Movie Recommendation System Using MovieLens Dataset

Jeong Sukchan

11/21/2021

Structure of the Report

———————

Abstract (Summary)

Recommendation systems are pervasive nowadays as they deal with overwhelmed information given to customers and companies filtering the big data. This report is to build a good machine learning model of the recommendation system to predict movie ratings with reasonable accuracy. The dataset used in the report is the MovieLens 10M Dataset. RMSE is used to evaluate the accuracy of algorithms to choose the best one. The best one is refer to the algorithm with the lowest RMSE. For the final evaluation, validation dataset is used, whereas testset dataset is

used to evaluate in the process of developing algorithm before the stage of the final test. The methods of building algorithms are interpreted regression model with different variables, regularized model, and factorization model. The best model was shown to the Parelled Matrix Factorization Model with a RMSE of 0.7829194.

1. Introduction

1.1. Problem Identification

Recommendation system is a critical role for both potential customers and e-commerce related companies. Appropriate recommendation of the product leads for customers to choose their right products tailoring their needs. The competitive recommendation system would help companies to increase their profits attracting potential customers by introducing the appropriate products to meet their tastes.The companies with the competing recommendation system would capture and analyze the user's preference and offer the products or services with higher likelihood of their purchases. The economic impact of the promising recommendation system on company-customer relationship is very clear. So many companies such as Netflix, Amazon, or etc. utilize the recommendation system.

In this report, we will try to build a good model of recommendation system among several machine learning algorithms in order to predict movie ratings with reasonable accuracy. This report is part of the assignment for the professional certificate in the Data Science Program at HarvardX.

1.2. Goal

The goal of this report is to create a good machine learning model for movie recommendation system to predict movie ratings. RMSE will be used to evaluate the quality of a model as to prediction of the results comparing the true values with predicted values. The true value is in the validation dataset separated from the beginning as if it is unknown data. To be specific, our goal is to build a movie recommendation model with the RMSE less than 0.86490 in the validation set.

1.3. Dataset

1.3.1. Data collection

The dataset used in the report is the MovieLens 10M Dataset. The GroupLens research lab generated their own database. The size of data is huge with over 20 million ratings for over 27,000 movies by more than 138,000 users. We will use a small subset of the original data from that lab using the 10M version of it to make the computation easier. The MovieLens dataset can be downloaded from "https://grouplens.org/datasets/movielens/10m/ (https://grouplens.org/datasets/movielens/10m/)". Each user is represented by an id, and no other demographic information is provided.

1.3.2. Dataset Information

The dataset given from edx course contains 6 variables: $ userId; $ movieId; $ rating; $ timestamp; $ title; $ genres. The rating variable is given by users on movies of various genres in the range of 0-5. The task is to predict the rating given by users on movies in the validation data. Each user can rate different movies, and each movies can have multiple ratings from different users.

Let's start with the chunk of code provided us from edx capstone project module in advance.

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

# Note: this process could take a couple of minutes

if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.2     v dplyr   1.0.6
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caret
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: data.table
```

```
##
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```
## The following object is masked from 'package:purrr':
##
##     transpose
```

```
library(tidyverse)
library(caret, warn.conflicts = FALSE) # warn.conflicts = FALSE is to avoid clash)
library(data.table)
library(tidyr)
library(dslabs)
library(dplyr)
library(ggplot2)
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:data.table':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
library(stringr)
library(recosystem)
library(Rcpp)
```

```
## Warning: package 'Rcpp' was built under R version 4.1.1
```

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

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

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

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

1.3.3. Data Pre-processing

```
# if using R 3.6 or earlier use below code:
# 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")
```

1.4. Key Steps

The main key steps are the followings:
- Data cleaning and exploration
- Splitting the data into separate trainset and testset to design and test the algorithm: edx; validation; train_set; test_set
- Developing a model attempting to train our dataset using several algorithms to choose a good model: Regression; Regularization; Matrix factorization algorithms

a. Naive Model
b. Movie Effect Model
c. A Linear Regression Algorithm_Movie Model
d. User Effect Model
e. Linear Regression Algorithm_User Model
f. Regularized Model
g. Matrix Factorization Model

- Comparing the RMSE results
- Conclusion

2. Method and Analysis

The techniques used in the report are regression, regularization, and matrix factorization algorithms. The evaluation method used is the RMSE (Root Mean Square Error). RMSE is defined below.

```
#Define RMSE_Root Mean Squared Error
RMSE<-function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings-predicted_ratings)^2, na.rm=T))
}
```

2.1. Data Preparation and Wrangling (Data Cleaning and Modification)

In this section, we will download and prepare the dataset to be used for our analysis. We will split the dataset into two parts: training set called “edx” and the test(evaluation) set called “validation” with 90% and 10% of the original dataset respectively. The edx set will be used for training and testing, and the validation set is used for evaluating of the final model. During the process, data cleaning will be done by removing the unnecessary files from the working directory.

Let’s build up the code provided from edx capstone project module continuously. MovieLens 10M Dataset will be divided into two subsets: “edx” for building algorithms; “validation” for the final evaluation.

```
# Generate the validation set (final hold-out test set)
# 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)`
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

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

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

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
```

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

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

rm(test_index, temp, removed)
```

Let’s check whether our columns can be readable or not.

```
head(edx,2)
```

```
##    userId movieId rating timestamp           title                genres
## 1:      1     122      5 838985046 Boomerang (1992)        Comedy|Romance
## 2:      1     185      5 838983525  Net, The (1995) Action|Crime|Thriller
```

The timestamp cannot be readable at the first glance, and title may be split into title and release year. We will clean and modify our data into more readable one.

Beforehand, We will check whether there are NAs and outliers. Then, modify our data, and further split edx into train_set and test_set for the purpose of training and testing in the process of building models. Finally, we will sample from train_set because of the smooth running in my machine due to the lack of efficiency of machine.

NA

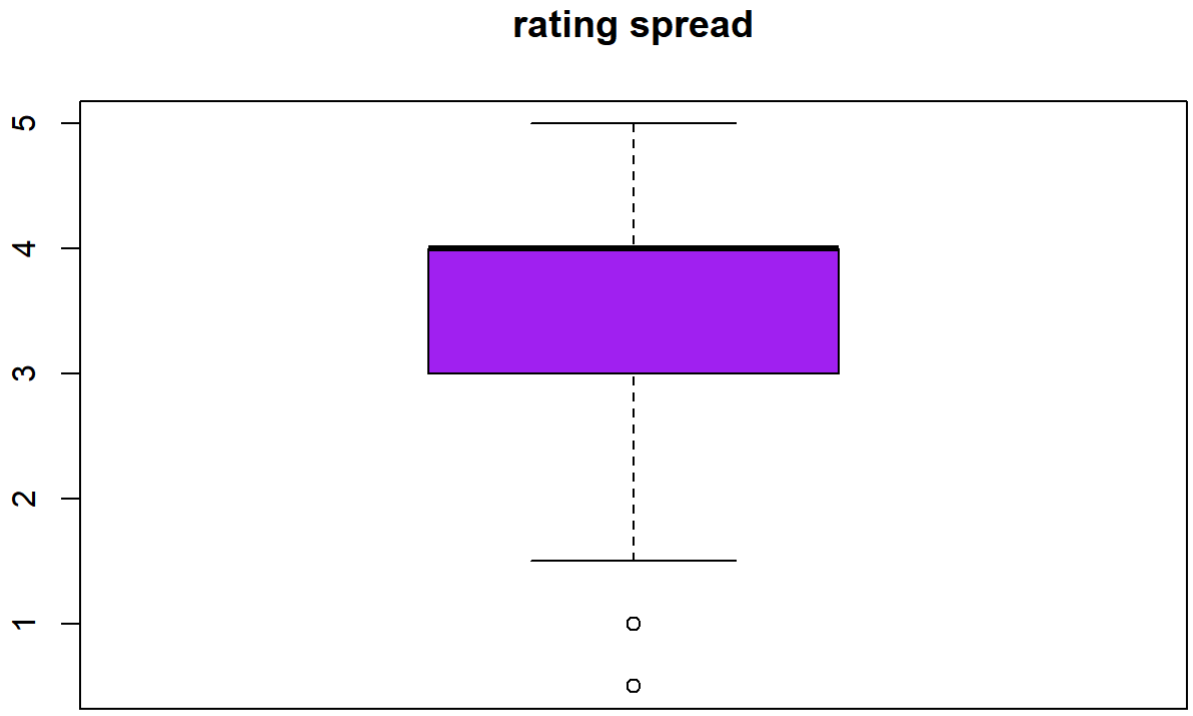
```
# Check there is any missing value
anyNA(edx)
```

```
## [1] FALSE
```

Outliers

Ratings are made on a 5-star scale, with half-star increments (0.5 stars - 5.0 stars).

```
# Check outliers of rating. For visualization, we can use the following code.
boxplot(edx$rating, main="rating spread", col="purple")
```



There seems outliers below 1. Let’s check exactly. We will consider values as outliers if the value of rating is less than 0.5 and greater than 5. Let’s check how many outliers are.

```
# Let's check the number of outliers.
sum(edx$rating<0.5| edx$rating>5)
```

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

There are no outliers.

timestamp

The timestamp column represents seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.). We will convert it into a familiar datetime format making datetime column and removing timestamp column.

```
edx_tidy<-edx%>%mutate(datetime=as.POSIXct(edx$timestamp,origin = "1970-01-01",tz = "UTC"))%>%sele
ct(-timestamp)
head(edx_tidy)
```

```
##    userId movieId rating                         title
## 1:      1     122      5              Boomerang (1992)
## 2:      1     185      5               Net, The (1995)
## 3:      1     292      5               Outbreak (1995)
## 4:      1     316      5               Stargate (1994)
## 5:      1     329      5 Star Trek: Generations (1994)
## 6:      1     355      5       Flintstones, The (1994)
##                            genres            datetime
## 1:                Comedy|Romance 1996-08-02 11:24:06
## 2:          Action|Crime|Thriller 1996-08-02 10:58:45
## 3:  Action|Drama|Sci-Fi|Thriller 1996-08-02 10:57:01
## 4:       Action|Adventure|Sci-Fi 1996-08-02 10:56:32
## 5: Action|Adventure|Drama|Sci-Fi 1996-08-02 10:56:32
## 6:       Children|Comedy|Fantasy 1996-08-02 11:14:34
```

title

We sill saperate title column into title and releaseyear columns. Along with the movie name there is the year which the movie was released. This can be seperated into 2 different features.

```
#library(stringr)
#extract release year from title
pattern <- "(?<=WW()WWd{4}(?=WW))"
edx_tidy$releaseyear <- edx_tidy$title %>% str_extract(pattern) %>% as.integer()

#extract only title without release year from title column removing redundant columns
edx_tidy%>%mutate(title = as.character(str_sub(edx_tidy$title,1, -7)))# the title is from the firs
t to the seventh place from the end
```

```
##          userId movieId rating                 title
##       1:      1     122    5.0             Boomerang
##       2:      1     185    5.0              Net, The
##       3:      1     292    5.0              Outbreak
##       4:      1     316    5.0              Stargate
##       5:      1     329    5.0 Star Trek: Generations
##      ---
## 9000051:  32620   33140    3.5         Down and Derby
## 9000052:  40976   61913    3.0           Africa addio
## 9000053:  59269   63141    2.0 Rockin' in the Rockies
## 9000054:  60713    4820    2.0  Won't Anybody Listen?
## 9000055:  64621   39429    2.5                Confess
##                                 genres            datetime releaseyear
##       1:                Comedy|Romance 1996-08-02 11:24:06        1992
##       2:         Action|Crime|Thriller 1996-08-02 10:58:45        1995
##       3:  Action|Drama|Sci-Fi|Thriller 1996-08-02 10:57:01        1995
##       4:       Action|Adventure|Sci-Fi 1996-08-02 10:56:32        1994
##       5: Action|Adventure|Drama|Sci-Fi 1996-08-02 10:56:32        1994
##      ---
## 9000051:               Children|Comedy 2007-03-10 21:39:07        2005
## 9000052:                   Documentary 2008-11-27 06:32:08        1966
## 9000053:        Comedy|Musical|Western 2008-11-11 22:41:58        1945
## 9000054:                   Documentary 2005-06-19 04:52:34        2000
## 9000055:                Drama|Thriller 2008-01-25 08:03:02        2005
```

splitting

We have already splitted movielens dataset into edx and validation. The “edx” dataset will be divided again into two subsets: “train_set” for training algorithms, “test_set” for testing the algorithms in the process of developing model.

```
# splitting edx_tidy dataset into train_set and test_set
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
# if using R 3.5 or earlier, use `set.seed(123)`
test_index2 <- createDataPartition(y = edx_tidy$rating, times = 1, p = 0.2, list = FALSE)
train_set <- edx_tidy[-test_index2,]
test_set <- edx_tidy[test_index2,]

test_set <- test_set %>% semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

sampling

We will use the sampling from train_set dataset in case there will come across limitation and slowness due to the lack of efficiency of my computer.

```
#Sampling from train_set
train_set_sample <- train_set[sample(nrow(train_set), 100000, replace = FALSE),]
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
# if using R 3.5 or earlier, use `set.seed(123)`
test_index3 <- createDataPartition(y = train_set$rating, times = 1, p = 0.2, list = FALSE)
train_set_sample <- train_set_sample[-test_index3,]
```

```
## Warning in `[.data.table`(train_set_sample, -test_index3, ): Item 19884 of i is
## -100003 but there are only 100000 rows. Ignoring this and 1420125 more like it
## out of 1440009.
```

```
test_set_sample <- train_set_sample[test_index3,]

test_set_sample <- test_set_sample %>% semi_join(train_set_sample, by = "movieId") %>%
  semi_join(train_set_sample, by = "userId")
```

```
dim(train_set)
```

```
## [1] 7200043       7
```

```
dim(train_set_sample)
```

```
## [1] 80117     7
```

```
dim(test_set_sample)
```

```
## [1] 15926     7
```

Until now, we have modified, cleaned, and split our dataset. Our train_set and test_set is tidy and ready for analysis and modeling. If we need, train_set_sample, test_set_sample datasets are ready for analysis as well.

2.2. EDA

In this section, we will explore our dataset using simple functions and statistics to understand and to find out the distribution and the relations of variables. We will construct some codes for visualization of our dataset to gain information and insights. We will create charts, table, and graphs, which might help to build our model.

2.2.1. Overviewing

```
str(edx_tidy)
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  7 variables:
##  $ userId     : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId    : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating     : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ title      : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)"
...
##  $ genres     : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "A
ction|Adventure|Sci-Fi" ...
##  $ datetime   : POSIXct, format: "1996-08-02 11:24:06" "1996-08-02 10:58:45" ...
##  $ releaseyear: int  1992 1995 1995 1994 1994 1994 1994 1994 1994 1994 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

We are able to see edx has 6 columns:
- userId(integer): the user information
- movieId(numeric): the desired outcome from 0.5 to 5 at intervals of 0.5.
- rating(numeric): the movie information
- title(character): the movie information
- genres(character):a few categories to which the movie belonging
- datetime(numeric) : rating time
- releaseyear: year to release each movie

Size of the dataset

```
edx_tidy%>% summarize(n_rating=n(), n_movies=n_distinct(movieId), n_users=n_distinct(userId))
```

```
##   n_rating n_movies n_users
## 1  9000055    10677   69878
```

- The number of row is 9000055.
- The number of unique movies is 10677.
- The number of unique users is 69878.

(insight)
The numbers indicate not every user rated every movie. If all users rated every movie, the number of row is 746,087,406 by multiplying the above two numbers: 69878 * 10677, but we know the number of rows is 9000055.

Summary

```
summary(edx_tidy)
```

```
##      userId         movieId          rating         title          
##  Min.   :    1   Min.   :    1   Min.   :0.500   Length:9000055    
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   Class :character  
##  Median :35738   Median : 1834   Median :4.000   Mode  :character  
##  Mean   :35870   Mean   : 4122   Mean   :3.512                     
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000                     
##  Max.   :71567   Max.   :65133   Max.   :5.000                     
##     genres             datetime                    releaseyear   
##  Length:9000055     Min.   :1995-01-09 11:46:49   Min.   :1915  
##  Class :character   1st Qu.:2000-01-01 23:11:23   1st Qu.:1987  
##  Mode  :character   Median :2002-10-24 21:11:58   Median :1994  
##                     Mean   :2002-09-21 13:45:07   Mean   :1990  
##                     3rd Qu.:2005-09-15 02:21:21   3rd Qu.:1998  
##                     Max.   :2009-01-05 05:02:16   Max.   :2008
```

The average rating of all movies is 3.512.

2.2.2. Variables

1) MovieId

```
movie_sum <- edx_tidy %>% group_by(movieId) %>%
  summarize(n_rating_of_movie = n(),
            mu_movie = mean(rating))
head(movie_sum)
```

```
## # A tibble: 6 x 3
##   movieId n_rating_of_movie mu_movie
##     <dbl>             <int>    <dbl>
## 1       1             23790     3.93
## 2       2             10779     3.21
## 3       3              7028     3.15
## 4       4              1577     2.86
## 5       5              6400     3.07
## 6       6             12346     3.82
```

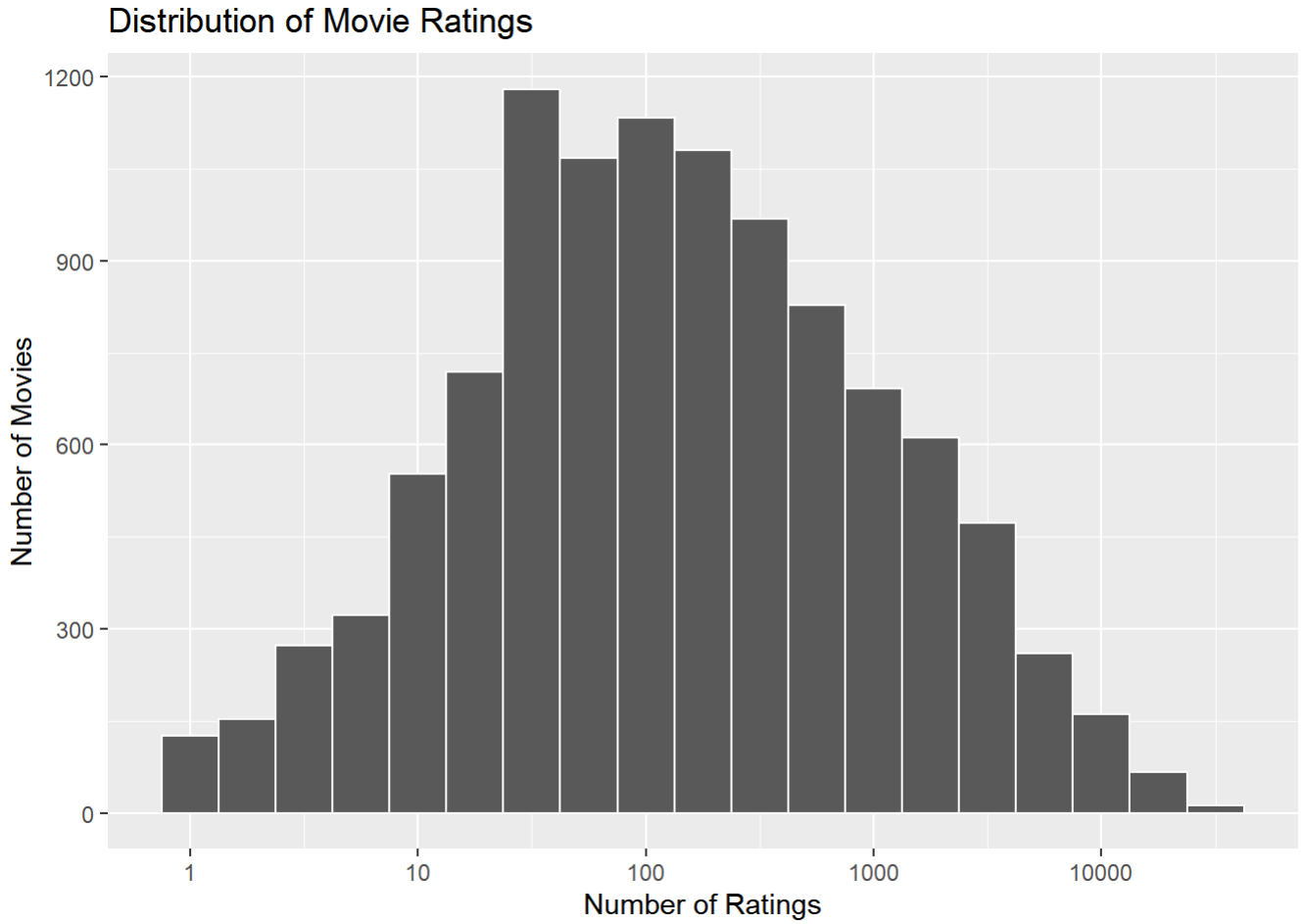
```
n_distinct(edx_tidy$movieId)
```

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

In the movieId column, there are 10,677 different movies in the edx dataset. Some are rated more than others. Each move is represented by a movieId.

Distribution of MovieId

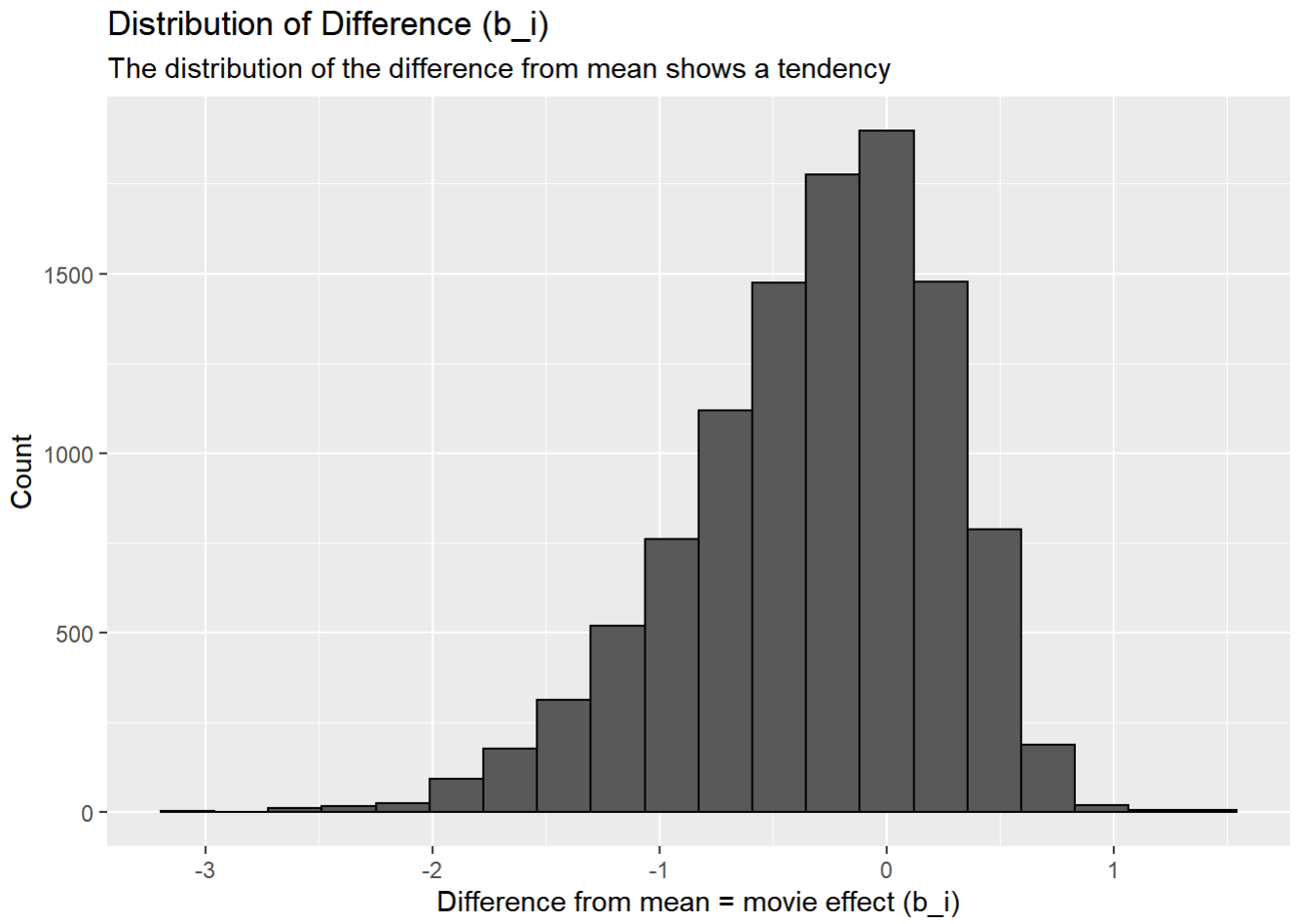
```
edx_tidy %>% group_by(movieId) %>%
  summarise(n_rating=n()) %>%
  ggplot(aes(n_rating)) +
    geom_histogram(binwidth = 0.25, color = "white") +
    scale_x_log10() +
    ggtitle("Distribution of Movie Ratings") +
    xlab("Number of Ratings") +
    ylab("Number of Movies")
```



Majority of movies have been rated approximately between 50 to 150 times by log10 unit. The ratings based on different movies are shown a distribution although it is not symmetric. So we may be able to consider movieId as a predictor considering the difference from mean rating.

Distribution of the Movie Effect (Difference from mean)

```
mu<-mean(edx_tidy$rating)
movie_mean_norm <- edx_tidy %>%
  group_by(movieId) %>%
  summarize(movie_effect = mean(rating - mu))
movie_mean_norm %>% qplot(movie_effect, geom ="histogram", bins = 20, data = ., color = I("black"
)) +
      ggtitle("Distribution of Difference (b_i)",
            subtitle = "The distribution of the difference from mean shows a tendency") +
    xlab("Difference from mean = movie effect (b_i)") +
    ylab("Count")
```



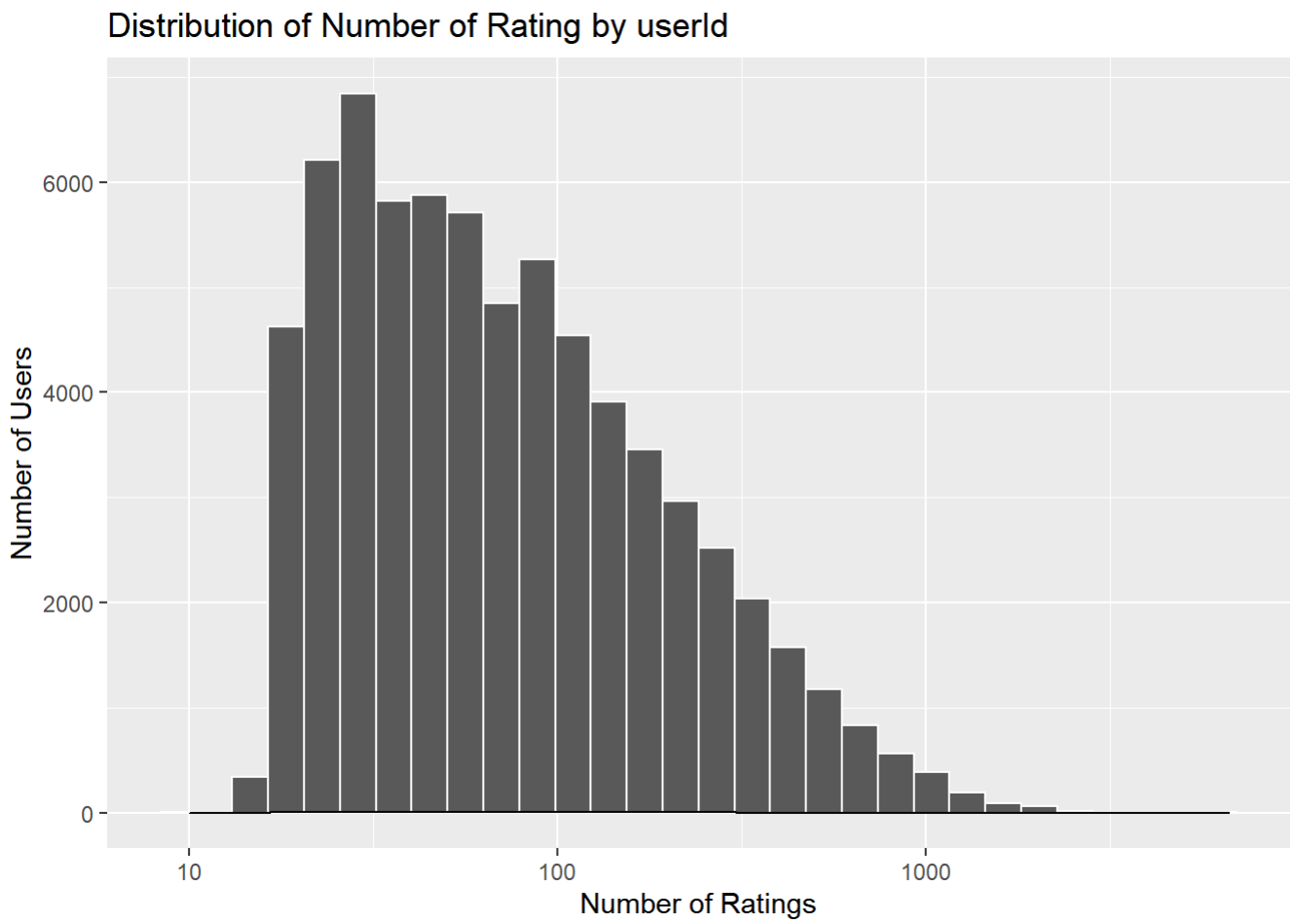
Different moives are rated differently. The above histogram shows although the distribution is not symmetric, there are surely relations between difference from mean and rating. We can consider this movie effect as a predictor.

2) UserId

Distribution of userId

```
# Histogram of User Ratings
edx_tidy %>% group_by(userId) %>%
  summarise(n=n()) %>%
  ggplot(aes(n)) +
    geom_histogram(color = "white") +
    scale_x_log10() +
    ggtitle("Distribution of Number of Rating by userId") +
    xlab("Number of Ratings") +
    ylab("Number of Users") +
    geom_density()
```

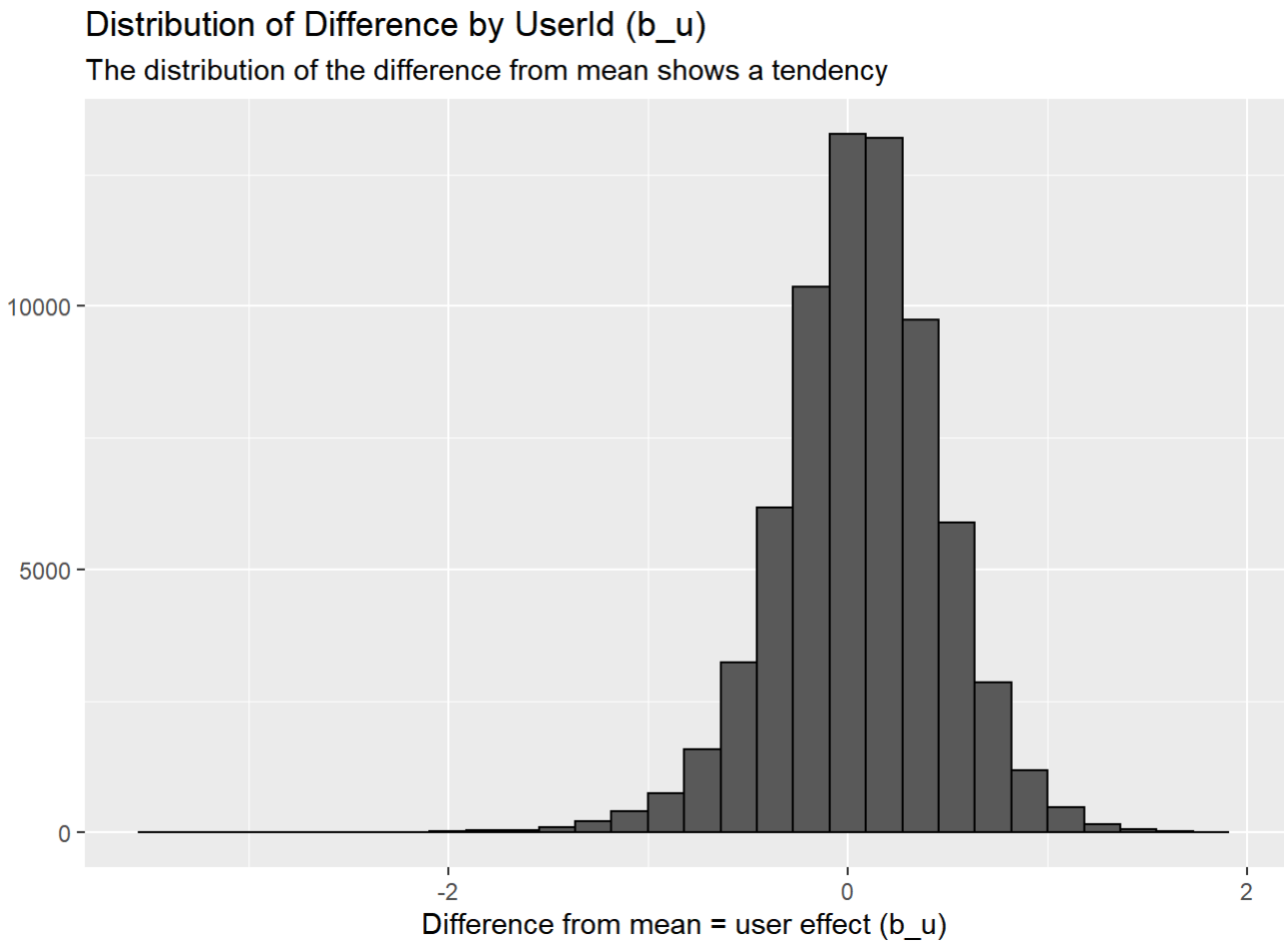
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



The majority of users have rated approximately between 30 to 120 by log10 unit. Some users rate a few movies, whereas other users rate more than one thousand. The ratings based on different users are shown a distribution although it is not symmetric. So we may be able to consider users as a predictor considering the difference from mean rating.

Distribution of the User Effect (Difference from user mean)

```
user_mean_norm <- edx_tidy %>%
  left_join(movie_mean_norm, by='movieId') %>%
  group_by(userId) %>%
  summarize(user_effect = mean(rating - mu - movie_effect))
user_mean_norm %>% qplot(user_effect, geom ="histogram", bins = 30, data = ., color = I("black"))+
        ggtitle("Distribution of Difference by UserId (b_u)",
        subtitle = "The distribution of the difference from mean shows a tendency") +
         xlab("Difference from mean = user effect (b_u)")
```

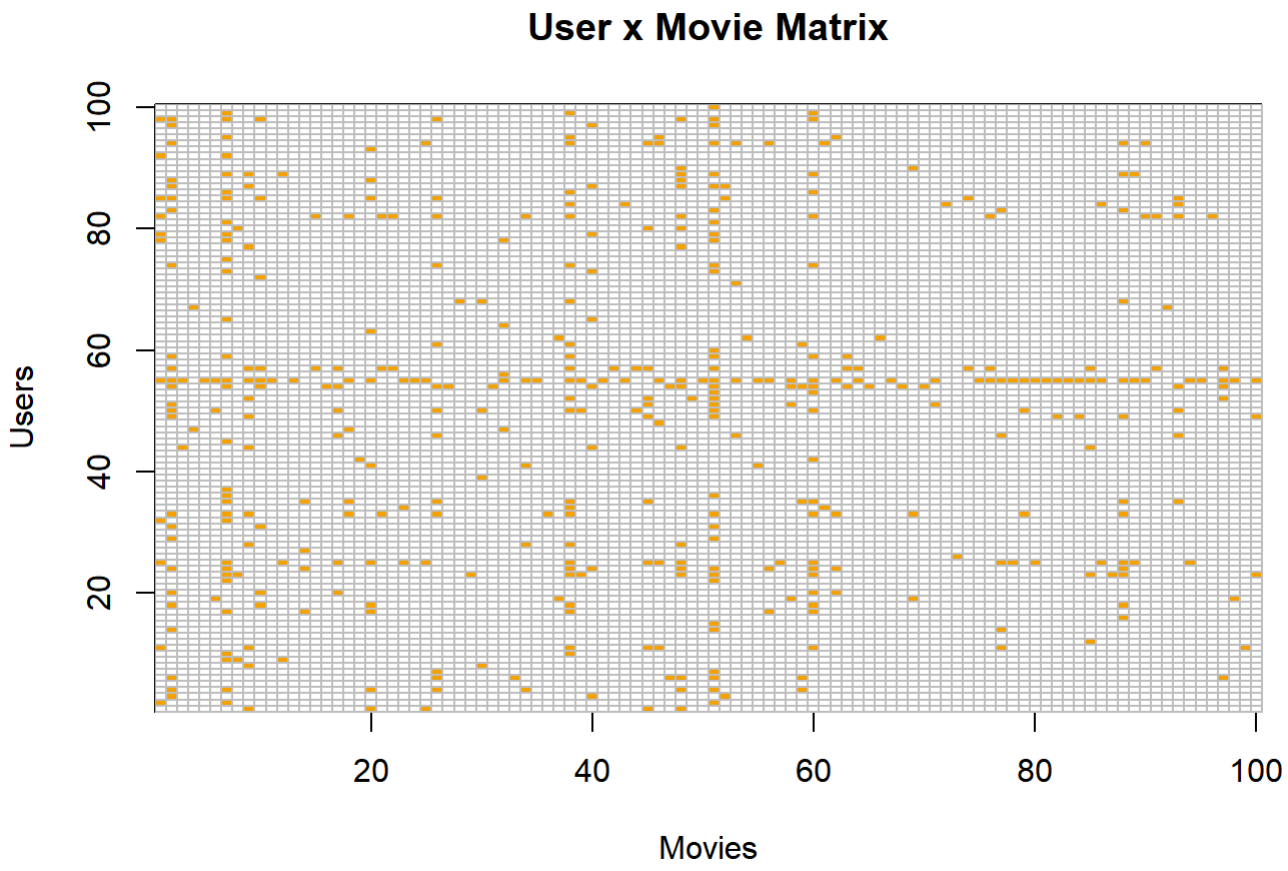


Different users are rated differently. The above histogram shows although the distribution is not symmetric, there are surely relations between difference from mean by userId and rating. We can consider this user effect as a predictor. Now movie effet and user effect are apparent. we consider movieId and userId as predictors.

Matrix between movieId and userId

If we see users in a large matrix, we can use the below code chunk.

```
users <- sample(unique(edx_tidy$userId), 100)
edx_tidy%>% filter(userId %in% users) %>%
  select(userId, movieId, rating) %>%
  mutate(rating = 1) %>%
  spread(movieId, rating) %>%
  select(sample(ncol(.), 100)) %>%
  as.matrix() %>% t(.) %>%
  image(1:100, 1:100,. , xlab="Movies", ylab="Users")
abline(h=0:100+0.5, v=0:100+0.5, col = "grey")
title("User x Movie Matrix")
```



This matrix is from a random sample of 100 movies and 100 users. The yellow indicate a user and move combination for which is a rating. This user-movie matrix is sparse uncovering the majority of empty cells and indicating the majority of users rate very few movies. We may think our task of recommendation system as filling in the NAs. In addition, the matrix also shows four to five vertical lines indicating specific movies have more ratings. It also shows a few horizontal lines indicating some users are more active than others.

3) rating

There are 10 possible values in the rating that the users have the option to choose from it. The below table shows 10 values of rating and the number of each ratings.

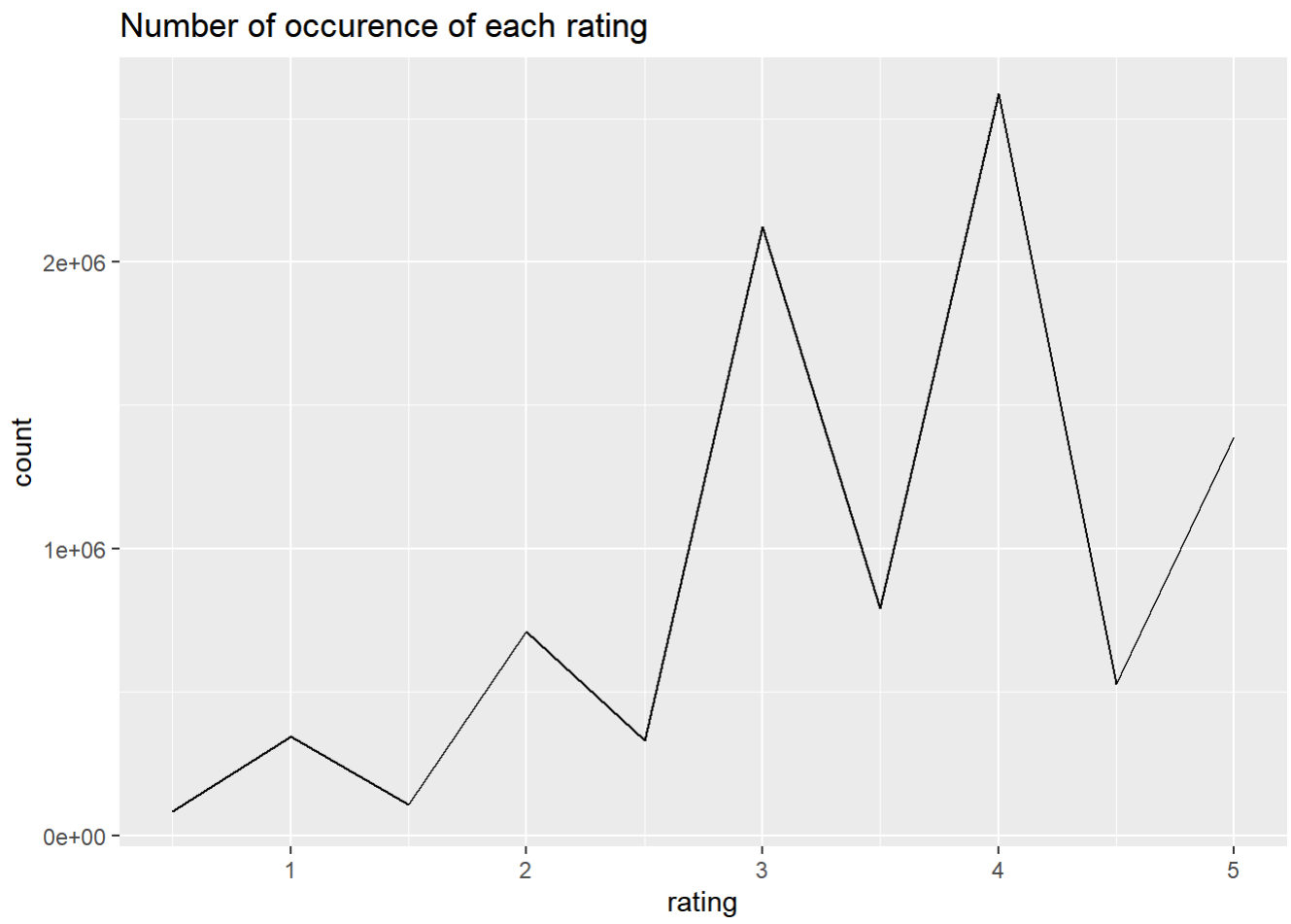
```
edx_tidy %>% group_by(rating) %>% summarize(count = n()) %>% arrange(desc(count))
```

```
## # A tibble: 10 x 2
##    rating   count
##     <dbl>   <int>
##  1    4   2588430
##  2    3   2121240
##  3    5   1390114
##  4    3.5  791624
##  5    2    711422
##  6    4.5  526736
##  7    1    345679
##  8    2.5  333010
##  9    1.5  106426
## 10    0.5   85374
```

In the rating column, there are 0 zeros whereas 2121240 threes in the edx_clean dataset. The five most given ratings from most to least are 4, 3, 5, 3.5, 2.

Distribution of rating

```
rating_count<-edx_tidy %>%
  group_by(rating) %>%
  summarize(count = n()) %>%
  ggplot(aes(x = rating, y = count)) +
  geom_line()+
  ggtitle("Number of occurence of each rating")
rating_count
```

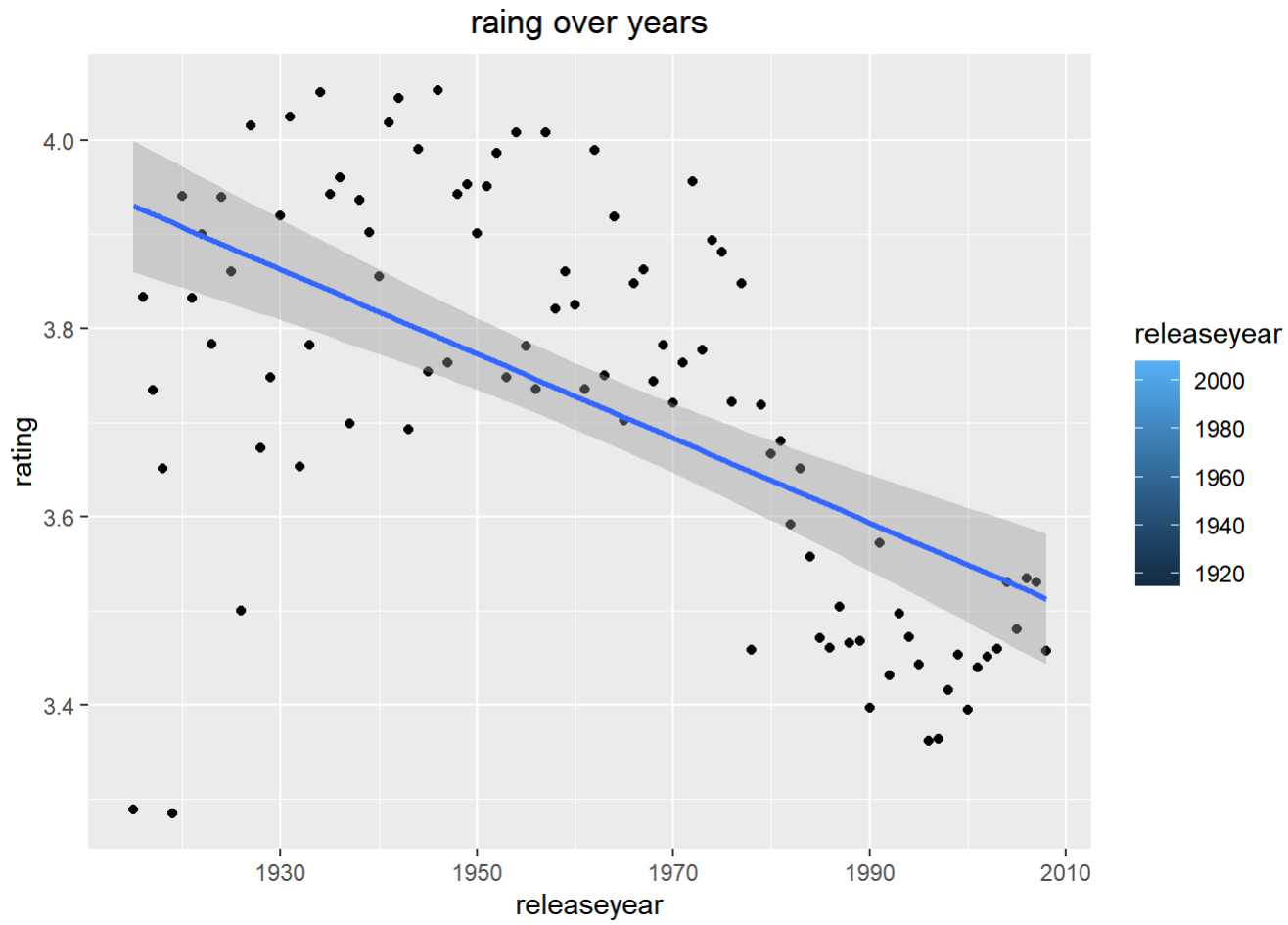


The most comman rating is 4 and the least common is 0. The positive rating is common because the graph is toward higher number of rating. The whole ratings such as 3, 4, 5 is common than half ratings such as 2.5, 3.5, 4.5.

4) releaseyear

```
# show relation between releaseyear and rating
edx_tidy %>% group_by(releaseyear) %>%
summarize(rating = mean(rating)) %>%
ggplot(aes(releaseyear, rating, fill = releaseyear)) +
  geom_point() +
  geom_smooth(method = "lm" ) +
  ggtitle("raing over years") +
  theme(plot.title = element_text(hjust = 0.5))
```

```
## `geom_smooth()` using formula 'y ~ x'
```



There is a decreasing tendency which indicates that people tend to rate less over time.

5) genres

The dataset provides the list of genres for each movie. It contains 797 different combinations of genre sets.

```
# Number of different combinations of genre sets
edx_tidy%>% summarize(n_genres=n_distinct(genres))
```

```
##   n_genres
## 1      797
```

```
# Examples of the combinations
edx_tidy%>%group_by(genres)%>%
summarise(n=n()) %>%
head()
```

```
## # 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
```

The table shows that most movies are classified in more than one genre. In this project, we will not use genres as a predictor due to inefficiency despite this information which may be used to make a better prediction. In explanation, it takes long time to compute in my computer.

3. Modeling

Modeling Method

We will use the typical error loss, the root mean squared error (RMSE) on a test_set to decide our algorithm is good. If RMSE is larger than 1, it indicates our typical error is larger than one star, which is not good.

The methods are:
a. to use the mean of ratings for baseline comparison (Naive Model)
b. to try regression model using lm() function using caret package
c. to try regression model using difference from mean (Movie Effect Model; User Effect Model)
d. to try regression linear model with regularisation (Regularized Model)
e. to apply a Matrix Factorization algorithm (Matrix Factorization Model)

Loss function

Our target is already set: the RMSE of our model should be less than 0.86490. Let’s check how to describe RMSE assessments.

The RMSE is defined as the following to evaluate the models:

```
#Define RMSE_Root Mean Squared Error
RMSE<-function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings-predicted_ratings)^2, na.rm=T))
}
```

3.1. A Naive Model

Let’s come up with the simplest model. We will predict the rating for all movies and users without considering other variability. The prediction will calculated with the true rating for all movies and users added with independent errors sampled from the same distribution centered at zero and the average of all ratings.

```
# Average of rating
mu<-mean(train_set$rating)

# Estimate RMSE
naive_rmse<-RMSE(test_set$rating, mu)

# Make a results table of the RMSE value and save the results to keep track of it for the sake of
 comparison with results of other models.
rmse_results <- tibble(Method = "Model 1: Mean Effect (Naive Model)", RMSE = naive_rmse)
rmse_results %>% knitr::kable()
```

Method	RMSE
Model 1: Mean Effect (Naive Model)	1.059904

This baseline prediction is to predict all ratings naively comparing with just mean of ratings. The result is 1.059904, which means the prediction in this way is off by about 1. This results is not good.

3.2. Movie Effect Model

Let’s add movie variability called the movie effect here to our Naive Model. Some movies are rated higher than others whereas others are not. Different movies are rated differently maybe due to the different popularity among users or time. We can build up from the previous model adding this movie effect, the average rating for a movie. This effect is called “bias”. Specifically, We will call this value as b_i to represent the bias of movie. We will estimate with the average value. Then, We will try to use lm() function of a linear regression algorithm with the least squares to estimate the movie bias.

```
# Movie Effect algorithm using mean statistic
mu<-mean(train_set$rating)
movie_avgs<-train_set%>%
  group_by(movieId)%>%
  summarise(b_i=mean(rating-mu))

predicted_ratings <- mu+test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

# Estimate RMSE
movie_rmse<-RMSE(predicted_ratings, test_set$rating)

# RMSE table
rmse_results <- bind_rows(rmse_results,
                          tibble(Method = "Model 2: Movie Effect",
                                 RMSE = movie_rmse))
rmse_results %>% knitr::kable()
```

Method	RMSE
Model 1: Mean Effect (Naive Model)	1.0599043

Method	RMSE
Model 2: Movie Effect	0.9437429

We can see some improvement but still we need to go further to narrow the gap toward the target rmse.

3.3. A Linear Regression Algorithm_Movie Model

```
# The first try to use a linear regression algorithm using lm() function
fit<-lm(rating~movieId, data=train_set)
y_hat <- fit$coef[1] + fit$coef[2]*test_set$movieId

# Estimate RMSE
regression_movie_rmse<-RMSE(test_set$rating, y_hat)

# RMSE table
rmse_results <- bind_rows(rmse_results,
                          tibble(Method = "Model 3: Linear Regression Model_Movie",
                                 RMSE = regression_movie_rmse))
rmse_results %>% knitr::kable()
```

Method	RMSE
Model 1: Mean Effect (Naive Model)	1.0599043
Model 2: Movie Effect	0.9437429
Model 3: Linear Regression Model_Movie	1.0598875

The result is not good compared to Model 2 (Movie Effect Model).

3.4. User Effect Model

Let’s add user variability called the User Effect to the previous Movie Effect Model. Some users are rated more frequently than others. Different users rated differently maybe due to their tendency of rating toward movies, other personal bias or preference. Some love every movie whereas some are very cranky. A cranky user may rate a great movie much less than the average rate such as 3 or 3.5 rather than 5. We can build up this idea adding the average user for a movie. This effect consideration is called “bias” as well. Specifically, we will call this value as b_u to represent the bias for user. We will compute the average rating for user that have rated over 100 movies for the efficiency. Then, we will try to use lm() function of a linear regression algorithm with the least squared to estimate this user bias.

```
# User Effect algorithm calculating the user average effect based on the movie effect algorithm
mu<-mean(train_set$rating)
movie_avgs<-train_set%>%
  group_by(movieId)%>%
  summarise(b_i=mean(rating-mu))

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

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)

# RMSE
user_rmse<-RMSE(predicted_ratings, test_set$rating)

rmse_results <- bind_rows(rmse_results,
                          tibble(Method = "Model 4: Movie + User Effect",
                                 RMSE = user_rmse))

rmse_results %>% knitr::kable()
```

Method	RMSE
Model 1: Mean Effect (Naive Model)	1.0599043
Model 2: Movie Effect	0.9437429
Model 3: Linear Regression Model_Movie	1.0598875
Model 4: Movie + User Effect	0.8659320

We got a better result with RMSE o.865932.

3.5. A Linear Regression Algorithm_User Model

```
# A try to use a linear regression algorithm using lm() function
fit2<-lm(rating~userId + movieId, data=train_set)
y_hat2 <- fit2$coef[1] + fit2$coef[2]*test_set$userId

# Estimate RMSE
regression_user_rmse<-RMSE(test_set$rating, y_hat2)

# RMSE table
rmse_results <- bind_rows(rmse_results,
                          tibble(Method = "Model 5: Linear Regression Model_User",
                                 RMSE = regression_user_rmse))
rmse_results %>% knitr::kable()
```

Method	RMSE
Model 1: Mean Effect (Naive Model)	1.0599043
Model 2: Movie Effect	0.9437429
Model 3: Linear Regression Model_Movie	1.0598875
Model 4: Movie + User Effect	0.8659320
Model 5: Linear Regression Model_User	1.0599063

We will not use lm() because of poor RMSE results compared to calculation of average models.

3.6. Trying Genre Effect Model

We tried the code the following but we couldn't get the result of the Genre Effect Model due to the need of more large size of space, 1031.2Gb. So we will not add more predictors anymore due to the restriction of my computer.

< # Genre Effect algorithm calculating the genre effect based on the user effect algorithm >
genre_avgs <- train_set %>% left_join(movie_avgs, by='movieId') %>% left_join(user_avgs, by='userId') %>% group_by(genres)

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

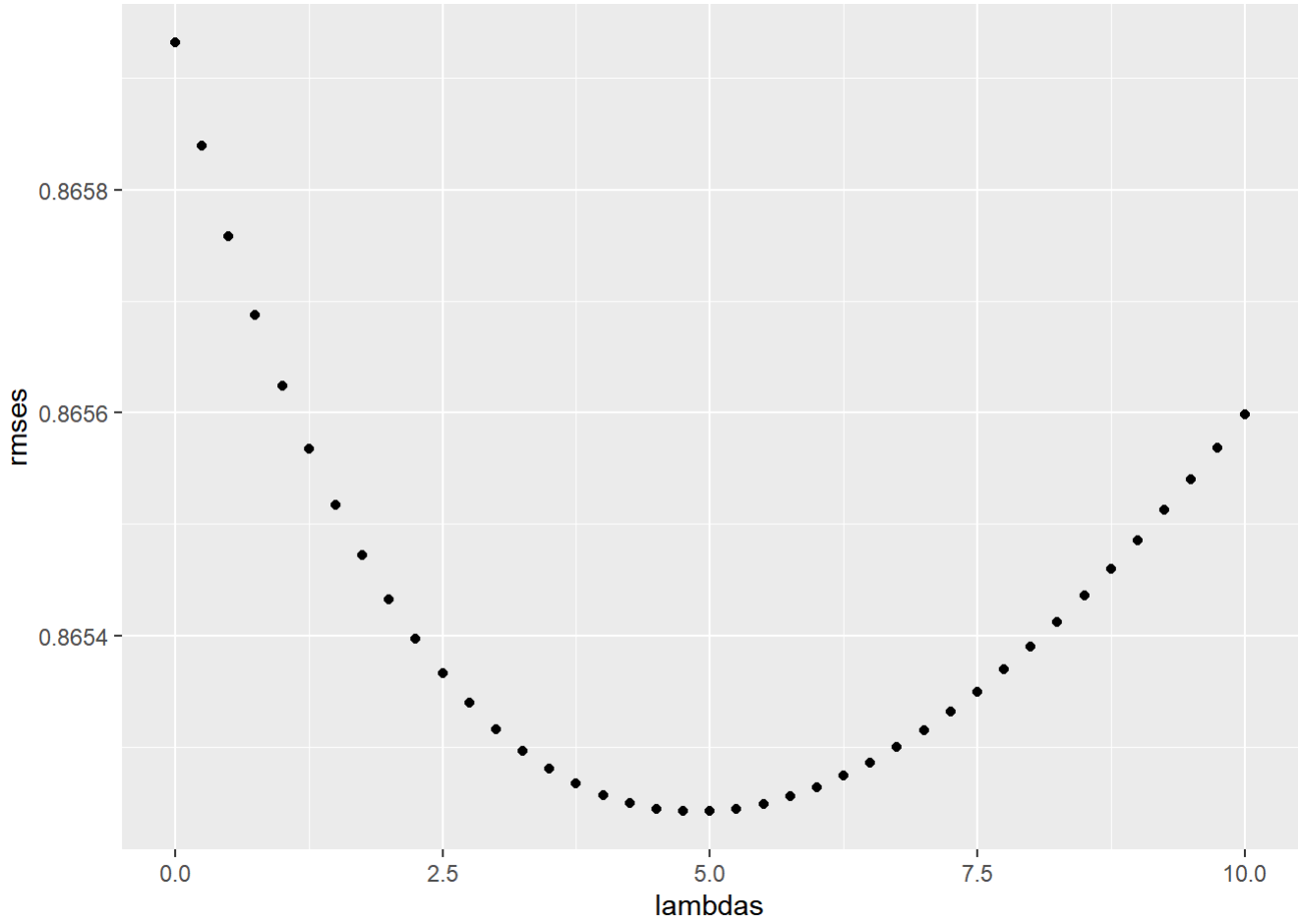
< # RMSE >
genre_rmse<-RMSE(predicted_ratings, test_set$rating) rmse_results <- bind_rows(rmse_results, tibble(Method = "Model 6: Movie + User + Genre Effect", RMSE = genre_rmse))
rmse_results

3.7. Regularization Model

Regularization allows us to penalize large estimates coming from using small sample sizes. We will add penalty not to be shrunk prediction to our model. We will use cross-validation to choose it. Then, we will use the value which makes minimum RMSE to compute the regularized estimates. Then, we will add this penalty to our prediction.

Regularized Movie + User effects

```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){

mu <- mean(train_set$rating)

b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+l))

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

predicted_ratings <- test_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%

  mutate(pred = mu + b_i + b_u ) %>%
  .$pred
  return(RMSE(predicted_ratings, test_set$rating)) })
qplot(lambdas, rmses)
```



```
# Use the lambda which minimises the RMSE to train the model and predict the test_set
lambda<-lambdas[which.min(rmses)]

mu <- mean(train_set$rating)

movie_reg_avgs<-train_set%>%
  group_by(movieId)%>%
  summarise(b_i=sum(rating-mu)/(n()+lambda))

user_reg_avgs <- train_set %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), b_u = sum(rating - mu - b_i)/(n()+lambda), n_i =
 n())

predicted_ratings <- mu + test_set %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  left_join(user_reg_avgs, by = "userId") %>%
  pull(b_u)

# RMSE
reg_genre_rmse<- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble (Method="Model 7: Regularized_Movie + User Effect",
                                  RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

Method	RMSE
Model 1: Mean Effect (Naive Model)	1.0599043
Model 2: Movie Effect	0.9437429
Model 3: Linear Regression Model_Movie	1.0598875
Model 4: Movie + User Effect	0.8659320
Model 5: Linear Regression Model_User	1.0599063
Model 7: Regularized_Movie + User Effect	0.8652421

The RMSE has improved slightly with RMSE 0.8652421. so the regularized model is the best than other models until now.

3.8. Matrix Factorization

We will use recosystem package so we have to arrange our dataset into matrix form saving as tables and selecting movieId, userId, rating variables.

```
# Fyi, it took 15mins to run below chunk of code in my computer.
# Select movieId, userId, and rating variables only
edx_fac <- edx_tidy %>% select(movieId, userId, rating)
validation_fac <- validation %>% select(movieId, userId,  rating)
train_set_fac<-train_set%>%select (movieId, userId,  rating)
test_set_fac<-test_set%>%select (movieId, userId, rating)

# Arrange the datasets into matrix forms
edx_fac <- as.matrix(edx_fac)
validation_fac <- as.matrix(validation_fac)
train_set_fac <- as.matrix(train_set_fac)
test_set_fac <- as.matrix(test_set_fac)

# Save the datasets as tables
write.table(edx_fac, file = "edxset.txt", sep = " ", row.names = FALSE,
            col.names = FALSE)
write.table(validation_fac, file = "validationset.txt", sep = " ",
            row.names = FALSE, col.names = FALSE)
write.table(train_set_fac, file = "trainset.txt", sep = " ", row.names = FALSE,
            col.names = FALSE)
write.table(test_set_fac, file = "testset.txt", sep = " ", row.names = FALSE,
            col.names = FALSE)
set.seed(1)
edx_dataset <- data_file("edxset.txt")
trainset_dataset <- data_file("trainset.txt")
testset_dataset <- data_file("testset.txt")
validation_dataset <- data_file("validationset.txt")

# Create a model object
r = Reco() # this will create a model object

# Tune the algorithm to find the optimal answer
opts = r$tune(trainset_dataset, opts = list(dim = c(10, 20, 30), lrate = c(0.1,
    0.2), costp_l1 = 0, costq_l1 = 0, nthread = 1, niter = 10))
```

```
# Train the model using the tuned parameters
r$train(trainset_dataset, opts = c(opts$min, nthread = 1, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9945   1.0045e+07
##    1       0.8790   8.0888e+06
##    2       0.8464   7.4953e+06
##    3       0.8241   7.1480e+06
##    4       0.8071   6.8963e+06
##    5       0.7939   6.7171e+06
##    6       0.7831   6.5772e+06
##    7       0.7738   6.4653e+06
##    8       0.7659   6.3725e+06
##    9       0.7592   6.2975e+06
##   10       0.7533   6.2332e+06
##   11       0.7480   6.1790e+06
##   12       0.7433   6.1303e+06
##   13       0.7390   6.0882e+06
##   14       0.7350   6.0511e+06
##   15       0.7314   6.0177e+06
##   16       0.7280   5.9856e+06
##   17       0.7249   5.9607e+06
##   18       0.7220   5.9357e+06
##   19       0.7192   5.9127e+06
```

```
stored_prediction = tempfile()

# Predict on the testset_dataset
r$predict(testset_dataset, out_file(stored_prediction))
```

```
## prediction output generated at C:WUsersWuserWAppDataWLocalWTempWRtmpG0fu0lWfile7245ec14a6f
```

```
real_ratings <- read.table("testset.txt", header = FALSE, sep = " ")$V3
pred_ratings <- scan(stored_prediction)

# RMSE
matrix_rmse <- RMSE(real_ratings, pred_ratings)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method = "Model 8: Matrix Factorization Model",
                                     RMSE = matrix_rmse ))
```

```
## Warning: `data_frame()` was deprecated in tibble 1.1.0.
## Please use `tibble()` instead.
```

```
rmse_results %>% knitr::kable()
```

Method	RMSE
Model 1: Mean Effect (Naive Model)	1.0599043
Model 2: Movie Effect	0.9437429

Method	RMSE
Model 3: Linear Regression Model_Movie	1.0598875
Model 4: Movie + User Effect	0.8659320
Model 5: Linear Regression Model_User	1.0599063
Model 7: Regularized_Movie + User Effect	0.8652421
Model 8: Matrix Factorization Model	0.7906779

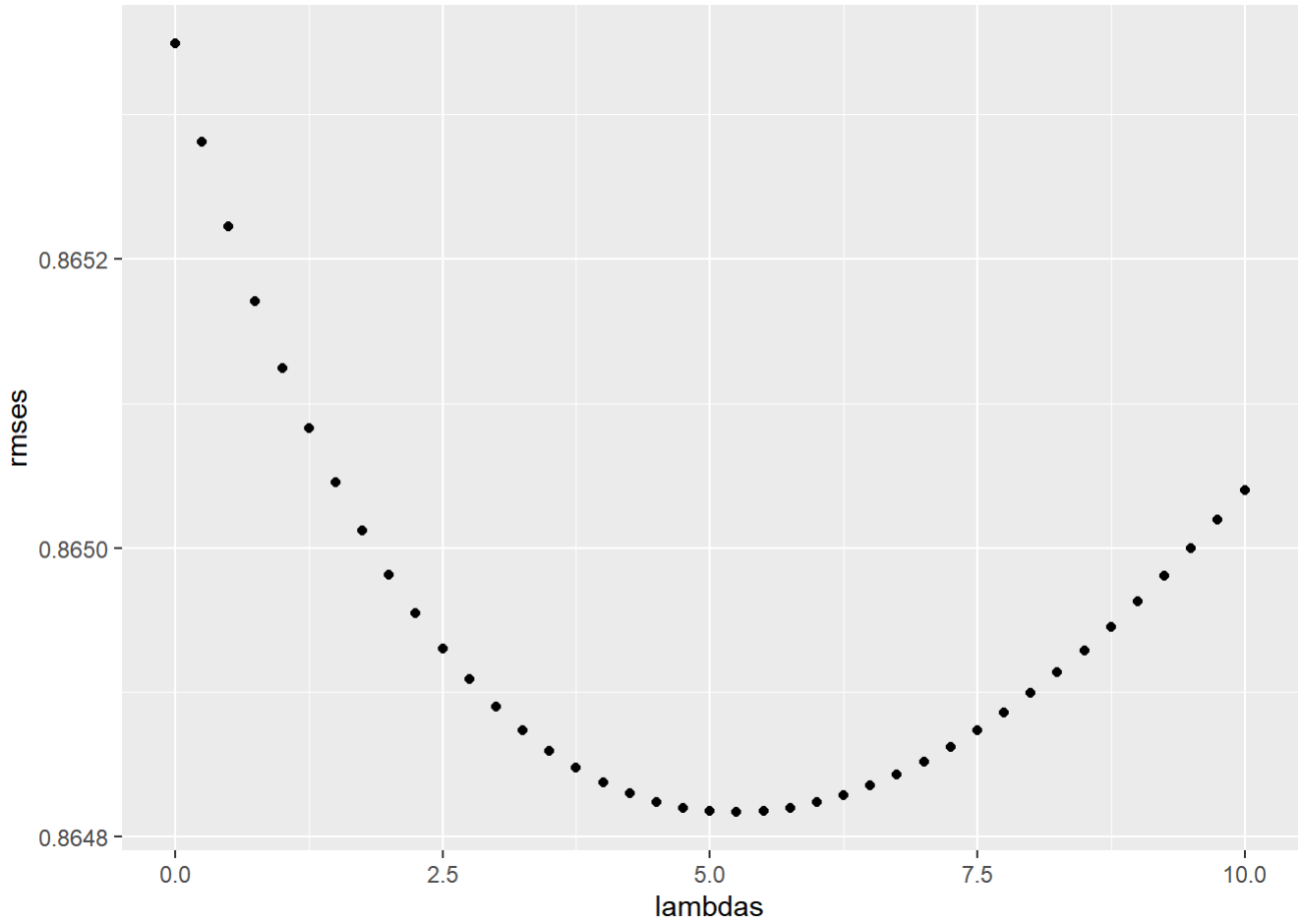
The RMSE result, around 0.7906779 shows that we has reached our goal with Model 8 along with 0.8652421 with Model 7.

4. Results

Let's apply Regularized Movie and User Effect Model (Model 7) and the Matrix Factorization Model(Model 8) to edx dataset and Validation dataset that was given to us.

Regularized Movie and User Effect Model(Model 7)

```
# Fyi, it took around 10 mins with my computer to run the below chunk of codes.
lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas, function(l){
mu <- mean(edx$rating)
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+l))

b_u <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+l))


predicted_ratings <- validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%

  mutate(pred = mu + b_i + b_u ) %>%
  .$pred
  return(RMSE(predicted_ratings, validation$rating)) })
qplot(lambdas, rmses)
```




```
lambda<-lambdas[which.min(rmses)]
lambda
```

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

```
lambda <- 5.25
mu <- mean(edx$rating)

movie_reg_avgs<-edx%>%
  group_by(movieId)%>%
  summarise(b_i=sum(rating-mu)/(n()+lambda))

user_reg_avgs <- edx %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), b_u = sum(rating - mu - b_i)/(n()+lambda), n_i =
 n())

predicted_ratings <- mu + validation %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  left_join(user_reg_avgs, by = "userId") %>%
  pull(b_u)
reg_genre_rmse<- RMSE(predicted_ratings, validation$rating)
reg_genre_rmse
```

```
## [1] 1.055805
```

```
rmse_results <- bind_rows(rmse_results,
                          tibble (Method="The Final Good Model: Regularized_Movie + User Effect",
                                  RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

Method	RMSE
Model 1: Mean Effect (Naive Model)	1.0599043
Model 2: Movie Effect	0.9437429
Model 3: Linear Regression Model_Movie	1.0598875
Model 4: Movie + User Effect	0.8659320
Model 5: Linear Regression Model_User	1.0599063
Model 7: Regularized_Movie + User Effect	0.8652421
Model 8: Matrix Factorization Model	0.7906779
The Final Good Model: Regularized_Movie + User Effect	0.8648170

Matrix Factorization Model(Model 8)

```
# Fyi, it took # Fyi, it took around 20 mins with my computer to run the below chunk of codes.

# Tune the algorithm to find the optimal answer
opts = r$tune(edx_dataset, opts = list(dim = c(10, 20, 30), lrate = c(0.1,
    0.2), costp_l1 = 0, costq_l1 = 0, nthread = 1, niter = 10))

# Train the model using the tuned parameters
r$train(edx_dataset, opts = c(opts$min, nthread = 1, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9739   1.2064e+07
##    1       0.8732   9.9033e+06
##    2       0.8398   9.1836e+06
##    3       0.8181   8.7688e+06
##    4       0.8019   8.4798e+06
##    5       0.7897   8.2801e+06
##    6       0.7799   8.1275e+06
##    7       0.7718   8.0034e+06
##    8       0.7649   7.9078e+06
##    9       0.7589   7.8247e+06
##   10       0.7538   7.7585e+06
##   11       0.7492   7.7015e+06
##   12       0.7452   7.6535e+06
##   13       0.7415   7.6084e+06
##   14       0.7381   7.5703e+06
##   15       0.7351   7.5374e+06
##   16       0.7322   7.5061e+06
##   17       0.7296   7.4766e+06
##   18       0.7272   7.4530e+06
##   19       0.7249   7.4305e+06
```

```
stored_prediction = tempfile()

# Predict on the validation_dataset
r$predict(validation_dataset, out_file(stored_prediction))
```

```
## prediction output generated at C:WUsersWuserWAppDataWLocalWTempWRtmpG0fu0lWfile72424864351
```

```
real_ratings <- read.table("validationset.txt", header = FALSE, sep = " ")$V3
pred_ratings <- scan(stored_prediction)

# RMSE
best_rmse <- RMSE(real_ratings, pred_ratings)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method = "The Final Best Model: Matrix Factorization Model",
                                     RMSE = best_rmse ))
rmse_results %>% knitr::kable()
```

Method	RMSE
Model 1: Mean Effect (Naive Model)	1.0599043
Model 2: Movie Effect	0.9437429
Model 3: Linear Regression Model_Movie	1.0598875
Model 4: Movie + User Effect	0.8659320
Model 5: Linear Regression Model_User	1.0599063
Model 7: Regularized_Movie + User Effect	0.8652421
Model 8: Matrix Factorization Model	0.7906779
The Final Good Model: Regularized_Movie + User Effect	0.8648170
The Final Best Model: Matrix Factorization Model	0.7829194

As we can see the RMSE results, the final good model, the Regularized Movie and User Effect Model shows 0.8648170 and the Matrix Factorization Model shows 0.7829194. Both is above our target 0.86490.

5. Conclusion

5.1. Summary

This project is part of the capstone course for the professional certificate in data science at HarvardX. The MovieLens 10M Dataset was used for developing a movie recommendation system with reasonable accuracy. Before developing algorithms, exploratory data analysis was used in order to see relations among variables. RMSE, the Root Mean Square Error was used to evaluate models as to how close the predictions were to the real values. The lowest RMSE was considered as the best one. The good models were evaluated again with the hold-out validation dataset as if it was not known and trained. The techniques used in this report were calculating average values, regression, regularization, and matrix factorization algorithms. Considering the results among eight models, the best model among 8 models was Matrix Factorization Model with the RMSE result of 0.7829194 followed by a good model, Regularization Model with the result of 0.8648170.

5.2. Limitations and Future Work

It should be noted that, however, we could not apply our predictors to regression algorithm although there was an apparent relations with rating variable, for example, year variable due to inefficiency of my computer and lack of space. Furthermore, there would be other ML algorithms that could be implemented such as Neural Networks and Evolutionary Algorithms. Also, the dataset size is also challenging with my resource. The skills of managing a big dataset as well as other efficient machine learning algorithms along with the more efficient computer would be greatly need for the future work.