:

```r
# Create training(edx) and test(validation) sets using `recosystem` package
train_reco <- with(edx, data_memory(user_index = userId,
                                    item_index = movieId, rating = rating))
test_reco <- with(validation, data_memory(user_index = userId,
                                          item_index = movieId, rating = rating))
# Create the model object
r = Reco()

# Select the 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),
```

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

```
                                          nthread = 4, niter = 10))

# Train the algorithm
# Please wait. The process may take some time.
r$train(train_reco, opts = c(opts$min, nthread = 4, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9723   1.2001e+07
##    1       0.8730   9.8900e+06
##    2       0.8385   9.1684e+06
##    3       0.8157   8.7435e+06
##    4       0.8003   8.4642e+06
##    5       0.7888   8.2707e+06
##    6       0.7791   8.1163e+06
##    7       0.7711   7.9974e+06
##    8       0.7643   7.9007e+06
##    9       0.7585   7.8231e+06
##   10       0.7533   7.7544e+06
##   11       0.7488   7.6984e+06
##   12       0.7446   7.6477e+06
##   13       0.7410   7.6045e+06
##   14       0.7377   7.5675e+06
##   15       0.7345   7.5330e+06
##   16       0.7317   7.5014e+06
##   17       0.7290   7.4743e+06
##   18       0.7266   7.4496e+06
##   19       0.7244   7.4272e+06
```

```
# Calculate the prediction
y_hat_final_reco <- r$predict(test_reco, out_memory())

# Update the results table
result <- bind_rows(results,
                    tibble(Method = "Model 5: Matrix Factorization with recosystem",
                           RMSE = RMSE(validation$rating, y_hat_final_reco)))
# Show the RMSE final result
result %>% knitr::kable()
```

| Method | RMSE |
|---|---|
| Model 1: Overall Average | 1.0612018 |
| Model 2: Average + Movie Effect | 0.9439087 |
| Model 3: Mean + Movie Bias + User Effect | 0.8653488 |
| Model 4: Regularized Movie and User Effects | 0.8648170 |
| Model 5: Matrix Factorization with recosystem | 0.7824981 |

The best prediction during data exploration became the `parallel matrix factorization` method. Aquired RMSE is 0,782 which is rather than regularized linear model.

In the end, it is interesting to discover top 5 best movies and the top 5 worst movies reached by using the best algorithm(`parallel matrix factorization`).

Top 5 best movies:

```
## # A tibble: 5 x 1
## # Groups:   title [4]
##   title
##   <chr>
## 1 Schindler's List (1993)
## 2 Lord of the Rings: The Return of the King, The (2003)
## 3 Shawshank Redemption, The (1994)
## 4 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)
## 5 Shawshank Redemption, The (1994)
```

Top 5 worst movies:

```
## # A tibble: 5 x 1
## # Groups:   title [5]
##   title
##   <chr>
## 1 Time Walker (a.k.a. Being From Another Planet) (1982)
## 2 Beast of Yucca Flats, The (1961)
## 3 Daddy Day Camp (2007)
## 4 Dead Girl, The (2006)
## 5 Free Willy 2: The Adventure Home (1995)
```

## CONCLUSION

Firstly, were prepared data for the analysis. Afterwards were explored the first insights to continue work with the datasets.

Secondly, were visualized informative findings.

Thirdly, were briefly explained the methods of data modeling. Were predicted the best value of RMSE.

### Limitations

During data analysis were observed some issues in processing speed of the `matrix factorization` algorithm. Despite the demanding resources of the laptop the prediction achieved a better result than was expected.

Several models were investigated. We found out that `regularized linear model` can reach the target RMSE also, without much sources of the computer.

### Future work

There is also other algorithms and methods to compute the RMSE value. For instance, artificial neural networks, collaborative filtering, `recommenderlab`[6] package for building and testing movie recommendation systems etc.

[6]https://cran.r-project.org/web/packages/recommenderlab/vignettes/recommenderlab.pdf