HarvardX: PH125.9x Data Science
MovieLens Rating Prediction Project

Chanathip Eamdeengamlert

2/6/2021

Contents

Introduction

This is a project as a part of the ninth and final course, HarvardX PH125.9x - Data Science: Capstone, in HarvardX's multi-part Data Science Professional Certificate series.

For this project, main objective is to create the recommendation system using all the tools we have shown throughout the courses in this series. Develop the algorithm and final test prediction of movie ratings on the Movielens dataset. RMSE will be used to evaluate how close the predictions are to the true values in the validation set.

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})^2}$$

```r
## RMSE Function ##
RMSE <- function(real , pred) {
  sqrt(mean((real - pred)^2))
}
```

Overview

The version of Movielens included in the dslabs package (which was used for some of the exercises in PH125.8x: Data Science: Machine Learning) is just a small subset of a much larger dataset with millions of ratings. In this project, We will use the 10M version of the MovieLens dataset to make the computation a little easier.

The code to generate datasets was prepared by HarvardX (The code will not be shown here). The prepared code generates two datasets, 'edx' set and 'validation' set. The development of the algorithm will be done on the edx set. And for a final test, the prediction of movie ratings will be done on the validation set.

```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"))
```

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

```
##      movieId title
## [1,] "1"     "Toy Story (1995)"
## [2,] "2"     "Jumanji (1995)"
## [3,] "3"     "Grumpier Old Men (1995)"
## [4,] "4"     "Waiting to Exhale (1995)"
## [5,] "5"     "Father of the Bride Part II (1995)"
## [6,] "6"     "Heat (1995)"
##      genres
## [1,] "Adventure|Animation|Children|Comedy|Fantasy"
## [2,] "Adventure|Children|Fantasy"
## [3,] "Comedy|Romance"
## [4,] "Comedy|Drama|Romance"
## [5,] "Comedy"
## [6,] "Action|Crime|Thriller"
```

```r
# 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")
```

For validation set, it took 10% of MovieLens dataset.

```r
# 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)
```

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

```r
edx <- rbind(edx, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

The dataset

The dataset contains total of 6 varibles; userId, movieId, rating, timestamp, title and genres. UserId and timestamp are stored in integer datatype, MovieId and rating are stored in numeric datatype, Title and genres are stored in character datatype as shown below;

```
str(edx,width=70,strict.width="cut")
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392..
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak ("..
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action"..
##  - attr(*, ".internal.selfref")=<externalptr>
```

Edx set contains total of 9,000,055 observations and validation set contains 999,999 observations (10% split).

```
str(validation,width=70,strict.width="cut")
```

```
## Classes 'data.table' and 'data.frame':   999999 obs. of  6 variables:
##  $ userId   : int  1 1 1 2 2 2 3 3 4 4 ...
##  $ movieId  : num  231 480 586 151 858 ...
##  $ rating   : num  5 5 5 3 2 3 3.5 4.5 5 3 ...
##  $ timestamp: int  838983392 838983653 838984068 868246450 868245645..
##  $ title    : chr  "Dumb & Dumber (1994)" "Jurassic Park (1993)" "H"..
##  $ genres   : chr  "Comedy" "Action|Adventure|Sci-Fi|Thriller" "Chi"..
##  - attr(*, ".internal.selfref")=<externalptr>
```

To be noticed that title column is stored both movie title and movie release year in character type. Also genres column contains multiple genres each row that separate by "|" symbol.

```
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
```

Executive Summary

After import and load all required library, data cleansing and data manipulation was done in first step. Time stamp column is formatted to date-time data type. Numbers feature were extracted from existing column for example movie release year, rating year, rating month and number of genres.

Then, exploratory data analysis was done in second step. Rating distribution, rating frequency, difference in mean rating and more were analyzed on many aspects for example userId, movieId, movie release year, rating year, rating month and number of genres of each movie. It was done to find out if any aspect is effecting to the movie rating.

Insights from EDA has shown that UserId, MovieID and Movie release year are affected the movie rating and there are rating frequency issue (highly difference in maximum count and minimum count) which may affect the performance of the algorithm.

The final approach is to use UserId, MovieID and Movie release year to create the algorithm and regularization may need to reduce the effect of frequency difference.

Since the validation data is only used for evaluating the RMSE of the final algorithm. The edx data is separated into training and test sets to design and test the algorithm.

The combination of Mean, UserId, MovieId and Movie release year are used to build and test the algorithm. The final model which give the best result, lowest RMSE, is the combination of mean rating + movieID effect + userId effect and movie release year with regularization.

Method and Analysis

Essential library

```r
library(tidyverse)
library(lubridate)
library(stringr)
```

Data manipulation

In order to have a proper and workable dataset structure, some manipulation has to be done. Also some feature has to be extracted from current column to be explored in the next step.

Firstly, timestamp column need to be formatted into workable format. UserId and movieId should be convert to factor.

```r
# Edit timestamp
edx <- edx %>% mutate(timestamp = as_datetime(timestamp)) %>%
  mutate(userId = as.factor(userId), movieId = as.factor(movieId))
```

Next, it is observed that movie release year is stored in the same column of movie title. Movie release year may effect to the rating, so it is extracted from title column to have a deeper explore.

```r
# Seperate movie release year
edx <- edx %>% mutate(movieYear = str_sub(title, -5,-2)) %>%
  mutate(movieYear = as.numeric(movieYear))
```

Rating year, rating month and number of genres are also suspected to effect the rating, so they are extracted into separate column in order to have a deeper explore.

```r
# Seperate rating year, month
edx <-  edx %>% mutate(timestamp = as_datetime(timestamp)) %>%
  mutate(ratingYear = year(timestamp), ratingMonth = month(timestamp))
```

```r
# get number of genres
edx <- edx %>% mutate(numberGenres = (str_count(genres, pattern = "\\|")+1))
```

The final edx set to be used looks like this.

```r
str(edx,width=70,strict.width="cut")
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  10 variables:
##  $ userId      : Factor w/ 69878 levels "1","2","3","4",..: 1 1 1 1 ..
##  $ movieId     : Factor w/ 10677 levels "1","2","3","4",..: 121 184 ..
##  $ rating      : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp   : POSIXct, format: "1996-08-02 11:24:06" ...
##  $ title       : chr  "Boomerang (1992)" "Net, The (1995)" "Outbrea"..
##  $ genres      : chr  "Comedy|Romance" "Action|Crime|Thriller" "Act"..
##  $ movieYear   : num  1992 1995 1995 1994 1994 ...
##  $ ratingYear  : num  1996 1996 1996 1996 1996 ...
##  $ ratingMonth : num  8 8 8 8 8 8 8 8 8 8 ...
##  $ numberGenres: num  2 3 4 3 4 3 4 3 5 2 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

```r
head(edx)
```

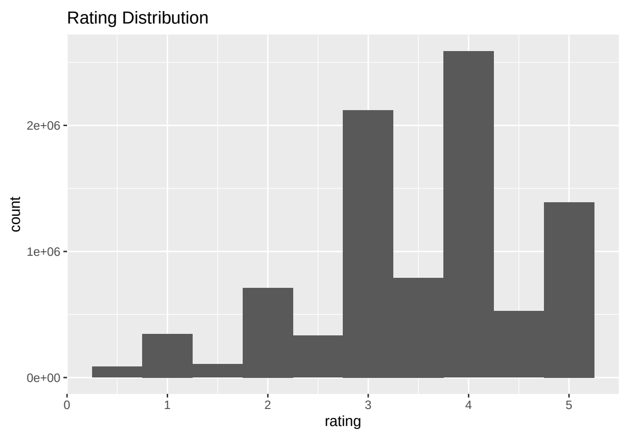
```
##    userId movieId rating           timestamp                         title
## 1:      1     122      5 1996-08-02 11:24:06              Boomerang (1992)
## 2:      1     185      5 1996-08-02 10:58:45               Net, The (1995)
## 3:      1     292      5 1996-08-02 10:57:01               Outbreak (1995)
## 4:      1     316      5 1996-08-02 10:56:32               Stargate (1994)
## 5:      1     329      5 1996-08-02 10:56:32 Star Trek: Generations (1994)
## 6:      1     355      5 1996-08-02 11:14:34       Flintstones, The (1994)
##                           genres movieYear ratingYear ratingMonth numberGenres
## 1:                Comedy|Romance      1992       1996           8            2
## 2:         Action|Crime|Thriller      1995       1996           8            3
## 3:  Action|Drama|Sci-Fi|Thriller      1995       1996           8            4
## 4:       Action|Adventure|Sci-Fi      1994       1996           8            3
## 5: Action|Adventure|Drama|Sci-Fi      1994       1996           8            4
## 6:       Children|Comedy|Fantasy      1994       1996           8            3
```

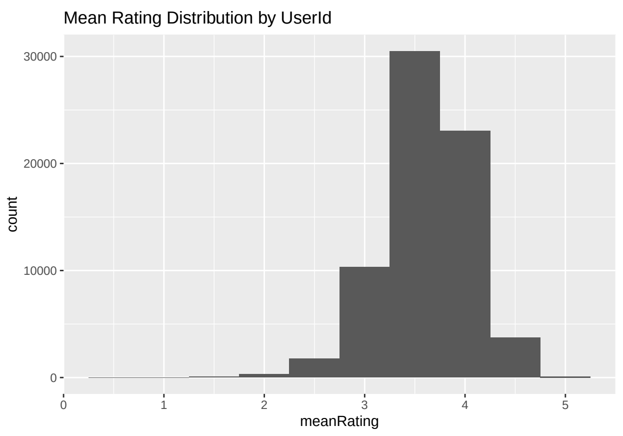
Exploratory Data Analysis

Explore Effect of UserID

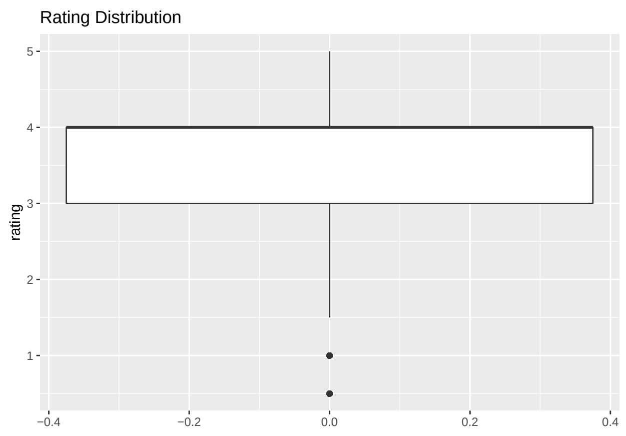
When compared between mean rating and mean rating by userId, it is clearly shows the significant difference by both histogram and boxplot. This can conclude that userId have an effect on movie rating and userId feature will be included in final algorithm.

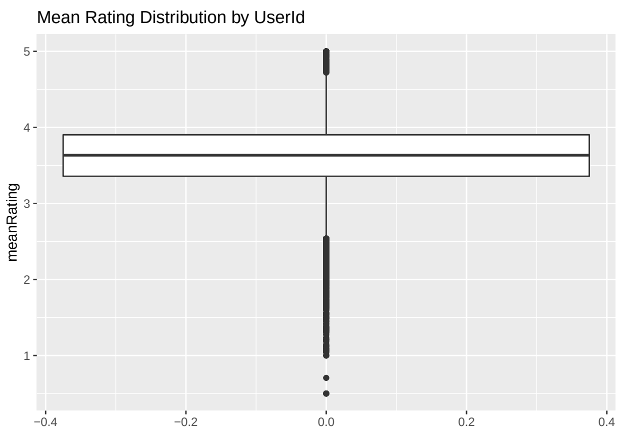
```
edx %>% ggplot(aes(x = rating)) + geom_histogram(bins = 10) +
  labs(title = 'Rating Distribution')
edx %>% group_by(userId) %>% summarize(meanRating = mean(rating)) %>%
  ggplot(aes(x = meanRating)) + geom_histogram(bins = 10)  +
  labs(title = 'Mean Rating Distribution by UserId')
```





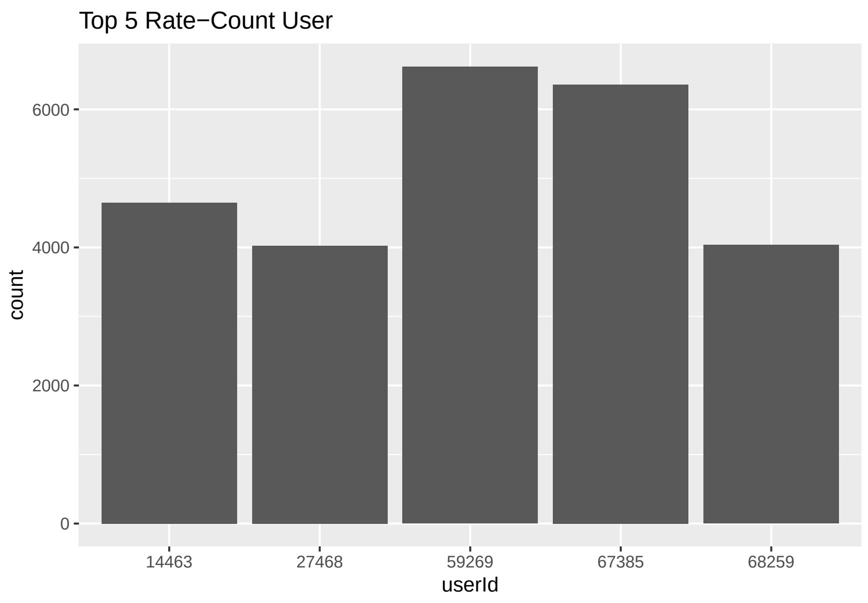
```
edx %>% ggplot(aes(y = rating)) + geom_boxplot() +
  labs(title = 'Rating Distribution')
edx %>% group_by(userId) %>% summarize(meanRating = mean(rating)) %>%
  ggplot(aes(y = meanRating)) + geom_boxplot()  +
  labs(title = 'Mean Rating Distribution by UserId')

# This shows bias from userId.
# Some user rated very low in average, at the same time, some vote very high in average.
```



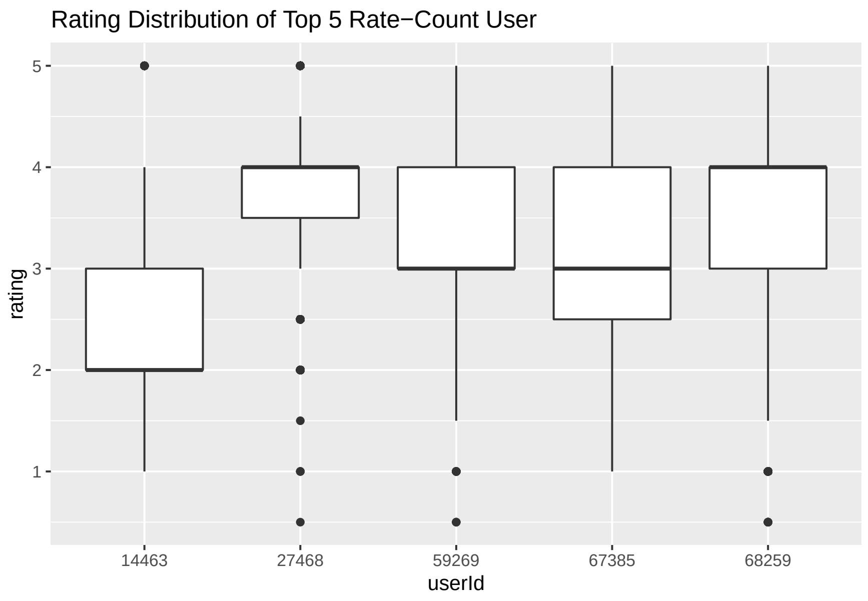


Deep down into top 5 rate-count userId, we can see clearly a significant difference in rating behavior and mean of rating.

```r
edx %>% group_by(userId) %>% summarize(count = n()) %>%
  arrange(desc(count)) %>% top_n(n = 5) %>%
  ggplot(aes(x = userId, y = count)) + geom_col() +
  labs(title = 'Top 5 Rate-Count User')
```

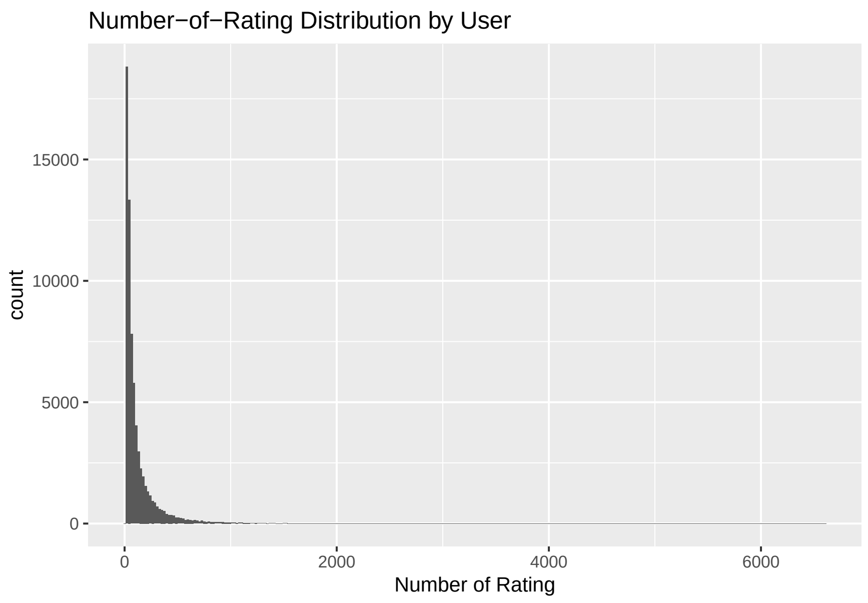


```r
edx %>% filter(userId %in% c(59269,67385,14463,68259,27468)) %>%
  ggplot(aes(y = rating, x = userId)) + geom_boxplot() +
  labs(title = 'Rating Distribution of Top 5 Rate-Count User')
```



After plotting Number-of-Rating per userId and its distribution, the plot shows some user have rated a lot but in the same time some user rated quite a few.

```r
edx %>% group_by(userId) %>% summarize(count = n()) %>%
  ggplot(aes(x = count)) + geom_histogram(bins = 300) +
  labs(title = 'Number-of-Rating Distribution by User') +
  xlab('Number of Rating')
```



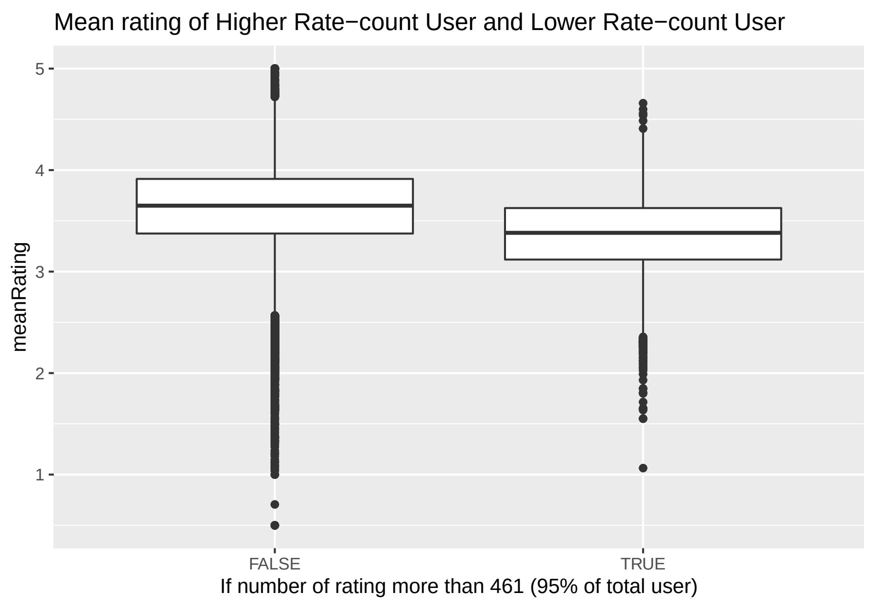
```r
# There are some users that have rated a lot.
```

A deeper analyze is done on mean rating of higher rate-count user and lower rate-count user. Higher rate count user tend to have a lower average rating score. From this effect of difference in number of rating, regularization may need to be done to reduce overfitting of the algorithm.

```r
# Try to see any difference average rating between users with difference numbers of rating
edx %>% group_by(userId) %>% summarize(count = n()) %>%
  summarize(quantile95th = quantile(count,probs =c(0.95)))
```

```
## # A tibble: 1 x 1
##   quantile95th
##          <dbl>
## 1          461
```

```r
edx %>%group_by(userId) %>% summarize(count = n(), meanRating = mean(rating)) %>%
  mutate(Q95 = (count>461)) %>%
  ggplot(aes(y = meanRating, x = Q95)) + geom_boxplot() +
  xlab('If number of rating more than 461 (95% of total user)') +
  labs(title = 'Mean rating of Higher Rate-count User and Lower Rate-count User ')
```



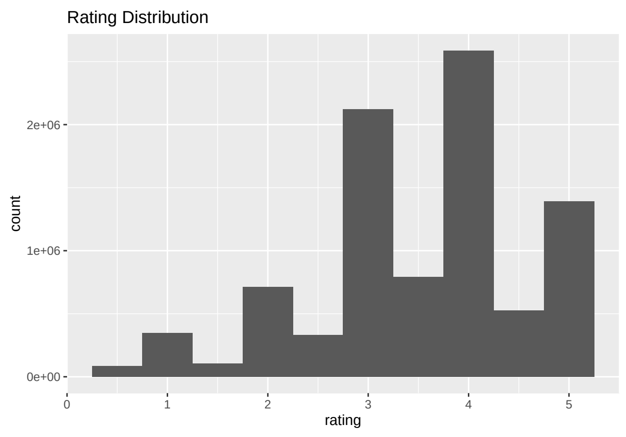
```
# user with more rating-count tends to have a lower average rating score
# Regularization is needed on UserId
```

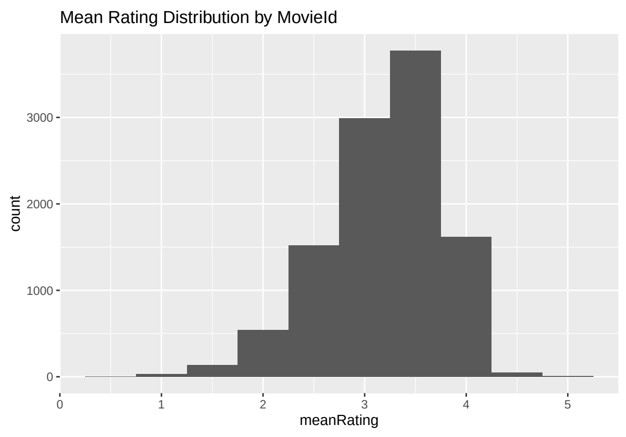
Explore Effect of MovieId

MovieId effect was also analyzed the same way as userId.

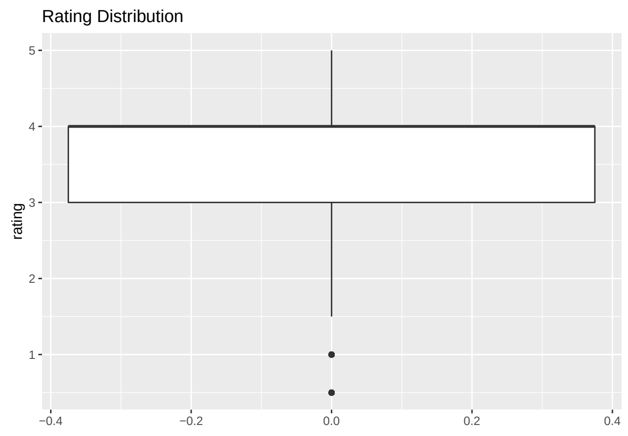
When compared between mean rating and mean rating by movieId, it is clearly shows the significant difference by both histogram and boxplot. This can conclude that movieId have an effect on movie rating and userId feature will be included in final algorithm.

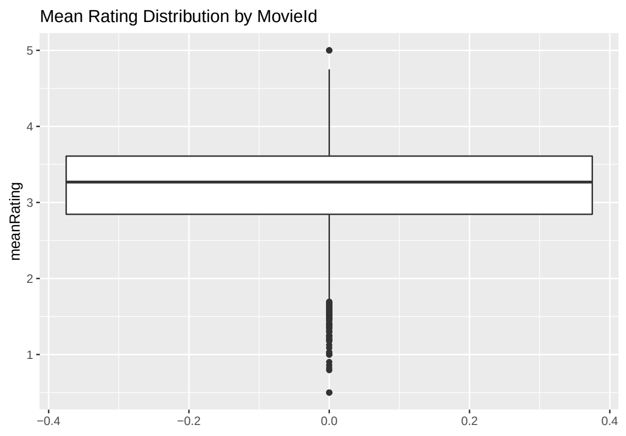
```
# let's see if the movie with more rating have any difference in mean rating or not #
edx %>% ggplot(aes(x = rating)) + geom_histogram(bins = 10) +
  labs(title = 'Rating Distribution')
edx %>% group_by(movieId) %>% summarize(meanRating = mean(rating)) %>%
  ggplot(aes(x = meanRating)) + geom_histogram(bins = 10) +
  labs(title = 'Mean Rating Distribution by MovieId')
```





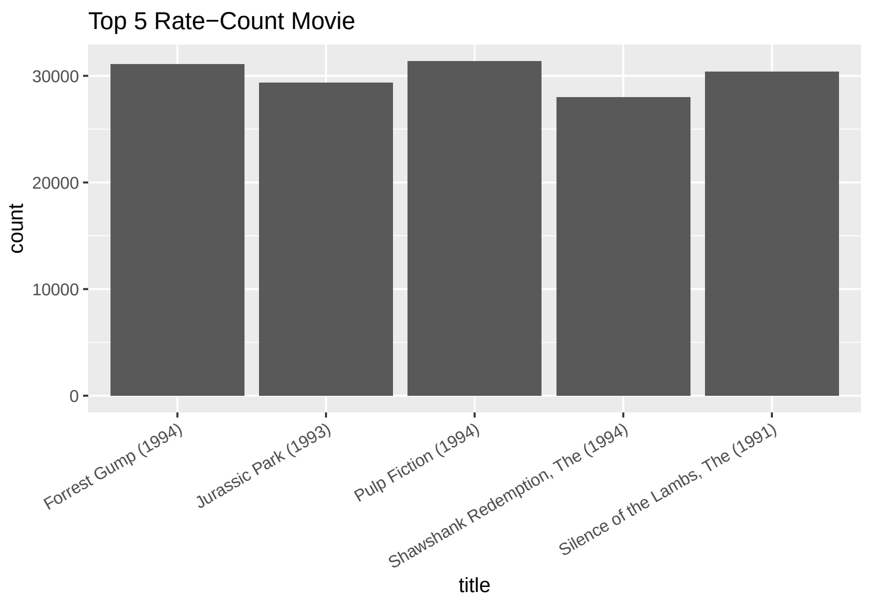
```r
edx %>% ggplot(aes(y = rating)) + geom_boxplot() +
  labs(title = 'Rating Distribution')
edx %>% group_by(movieId) %>% summarize(meanRating = mean(rating)) %>%
  ggplot(aes(y = meanRating)) + geom_boxplot() +
  labs(title = 'Mean Rating Distribution by MovieId')
# This shows bias from movieId
```



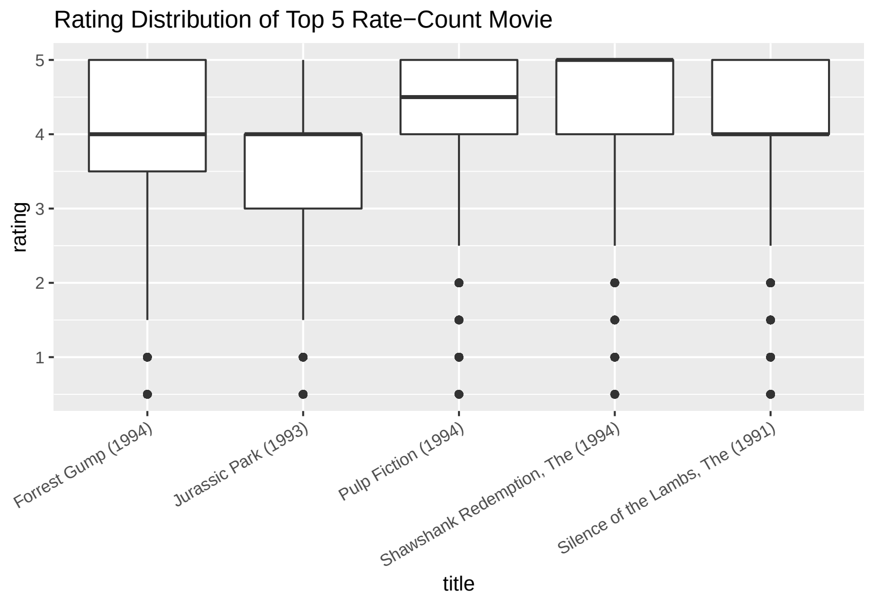


Deep down into top 5 rated movies, we can see clearly a significant difference in rating behavior and mean of rating.

```r
edx %>% group_by(movieId, title) %>% summarize(count = n()) %>%
  arrange(desc(count)) %>% head(n = 5) %>%
  ggplot(aes(x = title, y = count)) + geom_col() +
  scale_x_discrete(guide = guide_axis(angle = 30)) +
  labs(title = 'Top 5 Rate-Count Movie ')
```

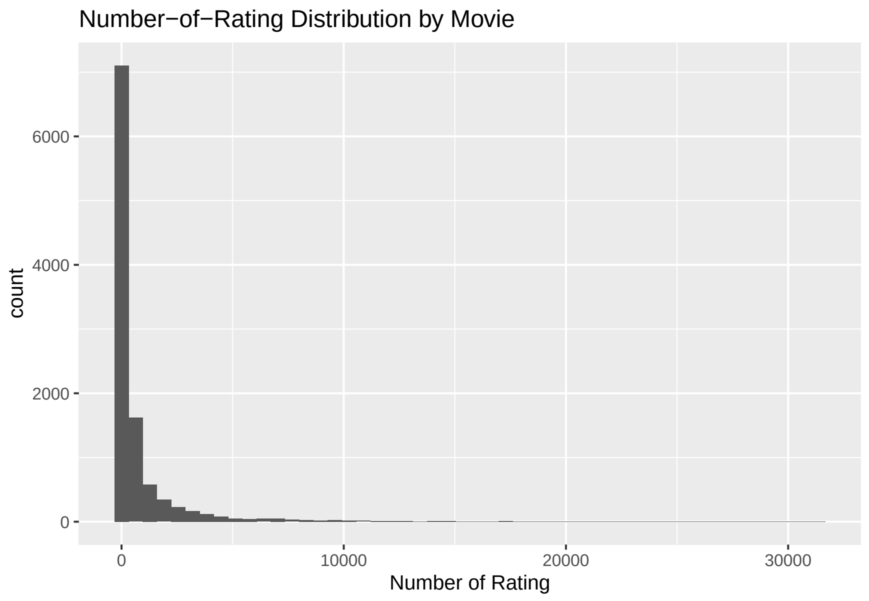


```r
edx %>% filter(movieId %in% c(296,356,593,480,318)) %>%
  ggplot(aes(y = rating, x = title)) + geom_boxplot() +
  scale_x_discrete(guide = guide_axis(angle = 30)) +
  labs(title = 'Rating Distribution of Top 5 Rate-Count Movie ')
```



After plotting Number-of-Rating per MovieId and it distribution, it shows that some movie have been rated a lot but in the same time some movie have been rated quite a few.

```
edx %>% group_by(movieId) %>% summarize(count = n()) %>%
  ggplot(aes(x = count)) + geom_histogram(bins = 50) +
  labs(title = 'Number-of-Rating Distribution by Movie') +
  xlab('Number of Rating')
```



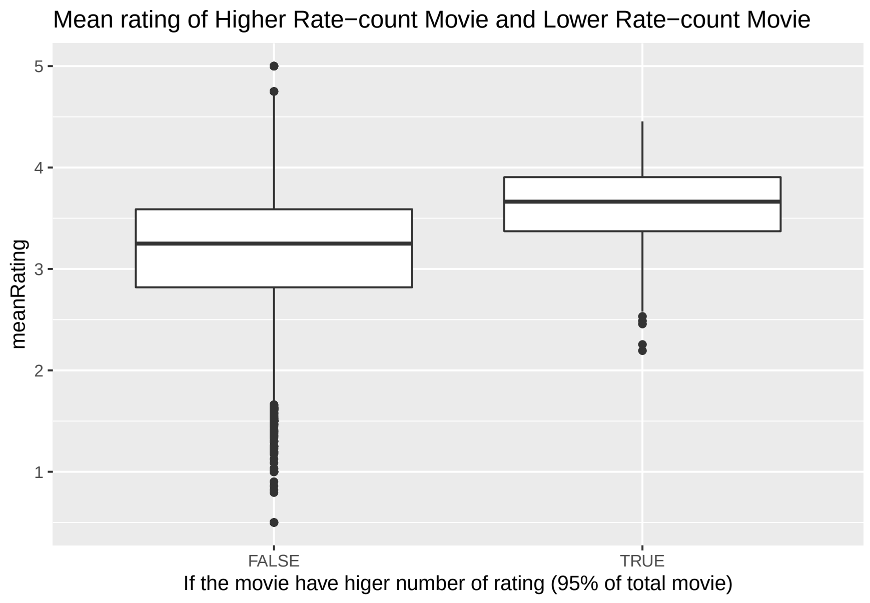
```r
# Some movie have alot more ratings than others
```

A deeper analyze is done on mean rating of higher rate-count movie and lower rate-count movie. Higher rate count movie tend to have a lower average rating score. From this difference in number of rating, regularization may need to be done to reduce overfitting of the algorithm.

```r
# Let's see if there is any difference in number of rating may effect #
edx %>% group_by(movieId) %>% summarize(count = n()) %>%
  summarize(quantile = quantile(count,probs =c(0.95)))
```

```
## # A tibble: 1 x 1
##   quantile
##      <dbl>
## 1    4026.
```

```r
edx %>%group_by(movieId) %>% summarize(count = n(), meanRating = mean(rating)) %>%
  mutate(Q95 = (count>4026)) %>%
  ggplot(aes(y = meanRating, x = Q95)) +
  geom_boxplot() +
  labs(title = 'Mean rating of Higher Rate-count Movie and Lower Rate-count Movie') +
  xlab('If the movie have higer number of rating (95% of total movie)')
```



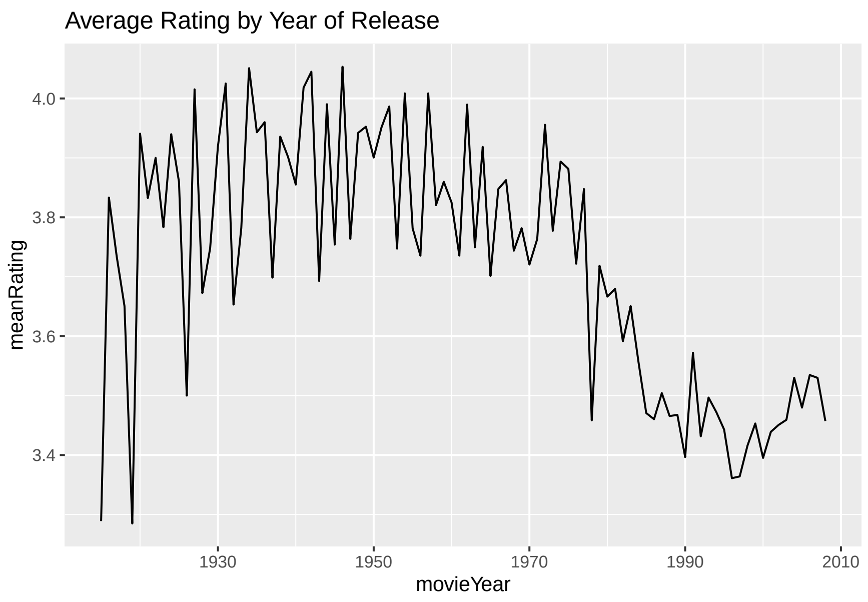
```r
# Movie with more rating tends to have higher average rating
# Regularization may need on movieId
```

Explore Effect of Movie Release Year

Movie release year was analyze to see the behavior and effect to the rating.

When plotting the mean rating by the release year, we can clearly see the difference in mean rating in difference period. The movie release on older year tend to have a higher rating when compare to the movie release on the later year.

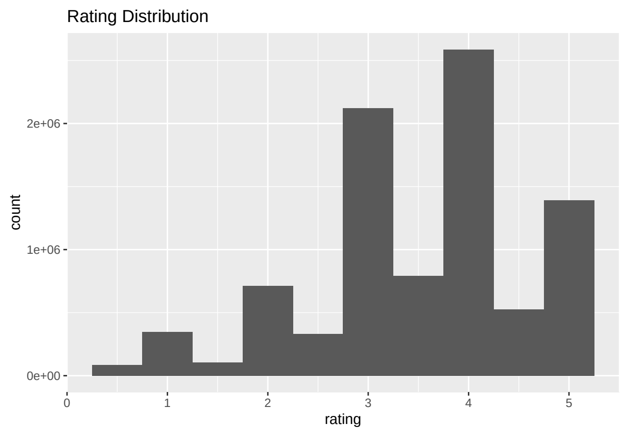
```
edx %>% group_by(movieYear) %>% summarize(meanRating = mean(rating)) %>%
  ggplot(aes(x = movieYear, y = meanRating, group = 1)) + geom_line() +
  labs(title = 'Average Rating by Year of Release')
```

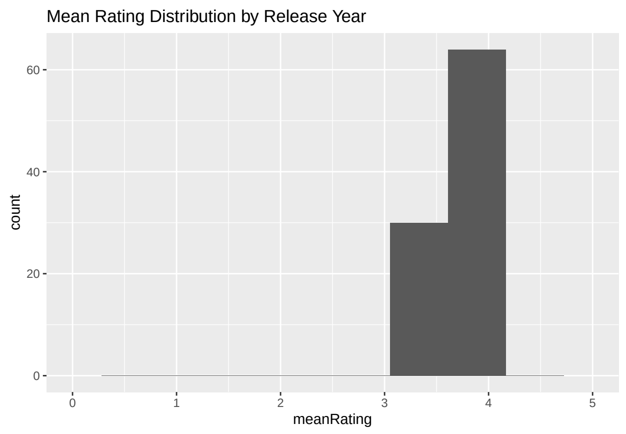


Looking into the distribution by movie release year both histogram and boxplot, it is clearly shows the significant difference in distribution, mean and also median. This can conclude that movieId have an effect on movie rating and userId feature will be included in final algorithm.

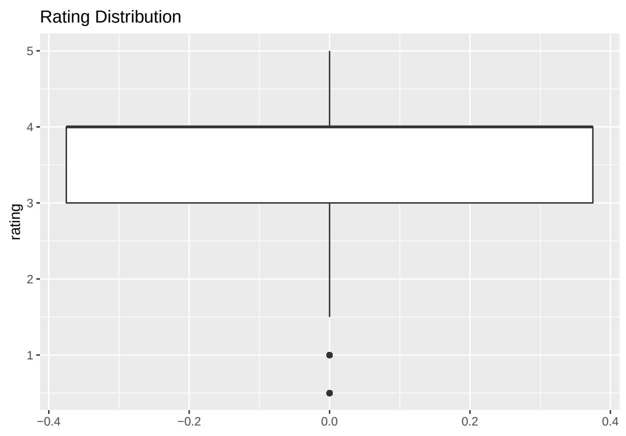
```
edx %>% ggplot(aes(x = rating)) + geom_histogram(bins = 10) +
  labs(title = 'Rating Distribution')
edx %>% group_by(movieYear) %>% summarize(meanRating = mean(rating)) %>%
  ggplot(aes(x = meanRating)) + geom_histogram(bins = 10) +
  scale_x_continuous(limits=c(0,5)) +
  labs(title = 'Mean Rating Distribution by Release Year')
```

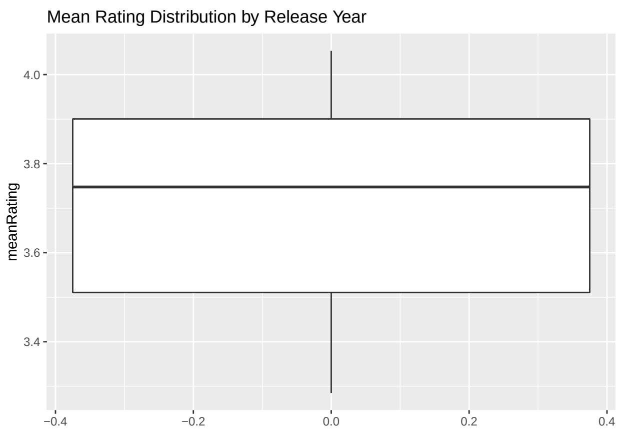
```
## Warning: Removed 2 rows containing missing values (geom_bar).
```





```
edx %>% ggplot(aes(y = rating)) + geom_boxplot() +
  labs(title = 'Rating Distribution')
edx %>% group_by(movieYear) %>% summarize(meanRating = mean(rating)) %>%
  ggplot(aes(y = meanRating)) + geom_boxplot() +
  labs(title = 'Mean Rating Distribution by Release Year')
# the average rating on each release year has a different, this shows bias on movie release year
# the average rating on each release year has no different, this shows no bias on movie release year
```

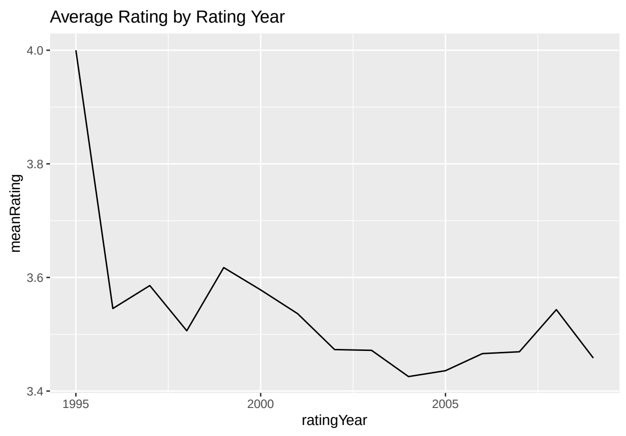


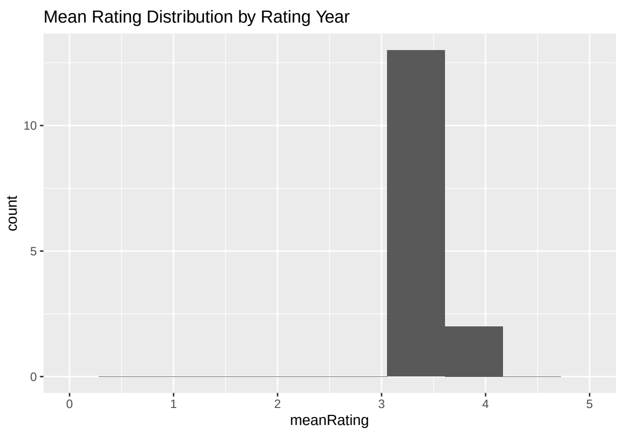


Effect of Movie Rating Year

The same thing was done on movie rating period to see if any time of rating cause a significant to the rating. The year of rating and the month of rating was explore but there is no significant in both rating year and rating month. The difference of rating between each period is very small and the distribution by rating year and month is very limited.

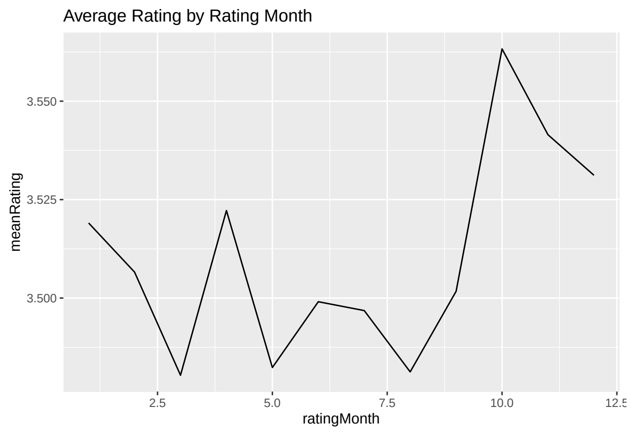
```
edx %>% group_by(ratingYear) %>% summarize(meanRating = mean(rating)) %>%
  ggplot(aes(x = ratingYear, y = meanRating, group = 1)) + geom_line() +
  labs(title = 'Average Rating by Rating Year')

edx %>% group_by(ratingYear)%>% summarize(meanRating = mean(rating)) %>%
  ggplot(aes(x = meanRating)) + geom_histogram(bins = 10) +
  scale_x_continuous(limits=c(0,5)) +
  labs(title = 'Mean Rating Distribution by Rating Year')
```

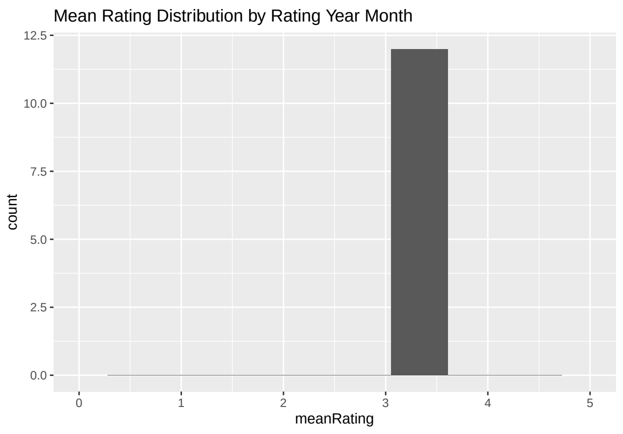




Effect of Movie Rating Month

```r
edx %>% group_by(ratingMonth) %>% summarize(meanRating = mean(rating)) %>%
  ggplot(aes(x = ratingMonth, y = meanRating, group = 1)) + geom_line() +
  labs(title = 'Average Rating by Rating Month')

edx %>% group_by(ratingMonth)%>% summarize(meanRating = mean(rating)) %>%
  ggplot(aes(x = meanRating)) + geom_histogram(bins = 10) +
  scale_x_continuous(limits=c(0,5)) +
  labs(title = 'Mean Rating Distribution by Rating Year Month')
```

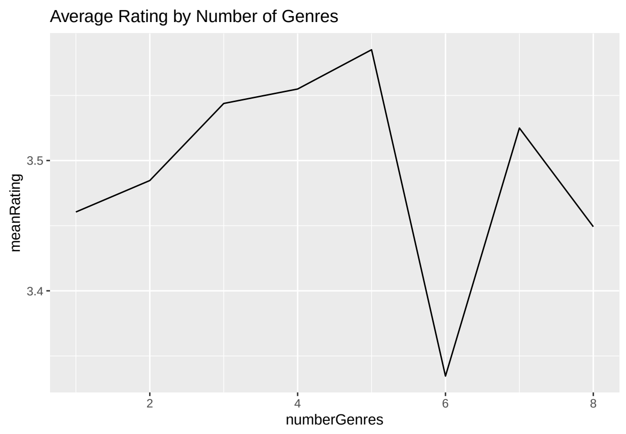


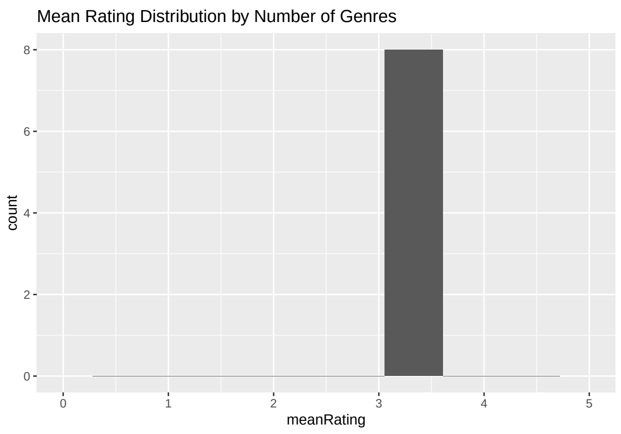


Effect of Number of Genres

The number genres was also explored to see any significant to rating but same behavior as rating period was observed. The difference of rating between the difference number of genres is very small and the distribution by very limited.

```r
edx %>% group_by(numberGenres) %>% summarize(meanRating = mean(rating)) %>%
  ggplot(aes(x = numberGenres, y = meanRating, group = 1)) + geom_line()  +
  labs(title = 'Average Rating by Number of Genres')

edx %>% group_by(numberGenres)%>% summarize(meanRating = mean(rating)) %>%
  ggplot(aes(x = meanRating)) + geom_histogram(bins = 10) +
  scale_x_continuous(limits=c(0,5)) +
  labs(title = 'Mean Rating Distribution by Number of Genres')
```





Modeling approach

Finally after the exploratory data analysis was done, the final approach for this dataset is to consider userId effect, movieId effect and movie release year effect. The movieId effect will be the first bias in the algorithm since it show the largest affect on mean rating follow by userId and movie release year. After all the regularization will be done the reduce the effect of the highly difference in number of rating which will results in overfitting algorithm.

Train-Test Split

Firstly, we split edx dataset to trainset and testset. Trainset wil be use to construct the model then testset will be used to validated the model.

```
set.seed(1, sample.kind="Rounding")
trainIndex <- createDataPartition(edx$rating, times = 1, p = 0.8, list = FALSE)
trainSet <- edx[trainIndex,]
testSet <- edx[-trainIndex,]

testSet <- testSet %>% semi_join(trainSet, by = 'userId') %>%
  semi_join(trainSet, by = 'movieId')
```

Mean Prediction

The first model to be considered is a basic naive model by computing the overall mean without taking into account any other factors that may affect movie rating. In this regression model the average for all movies is used with expected errors that conform to normality assumption.

The initial model can be stated as follows;

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

```
mu_rating = mean(trainSet$rating)

rmse_mu_rating <- RMSE(testSet$rating, mu_rating)
rmse_mu_rating
```

```
## [1] 1.059735
```

```
result <- data_frame(Method = 'Mean', RMSE = rmse_mu_rating)
result %>% knitr::kable()
```

Method	RMSE
Mean	1.059735

The first prediction by mean of total rating results in RMSE 1.0597347. The prediction is not quite good, more feature will be added into consideration to improve model performance.

MovieID Effect

Second model is an updated from the first model where we take into account other factors of how movies are rated. There is some bias in the way users rate movies.

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

```
movie_rating <- trainSet %>% group_by(movieId) %>%
  summarize(b_m = mean(rating - mu_rating))

Pred_RatingMovie <- testSet %>% inner_join(movie_rating, by = 'movieId') %>%
  mutate(Pred = mu_rating + b_m) %>% pull(Pred)

rmseRatingMovie <- RMSE(testSet$rating, Pred_RatingMovie)
rmseRatingMovie
```

```
## [1] 0.943203
```

```
result <- bind_rows(result, data_frame(Method = 'Movie_bias', RMSE = rmseRatingMovie))
result %>% knitr::kable()
```

Method	RMSE
Mean	1.059735
Movie_bias	0.943203

The prediction with movieId bias results in RMSE 0.943203. The prediction performance is improved nearly 10% which is quite a lot. From previous exploratory analysis, more feature is affect to the rating. It is believed that model performance can still be improved. Next, UserId will be take into consideration.

UserID Effect

Third model is a further introduces a penalty term for user effect to capture the possibility of bias as some users opt rating movies highly and this may distort the resulting output.

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

```r
movie_user_rating <- trainSet %>% left_join(movie_rating, by = 'movieId') %>%
  group_by(userId) %>% summarize(b_m_u = mean(rating - mu_rating - b_m))

Pred_RatingMovieUser <- testSet %>% left_join(movie_rating, by = 'movieId') %>%
  left_join(movie_user_rating, by = 'userId') %>%
  mutate(Pred = mu_rating + b_m + b_m_u) %>% pull(Pred)

rmseRatingMovieUser <- RMSE(testSet$rating, Pred_RatingMovieUser)
rmseRatingMovieUser
```

```
## [1] 0.8655254
```

```r
result <- bind_rows(result, data_frame(Method = 'Movie_User_bias',
                                       RMSE = rmseRatingMovieUser))
result %>% knitr::kable()
```

Method	RMSE
Mean	1.0597347
Movie_bias	0.9432030
Movie_User_bias	0.8655254

After add userId bias to the model, the RMSE is reduced to 0.8655254 which is a big improvement. The exploratory analysis is correct. So, last fearture, movie release year will be add to the model and we hope that this will result in the best model performance.

Movie Release Year Effect

Forth model is a introduces a penalty term for year of release effect to capture the possibility of bias as the movie release on older year tends to have a higher rating and this may distort the resulting output.

$$Y_{u,i,m} = \mu + b_i + b_u + b_m + \epsilon_{u,i,m}$$

```r
movie_user_movieYear_rating <- trainSet %>% left_join(movie_rating, by = 'movieId') %>%
  left_join(movie_user_rating, by = 'userId') %>%
  group_by(movieYear) %>% summarize(b_m_u_my = mean(rating - mu_rating - b_m - b_m_u))

Pred_RatingMovieUserMovieYear <- testSet %>% left_join(movie_rating, by = 'movieId') %>%
  left_join(movie_user_rating, by = 'userId') %>%
  left_join(movie_user_movieYear_rating, by = 'movieYear') %>%
  mutate(Pred = mu_rating + b_m + b_m_u + b_m_u_my) %>% pull(Pred)

rmseRatingMovieUserMovieYear <- RMSE(testSet$rating, Pred_RatingMovieUserMovieYear)
rmseRatingMovieUserMovieYear
```

```
## [1] 0.8652171
```

```r
result <- bind_rows(result, data_frame(Method = 'Movie_User_MovieYear_bias',
                                       RMSE = rmseRatingMovieUserMovieYear))
result %>% knitr::kable()
```

Method	RMSE
Mean	1.0597347
Movie_bias	0.9432030
Movie_User_bias	0.8655254
Movie_User_MovieYear_bias	0.8652171

The last feature, movie release year, is reduce RMSE to 0.8652171 which slightly improve the performance of the model. This feature is the last one that will be taken into account. Next step, we will try to do regularization since the analysis shows some imbalance rating in userId and movieId. By regularized userId effect, movieId effect and movie release year effect, this should help improve prediction performance especially for low-rate count date group.

Regularization

Regularization permits us to penalize large estimates that come from small sample sizes. The general idea is to add a penalty for large values of bi, bu and bm to the sum of squares equation that we minimize. So having many large bi, bu or bm makes it harder to minimize.

lambda is a tuning parameter used for the penalty and cross-validation is used to choose it the optimal value.

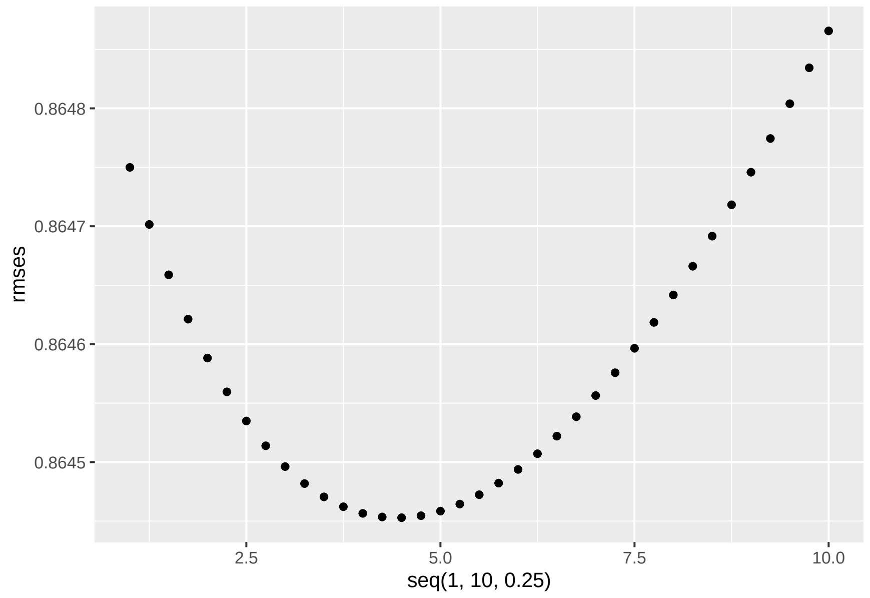
```
lambdas <- seq(1,10,0.25)
```

For each lambda,find b_i, b_u and b_m followed by rating prediction.

```
rmses <- sapply(lambdas, function(l){
  movie_reg_rating <- trainSet %>% group_by(movieId) %>%
    summarize(b_m_reg = sum(rating - mu_rating)/(n() + l))

  movie_user_reg_rating <- trainSet %>%
    left_join(movie_reg_rating, by = 'movieId') %>%
    group_by(userId) %>%
    summarize(b_m_u_reg = sum(rating - mu_rating - b_m_reg)/(n() + l))

  movie_user_movieYear_reg_rating <- trainSet %>%
    left_join(movie_reg_rating, by = 'movieId') %>%
    left_join(movie_user_reg_rating, by = 'userId') %>%
    group_by(movieYear) %>%
    summarize(b_m_u_my_reg = sum(rating - mu_rating - b_m_reg - b_m_u_reg)/(n() + l))

  Pred_RatingMovieUserMovieYearReg <- testSet %>%
    left_join(movie_reg_rating, by = 'movieId') %>%
    left_join(movie_user_reg_rating, by = 'userId') %>%
    left_join(movie_user_movieYear_reg_rating, by = 'movieYear') %>%
    mutate(Pred = mu_rating + b_m_reg + b_m_u_reg + b_m_u_my_reg) %>% pull(Pred)

  Pred_RatingMovieUserMovieYearReg
  Pred_RatingMovieUserMovieYearRegCort <- if_else(Pred_RatingMovieUserMovieYearReg < 0,
                                                  0, Pred_RatingMovieUserMovieYearReg)
  Pred_RatingMovieUserMovieYearRegCort <- if_else(Pred_RatingMovieUserMovieYearRegCort > 5,
                                                  5, Pred_RatingMovieUserMovieYearRegCort)

  rmseRatingMovieUserMovieYearReg <- RMSE(testSet$rating,
                                        Pred_RatingMovieUserMovieYearRegCort)
  rmseRatingMovieUserMovieYearReg
})
```

Plot lambda vs RMSE The result of RMSE by varying lambdas is shown below;

The plot below where rmses against lambdas will help us visualize the spread of rmse as lambda increases from 0 to 10.0.



The lambda that reduce RMSE to the least is selected to the final model.

```
# Minimum RMSE, Regularization Degree
reg[which.min(rmses),]
```

```
## # A tibble: 1 x 3
##      no   seq  RMSE
##   <int> <dbl> <dbl>
## 1    15   4.5 0.864
```

Regularization Prediction The total esults of the movie recommendation models reviewed in this project are published below. But still some correction to be done.

```
l = reg[[which.min(rmses),'seq']]
movie_reg_rating <- trainSet %>% group_by(movieId) %>%
  summarize(b_m_reg = sum(rating - mu_rating)/(n() + l))

movie_user_reg_rating <- trainSet %>%
  left_join(movie_reg_rating, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_m_u_reg = sum(rating - mu_rating - b_m_reg)/(n() + l))
```

```
movie_user_movieYear_reg_rating <- trainSet %>%
  left_join(movie_reg_rating, by = 'movieId') %>%
  left_join(movie_user_reg_rating, by = 'userId') %>%
  group_by(movieYear) %>%
  summarize(b_m_u_my_reg = sum(rating - mu_rating - b_m_reg - b_m_u_reg)/(n() + 1))

Pred_RatingMovieUserMovieYearReg <- testSet %>%
  left_join(movie_reg_rating, by = 'movieId') %>%
  left_join(movie_user_reg_rating, by = 'userId') %>%
  left_join(movie_user_movieYear_reg_rating, by = 'movieYear') %>%
  mutate(Pred = mu_rating + b_m_reg + b_m_u_reg + b_m_u_my_reg) %>% pull(Pred)

rmseRatingMovieUserMovieYearReg <- RMSE(testSet$rating, Pred_RatingMovieUserMovieYearReg)
rmseRatingMovieUserMovieYearReg
```

```
## [1] 0.8645692
```

```
result <- bind_rows(result, data_frame(Method = 'Movie_User_MovieYear_bias_Reg',
                                       RMSE = rmseRatingMovieUserMovieYearReg))
result %>% knitr::kable()
```

Method	RMSE
Mean	1.0597347
Movie_bias	0.9432030
Movie_User_bias	0.8655254
Movie_User_MovieYear_bias	0.8652171
Movie_User_MovieYear_bias_Reg	0.8645692

The result RMSE after regularization is significantly reduced to 0.8645692. The effect from low rate-count data group is reduce so a chance of overfitting is reduce and overall performance is improve. After some correction, this final model will be used to do the prediction on validation set which will shows the real performance of the model.

Correction Rating<0, Rating>5 After all bias and regularization be done, The final predictions still need the correction since some rating prediction is over 5 and some rating prediction is lower than 0 which impossible in reality. The rating prediction over 5 will be corrected to 5 at maximum and rating prediction lower than 0 will be corrected to 0 at minimum.

```
Pred_RatingMovieUserMovieYearRegCort <- if_else(Pred_RatingMovieUserMovieYearReg < 0,
                                                0,Pred_RatingMovieUserMovieYearReg)
Pred_RatingMovieUserMovieYearRegCort <- if_else(Pred_RatingMovieUserMovieYearRegCort > 5,
                                                5, Pred_RatingMovieUserMovieYearRegCort)

rmseRatingMovieUserMovieYearRegCort <- RMSE(testSet$rating,
                                            Pred_RatingMovieUserMovieYearRegCort)
rmseRatingMovieUserMovieYearRegCort
```

```
## [1] 0.8644529
```

```r
result <- bind_rows(result, data_frame(Method = 'Movie_User_MovieYear_bias_Reg_Cort',
                                      RMSE = rmseRatingMovieUserMovieYearRegCort))
result %>% knitr::kable()
```

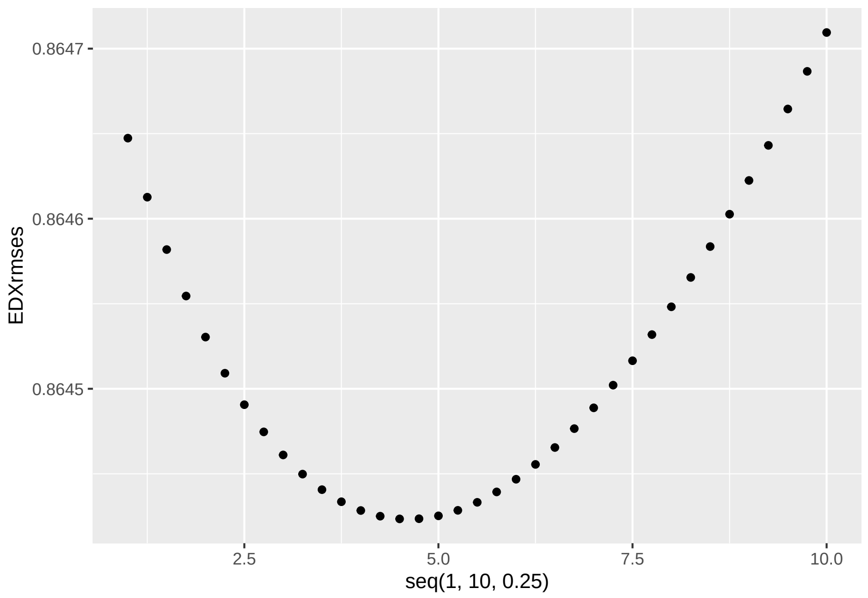
Method	RMSE
Mean	1.0597347
Movie_bias	0.9432030
Movie_User_bias	0.8655254
Movie_User_MovieYear_bias	0.8652171
Movie_User_MovieYear_bias_Reg	0.8645692
Movie_User_MovieYear_bias_Reg_Cort	0.8644529

The correction slightly improve the overall performance to RMSE 0.8644529.

Validation Prediction

Regularization

```r
qplot(seq(1,10,0.25),EDXrmses)
```



```r
# Minimum RMSE, Regularization Degree
EDXreg[which.min(rmses),]
```

```
## # A tibble: 1 x 3
##      no   seq  RMSE
##   <int> <dbl> <dbl>
## 1    15   4.5 0.864
```

Validation Prediction

```r
# Final Model
r = EDXreg[[which.min(rmses),'seq']]

edx_movie_reg_rating <- edx %>%
  group_by(movieId) %>%
  summarize(b_m_reg = sum(rating - mu_rating)/(n() + r))

edx_movie_user_reg_rating <- edx %>%
  left_join(edx_movie_reg_rating, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_m_u_reg = sum(rating - mu_rating - b_m_reg)/(n() + r))

edx_movie_user_movieYear_reg_rating <- edx %>%
  left_join(edx_movie_reg_rating, by = 'movieId') %>%
  left_join(edx_movie_user_reg_rating, by = 'userId') %>%
  group_by(movieYear) %>%
  summarize(b_m_u_my_reg = sum(rating - mu_rating - b_m_reg - b_m_u_reg)/(n() + r))

Pred_EDXRatingMovieUserMovieYearReg <- EDXvalidation %>%
  left_join(edx_movie_reg_rating, by = 'movieId') %>%
  left_join(edx_movie_user_reg_rating, by = 'userId') %>%
  left_join(edx_movie_user_movieYear_reg_rating, by = 'movieYear') %>%
  mutate(Pred = mu_rating + b_m_reg + b_m_u_reg + b_m_u_my_reg) %>% pull(Pred)

rmseEDXRatingMovieUserMovieYearReg <- RMSE(EDXvalidation$rating,
                                           Pred_EDXRatingMovieUserMovieYearReg)
rmseEDXRatingMovieUserMovieYearReg
```

```
## [1] 0.8645244
```

```r
# Correction RMSE<0, RMSE>5
Pred_EDXRatingMovieUserMovieYearRegCort <- if_else(Pred_EDXRatingMovieUserMovieYearReg < 0,
                                                   0,Pred_EDXRatingMovieUserMovieYearReg)
Pred_EDXRatingMovieUserMovieYearRegCort <- if_else(Pred_EDXRatingMovieUserMovieYearRegCort > 5,
                                                   5, Pred_EDXRatingMovieUserMovieYearRegCort)

rmseEDXRatingMovieUserMovieYearRegCort <- RMSE(EDXvalidation$rating,
                                               Pred_EDXRatingMovieUserMovieYearRegCort)
rmseEDXRatingMovieUserMovieYearRegCort
```

```
## [1] 0.8644235
```

The Result

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

Method	RMSE
Mean	1.0597347
Movie_bias	0.9432030
Movie_User_bias	0.8655254
Movie_User_MovieYear_bias	0.8652171
Movie_User_MovieYear_bias_Reg	0.8645692
Movie_User_MovieYear_bias_Reg_Cort	0.8644529
Validation	0.8644235

Based on the above predictions using four different models plus regularization and correction, the model with movie bias, user bias and release year bias with regularization and correction has produced the lowest RMSE 0.8644235 which is the best and the final performance of the model on this report.

This means that to be able to make a movie recommendation based on the MovieLens 10M dataset the regularized model that penalize movie itself with year of release in consideration and users who rated has the potential of making highly accurate recommendation for users rating on movies they had not seen.

Conclusion

This project has construct recommendation system using MovieLens data with different techniques. We still did not explore all possible candidate models that may result in alternative movie recommendation algorithms. There are many approaches to movie recommendation that would have been tested but did not consider in this report. All alternative computations can be improved to even better prediction e.g matrix factorisation, principal components analysis, other classification machine learning algorithm or even more advanced machine learning, deep learning neural network. Those will be left for others to explore.

END.