MOVIE RECOMMENDATION SYSTEM

MOVIELENS PROJECT

HarvardX PH125.9x
Data Science: Capstone

SYEDA AQEEL

Contents

1. INTRODUCTION

Recommendation System predicts users' responses to given options. A widely seen example is of suggestions given to customers of online retailer based on their product searches or purchase history. This is a type of Product Recommendation. Similar systems are created for movie recommendations based on ratings provided by users. And for news articles suggestions based on the articles that readers have read in the past.

The following report explores *MovieLens* data set to model Movie Recommendation System. This document is part of Professional Certificate Program in Data Science offered by HarvardX.

The R markdown code used to generate this document is available on GitHub.

1.1 MOVIELENS DATA SET

A research lab in the Department of Computer Science and Engineering at the University of Minnesota, GroupLens Research Project, collected MovieLens data sets.

The 10M data set contains 10000054 ratings of 10681 movies by 71567 random users of the online movie recommender service MovieLens. The users are represented only by an id and no other demographic information was collected.

1.2 PROJECT OBJECTIVE

This project aims to create a recommendation system with Machine Learning algorithms for MovieLens data set. The evaluation criterion for the project is based on Root Mean Square Error (RMSE) lower than 0.8649.

Root Mean Square Error (RMSE) is defined as follows:

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

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

It will evaluate how close our estimated ratings are to observed ratings.

1.3 PROJECT FRAMEWORK

This project will follow following structure:

i. **Importing Data set**

   The data set will be loaded from MovieLens website in to RStudio interface.

ii. **Cleaning Data set**

   Data will then be divided in to two sets: edx and validation set.

   Edx set will contain 90% of original data while Validation set will have 10% entries of original set. The latter will only be used to test final algorithm.

   We'll process edx set to be in tidy format such as converting timestamp feature in to date format and extracting year of release in which movie released from its title.

iii. **Exploring and visually analysing Data set**

   Bar graphs, boxplots, error bars and other visual and statistical representations of data will aid in understanding features of data sets.

iv. **Modelling Data set**

   We'll model data set based on insights gained in data exploration. To model, we'll generate two more data sets from edx set: Train set and Test set. 90% of edx entries will be in Train set while Test set will hold only 10% of edx entries. We'll predict ratings with Test set and evaluate results with RMSE.

v. **Analysing Results**

   Results obtained through all the models we'll apply to MovieLens data will be analyzed to find minimum RMSE. The method of modelling with lowest RMSE will then be applied to final set of data (edx set) to train and validate.

vi. **Reporting results.**

   Final results will be reported to form conclusion and observe limitations.

2. METHODS & ANALYSIS

2.1 DATA CLEANING

We'll begin with loading MovieLens dataset into our RStudio interface. And split it in to two sets: edx set and validation set. We'll train and test our final model on these sets respectively. Edx set will have 90% of original entries of MovieLens data set while rest 10% entries will be in validation set.

```
####################################
#MOVIE RECOMMENDATION CAPSTONE PROJECT
####################################



# 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  3.1.6     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.1.1     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflict
s() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()

if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-proje
ct.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)
library(data.table)

# 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/rati
ngs.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")

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.see
d(1)`

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

```
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, l
ist = 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", "genr
es")

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

2.2 DATA EXPLORATION

Exploratory analysis will aid in understanding data set's construct. It will help summarize data set's main characteristics. And discover patterns and visualize these patterns.

We'll begin with observing data set's dimension.

```
#################################
# Examining edx Data Set
#################################

# Number of rows and columns in edx data set
dim(edx)

## [1] 9000055       6
```

Edx data set consists 9000055 rows and 6 columns.

```r
# Structure of the edx data set
str(edx)

## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 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 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 83
8984474 838983653 838984885 838983707 838984596 ...
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995
)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Dra
ma|Sci-Fi|Thriller" "Action|Adventure|Sci-Fi" ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

The 6 variables in data set are userId, movieId, rating, timestamp, title, and genres. Variables fall under integer, numeric or character class.

```r
#Number of distinct movies in edx data set
n_distinct(edx$movieId)

## [1] 10677
```

There are 10677 unique movies in data set.

```r
# Number of distinct users in edx data set
n_distinct(edx$userId)

## [1] 69878
```

There are 69878 unique users.

```r
# Number of distinct ratings given to movies in edx data set
n_distinct(edx$rating)

## [1] 10
```

The ratings given to movies are of 10 distinct values.

The values range from rating of 0.5 to 5.

```
# Range of ratings in edx data set
range(edx$rating)

## [1] 0.5 5.0
```

The number unique genres in data set are 797.

```
# Number of distinct genres
n_distinct(edx$genres)

## [1] 797
```

Now we’ll load all required libraries.

```
# Loading required libraries
###########################

library(dplyr)
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(ggthemes)
library(scales)

##
## Attaching package: 'scales'
## The following object is masked from 'package:purrr':
##
##     discard

## The following object is masked from 'package:readr':
##
##     col_factor
```

We’ll tidy data to ease our analysis.

Timestamp is converted in to date format. And year in which movie is released is extracted from title column.

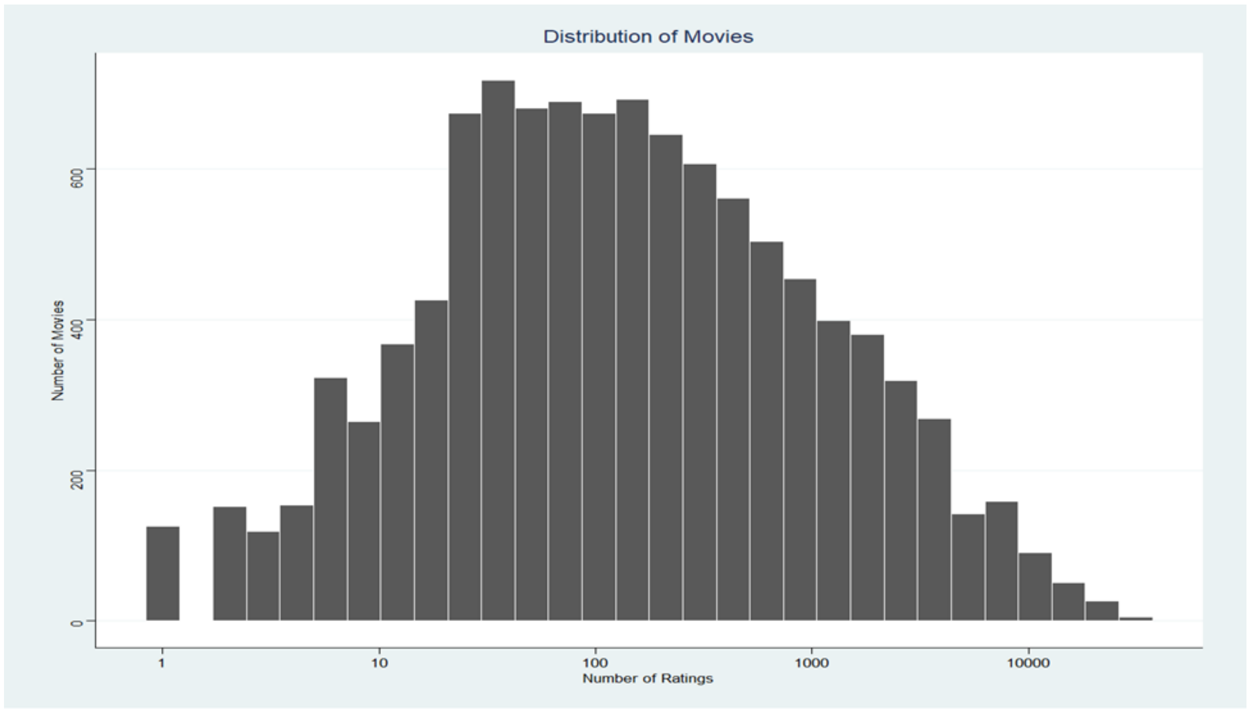
```
# Tidying edx data set
###########################

# Extracting Dates

edx <- mutate(edx, date = as_datetime(timestamp))

# Extracting year in which movie released

edx <- edx %>% mutate(title = str_trim(title)) %>%
  extract(title, c("Tp", "release"),
  regex = "^(.*) \\(([0-9 \\-]*)\\)$",remove = F)%>%
  mutate(release = if_else(str_length(release) > 4,
  as.integer(str_split(release, "-", simplify = T)[1]), as.integer(release)))
%>%
  mutate(title = if_else(is.na(Tp), title, Tp)) %>%
  select(-Tp)
```

2.3 DATA VISUALIZATION

2.3.1 MOVIE DATA

We’ll look in to movies data and visually explore trends and patterns.

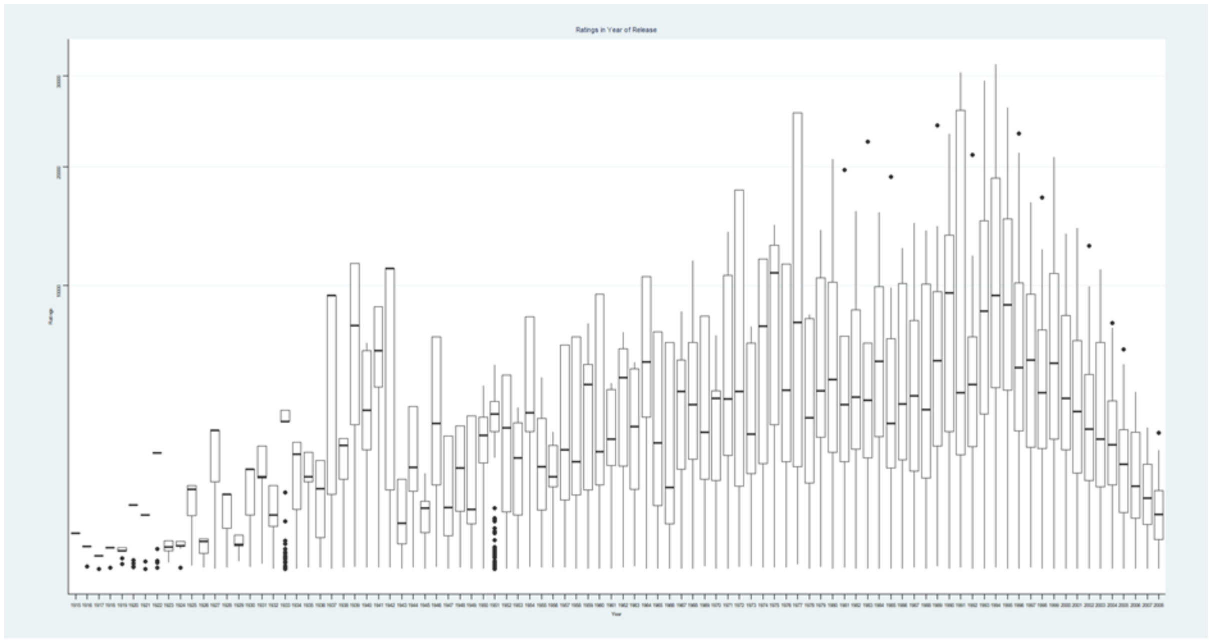
```
# Plot histogram of distribution of movies
edx %>% count(movieId) %>% ggplot(aes(n)) +
geom_histogram(bins = 30, color = "white") +
scale_x_log10() +
ggtitle("Distribution of Movies") +
xlab("Number of Ratings") +
ylab("Number of Movies") +
theme_stata()
```



The plot shows that distribution of movies with respect to their ratings follow a bell-shaped curve or normal distribution.

To further our analysis, we plotted box plot of movies in the year they released.

```
# Plot box plot of ratings for movies in release year

 edx %>% group_by(movieId) %>%
  summarize(Ratings = n(), Year = as.character(release)) %>%
  qplot(Year, Ratings, data = ., geom = "boxplot") +
  coord_trans(y = "sqrt") +
  ggtitle("Ratings in Year of Release") +
  theme_stata(base_size = 4)

## `summarise()` has grouped output by 'movieId'. You can override using the
`.groups` argument.
```

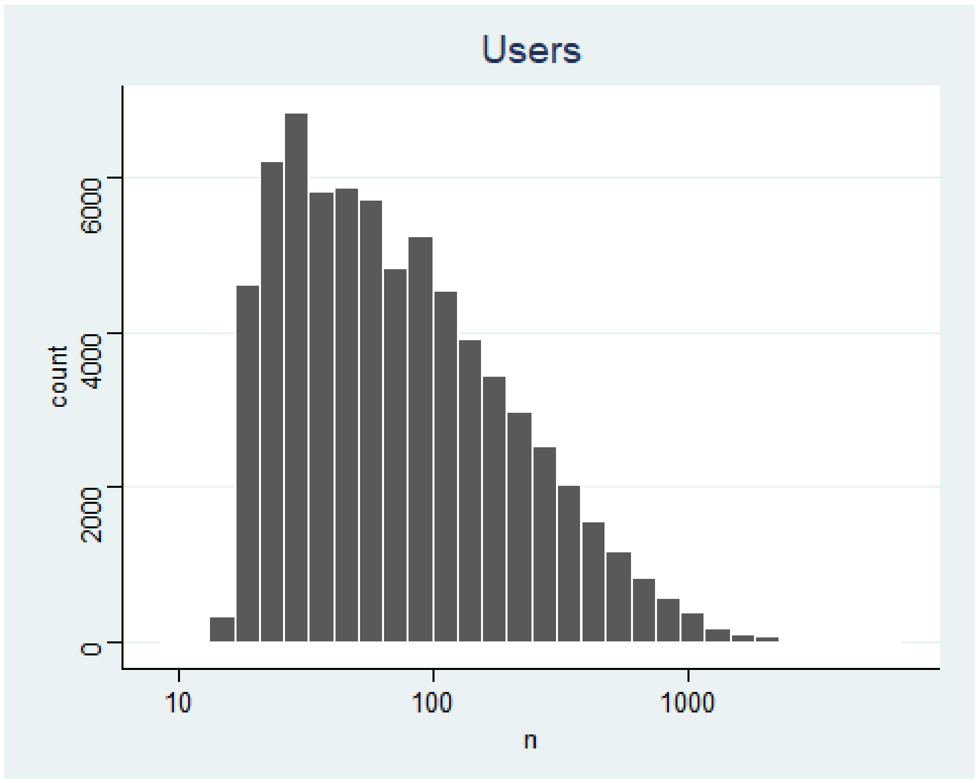


From the plot, you can see that the year with the highest median number of ratings is 1995.

2.3.2 USER DATA

Here we use visual tools to gain better insights of user data.

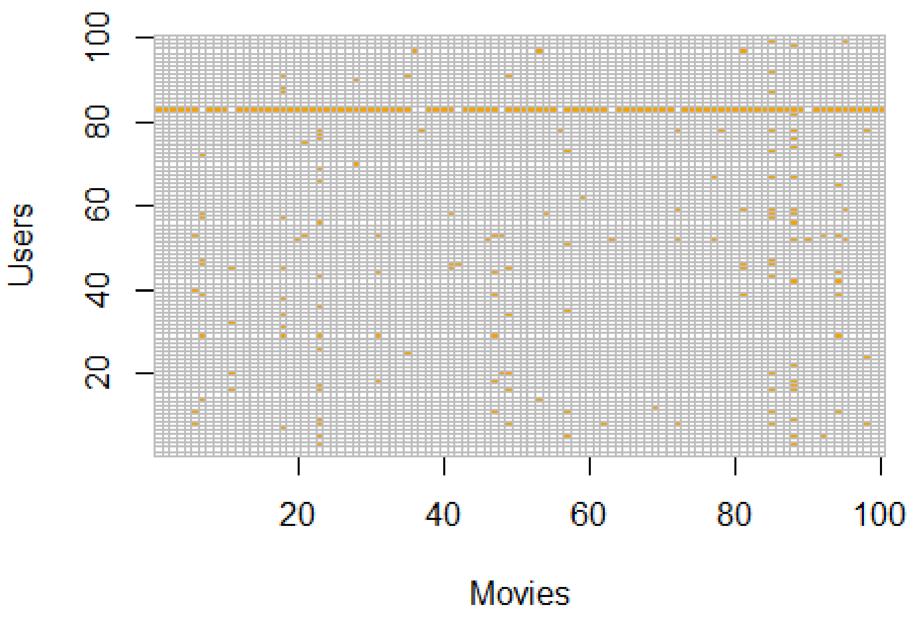
```
# Plot histogram of distribution of users

edx %>% count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bin = 30, color = "white") +
  scale_x_log10() +
  ggtitle("Users") +  theme_stata()

## Warning: Ignoring unknown parameters: bin

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



From bar chart we can conclude that number of ratings given by users don’t follow normal distribution and observations are denser toward the right tail.

To inspect if all movies were rated equally by users, we plot heatmap of users and movies.

```
# Plot heatmap of users and movies

users <- sample(unique(edx$userId), 100)
edx %>% 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")
```

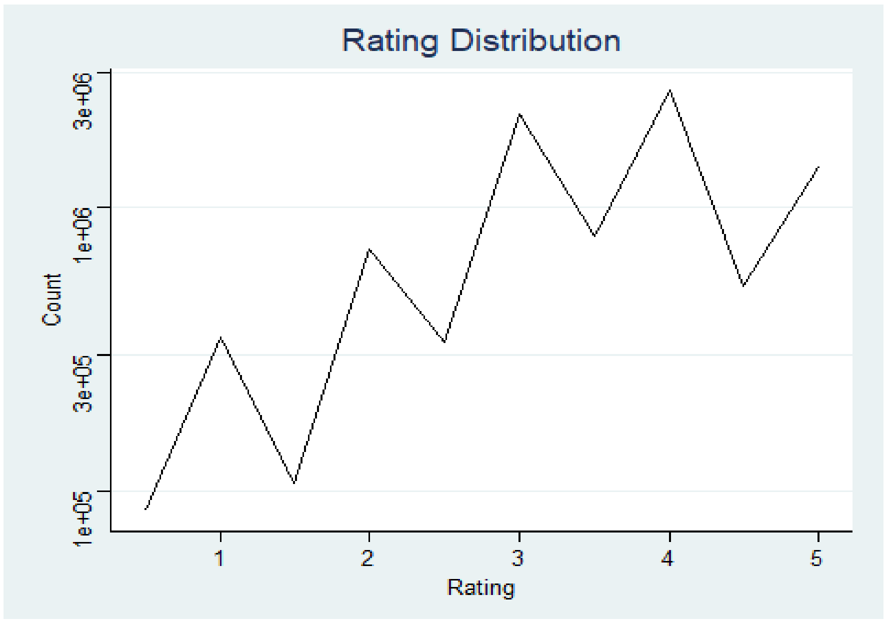


Result from plot shows that movies in data set didn’t receive equal number of ratings. Some users voted less than others.

2.3.3 RATINGS DATA

Now we look closely in to ratings data set.

```
# Distribution of ratings in edx data set

edx %>%
  group_by(rating) %>%
  summarize(count = n()) %>%
  ggplot(aes(x = rating, y = count)) +
  geom_line() +
  ggtitle("Rating Distribution") +
  scale_y_log10() +
  xlab("Rating") +
  ylab("Count") +
  theme_stata()
```



We can observe from the graph that most movies received a 3 – star or 4 – star ratings.

To quantify these ratings by type we performed following code.

```
# Count of ratings received

 edx %>% group_by(rating) %>% summarize(n=n())
```

```
## # 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
```

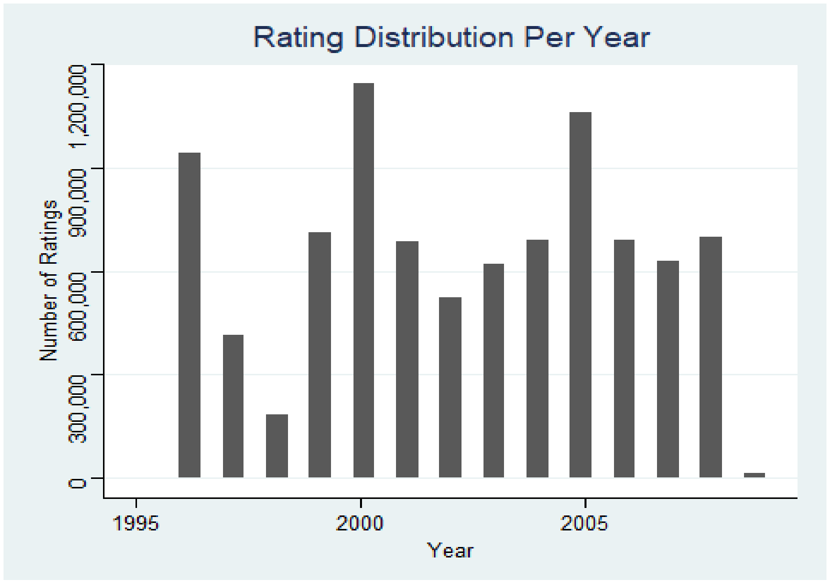
The highest type of rating that was recorded was 4- star rating with a count of 2588430.

Ratings were recorded for almost 14 years. The data set ratings record date back to year 1995.

```
# Total duration of ratings

tibble(`First Date of Record` = date(as_datetime(min(edx$timestamp), origin="
1970-01-01")),
`Last Date of Record` = date(as_datetime(max(edx$timestamp), origin="1970-01-
01"))) %>%
mutate(Duration = duration(max(edx$timestamp)-min(edx$timestamp)))

## # A tibble: 1 x 3
##   `First Date of Record` `Last Date of Record` Duration
##   <date>                 <date>                <Duration>
## 1 1995-01-09             2009-01-05            441479727s (~13.99 years)
```

To understand rating distribution more clearly, we plotted a histogram.

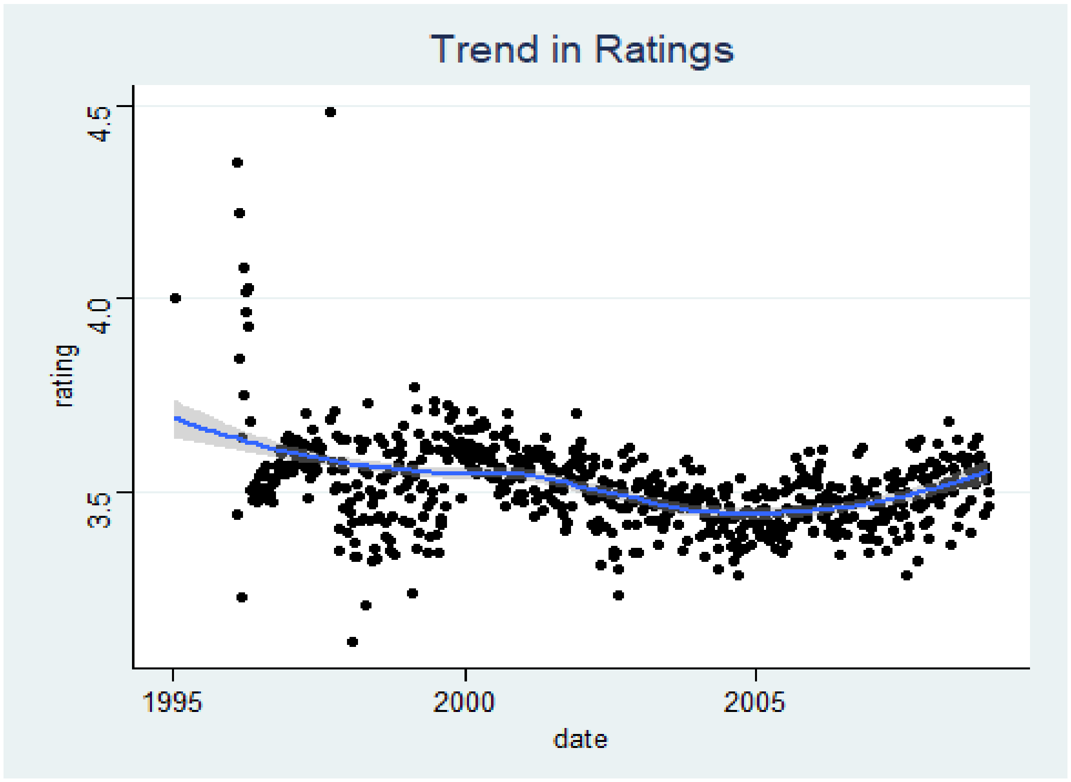
```
# Plot histogram of rating distribution per year

edx %>% mutate(year = year(date)) %>%
  ggplot(aes(year)) +
  geom_histogram() +
  ggtitle("Rating Distribution Per Year") +
  xlab("Year") +
  ylab("Number of Ratings") +
  scale_y_continuous(labels = comma) +
  theme_stata()

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



Year 2000 ranked top in receiving most ratings.

Now we'll observe trend in these ratings when rounded to nearest week.

```
# Trend in Ratings

edx %>% mutate(date = round_date(date, unit = "week")) %>%
  group_by(date) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(date, rating)) +
  geom_point() +
  geom_smooth() + ggtitle("Trend in Ratings") + theme_stata()

## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



We can see that there is some evidence of a time effect in the plot, but there is not a very strong effect of time.

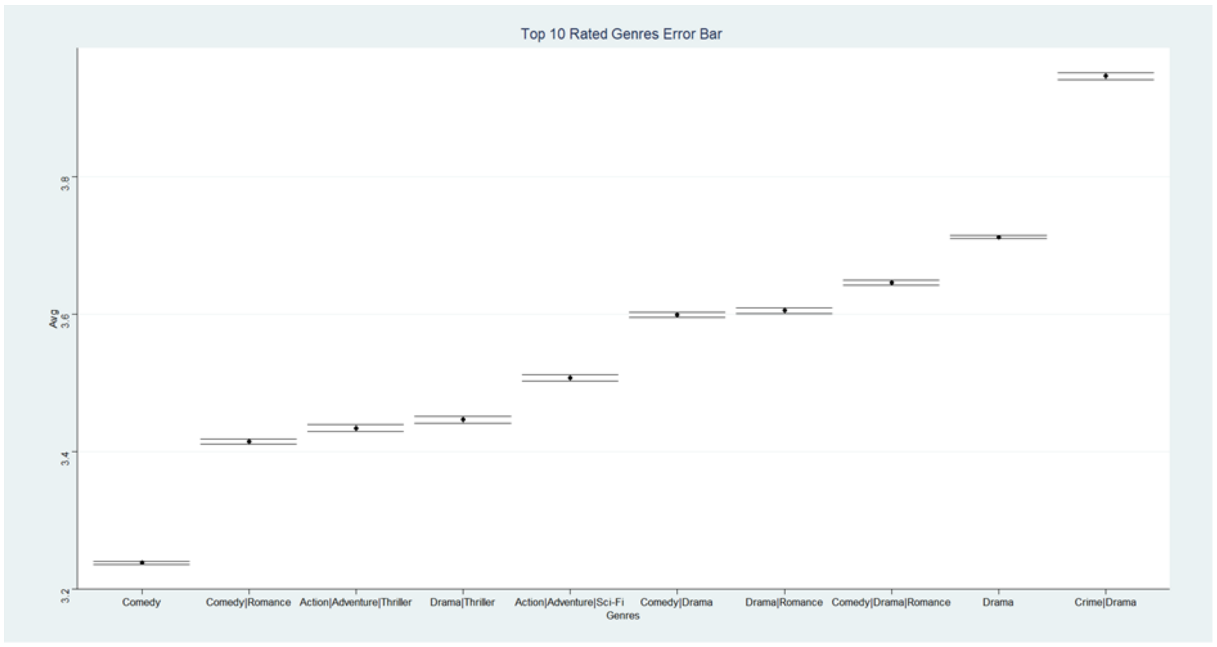
2.3.4 GENRES DATA

There are 787 unique genres in the set. Top three movie genres are Drama, Comedy, and Comedy and Romance respectively.

```
# Top 10 movie genres
edx %>% group_by(genres) %>%
  summarise(n=n()) %>%
  arrange(desc(n)) %>% top_n(10)

## Selecting by n

## # A tibble: 10 x 2
##    genres                          n
##    <chr>                       <int>
##  1 Drama                      733296
##  2 Comedy                     700889
##  3 Comedy|Romance             365468
##  4 Comedy|Drama               323637
##  5 Comedy|Drama|Romance       261425
##  6 Drama|Romance              259355
##  7 Action|Adventure|Sci-Fi    219938
##  8 Action|Adventure|Thriller 149091
##  9 Drama|Thriller             145373
## 10 Crime|Drama                137387
```

We generated Error bars of top 10 genres to graphically represent the variability in their data set.

```
# Top 10 rated genres Error Bar
edx %>% group_by(genres) %>%
  summarize(n = n(), Avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
  arrange(desc(n)) %>% slice(1:10) %>%
  mutate(Genres = reorder(genres, Avg)) %>%
  ggplot(aes(x = Genres, y = Avg, ymin = Avg - 2*se, ymax = Avg + 2*se)) +
  geom_point() +
  geom_errorbar() +
  ggtitle("Top 10 Rated Genres Error Bar ") +
  theme_stata()
```

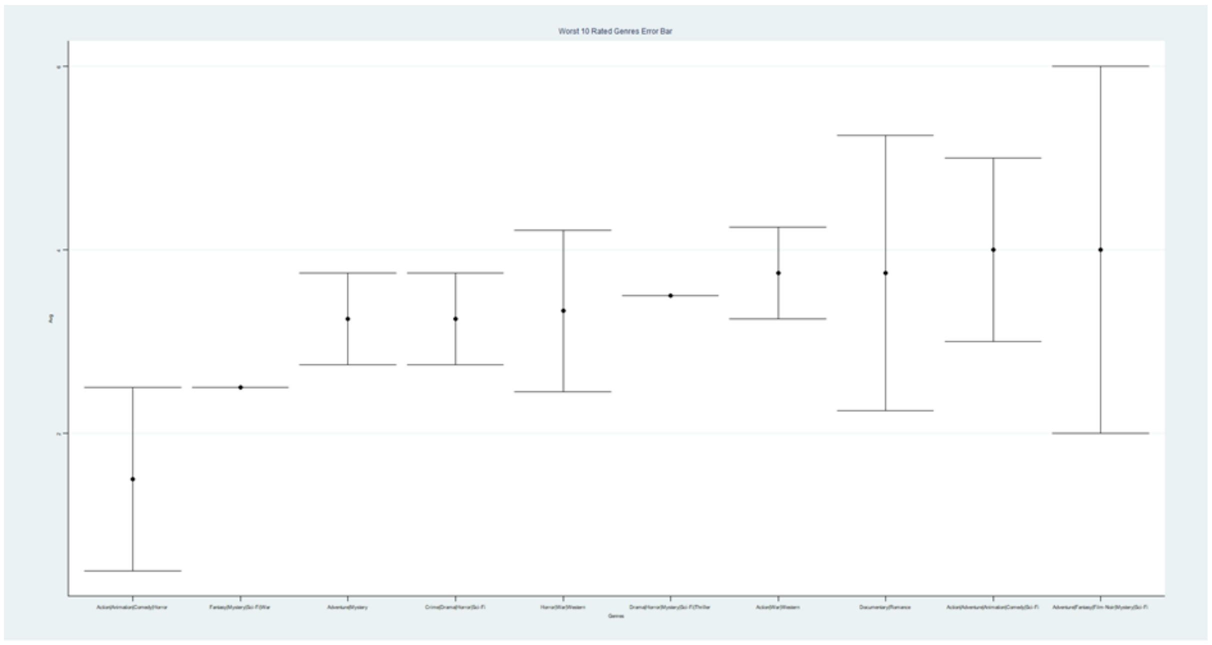


The genres which didn't do very well are as follows:

```
# 10 Worst Genres
edx %>% group_by(genres) %>%
  summarise(n=n()) %>%
  arrange(n) %>% top_n(-10)

## Selecting by n

## # A tibble: 10 x 2
##    genres                                        n
##    <chr>                                     <int>
##  1 Action|Animation|Comedy|Horror                2
##  2 Action|War|Western                            2
##  3 Adventure|Fantasy|Film-Noir|Mystery|Sci-Fi    2
##  4 Adventure|Mystery                             2
##  5 Crime|Drama|Horror|Sci-Fi                     2
##  6 Documentary|Romance                           2
##  7 Drama|Horror|Mystery|Sci-Fi|Thriller          2
##  8 Fantasy|Mystery|Sci-Fi|War                    2
##  9 Action|Adventure|Animation|Comedy|Sci-Fi      3
## 10 Horror|War|Western                            3
```

The error bars of worst ten genres show quiet a variability.

```
# Worst 10 rated genres Error Bar
edx %>% group_by(genres) %>%
        summarize(n = n(), Avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
        arrange(n) %>% slice(1:10) %>%
        mutate(Genres = reorder(genres, Avg)) %>%
     ggplot(aes(x = Genres, y = Avg, ymin = Avg - 2*se, ymax = Avg + 2*se)) +
          geom_point() +
          geom_errorbar() +
          ggtitle("Worst 10 Rated Genres Error Bar ") +
          theme_stata(base_size = 8)
```



2.4 DATA MODELING

2.4.1 Linear Model

One of the simplest approaches to build a Recommendation System is to predict same rating for all movies and users with all the differences explained by random variation. This approach is termed as *Naïve bayes approach.* Predicted ratings $\hat{y}$ are assumed to be equal to mean movie ratings µ and error term which is centered at 0.

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

One way to explain the same rating assumption is with statistical theory that mean reduces RMSE.

To explain variability in our data set more effectively, we'll weigh in movie effect bias in Naïve Bayes Approach. *Movie Effect bias* b_i captures variability in ratings because different movies are rated differently. Often blockbuster movies amass more viewership, hence are rated more often than those which are not so popular and are viewed not so frequently.

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

$$\hat{b}_i = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{\mu})$$

Similarly, different users also rate differently. We termed this as *User Effect bias* b_u. Some users more generous in giving ratings while others are not so much.

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

$$\hat{b}_u = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{\mu} - \hat{b}_i)$$

2.4.2 Regularized Movie & User Linear Model

Regularized Movie and User Linear model constrains the total variability of the movie and user effect bias. That is, it penalizes large estimates of ratings that are formed using small sample sizes of these biases. Instead of minimizing the least squares equation, we minimize an equation that adds a penalty λ.

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

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

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

where n_i is the number of ratings made for movie i.

When sample size n_i is very large, $n_i + \lambda \approx n_i$. Our estimate of bias is stable and penalty λ can be ignored. However, when n_i is small, then the estimate of bias is converged towards 0.

2.4.3 Matrix Factorization

We have described our liner model like this:

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

However, this model didn't factor in variation that is related to fact that groups of movies have similar rating patterns and groups of users have similar rating patterns as well. These patterns emerge by observing residuals.

Matrix Factorization estimates matrix of residuals r with product of matrix P and Q, where P is user rating score matrix and Q is movies rating score matrix.

$$r \approx \text{PQ}$$

3. RESULTS

Here edx set is partitioned in to two more subsets: Train set and Test set. 90 percent of edx entries are in train set whereas only 10 percent are in test set. We'll train models with train set and test them on test set.

```
# Partition edx in to Train & Test Set

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

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

# Test set is 10% of edx

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

# Ensure userId and movieId in test set are also in train set

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

# Add rows removed from test set back into train set

removed <- anti_join(temp, test_set)

## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "rele
ase", "genres", "date")

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

Model is evaluated with Root Mean Square Error (RMSE).

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

3. 1 Linear Model

We'll begin training our data set with linear model. Linear model here is estimated with this equation:

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

where μ = average ratings, b_i = movie effect, b_u= user effect, ϵ= error term centered at 0.

3.1.1 Naïve Bayes Approach

This approach models data assuming same rating for all movies and users with differences explained by random variation in error term.

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

```
mu_hat <- mean(train_set$rating)

naive_rmse <- RMSE(test_set$rating, mu_hat)

# create table to store results of approaches applied & respective RMSE

Result <- data.frame(Method = "Naive Bayes Approach", RMSE = naive_rmse)

Result

##                 Method     RMSE
## 1 Naive Bayes Approach 1.060054
```

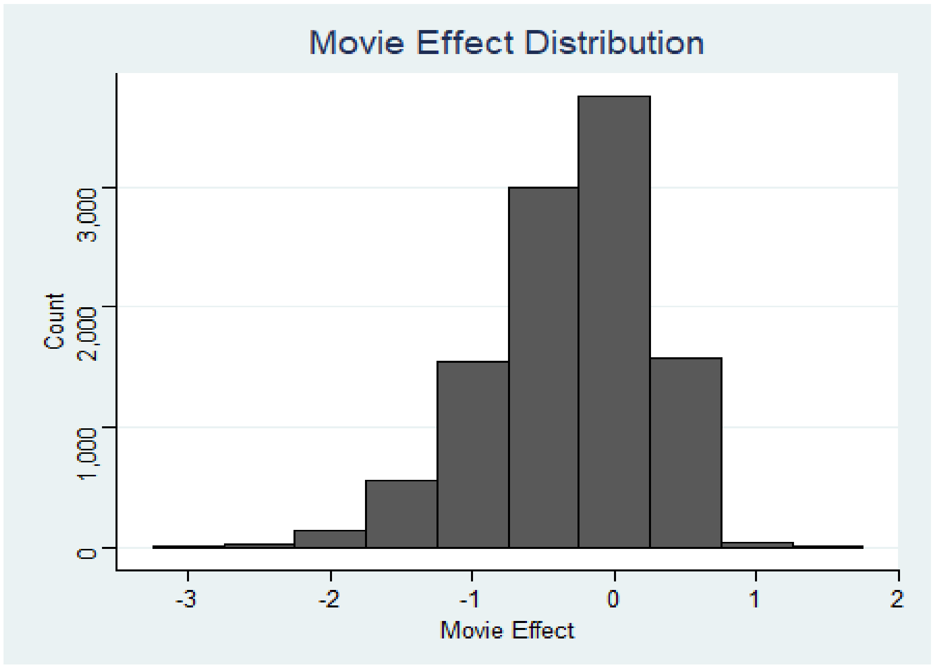
The RMSE of Naïve Bayes Approach is quite high.

3.1.2 Movie Effect Bias - b_i

Since RMSE from Naïve Bayes Approach turned out to be quite high, average rating for movie i is added in the model since different movies are rated differently.

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

```
mu <- mean(train_set$rating)


# determining movie effect


b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(bi = mean(rating - mu))


qplot(bi, data = b_i , bins = 10, color = I("black")) +
    ggtitle("Movie Effect Distribution") +
    xlab("Movie Effect") +
    ylab("Count") +
    scale_y_continuous(labels = comma) + theme_stata()
```



The plot shows that there is variation is bias and it is not normally distributed.

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

Result <- bind_rows(Result,
                         tibble(Method = "Movie Effect Model ",
                                RMSE = RMSE(test_set$rating, predicted_rating
s)))


Result
```

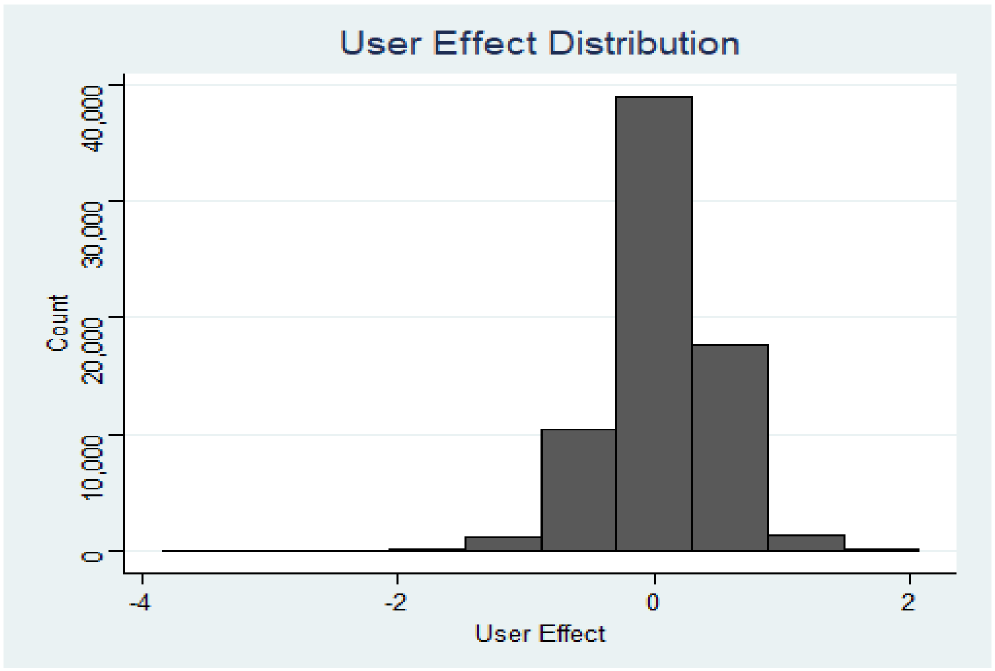
```
##                   Method      RMSE
## 1 Naive Bayes Approach 1.0600537
## 2   Movie Effect Model 0.9429615
```

3.1.3 User Effect Bias - b_u

The User Effect Model weights in variation in ratings because different users rate differently.

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

```
# determining user effect

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

qplot(bu, data = b_u , bins = 10, color = I("black")) +
  ggtitle("User Effect Distribution") +
  xlab("User Effect") +
  ylab("Count") +
  scale_y_continuous(labels = comma) + theme_stata()
```



The user effect distribution follows a bell-shaped curve.

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


Result <- bind_rows(Result,
                    tibble(Method = "Movie & User Effect Model ",
                           RMSE = RMSE(test_set$rating, predicted_ratings)))


Result

##                      Method      RMSE
## 1      Naive Bayes Approach  1.0600537
## 2        Movie Effect Model  0.9429615
## 3 Movie & User Effect Model  0.8646843
```

3.2 Regularized Movie & User Linear Model

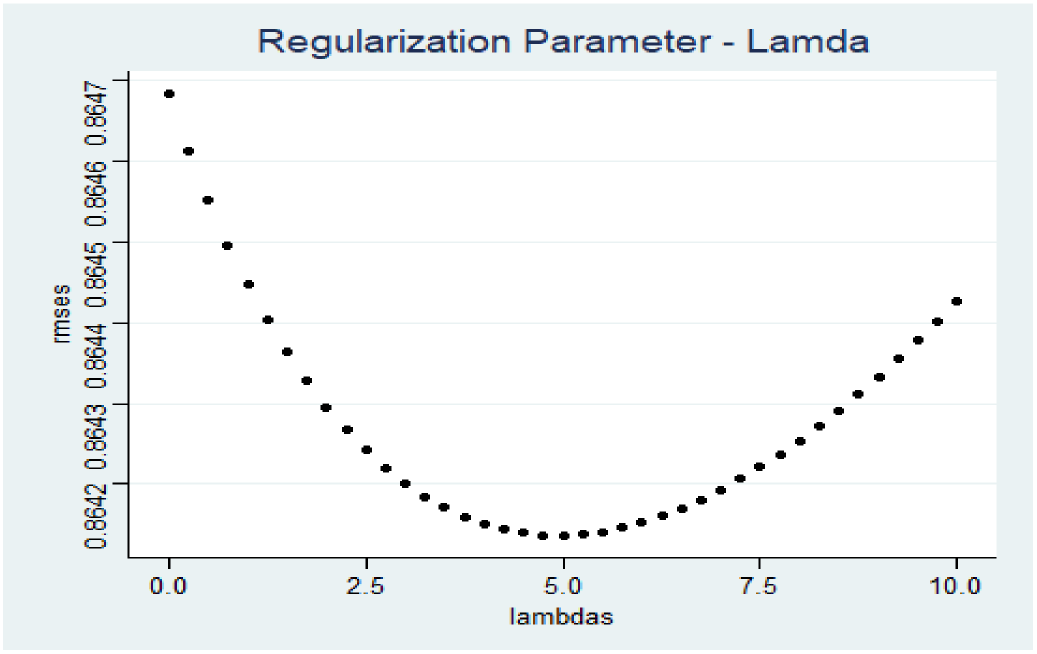
With insights gained in exploring data set, we came across large sets of estimates from small sample of ratings given to movies and ratings given by users. To penalize these estimated we use lambda – $\boldsymbol{\lambda}$ **.**

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

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

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

```
# use cross validation to pick lamda

lambdas <- seq(0, 10, 0.25)

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

  mu <- mean(train_set$rating)

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

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

  predicted_ratings <-
    test_set %>%
    left_join(bi, by = "movieId") %>%
    left_join(bu, by = "userId") %>%
    mutate(pred = mu + bi + bu) %>%
    pull(pred)

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

qplot(lambdas, rmses) + ggtitle("Regularization Parameter - Lamda") + theme_s
tata()
```



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

lambda

## [1] 5



regularized_rmse <- min(rmses)

regularized_rmse

## [1] 0.8641362



Result <- bind_rows(Result, data.frame(Method = "Regularized Movie & User Eff
ect Model ", RMSE = regularized_rmse))



Result

##                                    Method      RMSE
## 1                     Naive Bayes Approach  1.0600537
## 2                       Movie Effect Model  0.9429615
## 3                Movie & User Effect Model  0.8646843
## 4 Regularized Movie & User Effect Model  0.8641362
```

The RMSE is only 0.0007638 points lower than evaluation criteria set at 0.86490.

3.3 Matrix Factorization

The model factorizes the matrix of residuals r into a vector P and vector Q, where P is user rating score matrix and Q is movies rating score matrix.

$$r \approx \mathrm{PQ}$$

The Recommender system package (*recosystem*) in R estimates rating matrix R_{mn} by the product of two matrices of lower dimensions, P_{nk} and Q_{nk}.

To use *recosystem*, we follow these steps:

I. Create a model object (a Reference Class object in R) by calling Reco().

II. (Optionally) call the $tune() method to select best tuning parameters along a set of candidate values.

III. Train the model by calling the $train() method. A number of parameters can be set inside the function, possibly coming from the result of $tune().

IV. (Optionally) export the model via $output(), i.e. write the factorization matrices PP and QQ into files or return them as R objects.

V. Use the $predict() method to compute predicted values

```
#using Recommender System

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

## Loading required package: recosystem

# filtering required inputs

#user_index - integer vector giving the user indices of rating scores
#item_index - integer vector giving the item indices of rating scores
#rating     - numeric vector of the observed entries in the rating matrix

train_data <- with(train_set, data_memory(user_index = userId,
                                          item_index = movieId,
                                          rating = rating))
test_data <- with(test_set, data_memory(user_index = userId,
                                        item_index = movieId,
                                        rating = rating))
# r is object returned by Reco()

r <- Reco()

# Using cross validation to tune the model parameters

# Using default values for tuning

opts <- r$tune(train_data, opts = list(dim = c(10, 20),
                                          costp_l1 = c(0, 0.1),
                                          costp_l2 = c(0.01, 0.1),
                                          costq_l1 = c(0, 0.1),
                                          costq_l2 = c(0.01, 0.1),
                                          lrate = c(0.01, 0.1)))

# training model

r$train(train_data, opts = c(opts$min, nthread = 1, niter = 20))


## iter      tr_rmse          obj
##    0       0.9753   1.0908e+07
##    1       0.8764   8.9967e+06
##    2       0.8458   8.3739e+06
##    3       0.8267   8.0184e+06
##    4       0.8129   7.7695e+06
##    5       0.8018   7.5916e+06
##    6       0.7923   7.4499e+06
##    7       0.7843   7.3364e+06
##    8       0.7776   7.2433e+06
##    9       0.7720   7.1673e+06
##   10       0.7671   7.1030e+06
##   11       0.7630   7.0492e+06
##   12       0.7595   7.0050e+06
##   13       0.7564   6.9654e+06
##   14       0.7537   6.9323e+06
##   15       0.7513   6.9008e+06
##   16       0.7492   6.8756e+06
##   17       0.7473   6.8534e+06
##   18       0.7456   6.8339e+06
##   19       0.7440   6.8145e+06


# predicting ratings

predicted_ratings <- r$predict(test_data, out_memory())



Result <- bind_rows(Result,
                    tibble(Method = "Matrix Factorization Model ",
                           RMSE = RMSE(test_set$rating, predicted_ratings)))




Result

##                                  Method      RMSE
## 1                  Naive Bayes Approach  1.0600537
## 2                    Movie Effect Model  0.9429615
## 3               Movie & User Effect Model  0.8646843
## 4 Regularized Movie & User Effect Model  0.8641362
## 5              Matrix Factorization Model  0.7911444
```

The Matrix Factorization Model made significant improvement in RMSE.

3.4 Final Model – Matrix Factorization

We tried multiple approaches to achieve desired RMSE. However, it is evident from Result table that only Matrix Factorization model have helped us with our objective. Therefore, for our final model, we train edx set with Matrix Factorization and test the model with validation set.

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

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

library(recosystem)

# filtering required inputs from edx set & validation set

edx_rs <- with(edx, data_memory(user_index = userId,
                                item_index = movieId,
                                rating = rating))
validation_rs <- with(validation, data_memory(user_index = userId,
                                              item_index = movieId,
                                              rating = rating))
# model object r returned by Reco()

r <- Reco()

# Using cross validation to tune the model parameters

# Using default values for tuning

options <- r$tune(edx_rs , opts = list(dim = c(10, 20),
                                        costp_l1 = c(0, 0.1),
                                        costp_l2 = c(0.01, 0.1),
                                        costq_l1 = c(0, 0.1),
                                        costq_l2 = c(0.01, 0.1),
                                        lrate = c(0.01, 0.1)))

# training model

r$train(edx_rs, opts = c(options$min, nthread = 1, niter = 20))

## iter      tr_rmse          obj
##    0       0.9673   1.1908e+07
##    1       0.8740   9.9028e+06
##    2       0.8425   9.2160e+06
##    3       0.8220   8.8142e+06
##    4       0.8066   8.5342e+06
##    5       0.7954   8.3355e+06
##    6       0.7871   8.1922e+06
##    7       0.7803   8.0837e+06
##    8       0.7747   7.9922e+06
##    9       0.7699   7.9200e+06
##   10       0.7658   7.8595e+06
##   11       0.7621   7.8050e+06
##   12       0.7590   7.7623e+06
##   13       0.7562   7.7251e+06
##   14       0.7536   7.6907e+06
##   15       0.7514   7.6628e+06
##   16       0.7494   7.6346e+06
##   17       0.7476   7.6107e+06
##   18       0.7459   7.5900e+06
##   19       0.7444   7.5702e+06


# predicting ratings

predicted_ratings <- r$predict(validation_rs, out_memory())

Result <- bind_rows(Result,
                    tibble(Method = "Final Matrix Factorization Model ",
                           RMSE = RMSE(validation$rating, predicted_ratings))
)




Result

##                                    Method      RMSE
## 1                    Naive Bayes Approach  1.0600537
## 2                      Movie Effect Model  0.9429615
## 3               Movie & User Effect Model  0.8646843
## 4 Regularized Movie & User Effect Model  0.8641362
## 5              Matrix Factorization Model  0.7911444
## 6        Final Matrix Factorization Model  0.7885821


# RMSE of Final Matrix Factorization Model is 0.7885821 which is much lower

than evaluation criteria set at 0.86490
```

With Final Matrix Factorization Model, we accomplished an 8.82% lower RMSE than evaluation cutoff of 0.86490.

4. CONCLUSION

The report documented insights from modelling MovieLens data set through multiple approaches. The first approach, Naïve Bayes approach assumed that all movies received same ratings that is mean of observed ratings. When evaluated, the model didn't perform well with RMSE equaling to 1.0600537. The approach was then built upon with Movie Effect and Movie and User Effects respectively. After analyzing performance index of these approaches and insights that were gained during data exploration, small sample of ratings given to movies and small sample of ratings given by users were penalized in Regularized Movie and User Effect model. With RMSE still not lower than cut-off of 0.86490, Matrix Factorization Model was adopted to predict ratings. The model helped in achieving the desired RMSE value.

4.1 LIMITATIONS

Though Matrix Factorization did provide predictions closer to original ratings with RMSE of 0.7885821, there are some factors which limit its wide application. Firstly, computations are quite expensive in terms of time they cost to compute and memory they consume. Secondly, computations are based on only two predictors: users and movies. Other predictors such as genres were not taken in to account. And lastly, model takes in values of only existing ratings, movies and users. Hence, it must run over and over again if there is an update in values of these predictors. With large data sets, this process can get quite cumbersome and inefficient.

4.2 FUTURE WORK

Recommendation Systems are also built with other packages such as Recommender lab package. In future, we can overcome our short comings of models presented in this report with Recommender Lab package or other supporting packages available on CRAN.