edX HarvardX PH125.9x
Data Science: Capstone
Movie Recommendation System Report

Chan Kwok Leung

11/28/2020

Contents

FOREWORD

This report is part of the fulfillment for the course HarvardX PH125.9x Data Science: Capstone We are going present a movie recommendation system development with concepts and techniques we learnt from the series of courses in the HarvardX Data Science Professional Certificate program.

The soft copy of this pdf report and related programming code (R and Rmd) is available in GitHub

Here come the url for referenece. https://github.com/klchanhenry/edx-movie-recomm-proj

OVERVIEW

In old economy, an experienced successful salesman can have some sort of understanding a customer base on his appearance, manners and speech. These cue helps making a reasonable guessing for products recommendation. The ability of good recommendation will be one of the major factor drive the sales performance.

Today online business, those cue accessible previously is no longer available. However, the cue can be come from the statistics of those big data collected. By effective analysis the data and machine learning technique, we will able to extract event better cue for recommendation with scientific approach.

There may be hundreds of way on solving a data science problem. However, in real world, cost performance is usually major success factor of a project. We will prioritize on evaluating those approaches which computation cost (computation time and equipment cost) is fair or minimal and feasible to achieve the core technical requirement (RMSE < 0.86490) on this project.

To experience this project, you may only need to have 2 cores 4 Threads processor within 5 years old, 16MBytes ram and Windows 7 or above. One of our target in extra is meeting the project technical target(RMSE < 0.86490) without fancy computer equipment.

GOAL OF THE PROJECT

In this project, we will formulate a movie recommendation system base on user movie preference data provided. To measure the performance of the system, we will RMSE on final validation set less than 0.86490 as our target. RMSE, root-mean-square error, is a common used measurement for the different between the forecast and realistic data. The lower RMSE, in general, usually tell the better accuracy of the system.

With this system, we are going to estimate the rating of a movie. One of the possible commercial usage for this can be using this information as a recommendation reference to our online customers for boosting our hit rate or revenue.

ABOUT THE DATASET

The dataset obtain from grouplens.org.

The url for the dataset is (http://files.grouplens.org/datasets/movielens/ml-10m.zip) According to their information, the dataset contain 10,000,054 ratings and 95,580 tags applied to 10,681 movies by 71,567 users. Users are picked by random and only those rated at least 20 movies will be in the pool.

There are "ratings.dat", "tags.dat" and "movie.dat" from the zip file of the dataset, we are going to combine these files into a single dataframe which contain those information for our analysis.

Here come the metadata of the dataset:-

userId:

User identifier, user are randomly selected from those users rate 20 or more movies. Their ids have been anonymized.

movieId:

Each movie have a unique ID for better data structure purpose.

rating:

From 0.5 Star to 5 Stars which 5 Stars is highest rate. Ratings are made on a 5-star scale, with half-star increments.

timestamp:

Record timestamp represent in seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.

title:

This is the movie title in full name. This was enter manually via the information obtain from IMDB (http://www.imdb.com/) by grouplens.org. They claim errors and inconsistencies may exist.

genres:

It is a pipe-separated list. For example, it can be formatted as “Action|Crime|Thriller”. There are 18 kinds of Genres.

KEY STEPS

We are going to apply a common framework for data science, **OSEMN** framework, which covers major steps on data science project lifecycle.

STEP 1. (O) Obtaining data
In our project, this had been handled by grouplens.org. Raw data is well formatted and ready for obtain from their web site.

STEP 2. (S) Scrubbing data
After obtain data, in some case, the data may have missing values, inconsistence or other errors. We need to perform cleaning such as filling missing values or correct errors before further analysis begin.

STEP 3. (E) Exploring data
Once the clean data is ready, we will start exploring data. By visualization of data, we may find the pattern and possible to extract the features by statistics tools. This give us the detail insight on the set of data and more understand on the problem.

STEP 4. (M) Modelling data
Different machine learning techniques/models will give significantly different RMSE and also the computation cost (calculation time consume and equipment cost) can be a large different. In machine learning, method with higher computation cost do not imply a better RMSE.

To figure out which is the cost effective method to achieve the target RMSE involve experience on data insight and technical knowledge in machine learning.

In this project, we will use the approach similar in commercial world, we will begin with some very basic analysis method and take the result as a baseline, then we will move some other method step by step base on data insight. If the RMSE result keep improving, we will keep on evaluate the similar class of method until we meet the requirement or the improvement comes to flat out. For those computation intensive method, we will keep it as last resort unless the insight point toward that direction.

STEP 5. (N) iNterpreting data
The final step and an important one is about communicate with the project sponsor or project audients. This can be a storytelling process via appropriate level of technical language which is current report trying to deliver.
The audience understanding the development progress step by step can help convincing the support on the result findings. We know getting project sponsor buy-in is usually equally importance as reaching the good RMSE result.

ANALYSIS AND METHODS

Obtaining data

In this chunk of code in R, we are going to load a few library available from r-project.org which can help on futher data manipulation and machine learning. After that, we will download the "ml-10m.zip" dataset from grouplens.org and load the data to different variable to get ready for data scrubbing data.

```r
# load some library for tools on data manipulation
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

# START Code for download direct from grouplens.org
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)
# END Code for download direct from grouplens.org

colnames(movies) <- c("movieId", "title", "genres")
```

Scrubbing data

We are going to combine the data files loaded to variable previously into a dataframe. In order to make the dataframe simplest possible which can help reducing the loading of computer, only those information highly related to our project had been selected to construct the dataframe.

After the dataframe had been construct, we create two partition by random in a ratio of training:validation = 9:1.

We will do our machine learning model development only on the training set (named edx) and sooner we comfortable with our result, we will do a clean testing on the test set (named validation) to evaluate the expected performance on practical prediction scenario.

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


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

# 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)
#
#keep the movies varable for analysis
rm(dl, ratings, test_index, temp, movielens, removed)
```

Exploring data

Before we started, please be noted that we are intentionally NOT to evaluate the validation set of data to in order to provide a more fair and trustworthy result.

Let's have some basic understanding of the training set data on hand. We have 10677 of different movies and 69878 users had provide ratings on movies. We had totally 9000055 records and this simply tell us that not every users gives rating to every movies.

Here come some samples of the records from the data set:

```r
head(edx,5)
```

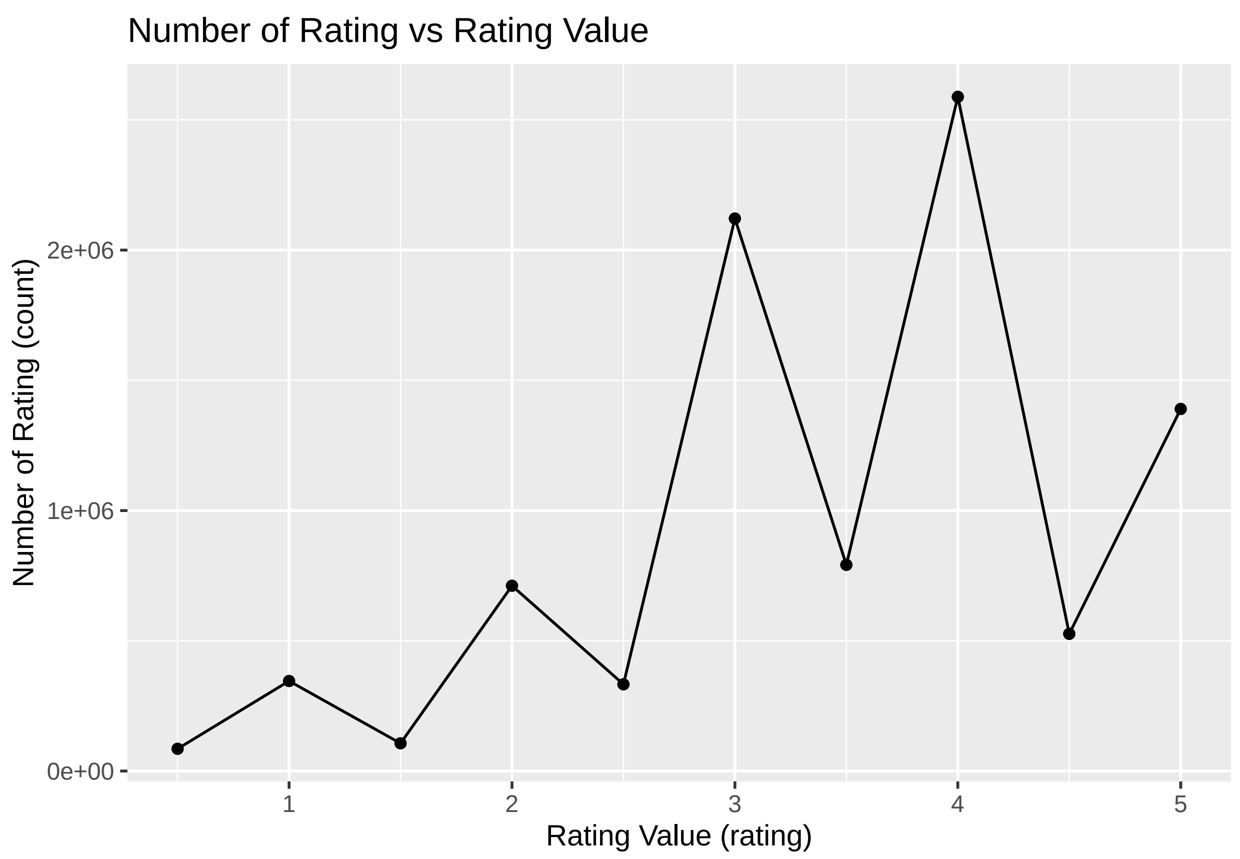
userId	movieId	rating	timestamp	title	genres
1	122	5	838985046	Boomerang (1992)	Comedy|Romance
1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
1	329	5	838983392	Star Trek: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

Every row of record present an individual user rate a movie with related information. There are about 9 millions rows of record in the training data set.

The plot of Number of Rating vs Rating Value may tell that the expected rating seems likely to be at the higher end.

However, it is a normal phenomenon, in general, people watch only those movie they are interested and rate it. This mean that movie rating is from those people having interest on the particular movie upfront. And therefore, the expected result of rating is tends to be higher. An other interesting observation is people tends to rate movie at whole number instead of half number.

```r
# Creating graph for Number of Rating vs Rating Value
edx %>%
  group_by(rating) %>%
  summarize(count = n()) %>%
  ggplot(aes(x = rating, y = count)) +
  labs(y="Number of Rating (count)", x = "Rating Value (rating)") +
  ggtitle("Number of Rating vs Rating Value") +
  geom_point() +
  geom_line()
```



Being noted that a movie can be classify as more than one genres, here come a list showing the number of rated movie under which genre.

```r
# collect the number of different type of genres from the training set and form a table
# The more the movie being rated, the more genres will be counted.
genres = c("Action","Adventure","Animation","Children","Comedy","Crime",
           "Documentary","Drama","Fantasy","Film-Noir","Horror","Musical",
           "Mystery","Romance","Sci-Fi","Thriller","War","Western")
stat_edx_genres <- sapply(genres, function(g) {
  sum(str_detect(edx$genres, g))
})
sort(stat_edx_genres,decreasing = TRUE)
```

```
##       Drama      Comedy      Action    Thriller   Adventure     Romance
##     3910127     3540930     2560545     2325899     1908892     1712100
##      Sci-Fi       Crime     Fantasy    Children      Horror     Mystery
##     1341183     1327715      925637      737994      691485      568332
##         War   Animation     Musical     Western   Film-Noir Documentary
##      511147      467168      433080      189394      118541       93066
```

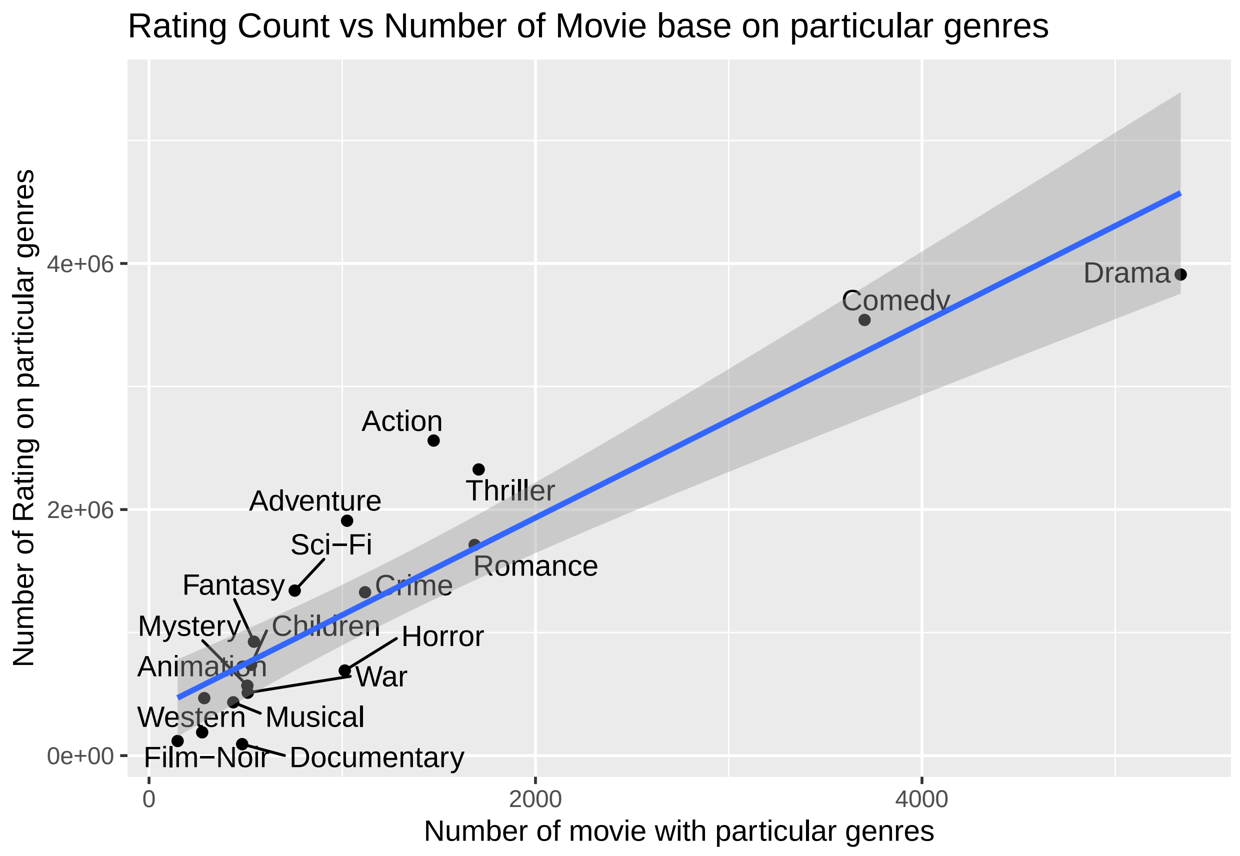
Let's take a look on the movie dataset which tell us distribution of movie in the pool classified with different kind of genres. Also we plot the Rating Count vs Number of Movie on particular genres.

```r
# collect the number of different type of genres from the movie dataframe and form a table
# movie dataframe contain all the available movies for rating and also the genres of each movie
```

```r
stat_movies_genres <- sapply(genres, function(g) {
  sum(str_detect(as.data.frame(movies)$genres, g))
})
sort(stat_movies_genres,decreasing = TRUE)
```

```
##       Drama      Comedy    Thriller     Romance      Action       Crime
##        5339        3703        1706        1685        1473        1118
##   Adventure      Horror      Sci-Fi     Fantasy    Children         War
##        1025        1013         754         543         528         511
##     Mystery Documentary     Musical   Animation     Western   Film-Noir
##         509         482         436         286         275         148
```

```r
# Prepare graph for Rating Count vs Number of Movie base on particular genres
library(ggrepel)
stat_movies_genres <- unname(stat_movies_genres)
stat_edx_genres <- unname(stat_edx_genres)
df <- data.frame(stat_movies_genres,stat_edx_genres,genres)
df %>% ggplot(aes(x=stat_movies_genres,y=stat_edx_genres)) +
  geom_point() +
  geom_text_repel(aes(label=genres)) +
  labs(y="Number of Rating on particular genres", x = "Number of movie with particular genres") +
  ggtitle("Rating Count vs Number of Movie base on particular genres") +
  geom_smooth(method='lm')
```



It lead us a reminder on higher rate count may not imply general public simply like particular genres of movie. Some genres is popular simply because it dominate the market. However, it also can possible turn into an causality dilemma and cannot come to a concrete conclusion which is some sort of common scenario in the middle of of data science process.

Here come some statistic about number of rate perform by each user and number of rate received by each movie:-

```r
# Statistic with focus on user and movie
edx %>% group_by(userId) %>%
  summarise(user_rated_count=n()) %>%
  summary()
```

```
##      userId      user_rated_count
##  Min.   :    1   Min.   :  10.0  
##  1st Qu.:17943   1st Qu.:  32.0  
##  Median :35799   Median :  62.0  
##  Mean   :35782   Mean   : 128.8  
##  3rd Qu.:53620   3rd Qu.: 141.0  
##  Max.   :71567   Max.   :6616.0  
```

```r
edx %>% group_by(movieId) %>%
  summarise(movie_rated_count=n()) %>%
  summary()
```

```
##     movieId      movie_rated_count
##  Min.   :    1   Min.   :    1.0  
##  1st Qu.: 2754   1st Qu.:   30.0  
##  Median : 5434   Median :  122.0  
##  Mean   :13105   Mean   :  842.9  
##  3rd Qu.: 8710   3rd Qu.:  565.0  
##  Max.   :65133   Max.   :31362.0  
```

Base on the figure show above, 75% of users provide rating on movie for at least 32 times and all users at least provide rating for 10 movies. This seems to be normal expect profile.

In regard to the movie, 75% of movies at least be rated for 30 times. 25% of movie had been rated for more than 565 times. To the extreme, there had movie rated over 31000 times and there is movie only rated 1 time. There are always some extreme figures in the real life statistic, as long as the data is trustworthy, we can use statistic technique to regular the raw data into useful information.

Modelling data

To measure the performance of a model, in the project, we use Root Mean Squared Error (RMSE) on the test set as a benchmark. To recap, the definition of the RMSE is as follow:

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

which We define $y_{u,i}$ as the rating for movie i by user u and denote our prediction with $\hat{y}_{u,i}$. N being the number of user/movie combinations and the sum occurring over all these combinations.

The project target is the prediction on the validation set should reach RMSE 0.86490 or lower.

Before we start, let review the data set on hand and clarify once again.

The whole set of data from grouplens.org had been divided randomly into 2 sets in the ratio edx:validation = 9:1.

The validation set is for final benchmarking. The set will not be accessed except final review on performance by RMSE calculation. There will be no any statistic cue leaking directly from this data set.

The edx set is all the data we can fully manipulate and analysis. To prepare for some technique we may use in upcoming analysis, we further split this edx set into train_set:test_set with a ratio 9:1.

The meaning of this test_set is totally different from the validation set above. This test_set is for us to tune our training model. For easy understanding, you can consider this as "internal testing" and the validation set is "real external testing" for performance conclusion.

```r
# Similar to the previous data set splitting, we split once again and named train_set and test_set
# test set will be 10% of Edx data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_set <- edx[-test_index,]
temp <- edx[test_index,]

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

# Add rows removed from test set back into train set
removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)

rm(test_index, temp, removed)
```

Modelling Framework

Let's start with some common model we use in machines learning and then further enhancing this step by step. A rating model that start with assuming same rating for all movies and then adjust with influences from different users and different movies can be present as this:

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

μ is the average of all ratings and $\epsilon_{i,u}$ is the independent errors from the real rating deviated from μ. While choosing the average instead of other arbitrary value is because it possible give a good initial value while using the RMSE as the performance measuring tools. The $\epsilon_{i,u}$ is the variable term which we need keep on exploring and replace with some logical substitute so that the $\epsilon_{i,u}$ will become smaller and this will lead us approaching the target RMSE.

Average This is definitely not likely to obtain good RMSE. However, we are try to record this as a benchmark and see how far we are from the target.

```
# calculating average of the rating from train_set dataframe
mu_hat <- mean(train_set$rating)
mu_hat
```

```
## [1] 3.512456
```

```
# run the RMSE to compare with the test_set(internal testing) to give insight on performance
just_average_rmse <- RMSE(test_set$rating, mu_hat)
just_average_rmse
```

```
## [1] 1.060054
```

```
# store the RMSE result into table for comparison
rmse_results <- tibble(method = "Project Target", RMSE = 0.86490)
rmse_results <- bind_rows(rmse_results,
                          tibble(method = "Just the average", RMSE = just_average_rmse))
rmse_results
```

method	RMSE
Project Target	0.864900
Just the average	1.060054

Modelling movie effect There is good movie and bad one. Therefore the rating will deviate from mean due to quality and general public preference and acceptance. As this is something subjective, we simply named it bias (b_i) in here.

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

Then we try to find the least square to estimate b_i with the following code. However, it turns out technically not feasible.

```
# try direct fitting linear model
fit <- lm(rating ~ as.factor(movieId), data = train_set)

Error: cannot allocate vector of size 644.4 Gb
```

This scenario or similar is something common happening in data science. The result cannot obtain due to computational constrain. Therefore, we need to find an alternative approach.

```
# This create movie bias by averaging rating per movie and minus the average rating of all
# different movies.
mu <- mean(train_set$rating)
```

```r
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# create the prediction by combining average plus per movie bias
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

# run the RMSE to compare with the test_set(internal testing) to give insight on performance
movie_effect_rmse <- RMSE(predicted_ratings, test_set$rating)

# store the RMSE result into table for comparison
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Effect Model",
                                     RMSE = movie_effect_rmse ))
rmse_results
```

method	RMSE
Project Target	0.8649000
Just the average	1.0600537
Movie Effect Model	0.9429615

Modelling movie effect with user effect After we finish modelling movie effect and found good improvement. We knew users do have different preference on movie and With the similar idea, we model the user effect in addition.

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

which b_i and b_u are bias from movie and user respectively.

We try again by using the following code to find the least square to estimate

```r
# try direct fitting linear model
lm(rating ~ as.factor(movieId) + as.factor(userId), data = train_set)

Error: cannot allocate vector of size 36.4 Gb
```

However, as expected, it returns us similar technical difficult.

Therefore we use similar alternative approach to find the b_u and with the b_i result we obtain previously, we the the prediction with both movie and user bias.

```r
# This create user bias by averaging rating per user and minus the average rating of all
# different users on all movies.
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

# create the prediction by combining average plus per user bias and also per movie bias
# obtain previously
```

```r
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

# run the RMSE to compare with the test_set(internal testing) to give insight on performance
movie_and_user_effect_rmse <- RMSE(predicted_ratings, test_set$rating)

# store the RMSE result into table for comparison
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User Effects Model",
                                     RMSE = movie_and_user_effect_rmse ))
rmse_results
```

method	RMSE
Project Target	0.8649000
Just the average	1.0600537
Movie Effect Model	0.9429615
Movie + User Effects Model	0.8646843

The current result is bit promising as it is very near to target. However, to enhance the likelihood of meeting the project target while validating with validation set, we need a estimation method with better performance.

Matrix Factorization First of all, let us revisit the problem we have and get some idea about matrix factorization via the following example.

$$\begin{bmatrix} 1.4 & ? & 1.1 & 0.7 & ? \\ ? & 0.3 & ? & 0.7 & 0.5 \\ 0.4 & 0.3 & ? & ? & 0.3 \\ 1.4 & ? & 1.2 & ? & 0.8 \end{bmatrix}$$

Each number represent a rating from a particular user on a particular item. For example, first user give a rating 1.4 to the first item. However, for second user, we have no idea on his rating to first item which denoted as a "?". In this example, we have 4 different users and 5 different items.
Our objective is to predict/estimate the "?" so that we have idea on our user preference for all available item. Then we can preform our recommendation task base on this rating.

$$\begin{bmatrix} ? & ? \\ ? & ? \\ ? & ? \\ ? & ? \end{bmatrix} \times \begin{bmatrix} ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? \end{bmatrix} = \begin{bmatrix} 1.4 & ? & 1.1 & 0.7 & ? \\ ? & 0.3 & ? & 0.7 & 0.5 \\ 0.4 & 0.3 & ? & ? & 0.3 \\ 1.4 & ? & 1.2 & ? & 0.8 \end{bmatrix}$$

$$U^T \times S = R$$

R is $m \times n$ result matrix, we are going to factorize it into user matrix U (dimension $m \times k$) and item matrix S (dimension $k \times n$). The k here we named Latent Factor. The dimension k is one of our hyper-parameters, which represents the amount of latent factors we are using to estimate the ratings matrix. The value varies on different data set or application and the optimal value is usually obtain by grid search.

The matrix factorization is worth trying, however, it is well known to be a very computational and memory intensive process.

```r
train_set_as_matrix <- train_set %>%
  select(userId, movieId, rating) %>%
  spread(movieId, rating) %>%
  as.matrix()
```

This can be shown by just explore the result set matrix, it is more than 7 hundreds million elements to be manipulated for calculation. To handle this, we are going deploy an library of tools which focus on tackle this problem with parallel computing technique for shared-memory system called LIBMF (A Library for Parallel Matrix Factorization in Shared-memory Systems). This was release at 2016 and it is a BSD-licensed software. In our project, we are going to install the library called "recosystem" which is R wrapper of the LIBMF.

The steps for using recosystem in our implementation as follows:

1. Create a model object (a Reference Class object in R) by calling Reco().
2. Call the $tune() method to select best tuning parameters along a set of candidate values.
3. Train the model by calling the $train() method. A number of parameters can be set inside the function, possibly coming from the result of $tune().
4. Use the $predict() method to compute predicted values.

```r
# install the required recosystem library
if(!require(recosystem))
  install.packages("recosystem")

set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`

# arrange the train and test data according to recosystem required
train_reco <- with(train_set, data_memory(
  user_index = userId, item_index = movieId, rating = rating))
test_reco <- with(test_set, data_memory(
  user_index = userId, item_index = movieId, rating = rating))

# initial the model object
r <- recosystem::Reco()

# tune the parameter base on the training data information, take around 5 - 10 minutes.
opts_tune <- r$tune(train_reco,
                   opts = list(dim      = c(10, 20, 30),
                               costp_l1 = 0,
                               costp_l2 = c(0.01, 0.1),
                               costq_l1 = 0,
                               costq_l2 = c(0.01, 0.1),
                               lrate    = c(0.01, 0.1),
                               nthread  = 8,
                               niter    = 10))

# train the model which take around 5 - 10 minutes.
r$train(train_reco, opts = c(opts_tune$min,
                            niter = 30, nthread = 8))
```

```
## iter      tr_rmse          obj
##    0       0.9792   1.0989e+07
##    1       0.8760   8.9947e+06
##    2       0.8425   8.3476e+06
##    3       0.8199   7.9490e+06
##    4       0.8039   7.6852e+06
##    5       0.7917   7.4966e+06
##    6       0.7817   7.3506e+06
##    7       0.7734   7.2370e+06
##    8       0.7663   7.1470e+06
##    9       0.7599   7.0663e+06
##   10       0.7543   7.0008e+06
##   11       0.7494   6.9462e+06
##   12       0.7448   6.8963e+06
##   13       0.7407   6.8515e+06
##   14       0.7369   6.8133e+06
##   15       0.7335   6.7782e+06
##   16       0.7305   6.7485e+06
##   17       0.7275   6.7196e+06
##   18       0.7249   6.6940e+06
##   19       0.7222   6.6702e+06
##   20       0.7199   6.6498e+06
##   21       0.7178   6.6314e+06
##   22       0.7158   6.6129e+06
##   23       0.7139   6.5957e+06
##   24       0.7120   6.5793e+06
##   25       0.7103   6.5664e+06
##   26       0.7087   6.5526e+06
##   27       0.7074   6.5411e+06
##   28       0.7060   6.5304e+06
##   29       0.7047   6.5200e+06
```

```r
# predict the test set result.
reco_hat <- r$predict(test_reco, out_memory())

# run the RMSE to compare with the test_set(internal testing) to give insight on performance
reco_matrix_factor_rmse <- RMSE(test_set$rating, reco_hat)

# store the RMSE result into table for comparison
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Matrix Factorization using recosystem",
                                     RMSE = reco_matrix_factor_rmse ))
rmse_results
```

method	RMSE
Project Target	0.8649000
Just the average	1.0600537
Movie Effect Model	0.9429615
Movie + User Effects Model	0.8646843
Matrix Factorization using recosystem	0.7845041

It takes around 15 minutes to finish modelling and prediction with the result reaching about RMSE 0.785. This result should be good enough for us moving to examine with the validation set and see if the result is as expected.

RESULTS

We are going to evaluate two of the methods which met our project target and see if they perform well in the validation set. They are "Movie + User Effects Model" and "Matrix Factorization using recosystem".

In validation process, the programming code below is basically the same as above with replacing some of the variable name for better understanding and the data sets "train_set" and "test_set" replace with "edx" and "validation" respectively.

```
#
# This is the final step, we are challenging our model with the validation set
#

# This is Modelling movie effect with user effect

# calculating average of the rating from whole set of training dataframe
mu_edx <- mean(edx$rating)

# calculating the movie bias
movie_avgs_edx <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_edx))

# calculating the user bias
user_avgs_edx <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_edx - b_i))

# create the validation set rating prediction by combining average plus per user
# bias and also per movie bias
predicted_ratings_validation <- validation %>%
  left_join(movie_avgs_edx, by='movieId') %>%
  left_join(user_avgs_edx, by='userId') %>%
  mutate(pred = mu_edx + b_i + b_u) %>%
  pull(pred)

# run the RMSE to compare with the validation set to give insight on performance
movie_and_user_effect_rmse_validation <- RMSE(predicted_ratings_validation, validation$rating)

# store the RMSE result into table for comparison
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User Effects Model(Validation)",
                                     RMSE = movie_and_user_effect_rmse_validation ))
rmse_results
```

method	RMSE
Project Target	0.8649000
Just the average	1.0600537
Movie Effect Model	0.9429615
Movie + User Effects Model	0.8646843
Matrix Factorization using recosystem	0.7845041
Movie + User Effects Model(Validation)	0.8653472

```r
#
# This is the final step, we are challenging our model with the validation set
#

# install the required recosystem library
if(!require(recosystem))
  install.packages("recosystem")

set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`

# arrange the train and validation data according to recosystem required
train_reco_edx <- with(edx, data_memory(
  user_index = userId, item_index = movieId, rating = rating))
test_reco_validation <- with(validation, data_memory(
  user_index = userId, item_index = movieId, rating = rating))

# initial the model object
r_edx <- recosystem::Reco()

# tune the parameter base on the training data information, take around 5 - 10 minutes.
opts_tune_edx <- r_edx$tune(train_reco_edx,
                   opts = list(dim      = c(10, 20, 30),
                               costp_l1 = 0,
                               costp_l2 = c(0.01, 0.1),
                               costq_l1 = 0,
                               costq_l2 = c(0.01, 0.1),
                               lrate    = c(0.01, 0.1),
                               nthread  = 8,
                               niter    = 10))

# train the model which take around 5 - 10 minutes.
r_edx$train(train_reco_edx, opts = c(opts_tune_edx$min,
                              niter = 30, nthread = 8))
```

```
## iter      tr_rmse          obj
##    0       0.9697   1.1968e+07
##    1       0.8737   9.9026e+06
##    2       0.8397   9.1796e+06
##    3       0.8173   8.7555e+06
##    4       0.8011   8.4717e+06
##    5       0.7887   8.2679e+06
##    6       0.7791   8.1177e+06
##    7       0.7709   7.9970e+06
##    8       0.7641   7.9009e+06
##    9       0.7584   7.8220e+06
##   10       0.7533   7.7534e+06
##   11       0.7487   7.6949e+06
##   12       0.7447   7.6470e+06
##   13       0.7411   7.6063e+06
##   14       0.7377   7.5667e+06
##   15       0.7346   7.5326e+06
##   16       0.7319   7.5025e+06
##   17       0.7292   7.4751e+06
##   18       0.7268   7.4489e+06
##   19       0.7247   7.4288e+06
##   20       0.7225   7.4062e+06
##   21       0.7206   7.3881e+06
##   22       0.7188   7.3709e+06
##   23       0.7171   7.3546e+06
##   24       0.7155   7.3401e+06
##   25       0.7140   7.3276e+06
##   26       0.7125   7.3140e+06
##   27       0.7111   7.3005e+06
##   28       0.7100   7.2906e+06
##   29       0.7088   7.2815e+06
```

```
# predict the validation set result by the model.
reco_hat_validation <- r_edx$predict(test_reco_validation, out_memory())

# run the RMSE to compare with the validation set to give insight on performance
reco_matrix_factor_rmse_validation <- RMSE(validation$rating, reco_hat_validation)

# store the RMSE result into table for comparison
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Matrix Factorization using recosystem(Validation)",
                                     RMSE = reco_matrix_factor_rmse_validation ))
rmse_results
```

method	RMSE
Project Target	0.8649000
Just the average	1.0600537
Movie Effect Model	0.9429615
Movie + User Effects Model	0.8646843
Matrix Factorization using recosystem	0.7845041
Movie + User Effects Model(Validation)	0.8653472
Matrix Factorization using recosystem(Validation)	0.7812048

With our project target is RMSE < 0.8649, the method "Matrix Factorization using recosystem" got RMSE $= 0.7812048$ on the final validation data set which is meet and exceed the project target.
We can conclude the project target met and completed.

CONCLUSION

```
rmse_results
```

method	RMSE
Project Target	0.8649000
Just the average	1.0600537
Movie Effect Model	0.9429615
Movie + User Effects Model	0.8646843
Matrix Factorization using recosystem	0.7845041
Movie + User Effects Model(Validation)	0.8653472
Matrix Factorization using recosystem(Validation)	0.7812048

In this project, we use OSEMN framework to cover major steps on data science project lifecycle. After step-by-step evaluation process, we finally successfully fulfill the RMSE target requirement by using the Matrix Factorization implement via LIBMF for modelling.

In some commercial case, decision of the final modelling method may not only focus on RMSE or similar type of performance measurement tools. Computational cost can be another important concern. Matrix Factorization is usually a computation intensive task which narrow the feasible platforms on apply this method in production environment. In some situations, if a rapid respond is needed or limited computation power, we may need to consider other model or refine the current model with different parameter.

MAJOR REFERENCE MATERIALS SOURCE

1. Introduction to Data Science by Rafael A. Irizarry
2. recosystem: Recommender System Using Parallel Matrix Factorization by Yixuan Qiu (https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html)
3. LIBMF: A Matrix-factorization Library for Recommender Systems (https://www.csie.ntu.edu.tw/~cjlin/libmf/)
4. 5 Steps to a Data Science Project Lifecycle (https://thelead.io/data-science/5-steps-to-a-data-science-project-lifecycle)
5. Various technical discussion forums, stackoverflow as an example.