Project Movie Lens Report

Le Thuc Anh

May 20, 2022

Contents

1 Packages installation

Requring packages and setting options in R

```
# Require packages
Packages <- c("data.table", "tidyverse", "ggplot2", "caret",
    "Matrix", "hrbrthemes", "viridis", "recosystem")
# Print non-scientific number with maximum 4 signifcant
# digits
options(scipen = 999, digits = 4)
```

2 Introduction

The *Movie Lens* dataset are being generated with the inspiration from the infamous *Netflix challenge* which prized the team who could improve their recommendation system by 10%. This project is trying to apply the process of the Netflix challenge winner on the *Movie Lens* dataset.

3 Data Preparation

The following chunk of code is to create edx set (training) and validation set (test set) from the *Movie Lens* dataset.

```
# Create edx set, validation set (final hold-out test set)

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

dl <- tempfile()
download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip",
    dl)

ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl,
    "ml-10M100K/ratings.dat"))), col.names = c("userId", "movieId",
    "rating", "timestamp"))

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

movies <- as.data.frame(movies) %>%
    mutate(movieId = as.numeric(movieId), title = as.character(title),
        genres = as.character(genres))


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

# Validation set will be 10% of MovieLens data
set.seed(1, 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)
```

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

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

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

Prior to build any models, we need to have a measure of success to evaluate models prediction. **RMSE** is a good measure in this case for recommendation models.

So that, the RMSE function is generated from the RMSE formula to calculate the **RMSE** from models built as the code:

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

4 Data Exploration & Visualization

Firstly, let's exploring the *Movie Lens* dataset provided!

4.1 Overview

The *Movie Lens* dataset:

```
print(edx)
```

```
     userId movieId rating  timestamp                         title                        genres
  1:      1     122    5.0  838985046              Boomerang (1992)                Comedy|Romance
  2:      1     185    5.0  838983525               Net, The (1995)         Action|Crime|Thriller
  3:      1     292    5.0  838983421               Outbreak (1995)  Action|Drama|Sci-Fi|Thriller
  4:      1     316    5.0  838983392               Stargate (1994)       Action|Adventure|Sci-Fi
  5:      1     329    5.0  838983392 Star Trek: Generations (1994) Action|Adventure|Drama|Sci-Fi
 ---
```

9000051: 32620 33140 3.5 1173562747 Down and Derby (2005) Children|Comedy 9000052: 40976 61913 3.0 1227767528 Africa addio (1966) Documentary 9000053: 59269 63141 2.0 1226443318 Rockin' in the Rockies (1945) Comedy|Musical|Western 9000054: 60713 4820 2.0 1119156754 Won't Anybody Listen? (2000) Documentary 9000055: 64621 39429 2.5 1201248182 Confess (2005) Drama|Thriller

The shape of the dataset:

```
dim(edx)
```

```
## [1] 9000055       6
```

Movie Lens dataset has 9000055 rows and 6 columns.

```
sum(is.na(edx))
```

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

There are no NAs values in *Movie Lens* dataset so I could say that the dataset has:
* 9000055 ratings for movies
* 6 columns of related information

In the *Movie Lens* dataset, the distinct number of users and movies are:

```
edx %>%
    summarize(user_n = n_distinct(userId), movie_n = n_distinct(movieId))
```

```
##   user_n movie_n
## 1  69878   10677
```

And, we could see how sparse the *Movie Lens* matrix really is using this code:

Each row represent one user and each column represent one movie.

Each dot is the missing rating that one of the user did not rate one particular movie.

```
edxmatrix = sparseMatrix(i = as.integer(as.factor(edx$userId)),
    j = as.integer(as.factor(edx$movieId)), x = edx$rating)
print(edxmatrix)
```

69878 x 10677 sparse Matrix of class "dgCMatrix"

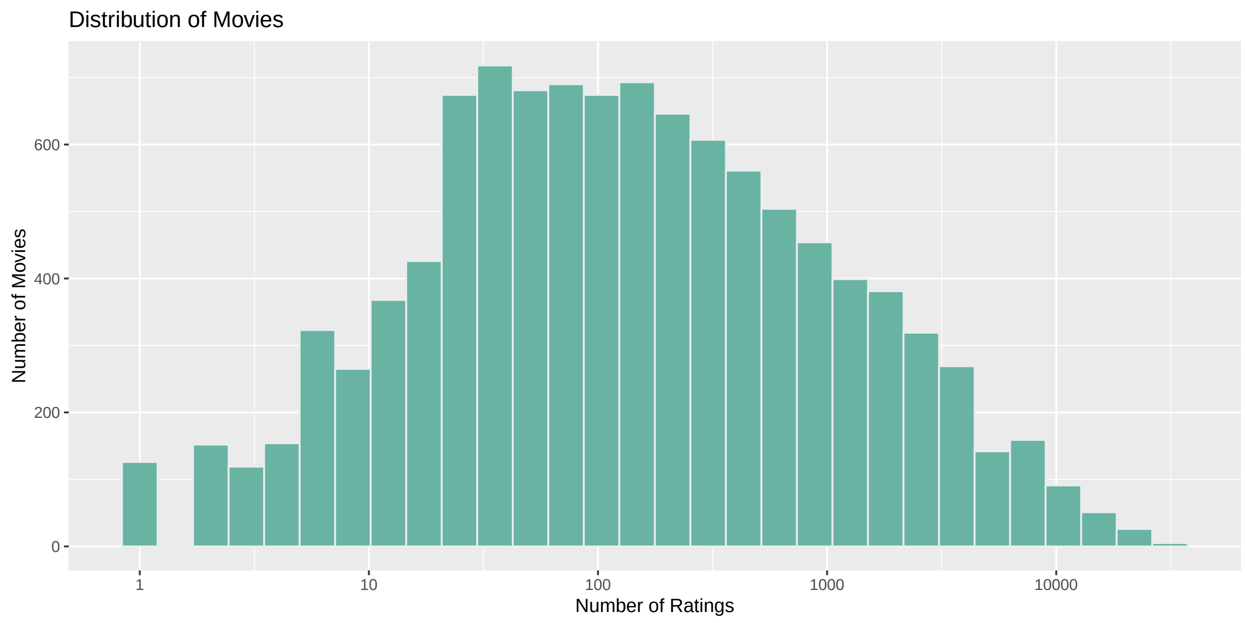
[1,] . . . .
. . . . 5 . . . .
. . . . 5 . . . .
. . . .
. . . . 5 . . . . 5 5
. . . . 5 5 . . . . . 5 . 5 . . . . . 5 . . . . . . 5 . . . .
. . . . 5 . . . .
. . . . . 5 . . . .

....suppressing 10185 columns and 69876 rows in show(); maybe adjust 'options(max.print= , *width* =)' ....

```
[69878,] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3 . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5 . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . 4 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3 . . .
5 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4 . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2 . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .......
```

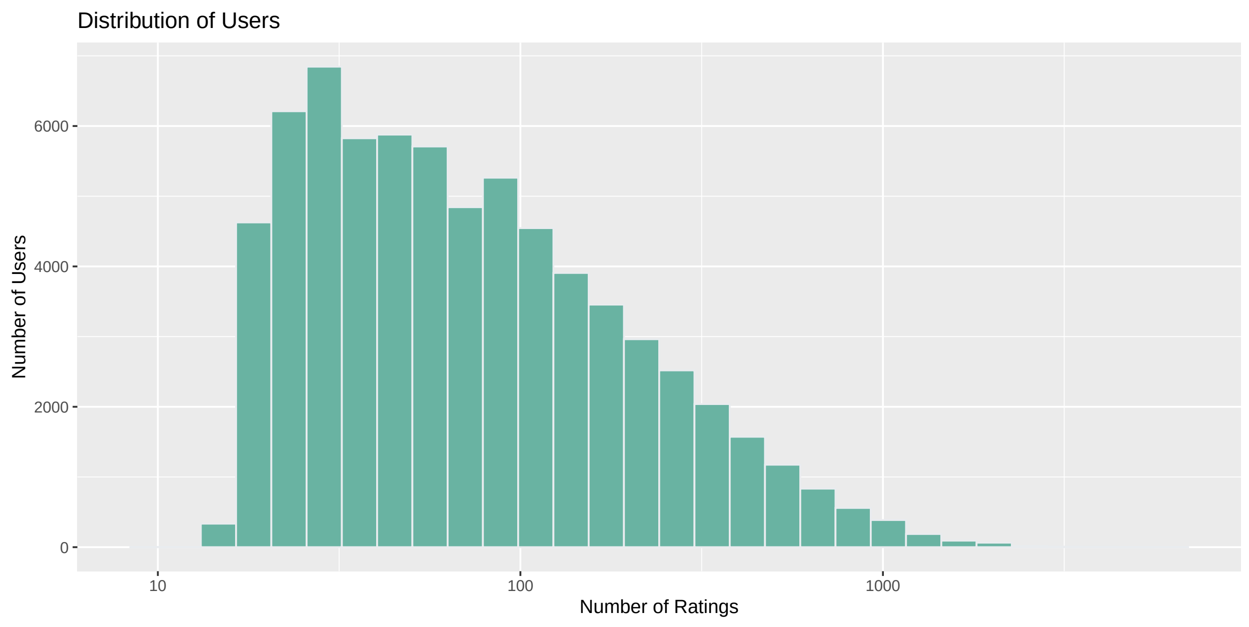
4.2 Visualizations

Number of ratings per movie is visualized using below barplot that has been scaled by log



By observing this barplot, we learned that movies are being rated differently in terms of number of ratings. Some movies are being rated a lot more than others.

Number of ratings per user is visualized using below barplot that has been scaled by log



The same insight also applied for users, users has different number of times they rated and some users are more active than others.

5 Method and Analysis

Based on the Netflix Challenge winner, they used several methods to estimate effects to predict the ratings.

- The movie and user effects
- Regularization
- Matrix Factorization

These method will be explained in details below.

6 The Movie and User Effects

First, let's build the simplest predictive model, the naive model that predicts the same rating for each missing point in the dataset.

6.1 Naive model

```
mu = mean(edx$rating)
mu
```

```
## [1] 3.512
```

The mean of all the rating inside *Movie Lens* dataset are around 3 and a half star.

So that, the RMSE of the *Naive model* is calculated as follow:

```
naive_rmse = RMSE(validation$rating, mu)
naive_rmse
```

```
## [1] 1.061
```

Testing the *Naive model* on the test set gave the **RMSE** of around 1.6

6.2 The movie effects

As learned from the above barplot about the different movie, each movie has different number of ratings.

So that, to add the movie effects to the model, we need to add a buffer to the rating predictions. This buffer will determine the final ratings of each movie compare to the mean of all ratings.

To calculate effects of each Movie, we will calculate the amount of stars the rated rating differ from the overall mean of all movie rating.

And the buffer for each movie is the mean of the amount different for each movie.

```
mu = mean(edx$rating)
buffer_mv = edx[, mean(rating - mu), by = movieId][, .(movieId,
    bf_mv = V1)]
```

So that, rating of movie *i* will be:

Rating_i = mu + buffer_mv_i + error

As this equation is not absolutely right in all circumstances, we will add an error number represent the error amount different from the real ratings. Otherwise, let's determine rating mainly by mu and buffer of each movie.

```
pred_movie_effects = validation[buffer_mv, on = "movieId", `:=`(pred_rating,
    mu + bf_mv)][, pred_rating]
```

```
RMSE(validation$rating, pred_movie_effects)
```

```
## [1] 0.9439
```

After adding the special effects of each movie, we got a better *RMSE* score of 0.94.

6.3 The user effects

Not only each movie has its own effect, each user also has unique effects to the rating, too. As users has different interest in movies and genres.

Now we will add user buffer to the equation, so that rating of a movie i will be: Rating_i = mu + buffer_mv_i + buffer_us_i + error

Similarly, we will calculate the amount of stars the rated rating differ from the overall mean of all user rating.

And the buffer for each movie is the mean of the amount different for each user.

```
mu = mean(edx$rating)
buffer_us = edx[, mean(rating - mu), by = userId][, .(userId,
    bf_us = V1)]
```

```
pred_us_mv_effects = validation[buffer_mv, on = "movieId", `:=`(bf_mv,
    i.bf_mv)][buffer_us, on = "userId", `:=`(bf_us, i.bf_us)][,
    `:=`(pred, mu + bf_mv + bf_us)][, pred]
RMSE(validation$rating, pred_us_mv_effects)
```

```
## [1] 0.885
```

We got the *RMSE* score of 0.885 which is nearer to the target 0.885 of the *Netflix Challenge*

7 Regularization

7.1 Why we need regularization?

Let's dive deep in the predicted ratings we got so far. To check the rightness of our predicted ratings, let's see the extreme mistakes we made.

Here are the top 10 worst mistakes in our movie effect models:

```
head(validation[, `:=`(residual, (rating - (mu + bf_mv)))][order(-abs(residual))][,
    title], 10)
```

[1] "Pokémon Heroes (2003)" "Shawshank Redemption, The (1994)" "Shawshank Redemption, The (1994)" "Shawshank Redemption, The (1994)" "Godfather, The (1972)" "Godfather, The (1972)" "Godfather, The (1972)" "Usual Suspects, The (1995)" "Usual Suspects, The (1995)" "Usual Suspects, The (1995)"

The **"Shawshank Redemption, The (1994)"** has 3111, **"Usual Suspects, The (1995)"** has 2389 ratings and **"Godfather, The (1972)"** has 2067 raitngs. These movies have been rated quite a lot.

First, we need to map movieId with movie titles:

```
movie_titles = unique(edx[, c("movieId", "title")])
```

Let's see the 10 best movies:

```
ten_best = head(buffer_mv[movie_titles, on = "movieId", `:=`(title,
    title)][order(-bf_mv)][, title], 10)
ten_best
```

[1] "Sun Alley (Sonnenallee) (1999)" "Fighting Elegy (Kenka erejii) (1966)" "Satan's Tango (Sátántangó) (1994)" "Shadows of Forgotten Ancestors (1964)" "Hellhounds on My Trail (1999)" "Blue Light, The (Das Blaue Licht) (1932)" "Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)" "Constantine's Sword (2007)" "Human Condition II, The (Ningen no joken II) (1959)" "Human Condition III, The (Ningen no joken III) (1961)"

Let's see the 10 worst movies:

```r
ten_worst = head(buffer_mv[movie_titles, on = "movieId", `:=`(title,
    title)][order(bf_mv)][, title], 10)
ten_worst
```

[1] "Besotted (2001)" "Accused (Anklaget) (2005)" "War of the Worlds 2: The Next Wave (2008)" "Confessions of a Superhero (2007)" "Hi-Line, The (1999)" "SuperBabies: Baby Geniuses 2 (2004)" "Hip Hop Witch, Da (2000)" "Disaster Movie (2008)" "From Justin to Kelly (2003)" "Stacy's Knights (1982)"

Let's see how often they are rated:

```r
edx[movie_titles, on = "movieId", `:=`(title, title)][buffer_mv,
    on = "movieId", `:=`(bf_mv, i.bf_mv)][title %in% ten_best,
    .N, by = c("movieId", "bf_mv")][order(-bf_mv), N]
```

[1] 1 1 2 1 1 1 4 2 4 4

```r
edx[movie_titles, on = "movieId", `:=`(title, title)][buffer_mv,
    on = "movieId", `:=`(bf_mv, i.bf_mv)][title %in% ten_worst,
    .N, by = c("movieId", "bf_mv")][order(bf_mv), N]
```

[1] 2 1 2 1 1 56 14 32 199 1

These movie were rated by very few users. Most of the movie rated only have 1 rating.

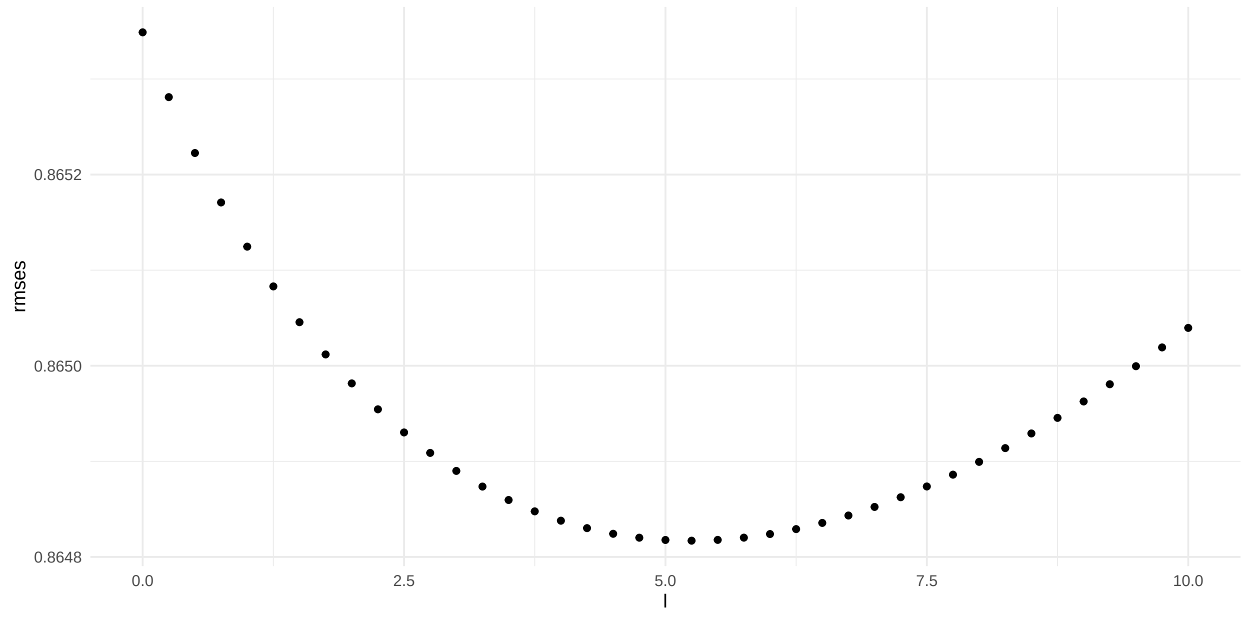
7.2 Penalized least square

The general idea of penalized regression is to control the total variability of the movie effects. Specifically, instead of minimizing the least squares equation, we minimize an equation that adds a penalty: This approach will have our desired effect: when our sample size is very large, a case which will give us a stable estimate, then the penalty. However, when the sample size is small, then the estimate is shrunken towards 0. The larger the *lambda* the more we shrink.

7.3 Choosing penalty term (lambda)

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

fn_lmbd_rmse <- sapply(lambdas, function(l) {
    mu <- mean(edx$rating)
    mv <- edx %>%
        group_by(movieId) %>%
        summarize(b_i = sum(rating - mu)/(n() + l))
    us <- edx %>%
        left_join(mv, by = "movieId") %>%
        group_by(userId) %>%
        summarize(b_u = sum(rating - b_i - mu)/(n() + l))
    predicted_ratings <- validation %>%
        left_join(mv, by = "movieId") %>%
        left_join(us, by = "userId") %>%
        mutate(pred = mu + b_i + b_u) %>%
        pull(pred)
    return(RMSE(predicted_ratings, validation$rating))
})
data.table(l = lambdas, rmses = fn_lmbd_rmse) %>%
    ggplot(aes(x = l, y = rmses)) + geom_point() + theme_minimal()
```



```
lambda = lambdas[which.min(fn_lmbd_rmse)]
print(lambda)
```

```
## [1] 5.25
```

The lambda has the smallest rmse is *5.25*. So that, we will use this lambda to find the final *RMSE* applying on both movie and user effects.

```
mv <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n() + lambda))
us <- edx %>%
    left_join(mv, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n() + lambda))
predicted_ratings_reg <- validation %>%
    left_join(mv, by = "movieId") %>%
    left_join(us, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)
RMSE(predicted_ratings_reg, validation$rating)
```

```
## [1] 0.8648
```

RMSE of *Regularization* is 0.8648. Better than only applying user and movie effects of 0.885.

8 Matrix Factorization

One more step we could try is using *Matrix Factorization*. This method will consider a group of similar movie having similar rating patterns and groups of users having similar rating patterns as well.

Here we will use package recosystem with algorithms to predict the rating using factors analysis.

First, we need to transform the train and test dataset to a real matrix of rating which contains both non-missing and missing values of rating.

```
# transform train data
train_edx <- with(edx, data_memory(user_index = userId, item_index = movieId,
    rating = rating))
# transform test data
test_vali <- with(validation, data_memory(user_index = userId,
    item_index = movieId, rating = rating))
```

And we will train with the train dataset:

```
# create model object
r <- recosystem::Reco()
# training model
r$train(train_edx)
```

```
## iter      tr_rmse          obj
##    0       0.9560   1.4629e+07
##    1       0.8770   1.3352e+07
##    2       0.8533   1.3044e+07
##    3       0.8452   1.2933e+07
##    4       0.8417   1.2885e+07
##    5       0.8389   1.2860e+07
##    6       0.8355   1.2832e+07
##    7       0.8321   1.2810e+07
##    8       0.8291   1.2788e+07
##    9       0.8269   1.2766e+07
##   10       0.8254   1.2755e+07
##   11       0.8243   1.2744e+07
##   12       0.8235   1.2738e+07
##   13       0.8228   1.2732e+07
##   14       0.8222   1.2722e+07
##   15       0.8218   1.2720e+07
##   16       0.8214   1.2717e+07
##   17       0.8212   1.2717e+07
##   18       0.8208   1.2713e+07
##   19       0.8206   1.2710e+07
```

Then, we will test the trained model with validation dataset:

```
# predict with model
y_hat_edx <- r$predict(test_vali, out_memory())
```

```
# RMSE
RMSE(validation$rating, y_hat_edx)
```

```
## [1] 0.8323
```

And our Final RMSE is *0.832*

9 Results

So that our *RMSE* has been improved through techniques used. Let's summary all *RMSE*:

```
##                    method   RMSE pct_improved
## 1:          Naive model 1.0612           NA
## 2:         Movie effect 0.9439       11.053
## 3:          User effect 0.8850        6.237
## 4:       Regularization 0.8648        2.285
## 5: Matrix Factorization 0.8323        3.766
```

The best *RMSE* that we achieved is *0.8322*.

10 Conclusion

This project was using Movie Lens dataset to trying to predict the ratings of movies by users. There were total of 4 methods used: Movie Effects, User Effects, Regularization and Matrix Factorization.

After each method, the *RMSE* improved with the best improvement at around *11%*. The best *RMSE* achieved is *0.8322*.