MovieLens Project

Viktoriia Ilina

April 2021

Overview

Through the last decade, systems of personalized content suggestions gained a large spread around the entire web. The systems are used widely by almost every larger digital service starting from Youtube, Twitter, Netflix, or Instagram, to eBay, Amazon, TikTok, Spotify or Microsoft to deliver personalized content recommendations for their users. The systems maintain continuous content or product purchase suggestions or off-line services – from upselling consumer goods, to music, movies, or articles, up to food delivery or nearby restaurants. A number of companies have their entire strategy built around the recommendation systems, and here Netflix is one of the brightest examples. In 2012 at least 75% of Netflix's downloads reportedly came as conversion of their recommender algorithms[1]. How are such numbers even possible?

Netflix utilizes its custom recommendation engine, that provides personalized content suggestions to the users. In 2006 Netflix launched the Netflix Prize challenge, that offered one million dollars to anyone to beat their current recommendation accuracy by at least 10% on RMSE. For the 5.000 assigned competitors, Netflix offered a dataset of 100 million real users' ratings and the testing infrastructure. In 2009 the prize finally found the winner, a team called BellKor's Pragmatic Chaos, who managed to hit the 10,06% accuracy improvement[2]. Since then, BellKor's algorithm has been in use. Over time, along with Netflix turning towards video streaming service, the recommendation system required a corresponding transformation to fit the new specificity. As the company started to expend geographically (and, obviously, demographically), a need for more adjustments to satisfy the wider and more versatile customer pool came out. For that purpose, Netflix established a team of 300 suggestion engine maintainers, developers, and testers worth $150 million (nearly 3% the company's revenue at that point) to adapt the system to the changing environment. In 2016 Netflix introduced their global service based on a unified recommender system that functions upon data collected from around the whole area covered by the service. These days' Netflix recommendation engine is a multi-purpose complex of algorithms driven by machine learning, that create a synergy that forms the current streaming experience. These algorithms include: Personalized Video Ranker (PVR), Top N Video Ranker, Continue Watching Video Ranker (CWR), Video-Video Similarity (Sims)[3].

The aim of this project is to develop and train a recommendation machine learning algorithm to predict a rating given by users to a movie in the dataset. The dataset provided by GroupLens (a research lab in the Department of Computer Science and Engineering at the University of Minnesota, Twin Cities specializing in recommender systems, online communities, mobile and ubiquitous technologies, digital libraries, and local geographic information systems). This data set contains 10000054 ratings and 95580 tags applied to 10681 movies by 71567 users of the online movie recommender service MovieLens[4]. Users were selected at random for inclusion. All users selected had rated at least 20 movies. Unlike previous MovieLens data sets, no demographic information is included. Each user is represented by an id, and no other information is provided[5]. The Residual Mean Square Error (RMSE) will be used to evaluate the accuracy of the algorithm. The target RMSE for this project is lower than 0.86490.This report will present an exploratory analysis of the data, methodology and training the models, results, conclusion and discussion.

[1]https://www.researchgate.net/publication/309600906_Recommender_systems---_beyond_matrix_completion
[2]https://en.wikipedia.org/wiki/Netflix_Prize
[3]https://www.newamerica.org/oti/reports/why-am-i-seeing-this/case-study-netflix/
[4]https://movielens.org/
[5]http://files.grouplens.org/datasets/movielens/ml-10m-README.html

Data

Loading data

To generate our datasets, we use the following code provided by the course page:

```r
##########################################################
# Create edx set, validation set (final hold-out test set)
##########################################################

# Note: this process could take a couple of minutes

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")

library(tidyverse)
library(caret)
library(data.table)

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

dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")

movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(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)
```

The ‘edx’ dataset comprises 9000061 rows and 6 columns. The ‘validation’ dataset has 999993 rows and 6 columns respectively.

Data exploration and visualization

To begin with, let’s take a quick look at the data. As we can see, ‘edx’ dataset contains 6 variables: ‘userId’, ‘movieId’, ‘rating’, ‘timestamp’, ‘title’ and ‘genres’. Each row represents a single rating a unique user gave a particular movie. Some films belong to several genres at once, so it is advisable for further analysis to split the ‘genre’ column data into single genres.

```
##    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
```

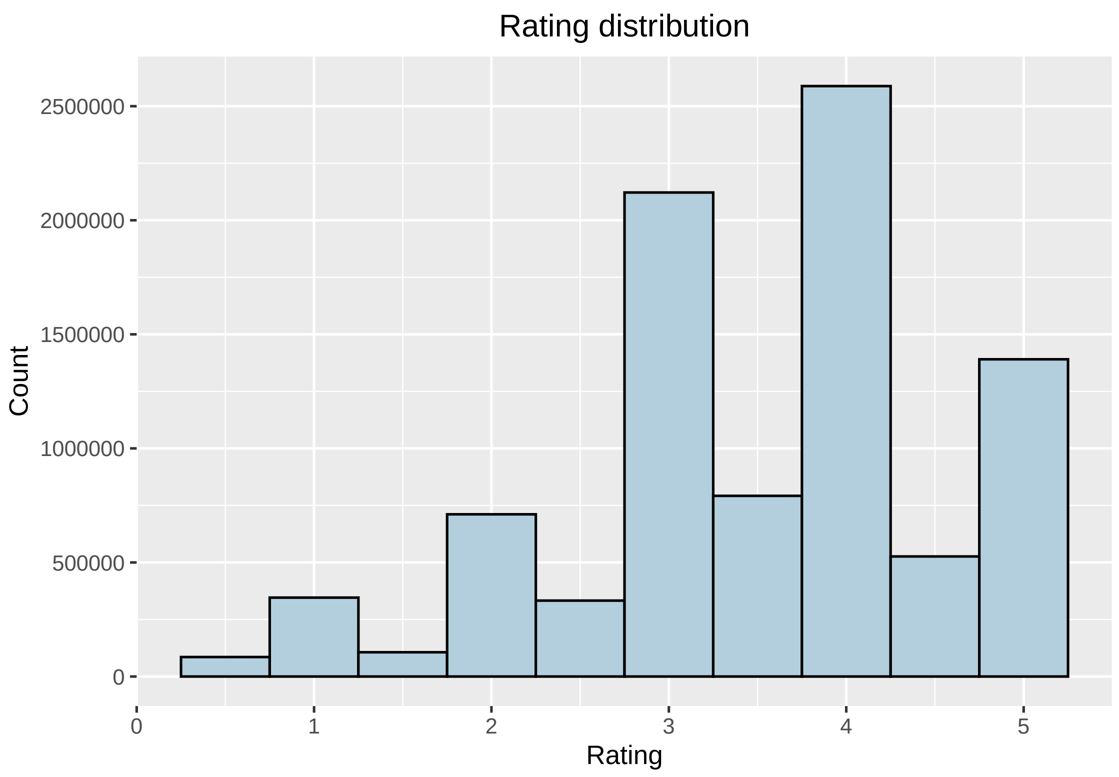
According to the output of summary() function, there are no missing values.

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18122   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35743   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35869   Mean   : 4120   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53602   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000061     Length:9000061
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

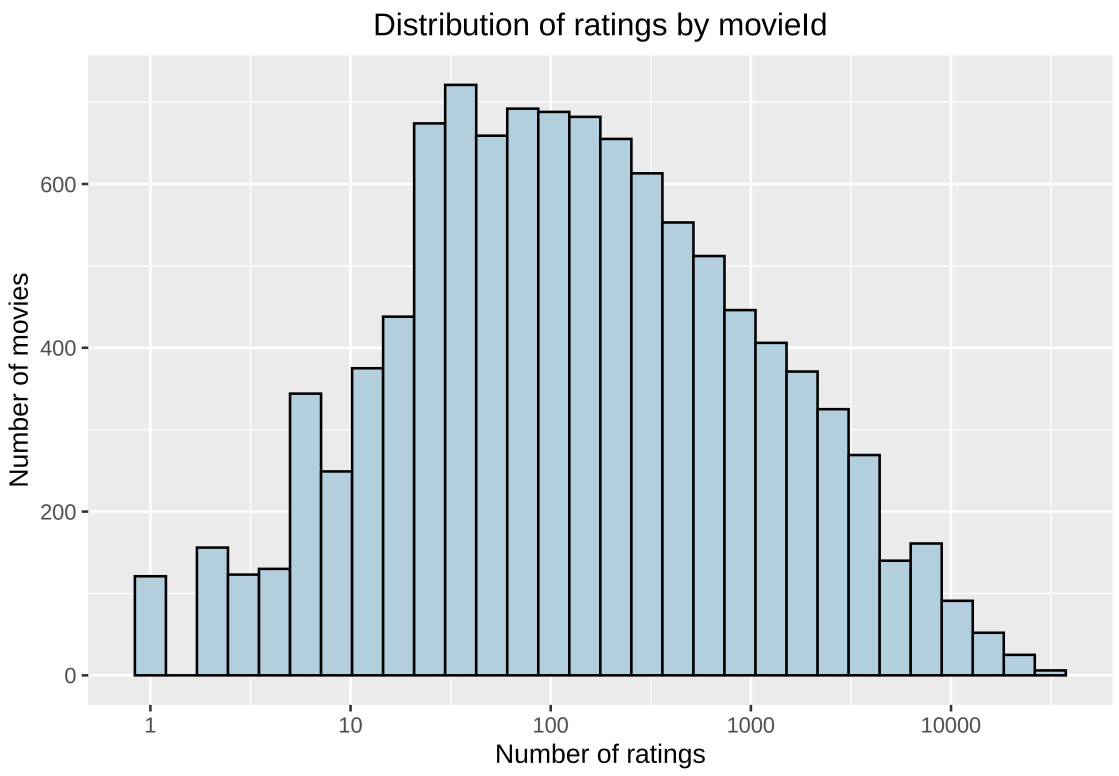
In general, the dataset includes 69878 users and 10677 movies in 797 genres.

```
##   unique_movies unique_users unique_genres
## 1         10677        69878           797
```

In the histogram below, we can see that users tend to rate movies relatively high, since only 25% of ratings are below 3, and use whole numbers.



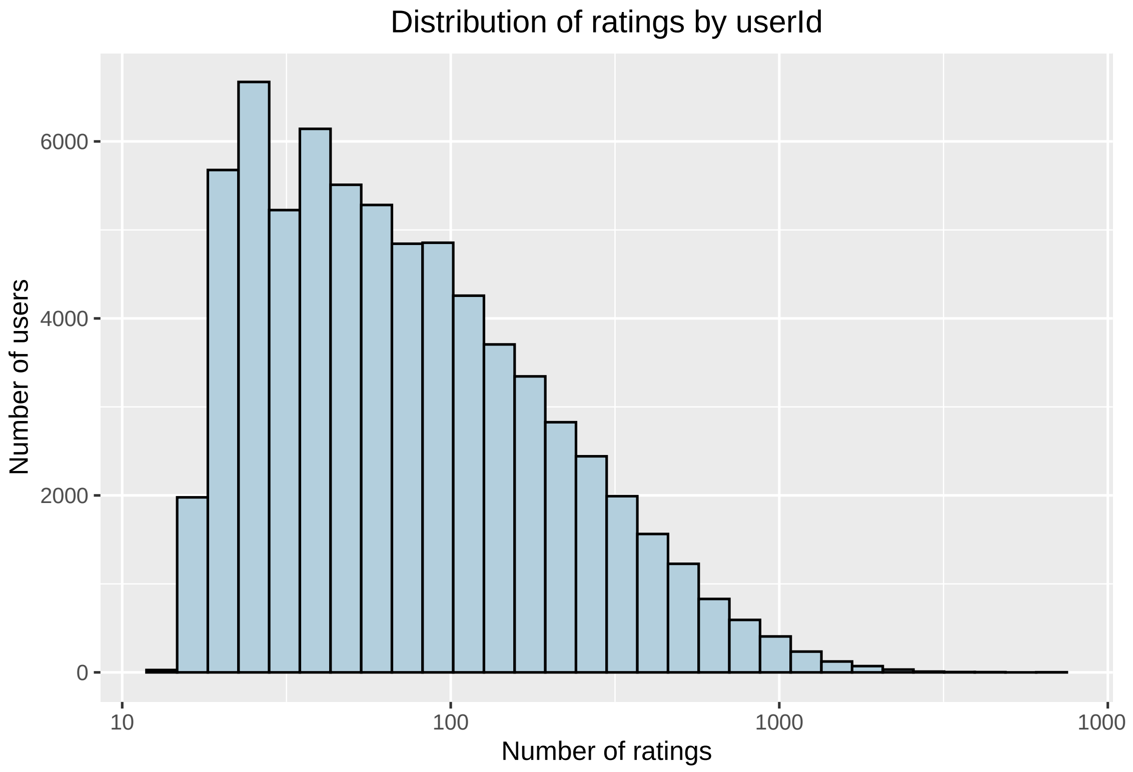
There is a large imbalance in the number of ratings between films: some movies were rated once while other movies were rated more than 10,000 times. As a result, the mean number of rating by movie is 842.9, while the median value is 122. There are a range of reasons for this disjuncture: the year of film's release, genre, star power, cast, awards, marketing and so on.



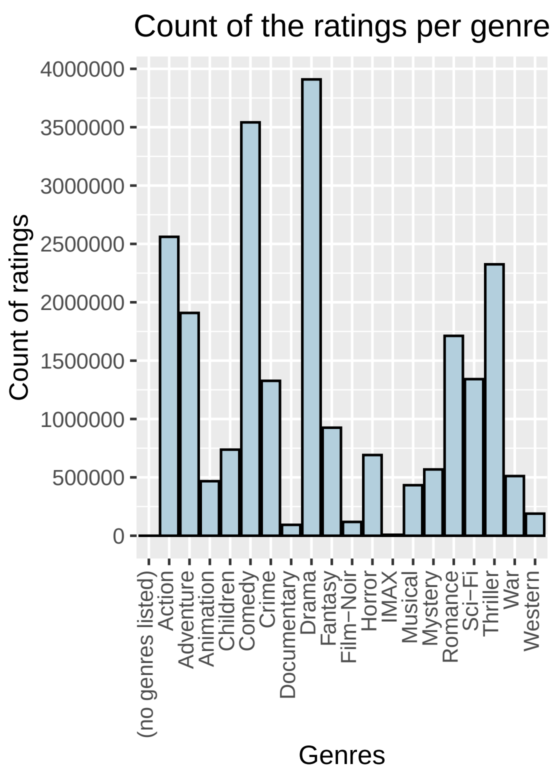
Now, we can see which films have the greatest number of ratings:

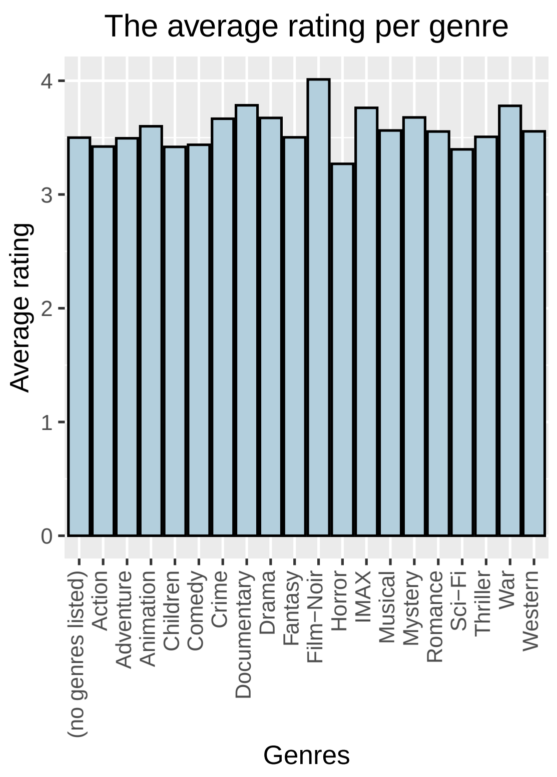
```
## # A tibble: 7 x 3
## # Groups:   movieId [7]
##   movieId title                             count
##     <dbl> <chr>                             <int>
## 1     296 Pulp Fiction (1994)               31336
## 2     356 Forrest Gump (1994)               31076
## 3     593 Silence of the Lambs, The (1991)  30280
## 4     480 Jurassic Park (1993)              29291
## 5     318 Shawshank Redemption, The (1994)  27988
## 6     110 Braveheart (1995)                 26258
## 7     589 Terminator 2: Judgment Day (1991) 26115
```

Number of ratings per user is skewed right. The majority of users rated between 13 and 140 movies, nevertheless the maximum number of ratings given by one user - 6637.



In respect to genres, we can observe that some genres have a lot more ratings than others and that the ratings appear to be different between genres. The most popular rated genre types are Drama and Comedy, whereas Documentary, IMAX, and (no genres listed) have the smallest count of ratings. Drama and film-noir are some of the better-rated genre types, while horror is the worst rated.





Methodology and analysis

Before proceeding to the models and algorithms, we should split the 'edx' dataset into training (90%) and testing (10%) sets.

```r
# Splitting edx data set into training and testing sets

set.seed(1)
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
training <- edx[-test_index,]
temp <- edx[test_index,]

# Checking that all 'userId' and 'movieId' in testing set are also in training set

testing <- temp %>%
  semi_join(training, by = "movieId") %>%
  semi_join(training, by = "userId")

# Adding back missing rows to training set and removing extraneous data

removed <- anti_join(temp, testing)
training <- rbind(training, removed)

rm(test_index, temp, removed)
```

For evaluating trained models for usefulness / accuracy we will use Root Mean Square Error (RMSE), defined by the following function:

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(y_i - \hat{y}_i)^2}{n}}$$

Where:

- $\hat{y}_1$, $\hat{y}_2$, …, $\hat{y}_n$ are predicted ratings;
- y_1, y_2,…, y_n are true ratings;
- n is the number of observations.

or

```
rmse <- function(true_ratings, predicted_ratings){
    sqrt(mean((true_ratings - predicted_ratings)^2))
  }
```

Basic prediction via mean rating

We start with the simplest possible model to have a baseline: suppose we decided to do no statistical work at all and simply predict the same rating for all movies regardless of user or genre. The following baseline prediction model employs the mean of ratings contained in the training dataset, assuming the same rating for all movies and users with all the differences explained by random variation:

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

Where:

- $Y_{u,i}$ is predicted value;
- μ is the mean rating for all movies;
- $\varepsilon_{u,i}$ is a random term explaining the differences between ratings;
- i is movie index;
- u is user index.

```
# Computing the mean

mu <- mean(training$rating)

# Testing results

rmse_naive_model <- rmse(testing$rating, mu)
rmse_naive_model
```

```
## [1] 1.059
```

As expected, our result is far from perfect.

The movie effect model

Data analysis performed in the previous chapter has showed that some movies have higher ratings than others (more highly-rated on average). The following model includes movie effect to attempt to overcome a possible bias related to this phenomenon: the term b_i is added to the formula to represent average ranking for movie i:

$$Y_{u,i} = \mu + b_i + \varepsilon_{u,i}$$

To make a prediction we need to estimate b_i. For this purpose, we might use the lm() function, but owing to the large size of the dataset, this function will be very slow here. Besides, the memory might be overloaded. There are tools in R for such situation, but because we know that the least squares estimate $\hat{b}_i$ is just the average of $Y_{i,u} - \hat{\mu}$ for each movie i, we can compute them manually.

```r
# Computing the averages and predicted ratings

movie_averages <- training %>%
  group_by (movieId) %>%
  summarise(b_i = mean(rating-mu))

predicted_ratings <- mu + testing %>%
  left_join(movie_averages, by = "movieId") %>%
  pull(b_i)

# Testing results

rmse_movie_effect_model <- rmse(testing$rating, predicted_ratings)
rmse_movie_effect_model
```

```
## [1] 0.9426564
```

Taking into account movie effect b_i generates a lower RMSE value; nevertheless, the result is still mediocre.

The movie and user effect model

Besides the movie disjuncture, there is substantial variability across users as well (some users rate movies higher than others). So, adding a user-specific effect b_u to our model can potentially improve the result:

$$Y_{u,i} = \mu + b_i + b_u + \varepsilon_{u,i}$$

```r
# Computing the averages and predicted ratings

user_averages <- training %>%
  left_join(movie_averages, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

predicted_ratings <- testing %>%
  left_join(movie_averages, by = "movieId") %>%
  left_join(user_averages, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

# Testing results

rmse_movie_and_user_effect_model <- rmse(testing$rating, predicted_ratings)
rmse_movie_and_user_effect_model
```

```
## [1] 0.8646047
```

Great, we see an improvement in the RMSE by 8.28%.

The movie, user and genre effect model

As pointed before, the movie ratings vary per genre, so it might be reasonable to include the genre effect to our model. Here We estimate the genre effect b_g as the average of the ratings per genre. We should not forget to split 'genres' column data into single genres in advance.

$$Y_{u,i} = \mu + b_i + b_u + b_g + \varepsilon_{u,i}$$

```r
# Splitting data into single genres

training_genres <- training %>%
  separate_rows(genres, sep = "\\|", convert = TRUE)

testing_genres <- testing %>%
  separate_rows(genres, sep = "\\|", convert = TRUE)

# Computing the averages and predicted ratings

genres_averages <- training_genres %>%
  left_join(movie_averages, by = "movieId") %>%
  left_join(user_averages, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u))

predicted_ratings <- testing_genres %>%
  left_join(movie_averages, by = "movieId") %>%
  left_join(user_averages, by = "userId") %>%
  left_join(genres_averages, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)

# Testing results

rmse_movie_user_and_genre_model <- rmse(testing_genres$rating, predicted_ratings)
rmse_movie_user_and_genre_model
```

```
## [1] 0.8626314
```

We slightly inhence our result, but is it possible to improve it even further?

Regularized movie,user and genre effect model

Now, we introduce the concept of regularization to our model. The general idea behind regularization is to constrain the total variability of the effect sizes. Furthermore, it might reduce possibility of overfitting. Here we use cross-validation to choose a hyperparameter lambda, that controls the regularization strength.

```r
# Note: this process could take a couple of minutes

# Tuning the hyperparameter

lambdas <- seq(0, 10, 0.15)

# Function for computing predicted ratings and testing for each lambda
```

```r
rmses <- sapply(lambdas, function(l){

  mu <- mean(training$rating)

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

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

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

  predicted_ratings <- testing_genres %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_g) %>%
    pull(pred)

  return(rmse(testing_genres$rating, predicted_ratings))

})

# Choosing the minimum of function

rmse_regularization <- min(rmses)
rmse_regularization
```

```
## [1] 0.8620473
```

There is very little improvement compared to the previous model.

Results

As seen in the table below, the best result (RMSE = 0.8620) was achieved by testing regularized movie, user and genre effect model.

Method	RMSE
Basic prediction via mean rating	1.0590002
The movie effect model	0.9426564
The movie and user effect model	0.8646047
The movie, user and genre effect model	0.8626314
Regularized movie,user and genre effect model	0.8620473

However, we should keep in mind that we used ‘edx’ dataset for training, developing and selecting our algorithm.

So, we'll have to go back to the original 'movielens' dataset and try to predict movie ratings in the 'validation' set.

```r
# Note: this process could take a couple of minutes

# Creating the separated genres versions of our datasets

edx_genres <- edx %>%
  separate_rows(genres, sep = "\\|", convert = TRUE)

validation_genres <- validation %>%
  separate_rows(genres, sep = "\\|", convert = TRUE)

# Tuning the hyperparameter

lambdas <- seq(0, 10, 0.15)

# Function for computing predicted ratings and testing for each lambda

rmses <- sapply(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))

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

  predicted_ratings <- validation_genres %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_g) %>%
    pull(pred)

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

# Choosing the minimum of function

rmse_final <- min(rmses)
rmse_final
```

```
## [1] 0.8628443
```

Thus, Root Mean Square Error (RMSE) on 'validation' data is 0.8628443.

Conclusion and discussion

The aim of the project was to develop and train a recommendation machine learning algorithm to predict a rating given by users to a movie in the dataset. To do that, we considered the impact of movies, users and genres to the ratings. The Residual Mean Square Error (RMSE) was used to evaluate the accuracy of the algorithm. The model with the best result is the 'Regularized movie, user and genre effect model' with an RMSE 0.8628443.

While this outcome is promising, future work could likely improve predictive performance in the following ways:

- using cross-validation on the train subset of the data;
- exploring more biases as year, genre/user effect and so on;
- using matrix decomposition;
- using other machine learning algorithms (e.g. XGBoost).

Besides, according to Mhowwala Z., Sulthana A.R. and D.S. Sujala (2020), popularity of directors, actors and writers affect the ratings most[6]. Thus, it might be useful to have additional information about the films.

[6]https://thesai.org/Downloads/Volume11No8/Paper_49-Movie_Rating_Prediction.pdf