Movielens project

Magdalena Sottanellli

2022-10-05

Introduction

Movie recommendation systems use machine learning approach to predict the users' film preferences based on their past choices and impressions, expressed by a given rating. The main scope of the recommendation system is to filter those movies that a corresponding user will most likely want to watch. The aim of this assignment is to create a movie recommendation system based on the Movielens dataset.

About Movielens dataset

To create the movie recommendation system we will be using Movielens dataset. The dataset belongs to GroupLens research lab in the Department of Computer Science and Engineering at the University of Minnesota. They generated their own database with over 20 million ratings for over 27,000 movies by more than 138,000 users. In the task we will be using a subset of this data.

Loading and preparing the data

Code to load and prepare the data was provided by Edx. In order to perform additional analysis some more labriaries were uploaded (ggplot and lubridate). The provided code is as follow:

```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")
```

```
## 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   1.0.9
## v tidyr   1.2.0     v stringr 1.4.1
## v readr   2.1.2     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(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: data.table
##
## Attaching package: 'data.table'
##
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
##
## The following object is masked from 'package:purrr':
##
##     transpose
```

```
library(tidyverse)
library(caret)
library(data.table)
library(ggplot2) #added to perform analysis on plots
library(lubridate) #added to perform analysis on dates
```

```
##
## Attaching package: 'lubridate'
##
## The following objects are masked from 'package:data.table':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year
##
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

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

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"))),
```

```r
                 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(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) # if using R 3.5 or earlier, use `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)
```

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

```r
edx <- rbind(edx, removed)
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

Exploratory Data Analysis

The first step of any analysis is to check the data set and its structure. In the following part we will learn more about Movielens data and the variables that we will analyse one by one.

First we check the structure of the data, the variables and their data type.

```r
str(edx)
```

```
## Classes 'data.table' and 'data.frame':   9000061 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 231 292 316 329 355 356 362 364 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983392 838983421 838983392 838983392 838984474 838983653 8
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Dumb & Dumber (1994)" "Outbreak (1995)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Comedy" "Action|Drama|Sci-Fi|Thriller"
##  - attr(*, ".internal.selfref")=<externalptr>
```

We can see, that our dataset contains 9000061 obs. of 6 variables.

In order to check if our data is tidy, we will print first few rows with the head() function.

```r
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     231      5 838983392          Dumb & Dumber (1994)
## 4:      1     292      5 838983421               Outbreak (1995)
## 5:      1     316      5 838983392               Stargate (1994)
## 6:      1     329      5 838983392 Star Trek: Generations (1994)
##                           genres
## 1:                Comedy|Romance
## 2:         Action|Crime|Thriller
## 3:                        Comedy
## 4:  Action|Drama|Sci-Fi|Thriller
## 5:       Action|Adventure|Sci-Fi
## 6: Action|Adventure|Drama|Sci-Fi
```

From this view we see that data is in tidy format, which means that each row represents one rating given by one user to one movie.

As a next step we will check the number of unique users and unique movies in dataset.

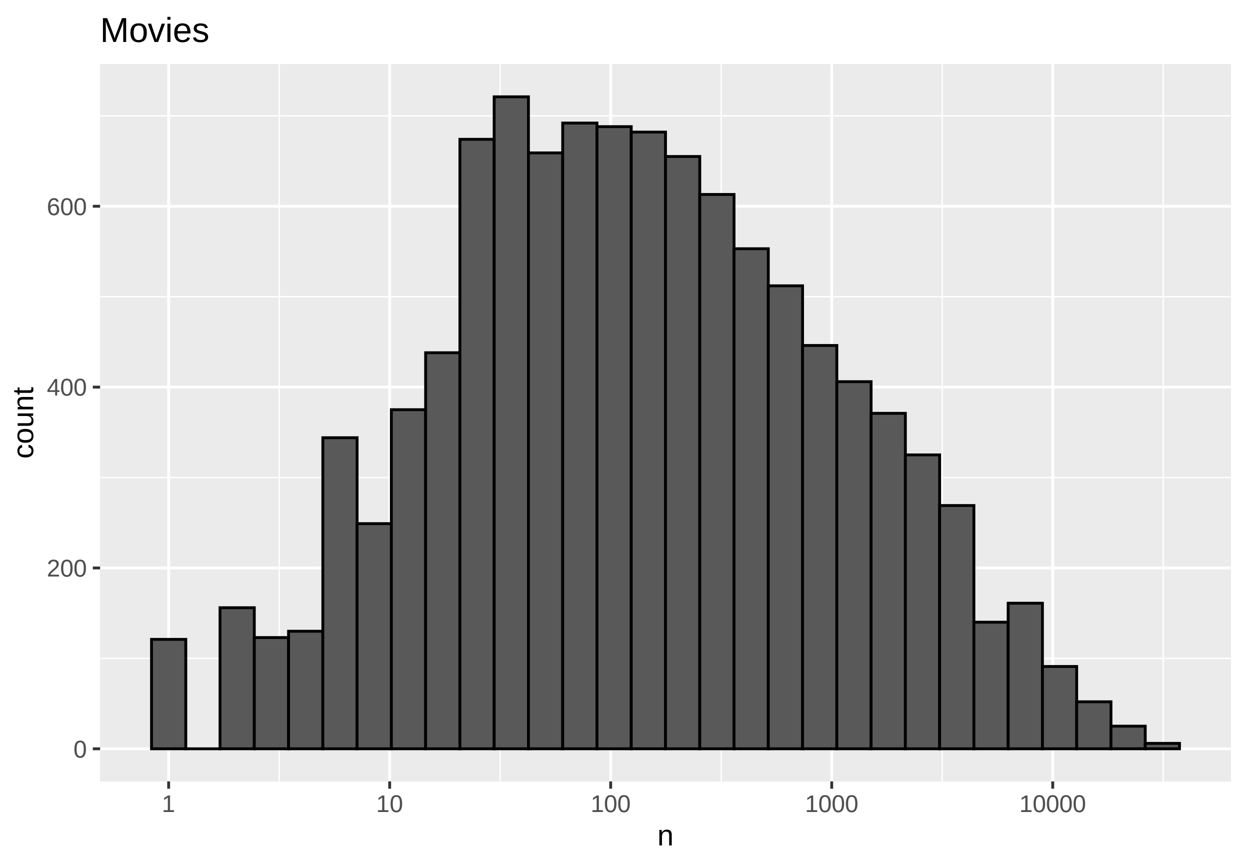
```r
edx %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

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

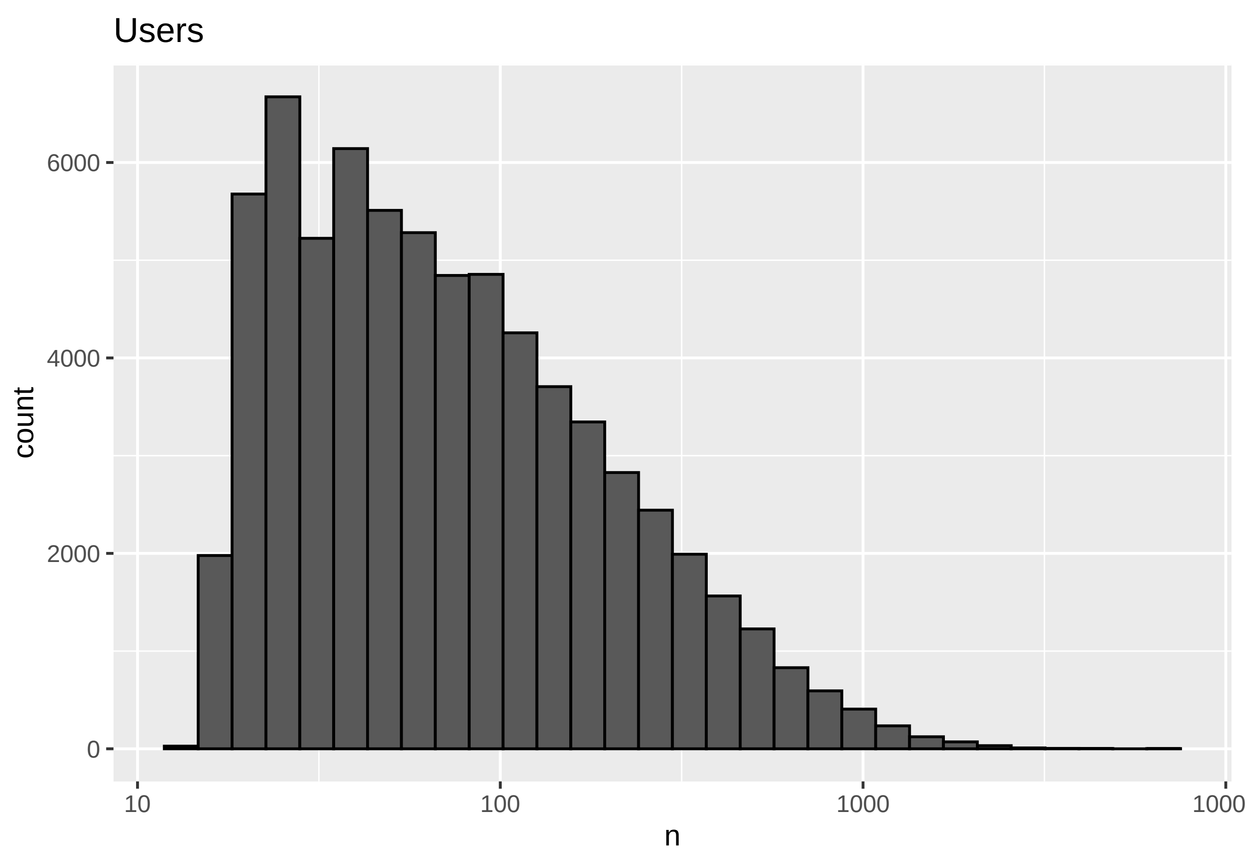
We can see that the number of unique users multiplied by number of unique movies is much higher than the total number of rows in our dataset. That means that each user did not rate each movie. The aim of our task will be to predict the 'missing' ratings, so we can recommend the most accurate movie to the users' preferences.

To better understand the data set we will visualize the distribution of movies and users.

```r
edx %>%
  dplyr::count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Movies")
```



```
edx %>%
  dplyr::count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Users")
```



From first graph we can notice that some movies received more rates than others, which is quite obvious taking into account that database includes records for popular movies as well as for independent movies.

In the graph for users we see that some users are more active in giving ratings than others.

Now we check the genres column. As we can see from head() function this column contains information about genres but in some cases, movies have more than one genres to describe them. To solve this problem we will split the column into single genres:

```r
edx %>% separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarize(count = n()) %>%
  arrange(desc(count))
```

```
## # A tibble: 20 x 2
##    genres               count
##    <chr>                <int>
##  1 Drama              3909401
##  2 Comedy             3541284
##  3 Action             2560649
##  4 Thriller           2325349
##  5 Adventure          1908692
##  6 Romance            1712232
##  7 Sci-Fi             1341750
##  8 Crime              1326917
##  9 Fantasy             925624
## 10 Children            737851
## 11 Horror              691407
## 12 Mystery             567865
## 13 War                 511330
## 14 Animation           467220
## 15 Musical             432960
## 16 Western             189234
## 17 Film-Noir           118394
## 18 Documentary          93252
## 19 IMAX                  8190
## 20 (no genres listed)       6
```

We can see that top 3 genres are Drama, Comedy and Action - they have the highest count of movies related to this categories.

We can also notice the most popular rated movies with the code below:

```
edx %>% group_by(movieId, title) %>%
    summarize(count = n()) %>%
    arrange(desc(count))
```

```
## `summarise()` has grouped output by 'movieId'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 10,677 x 3
## # Groups:   movieId [10,677]
##    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
##  8     457 Fugitive, The (1993)                                         26050
##  9     260 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977) 25809
## 10     592 Batman (1989)                                                24343
## # ... with 10,667 more rows
```

In this category prevail 90s movies, which have received most ratings.

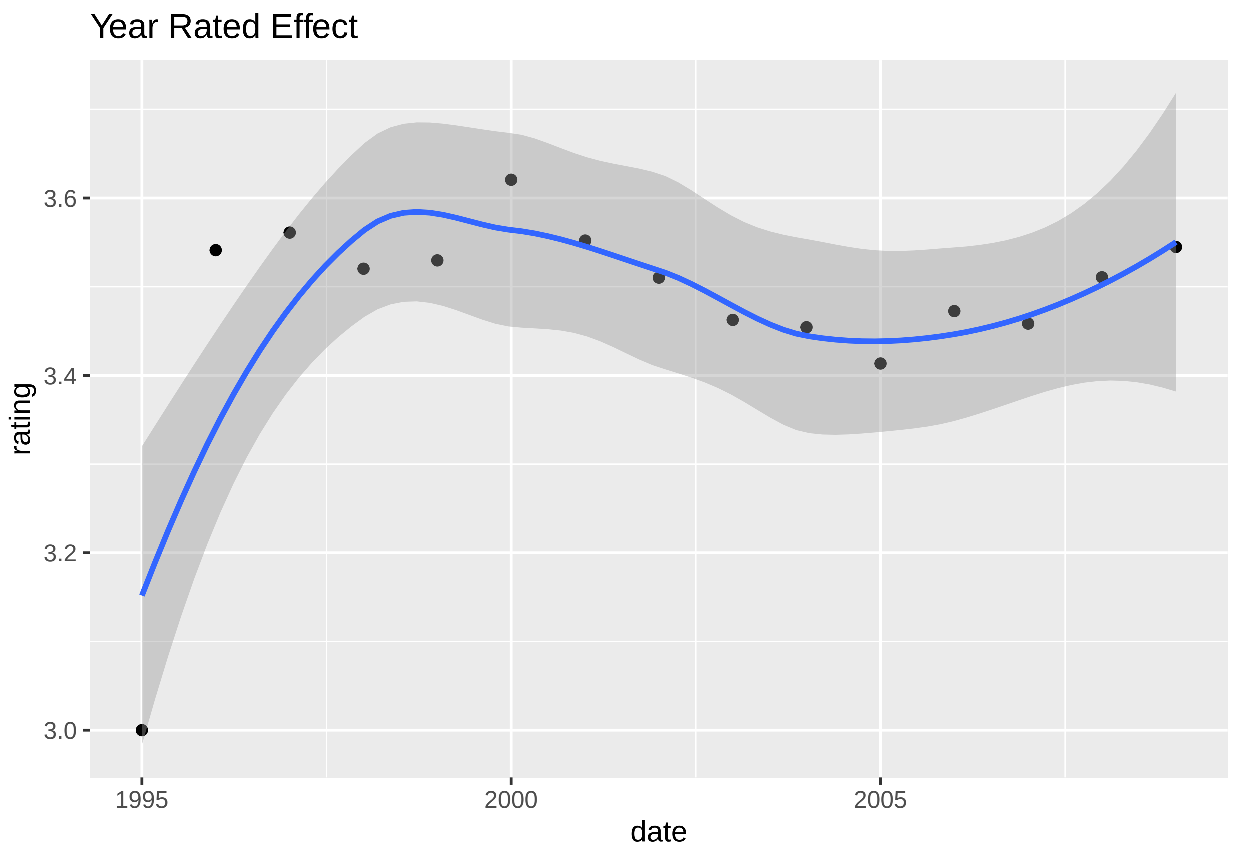
We can also see, how ratings changed over time. The edx dataset includes a timestamp column. This variable represents the time and data in which the rating was provided. For our analysis we will add the column 'date' to edx dataset.

```
edx <- mutate(edx, date = as_datetime(timestamp))
```

Next we will compute the average rating for each year and plot this average against date.

```
edx %>% mutate(date = round_date(date, unit = "year")) %>%
  group_by(date) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(date, rating)) +
  geom_point() +
  geom_smooth()+
  ggtitle("Year Rated Effect")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



We can see that there is some evidence of a time effect in the plot - movies that were rated in the second part of 90s used to receive on avarage higher rates. In 2000s this effect is weaker, as the line on the plot get smoother.

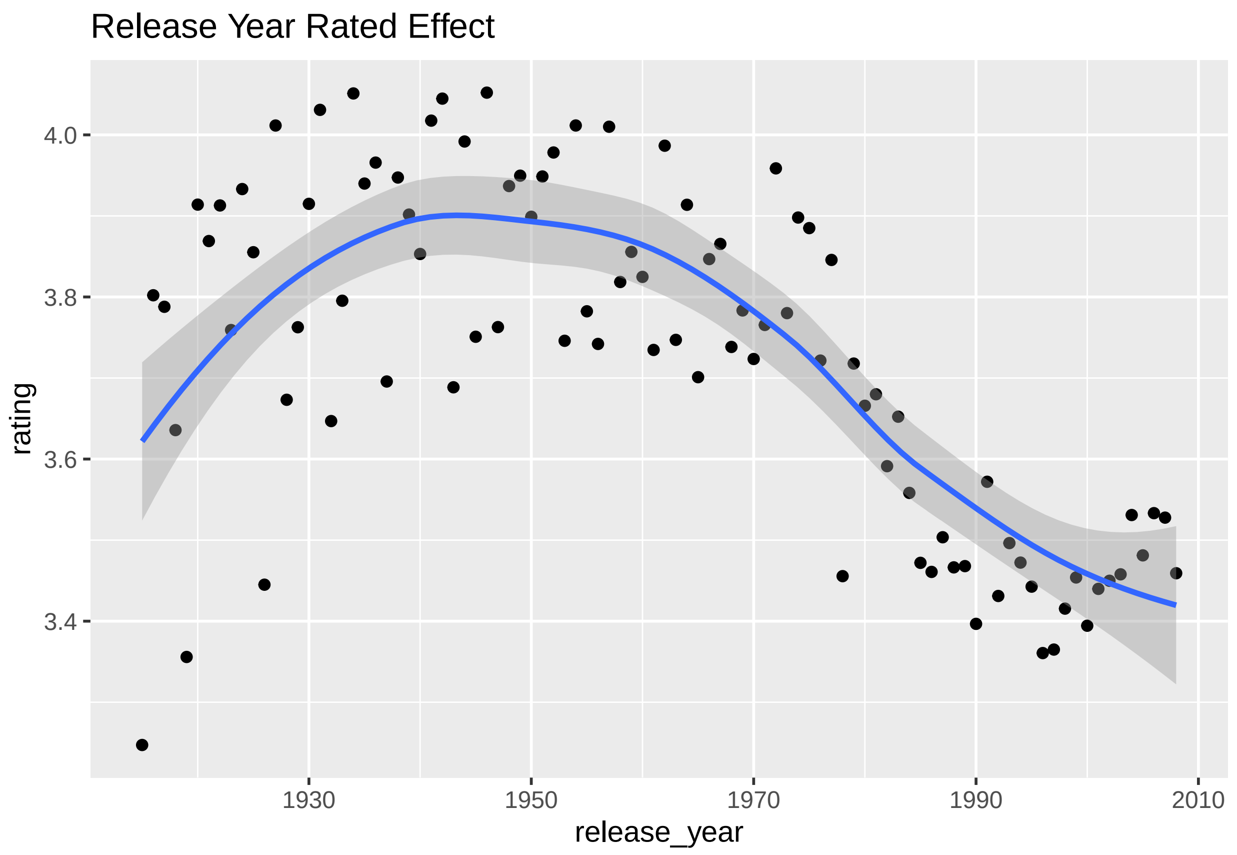
Lastly we will check if there is some connection between released year and the average rating. To obtain information about release year, we need to withdraw it from the movie title.

```
edx <- edx %>%
  mutate( release_year = as.numeric(substr(title,
                                           nchar(title)-4,
                                           nchar(title)-1)))
```

Next we will plot theavarage rating against the release year.

```
edx %>%
  group_by(release_year) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(release_year, rating)) +
  geom_point() +
  geom_smooth() +
  ggtitle("Release Year Rated Effect")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



From the plot we can see that older movies (before 1970) use to have a higher average rating. This can also be related to the fact, that older movies used to have more ratings, hence there was 'more time' to rate them.

Creating and valuating different movie rating predictions

In order to valuate a particular approach, we will be using residual mean squared error (RMSE) on a test set and later at the final step on validation set. In other words, RMSE will be the measurement of the accuracy of predicted models. The scope will be creating the model with the lowest RMSE value.

The formula to calculate RMSE is as follow:

```
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2, na.rm=TRUE))
}
```

Dividing edx dataset into test and training set

Hence we will use validation data set only for final measurement of accuracy, we need to divide edx data set into training and test set to valuate particular models.

```
set.seed(1)
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]
```

Model 1: The simplest approach

In this approach we assume the same rating for all movies regardless of user. We use below equation to predict the ratings:

Y= mu + e

where e represents independent errors sampled from the same distribution centered at 0 and mu is the rating for all movies. We know that the estimate that minimizes the RMSE is the least squares estimate of mu and, in this case, is the average of all ratings.

```
mu_hat <- mean(train_set$rating)
mu_hat
```

```
## [1] 3.51238
```

If we predict all unknown ratings with mu (hat), we get the following RMSE, which we will call naive RMSE:

```
naive_rmse <- RMSE(test_set$rating, mu_hat)
naive_rmse
```

```
## [1] 1.059648
```

We get RMSE above 1, so it is not the best model.

We will store the result in common table rmse_all:

```
rmse_all<- data_frame(Method = "Just the average", RMSE = naive_rmse)
```

```
## Warning: ‘data_frame()‘ was deprecated in tibble 1.1.0.
## Please use ‘tibble()‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was generated.
```

```
rmse_all%>% knitr::kable()
```

Method	RMSE
Just the average	1.059648

Model 2: Modeling movie effect

As we could see previously in exploratory data analysis, not all movies are receving the same amount of ratings. To include the movie effect in model we add additional parameter in the previous equation, which will represent average rating of particular movie.

Y= mu + e + b_i

we know that the least squares estimate of b_i is just the average of
Y - mu for each movie, so we can compute it this way:

```
mu <- mean(train_set$rating)
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

We predict the ratings:

```
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
```

We calculate RMSE for the model and we add it to the table with results:

```
rmse_m <- RMSE(predicted_ratings, test_set$rating)
rmse_m
```

```
## [1] 0.9431724
```

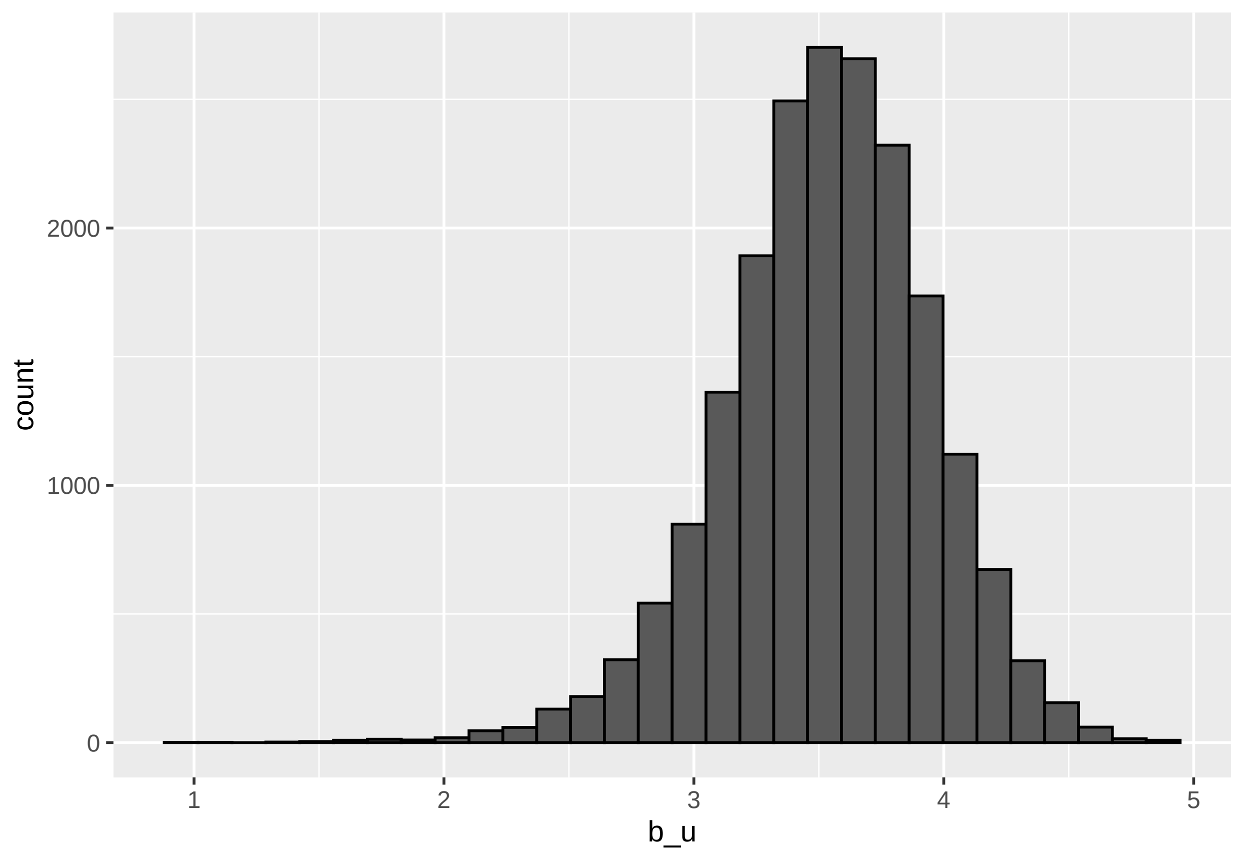
```
rmse_all <- bind_rows(rmse_all,
                      data_frame(Method="Movie Effect Model",
                                 RMSE = rmse_m))
rmse_all %>% knitr::kable()
```

Method	RMSE
Just the average	1.0596481
Movie Effect Model	0.9431724

Model 3: Modeling movie and user effect

First we will plot the average rating for user u for those that have rated 100 or more movies:

```
train_set %>%
  group_by(userId) %>%
  filter(n()>=100) %>%
  summarize(b_u = mean(rating)) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black")
```



We can see a great variability across users as well. This led us to adjust the previous model, including user effect (b_u).

Y= mu + e + b_i + b_u

```r
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
```

We can now construct predictors:

```r
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
```

We calculate RMSE and add the result to the table to compare with previuos models.

```r
rmse_u <- RMSE(predicted_ratings, test_set$rating)
rmse_u
```

```
## [1] 0.8655154
```

```
rmse_all <- bind_rows(rmse_all,
                      data_frame(Method="Movie and User Effect Model",
                                 RMSE = rmse_u))
rmse_all %>% knitr::kable()
```

Method	RMSE
Just the average	1.0596481
Movie Effect Model	0.9431724
Movie and User Effect Model	0.8655154

Obtained RMSE value is lower than for two previous models. We will try to add additional variables like release year and genre to see if we can improve our results.

Model 4: Modeling movie, user and release year effect

As we could see in the exploratory data analysis part, relase year has an impact on the average rating.

This led us to adjust the previous model, including release year effect (b_ry).

Y= mu + e + b_i + b_u + b_ry

```
ry_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  group_by(release_year) %>%
  summarise(b_ry = mean(rating - mu - b_i - b_u, na.rm=TRUE))
```

We can now construct predictors:

```
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(ry_avgs, by='release_year') %>%
  mutate(pred = mu + b_i + b_u + b_ry) %>%
  pull(pred)
```

We calculate RMSE and add the result to the table to compare it with previous models.

```
rmse_ry <- RMSE(predicted_ratings, test_set$rating)
rmse_ry
```

```
## [1] 0.8651955
```

```
rmse_all <- bind_rows(rmse_all,
                      data_frame(Method="Movie, User and Realease Year Effect Model",
                                 RMSE = rmse_ry))
rmse_all %>% knitr::kable()
```

Method	RMSE
Just the average	1.0596481
Movie Effect Model	0.9431724
Movie and User Effect Model	0.8655154
Movie, User and Realease Year Effect Model	0.8651955

As we can see, adding the release year effect to the model decreased RMSE slightly compared to movie and user effect model. In next step we will see, if genre has also impact on the model.

Model 5: Modeling movie, user, release year and genre effect

As we could see in the data exploratory analysis part, some genres are more popular than others. We will now check how the genre effect (b_g) will influence the final RMSE value.

Y= mu + e + b_i + b_u + b_ry + b_g

```r
genres_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(ry_avgs, by = 'release_year') %>%
  group_by(genres) %>%
  summarise(b_g = mean(rating - mu - b_i - b_u - b_ry, na.rm=TRUE))
```

We can now construct predictors:

```r
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(ry_avgs, by='release_year') %>%
  left_join(genres_avgs, by='genres') %>%
  mutate(pred = mu + b_i + b_u + b_ry + b_g) %>%
  pull(pred)
```

We calculate RMSE and add the result to the table to compare it with previous models.

```r
rmse_g <- RMSE(predicted_ratings, test_set$rating)
rmse_g
```

```
## [1] 0.8649508
```

```r
rmse_all <- bind_rows(rmse_all,
                      data_frame(Method="Movie, User, Realease Year and Genres Effect Model",
                                 RMSE = rmse_g))
rmse_all %>% knitr::kable()
```

Method	RMSE
Just the average	1.0596481
Movie Effect Model	0.9431724

Method	RMSE
Movie and User Effect Model	0.8655154
Movie, User and Realease Year Effect Model	0.8651955
Movie, User, Realease Year and Genres Effect Model	0.8649508

As we can see, this model has the lowest RSME, so we will check if we can decrease this value even more through regularization.

Model 6: Regularization of movie, user, release year and genre effect model

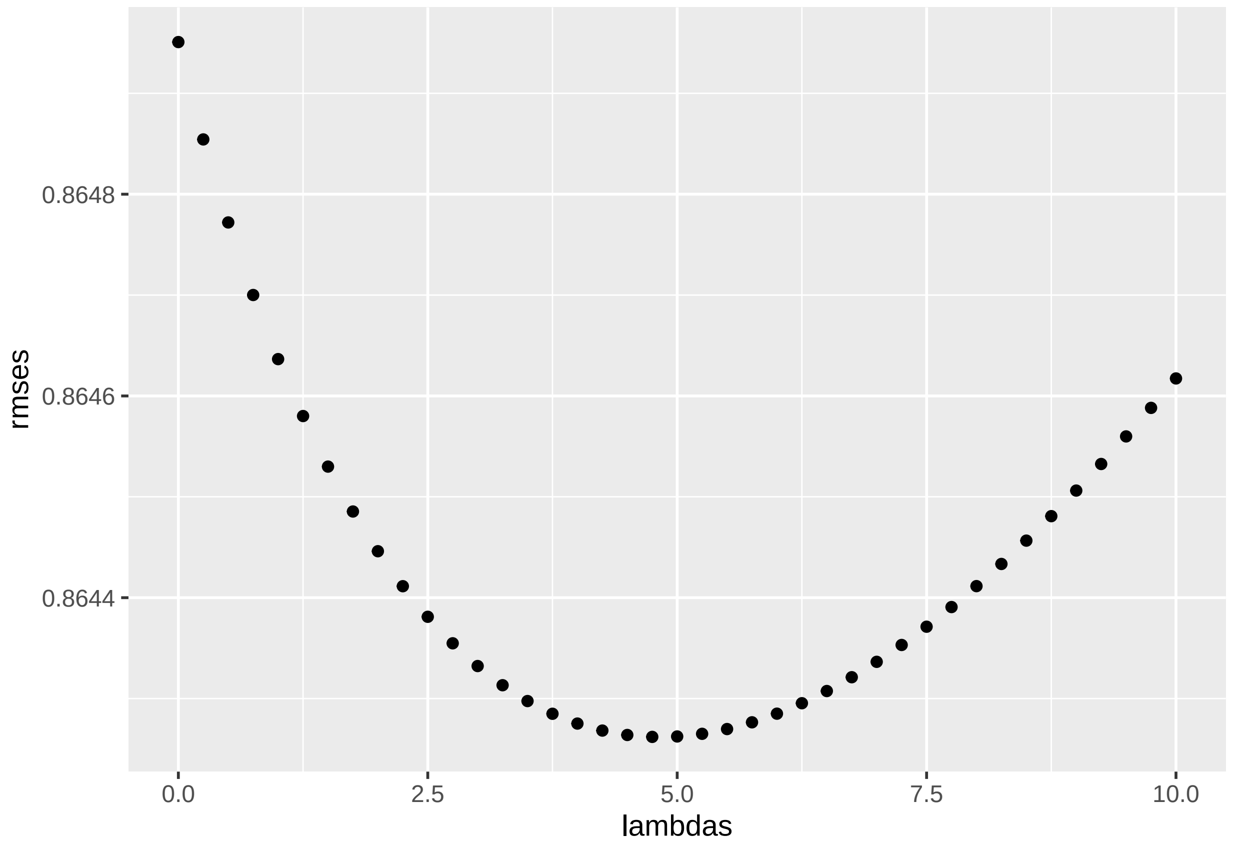
The idea behind regularization is to constrain the total variability of the effect sizes - in our case the movie, user, release year and genres effect.

As lambda is a tuning parameter and we will select its value using cross-validation.

```
lambdas <- seq(0, 10, 0.25)


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

  mu <- mean(train_set$rating)

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

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

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

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

  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_ry, by = "release_year") %>%
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_ry + b_g) %>%
    pull(pred)
  return(RMSE(predicted_ratings, test_set$rating))
})
```

We plot the lambdas to see the optimal value:

```
qplot(lambdas, rmses)
```



```
lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 4.75
```

We calculate RMSE and add the result to the table to compare it with previuos models.

```
rmse_r <- min(rmses)
rmse_r
```

```
## [1] 0.864262
```

```
rmse_all <- bind_rows(rmse_all,
                      data_frame(Method="Regularization of movie, user, release year and genre effects 
                                 RMSE = rmse_r))
rmse_all %>% knitr::kable()
```

Method	RMSE
Just the average	1.0596481
Movie Effect Model	0.9431724
Movie and User Effect Model	0.8655154
Movie, User and Realease Year Effect Model	0.8651955
Movie, User, Realease Year and Genres Effect Model	0.8649508
Regularization of movie, user, release year and genre effects model	0.8642620

Conclusion

Among all the presented methods the lowest RMSE has the regularization of movie, user, release year and genre effect model.

```
rmse_all
```

```
## # A tibble: 6 x 2
##   Method                                                         RMSE
##   <chr>                                                         <dbl>
## 1 Just the average                                              1.06
## 2 Movie Effect Model                                            0.943
## 3 Movie and User Effect Model                                   0.866
## 4 Movie, User and Realease Year Effect Model                    0.865
## 5 Movie, User, Realease Year and Genres Effect Model            0.865
## 6 Regularization of movie, user, release year and genre effects model 0.864
```

This method will be used with validation dataset.

First we need to add column 'release_year' to validation data set.

```
validation <- validation %>%
  mutate( release_year = as.numeric(substr(title,
                                          nchar(title)-4,
                                          nchar(title)-1)))
```

```
lambdas <- seq(0, 10, 0.25)

rmses_v <- sapply(lambdas, function(l){

  mu <- mean(train_set$rating)

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

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

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

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

  predicted_ratings_v <-
    validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_ry, by = "release_year") %>%
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_ry + b_g) %>%
    pull(pred)

  return(RMSE(predicted_ratings_v, validation$rating))
})
```

The final RMSE for validation set is:

```
rmse_v <- min(rmses_v)
rmse_v
```

```
## [1] 0.865362
```

The objective of this Movielens project was to create a well performed rating recomendation system through machine learning training. The goal of this project was to train an algorithm with RMSE less than 0.86490. The final achieved RMSE on validation set is 0.865362.