Movies Recommender System - based on Movielens dataset

Mohammed Alhessi

12/29/2021

Contents

1 Introduction

This report presents the whole cycle of building Movie Recommender System based on Movielens dataset. It outlines the data cleaning, processing, analysis, and modeling steps as well as results and conclusion.

The report is part of the capstone project of HarvardX's Data Science Professional Certificate1 program. In this project, we aimed to build Movie recommendation System through different models to reach the target Root Mean Square Error (RMSE) of 0.86490 or less.

The report is structured into 4 chapters: (1) Chapter 1 that introduces the problem at our hand, the dataset, and the accuracy metrics for evaluation, (2) Chapter 2 that discusses the methods used to solve the problem and to reach target RMSE less than 0.86490. This chapter also explores the dataset through descriptive statistics and data visualization so that we could get more insights from the data, (3) Chapter 3 presents and discusses the results of different models and their performance, and finally (4) Chapter 4 the concludes the project and gives recommendations for future work.

1.1 Recommendation Systems

One of the key tools that is used in modern marketing is understanding the customer behavior, special needs and taste, and what he likes and what he dislikes. Modern customers are inundated with huge number of products that is offered by online retailers. Matching the special needs and taste of customers to the appropriate products is a key factor in enhancing the user/customer satisfaction and loyalty.

Therefore, more retailers have become interested in recommender systems, which analyze patterns of user interest in products to provide personalized recommendations that suit a user's taste. Because good personalized recommendations can add another dimension to the user experience, e-commerce leaders like Amazon.com and Netflix have made recommender systems a salient part of their websites(Koren et al., 2009).

In pursuit to improve its movie recommender system, Netflix announced a contest in 2006. The first person/team who can build a new algorithm with performance that exceeds Netflix algorithms by 10%, will win US$ 1 million prize. The performance is measured in terms of RMSE metric; so any decrease by 10% or more in the reference RMSE (recorded by Netflix algorithm), will be considered as winner. In 2009, the grand prize of US US$ 1,000,000 was given to the BellKor's Pragmatic Chaos team which bested Netflix's own algorithm for predicting ratings by 10.06%(Pantelis Monogioudis, 2020).

The winner team employed the matrix factorization using Single Value Decomposition besides regularization technique to improve the performance. For more information on Matrix Factorization within the context of recommender systems, read this article.

1.2 MovieLens Dataset

In this project, we used Movielens dataset to achieve target RMSE of 0.86490 or less. The Netflix data is not publicly available, but the GroupLens research lab generated their own database with millions of ratings for thousands of movies rated by thousands of users. GroupLens Research has collected and made available rating data sets from the MovieLens web site. The data sets were collected over various periods of time, depending on the size of the set.

MovieLens 25M Dataset is Stable benchmark dataset with 25 million ratings on 62,000 movies by 162,000 users[^1]. The latest dataset is 27M Dataset with 27 million ratings on 58,000 movies by 280,000 users[1].

In this project, MovieLens 10M Dataset is used. It is a subset with 10 million ratings on 10,000 movies by 72,000 users. We will explore this dataset in the subsequent sections.

[1] https://grouplens.org/datasets/movielens/

1.3 What is RMSE? and How is calculated?

RMSE stands for Root Mean Square Error. It is a mteric that is used in evaluating performance of different models. The smaller RMSE is, the better performance is the model.
RMSE measures the error/deviance of the predicted rating from the true one and is calculated as follows:
1. Take the difference between the true rating and predicted one,
2. square the difference,
3. take the average of squared differences for all ratings, and
4. Then finally take the square root of the average.
or mathematically, it can be expressed by the following formula:

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

where N is the number of ratings, $y_{u,i}$ is the rating of movie i by user u and $\hat{y}_{u,i}$ is the prediction of movie i by user u.

The goal of this project is to create a recommendation system with target RMSE of 0.8649 or less.

1.4 General Workflow

The workflow we used is, in general, the same as for any data modeling process.

The first step is preparing the dataset by downloading the data files , read and store them in R data frame, then split the dataset into two datasets: the training dataset (90%) and the validation one (10%).

The second step was exploring the data through descriptive statistics and visualization to understand the data patterns, and to check if any cleaning process is needed.

The third step is cleaning the data and preparing it for modeling.

The fourth step is building the recommendation model by starting with the simplest model and iterate until you find the best model that achieve the target RMSE of 0.8649 or less.

Finally, communicate the results through tables, charts, and final report.

2 Methods and Analysis

2.1 Download and Prepare the data

In this section, we download the data and prepare it for next steps. After downloading the data, the dataset is split into two datasets: (1) Traing dataset with 90% of the original data, called `edx`, and (2) Validation dataset with 10% of the orignal dataset, called `validation`. `edx` dataset is used mainly for training the models, and it is further split into train (90% of `edx`) and test datasets (10% of `edx`) for parameters tuning, if necessary. `validation` dataset will be used for selecting the final model, and will be used for evaluation purposes and calculating final RMSE.

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

# Note: this process could take a couple of minutes

# MovieLens 10M dataset:
```

```r
# 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))


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)
```

2.2 Explore and Clean the data

In this section, we will explore the data in more detail to understand the hidden patterns. Visualization and descriptive statistics will help in understanding the interaction between users and movies.

2.2.1 Explore the distribution of the ratings

The total number of ratings in the `edx` data is 9000055 ratings. Here is a sample of 10 ratings:

```
knitr::kable(sample_n(edx,10,replace=TRUE ))
```

userId	movieId	rating	timestamp	title	genres
50805	1095	4.0	952366936	Glengarry Glen Ross (1992)	Drama
42011	6303	3.5	1121891132	Andromeda Strain, The (1971)	Mystery|Sci-Fi
45985	595	4.0	854492184	Beauty and the Beast (1991)	Animation|Children|Fantasy|Musical|Romance
19660	4223	2.0	1207989006	Enemy at the Gates (2001)	Drama|War
46073	356	4.0	1124976754	Forrest Gump (1994)	Comedy|Drama|Romance|War
13268	490	4.0	923662715	Malice (1993)	Thriller
6910	720	0.5	1190501377	Wallace & Gromit: The Best of Aardman Animation (1996)	Adventure|Animation|Comedy
30265	2430	4.0	974963411	Mighty Joe Young (1949)	Adventure|Children|Drama
65174	1080	2.0	1159818290	Monty Python's Life of Brian (1979)	Adventure|Comedy
18227	6249	3.5	1073577266	Goat Trip (2003)	Comedy

As it shown in the table, the `edx` dataset has 6 columns with names: (userId, movieId, rating, timestamp, title, genres).

A summary of the `edx` dataset is presented below:

```
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

To visualize the distribution of the ratings, a bar chart is created as shown below:

```
edx %>% ggplot(aes(x=rating)) +
  geom_bar(fill="#006bbd") +
  geom_text(aes(label=..count..), stat="count", color="white", vjust=1.5) +
  geom_vline(xintercept =mean(edx$rating), color="red" ) +
  geom_vline(xintercept =median(edx$rating), color="blue")
```

The bar chart shows that nearly half of the ratings have scores 4 or larger.

let's have a look at the number of unique users and movies in the `edx` dataset.

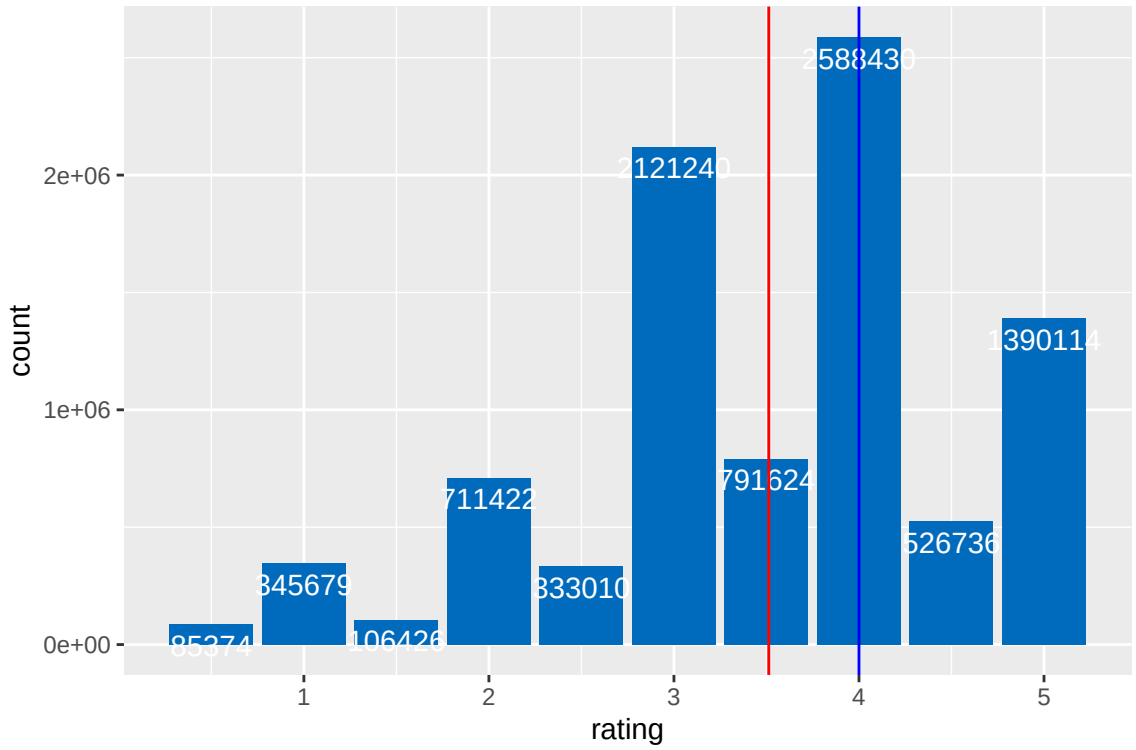
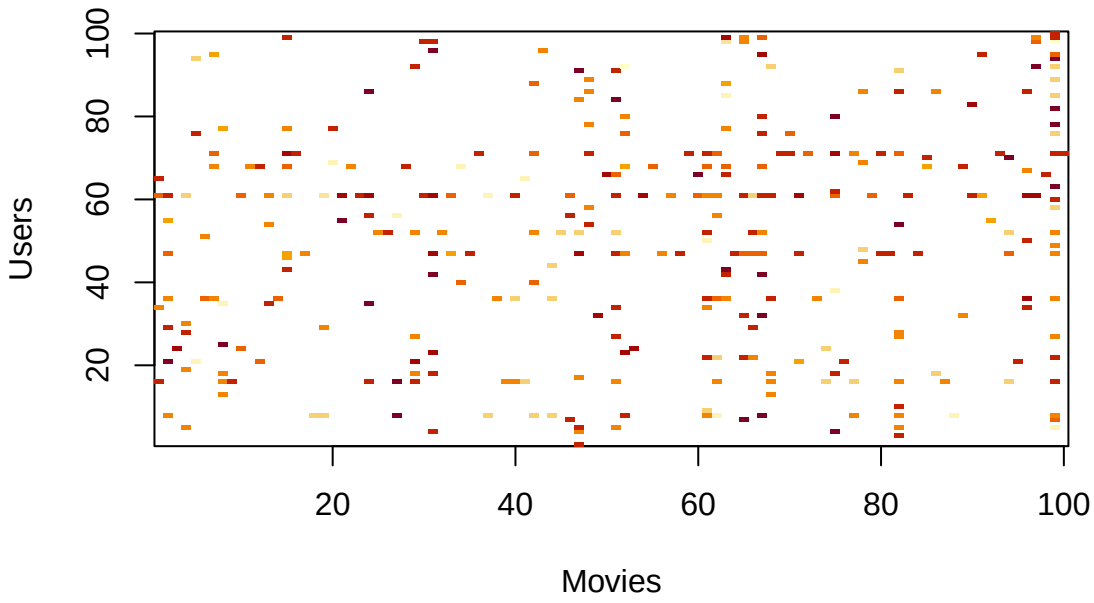


Figure 1: Distribution of ratings

```
# Number of users
nusers <- edx %>% select(userId) %>% n_distinct()

# Number of movies
nmovies <- edx %>% select(movieId) %>% n_distinct()
```

A total of 69878 users rated a total of 10677 movies. If every user rates every movie, then the total number of ratings should be 746087406 ratings, which is much larger than the actual ratings (9000055. That means some users didn't rate some movies. If we rearranged the ratings in a matrix form where the rows represent the users and the columns represent the movies, then we will end up with very sparse matrix that have many missing values. Predicting the missing values is the goal of any recommender system.

```
# Take a sample 100 distinct users with unique ids
sample_users <- sample(unique(edx$userId), 100)

# Create and plot a matrix of 100 different users who rated a sample of
# 100 movies. This is to visualize part of the large sparse matrix.
x<-edx %>% filter(userId %in% sample_users) %>%
  select(userId, movieId, rating) %>%
  spread(movieId, rating) %>%
  select(sample(ncol(.), 100)) %>%
  as.matrix() %>% t(.) %>%
  image(1:100, 1:100,. , xlab="Movies", ylab="Users")
```

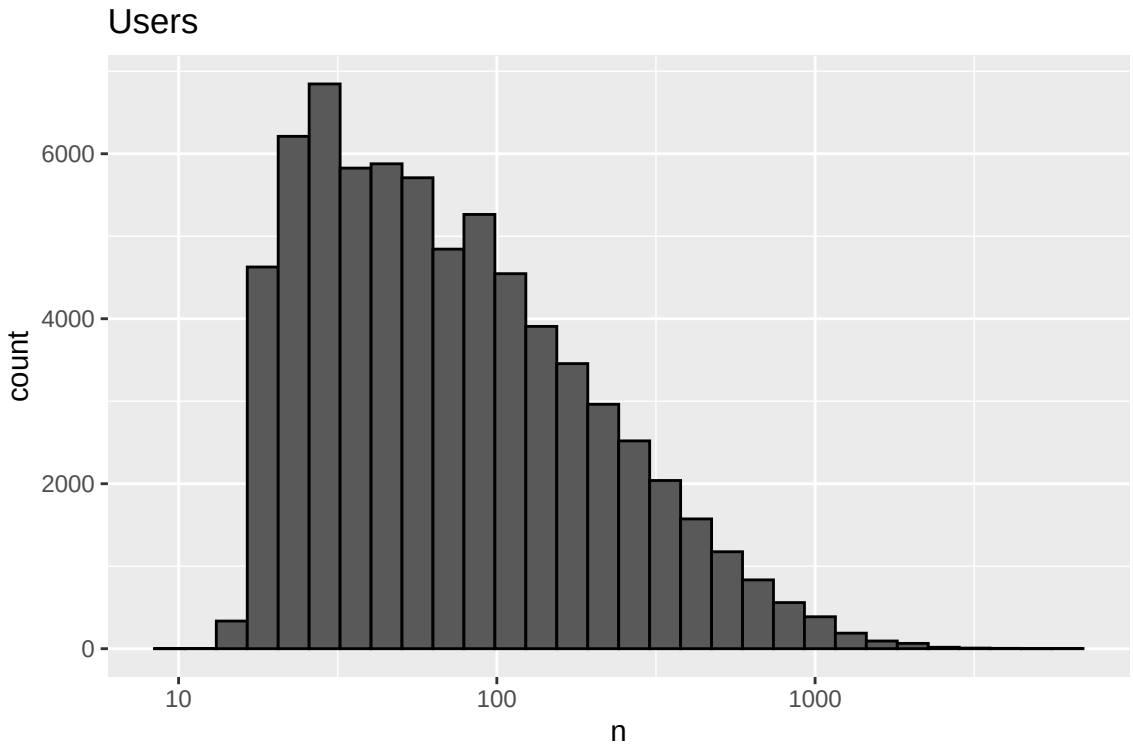


As you see, the matrix has a large number of white cells, which means missing values. For example, user 10 has rated only few movies of the sampled list of 100 movies.

2.2.2 Explore the distribution/ variability of the users

It is good to understand the variability of the users in terms of ratings. For each user, the total number of ratings of all movies is calculated and then a histogram is built, which will convey the variability among the users.

```
edx %>%
  group_by(userId) %>%
  summarize(n=n()) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Users")
```

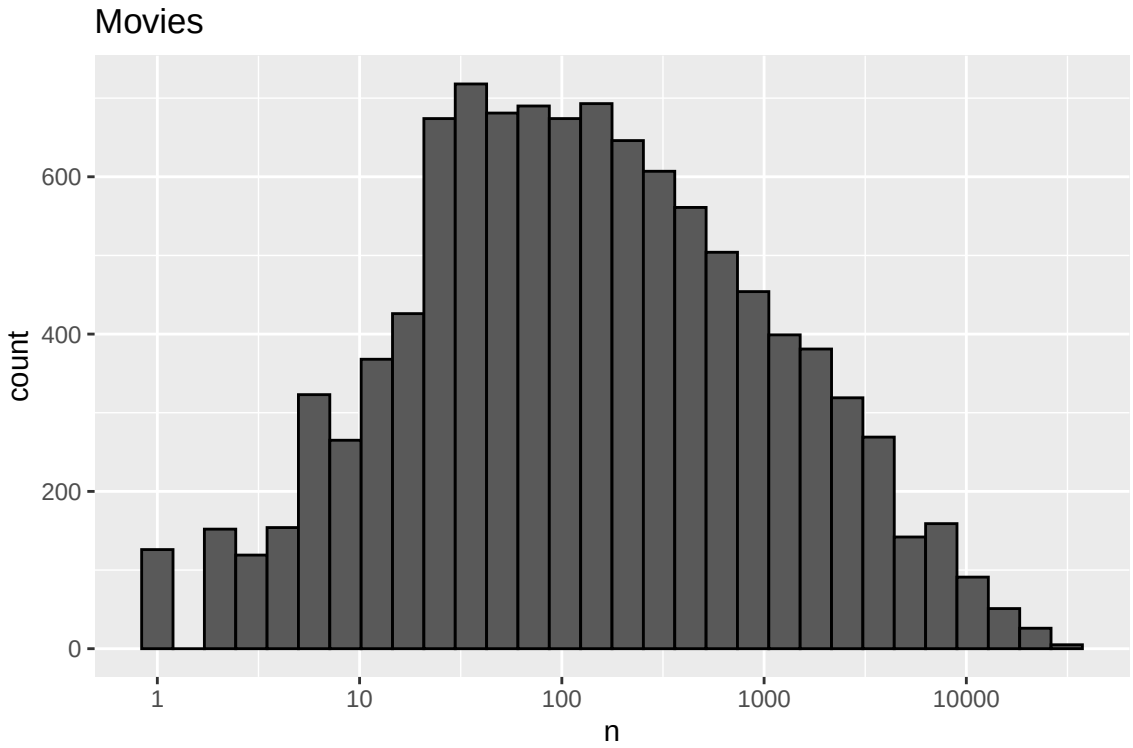


It is clear that users are variable in the total number of ratings made by them. It is found that there are few users who have 10 or less ratings, and on the other side there are also few users who have 1000 or more ratings. This shows that there are variability among users which should be considered in the modeling process. This variability refers to "Users Effect".

2.2.3 Explore the distribution of the movies

It is also good to understand the variability among the movies in terms of ratings. In a similar way, the total number of ratings of all users for each movie is calculated, then a histogram is created to visualize the variability among the movies.

```
edx %>%
  group_by(movieId) %>%
  summarize(n=n()) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Movies")
```



Again, the movies are variable in terms of the total number of ratings for each movie. It also is found that there are few movies who are rated by few users (10 or less ratings), and on the other side there are also few movies who are rated by large number of users (1000 or more ratings). This shows that there are variability among movies which should be considered in the modeling process. This variability refers to "Movies Effect".

2.3 Building the Model

In this section, we seek to find appropriate recommendation model which can achieve a target RMSE less than 0.8649.

Let's first create a function that calculates RMSE. This make things easier.

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

2.3.1 Partition the dataset `edx`

Since we need to tune the parameters in some models, it is good practice to partition the `edx` dataset into training (75%) and testing (25%) subsets. by doing so, we,therefore, have three subsets:
1) Training subset(`train_edx`): this subset will be used in training the models.
2) Testing subset(`test_edx`): this subset will be used in parameters tuning, if any.
3) Validation subset(`validation`): this subset will be used to select the final model that achieve the target RMSE.

```
# Partition EDX Dataset into Training and Test Datasets
set.seed(1985)
```

```
test_indices <- createDataPartition(edx$rating, times = 1, p=0.25, list=FALSE)
train_edx <- edx[-test_indices,]
test_edx <- edx[test_indices,]

#  Remove the users and movies that don't appear in both test and training datasets.
test_edx <- test_edx %>% semi_join(train_edx,by="movieId") %>%
  semi_join(train_edx, by="userId")
```

2.3.2 Average Model

This is a trivial model where we predict the same rating for all movies regardless of the users. This model can be mathematically expressed as:

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

$Y_{u,i}$: the rating made by user u for movie i.
μ: true mean of ratings.
$\epsilon_{u,i}$: Gaussian residuals with mean 0.

The least square solution of this model is the average of ratings μ.

```
#============= First Model = Constant Mean =============
# Y(u,i) = Mu + E(u,i) u: user, i: movie
mu_hat <- mean(edx$rating) # mu_hat=3.5125

# Calculate RMSE
rmse_muhat <- RMSE(validation$rating, mu_hat) #RMSE=1.06120

# Store the result
accuracy_results <- tibble(Model = "Average_Model", RMSE = rmse_muhat)
```

It is found that the average of ratings is 3.5124652 with RMSE of 1.0612018.

2.3.3 Movie Effect Model

We have previously seen that the number of ratings are variable from movie to movie. Some movies are rated more than others. There are very few movies that rated once, while ther other few that have large ratings. This variability should be considered in the model. This is called "Movie Effect". It can be mathematically expressed as:

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

$Y_{u,i}$: the rating made by user u for movie i.
μ: true mean of ratings.
b_i: Movie Effect; average of all ratings made by all users for movie i.
$\epsilon_{u,i}$: Gaussian residuals with mean 0.

The approximate solution of this model is $\hat{b_i} = Y_{u,i} - \hat{\mu}$.

```r
# ============= Second Model = Movie Effect  =============
# Y(u,i) = Mu + b(i) + E(u,i) u:user, i:movie, b:movie effect
# b(i) = Y(u,i) - Mu
movie_effect <- edx %>% group_by(movieId) %>% summarize(bi = mean(rating-mu_hat))

# Calculate the predicted ratings after considering the movie effect.
predicted_ratings_mv <- validation %>%
  left_join(movie_effect, by="movieId") %>%
  mutate(pred_mv = mu_hat + bi)

# Calculate RMSE
rmse_bi <- RMSE(validation$rating, predicted_ratings_mv$pred_mv)

# Store the result
accuracy_results <- bind_rows(accuracy_results,
                              tibble(Model = "Movie_Effect_Model", RMSE = rmse_bi))
```

After running the model, It is found that the achieved RMSE is 0.9439087. It is still so far away from the target RMSE (0.8649).

2.3.4 Movie and User Effects Model

We have also previously seen that the number of ratings are variable from user to user. Some users rated movies more than others, While there are few users who rated very few movies, there are also other few who rated alot of movies. This variability should be considered in the model. This is called "User Effect". It can be mathematically expressed as:

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

$Y_{u,i}$: the rating made by user u for movie i.
μ: true mean of ratings.
b_i: Movie Effect; average of all ratings made by all users for movie i.
b_u: User Effect; average of all ratings made by user u for all movies.
$\epsilon_{u,i}$: Gaussian residuals with mean 0.

The approximate solution of this model is $\hat{b_u} = Y_{u,i} - \hat{\mu} - \hat{b_i}$.

```r
# =============  Third Model = User Effect =============
# Y(u,i) = Mu + b(i) + b(u) + E(u,i) u:user, i:movie, b:movie effect
# b(u) = Y(u,i) - Mu - b(i)
user_effect <- edx %>%
  left_join(movie_effect, by="movieId") %>%
  group_by(userId) %>%
  summarize(bu = mean(rating-mu_hat-bi))

predicted_ratings_ur <- validation %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect, by="userId") %>%
  mutate(pred_ur = mu_hat + bi + bu)

# Calculate RMSE
rmse_bu <- RMSE(validation$rating, predicted_ratings_ur$pred_ur)
```

```r
# Store the result
accuracy_results <- bind_rows(accuracy_results,
                              tibble(Model = "User_Effect_Model", RMSE = rmse_bu))
```

After taking the effect of users variability in the model, RMSE is decreased to reach 0.8653488. This is a high improvement towards the target.

2.3.5 Regularized Movie and User Effects Model

We have learned from statistics that as we have larger sample size, then the estimated parameters are more reliable and have less uncertainty. As will see now, we will find that b_i and b_u are estimated from one rating only, which introduced high uncertainty and overfitting. To understand how?, please refer to This section form "Introduction to Data Science" Book.

The regularized movie and users effects model can be mathematically expressed as follows:

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

Minimizing the following loss function:

$$\sum_{u,i}(y_{u,i} - \mu - b_i - b_u)^2 + \lambda(\sum_i b_i^2 \ + \sum_u b_u^2)^2$$

$Y_{u,i}$: the rating made by user u for movie i.
μ: true mean of ratings.
b_i: Movie Effect; average of all ratings made by all users for movie i.
b_u: User Effect; average of all ratings made by user u for all movies.
$\epsilon_{u,i}$: Gaussian residuals with mean 0.
λ: Regularization term/penalty term.

The solution to the previous model is given by the following formulas:
1) First, estimate the regularized movie effect from the following formula:

$$\hat{b_i}(\lambda) = \frac{1}{\lambda + n_i}\sum_{u=1}^{n_i}(Y_{u,i} - \hat{\mu})$$

2) Second, estimate the regularized user effect from the following formula:

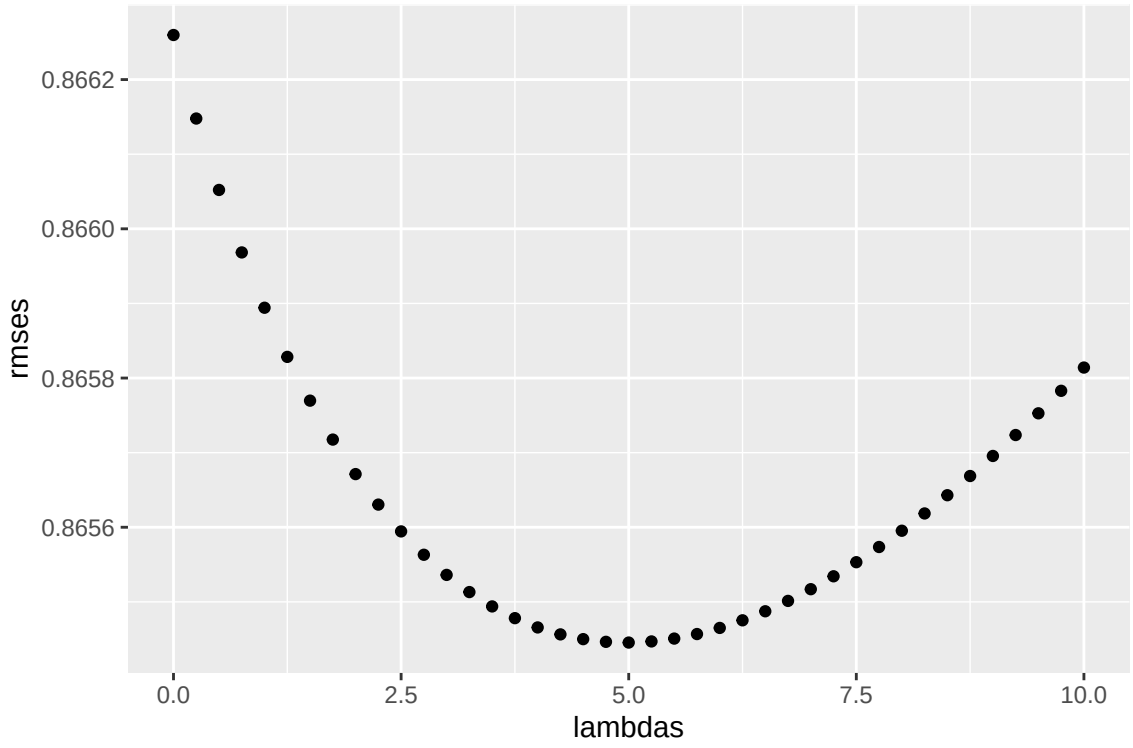
$$\hat{b_u}(\lambda) = \frac{1}{\lambda + n_u}\sum_{i=1}^{n_u}(Y_{u,i} - \hat{\mu} - \hat{b_i})$$

Then the predicted rating can be calculated from the following formula:

$$\hat{y}_{u,i} = \hat{\mu} + \hat{b_i} + \hat{b_u}$$

It is worth mentioning that λ can be tuned so that the optimal λ gives the lowest RMSE. So, we will use the testing data `test_edx` to for parameter tuning, and validation dataset `validation` for calculating RMSE of the tuned model.

```r
# =====  Fourth Model = Regularized Movie and User Effects =====

lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas, function(lambda){

  mu_hat <- mean(train_edx$rating)

  b_i_hat <- train_edx %>%
    group_by(movieId) %>%
    summarize(b_i_hat = sum(rating - mu_hat)/(n()+lambda))

  b_u_hat <- train_edx %>%
    left_join(b_i_hat, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u_hat = sum(rating - b_i_hat - mu_hat)/(n()+lambda))

  predicted_ratings <-
    test_edx %>%
    left_join(b_i_hat, by = "movieId") %>%
    left_join(b_u_hat, by = "userId") %>%
    mutate(pred_rate = mu_hat + b_i_hat + b_u_hat) %>%
    pull(pred_rate)

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

qplot(lambdas, rmses)
```



```r
# Selected Lambda
sel_lambda <- lambdas[which.min(rmses)]
sel_lambda
```

```
## [1] 5
```

```r
# Predict the ratings using the selected Lambda
mu_hat <- mean(edx$rating)

b_i_hat <- edx %>%
  group_by(movieId) %>%
  summarize(b_i_hat = sum(rating - mu_hat)/(n()+sel_lambda))

b_u_hat <- edx %>%
  left_join(b_i_hat, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u_hat = sum(rating - b_i_hat - mu_hat)/(n()+sel_lambda))

predicted_ratings <-
  validation %>%
  left_join(b_i_hat, by = "movieId") %>%
  left_join(b_u_hat, by = "userId") %>%
  mutate(pred_rate = mu_hat + b_i_hat + b_u_hat)

# Calculate RMSE
rmse_reg <- RMSE(validation$rating, predicted_ratings$pred_rate)

# Store the result
accuracy_results <- bind_rows(
  accuracy_results,
  tibble(Model = "Regularized Movie_and_User_Effect_Model", RMSE = rmse_reg)
  )
```

After running the regularized model, RMSE is decreased to reach 0.8648177, which is less than the target RMSE (0.8649)

2.3.6 Regularized Model with Matrix Factorization

It is now the time to show a very effective method in building a recommendation system; it is the matrix factorization method.

Matrix factorization tries to factorize the users-movies matrix $M_{m\times n}$ into two matrices $P_{m\times f}$ and $Q_{f\times n}$. This factorization process tries to uncover the hidden patters or what is called latent features in the behavior of users while they are rating the movies. It tries to find similar users and similar movies based on the latent features. This similarity could be based on the movie genre or whatever feature. It is not explicit in the rating matrix. So, one of the parameters that should be tuned is the number of latent features. As the features increase more patterns are revealed and thus more accurate predictions can be obtained.

Detailed information can be found in This section form "Introduction to Data Science" Book.

In this project, we made use of recosystem package, which is an R wrapper of the LIBMF library developed by Yu-Chin Juan, Wei-Sheng Chin, Yong Zhuang, Bo-Wen Yuan, Meng-Yuan Yang, and Chih-Jen Lin (https://www.csie.ntu.edu.tw/~cjlin/libmf/), an open source library for recommender system using parallel marix factorization(Qiu, 2021).

```
set.seed(19850, sample.kind = "Rounding")

# Convert 'edx' and 'validation' subsets to recosystem input objects
# edx_reco1 <-  with(edx, data_memory(user_index = userId,
#                                     item_index = movieId,
#                                     rating = rating))
edx_reco <-  with(edx, data_memory(user_index = userId,
                                   item_index = movieId,
                                   rating = rating))

validation_reco  <- with(validation, data_memory(user_index = userId,
                                                 item_index = movieId,
                                                 rating = rating))

# Create the model object
reco_model <-  recosystem::Reco()

# Tune the parameters
opts <-  reco_model$tune(edx_reco, opts = list(dim = c(10, 20,30),
                                    lrate = c(0.1, 0.2),
                                    costp_l2 = c(0.01, 0.1),
                                    costq_l2 = c(0.01, 0.1),
                                    nthread  = 4, niter = 10))

# Train the model
reco_model$train(edx_reco, opts = c(opts$min, nthread = 4, niter = 20))
```

2.3.6.1 MAtrix Factorization using recosystem package

```
## iter      tr_rmse          obj
##    0       0.9708   1.1970e+07
##    1       0.8718   9.8682e+06
##    2       0.8387   9.1658e+06
##    3       0.8172   8.7541e+06
##    4       0.8019   8.4733e+06
##    5       0.7903   8.2759e+06
##    6       0.7810   8.1289e+06
##    7       0.7730   8.0120e+06
##    8       0.7661   7.9165e+06
##    9       0.7599   7.8328e+06
##   10       0.7547   7.7651e+06
##   11       0.7499   7.7079e+06
##   12       0.7457   7.6577e+06
##   13       0.7417   7.6109e+06
##   14       0.7382   7.5721e+06
##   15       0.7350   7.5362e+06
##   16       0.7321   7.5050e+06
##   17       0.7293   7.4740e+06
##   18       0.7268   7.4505e+06
##   19       0.7245   7.4271e+06
```

```
# Calculate the prediction
predicted_ratings_mf <-  reco_model$predict(validation_reco, out_memory())

# Calculate RMSE
rmse_mf <- RMSE(validation$rating, predicted_ratings_mf)

# Store the result
accuracy_results <- bind_rows(
  accuracy_results,
  tibble(Model = "Matrix Factorization Model", RMSE = rmse_mf)
  )
```

Substantial decrease in RMSE was observed after running the matrix factorization model. RMSE of 0.7825342 was obtained, which is below the target RMSE (0.8649).

3 Results

Let's view the final table of all models with their corresponding RMSEs as below:

```
knitr::kable(accuracy_results,
             caption="The implemented Models with thier accuracies.")
```

Table 2: The implemented Models with thier accuracies.

Model	RMSE
Average_Model	1.0612018
Movie_Effect_Model	0.9439087
User_Effect_Model	0.8653488
Regularized Movie_and_User_Effect_Model	0.8648177
Matrix Factorization Model	0.7825342

It is clear that the matrix factorization has made substantial progress with achieved RMSE of 0.7825342. Then the regularized model comes in the second place in terms of low RMSE of (0.8648177).

4 Conclusion and Recommendations

The results prove the importance of Regularization and Matrix Factorization in building efficient recommendation systems. There are several implementations of the two methods that worth to be tested in future works. For example, `recommenderlab` package have several methods and models to build a recommendation system.

Parameters tuning deserve more effort to try different parameters and test how the accuracy change.

References

Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37.

Pantelis Monogioudis, P. (2020). *CS301 Introduction to Data Science*.

Qiu, Y. (2021). *recosystem: Recommender System Using Parallel Matrix Factorization*.