HarvardX: PH125.9x Data Science: Capstone Course-MovieLens Project

Ioannis Dimitriou

10 February 2020

Intoduction

HarvardX PH125.9x: Capstone is not only the final course of the Professional Certificate in Data Science, but also a chance to apply all the knowledge gained throughout the program. The first part of the Capstone is based on the MovieLens Project. This project is aiming to develop a machine learning code to predict a rating given by a user to a movie. In this report will be presented: 1. An overview of the data in the dataset 2. Analysis 3. Results that will be produced 4. A conclusion

Dataset

Before starting with the dataset, it is necessary to refer a limitation on my PC system (32-bit) that made it impossible to load the Movielens 10M dataset through the code provided by the course. After contacting with the staff i was encouraged to download the "edx"" and the "validation" set directly from the Google Drive, through the link provided below: https://drive.google.com/drive/folders/1IZcBBX0OmL9wu9AdzMBFUG8GoPbGQ38D?usp=sharing I would also like to mention that there are some faults in the pdf report because of that limitation in my system.

```
#package installs
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")

#libraries
library(tidyverse)
library(caret)

# MovieLens 10M dataset:
# edx: https://drive.google.com/open?id=1e2hKKMvgcyM4pHQYRUC_4l55Qudep46z
 #validation: https://drive.google.com/open?id=19Nh8bw3MKoxF7fN1Maa3cYIvVKTsG82i
# Due to the limitation mentioned above the files from Drive were downloaded on my pc and then were rea
 edx <- readRDS("C:/Users/giand/Desktop/edx.rds")
 validation <- readRDS("C:/Users/giand/Desktop/validation.rds")

#################################
# Create edx set, validation 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")

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

# 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)` instead
#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)
```

Edx dataset contains rows that correspond to a user's rating of a movie. The following variables are met in this set: "userId", "movieId", "rating", "timestamp", "title", "genres".

```
#Summarise Data
head(edx)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1                 Comedy|Romance
## 2          Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5        Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7        Children|Comedy|Fantasy
```

Summarising shows us that the movies are rated between 0.5 and 5.0 with 9000055 rows in total.

```
summary(edx)
```

```
##      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
##
##
##
```

The dataset consists of * 10677 unique movies * 69878 unique users * 797 unique genres and the mean movie rating is ~ 3.5/5

```
# Movies, Users and Genres in Database
edx %>% summarise(
  unique_movies = n_distinct(movieId),
  unique_users = n_distinct(userId),
  unique_genres = n_distinct(genres))
```

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

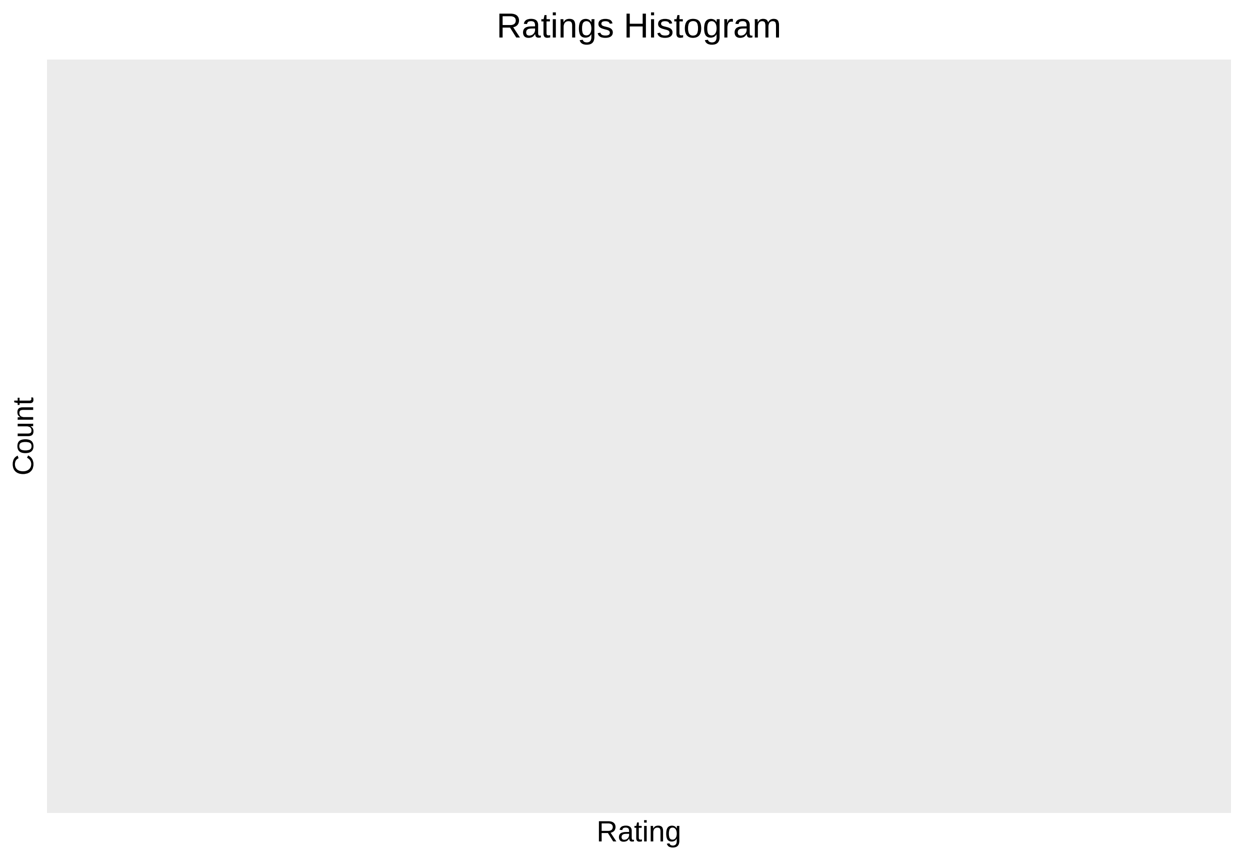
```
# Ratings Mean
rating_mean <- mean(edx$rating)
rating_mean
```

```
## [1] 3.512465
```

A histogram can be very helpful to show the ratings and their counts over the users in the dataset. The most common ratings are 5.0, 3.0, 4.0 in increasing order. It can also be assumed tha users are more likely to rate full ratings rather than half ratings in the scale.

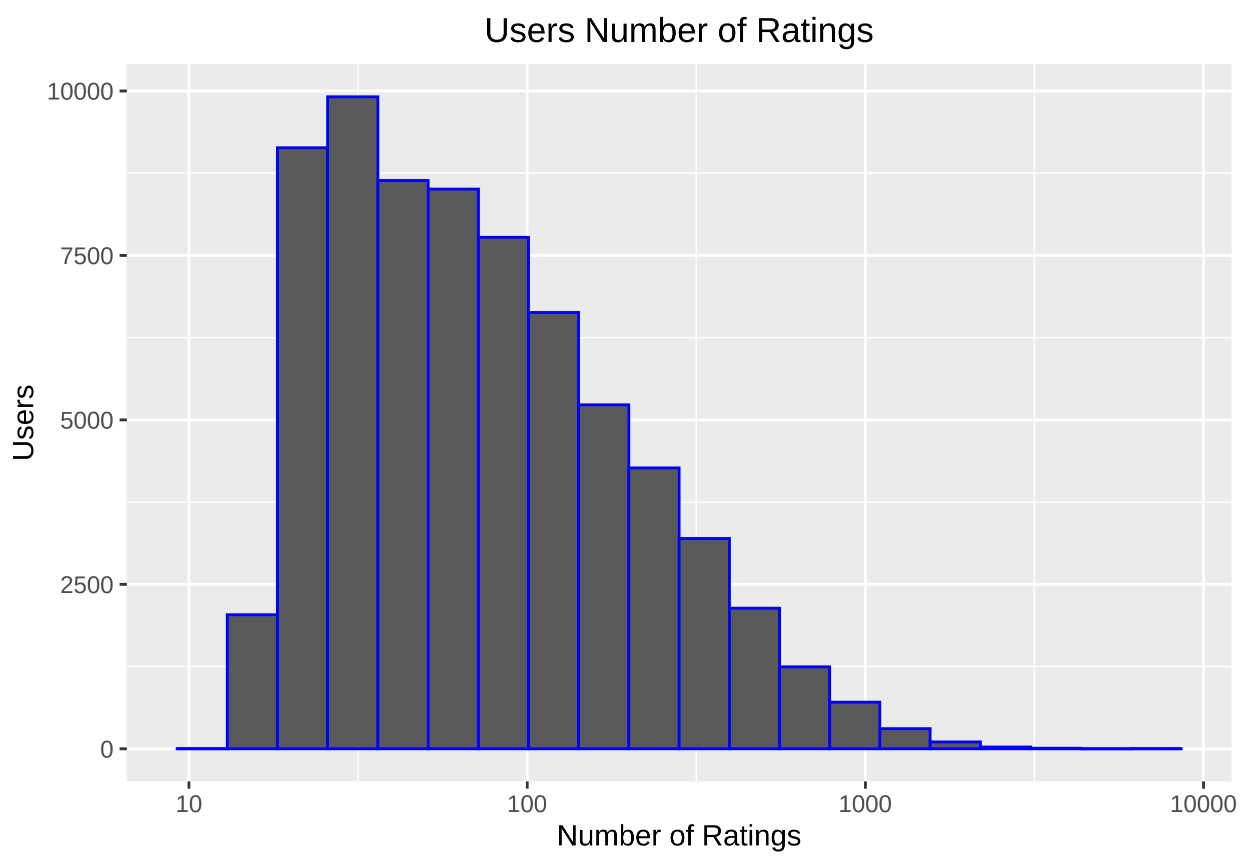
```
edx %>%
  ggplot(aes(rating)) +
  geom_histogram(binwidth = 0.5, color = "blue") +
  xlab("Rating") +
  ylab("Count") +
  ggtitle("Ratings Histogram") +
  theme(plot.title = element_text(hjust = 0.5))
```

```
## Warning: Computation failed in `stat_bin()`:
## cannot allocate vector of size 4.0 Mb
```



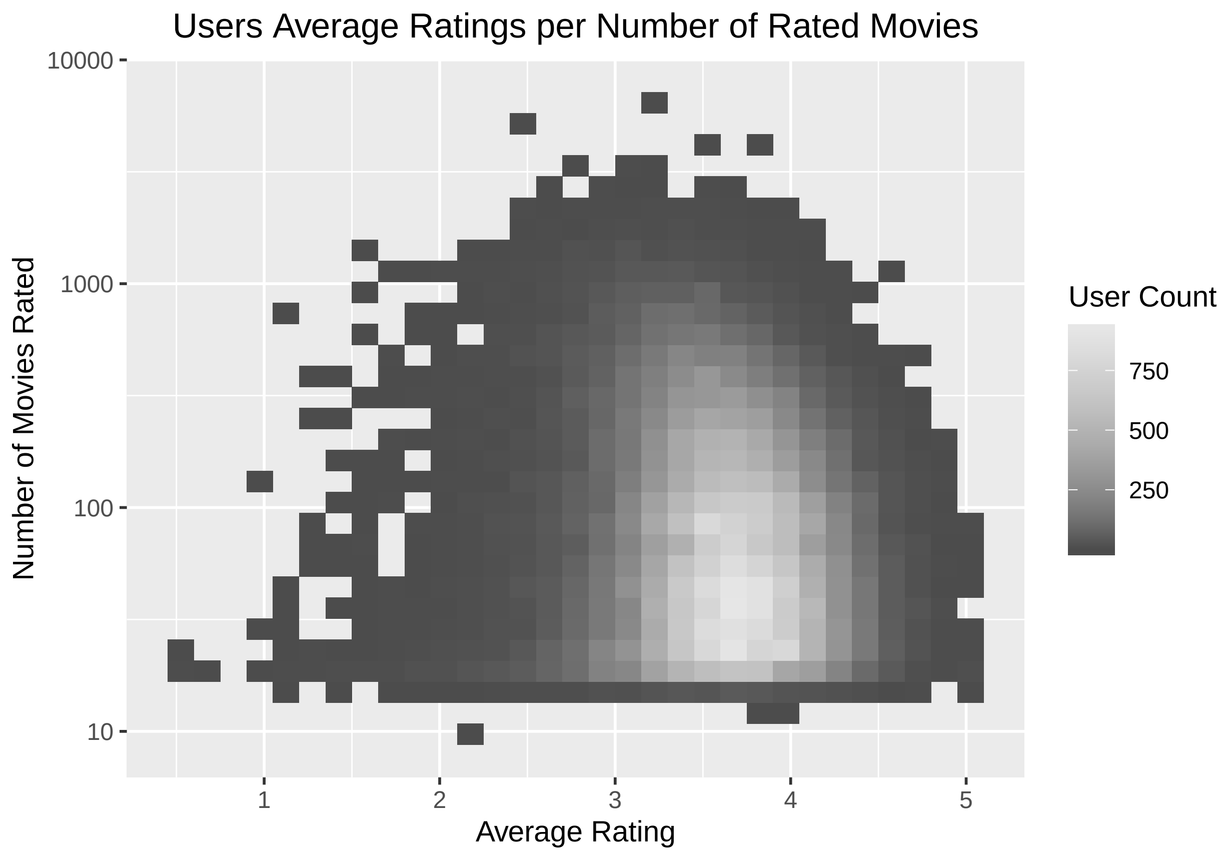
Most users rate between 10 and 100 movies, nevertheless some may rate over 1000. As a result a variable in the model for the number of ratings by the users, may be really useful and reasonable.

```
# Users Number of Ratings
edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(color = "blue", bins=20) +
  scale_x_log10() +
  xlab("Number of Ratings") +
  ylab("Users") +
  ggtitle("Users Number of Ratings") +
  theme(plot.title = element_text(hjust = 0.5))
```



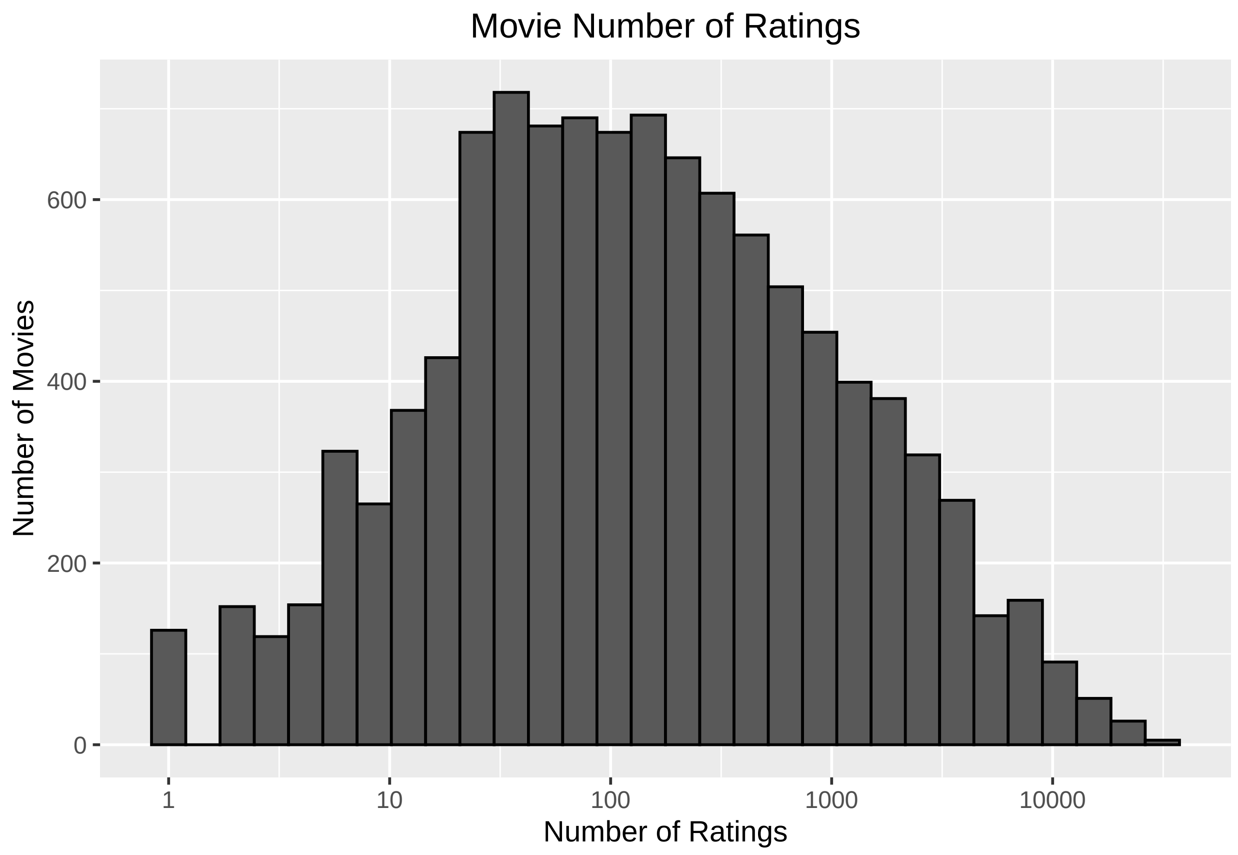
We can see average movie rating against the number of movies rated. A heat map can use highlighting to show the number of movies rated and the most common reting.

```
# Movies Rated- Users average rating
edx %>%
  group_by(userId) %>%
  summarise(mu_user = mean(rating), number = n()) %>%
  ggplot(aes(x = mu_user, y = number)) +
  geom_bin2d( ) +
  scale_fill_gradientn(colors = grey.colors(10)) +
  labs(fill="User Count") +
  scale_y_log10() +
  ggtitle("Users Average Ratings per Number of Rated Movies") +
  xlab("Average Rating") +
  ylab("Number of Movies Rated") +
  theme(plot.title = element_text(hjust = 0.5))
```



Inspecting the dataset in more detail, it can be easily seen that there are movies having 10 ratings or less in total. This makes it difficult to predict the rating for them because there is no sufficient sample.

```
# Ratings Movies - Number of Ratings
edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(color = "black", bins=30) +
  scale_x_log10() +
  xlab("Number of Ratings") +
  ylab("Number of Movies") +
  ggtitle("Movie Number of Ratings") +
  theme(plot.title = element_text(hjust = 0.5))
```


Movie Number of Ratings
Number of Movies
0
200
400
600
Number of Ratings
1
10
100
1000
10000

Analysis and Results

The RMSE, Residual Mean Squared Error, is used in that report in order to measure the accuracy and the typical error of the model. Three models will be invesigated until we can reach the acceptable typical error for our data. Note: On this point i would like to mention that i used the validation set on the models #2, #3 and the regulized model, instead of the edx set, on finding mu, b__i and b__u because my system was crashing processing edx data set and could not "lift its weight". It is wrong, but it was the only way t produce a report through my system.

1. Prediction based on Mean Rating

This is the simpliest prediction model based only on tha mean of the ratings. It is implied that all the diffirences between the expected and the actual outcome of the rating is due to the typical error and does not take into account any other impact. The mathematical expression of that model is shown below:

$$Yu, i = \mu + \epsilon u, i$$

Where mu stands for the mean rating and epsilon the typical error. Investigating the dataset will produce the following RMSE.

```
# Prediction based on Mean Rating
mu<- mean(edx$rating)
mu
```

```
## [1] 3.512465
```

```
mean_rmse<- RMSE(validation$rating, mu)
mean_rmse
```

```
## [1] 1.061202
```

```
#Save the results in data frame
rmse_results<- data.frame(Method= "Mean Rating", RMSE=mean_rmse)
rmse_results%>% knitr::kable()
```

Method
Mean Rating
The RMSE of th   at model is not acceptable and we must investigate the dataset further using model with more variables

2. Movie Effect Model

The Movie Effect model inserts a bias term in our prediction tha pictures and quantifies the effect of each movie on their ratings. It is based on the difference between each movie's mean rating and the overall mean rating of all the movies in the dataset. The mathematical expression of that model is shown below.

$$Yu, i = \mu + b_i + \epsilon u, i$$

Where b__i is the bias from the Movie Effect

```
#Movie effect Model, importing movie's mean rating bias (b_i)
mu<- mean(validation$rating)
movie_avg<- validation %>%
  group_by(movieId) %>%
  summarise(b_i=mean(rating-mu))
```

```r
predicted_ratings<- mu + validation %>%
  left_join(movie_avg, by= "movieId") %>%
  .$b_i

movie_RMSE<-RMSE(predicted_ratings, validation$rating)
rmse_results<-bind_rows(rmse_results, data.frame(Method= " Movie Effect Model", RMSE= movie_RMSE))
```

```
## Warning in bind_rows_(x, .id): Unequal factor levels: coercing to character

## Warning in bind_rows_(x, .id): binding character and factor vector, coercing
## into character vector

## Warning in bind_rows_(x, .id): binding character and factor vector, coercing
## into character vector
```

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

Method	RMSE
Mean Rating	1.0612018
Movie Effect Model	0.9383091

In that model we see an improved RMSE, but we can still do better by investigating further and importing other variables that affect the rating of a movie in our model.

3. Movie and User Effect Model

We can easily think that, except from the movie rating effect, there is the user effect, that is every user has his own preferances on movies so in this way the import an extra bias in our model called the user effect. The mathematical expression of that model is shown below.

$$Yu,i = \mu + b_i + b_u + \epsilon_{u,i}$$

Where b_u stands for the specific user effect.

```r
user_avgs<- validation %>% left_join(movie_avg, by="movieId") %>%
  group_by(userId)%>%
  summarise(b_u = mean(rating-mu-b_i))

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

user_movie_model_RMSE<- RMSE(predicted_ratings, validation$rating)
rmse_results<- bind_rows(rmse_results, data.frame(Method= "Movie and User Effct Model", RMSE= user_movi
```

```
## Warning in bind_rows_(x, .id): binding character and factor vector, coercing
## into character vector
```

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

Method	RMSE
Mean Rating	1.0612018
Movie Effect Model	0.9383091

Method	RMSE
Movie and User Effct Model	0.8251770

As we can see, the RMSE is improved by importing an extra bias and its effect on the model.

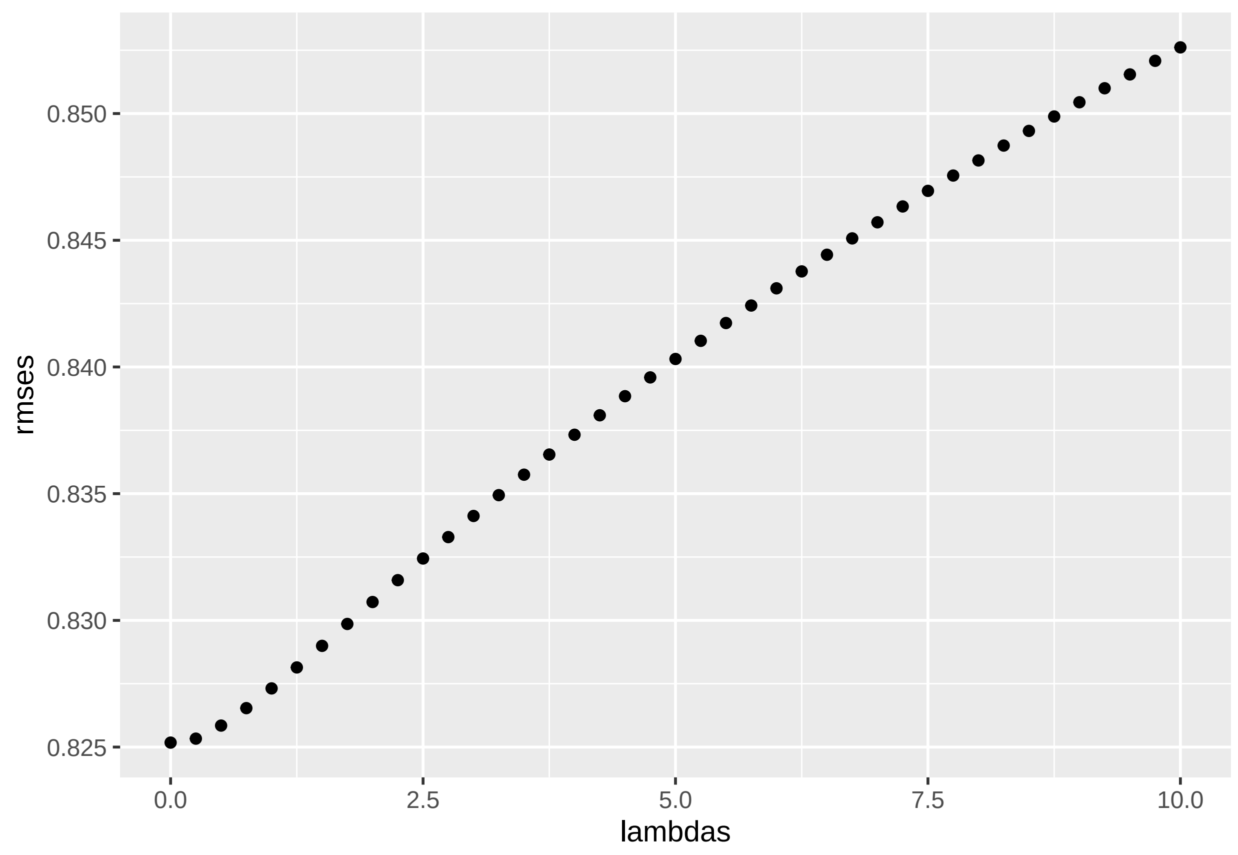
Regularization

Regularization can provide reduced errors by penalizing the movies that have very few ratings. This happens because those movies do not provide a sufficient sample for prediction so the possibility to lead to errors is very big and as result they increase the overall error of the model. The method is using a parametel called "lambda", in order to restrict the impact of those movies with small samples, on the model. We use different lambdas to find the one that returns the smallest RMSE and choose it for our model.

```r
#Regularization on the Movie and User Model
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  mu <- mean(validation$rating)
  b_i <- validation %>%
    group_by(movieId) %>%
    summarise(b_i = sum(rating - mu)/(n()+l))

    b_u <- validation %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarise(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) %>%
    pull(pred)
  return(RMSE(predicted_ratings, validation$rating))

})

rmse_regularisation <- min(rmses)
rmse_regularisation
```

```
## [1] 0.825177
```

```r
# Plot RMSE against Lambdas to find optimal lambda
qplot(lambdas, rmses)
```



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

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

```
rmse_results <- bind_rows(rmse_results,                    data_frame(Method="Regularised Movie and User
```

```
## Warning: `data_frame()` is deprecated, use `tibble()`.
## This warning is displayed once per session.
```

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

Method	RMSE
Mean Rating	1.0612018
Movie Effect Model	0.9383091
Movie and User Effct Model	0.8251770
Regularised Movie and User Effects Model	0.8251770

We succeeded the lowest RMSE via the Regularization of the Movie and User Effects Model for the MovieLens dataset.

Results and Discussion

The final results of the prediction models are shown below

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

Method	RMSE
Mean Rating	1.0612018
Movie Effect Model	0.9383091
Movie and User Effct Model	0.8251770
Regularised Movie and User Effects Model	0.8251770

The accuracy of each model is shown on the above table and the most accurate one is the "Regulized Movie and User Effects Model", because via this model we succeeded the lowest RMSE. The lowest RMSE value predicted is. . . .(The RMSE would be at about 0.860-0.868 if my system could run the code properly.) The final model used for the prediction is:

$$Y_{u,i} = \mu + b_{i,n,\lambda} + b_{u,n,\lambda} + \epsilon_{u,i}$$

Conclusion

The aim of this project was to construct a machine learning algorithm to predict the ratings from the movie recommendation dataset "MovieLens". The optimal model took into account the movie and the user effect, except from the typical error, and finally was regulized, in order to restrict the impact of the movies with very few ratings on the overall model and its accuracy. The main goal was to construct a model with a RMSE lower than 0.87750 and it was achieved via the "Regulized Movie and User Effects Model". I would be thankful if you took into account my system's limitation

Appendix

Environment

```
##                _
## platform       i386-w64-mingw32
## arch           i386
## os             mingw32
## system         i386, mingw32
## status
## major          3
## minor          6.1
## year           2019
## month          07
## day            05
## svn rev        76782
## language       R
## version.string R version 3.6.1 (2019-07-05)
## nickname       Action of the Toes
```