HarvardX Data Science Capstone Project - Movielens

Toby Wong

16 May 2021

#1. Introduction

This report is part of the capstone project for Harvardx Data Science Professional Certificate.

The objective of the capstone project is to compare the performances of movie rating prediction from different machine learning algorithms and find the best machine learning algorithm model which generates a residual mean squared error (RMSE) score below the target score below:

```
#Target RMSE
target<-0.86490
target
```

```
## [1] 0.8649
```

RMSE is computed by using the formula shown below:

$$RMSE = \sqrt{\frac{1}{N}\sum(y_t - \hat{y}_p)^2}$$

Being:
N = number of samples
$\hat{y}_p$ = predicted value y_t = target value

```
# RMSE formula
RMSE <- function(true_ratings, predicted_ratings){
    sqrt(mean((true_ratings - predicted_ratings)^2))
  }
```

This report contains the following sections:

1. Introduction
2. Movielens Data-set
3. Data Exploration
4. Training set and Test Set
5. Models

6. Result
7. Conclusion

#2. Movielens Data-set

The machine learning algorithms in this report used 10M version of the Movielens data-set (https://grouplens.org/datasets/movielens/10m/) which contains the past rating of movies given by different users. The following code is supplied by the course and it downloads the Movielens data-set and then splits the data-set into two parts:

1. "edx" data-set (90% of the original data-set)
2. "validation" data-set (10% of the original data-set)

```
##########################################################
# 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))
```

```r
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)`
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)
```

#3. Data Exploration

Before building the machine learning algorithm models, "edx" data-set is explored to understand the data it contains and this will also help to determine how the models will be built.

```r
# Summary of "edx" data-set
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   : 789652009
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.: 946768283
##  Median :35738   Median : 1834   Median :4.000   Median :1035493918
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1032615907
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1126750881
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1231131736
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

The "edx" data-set is a data.table, data.frame and it contains 6 columns/variables and 9000055 rows/observations. The columns in "edx" are "userID", "movieID", "rating", "timestamp", "title" and "genres".

```r
# Display the first 5 rows of "edx"
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
```

The information above shown that "userID" and "timestamp" are in integer class, "movieID" and "rating" are in numeric class while "title" and "genres" are in character class. It also shown the data in "genres" column is combinations of genres and "timestamp" will need to be converted into a suitable format in order for it to be useful in the algorithms. In addition, "title" column contains both the movie title as well as the year the movie was released/premiered.

```r
# Number of distinct user in "edx"
n_distinct(edx$userId)
```

```
## [1] 69878
```

```r
# Number of distinct movie in "edx"
n_distinct(edx$movieId)
```

```
## [1] 10677
```

There are 10677 different movies and 69878 different users in "edx".

```r
# Number of distinct genre in "edx"
n_distinct(edx$genres)
```

```
## [1] 797
```

```r
# Check the genres in "edx"
head(edx$genres)
```

```
## [1] "Comedy|Romance"                "Action|Crime|Thriller"
## [3] "Action|Drama|Sci-Fi|Thriller"  "Action|Adventure|Sci-Fi"
## [5] "Action|Adventure|Drama|Sci-Fi" "Children|Comedy|Fantasy"
```

There are in total `n_distinct(edx$genres)` different genres and the high number of different genres is due to there are more than one genre or combination of genres assigned to the each of the movies.

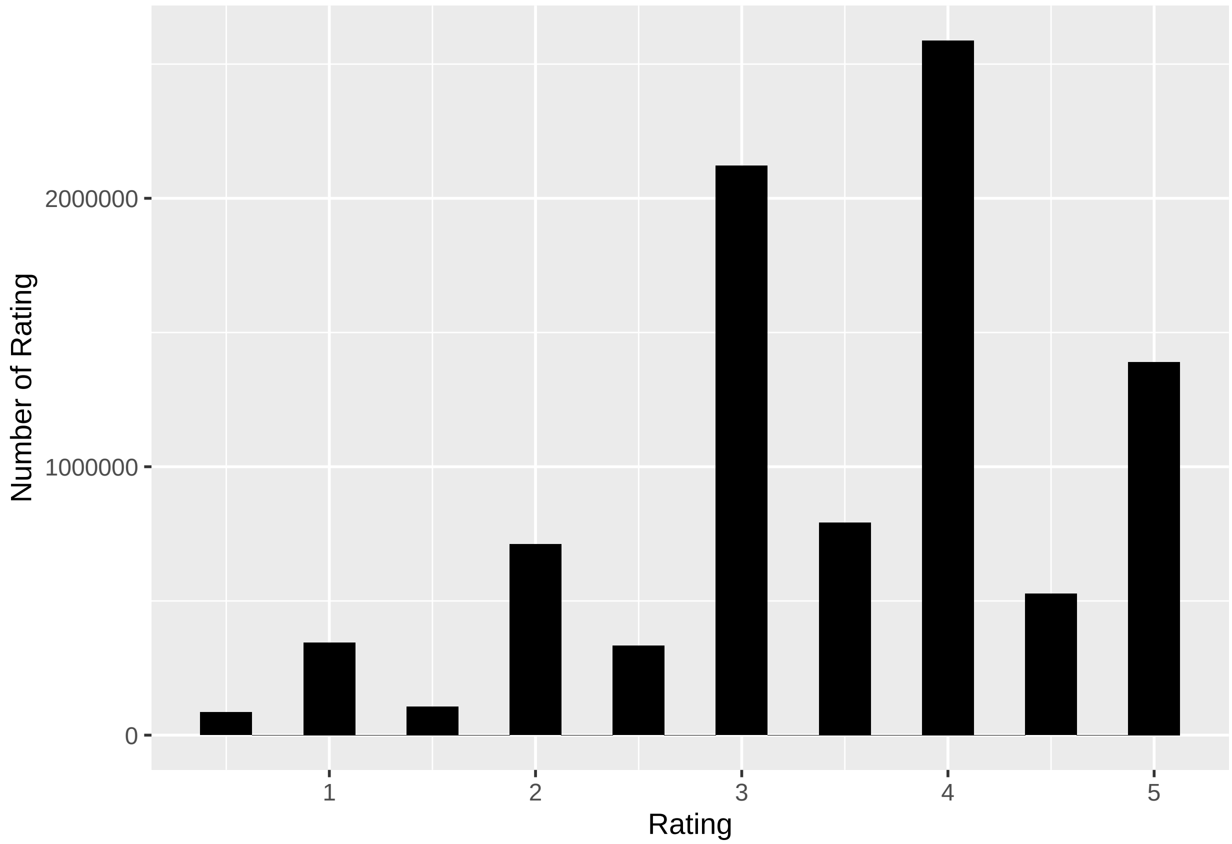
```
# Separate the genres in "edx"
edx_s_g<-edx%>%separate_rows(genres, sep = "\\|")

# Summarised genres in a table
edx_s_g%>%group_by(genres) %>%
  summarize(count = n(), .groups='drop') %>%
  arrange(desc(count))
```

```
## # A tibble: 20 x 2
##    genres               count
##    <chr>                <int>
##  1 Drama              3910127
##  2 Comedy             3540930
##  3 Action             2560545
##  4 Thriller           2325899
##  5 Adventure          1908892
##  6 Romance            1712100
##  7 Sci-Fi             1341183
##  8 Crime              1327715
##  9 Fantasy             925637
## 10 Children            737994
## 11 Horror              691485
## 12 Mystery             568332
## 13 War                 511147
## 14 Animation           467168
## 15 Musical             433080
## 16 Western             189394
## 17 Film-Noir           118541
## 18 Documentary          93066
## 19 IMAX                  8181
## 20 (no genres listed)       7
```

From the table above it shown there are only 20 distinct genres once the genres are separated and the genre rated the most is "Drama".

```
# Plot the total count of each rating
edx %>% ggplot(aes(rating)) +
  geom_histogram(binwidth = 0.25, fill = "black")+
  xlab("Rating")+
  ylab("Number of Rating")
```



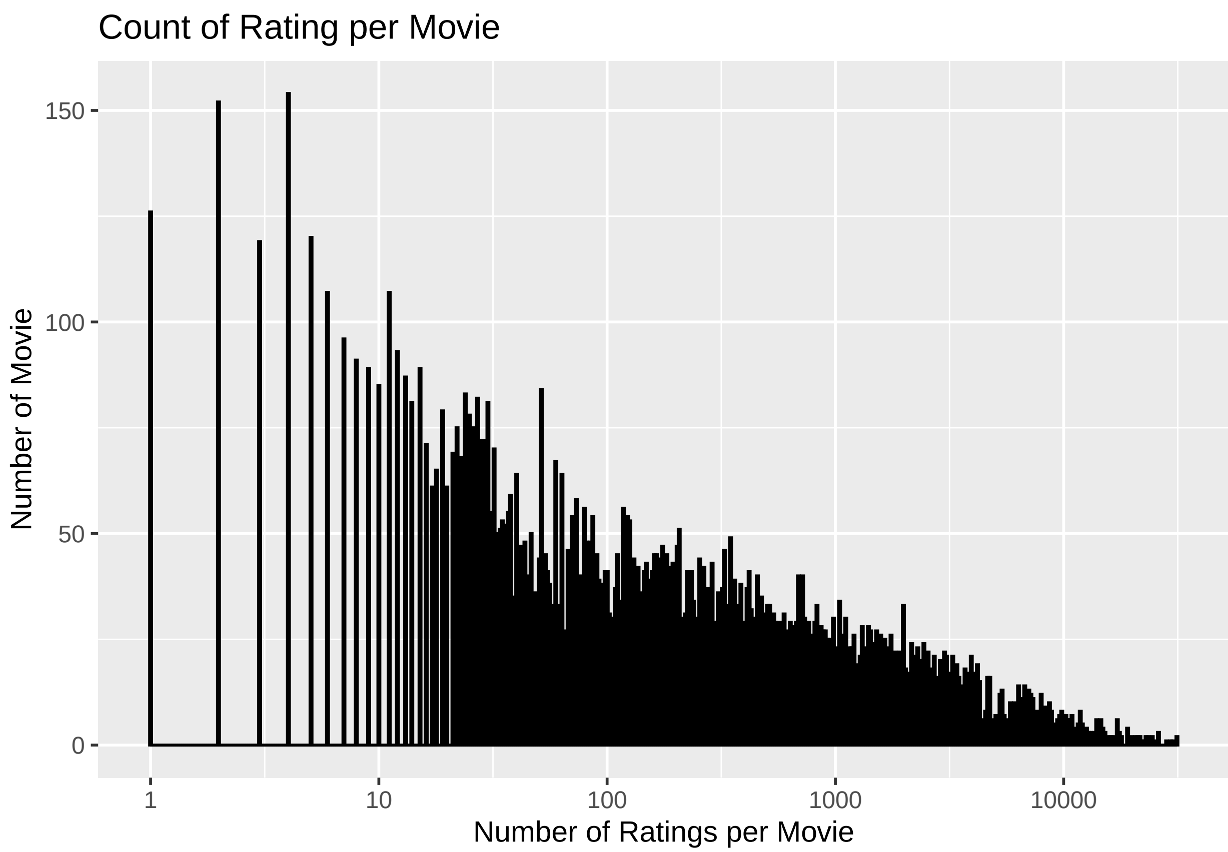
From the plot above, it is observed that the most of the rating given is 4 and in general, half star ratings are less common than whole star ratings.

```r
# Display the top 10 movies with hightest number of rating
edx %>% group_by(movieId, title) %>%
    summarize(count = n(), .groups='drop') %>%
    arrange(desc(count))
```

```
## # A tibble: 10,677 x 3
##    movieId title                                                        count
##      <dbl> <chr>                                                        <int>
##  1     296 Pulp Fiction (1994)                                          31362
##  2     356 Forrest Gump (1994)                                          31079
##  3     593 Silence of the Lambs, The (1991)                             30382
##  4     480 Jurassic Park (1993)                                         29360
##  5     318 Shawshank Redemption, The (1994)                             28015
##  6     110 Braveheart (1995)                                            26212
##  7     457 Fugitive, The (1993)                                         25998
##  8     589 Terminator 2: Judgment Day (1991)                            25984
##  9     260 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977) 25672
## 10     150 Apollo 13 (1995)                                             24284
## # ... with 10,667 more rows
```

The above table shown movie which rated the most is "Pulp Fiction" with 31362 counts.

```
# plot the number of times movies were rated
edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins=500, color = "black") +
  xlab("Number of Ratings per Movie")+
  ylab("Number of Movie")+
  scale_x_log10() +
  ggtitle("Count of Rating per Movie")
```



The above histogram shown there is a wide range in how many times a movie is rated as a few of the movies rated more than 10000 times while around 125 movies rated only one time.

#4. Training set and Test Set Before the modelling of the algorithms, the `edx` data-set is spitted into "training_set" and "test_set". "training_set" is 90% of "edx" while "test_set" is 10% of "edx". "training_set" will be used to build the models and then using "test_set" set to test and generate RMSE scores.

After the models are developed and tested, "edx" and "validation" data-sets will be used to train and then validate the final model(s) to verify if it can achieve a score less than the target RMSE.

```r
####################################
# Generate train_set and test_set #
####################################

test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]

# Make sure userId and movieId in test set are also in train set
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

rm(test_index)
```

#5. Models A series of models using linear regression will be developed step by step by adding and accumulating the effects of the variables in the data-set. After all the effects of the variables are included in the model, regularization will be used in the final model to try to enhance the performance.

##5.1 Model 1 Average of All Ratings

The first model is the simplest one and it calculated the average of the ratings while disregarded all other variables. The resulted RMSE for Model 1 is used as a benchmark for the other models.

```r
##################################
# Model 1 Average of All Ratings #
##################################

mu<-mean(train_set$rating)
model1 <- mu

# RMSE calculation
rmse1 <- RMSE(test_set$rating, model1)
rmse1
```

```
## [1] 1.061135
```

#5.2 Model 2 Average with Movie Effect

The second model is built on top of Model 1 by taking into account the effect of the movies using the average rating of each of the movies (movieID). The result RMSE shown improvement over Model 1.

```r
#####################################
# Model 2 Average with Movie Effect #
#####################################
```

```r
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu), .groups='drop')

# Prediction
model2 <- mu + test_set %>%
  left_join(b_i, by='movieId') %>%
  pull(b_i)

# RMSE calculation
rmse2<-RMSE(test_set$rating, model2)
rmse2
```

```
## [1] 0.9441568
```

##5.3 Model 3 Average with Movie and User Effects

The third model combines average rating, movie effect and user effect in the algorithm by adding the average rating of individual users (userID).

```r
###############################################
# Model 3 Average with Movie and User Effects #
###############################################

b_u <- train_set %>%
  left_join(b_i, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i), .groups='drop')

# Prediction
model3 <- test_set %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

# RMSE calculation
rmse3 <- RMSE(test_set$rating, model3)
rmse3
```

```
## [1] 0.8659736
```

##5.4 Model 4 Average with Movie, User and Genre Effects

Model 4 built on Model 3 by adding the effect of the movie genre. As shown during data exploration, there are `n_distinct(edx$genres)` different combinations of genres and this model includes the effect of the average rating of these combinations to improve the the prediction.

```
#####################################################
# Model 4 Average with Movie, User and Genre Effects #
#####################################################

b_g <- train_set %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u), .groups='drop')

# Prediction
model4 <- test_set %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  left_join(b_g, by="genres") %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)

# RMSE calculation
rmse4 <- RMSE(test_set$rating, model4)
rmse4
```

```
## [1] 0.8656019
```

##5.5 Model 5 Average with Movie, User, Genre and Time Effects

The time when the movie is rated may also affected the rating. Model 5 included the effect of time when the movie is rated in addition to the movie, user and genre effects from Model 4. As noted during data exploration, "timestamp" data needs to be converted to useful format and it is converted into the week when the movie was rated by using "lubridate" package.

```
###########################################################
# Model 5 Average with Movie, User, Genre and Time Effects #
###########################################################

#Load lubridate to convert timestamp to date
if(!require(lubridate)) install.packages("data.table", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: lubridate

##
## 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
```

```r
library(lubridate)

b_t <- train_set %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  left_join(b_g, by='genres') %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week"))%>%
  group_by(date) %>%
  summarize(b_t = mean(rating - mu - b_i - b_u - b_g), .groups='drop')

# Prediction
model5 <- test_set %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  left_join(b_g, by="genres") %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week"))%>%
  left_join(b_t, by='date')%>%
  mutate(pred = mu + b_i + b_u + b_g + b_t) %>%
  pull(pred)

# RMSE calculation
rmse5 <- RMSE(test_set$rating, model5)
rmse5
```

```
## [1] 0.8654975
```

##5.6 Model 6 Average with Movie, User, Genre, Time and Premiere Year Effects

Further to the time when the movie was rated, the year the movie was released/premiered may also contributed to how it was rated. Model 6 included the effect of the premiere year by averaging the rating according to the premiere year.

```r
############################################################################
# Model 6 Average with Movie, User, Genre, Time and premiere Year Effects #
############################################################################

b_y <- train_set %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  left_join(b_g, by='genres') %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week"))%>%
  left_join(b_t, by='date') %>%
  mutate(premiere = as.numeric(str_sub(title,-5,-2))) %>%
```

```
  group_by(premiere) %>%
  summarize(b_y = mean(rating - mu - b_i - b_u - b_g - b_t), .groups='drop')

# Prediction
model6 <- test_set %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  left_join(b_g, by="genres") %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week"))%>%
  left_join(b_t, by='date')%>%
  mutate(premiere = as.numeric(str_sub(title,-5,-2))) %>%
  left_join(b_y, by="premiere")%>%
  mutate(pred = mu + b_i + b_u + b_g + b_t + b_y) %>%
  pull(pred)

# RMSE calculation
rmse6 <- RMSE(test_set$rating, model6)
rmse6
```

```
## [1] 0.8652291
```

##5.7 Model 7 Regularized Movie, User, Genre, Time and Premiere Year Effects

Data exploration shown some of the movies were rated very few times, and the same can also applied to user, genre, time and premiere year. The accuracy of the prediction is affected by these small number of ratings and therefore the performance can be improved by penalizing the data with few ratings through regularization. Model 7 computes the prediction using a penalty term in a regularized model.

```
##########################################################################
# Model 7 Regularized Movie, User, Genre, Time and Premiere Year Effects #
##########################################################################

# Finding the optimum tuning value through cross validation
lambdas <- seq(4, 6, 0.25)

# For each lambda, find b_i, b_u, b_g, b_t and b_y followed by rating prediction
rmses1 <- sapply(lambdas, function(l){

  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l), .groups='drop')

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

```
  b_g <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - b_i - b_u)/(n()+l), .groups='drop')

  b_t <- train_set %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    left_join(b_g, by='genres') %>%
    mutate(date = round_date(as_datetime(timestamp), unit = "week"))%>%
    group_by(date) %>%
    summarize(b_t = mean(rating - mu - b_i - b_u - b_g)/(n()+l), .groups='drop')

  b_y <- train_set %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    left_join(b_g, by='genres') %>%
    mutate(date = round_date(as_datetime(timestamp), unit = "week"))%>%
    left_join(b_t, by='date') %>%
    mutate(premiere = as.numeric(str_sub(title,-5,-2))) %>%
    group_by(premiere) %>%
    summarize(b_y = mean(rating - mu - b_i - b_u - b_g - b_t)/(n()+l), .groups='drop')

  predicted_ratings <- test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    mutate(date = round_date(as_datetime(timestamp), unit = "week"))%>%
    left_join(b_t, by='date')%>%
    mutate(premiere = as.numeric(str_sub(title,-5,-2))) %>%
    left_join(b_y, by="premiere")%>%
    mutate(pred = mu + b_i + b_u + b_g + b_t + b_y) %>%
    pull(pred)

    return(RMSE(test_set$rating, predicted_ratings))
})

lambda<-lambdas[which.min(rmses1)]

# Movie Effect
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), .groups='drop')

# User Effect
b_u <- train_set %>%
```

```r
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda), .groups='drop')

# Genre Effect
b_g <- train_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - b_i - b_u - mu)/(n()+lambda), .groups='drop')

# Time Effect
b_t <- train_set %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  left_join(b_g, by='genres') %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week"))%>%
  group_by(date) %>%
  summarize(b_t = mean(rating - mu - b_i - b_u - b_g)/(n()+lambda), .groups='drop')

# premiere Year Effect
b_y <- train_set %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  left_join(b_g, by='genres') %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week"))%>%
  left_join(b_t, by='date') %>%
  mutate(premiere = as.numeric(str_sub(title,-5,-2))) %>%
  group_by(premiere) %>%
  summarize(b_y = mean(rating - mu - b_i - b_u - b_g - b_t)/(n()+lambda), .groups='drop')

# Prediction
model7 <- test_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week"))%>%
  left_join(b_t, by='date')%>%
  mutate(premiere = as.numeric(str_sub(title,-5,-2))) %>%
  left_join(b_y, by="premiere")%>%
  mutate(pred = mu + b_i + b_u + b_g + b_t + b_y) %>%
  pull(pred)

# RMSE calculation
rmse7 <- RMSE(test_set$rating, model7)
rmse7
```

```
## [1] 0.8651294
```

##5.8 Model 8 Regularized Movie, User, Separated Genre, Time and Premiere Year Effects As observed during data exploration, the genres of the movies are combined in the data-set and this affected the accuracy of the prediction due to over-fitting. Model 8 built from the regularization and effects incorporated in Model 7 while separated the genres to improve the RMSE score.

```
####################################################################################
# Model 8 Regularized Movie, User, Separated Genre, Time and Premiere Year Effects #
####################################################################################

# Separate the genres in "test_set" and "train_set"
test_set_s_g<-test_set%>%separate_rows(genres, sep = "\\|")
train_set_s_g<-train_set%>%separate_rows(genres, sep = "\\|")

# Finding the optimum tuning value through cross validation
lambdas2 <- seq(12, 14, 0.25)

# Calculate the average rating after separarted the genres
mu2 <- mean(train_set_s_g$rating)

# For each lambda, find b_i, b_u, b_g, b_t and b_y followed by rating prediction
rmses2 <- sapply(lambdas2, function(l){

  b_i2 <- train_set_s_g %>%
    group_by(movieId) %>%
    summarize(b_i2 = sum(rating - mu2)/(n()+l), .groups='drop')

  b_u2 <- train_set_s_g %>%
    left_join(b_i2, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u2 = sum(rating - mu2 - b_i2)/(n()+l), .groups='drop')

  b_g2 <- train_set_s_g %>%
    left_join(b_i2, by="movieId") %>%
    left_join(b_u2, by="userId") %>%
    group_by(genres) %>%
    summarize(b_g2 = sum(rating - mu2 - b_i2 - b_u2)/(n()+l), .groups='drop')

  b_t2 <- train_set_s_g %>%
    left_join(b_i2, by='movieId') %>%
    left_join(b_u2, by='userId') %>%
    left_join(b_g2, by='genres') %>%
    mutate(date = round_date(as_datetime(timestamp), unit = "week"))%>%
    group_by(date) %>%
    summarize(b_t2 = mean(rating - mu2 - b_i2 - b_u2 - b_g2)/(n()+l), .groups='drop')

  b_y2 <- train_set_s_g %>%
```

```r
    left_join(b_i2, by='movieId') %>%
    left_join(b_u2, by='userId') %>%
    left_join(b_g2, by='genres') %>%
    mutate(date = round_date(as_datetime(timestamp), unit = "week"))%>%
    left_join(b_t2, by='date') %>%
    mutate(premiere = as.numeric(str_sub(title,-5,-2))) %>%
    group_by(premiere) %>%
    summarize(b_y2 = mean(rating - mu2 - b_i2 - b_u2 - b_g2 - b_t2)/(n()+l), .groups='drop')

  predicted_ratings <- test_set_s_g %>%
    left_join(b_i2, by = "movieId") %>%
    left_join(b_u2, by = "userId") %>%
    left_join(b_g2, by = "genres") %>%
    mutate(date = round_date(as_datetime(timestamp), unit = "week"))%>%
    left_join(b_t2, by='date')%>%
    mutate(premiere = as.numeric(str_sub(title,-5,-2))) %>%
    left_join(b_y2, by="premiere")%>%
    mutate(pred = mu2 + b_i2 + b_u2 + b_g2 + b_t2 + b_y2) %>%
    pull(pred)

    return(RMSE(test_set_s_g$rating, predicted_ratings))
})

lambda2<-lambdas2[which.min(rmses2)]

# Movie Effect
b_i2 <- train_set_s_g %>%
  group_by(movieId) %>%
  summarize(b_i2 = sum(rating - mu2)/(n()+lambda2), .groups='drop')

# User Effect
b_u2 <- train_set_s_g %>%
  left_join(b_i2, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u2 = sum(rating - b_i2 - mu2)/(n()+lambda2), .groups='drop')

# Genre Effect
b_g2 <- train_set_s_g %>%
  left_join(b_i2, by="movieId") %>%
  left_join(b_u2, by="userId") %>%
  group_by(genres) %>%
  summarize(b_g2 = sum(rating - b_i2 - b_u2 - mu2)/(n()+lambda2), .groups='drop')

# Time Effect
b_t2 <- train_set_s_g %>%
  left_join(b_i2, by='movieId') %>%
  left_join(b_u2, by='userId') %>%
```

```
  left_join(b_g2, by='genres') %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week"))%>%
  group_by(date) %>%
  summarize(b_t2 = mean(rating - mu2 - b_i2 - b_u2 - b_g2)/(n()+lambda2), .groups='drop')

# premiere Year Effect
b_y2 <- train_set_s_g %>%
  left_join(b_i2, by='movieId') %>%
  left_join(b_u2, by='userId') %>%
  left_join(b_g2, by='genres') %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week"))%>%
  left_join(b_t2, by='date') %>%
  mutate(premiere = as.numeric(str_sub(title,-5,-2))) %>%
  group_by(premiere) %>%
  summarize(b_y2 = mean(rating - mu2 - b_i2 - b_u2 - b_g2 - b_t2)/(n()+lambda2), .groups='drop

# Prediction
model8 <- test_set_s_g %>%
  left_join(b_i2, by = "movieId") %>%
  left_join(b_u2, by = "userId") %>%
  left_join(b_g2, by = "genres") %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week"))%>%
  left_join(b_t2, by='date')%>%
  mutate(premiere = as.numeric(str_sub(title,-5,-2))) %>%
  left_join(b_y2, by="premiere")%>%
  mutate(pred = mu2 + b_i2 + b_u2 + b_g2 + b_t2 + b_y2) %>%
  pull(pred)

# RMSE calculation
rmse8 <- RMSE(test_set_s_g$rating, model8)
rmse8
```

```
## [1] 0.863828
```

#4. Result ##4.1 Validation Out of the model tested, Model 8 Regularized Movie, User, Separated Genre, Time and Premiere Year produced the best RMSE score of 0.863828 which met the target RMSE score 0.8649. Model 7 and Model 8 were then used to validate the RMSE scores by using `edx` data-set and `validation` data-set.

##4.2 Validation 1-Regularized Movie, User, Genre, Time and premiere Year Effects

```
##############################################################################
# Validation 1-Regularized Movie, User, Genre, Time and premiere Year Effects#
##############################################################################

# Calculate the average of the ratings in "edx" data-set
mu_val <- mean(edx$rating)
```

```r
# Movie Effect
b_i_val <- edx %>%
  group_by(movieId) %>%
  summarize(b_i_val = sum(rating - mu_val)/(n()+lambda), .groups='drop')

# User Effect
b_u_val <- edx %>%
  left_join(b_i_val, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u_val = sum(rating - b_i_val - mu_val)/(n()+lambda), .groups='drop')

# Genre Effect
b_g_val <- edx %>%
  left_join(b_i_val, by="movieId") %>%
  left_join(b_u_val, by="userId") %>%
  group_by(genres) %>%
  summarize(b_g_val = sum(rating - b_i_val - b_u_val - mu_val)/(n()+lambda), .groups='drop')

# Time Effect
b_t_val <- edx %>%
  left_join(b_i_val, by='movieId') %>%
  left_join(b_u_val, by='userId') %>%
  left_join(b_g_val, by='genres') %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week"))%>%
  group_by(date) %>%
  summarize(b_t_val = mean(rating - mu_val - b_i_val - b_u_val - b_g_val)/(n()+lambda), .groups

# premiere Year Effect
b_y_val <- edx %>%
  left_join(b_i_val, by='movieId') %>%
  left_join(b_u_val, by='userId') %>%
  left_join(b_g_val, by='genres') %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week"))%>%
  left_join(b_t_val, by='date') %>%
  mutate(premiere = as.numeric(str_sub(title,-5,-2))) %>%
  group_by(premiere) %>%
  summarize(b_y_val = mean(rating - mu_val - b_i_val - b_u_val - b_g_val - b_t_val)/(n()+lambda

# Prediction
model_val1 <- validation %>%
  left_join(b_i_val, by = "movieId") %>%
  left_join(b_u_val, by = "userId") %>%
  left_join(b_g_val, by = "genres") %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week"))%>%
  left_join(b_t_val, by='date')%>%
  mutate(premiere = as.numeric(str_sub(title,-5,-2))) %>%
  left_join(b_y_val, by="premiere")%>%
```

```
  mutate(pred = mu_val + b_i_val + b_u_val + b_g_val + b_t_val + b_y_val) %>%
  pull(pred)

# RMSE calculation
rmse_val1 <- RMSE(validation$rating, model_val1)
rmse_val1
```

```
## [1] 0.8644543
```

The validation shown Model 7 achieve a better result with "edx" and "validation" data-set and met the target RMSE.

```
##########################################################################################
# Validation 2-Regularized Movie, User, Separated Genre, Time and Premiere Year Effects#
##########################################################################################

# Separate the genres in "validation" data-set
validation_s_g<-validation%>%separate_rows(genres, sep = "\\|")

# Calculate the average rating after separarted the genres
mu_val2 <- mean(edx_s_g$rating)

# Movie Effect
b_i_val2 <- edx_s_g %>%
  group_by(movieId) %>%
  summarize(b_i_val2 = sum(rating - mu_val2)/(n()+lambda2), .groups='drop')

# User Effect
b_u_val2 <- edx_s_g %>%
  left_join(b_i_val2, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u_val2 = sum(rating - b_i_val2 - mu_val2)/(n()+lambda2), .groups='drop')

# Genre Effect
b_g_val2 <- edx_s_g %>%
  left_join(b_i_val2, by="movieId") %>%
  left_join(b_u_val2, by="userId") %>%
  group_by(genres) %>%
  summarize(b_g_val2 = sum(rating - b_i_val2 - b_u_val2 - mu_val2)/(n()+lambda2), .groups='drop

# Time Effect
b_t_val2 <- edx_s_g %>%
  left_join(b_i_val2, by='movieId') %>%
  left_join(b_u_val2, by='userId') %>%
  left_join(b_g_val2, by='genres') %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week"))%>%
  group_by(date) %>%
```

```
  summarize(b_t_val2 = mean(rating - mu_val2 - b_i_val2 - b_u_val2 - b_g_val2)/(n()+lambda2),

# premiere Year Effect
b_y_val2 <- edx_s_g%>%
  left_join(b_i_val2, by='movieId') %>%
  left_join(b_u_val2, by='userId') %>%
  left_join(b_g_val2, by='genres') %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week"))%>%
  left_join(b_t_val2, by='date') %>%
  mutate(premiere = as.numeric(str_sub(title,-5,-2))) %>%
  group_by(premiere) %>%
  summarize(b_y_val2 = mean(rating - mu_val2 - b_i_val2 - b_u_val2 - b_g_val2 - b_t_val2)/(n()+

# Prediction
model_val2 <- validation_s_g %>%
  left_join(b_i_val2, by = "movieId") %>%
  left_join(b_u_val2, by = "userId") %>%
  left_join(b_g_val2, by = "genres") %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week"))%>%
  left_join(b_t_val2, by='date')%>%
  mutate(premiere = as.numeric(str_sub(title,-5,-2))) %>%
  left_join(b_y_val2, by="premiere")%>%
  mutate(pred = mu_val2 + b_i_val2 + b_u_val2 + b_g_val2 + b_t_val2 + b_y_val2) %>%
  pull(pred)

# RMSE calculation
rmse_val2 <- RMSE(validation_s_g$rating, model_val2)
rmse_val2
```

```
## [1] 0.8626743
```

As expected, the of RMSE score of Model 8 produced the best RMSE score of all the models and it also met the target RMSE.

##4.2 Summary of Results

The RMSE scores produced by different models are summarized in the table below.

```
#################
# Summary table #
#################

summary <- tibble(Method = c("Target", "Model 1-Average of All Ratings", "Model 2-Average with
summary
```

```
## # A tibble: 11 x 5
##    Method             RMSE 'Difference to Mo~ 'Difference to L~ 'Difference to ~
```

```
##    <chr>              <dbl>                <dbl>            <dbl>             <dbl>
##  1 Target             0.865               -0.196          0.865             0.865
##  2 Model 1-Average ~ 1.06                  0               0.196             0.196
##  3 Model 2-Average ~ 0.944               -0.117          -0.117             0.0793
##  4 Model 3-Average ~ 0.866               -0.195          -0.0782            0.00107
##  5 Model 4-Average ~ 0.866               -0.196          -0.000372          0.000702
##  6 Model 5-Average ~ 0.865               -0.196          -0.000104          0.000598
##  7 Model 6-Average ~ 0.865               -0.196          -0.000268          0.000329
##  8 Model 7-Regulari~ 0.865               -0.196          -0.0000997         0.000229
##  9 Model 8-Regulari~ 0.864               -0.197          -0.00130          -0.00107
## 10 Validation 1-Reg~ 0.864               -0.197           0.000626         -0.000446
## 11 Validation 2-Reg~ 0.863               -0.198          -0.00178          -0.00223
```

Both Model 7 and Model 8 successfully validated to have met the target RMSE and the following observations were noted:

1. The time of review (timestamp) and the year when the movie was released/premiered had very little impact on the ratings.
2. The genre (genres) in the original combined format also had very little impact on the ratings however once the genres were separated, the impact became quite significant.
3. Movie (movieID), and user (userID) made significant impacts to the ratings.
4. The validation models using "edx" and "validation" data-set provides better results and this may due to the sample size is larger.

#6. Conclusion After going through the models, Model 8 Regularized Movie, User, Separated Genre, Time and Premiere Year Effects model achieved the best RMSE score. The final model is validated by using the "validation" data-set and the resulted a RMSE score of 0.8626743, successfully passed the target RMSE of 0.8649 and met the project objective.

#7. Reference Rafael A. Irizarry (2019), Introduction to Data Science: Data Analysis and Prediction Algorithms with R