HarvardX: PH125.9x Data Science Capstone Project

(MovieLens)

Gökhan USTA

January 05, 2021

İçindekiler

1. Introduction

This report is prepared for HarvardX: PH125.9x Data Science Capstone Project. In this project, creating a movie recommendation system using the MovieLens dataset is aimed.

Recommendation systems are one of the most used models in machine learning algorithms. Generally, recommendation systems have the ability to predict whether a particular user would prefer an item or not based on the user's profile. They have also proved to improve decision making process and quality. In scientific libraries, they support users by allowing them to move beyond catalog searches. Therefore, the need to use efficient and accurate recommendation techniques within a system that will provide relevant and dependable recommendations for users cannot be over-emphasized.

Also, recommendation systems use ratings that users have given to items to make specific recommendations. Companies that sell many products to many customers and permit these customers to rate their products, like Amazon or Netflix, are able to collect massive datasets that can be used to predict what rating a particular user will give to a specific item. Items for which a high rating is predicted for a given user are then recommended to that user.

As in our case, these techniques can be used in movies. For this project we will focus on create a movie recommendation system using the MovieLens dataset.

2. Overview

In the project, a machine learning algorithm that predicts user ratings (from 0.5 to 5) is developed using the dataset provided by Edx to predict movie ratings in a provided validation set. RMSE (Root Mean Square Error) will be used to evaluate how close predictions are to the true values in the validation set (the final hold-out test set).

An exploratory data analysis is carried out in order to develop a machine learning algorithm that could predict movie ratings until a final model. Results will be explained.

Root Mean Square Error (RMSE) is a standard way to measure the error of a model in predicting quantitative data. In other words, RMSE is a frequently used measure of the differences between values (sample or population values) predicted by a model or an estimator and the values observed.

Formally it is defined as follows:

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

In data science, RMSE has a double purpose:

- To serve as a heuristic for training models
- To evaluate trained models for usefulness / accuracy

RMSE is a measure of accuracy, to compare forecasting errors of different models for a particular dataset, a lower RMSE is better than a higher one. The effect of each error on RMSE is proportional to the size of the squared error, thus larger errors have a disproportionately large effect on RMSE. As a result, RMSE is sensitive to outliers.

Four models will be compared using their resulting RMSE in order to assess their quality. The evaluation criteria for this algorithm is a RMSE expected to be lower than 0.86490. Finally, the best resulting model will be used to predict the movie ratings.

3. Dataset

The following code used to generate the datasets.

```
##########################################################
# 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")

# if using R 3.6 or earlier:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))
# if using R 4.0 or later:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))


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

The MovieLens dataset is splitted into 2 subsets that will be the “edx”, a training subset to train the algorithm, and “validation” a subset to test the movie ratings, in order to predict in the most possible accurate way the rating of the users.

```
# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
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)
```

Algorithm development is to be carried out on the “edx” subset only, as “validation” subset will be used to test the final algorithm.

4. Methods, Analysis and Visualizations

In order to figure out the dataset, you can find the first rows of “edx” and “validation” subsets below. The subsets contain six variables, “userID”, “movieID”, “rating”, “timestamp”, “title”, and “genres”. Each row represents a single rating of a user for a single movie.

“edx” subset ;

```
##    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
```

“validation” subset ;

```
##    userId movieId rating timestamp
## 1:      1     231      5 838983392
## 2:      1     480      5 838983653
## 3:      1     586      5 838984068
## 4:      2     151      3 868246450
## 5:      2     858      2 868245645
## 6:      2    1544      3 868245920
##                                                       title
## 1:                                       Dumb & Dumber (1994)
## 2:                                       Jurassic Park (1993)
## 3:                                          Home Alone (1990)
## 4:                                             Rob Roy (1995)
## 5:                                      Godfather, The (1972)
## 6: Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
##                                        genres
## 1:                                     Comedy
## 2:        Action|Adventure|Sci-Fi|Thriller
## 3:                            Children|Comedy
## 4:                    Action|Drama|Romance|War
## 5:                                Crime|Drama
## 6: Action|Adventure|Horror|Sci-Fi|Thriller
```

The **summary** of the subsets confirms that there are no missing values.

Summary of the “edx” subset ;

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

Summary of the “edx” subset ;

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18096   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.467e+08
##  Median :35768   Median : 1827   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4108   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53621   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:999999      Length:999999
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
```

The total number of unique movies and users in the “edx” subset is 69.898 unique users and 10.677 different movies:

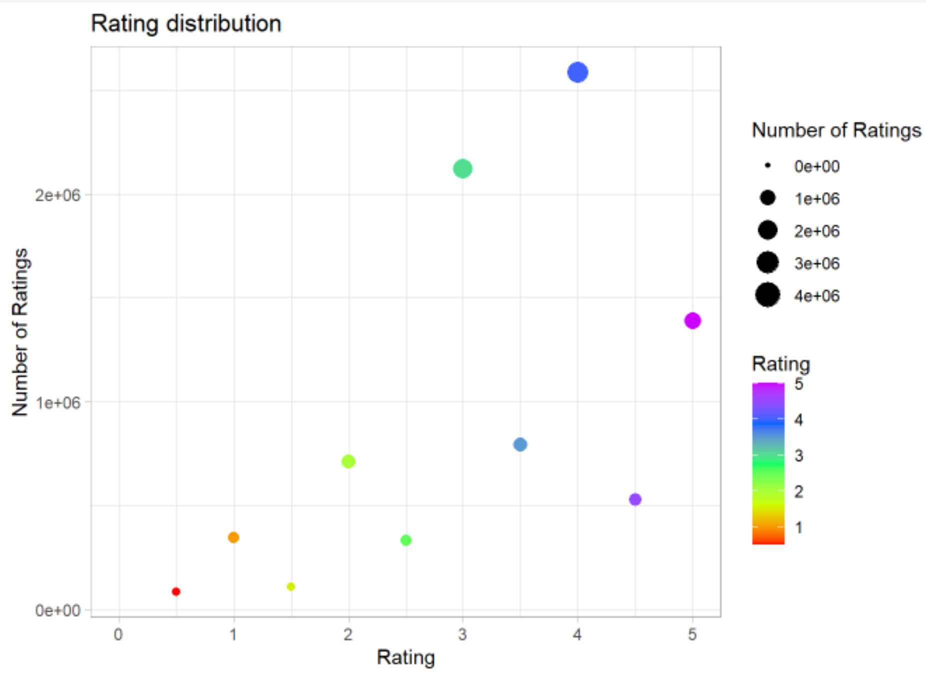
```
##   n_users n_movies
## 1   69878    10677
```

Rating Distributions:

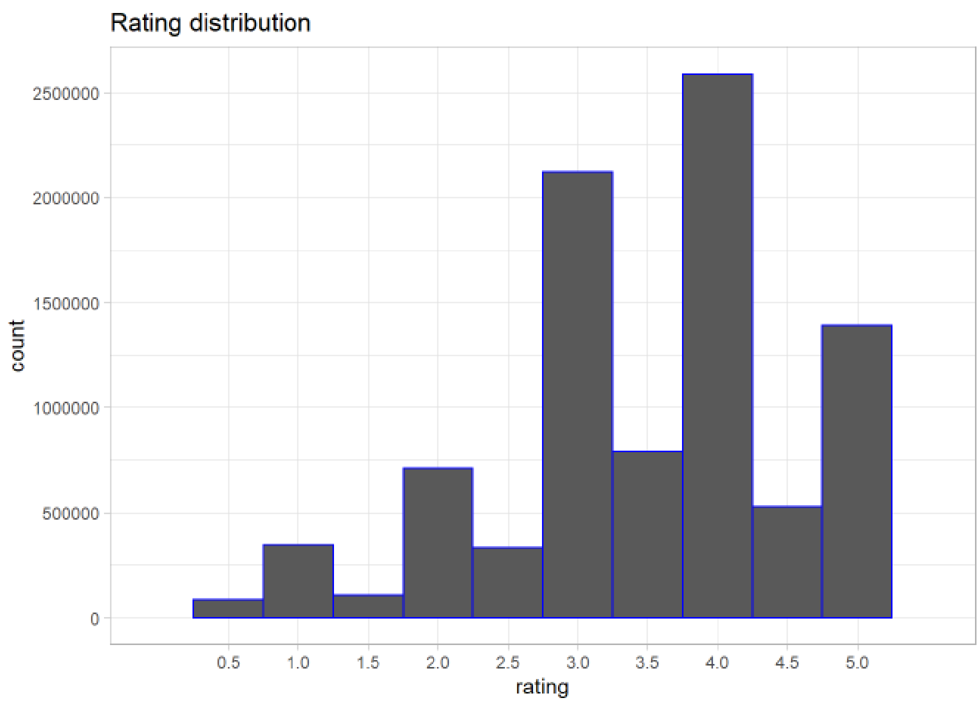
When we explore the distributions of the ratings, it is seen that users prefer to rate movies rather higher than lower as shown by the graphics below. 4 is the most common rating, followed by 3 and 5. 0.5 is the least common rating. In general, half ratings are less common than the whole star ratings.

```
## # A tibble: 10 x 2
##     rating num_ratings
##      <dbl>       <int>
##  1     4       2588430
##  2     3       2121240
##  3     5       1390114
##  4     3.5      791624
##  5     2        711422
##  6     4.5      526736
##  7     1        345679
##  8     2.5      333010
##  9     1.5      106426
## 10     0.5       85374
```

```
edx_ratings %>% # for each rating, plot frequency
  ggplot(aes(rating, num_ratings, color = rating)) +
  geom_point(aes(size = num_ratings)) +
  scale_color_gradientn(colours = rainbow(5)) +
  scale_size_continuous(limits = c(0, 4e+06)) +
  xlim(0,5) +
  labs(x = "Rating", y = "Number of Ratings", title = "Rating distribution", color = "Rating", size = "Number of
 Ratings") +
  theme_light()
```

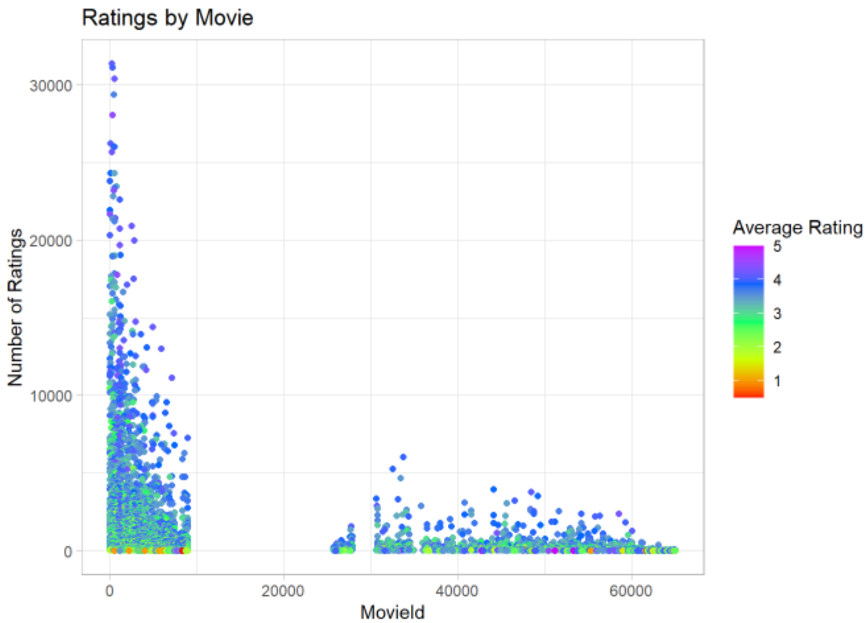


```r
edx %>% # for each rating, plot frequency
  ggplot(aes(rating)) +
  geom_histogram(binwidth = 0.50, color = "blue") +
  scale_x_discrete(limits = c(seq(0.5,5,0.5))) +
  scale_y_continuous(breaks = c(seq(0, 3000000, 500000))) +
  ggtitle("Rating distribution") +
  theme_light()
```

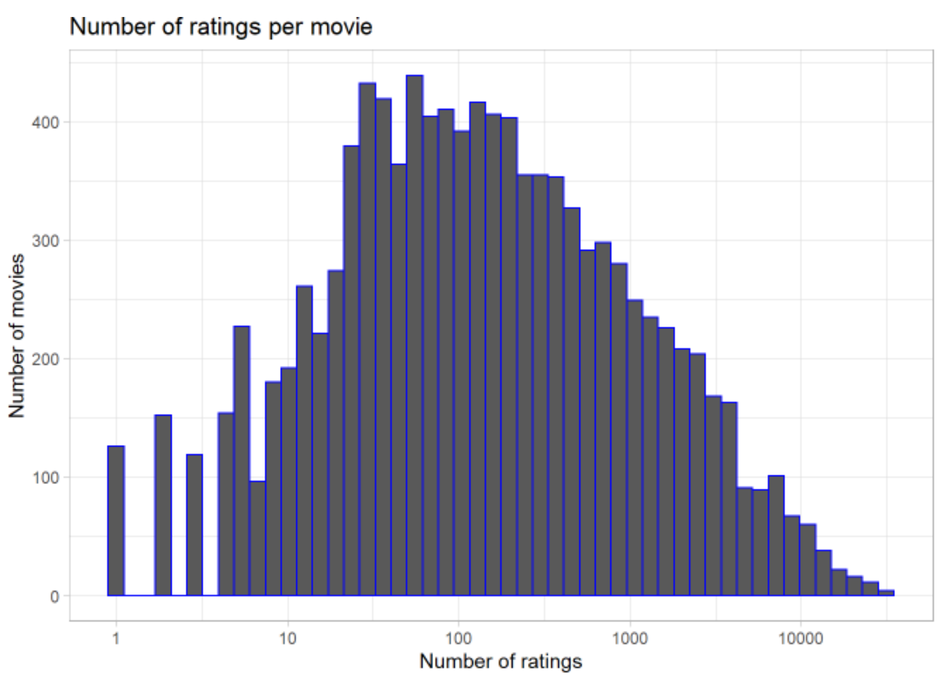


As seen in the graphics showing "MovieId vs number of ratings" and "number of ratings vs. number of movies" below, some movies have been rated much often than other, while some have very few ratings and even some have only one rating. This will be important for our model as very low rating numbers might results in unreliable estimate for our predictions. It is seen that 126 movies in total have been rated only once.

```
edx_movies %>% # for each movie, plot the number of ratings
  ggplot(aes(movieId, num_ratings, color = avg_rating)) +
  geom_point() +
  scale_color_gradientn(colours = rainbow(5)) +
  labs(x = "MovieId", y = "Number of Ratings", title = "Ratings by Movie", color = "Average Rating") +
  theme_light()
```




```
edx %>% # plot the number of ratings per movie (log10 scaled)
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 50, color = "blue") +
  scale_x_log10() +
  xlab("Number of ratings") +
  ylab("Number of movies") +
  ggtitle("Number of ratings per movie") +
  theme_light()
```



As 126 movies that were rated only once, predictions of future ratings for them will be difficult. Below listed only ten of that movies.

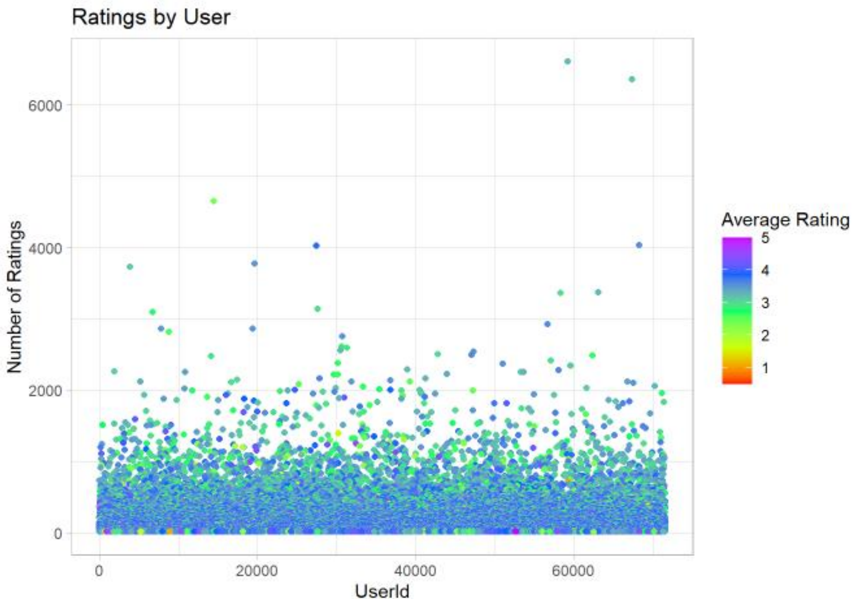
```
# Table of 10 movies rated only once

edx %>%
  group_by(movieId) %>%
  summarize(count = n()) %>%
  filter(count == 1) %>%
  left_join(edx, by = "movieId") %>%
  group_by(title) %>%
  summarize(rating = rating, n_rating = count) %>%
  slice(1:10) %>%
  knitr::kable()
```

title	rating	n_rating
1, 2, 3, Sun (Un, deuz, trois, soleil) (1993)	2.0	1
100 Feet (2008)	2.0	1
4 (2005)	2.5	1
Accused (Anklaget) (2005)	0.5	1
Ace of Hearts (2008)	2.0	1
Ace of Hearts, The (1921)	3.5	1
Adios, Sabata (Indio Black, sai che ti dico: Sei un gran figlio di…) (1971)	1.5	1
Africa addio (1966)	3.0	1
Aleksandra (2007)	3.0	1
Bad Blood (Mauvais sang) (1986)	4.5	1

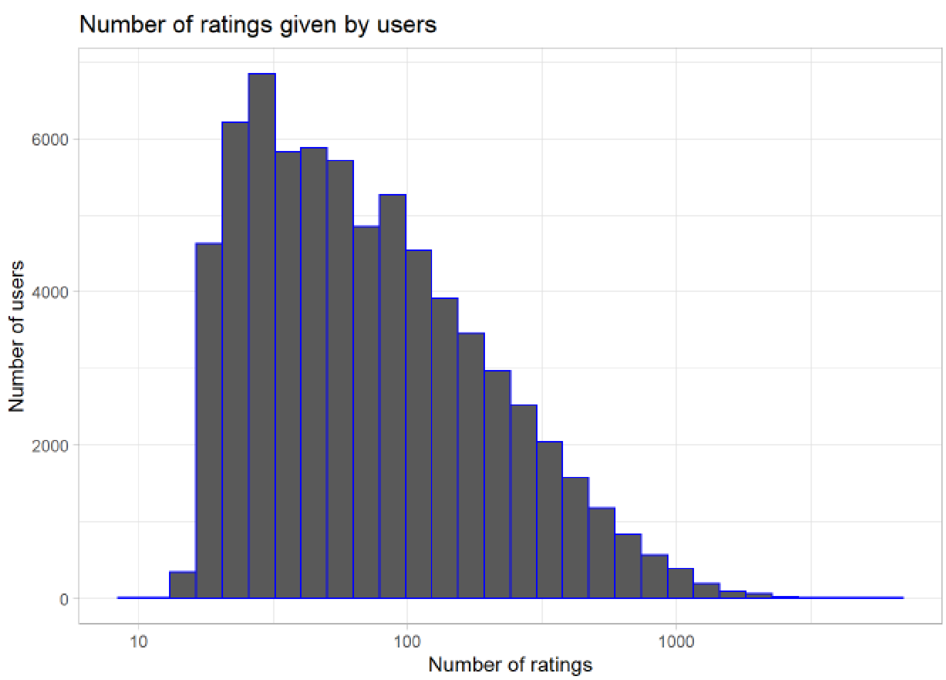
Regularization is a technique used for tuning the function by adding an additional penalty term in the error function. The additional term controls the excessively fluctuating function such that the coefficients don't take extreme values. By this way, it is used to reduce the error by fitting a function appropriately on the given training set and avoid overfitting.

We can see that the majority of users have rated between 30 and 100 movies. So, a user penalty term will be included later in our models.

```
edx_users %>% # for each movie, plot the number of ratings v num of ratings
  ggplot(aes(userId, num_ratings, color = avg_rating)) +
  geom_point() +
  scale_color_gradientn(colours = rainbow(5)) +
  labs(x = "UserId", y = "Number of Ratings", title = "Ratings by User", color = "Average Rating")
```

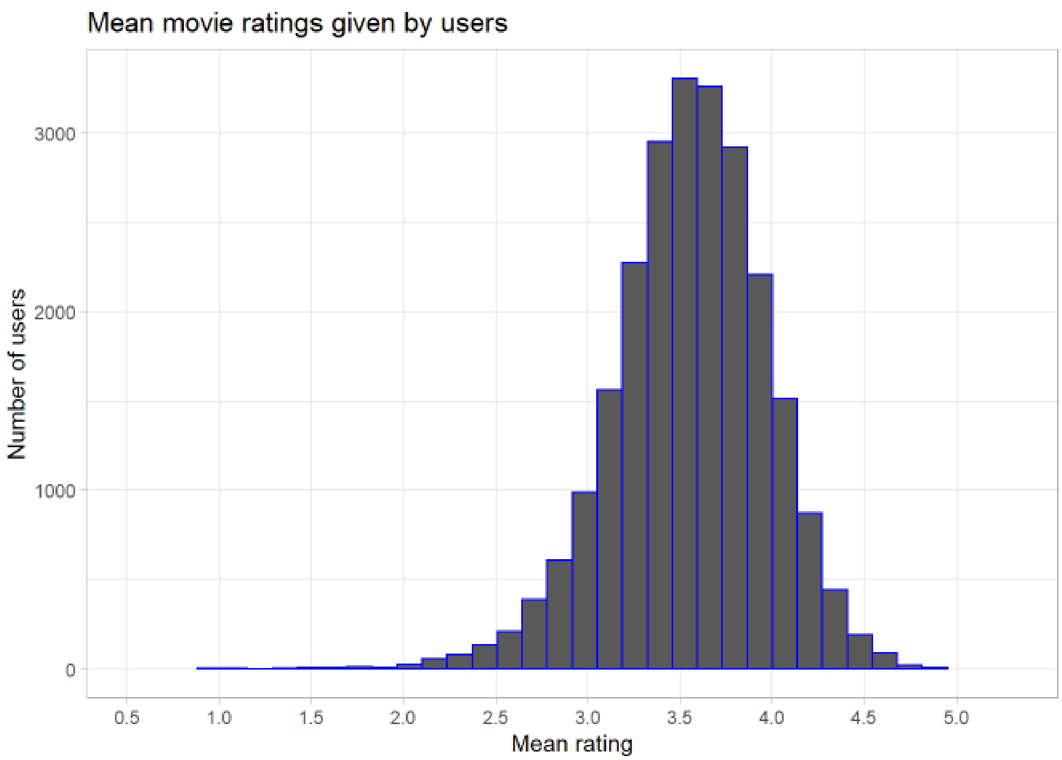



```
edx %>% # plot the number of ratings v number of users (log10 scaled)
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "blue") +
  scale_x_log10() +
  xlab("Number of ratings") +
  ylab("Number of users") +
  ggtitle("Number of ratings given by users") +
  theme_light()
```



Also, it is seen that some users tend to give much lower ratings and some users tend to give higher ratings than average. The visualization below includes only users that have rated at least 100 movies.

```
# Mean movie ratings given by users
edx %>%
  group_by(userId) %>%
  filter(n() >= 100) %>%
  summarize(b_u = mean(rating)) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "blue") +
  xlab("Mean rating") +
  ylab("Number of users") +
  ggtitle("Mean movie ratings given by users") +
  scale_x_discrete(limits = c(seq(0.5,5,0.5))) +
  theme_light()
```



5. Modelling Approach

We write now the loss-function, previously explaned, that compute the RMSE, defined as follows:

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

The RMSE is our measure of model accuracy.

RMSE (Root Mean Squared Error) measures the model prediction error. It corresponds to the average difference between the observed known values of the outcome and the predicted value by the model. RMSE is computed as;

RMSE = mean((observeds - predicteds)^2) %>% sqrt()

The lower the RMSE, the better the model.

a. Average movie rating model

The first basic model predicts the same rating for all movies, so we compute the dataset's mean rating. The expected rating of the dataset is between 3 and 4. We start by building the simplest possible recommendation system by predicting the same rating for all movies regardless of the user. A model based approach assumes the same rating for all movie with all differences explained by random variation:

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

with su,i independent error sample from the same distribution centered at 0 and µ the "true" rating for all movies. This very simple model assumes that all differences in movie ratings are explained by random variation alone. We know that the estimate that minimize the RMSE is the least square estimate of Yu,i , in this case, is the average of all ratings: The expected rating of the underlying dataset is between 3 and 4.

```
# Dataset's average rating

avg_ratg <- mean(edx$rating)
avg_ratg
```

[1] **3.512465**

If we predict all unknown ratings with µ or mu, we obtain the first naive RMSE:

```
# Test results based on simple prediction

naive_rmse <- RMSE(validation$rating, avg_ratg)
naive_rmse
```

[1] **1.061202**

Here, we represent results table with the first RMSE:

```
# Check results and save prediction in data frame
rmse_results <- data_frame(method = "Average movie rating model", RMSE = naive_rmse)
```

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

method	RMSE
Average movie rating model	1.061202

This give us our baseline RMSE to compare with next modelling approaches.

In order to do better than simply predicting the average rating, we will consider some of insights we gained during the exploratory data analysis.

b. Movie effect model

To improve above model, we focus on the fact that some movies are just generally rated higher than others. Higher ratings are mostly linked to popular movies among users and the opposite is true for unpopular movies. We compute the estimated deviation of each movies' mean rating from the total mean of all movies µ. The resulting variable is called "b" (as bias) for each movie "i" bi, that represents average ranking for movie i:

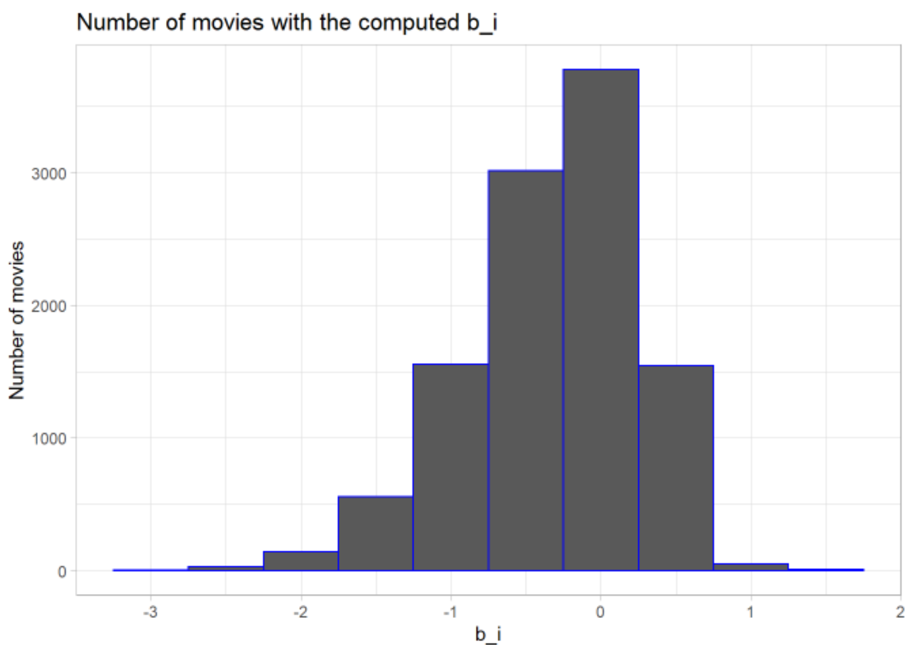
$Y_{u,i} = \mu + b_i + s_{u,i}$

```
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - avg_ratg))
```

```
## # A tibble: 10,677 x 2
##    movieId     b_i
##      <dbl>   <dbl>
##  1       1  0.415
##  2       2 -0.307
##  3       3 -0.365
##  4       4 -0.648
##  5       5 -0.444
##  6       6  0.303
##  7       7 -0.154
##  8       8 -0.378
##  9       9 -0.515
## 10      10 -0.0866
## # ... with 10,667 more rows
```

The histogram implies that more movies have negative effects;

```
movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = ., color = I("blue"),
                     ylab = "Number of movies", main = "Number of movies with the computed b_i") +
  theme_light()
```



This is called the penalty term movie effect. Our prediction will improve once we predict using this model.

```
# Test and save rmse results
predicted_ratings <- avg_ratg +  validation %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
model1_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie effect model",
                                     RMSE = model1_rmse ))


# Check results
rmse_results %>% knitr::kable()
```

method	**RMSE**
Average movie rating model	1.0612018
Movie effect model	0.9439087

So, we have predicted movie rating since movies are rated differently by adding the computed bi to µ. If an individual movie is on average rated worse that the average rating of all movies µ, we predict that it will be rated lower that µ by bi, the difference of the individual movie average from the total average.

We can see an improvement, but this model does not consider the individual user rating effect yet.

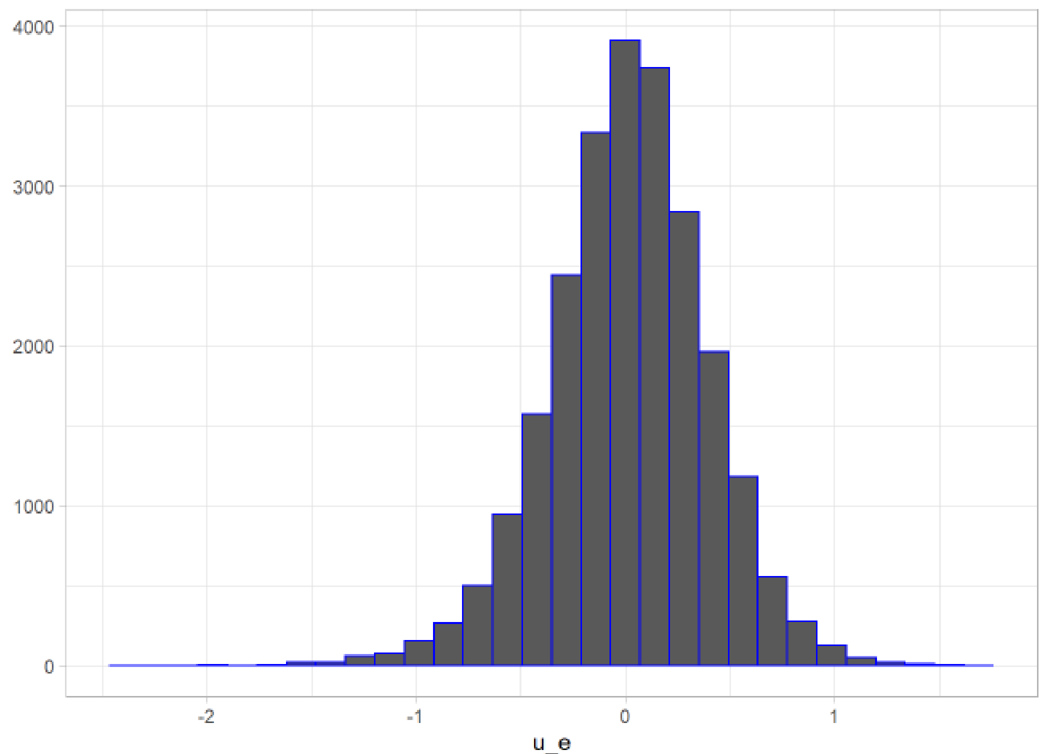
c. Movie and user effect model

We compute the average rating for user µ, for those that have rated over 100 movies, said penalty term user effect.

```
user_avgs<- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  filter(n() >= 100) %>%
  summarize(u_e = mean(rating - avg_ratg - b_i))
```

```
user_avgs%>% qplot(u_e, geom ="histogram", bins = 30, data = ., color = I("blue")) +
  theme_light()
```

```
## # A tibble: 24,115 x 2
##    userId     u_e
##     <int>   <dbl>
##  1      8  0.203
##  2     10  0.0833
##  3     13  0.0186
##  4     18  0.100
##  5     19  0.0682
##  6     30  0.883
##  7     34 -0.494
##  8     35 -0.363
##  9     36  0.0403
## 10     38  0.352
## # ... with 24,105 more rows
```



There is substantial variability across users as well: some users are weird and other love every movie. This implies that further improvement to our model may be:

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

where u_e is a user-specific effect. If a weird user (negative u_e rates a great movie (positive bi), the effects counter each other and we may be able to correctly predict that this user gave this great movie a 3 rather than a 5.

We compute an approximation by computing µ and bi, and estimating ue, as the average of

$Y_{u,i} - \mu - b_i$

```
user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(u_e = mean(rating - avg_ratg - b_i))
```

```
## # A tibble: 69,878 x 2
##    userId     u_e
##     <int>   <dbl>
##  1      1  1.68
##  2      2 -0.236
##  3      3  0.264
##  4      4  0.652
##  5      5  0.0853
##  6      6  0.346
##  7      7  0.0238
##  8      8  0.203
##  9      9  0.232
## 10     10  0.0833
## # ... with 69,868 more rows
```

We can now construct predictors and see RMSE improves:

```
# Test and save rmse results

predicted_ratings <- validation%>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(predct = avg_ratg + b_i + u_e) %>%
  pull(predct)
```

```
model2_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,

                          data_frame(method="Movie and user effect model",

                                     RMSE = model2_rmse))

# Check result
rmse_results %>% knitr::kable()
```

method	RMSE
Average movie rating model	1.0612018
Movie effect model	0.9439087
Movie and user effect model	0.8653488

Our rating predictions with movie and user effect model reduced the RMSE.

Until now, we computed standard error and constructed confidence intervals to account for different levels of uncertainty. However, when making predictions, we need one number, one prediction, not an interval. For this we introduce the concept of regularization, that permits to penalize large estimates that come from small sample sizes. The general idea is to add a penalty for large values of bi to the sum of squares equation that we minimize. So, having many large bi, make it harder to minimize. Regularization is a method used to reduce the effect of overfitting.

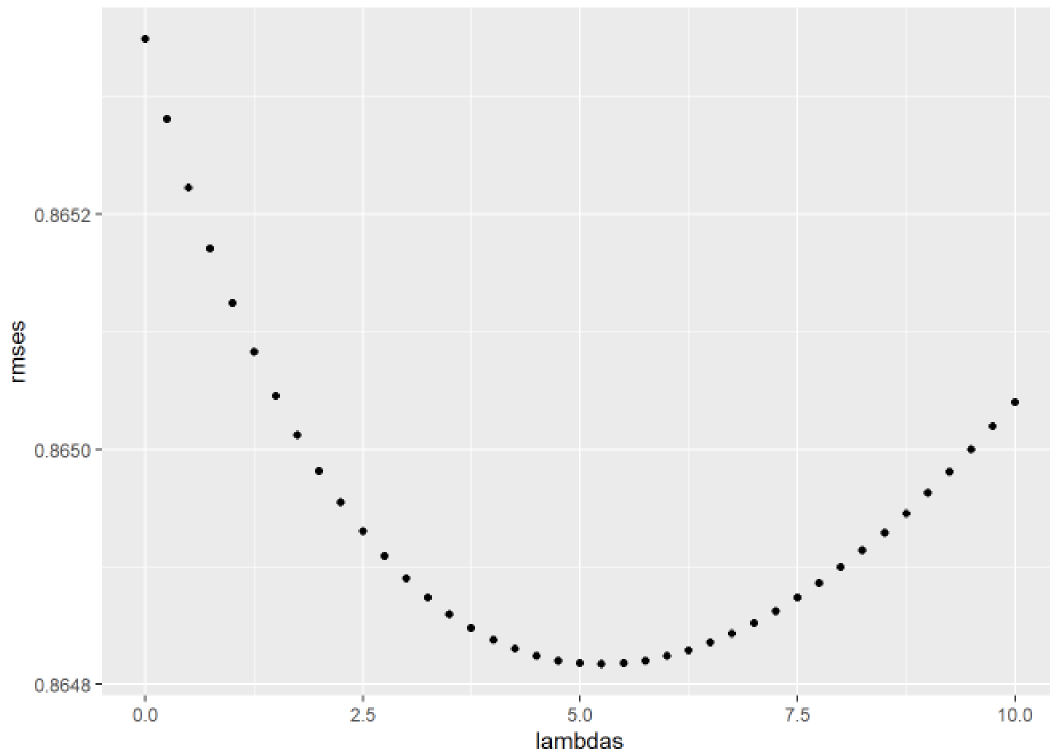
d. Regularized movie and user effect model

So, estimates of bi and ue are caused by movies with very few ratings and in some users that only rated a very small number of movies. Hence this can strongly influence the prediction. The use of the regularization permits to penalize these aspects. We should find the value of lambda (that is a tuning parameter) that will minimize the RMSE. This shrinks the bi and ue in case of small number of ratings.

```
# lambda is a tuning parameter
# Use cross-validation to choose it.

lambdas <- seq(0, 10, 0.25)


# For each lambda,find b_i & u_e, followed by rating prediction & testing

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

  avg_ratg <- mean(edx$rating)

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

u_e <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(u_e = sum(rating - b_i - avg_ratg)/(n()+r))

predicted_ratings <-
  validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(u_e, by = "userId") %>%
  mutate(predct = avg_ratg + b_i + u_e) %>%
  pull(predct)

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

We plot RMSE vs lambdas to select the optimal lambda

```
# Plot rmses vs lambdas to select the optimal lambda
qplot(lambdas, rmses)
```



For the full model, the optimal lambda is:

```
# The optimal lambda
opt_lambda <- lambdas[which.min(rmses)]
opt_lambda
```

[1] 5.25

The new results will be:

```
# Test and save results
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized movie and user effect model",
                                     RMSE = min(rmses)))

# Check result
rmse_results %>% knitr::kable()
```

method	RMSE
Average movie rating model	1.0612018
Movie effect model	0.9439087
Movie and user effect model	0.8653488
Regularized movie and user effect model	0.8648170

6. Results

The RMSE values of each models are the following:

method	RMSE
Average movie rating model	1.0612018
Movie effect model	0.9439087
Movie and user effect model	0.8653488
Regularized movie and user effect model	**0.8648170**

We therefore found the lowest value of RMSE that is 0.8648170.

We can confirm that the final model for our project is the following:

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

This model work well if the average user doesn't rate a particularly good/popular movie with a large positive bi, by disliking a particular movie.

7. Conclusion

We can affirm to have built a machine learning algorithm to predict movie ratings with MovieLens dataset. The regularized model including the effect of user is characterized by the lower RMSE value and is hence the optimal model to use for the present project. The optimal model characterized by the lowest RMSE value (0.8648170) lower than the evaluation criteria (0.86490) designated in the MovieLens Grading Rubric. We could also affirm that improvements in the RMSE could be achieved by adding other effects such as genre, year, age etc. Other different machine learning models could also improve the results further, but hardware limitations, as the RAM, are a constraint.

8. Appendix - Enviroment

```
#### Appendix ####
print("Operating System:")
```

```
##                _
## platform       x86_64-w64-mingw32
## arch           x86_64
## os             mingw32
## system         x86_64, mingw32
## status
## major          4
## minor          0.3
## year           2020
## month          10
## day            10
## svn rev        79318
## language       R
## version.string R version 4.0.3 (2020-10-10)
## nickname       Bunny-Wunnies Freak Out
```