Movielens Porject

Hatem RABEH

2022-11-06

1. Scope:

This is a Movie recommendation system project created in the process of obtaining the Data Science professional certificate from Harvardx University.

2. Introduction

A recommendation system is a subclass of information filtering systems that seeks to predict the "preference" or "rating" that a user would attribute to an item. Recommendation systems are used in several fields such as music, news, research articles, search queries, movies, and products in general. Companies that sell products to a wide range of customers like Netflix uses the recommendation system to predict how many stars a user will give to a specific movie. One star represents a bad rating, whereas five stars represent an excellent one. In this project, we will combine several machine-learning strategies, learned from the Harvardx data science program, to construct a movie recommendation system using the Movie Lens data set, collected by GroupLens Research

2.1. Dataset presentation:

The Full version of the MovieLens dataset contains 27,000,000 ratings and 1,100,000 tag applications applied to 58,000 movies by 280,000 users. Includes tag genome data with 14 million relevance scores across 1,100 tags (Last updated 9/2018). For more information visit: https://grouplens.org/datasets/movielens/%20latest/. For this project, the dataset used is the MovieLens 10M movie ratings, a Stable benchmark dataset of 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users. Released in 2009 (https://grouplens.org/datasets/movielens/10m/). This dataset is split into two sub-datasets: EDX subset (90%) and validation subset(10%). The EDX dataset will be used for developing the algorithm and model construction and the validation data set to assess the performance of the final model. The value that will be used to evaluate the algorithm performance is the root mean square error or the RMSE (for more information visit: Root-mean-square deviation - Wikipedia). The evaluation criteria for this algorithm is an RMSE expected to be lower than 0.86490.

2.2. Method and steps to be implemented:

1) Downloading the MovieLens dataset
2) Split it into Edx and validation subsets
3) Analyze the Edx subset
4) Develop modeling concepts
5) Evaluate the models developed by using the RMSE
6) Train the best model using the Edx subset and evaluate it using the validation subset

3. Methods and analysis

3.1. Downloading the MovieLens dataset

The MovieLens 10M can be downloaded from this link: (https://grouplens.org/datasets/movielens/10m/).

```
dl <- tempfile()
download.file("https://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_tibble(movies) %>% mutate(movieId = as.numeric(movieId),
title = as.character(title),
genres = as.character(genres))


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

3.2. Splitting the MovieLens dataset into EDX and validation datasets:

To predict the movie rating, the MovieLens dataset is split into two subsets :

- Edx: a training subset (90%) and,
- Validation of the test subset (10%)

```
# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding")
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)
```

3.3. Analyzing the Edx subset:

3.3.1. Overall analysis: The following table 1 shows the first 5 rows of the "edx" subset :

```
## 3.3.1.Overall analysis:
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
```

The Edx subset contains 9,000,055 obs and 6 variables with ratings provided by a total of 69,878 unique users for a total of 10,677 unique movies. Each row represents a rating given by one user to one movie (see table 2) :

```
### Summery of Edx subset:#
edx_subset_summary <- tibble(rows_number = nrow(edx),
                     columns_number = ncol(edx),
                     users_number= n_distinct(edx$userId),
                     movies_number = n_distinct(edx$movieId),
                     average_rating = round(mean(edx$rating),3),
                    genres_number = n_distinct(edx$genres))

edx_subset_summary
```

```
## # A tibble: 1 x 6
##   rows_number columns_number users_number movies_number average_rating genres_~1
##         <int>          <int>        <int>         <int>          <dbl>     <int>
## 1     9000055              6        69878         10677           3.51       797
## # ... with abbreviated variable name 1: genres_number
```

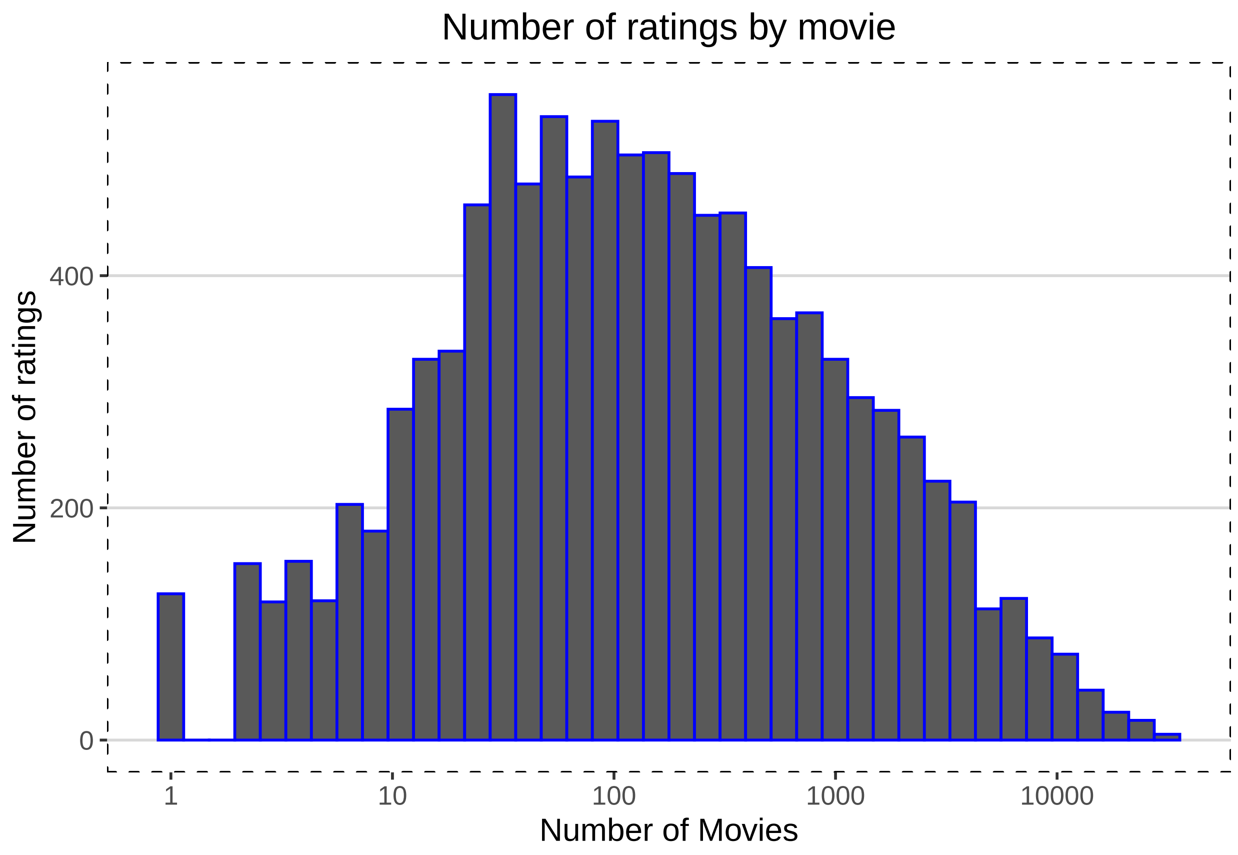
The outcome of the Edx subset is the movie ratings. Thus, we will study the impact of different variables (Movies, Users, Genres) on this outcome.

```

## 3.3.2. Analysis of rating by Movie

edx %>% group_by(movieId) %>%
  summarize(num_movie_rating = n(),
            mu_movies = mean(rating),
            sd_movies = sd(rating)) %>% ggplot(aes(x = num_movie_rating))+
  geom_histogram(bins = 40, color = "blue")+
  theme_hc() +
  scale_x_log10()+
  ggtitle("Number of ratings by movie") +
  labs(x="Number of Movies",
      y="Number of ratings",)+
theme(plot.background = element_rect(colour= NA, linetype = "solid", fill = NA, size = 1), panel.border
```

3.3.2. Analysis of rating by Movie:

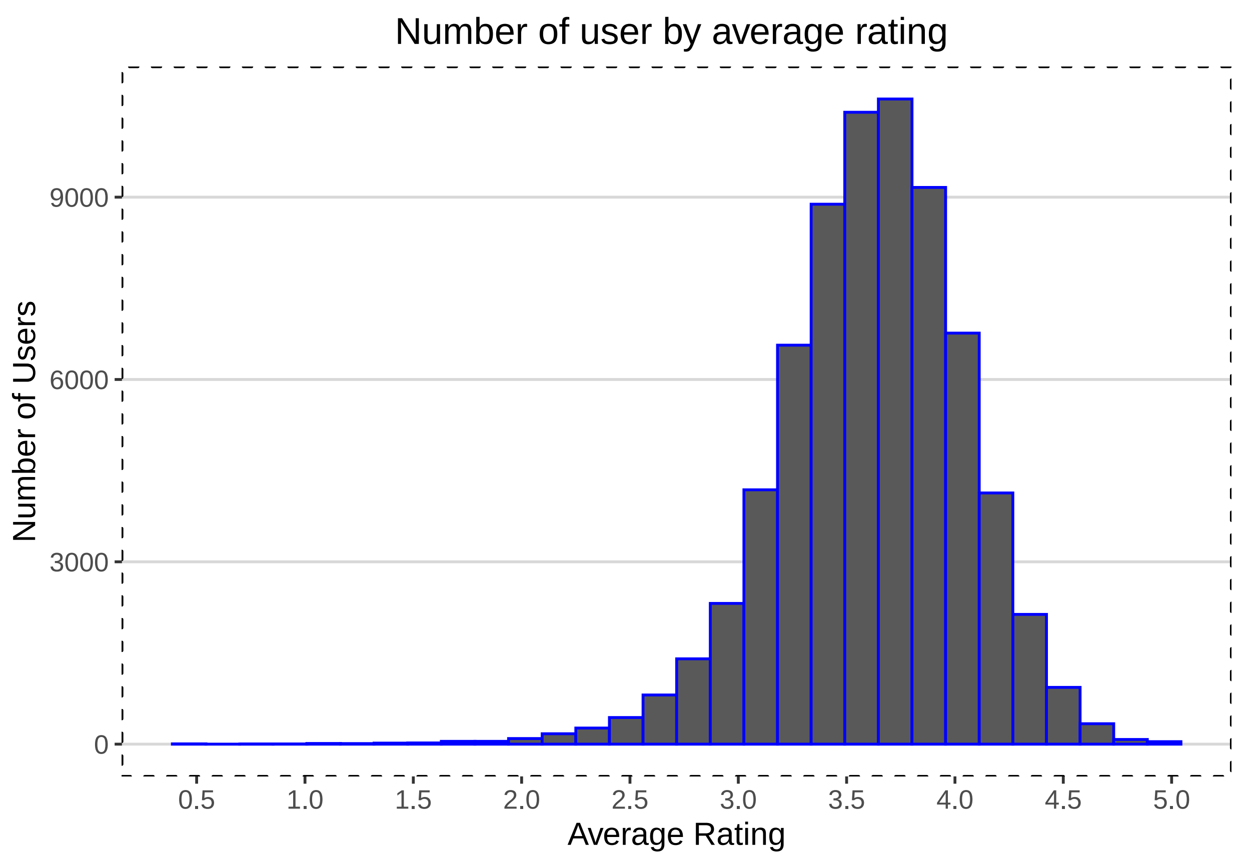


We can conclude that the average movie rating increase when the number of the rating increase

```
## 3.3.3 Rating by User:

edx %>% group_by(userId) %>%
  summarise(user_ave_rating = sum(rating)/n()) %>%
  ggplot(aes(user_ave_rating)) +
  geom_histogram(bins=30, color = I("blue")) +
  theme_hc() +
  scale_x_continuous(breaks=seq(0, 5, by= 0.5)) +
  ggtitle("Number of user by average rating ") +
  labs(x="Average Rating",
       y="Number of Users",
      )+
  theme(plot.background = element_rect(colour= NA, linetype = "solid", fill = NA, size = 1), panel.bord
```

3.3.3. Analysis of rating by User:

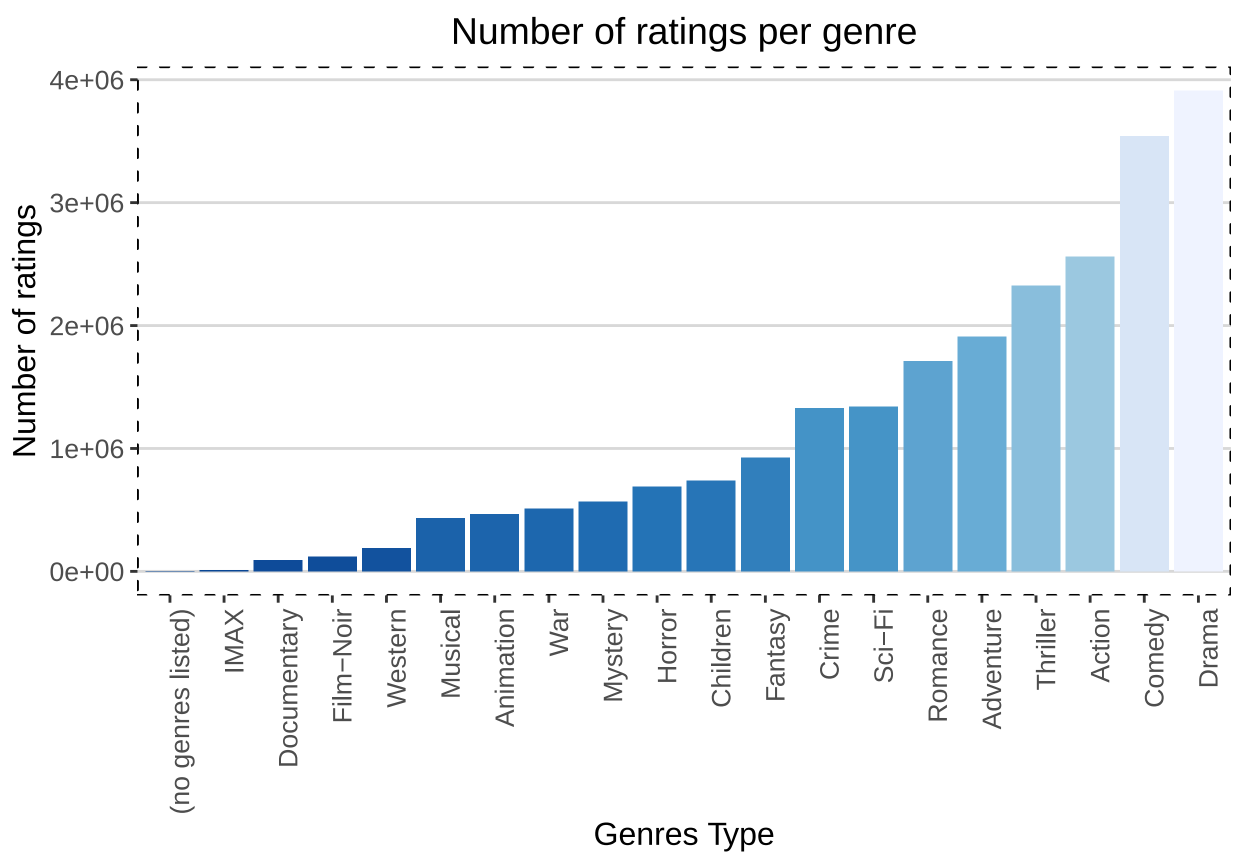


The majority of users rated the movies from 3 to 4 The average user rating tends to increase when the number of rating increases

```
 ## .3.3.4. Analysis of rating by Genres:

genres_summarize <- edx %>% separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarize( num_movie_per_genres = n(), avg_movie_per_genres = mean(rating)) %>%
  arrange(desc(num_movie_per_genres))
genres_summarize <- edx %>% separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarize( num_movie_per_genres = n(), avg_movie_per_genres = mean(rating)) %>%
  arrange(desc(num_movie_per_genres))
genres_summarize %>%
  ggplot(aes(num_movie_per_genres,reorder(genres, num_movie_per_genres),  fill= num_movie_per_genres)) +
  geom_bar(stat = "identity") + coord_flip() +
  scale_fill_distiller()+
  ggtitle("Number of ratings per genre") +
  labs(y = "Genres Type",
       x = "Number of ratings",
       )+
  theme_hc()+
  theme(plot.background = element_rect(colour= NA, linetype = "solid", fill = NA, size = 1), legend.pos
```

3.3.4. Analysis of rating by Genres



We conclude that the movie genre slightly affects the number of ratings.

3.4. Developing modeling concepts:

This project aims to create a recommendation system with an RMSE of less than 0.86490. After a search process, the best way is to split the Edx subset into an Edx_trainset (80%) and Edx_testset (20%). The following models will be created to hit the target above mentioned;

```r
### Split the Edx subset into Edx_test_set and Edx_train_set

set.seed(20, sample.kind="Rounding")
edx_test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
edx_train_set <- edx %>% slice(-edx_test_index)
edx_temp <- edx %>% slice(edx_test_index)
edx_test_set <- edx_temp %>%
  semi_join(edx_train_set, by = "movieId") %>%
  semi_join(edx_train_set, by = "userId")
rows_removed <- anti_join(edx_temp, edx_test_set)
edx_train_set <- rbind(edx_train_set, rows_removed)
```

3.4.1. Average model: The simplest and baseline model is the average rating: which consists of the rating mean calculation

3.4.2. Movie impact model: According to the descriptive analysis of section 3.3, the movie type can affect the ratings. Thus the average rating on that specific movie will have different from the overall average rating of all movies.

3.4.3. Movie and Users impact model: According to the descriptive analysis of section 3.3. A User type could affect the rating of a movie. Thus an adjusted model including the movie type and the user stereotype could enhance the accuracy of the prediction.

3.4.4. Regularization model: In mathematics, statistics, finance, and computer science, particularly in machine learning and inverse problems, regularization is a process that changes the resulting answer to be "simpler". It is often used to obtain results for ill-posed problems or to prevent overfitting. For more information see: Regularization (mathematics) - Wikipedia For that, we will use this method to improve the RMSE.

3.4.5. Matrix Factorization : Matrix factorization is a class of collaborative filtering algorithms used in recommender systems. Matrix factorization algorithms work by decomposing the user-item interaction matrix into the product of two lower-dimensionality rectangular matrices. This family of methods became widely known during the Netflix prize challenge due to its effectiveness as reported by Simon Funk in his 2006 blog post, where he shared his findings with the research community. The prediction results can be improved by assigning different regularization weights to the latent factors based on items' popularity and users' activeness (quoted from Wikipedia) For more information see: Matrix factorization (recommender systems) - Wikipedia

4. Results:

The target RMSE is set to 0.86490

4.1. The Average model Results :

For the Average Model used the results are the following :

```
## The average Model Results:

mu <- mean(edx_train_set$rating)
Average_Model_rmse <- rmse(edx_test_set$rating, mu)
rmse_results <- tibble(Model = "Average Model",
                           RMSE = round(Average_Model_rmse, 4))
rmse_results
```

```
## # A tibble: 1 x 2
##   Model          RMSE
##   <chr>         <dbl>
## 1 Average Model  1.06
```

4.2. The Movie impact Model:

```
## The Movie impact Model:

movie_avgs <- edx_train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
predict_rating_mi <- mu + edx_test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i
Movie_impact_model_rmse <- rmse(edx_test_set$rating,predict_rating_mi)
rmse_results <- bind_rows(rmse_results,tibble(Model = "Movie Impact Model",
                                                   RMSE = round(Movie_impact_model_rmse, 4)))

rmse_results
```

```
## # A tibble: 2 x 2
##   Model                RMSE
##   <chr>               <dbl>
## 1 Average Model      1.06
## 2 Movie Impact Model 0.944
```

The addition of the movie impact into the code improved the accuracy of the algorithm but the RMSE was still above the target

4.3. Movie and Users impact model:

```
## Movie and Users impact model:

user_avgs <- edx_train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
predict_rating_mum <- edx_test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u)
Movie_and_User_rmse <- rmse(edx_test_set$rating,predict_rating_mum$pred)
rmse_results <- bind_rows(rmse_results,
                          tibble(Model="Movie and User Impact model",
                                 RMSE = round(Movie_and_User_rmse, 4)))
rmse_results
```

```
## # A tibble: 3 x 2
##   Model                        RMSE
##   <chr>                       <dbl>
## 1 Average Model               1.06
## 2 Movie Impact Model          0.944
## 3 Movie and User Impact model 0.866
```

After adding the Movie and the user into the code the RMSE improved but was still above the target

4.4. Regularization of the user and the movie impacts:

```
## Regularization of the user and the movie impact:
lambdas <- seq(0, 10, 0.25)
set.seed(21, sample.kind = "Rounding")
rmses <- sapply(lambdas, function(l){

  mu <- mean(edx_train_set$rating)

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

  b_u <- edx_train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  predicted_ratings <- edx_test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred

  return(rmse(edx_test_set$rating,predicted_ratings))
})
lambda <- lambdas[which.min(rmses)]
reg_movie_avgs <- edx_train_set %>%
  group_by(movieId) %>%
  summarize(reg_b_i = sum(rating - mu)/(n()+lambda), n_i = n())
reg_user_avgs <- edx_train_set %>%
  left_join(reg_movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(reg_b_u = sum(rating - mu - reg_b_i)/(n()+lambda), n_u = n())
reg_predicted_ratings <- edx_test_set %>%
  left_join(reg_movie_avgs, by='movieId') %>%
  left_join(reg_user_avgs, by='userId') %>%
  mutate(pred = mu + reg_b_i + reg_b_u) %>%
  .$pred
Regularisation_rmse <- rmse(edx_test_set$rating,reg_predicted_ratings)
rmse_results <- bind_rows(rmse_results,
                          tibble(Model="Regularized Movie and User impacts Model",
                                 RMSE = round(Regularisation_rmse, 4)))
rmse_results
```

```
## # A tibble: 4 x 2
##   Model                                     RMSE
##   <chr>                                    <dbl>
## 1 Average Model                            1.06
## 2 Movie Impact Model                       0.944
## 3 Movie and User Impact model              0.866
## 4 Regularized Movie and User impacts Model 0.865
```

The regularization model improved the results but the RMSE did not hit the target

4.5. Parallel Matrix factorization Model:

This model is built using the recosystem library; for more information visit: CRAN - Package recosystem (r-project.org)

I used the parallel Matrix factorization method based on the residuals of the best model so far (Regularization Movie and User impact model)

```
## Parallel Matrix Factorization Model:

residual_edx <- edx_train_set %>%
  left_join(reg_movie_avgs, by = "movieId") %>%
  left_join(reg_user_avgs, by = "userId") %>%
  mutate(residual = rating - mu - reg_b_i - reg_b_u) %>%
  select(userId, movieId, residual)
residual_mf <- as.matrix(residual_edx)
edx_test_mf <- edx_test_set %>%
  select(userId, movieId, rating)
edx_test_mf <- as.matrix(edx_test_mf)
write.table(residual_mf , file = "trainset.txt" , sep = " " , row.names = FALSE, col.names = FALSE)
```

```r
write.table(edx_test_mf, file = "testset.txt" , sep = " " , row.names = FALSE, col.names = FALSE)
train_set <- data_file("trainset.txt")
test_set <- data_file("testset.txt")
r <-Reco()
opts <- r$tune(train_set, opts = list(dim = c(10, 20, 30), lrate = c(0.1, 0.2),
                                      costp_l1 = 0, costq_l1 = 0,
                                      nthread = 1, niter = 10))
r$train(train_set, opts = c(opts$min, nthread = 1, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.8599   5.5634e+06
##    1       0.8375   5.1786e+06
##    2       0.8205   5.0355e+06
##    3       0.8036   4.9058e+06
##    4       0.7882   4.7892e+06
##    5       0.7743   4.6879e+06
##    6       0.7622   4.6045e+06
##    7       0.7517   4.5321e+06
##    8       0.7426   4.4727e+06
##    9       0.7348   4.4208e+06
##   10       0.7279   4.3768e+06
##   11       0.7219   4.3393e+06
##   12       0.7166   4.3070e+06
##   13       0.7119   4.2774e+06
##   14       0.7076   4.2526e+06
##   15       0.7038   4.2306e+06
##   16       0.7004   4.2097e+06
##   17       0.6973   4.1916e+06
##   18       0.6945   4.1748e+06
##   19       0.6918   4.1598e+06
```

```r
pred_file <- tempfile()
r$predict(test_set, out_file(pred_file))
```

```
## prediction output generated at C:\Users\HATEM~1.RAB\AppData\Local\Temp\RtmpAb0tkk\file38b041a96323
```

```r
predicted_residuals_mf <- scan(pred_file)
predicted_ratings_mf <- reg_predicted_ratings + predicted_residuals_mf
Factorization_rmse <- rmse(edx_test_set$rating, predicted_ratings_mf)
rmse_results <- bind_rows(rmse_results,
                          tibble(Model="Parallel Matrix Factorization",
                                 RMSE = round(Factorization_rmse, 4)))

rmse_results
```

```
## # A tibble: 5 x 2
##   Model                                   RMSE
##   <chr>                                  <dbl>
## 1 Average Model                          1.06
## 2 Movie Impact Model                     0.944
## 3 Movie and User Impact model            0.866
## 4 Regularized Movie and User impacts Model 0.865
## 5 Parallel Matrix Factorization          0.796
```

The Parallel Matrix factorization allowed the RMSE to reach 0.7962 which is beyond the target

4.6. Final model to be used

After testing different models using the Edx _test_subset the best model result was the parallel factorization model. Thus, we will use this method to create our final model and test it using the validation set.

```r
## Final Model used :

set.seed(22, sample.kind = "Rounding")
f_mu <- mean(edx$rating)
f_lambdas <- seq(0, 10, 0.25)

f_rmses <- sapply(f_lambdas, function(l){

  mu <- mean(edx$rating)

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

  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  predicted_ratings <- validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred

  return(rmse(validation$rating,predicted_ratings))
})

f_lambda <- f_lambdas[which.min(f_rmses)]
final_movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(f_b_i = sum(rating - f_mu)/(n()+f_lambda), n_i = n())
final_user_avgs <- edx %>%
  left_join(final_movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(f_b_u = sum(rating - f_mu - f_b_i)/(n()+f_lambda), n_u = n())
final_reg_predicted_ratings <- validation %>%
  left_join(final_movie_avgs, by='movieId') %>%
  left_join(final_user_avgs, by='userId') %>%
  mutate(pred = f_mu + f_b_i + f_b_u) %>%
  .$pred
final_residual_edx <- edx %>%
  left_join(final_movie_avgs, by = "movieId") %>%
  left_join(final_user_avgs, by = "userId") %>%
  mutate(residual = rating - f_mu - f_b_i - f_b_u) %>%
  select(userId, movieId, residual)
final_residual_mf <- as.matrix(final_residual_edx)
validation_mf <- validation %>%
  select(userId, movieId, rating)
validation_mf <- as.matrix(validation_mf)
```

```
write.table(final_residual_mf , file = "final_trainset.txt" , sep = " " , row.names = FALSE, col.names =
write.table(validation_mf, file = "final_testset.txt" , sep = " " , row.names = FALSE, col.names = FALSE
final_train_set <- data_file("final_trainset.txt")
final_test_set <- data_file("final_testset.txt")
f_r <-Reco()
f_opts <- f_r$tune(final_train_set, opts = list(dim = c(10, 20, 30), lrate = c(0.1, 0.2),
                                                costp_l1 = 0, costq_l1 = 0,
                                                nthread = 1, niter = 10))
f_r$train(final_train_set, opts = c(f_opts$min, nthread = 1, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.8587   6.9614e+06
##    1       0.8338   6.4443e+06
##    2       0.8146   6.2511e+06
##    3       0.7973   6.0795e+06
##    4       0.7828   5.9456e+06
##    5       0.7708   5.8349e+06
##    6       0.7611   5.7520e+06
##    7       0.7528   5.6829e+06
##    8       0.7458   5.6246e+06
##    9       0.7396   5.5757e+06
##   10       0.7341   5.5334e+06
##   11       0.7293   5.4977e+06
##   12       0.7249   5.4645e+06
##   13       0.7210   5.4353e+06
##   14       0.7175   5.4096e+06
##   15       0.7144   5.3876e+06
##   16       0.7115   5.3668e+06
##   17       0.7088   5.3476e+06
##   18       0.7064   5.3310e+06
##   19       0.7041   5.3152e+06
```

```
final_pred_file <- tempfile()
f_r$predict(final_test_set, out_file(final_pred_file))
```

```
## prediction output generated at C:\Users\HATEM~1.RAB\AppData\Local\Temp\RtmpAb0tkk\file38b01fb0266
```

```
final_predicted_residuals_mf <- scan(final_pred_file)
final_predicted_ratings_mf <- final_reg_predicted_ratings + final_predicted_residuals_mf
final_rmse <- rmse(validation$rating, final_predicted_ratings_mf)
final_rmse_results <- tibble(Model = "Final model used",
                                  RMSE = round(final_rmse, 4))

final_rmse_results
```

```
## # A tibble: 1 x 2
##   Model             RMSE
##   <chr>            <dbl>
## 1 Final model used 0.787
```

5. Conclusion

This project is created in the process of passing the Harvardx Professional Data Science Certificate. The objective of this project was to create a machine-learning algorithm to predict movie ratings. We tested several models on the MovieLens dataset. The aim was to get an RMSE less than: 0.86490. Using the Parallel Matrix Factorization we succeeded to reach our Goal with an RMSE of 0.7867.