MovieLens project

Viktoriia Pylypets_Romaniuk

2023-03-19

Recommendation system project using the MovieLens dataset

1.Introduction

The goal of MovieLens project is to train a machine learning algorithm, which will predict the movie rating, that certain user will give. For estimating accuracy of this algorithm we will compute the root mean squared error(RMSE).In this project we aim to achieve the RMSE result less than 0.86490.In the project we try 2 approaches to solve the problem - model that uses movie and user effect and matrix factorization model, which is a popular technique to solve the recommendation system problem.

2.Data preparation

For this project we use MovieLens dataset, that has 10M ratings. We get it from GroupLens site. On this site we can find datasets with information collected from MovieLens site. MovieLens(http://movielens.org) is a web site that helps people find movies to watch. It has a huge number of registered users, that give ratings to movies.

First of all we need download the dataset and unzip it. After we create the edx dataset for training purposes and final holdout test for testing our final model.

```
############################################################
# Create edx and final_holdout_test sets
############################################################

# Note: this process could take a couple of minutes

if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6      v purrr   0.3.4
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.4.1
## v readr   2.1.3      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caret
## Loading required package: lattice
##
## Attaching package: 'caret'
##
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

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

```
## Loading required package: recosystem
```

```
## Warning: package 'recosystem' was built under R version 4.2.2
```

```
library(tidyverse)
library(caret)
library(recosystem)

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

options(timeout = 120)

dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)
```

After downloading and unzipping we create edx set(90%) - for training and testing purposes of different models and final_holdout set(10%) - for our final model testing. Also we make sure that all users and movies that in final_holdout test also in edx set.

```
ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE),
                         stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

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

# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
```

used

```
# set.seed(1) # if using R 3.5 or earlier
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 final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

```
edx <- rbind(edx, removed)
# remove from memory unnecessary information
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

3.Data exploratory analysis

Let's start to explore edx dataset.We can see that it is a table in tidy format and contains 9000055 rows and 6 columns with names userId, movieID, rating, timestamp, title and genres. It has 10677 different movies and 69878 different users.

```
#Check the number of columns and rows in edx set
dim(edx)
```

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

```
#Have a look of first rows of edx set and names of columns
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
```

```
#Check the number of different movies
n_distinct(edx$movieId)
```

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

```
# Check the number of different users
n_distinct(edx$userId)
```

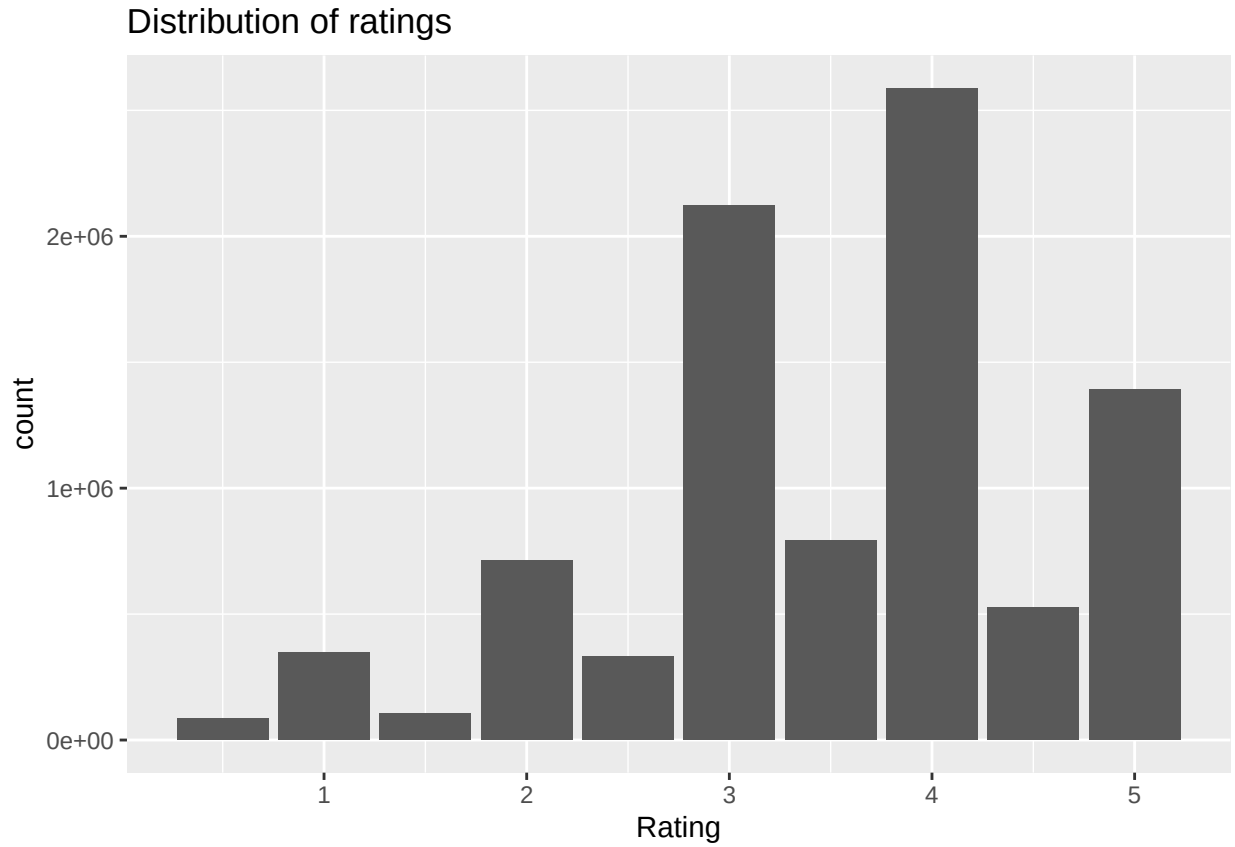
```
## [1] 69878
```

Also we check if our dataset has NA. It does not.

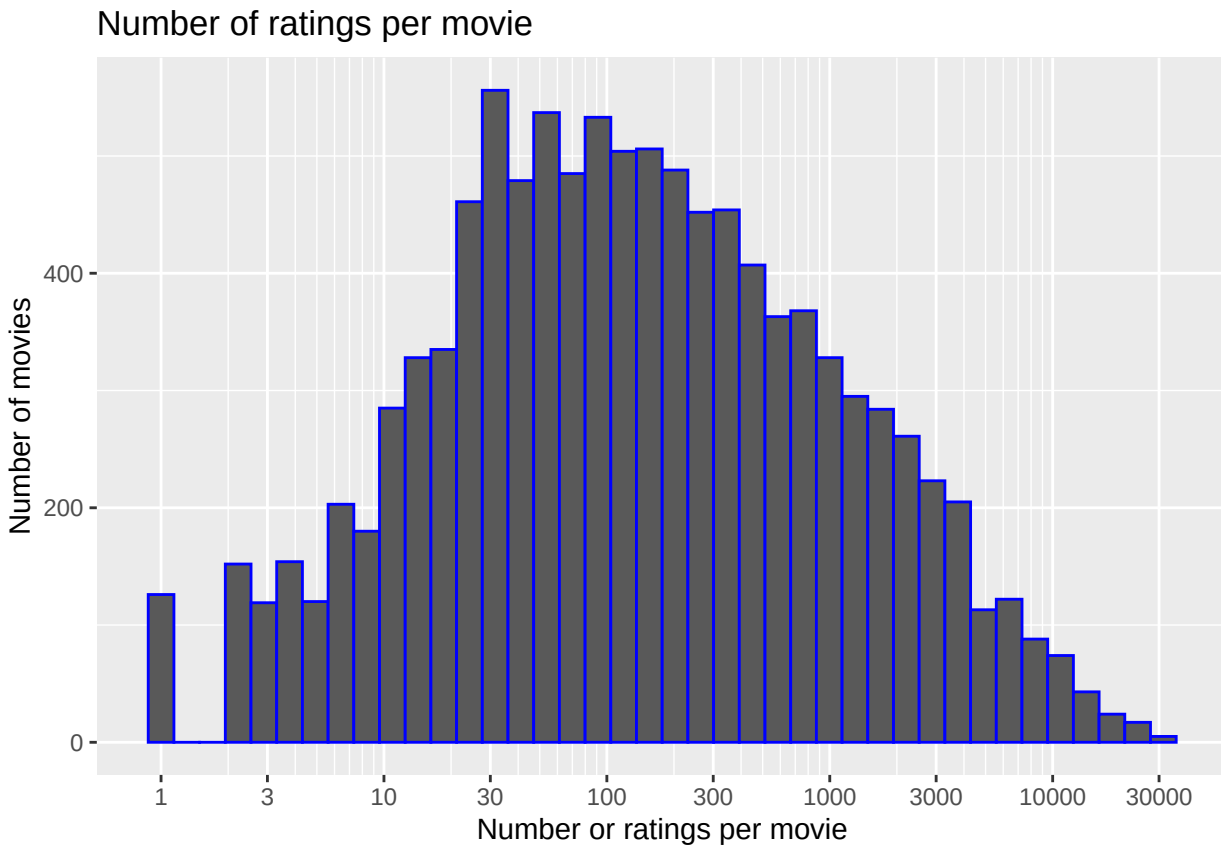
```
# Check if we have some NA
sum(is.na(edx))
```

```
## [1] 0
```

Next, we would like to explore how ratings are distributed. For this purpose we will make a plot. From it we can see that user a willing to give an integer instead of half-starred rating.The most often given rating is 4.

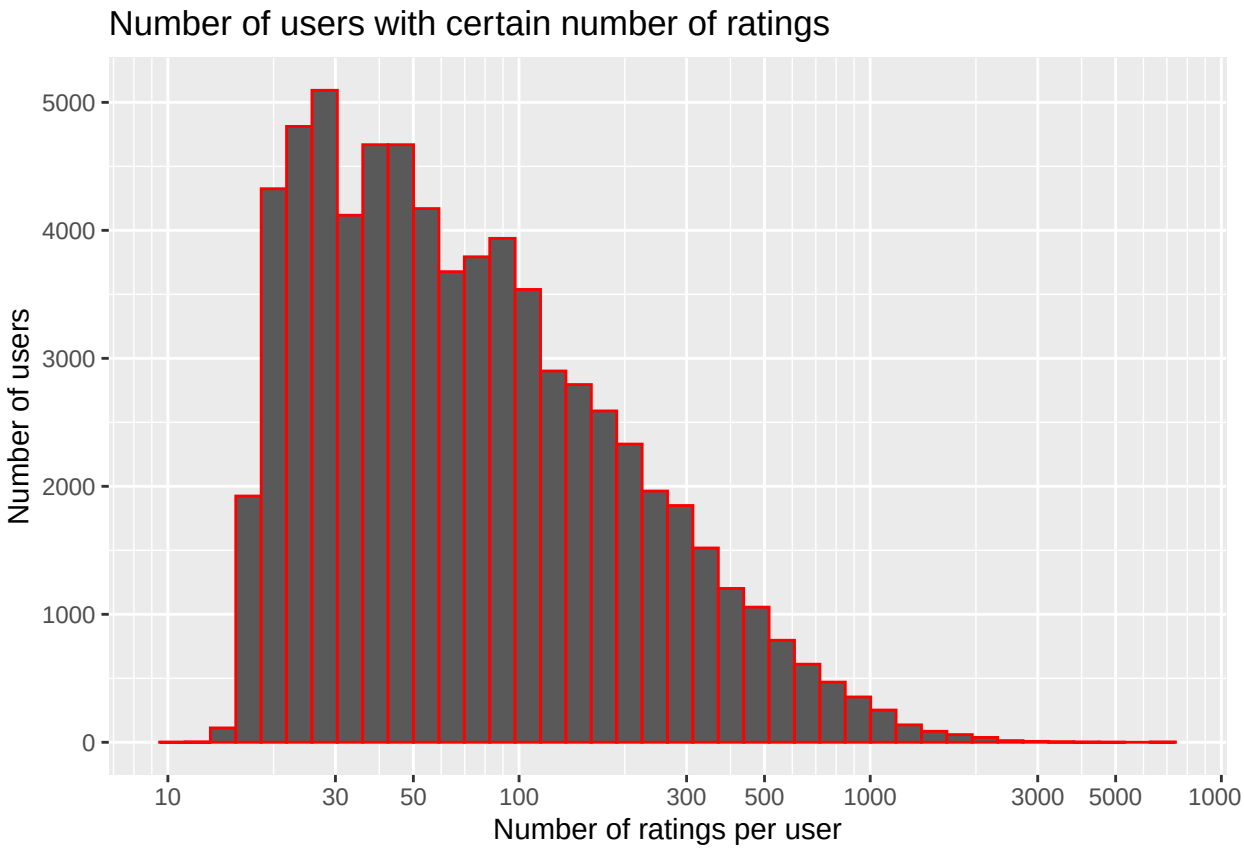


On the figure "Number of ratings per movie" we can see that we have movies, that were rated more than 10000 times and movies with a very few ratings. Most of the movies have 10-3000 ratings. Some movies have more



than 10000 ratings

Figure “Number of ratings per user” shows us that some users are more active than others. Most of them rated



20-500 movies.

4.Methods and analysis For training our models we create train set(which consist 80% of edx set) and test_set(20% of edx set) to test how our models perform. Also we make sure that we do not include movies and users in test set, which do not appear in train set.

```
#Create the test set and train set from edx set
test_index_1 <- createDataPartition(y = edx$rating, times = 1,
                                    p = 0.2, list = FALSE)
train_set <- edx[-test_index_1,]
temp_test_set <- edx[test_index_1,]

#Make sure we do not include movies and users in the test set that do not appear in the train set
test_set <- temp_test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

# Add rows removed from the testing set back into the training set set
removed <- anti_join(temp_test_set, test_set)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

```
train_set <- rbind(train_set, removed)
```

For calculating RMSE (root mean squared error) we create a function:

```
#Function for calculating RMSE
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Next, we calculate the average rating for all users across all movies. In the train set it is 3.512

```
#Compute the average rating for all users across all movies
mu <- mean(train_set$rating)
mu
```

```
## [1] 3.51257
```

4.1.Just the average model

The first model we try is the model, where all predicted ratings are the average rating across all movies. In the table we can see that just the average model gives us RMSE=1.06, which is not our goal.

```
#Compute  RMSE if we assign all predicted ratings to the average rating
naive_rmse <- RMSE(test_set$rating, mu)
naive_rmse
```

```
## [1] 1.060704
```

```
#Make tibble with RMSE results
model_result <- tibble(Model = "Just the average",
                       RMSE = naive_rmse)
model_result
```

```
## # A tibble: 1 x 2
##   Model             RMSE
##   <chr>            <dbl>
## 1 Just the average  1.06
```

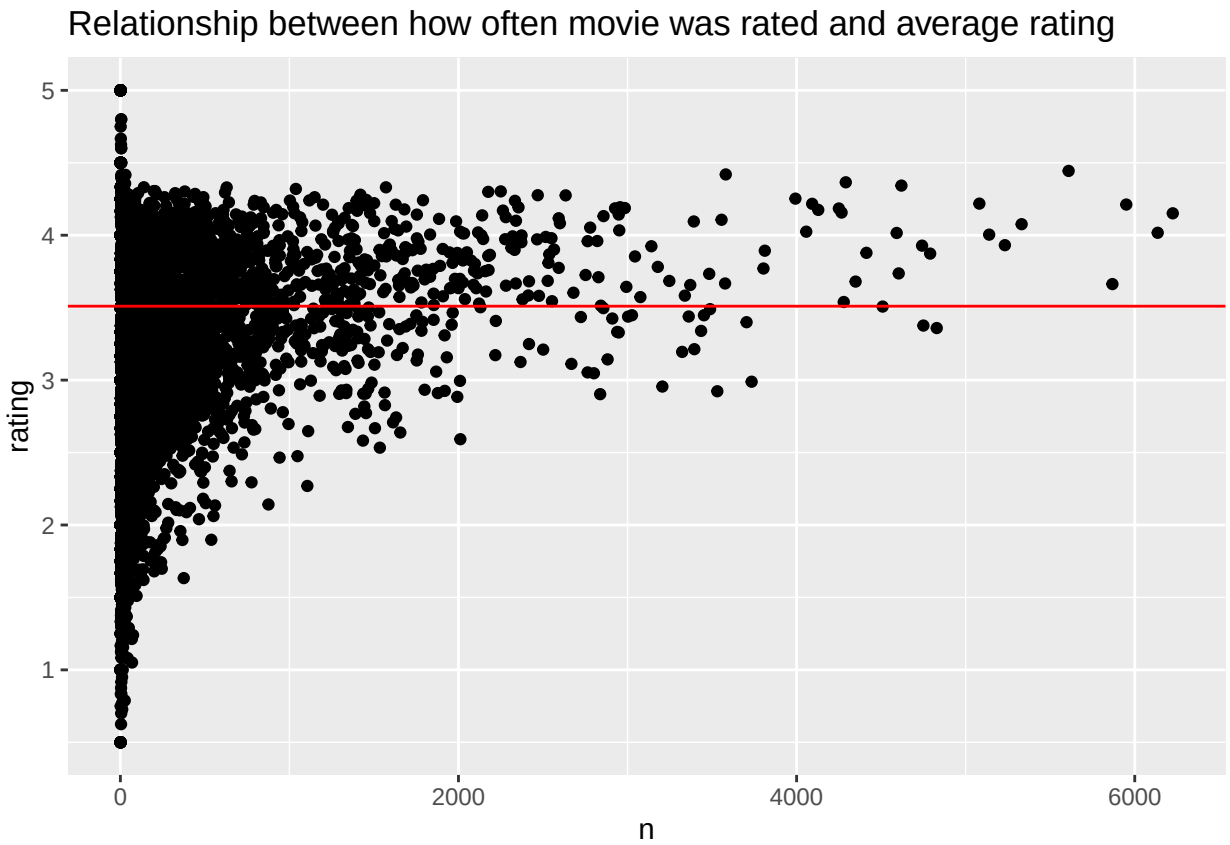
4.2. Movie and user effect model

Let's try to find out if some movies have higher than average ratings and if there some relationship how often movie was rated and was rating it got. Because we can assume that more popular movies have more ratings and their rating is somewhat higher than the average rating across all movies. For this purpose we built a plot, where we can see a relationship between how often movie was rated and its rating. To make more clear is this rating above or less than average across all movies we add an intercept red line. From this figure we can see that movies with a lot of ratings tend to have higher average rating.

```
#Check if there are some relationship between number of movie ratings and average rating of the movie
num_rate <-test_set %>%
  group_by(movieId) %>%
  summarize(n = n(),
            rating = mean(rating))
head(num_rate)
```

```
## # A tibble: 6 x 3
##   movieId     n rating
##     <int> <int>  <dbl>
## 1       1  4742   3.93
## 2       2  2218   3.17
## 3       3  1403   3.13
## 4       4   330   2.77
## 5       5  1259   3.09
## 6       6  2532   3.81
```

```
num_rate %>% ggplot(aes(n, rating)) + geom_point() + geom_hline(yintercept = 3.51, col = "red")+
  ggtitle("Relationship between how often movie was rated and average rating")
```

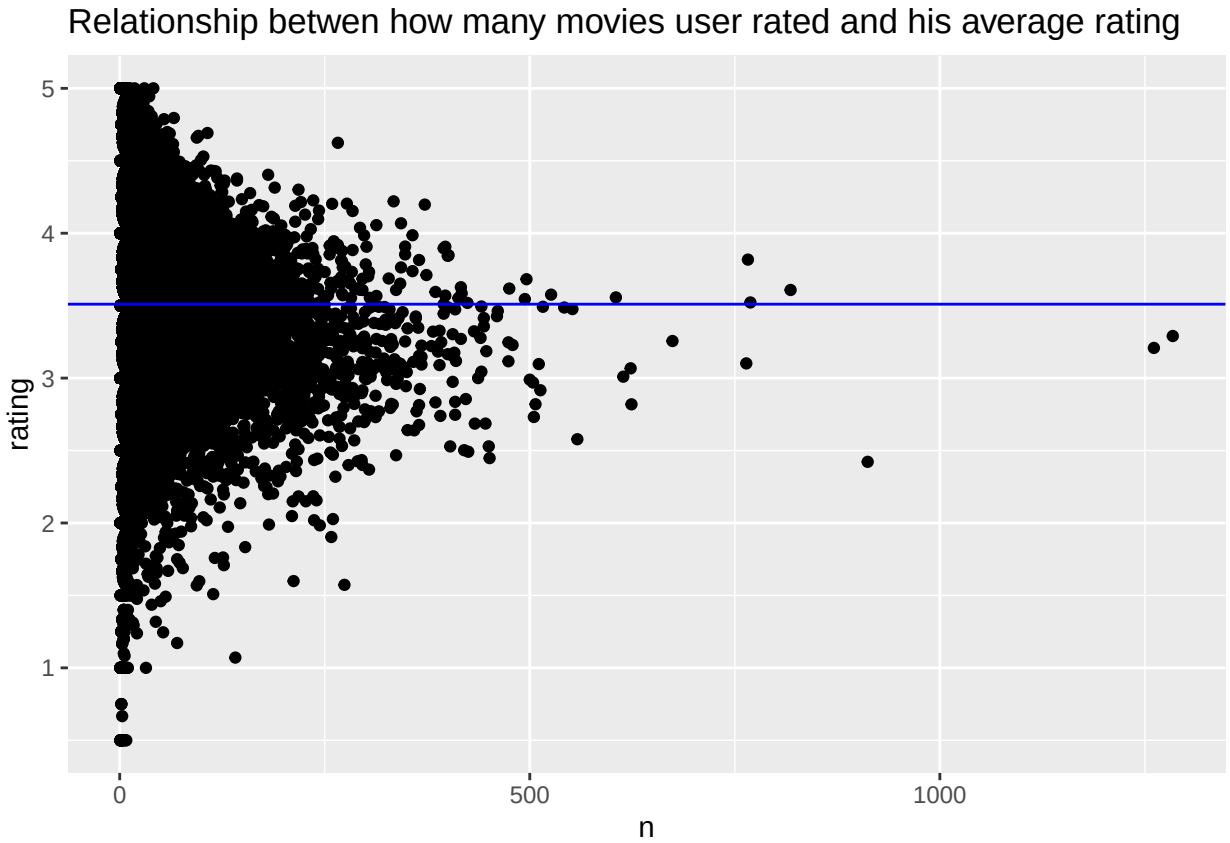


Next, let's check the situation if user rated more movies, how it affects the average rating this user gives. We plot number of ratings per user versus average rating user gives with blue intercept that shows average rating for all users across all movies. We can see an almost normal distribution of ratings with a very slight tendency to give a little bit lower ratings for users who rated a lot of movies.

```
# Check if there are some relationship between how many movies user rated and
#average rating this user gives to movies
user_rate <- test_set%>%
  group_by(userId) %>%
  summarize(n = n(),
            rating = mean(rating))
head(user_rate)
```

```
## # A tibble: 6 x 3
##   userId     n rating
##    <int> <int>  <dbl>
## 1      1     2   5
## 2      2     5   3.2
## 3      3     8   3.56
## 4      4     7   3.57
## 5      5    12   4.08
## 6      6     8   4
```

```
user_rate %>% ggplot(aes(n, rating)) + geom_point() + geom_hline(yintercept = 3.51, col = "blue")+
  ggtitle("Relationship betwen how many movies user rated and his average rating")
```



4.2.1.Movie effect model From the previous figure "Relationship between how often movie was rated and average rating" we can see some movies have higher rating than others. For our movie effect model we calculate b_i, which is the difference between average rating across all movies and average rating of certain movie.Our movie effect model predicts the ratings simply by adding to average rating across all movie the value of b_i, that this movie has. This model gives us RMSE = 0.9437, which is better than just the average model.

```
#calculate b_i (a difference between average movie rating and average rating across all movies)
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
# Calculate predicted rating by adding movie effect b_i to average rating across all movies
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  mutate(pred = mu + b_i) %>%
  .$pred
head(predicted_ratings) # just for checking that we got what we want
```

```
## [1] 2.867006 3.628773 3.284787 3.704053 3.991580 3.667716
```

```
# calculate movie model RMSE
model_movie_rmse <- RMSE(predicted_ratings, test_set$rating)
model_movie_rmse
```

```
## [1] 0.9437144
```

```
# Add movie model RMSE to the table
model_result <- bind_rows(model_result, tibble(Model = "Movie effect",
                                               RMSE = model_movie_rmse))
```

```
model_result
```

```
## # A tibble: 2 x 2
##   Model              RMSE
##   <chr>             <dbl>
## 1 Just the average 1.06
## 2 Movie effect     0.944
```

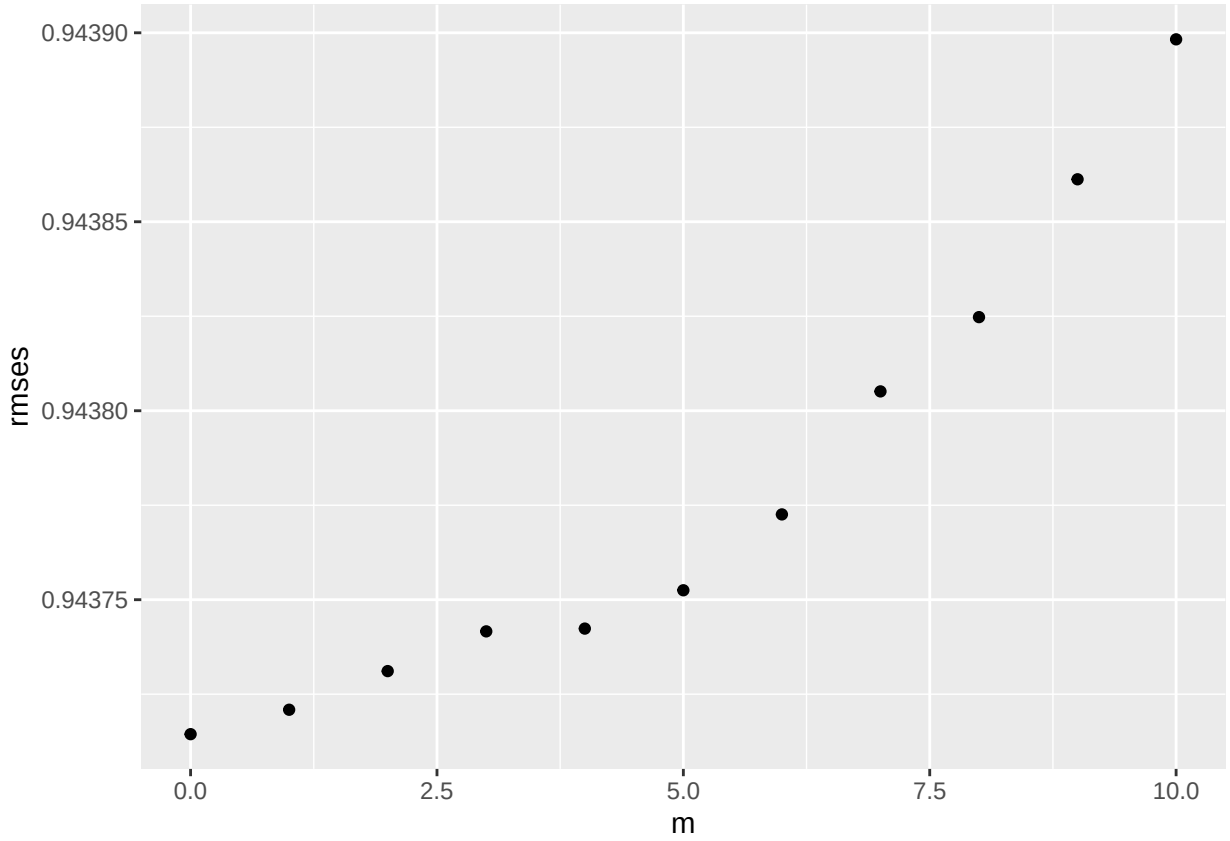
Let's check how this model perform if we will predict for movie just the average across all movies(b_i = 0), if this movie has less than m ratings; and average + b_i, if it has more. We try values for m from 0 to 10. On the plot we can see that this model doesn't give us any improvements in RMSE.

```
# check what result it will give, if we calculate b-i(movie effect) in the case when movie has
# more than m ratings, and if less, we assign the predicted rating to average rating across all movies

m <- seq(0, 10, 1)

rmses <- sapply(m, function(m){
  movie_avgs_reg <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = ifelse(n()>m,mean(rating - mu),0))

  predicted_ratings <- test_set %>% left_join(movie_avgs_reg, by='movieId') %>%
    mutate(pred = mu + b_i) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})
qplot(m, rmses)
```




```
m[which.min(rmses)]
```

```
## [1] 0
```

4.2.2.Movie-user model

Next we will try to add to our movie effect model user effect. In this model we assume that some users are tend to give lover ratings and some - higher.For this model we calculate b_u - the difference between the average rating certain user gives and the average rating across all movies minus movie effect b_i.From the result table we can see that adding user effect to our model give us better RMSE. Now it is 0.866

```
# calculate b_u(the difference between average rating the user gives and average rating across all movi
# and b_i)
movie_user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
# Calculate predicted ratings, using b_u
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(movie_user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
# Calculate RMSE for movie-user model
model_movie_user_rmse <- RMSE(predicted_ratings, test_set$rating)
model_movie_user_rmse
```

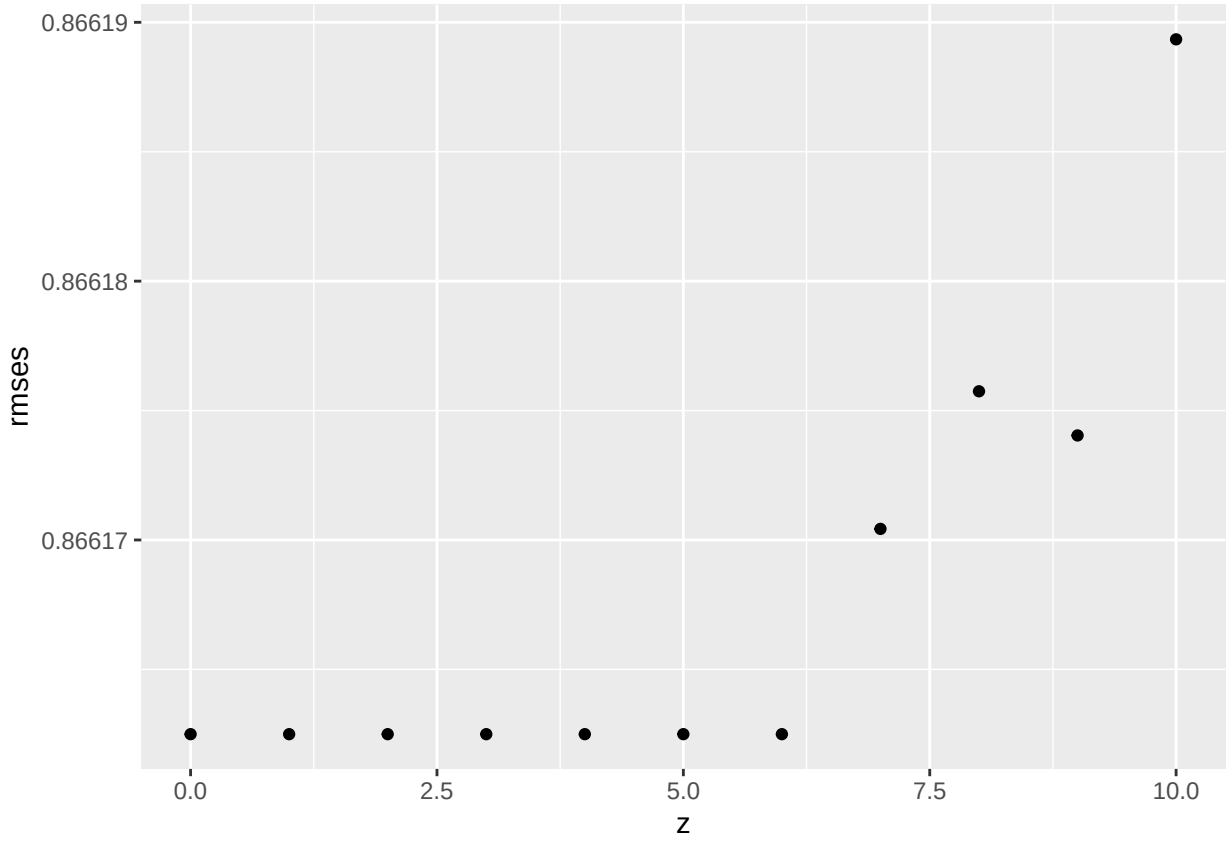
```
## [1] 0.8661625
```

```
#Add RMSE results of movie_user model to results table
model_result <- bind_rows(model_result, tibble(Model = "Movie-user effect",
                                               RMSE = model_movie_user_rmse))
model_result
```

```
## # A tibble: 3 x 2
##   Model              RMSE
##   <chr>             <dbl>
## 1 Just the average  1.06
## 2 Movie effect      0.944
## 3 Movie-user effect 0.866
```

Let's check if we will see some improvements, if in our model use user effect, if user rated more than z movies.From the plot we can see that we have almost the same RMSE, if we don't use user effect in situation when user rated 6 and less movies.

```
#check, if we will ignore users who rated less than z movies
z<- seq(0,10,1)
rmses <- sapply(z, function(z){
  movie_user_avgs_reg <- train_set %>%
    left_join(movie_avgs, by='movieId') %>%
    group_by(userId) %>%
    summarize(b_u = ifelse(n() > z, mean(rating - mu - b_i),0))
  predicted_ratings <- test_set %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(movie_user_avgs_reg, by='userId') %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})
qplot(z, rmses)
```




```
z[which.min(rmses)]
```

```
## [1] 0
```

4.3. Parallel Matrix Factorization model using recosystem package

4.3.1.Recommender system with default parameters

A popular technique to solve the recommender system problem is the matrix factorization method.For that we use recosystem package.Recosystem is an R wrapper of the LIBMF library developed by Yu-Chin Juan, Wei-Sheng Chin, Yong Zhuang, Bo-Wen Yuan, Meng-Yuan Yang, and Chih-Jen Lin, an open source library for recommender system using parallel matrix factorization.

LIBMF is a high-performance C++ library for large scale matrix factorization. LIBMF itself is a parallelized library, meaning that users can take advantage of multicore CPUs to speed up the computation. It also utilizes some advanced CPU features to further improve the performance. (Chin, Yuan, et al. 2015)

For training and testing this model we have to change the data format.For training we use model with default parameters.This recommender model gives us RMSE=0.8345, which is much better than previous models.

```
set.seed(5, sample.kind = "Rounding")
```

```
## Warning in set.seed(5, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
# Return data format for train_set
train_reco <- with(train_set, data_memory(user_index = userId, item_index = movieId,
                                          rating = rating))
```

```
#Return data format for test set
test_reco <- with(test_set, data_memory(user_index = userId, item_index = movieId,
                                        rating = rating))

#Return an object of class "RecoSys"
recommendation_system <- Reco()

#Train the model with default parameters
recommendation_system$train(train_reco)
```

```
## iter      tr_rmse          obj
##    0       0.9716   1.1952e+07
##    1       0.8834   1.0678e+07
##    2       0.8693   1.0595e+07
##    3       0.8503   1.0399e+07
##    4       0.8433   1.0332e+07
##    5       0.8392   1.0296e+07
##    6       0.8358   1.0273e+07
##    7       0.8330   1.0254e+07
##    8       0.8305   1.0236e+07
##    9       0.8283   1.0220e+07
##   10       0.8266   1.0209e+07
##   11       0.8253   1.0200e+07
##   12       0.8241   1.0194e+07
##   13       0.8231   1.0182e+07
##   14       0.8224   1.0177e+07
##   15       0.8217   1.0175e+07
##   16       0.8211   1.0169e+07
##   17       0.8206   1.0166e+07
##   18       0.8201   1.0161e+07
##   19       0.8197   1.0160e+07
```

```
#Calculate predicted ratings with default model
predicted_ratings <-  recommendation_system$predict(test_reco, out_memory())

#Calculate RMSE of default matrix factorization model
reco_model <- RMSE(predicted_ratings, test_set$rating)
reco_model
```

```
## [1] 0.834549
```

```
#Add RMSE results of reco_model to the results table
model_result <- bind_rows(model_result, tibble(Model = "Matrix fact",
                                               RMSE = reco_model))
model_result
```

```
## # A tibble: 4 x 2
##   Model              RMSE
##   <chr>             <dbl>
## 1 Just the average  1.06
## 2 Movie effect      0.944
## 3 Movie-user effect 0.866
## 4 Matrix fact       0.835
```

4.3.2.Recommender system with tuned parameters

We can see that previous model gives a result, that we wanted. Let's explore, if tuning this model will give us even better results.We tune number of latent factors and the learning rate. Tuned model gives us even better RMSE=0.790 with latent factors = 30 and learning rate = 0.1.

```
# Tune the model parameters, using cross validation - takes time(around 20 min)
set.seed(5, sample.kind = "Rounding")
```

```
## Warning in set.seed(5, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
tuning <- recommendation_system$tune(train_reco, opts = list(dim = c(20L, 30L),
                                                             lrate = c(0.05, 0.1),
                                                             nthread  = 6,
                                                             niter = 10))
# Check which tuning parameters work the best
tuning$min
```

```
## $dim
## [1] 30
##
## $costp_l1
## [1] 0
##
## $costp_l2
## [1] 0.01
##
## $costq_l1
## [1] 0
##
## $costq_l2
## [1] 0.1
##
## $lrate
## [1] 0.1
##
## $loss_fun
## [1] 0.8067558
```

```
#Train the model using tuned parameters
recommendation_system$train(train_reco, opts = c(tuning$min,
                                                 nthread = 4,
                                                 niter = 25))
```

```
## iter      tr_rmse          obj
##    0       0.9947   1.0034e+07
##    1       0.8786   8.0694e+06
##    2       0.8466   7.5022e+06
##    3       0.8251   7.1486e+06
##    4       0.8084   6.9060e+06
##    5       0.7955   6.7305e+06
##    6       0.7845   6.5925e+06
##    7       0.7752   6.4767e+06
##    8       0.7671   6.3832e+06
##    9       0.7601   6.3067e+06
##   10       0.7538   6.2385e+06
```

```
##   11       0.7483   6.1826e+06
##   12       0.7433   6.1332e+06
##   13       0.7389   6.0891e+06
##   14       0.7348   6.0518e+06
##   15       0.7310   6.0170e+06
##   16       0.7276   5.9861e+06
##   17       0.7244   5.9586e+06
##   18       0.7215   5.9339e+06
##   19       0.7187   5.9103e+06
##   20       0.7162   5.8895e+06
##   21       0.7138   5.8709e+06
##   22       0.7116   5.8534e+06
##   23       0.7096   5.8358e+06
##   24       0.7077   5.8203e+06
```

```
# Calculate the predicted ratings using tuned model
predicted_ratings <-  recommendation_system$predict(test_reco, out_memory())

#Calculate RMSE of tuned model
reco_model_tuned <- RMSE(predicted_ratings, test_set$rating)
reco_model_tuned
```

```
## [1] 0.7898583
```

```
#Add results of tuned recosystem model to the results table
model_result <- bind_rows(model_result, tibble(Model = "Matrix fact tuned",
                                               RMSE = reco_model_tuned))
model_result
```

```
## # A tibble: 5 x 2
##   Model              RMSE
##   <chr>             <dbl>
## 1 Just the average  1.06
## 2 Movie effect      0.944
## 3 Movie-user effect 0.866
## 4 Matrix fact       0.835
## 5 Matrix fact tuned 0.790
```

5.Results.Tuned matrix factorization model testing on the final holdout test

As we can see the tuned matrix factorization model gives us the best RMSE.Let's check, what result this model give us on final holdout test. We can see, that RMSE is very close to result, what we had on the test set. Using this model we get RMSE = 0.7898, which is according to our assessment conditions, is a good result.

```
## Check the results on Final holdout test set using tuned recosystem model##

#Prepare data format for the recosystem model
final_test_reco <- with(final_holdout_test,data_memory(user_index = userId,
                                                       item_index = movieId,
                                                       rating= rating))
# Compute predicted ratings
final_predicted_ratings <- recommendation_system$predict(final_test_reco, out_memory())

# Compute RMSE
reco_model_final_rmse <- RMSE(final_predicted_ratings, final_holdout_test$rating)
```

```
reco_model_final_rmse
```

```
## [1] 0.7897793
```

6.Conclusion

In the project we tried two ways to get RMSE lower than 0.86499. The formula in the first movie-user model looks like this: Y(i,u) = average(mu) + movie-effect(b_i) + user-effect(b_u), where: Y(i,u) - predicted rating for movie by certain user average(mu) - average movie rating across all users for all movies movie-effect(b_i) - the difference between average movie rating and average rating across all users and all movies user-effect(b_u) - average rating certain user gives minus sum of average(mu) and minus movie-effect(b_i) The movie-user model gives us RMSE = 0.8661, which is a little bit more than we wanted. Also this model will not give us a good result, if we tried it on the data with new user or new movies.

The second model we tried on is parallel matrix factorization using recosystem package. As shown by its performance on the test and final holdout test, recosystem model with default parameters gives us RMSE = 0.8345 and tuned recosystem model with latent factors = 30 and learning rate = 0.1 gives us better results: test set RMSE = 0.7898 and final holdout set RMSE = 0.7897. We can see, that our tuned recosystem model gives the same good RMSE result on the final holdout test as on the test set.

7. Future work

Using recosystem package, we can continue our work with tuning parameters as costp_l1, costp_l2, costq_l1, costq_l2 dim and lrate. More iterations could give us better RMSE results (but take more resources for computing).

References

James Bennet, Stan Lanning.The Netflix Prize, 2007

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

Yixuan Qui, David Cortes, Chin-Jen Lin, Yu-Chin Juan,Wei-Sheng Chin,Yong Zhuang, Bo-Wen Yuan, Meng-Yuan Yang, and other contributors.Recommender system using Matrix Factorization. Package 'recosystem', 2022