HarvardX Data Science Professional Certificate Capstone Project: Movie Recommendation System

Ronald Chow

11/14/2020

Table of Contents

1 Summary

1.1 Introduction

There have been countless movies published throughout human history. Movies are integral part of our lives, and have great influences on our personalities, worldview, etc. With a vast number of movies, there is

even a larger group of audiences, all with slightly different preferences and tastes for movies. Each one of us are shaped by our respective cultures, educations and peers, all accumulating into building our own very unique tastes for movies. It proves to be a difficult task to accuartely predict how each movie is rated in each viewer's eyes.

With the help of data science, specifically machine learning, it is now possible to predict what movies each of us might like, and recommend them to us, identifying and taking into account numerous factors that constitute our tastes for movies. The Movie Recommendation System that is built in this project does all the above, and even predicts how any particular user might rate any particular movie, even ones they haven't rated yet.

This project is a Capstone Project in the HarvardX Data Science Professional Certificate programme. I am to build a Movie Recommendation System with R. The movielens dataset used in this project is a 10M version (found at: https://grouplens.org/datasets/movielens/10m/) of the full original dataset (found at: https://grouplens.org/datasets/movielens/latest/). This is mainly due to the original dataset being too large and computations would be timely and difficult.

1.2 Dataset

The full Movielens Dataset is a dataset with 27 million ratings for 58,000 movies by 280,000 users. The Movielens 10M Dataset used in this project is, as the name suggests, a dataset with 10 million ratings for 10,000 movies by 72,000 users.

1.3 Methodology

The Movie Recommendation System born of this Capstone Project aims to have a RMSE (Root-Mean-Square Error) of smaller than 0.86490 in predicting user ratings for movies. To achieve this goal, I will make use of Linear Models, Regularization of said Linear Model and finally Matrix Factorization to build such a prediction model. The prediction models will be built out of considerations for different factors/effects that could guide user ratings. I will evaluate the relevant variables in later sections to determine consistent and reliable factors that could constitute a linear model that, to the largest extent, predict user ratings for movies.

2 Data Analysis

2.1 Preparations

First we must download the useful packages.

```
# 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
```

Next we download the required dataset and tidy the data.

```
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 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")
```

Create two sets of data, one for Training (`edx`), and one for Validation (`validation`). The `movielens` dataset is partitioned in such a way that the Training Set amounts to 90% of the data, while the Validation Set amounts to the remaining 10%.

```
# 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)
```

The same procedures are further performed within the Training Set (`edx`) to partition it into Training (`train_set_edx`) and Validation (`test_set_edx`) Sets. This enables us to perform validations after building every model, so that the RMSE can be monitored to see if our models worked in reducing the RMSE.

```
set.seed(1, sample.kind="Rounding")

#Randomly splitting dataset to obtain test index (train:test = 9:1)
test_index_edx <- createDataPartition(y = edx$rating,
```

```
                                    times = 1,
                                    p = 0.1,
                                    list = FALSE)
train_set_edx <- edx[-test_index_edx,]
temp <- edx[test_index_edx,]

#userId and movieId in test dataset are also present in train dataset
test_set_edx <- temp %>%
  semi_join(train_set_edx, by = "userId") %>%
  semi_join(train_set_edx, by = "movieId")

#For removed rows in test dataset, add them back into train dataset
rejected <- anti_join(temp, test_set_edx)
train_set_edx <- rbind(train_set_edx, rejected)

remove(test_index_edx, temp, rejected)
```

2.2 Data Exploration

First we download the useful packages for effective and easily comprehensible data analysis.

```
#Installing ggthemes and scales for plotting graphs
if(!require(ggthemes))
  install.packages("ggthemes", repos = "http://cran.us.r-project.org")
if(!require(scales))
  install.packages("scales", repos = "http://cran.us.r-project.org")
```

Let us take a look at the data we are working with, and pay attention to its structure.

```
# Overview of edx dataset
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
```

We see that there are six variables, namely `userId`, `movieId`, `rating`, `timestamp`, `title` and `genres` in the dataset.

```r
# Summary of edx dataset
dim(edx) #9000055 rows and 6 columns
```

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

There are 9000055 rows and 6 columns in the **edx** dataset. Now we shall dive deeper and analyze the variables themselves. Useful information like data types and number of distinct entries for each variable can be obtained.

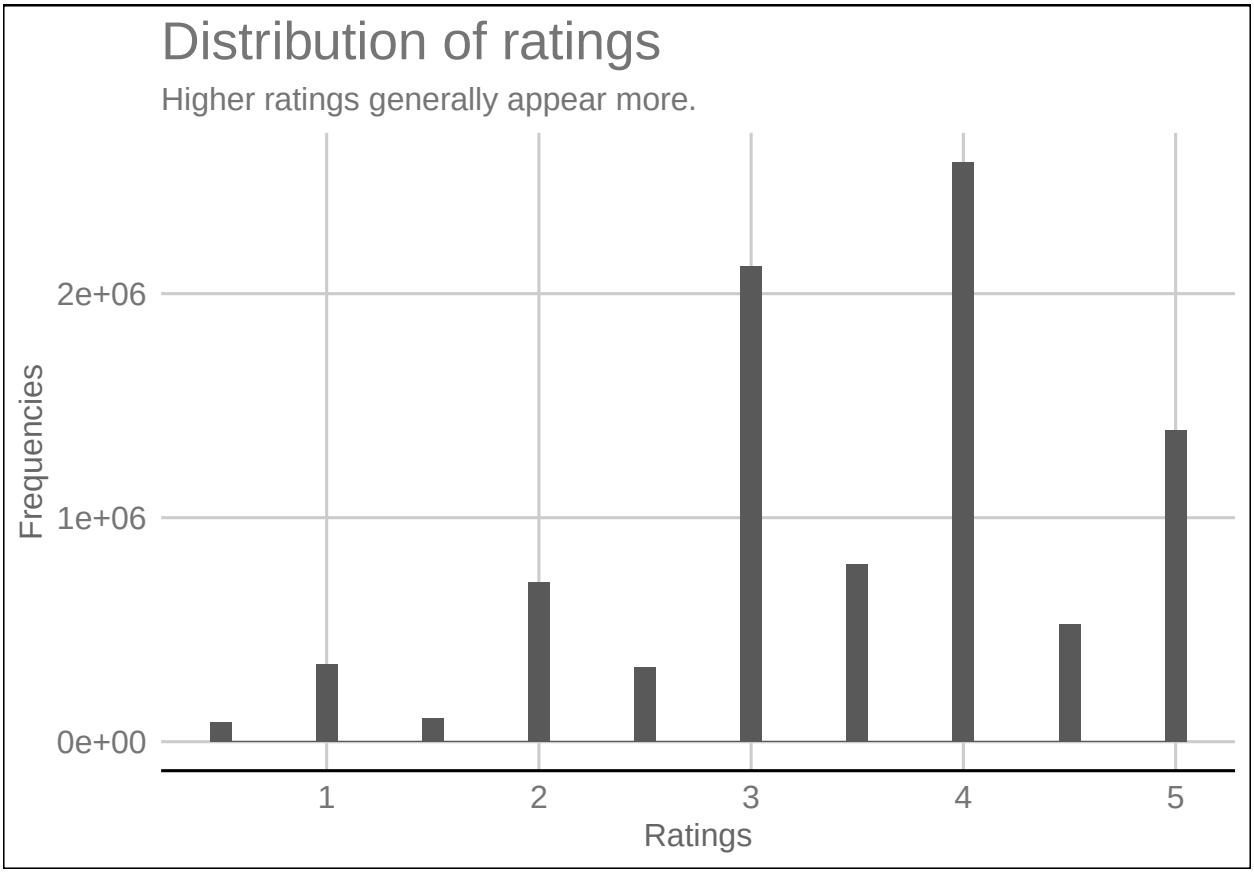
```r
edx_summary <- data.frame(variables =
                            c("userID",
                              "movieID",
                              "rating",
                              "timestamp",
                              "title",
                              "genres"),
                          variables_class = c(class(edx$userId),
                                              class(edx$movieId),
                                              class(edx$rating),
                                              class(edx$timestamp),
                                              class(edx$title),class(edx$genres)),
                          variables_distinct_n = c(n_distinct(edx$userId),
                                                   n_distinct(edx$movieId),
                                                   "N/A",
                                                   "N/A",
                                                   n_distinct(edx$title),
                                                   n_distinct(edx$genres)))
edx_summary
```

```
##   variables variables_class variables_distinct_n
## 1    userID         integer                69878
## 2   movieID         numeric                10677
## 3    rating         numeric                  N/A
## 4 timestamp         integer                  N/A
## 5     title       character                10676
## 6    genres       character                  797
```

```r
#69878 unique users/raters
#10677 unique movies
#10676 titles
#797 combinations of genres
```

Above shows the class of each variable and their respective number of distinct entries. This gives us insight as to how the data is distributed. For instance, we might interestingly notice that the number of unique movieIds and number of unique titles do not match. This means there might be two movieIds registered under the same title. Let us inspect the distributions of the ratings.

```r
edx %>% ggplot(aes(x = rating)) +
  geom_histogram(binwidth = 0.1) +
  xlab("Ratings") +
  ylab("Frequencies") +
  ggtitle("Distribution of ratings", subtitle = "Higher ratings generally appear more.") +
  theme_gdocs()
```




```
#ratings are from 0 to 5 in 0.5 intervals
```

As we can see, the ratings range from 0 to 5 in 0.5 intervals. Users also tend to give higher ratings than lower ones, and prefer to give integer ratings over decimal ones. Similar to before, we create summary for the Validation Set as well.

```
#summary of validation set
dim(validation) #999999 rows and 6 columns
```

```
## [1] 999999      6
```

```
val_summary <- data.frame(variables =
                            c("userID",
                              "movieID",
                              "title",
                              "genres"),
                          variables_distinct_n =
                            c(n_distinct(validation$userId),
                              n_distinct(validation$movieId),
                              n_distinct(validation$title),
                              n_distinct(validation$genres)))

val_summary
```

```
##   variables variables_distinct_n
```

```
## 1    userID               68534
## 2   movieID                9809
## 3     title                9808
## 4    genres                 773
```

```
#68534 unique users/raters
#9809 unique movies
#9809 unique titles
#773 combinations of genres
```

2.3 Variable Analysis

In order to build a prediction model, we must have a deeper understanding of the variables and to what extent they influence ratings, if at all.

2.3.1 User

First we inspect the users themselves. We will first create a summary of how many ratings each of the users gave in total, their individual mean ratings, as well as standard deviation of the ratings they gave.

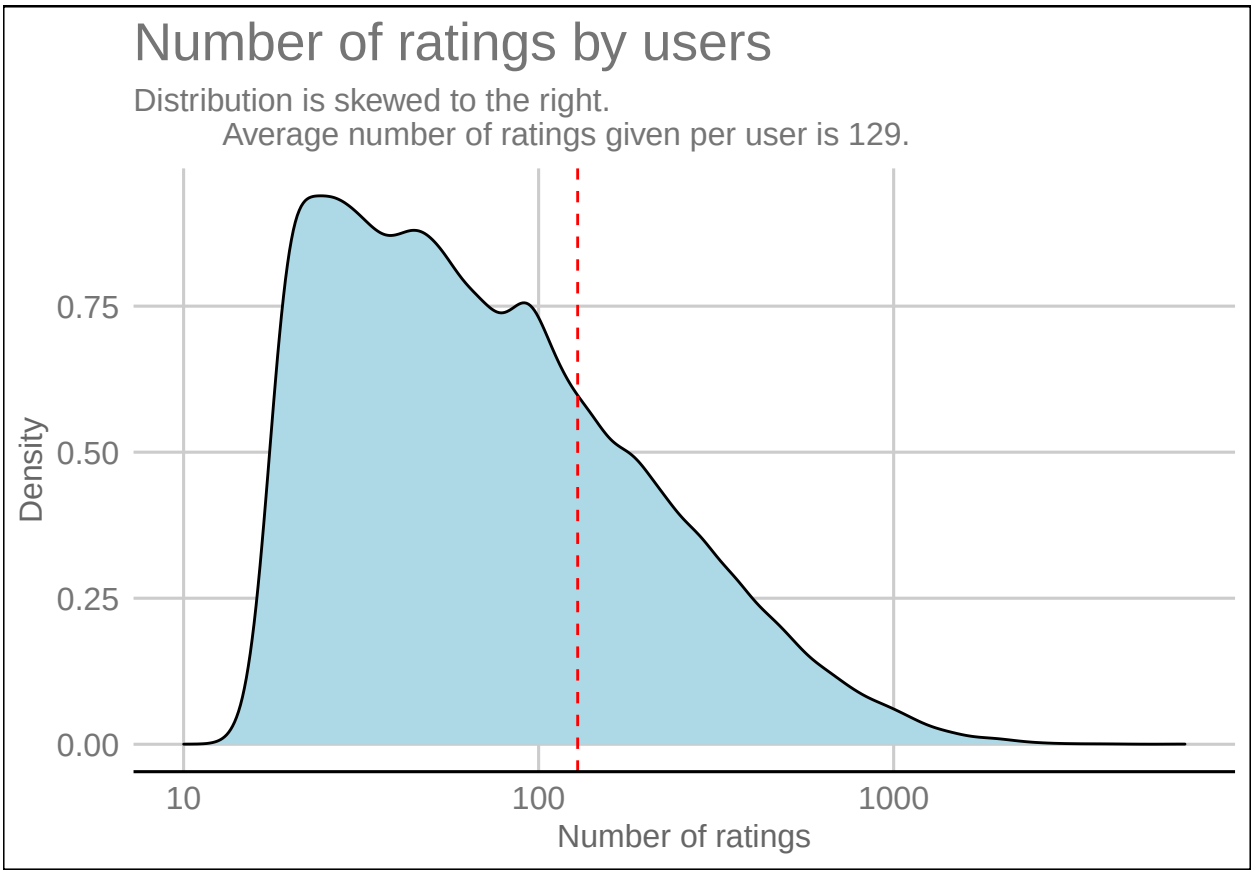
```
#Create summary of user ratings
user_summary <- edx %>% group_by(userId) %>%
  summarize(n_ratings = n(),
            mu = mean(rating),
            sd = sd(rating))

head(user_summary)
```

```
## # A tibble: 6 x 4
##   userId n_ratings    mu    sd
##    <int>     <int> <dbl> <dbl>
## 1      1        19  5    0
## 2      2        17  3.29 0.920
## 3      3        31  3.94 0.761
## 4      4        35  4.06 1.16
## 5      5        74  3.92 1.11
## 6      6        39  3.95 1.10
```

Then we plot the user-rating distribution to visualise the relationship between users and ratings.

```
#Plotting number of users and their ratings
edx %>% group_by(userId) %>%
  summarise(n=n()) %>%
  ggplot(aes(n)) +
  geom_density(fill="light blue") +
  geom_vline(aes(xintercept = mean(user_summary$n_ratings)),
             color="red",
             linetype="dashed") +
  scale_x_log10() +
  xlab("Number of ratings") +
  ylab("Density") +
```

```
  ggtitle("Number of ratings by users",
          subtitle = "Distribution is skewed to the right.
          Average number of ratings given per user is 129.") +
  scale_y_continuous(labels = comma) +
  theme_gdocs()
```



As seen in the plot, the distribution is right skewed. This shows that most data falls on the left, meaning most of the users have given relatively small amount of movie ratings, which is confirmed by the low average number of ratings given per user (129).

2.3.2 Movie

The movies and their ratings will be inspected in this section. We will first create a summary of how many ratings each movieId is associated with, their respective mean ratings, as well as the standard deviation of the ratings received.

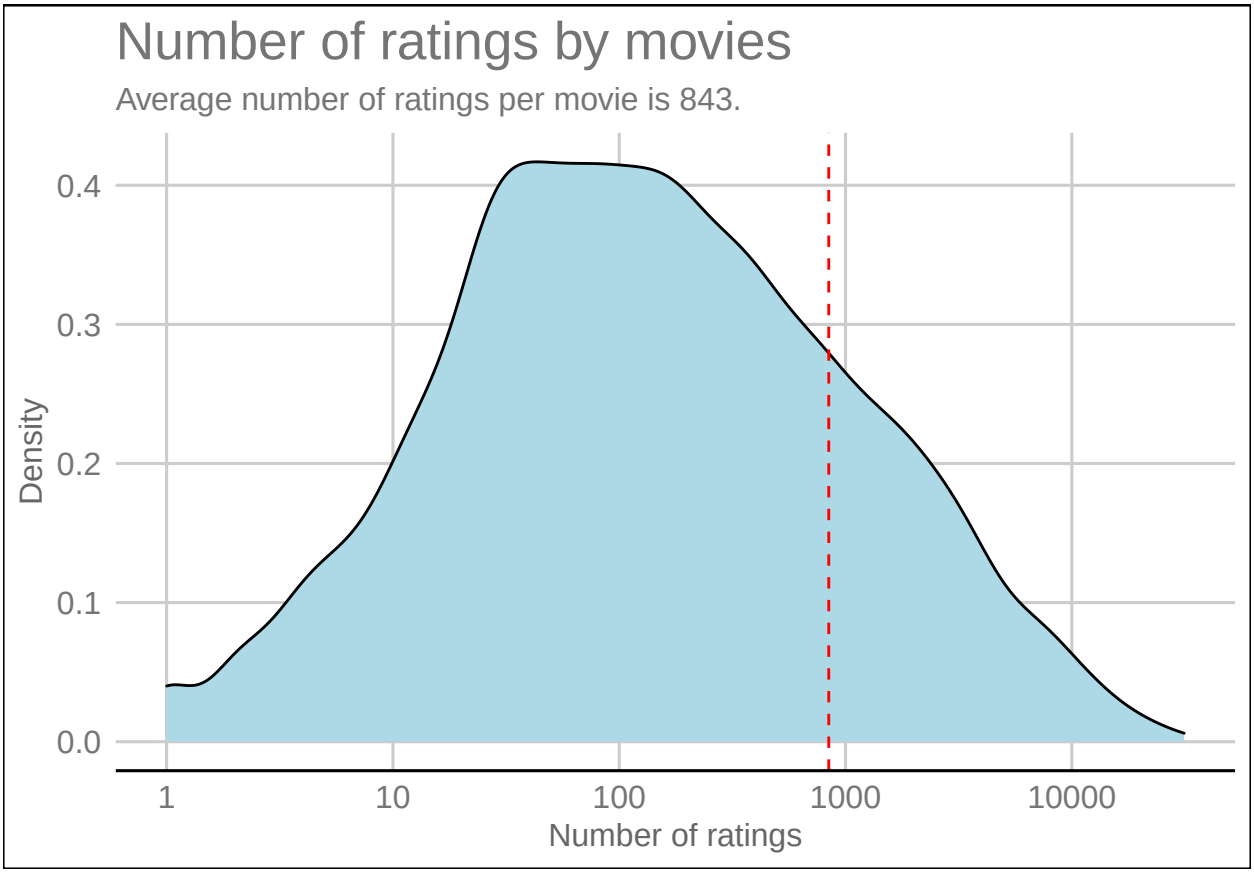
```
#Summary of movies and their respective number of ratings
movie_summary <- edx %>% group_by(movieId) %>%
  summarize(n_ratings = n(),
            mu = mean(rating),
            sd = sd(rating))

movie_summary
```

```
## # A tibble: 10,677 x 4
```

```
##    movieId n_ratings    mu    sd
##      <dbl>     <int> <dbl> <dbl>
##  1       1     23790  3.93 0.897
##  2       2     10779  3.21 0.951
##  3       3      7028  3.15 0.999
##  4       4      1577  2.86 1.09
##  5       5      6400  3.07 0.964
##  6       6     12346  3.82 0.885
##  7       7      7259  3.36 0.957
##  8       8       821  3.13 0.979
##  9       9      2278  3.00 0.968
## 10      10     15187  3.43 0.864
## # ... with 10,667 more rows
```

Next we plot the movie-rating distribution to visualize how many ratings there are for the movies in general.

```
#Plotting movies and number of ratings
edx %>% group_by(movieId) %>%
  summarise(n=n()) %>%
  ggplot(aes(n)) +
  geom_density(fill="light blue") +
  geom_vline(aes(xintercept=mean(movie_summary$n_ratings)),
             color="red",
             linetype="dashed") +
  scale_x_log10() +
  ggtitle("Number of ratings by movies",
          subtitle = "Average number of ratings per movie is 843.") +
  xlab("Number of ratings") +
  ylab("Density") +
  theme_gdocs()
```



Average number of ratings per movie is 843, which is a fairly large amount. Also, some movies are rated more than others, due to varying popularity among movies.

2.3.3 Ratings

Next, we inspect the ratings themselves. First we create a summary for these ratings and their resepective frequencies.

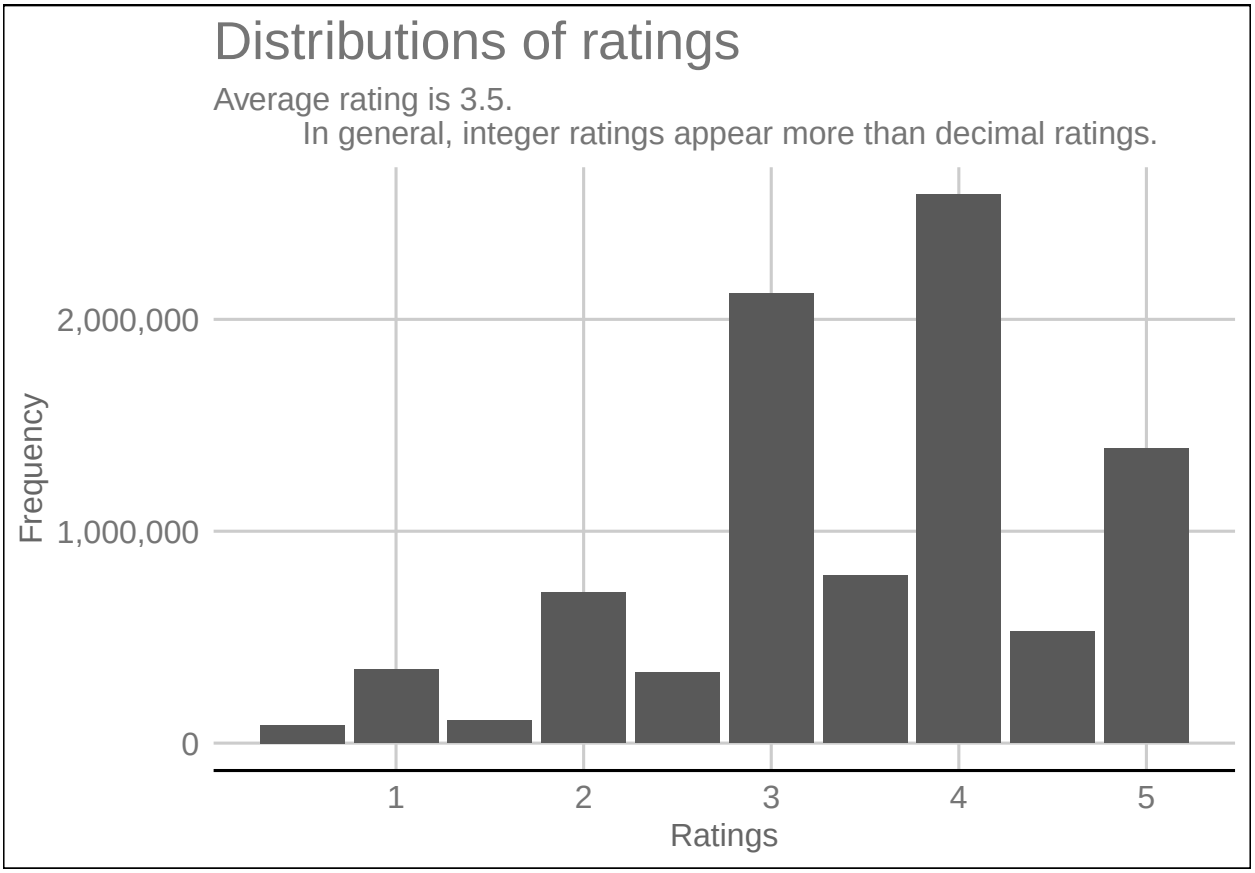
```
#Creating summary of ratings
ratings_summary <- edx %>% group_by(rating) %>%
  summarize(n=n())

ratings_summary
```

```
## # A tibble: 10 x 2
##    rating       n
##     <dbl>   <int>
##  1    0.5   85374
##  2    1    345679
##  3    1.5  106426
##  4    2    711422
##  5    2.5  333010
##  6    3   2121240
##  7    3.5  791624
##  8    4   2588430
##  9    4.5  526736
```

```
## 10    5   1390114
```

Then we visualise the above summary statistics into a bar plot.

```
#Plotting frequencies of each rating(from 0.5 to 5)
edx %>% group_by(rating) %>%
  summarise(count=n()) %>%
  ggplot(aes(x=rating, y=count)) +
  geom_bar(stat="identity") +
  scale_y_continuous(labels = comma) +
  ggtitle("Distributions of ratings",
          subtitle = "Average rating is 3.5.
          In general, integer ratings appear more than decimal ratings.") +
  xlab("Ratings") +
  ylab("Frequency") +
  theme_gdocs()
```



We see 3 and 4 are two ratings that are much more prevalent than the other ratings. This is perhaps due to users' psychological tendency to adopt neutral attitudes towards movies they do not have strong opinions for or against[1]. This could be helpful since this means most ratings are close to the average rating, which is 3.5, and we can, to some extent, rely on the mean method to predict most of the ratings from users.

[1]Coping with Ambivalence: The Effect of Removing a Neutral Option on Consumer Attitude and Preference Judgments - https://pdfs.semanticscholar.org/46cd/f5a610103103bc59b214e5ac66419ede7d94.pdf

2.3.4 Genres

Genres are what define movies. It is also arguably the main factor that determines user reactions. First we shall take a look at different combinations of movie genres and how many different movies fall under these categories.

```
#Creating summary of genres (combinations of genres and their appearances)
genres_summary1 <- edx %>% group_by(genres) %>%
  summarise(n=n())

head(genres_summary1)
```

```
## # A tibble: 6 x 2
##   genres                                                n
##   <chr>                                             <int>
## 1 (no genres listed)                                    7
## 2 Action                                            24482
## 3 Action|Adventure                                  68688
## 4 Action|Adventure|Animation|Children|Comedy          7467
## 5 Action|Adventure|Animation|Children|Comedy|Fantasy   187
## 6 Action|Adventure|Animation|Children|Comedy|IMAX       66
```

There are 7 movies with no genres listed. Next we shall take a deeper look into how many movies can be classified under each genre.

```
#Creating summary of genres (each individual genre and their appearances)
genres_summary2 <- edx %>% separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarize(n=n())

head(genres_summary2)
```

```
## # A tibble: 6 x 2
##   genres                 n
##   <chr>              <int>
## 1 (no genres listed)     7
## 2 Action           2560545
## 3 Adventure        1908892
## 4 Animation         467168
## 5 Children          737994
## 6 Comedy           3540930
```

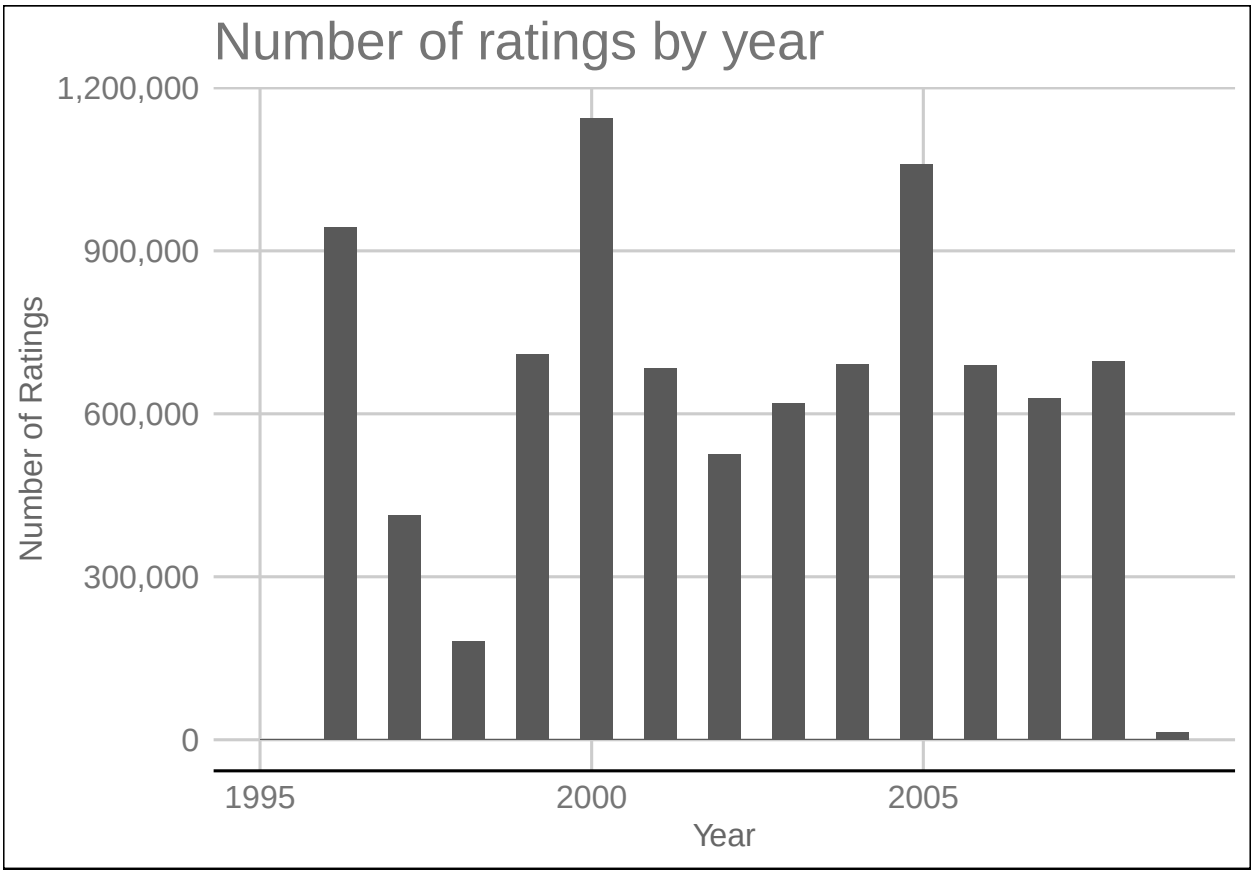
2.3.5 Time

The years when the movies came out are also worth inspecting. Previous steps have already revealed that the variable timestamp does not directly tell us the time in years. Therefore we must first call the lubridate package to help us wrangle the entries into readable format.

```
#Calling lubridate for date-time wrangling
library(lubridate)
```

Now we can plot the years and their respective ratings.

```r
#Plotting years and their respective number of ratings
edx %>% mutate(year = year(as_datetime(timestamp))) %>%
  ggplot(aes(x=year)) +
  geom_histogram() +
  scale_y_continuous(labels = comma) +
  ggtitle("Number of ratings by year") +
  xlab("Year") +
  ylab("Number of Ratings") +
  theme_gdocs()
```



We observe that the year 2000 and onwards had more ratings from users than before.

2.4 Modeling Approach

From the above analysis, we learnt that the mean rating is a reliable predictor for predicting most of the ratings. Moreover, there is significant bias on the ratings resulted from User Effect and Movie Effect. To understand these two effects, we must understand what determines the ratings. There are two facets that determine what ratings any particular movie receive from any particular user. A user that tends to like what he or she watches will give a higher rating than the average; and on the other hand, a movie that is more popular than the others will receive better ratings. These two facets represent two types of bias, which can be accounted in our prediction model by quantifying the tendencies individual users have when giving ratings, and the tendencies of the ratings individual movies receive – User Effect and Movie Effect, as they are aptly named.

In this project, the baseline model will be a linear model based on the mean rating, user effect and movie

effect. This is to serve as a rudimentary prediction model. The model looks like this:

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

where $\hat{Y}_{u,i}$ represents the predicted rating, μ represents the mean ratings for all movies, b_i represents the Movie Effect, b_u represents the User Effect, and $\epsilon_{u,i}$ represents the error distribution.

The model will then be improved using regularization method.

As seen in previous analysis, outliers exist in the rating distribution from both the user side and the movie side. Some movies are rated a lot less by users and some users rate a lot less movies compared to others. These extreme cases, or outliers, have very small sample sizes. They present high risk of error in predictions and therefore has to be properly accounted for in our prediction model. The way to account for these unreliable estimates is by regularization. In regularization, we penalize large estimates resulted from small sample sizes. To achieve this, we use a tuning parameter λ that only takes effect when the sample size is smaller than a certain value. What exact value λ will be, is determined by running simulations and cross-validations to find out the value that corresponds to the smallest RMSE. For Movie Effect, the model looks like this:

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

Next, for User Effect, the model looks like this:

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

As we can see, the smaller the value of n_i in the above models, the more significant the shrinking effect $\frac{1}{\lambda+n_i}$ has on the estimates. For a large n_i, the effect of the penalty factor λ can be practically ignored both mathematically and statistically. Finally we summarize the equation as follows:

$$\frac{1}{N} \sum_{\mu,i} (Y_{u,i} - \mu - b_i - b_u)^2 + \lambda (\sum_i b_i^2 + \sum_i b_u^2)$$

Regularization effectively penalizes large estimates resulted from small sample sizes.

Apart from calculating and predicting ratings on a statistical basis, we also could account for similar rating patterns by groups of movies and users. This could be achieved by Matrix Factorization. In Matrix Factorization, we decompose user-movie interaction and convert into a matrix with users as rows and movies as columns. The algorithm discovers both implicit and explicit rating patterns and gives estimates based on these correlations. To run the analysis, I will be using the `recosystem` package.

3 Results

3.1 Defining RMSE function

To evaluate our model(s), we will be using Root-Mean-Square Error (RMSE) as our loss function. Root-Mean-Square Error is the standard deviation of residuals, which represents prediction errors, and is commonly used to evaluate prediction models and recommendation systems.

```
rmse <- function(actual_rating, predicted_rating){
  sqrt(mean((actual_rating - predicted_rating)^2))
}
```

3.2 RMSE Goal

The RMSE this project aims to reach lower than is a value of 0.86490.

```
#Set RMSE Goal and creating result tibble
result <- tibble(Method = "Goal", RMSE = 0.86490)
```

3.3 Mean Method

$$\hat{Y}_{u,i} = \mu + \epsilon_{u,i}$$

To begin building our model, we will first use only the mean rating for all predictions and what RMSE we will obtain.

```
#Calculate mean rating in train dataset
mu <- mean(train_set_edx$rating)

#Calculate the RMSE and including RMSE by mean method into result tibble
result <- bind_rows(result,tibble(Method = "Mean",
                                  RMSE = rmse(test_set_edx$rating, mu)))

result #RMSE by mean is 1.06
```

```
## # A tibble: 2 x 2
##   Method  RMSE
##   <chr>  <dbl>
## 1 Goal   0.865
## 2 Mean   1.06
```

3.4 Movie Effect (b_i)

$$\hat{Y}_{u,i} = \mu + b_i + \epsilon_{u,i}$$

Now we include the Movie Effect (b_i) into our prediction model to account for the bias in ratings for individual movies.

```
#Calculating Movie Effect (bi)
bi <- train_set_edx %>%
  group_by(movieId) %>%
  summarize(bi = mean(rating - mu))

#Accounting movie effect (bi) into ratings of each movie
y_hat_1 <- mu + test_set_edx %>%
  left_join(bi, by = "movieId") %>%
  pull(bi)

#Calculate the RMSE and including it into result tibble
result <- bind_rows(result,
                    tibble(Method = "Mean + Movie Effect",
                           RMSE = rmse(test_set_edx$rating, y_hat_1)))


result #RMSE by Mean + Movie Effect is 0.943
```

```
## # A tibble: 3 x 2
##   Method                RMSE
##   <chr>                <dbl>
## 1 Goal                 0.865
## 2 Mean                 1.06
## 3 Mean + Movie Effect 0.943
```

3.5 User Effect (b_u)

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

The last predictor in our linear model is the User Effect (b_u), which accounts for the users' individual tastes for movies that could affect the ratings they give.

```
#Calculating User Effect (bu)
bu <- train_set_edx %>%
  left_join(bi, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(bu = mean(rating - mu - bi))

#Prediction based on Mean, Movie Effect and User Effect
y_hat_2 <- test_set_edx %>%
  left_join(bi, by='movieId') %>%
  left_join(bu, by='userId') %>%
  mutate(predict = mu + bi + bu) %>%
  pull(predict)

#Calculate the RMSE and including into result tibble
result <- bind_rows(result,
                    tibble(Method = "Mean + Movie Effect + User Effect",
                           RMSE = rmse(test_set_edx$rating, y_hat_2)))

result #RMSE by Mean + Movie Effect + User Effect is 0.865
```

```
## # A tibble: 4 x 2
##   Method                              RMSE
##   <chr>                              <dbl>
## 1 Goal                               0.865
## 2 Mean                               1.06
## 3 Mean + Movie Effect                0.943
## 4 Mean + Movie Effect + User Effect 0.865
```

We are close to achieving a RMSE of 0.86490 with our linear model. With regularization, we can do better.

3.6 Regularization

$$\frac{1}{N}\sum_{\mu,i}(Y_{u,i} - \mu - b_i - b_u)^2 + \lambda(\sum_i b_i^2 + \sum_i b_u^2)$$

In this part, we perform regularization to account for large estimates from small sample sizes. The tuning parameter/penalty factor λ will be assigned a range of values from 0 to 10 for tuning.

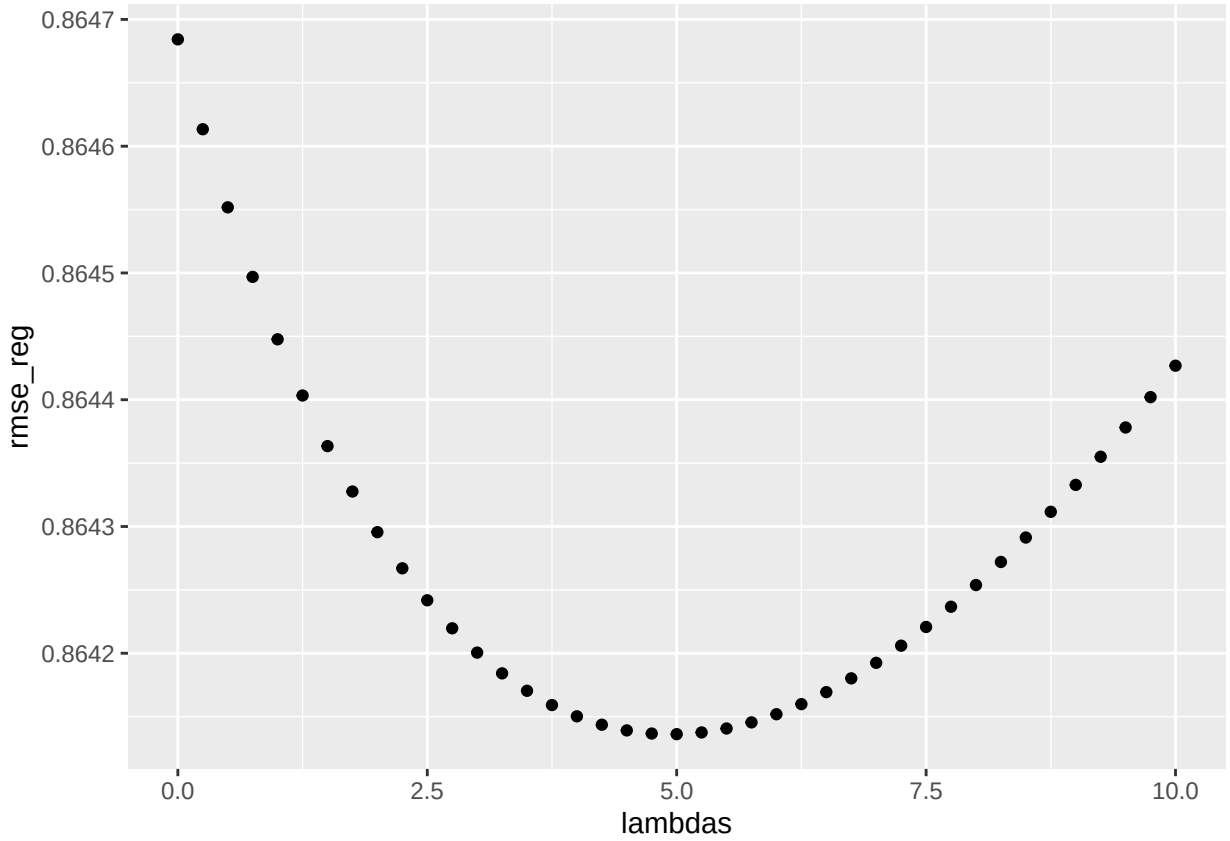
```r
#Defining range of values for lambda for testing
lambdas <- seq(0, 10, 0.25)
```

We first define the regularization function.

```r
#Defining regularization function
regularization <- function(lambda, train, test){
  #Calculating mean rating
  mu <- mean(train$rating)
  #calculating Movie Effect (bi)
  bi <- train %>%
    group_by(movieId) %>%
    summarize(bi = sum(rating - mu)/(n()+lambda))
  #Calculating User Effect (bu)
  bu <- train %>%
    left_join(bi, by="movieId") %>%
    group_by(userId) %>%
    summarize(bu = sum(rating - bi - mu)/(n()+lambda))
  #Mean + Movie Effect + User Effect
  predictions <- test %>%
    left_join(bi, by = "movieId") %>%
    left_join(bu, by = "userId") %>%
    mutate(pred_rating = mu + bi + bu) %>%
    pull(pred_rating)

  return(rmse(predictions, test$rating))
}
```

We then apply the regularization function onto the datasets, and test for RMSE with different λs, plotting the resulting RMSEs and their corresponding λ.

```r
#Applying regularization function for each lambda
rmse_reg <- sapply(lambdas,regularization,
                   train=train_set_edx,test=test_set_edx)

#Plotting values of lambda and their respective RMSE
qplot(lambdas, rmse_reg)
```



As seen in the plot, we have successfully found a value for the tuning parameter λ that yields an RMSE of 0.864. Now to find out what that value of λ is exactly:

```
#Determining which value of lambda yields the smallest value of RMSE
lambda <- lambdas[which.min(rmse_reg)]

lambda
```

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

The best value for λ is 5.

```
#Calculating RMSE and including RMSE with Regularization Model into result tibble
result <- bind_rows(result,
                    tibble(Method="Regularized Model",
                           RMSE = min(rmse_reg)))

result #RMSE by Regularization method is 0.864
```

```
## # A tibble: 5 x 2
##   Method                              RMSE
##   <chr>                              <dbl>
## 1 Goal                               0.865
## 2 Mean                               1.06
## 3 Mean + Movie Effect                0.943
## 4 Mean + Movie Effect + User Effect 0.865
## 5 Regularized Model                  0.864
```

We have successfully reduced the RMSE down to 0.864.

3.7 Matrix Factorization

Matrix Factorization, popularized by its usage in the recommendation system built by Simon Funk in the Netflix Prize Challenge, is an effective alternative to conventional modeling approaches in the above sections. To find out how much better it is at reducing prediction error compared to what linear models can offer, we will be using the `recosystem` package to run the analysis.

The `recosystem` package is an R wrapper of the 'libmf' library for recommendation system using matrix factorization. It was jointly developed by Yu-Chin Juan, Yong Zhuang, Wei-Sheng Chin and Chih-Jen Lin. It works by creating an incomplete matrix of user-item (in this case, user-movie) interaction, where there are both known and unknown entries (ratings), and predicts the value of the unknown entries based on known ones, or observed ones[2].

First we install the `recosystem` package.

```
#Installing recosystem for Matrix Factorization
if(!require(recosystem))
  install.packages("recosystem", repos = "http://cran.us.r-project.org")

set.seed(1, sample.kind = "Rounding")
```

As `recosystem` does have a required format (sparse matrix triplet form) for the data files for both the training and testing datasets, we will first convert the data formats to fit `recosystem`. This also includes the step of specifying the what the user and item indice, as well as rating, correspond to in the dataset.

```
#Data wrangling for train dataset and test dataset to fit the required format
training_data <-  with(train_set_edx, data_memory(user_index = userId,
                                                  item_index = movieId,
                                                  rating = rating))
testing_data  <-  with(test_set_edx,  data_memory(user_index = userId,
                                                  item_index = movieId,
                                                  rating = rating))
```

To use the `recosystem` package properly, there are a few functions we need to be familiar with.

- `Reco()`: Constructing Recommender System Object
- `tune()`: Tuning Model Parameters
- `train()`: Training Recommender Model
- `predict()`: Recommender Model Predictions
- `output()`: Outputting Factorization Matrices

To simplify things, we will not be using the `output()` function to generate the factorization matrices in this report.

The essential arguments and tuning parameters in both `tune()` and `train()` functions are as follows.

- `opts`: Contains the tuning parameters and options
- `dim`: Number of latent factors. Default value = 10
- `cost`: Regularization cost for latent factors. Default vlue = 0.1
- `lrate`: Learning rate, also know as step size in gradient descent. Defualt value = 0.1

[2]https://www.rdocumentation.org/packages/recosystem/versions/0.3

- `niter`: Number of iterations. Default value = 20
- `nthread`: Number of threads for parallel computing. Default value = 1
- `nmf`: Indication of whether to perform non-negative matrix factorization. Default value = `FALSE`
- `verbose`: Indication of whether to show detailed information. Default value = `FALSE`

We will be using default values for most of the tuning parameters.

```
# Creating model object r
r <-  recosystem::Reco()

# Selecting tuning parameters
opts <- r$tune(training_data, opts = list(dim = c(10, 20, 30), lrate = c(0.1, 0.2),
                                          costp_l1 = 0, costq_l1 = 0,
                                          nthread = 1, niter = 10))
# Training algorithm
r$train(training_data, opts = c(opts$min, nthread = 1, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9821   1.1049e+07
##    1       0.8760   8.9994e+06
##    2       0.8419   8.3427e+06
##    3       0.8191   7.9379e+06
##    4       0.8033   7.6798e+06
##    5       0.7912   7.4925e+06
##    6       0.7813   7.3483e+06
##    7       0.7727   7.2323e+06
##    8       0.7654   7.1386e+06
##    9       0.7591   7.0612e+06
##   10       0.7535   6.9942e+06
##   11       0.7486   6.9378e+06
##   12       0.7442   6.8889e+06
##   13       0.7401   6.8457e+06
##   14       0.7365   6.8088e+06
##   15       0.7330   6.7741e+06
##   16       0.7299   6.7426e+06
##   17       0.7270   6.7153e+06
##   18       0.7243   6.6901e+06
##   19       0.7219   6.6686e+06
```

```
#Extracting prediction from r
y_hat_3 <- r$predict(testing_data, out_memory())

#Calculating RMSE by Matrix Factorization and including it into result tibble
result <- bind_rows(result,
                    tibble(Method = "Matrix Factorization by recosystem",
                           RMSE = rmse(test_set_edx$rating, y_hat_3)))

result #RMSE by Matrix Factorisation is 0.786
```

```
## # A tibble: 6 x 2
##   Method                              RMSE
##   <chr>                              <dbl>
## 1 Goal                               0.865
```

```
## 2 Mean                                1.06
## 3 Mean + Movie Effect                0.943
## 4 Mean + Movie Effect + User Effect  0.865
## 5 Regularized Model                  0.864
## 6 Matrix Factorization by recosystem 0.786
```

As we can see, matrix factorization improves the accuracy by 9.0277778%.

3.8 Final Validation

As Matrix Factorization gives the best RMSE, we will be using Matrix Factorization for our final validation. As the name 'Final Validation' suggests, this part will be done using `edx` dataset and `validation` dataset as the training dataset and testing dataset respectively.

```
set.seed(1, sample.kind = "Rounding")
```

As done before, we need to convert the datasets into preferred data formats for `recosystem` package to read.

```
#Data wrangling for edx dataset and validation dataset to fit the required format
edx_recosystem <-  with(edx, data_memory(user_index = userId,
                                         item_index = movieId,
                                         rating = rating))
validation_recosystem  <-  with(validation, data_memory(user_index = userId,
                                                        item_index = movieId,
                                                        rating = rating))
```

```
# Creating model object r
r <-  recosystem::Reco()

# Selecting tuning parameters
opts <-  r$tune(edx_recosystem, opts = list(dim = c(10, 20, 30), lrate = c(0.1, 0.2),
                                            costp_l1 = 0, costq_l1 = 0,
                                            nthread = 1, niter = 10))

# Training algorithm
r$train(edx_recosystem, opts = c(opts$min, nthread = 1, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9724   1.2032e+07
##    1       0.8729   9.8946e+06
##    2       0.8381   9.1701e+06
##    3       0.8155   8.7335e+06
##    4       0.8003   8.4614e+06
##    5       0.7886   8.2650e+06
##    6       0.7790   8.1139e+06
##    7       0.7709   7.9932e+06
##    8       0.7642   7.8987e+06
##    9       0.7584   7.8199e+06
##   10       0.7533   7.7516e+06
##   11       0.7488   7.6957e+06
##   12       0.7447   7.6467e+06
```

```
##   13       0.7410   7.6036e+06
##   14       0.7377   7.5665e+06
##   15       0.7346   7.5325e+06
##   16       0.7318   7.5010e+06
##   17       0.7291   7.4741e+06
##   18       0.7267   7.4482e+06
##   19       0.7244   7.4270e+06
```

```
#Extracting prediction from r
y_hat_final <-  r$predict(validation_recosystem, out_memory())

#Calculating Final RMSE by Matrix Factorization and including into result tibble
result <- bind_rows(result,
                    tibble(Method = "Final Validation",
                           RMSE = RMSE(validation$rating, y_hat_final)))

result #Final Validation yields RMSE of 0.783
```

```
## # A tibble: 7 x 2
##   Method                               RMSE
##   <chr>                               <dbl>
## 1 Goal                                0.865
## 2 Mean                                1.06
## 3 Mean + Movie Effect                 0.943
## 4 Mean + Movie Effect + User Effect   0.865
## 5 Regularized Model                   0.864
## 6 Matrix Factorization by recosystem 0.786
## 7 Final Validation                    0.783
```

Final Validation with Matrix Factorization yields a satisfactory RMSE of 0.783, 9.4693028% lower than the project RMSE Goal of 0.86490.

4 Conclusion

4.1 Summary

In this project, I analyzed various different variables in the dataset to look for useful insights that could tell us what guide(s) ratings. This was a crucial step as it allowed me to identify and verify the existence of Movie Effect (bias) and User Effect (bias), which I could then include in my linear model.

The linear model is built out of mean rating, Movie Effect and User Effect. The resultant RMSE was 0.865, very close to the Project Goal of 0.86490. To further account for statistical outliers that would reduce accuracy of the linear model, I applied regularization method to penalize said groups of data. This produced an RMSE of 0.864.

Having seen as to how Matrix Factorization is utilized in the Netflix Prize Challenge and yielded impressive results, I then applied Matrix Factorization on the dataset to find out how much better it does compared to the linear model I built. This yielded an impressive resultant RMSE of 0.786, 9.02% lower than the RMSE achieved by my linear model.

Applying Matrix Factorization in the final validation gave us a final RMSE of 0.783, which is 9.47% lower than the Project Goal RMSE of 0.86490.

4.2 Limitations

In this project, not all variables were used in building the prediction models. In truth, other predictors also have significant effect in guiding user ratings. For instance, the genres that define what any particular movie is categorized as, are crucial for the users in determining what they feel about that movie. Different users have different taste – some may like horror movies, while others love watching comedies. These individual tastes, much like rating patterns, can be detected by algorithms like Matrix Factorization, so the recommendation system can 'learn' them and predict what other movies they might like or dislike. If we can include these biases, our recommendation system should be much more accurate.

On the other hand, due to limited memory capacity and computing power, it could oftentimes be extremely difficult and time-consuming to run full analysis and estimation. For example, in estimating Movie Effect and User Effect, we did not use `lm()` function to directly build the model, but instead computed the mean bias for each movieId and userId. This was in fact due to the long computing time for running the `lm()` function. Although both generate the same results mathematically, the latter method was not an orthodox way to approach the problem.

4.3 Future Work

Towards the end of this course, we were introduced to `recommenderlab` package, which is a package containing algorithms like User-based collaborative filtering (UBCF), Item-based collaborative filtering, Association rule-based recommender (AR), etc. Unfortunately, the package was not included in the course materials, so we were not properly familiar with the content. Nonetheless, the `recommenderlab` package provides me with algorithms that I can experiment with to build my very own accurate recommendation system in the future.