Movielens Edx Machine Learning Project

Eddwin Cheteni

2021-03-16

Table of contents

1.0 Project Overview

Introduction

Recommendation system provides suggestions to the users through a filtering process that is based on user preferences and browsing history. The information about the user is taken as an input. The information is taken from the input that is in the form of browsing data. This recommender system has the ability to predict whether a particular user would prefer an item or not based on the user's profile. Recommender systems are beneficial to both service providers and users. They reduce transaction costs of finding and selecting items in an online shopping environment.

Problem Statement

For this project, a movie recommendation system is created using the MovieLens dataset. The version of Movielens included in the dslabs package (which was used for some of the exercises in PH125.8x: `Data Science: Machine Learning`) is just a small subset of a much larger dataset with millions of ratings. The entire latest MovieLens dataset can be found https://grouplens.org/datasets/movielens/latest/. Recommendation system is created using all the tools learnt throughout the courses in this series. `MovieLens` data is downloaded using the available code to generate the datasets.

First, the datasets will be used to answer a short quiz on the `MovieLens` data. This will give the researcher an opportunity to familiarize with the data in order to prepare for the project submission. Second, the dataset will then be used to train a machine learning algorithm using the inputs in one subset to predict movie ratings in the validation set.

The value used to evaluate algorithm performance is the `Root Mean Square Error`, or `RMSE.` RMSE is one of the most used measure of the differences between values predicted by a model and the values observed. RMSE is a measure of accuracy, to compare forecasting errors of different models for a particular dataset, a lower RMSE is better than a higher one. The effect of each error on `RMSE` is proportional to the size of the squared error; thus larger errors have a disproportionately large effect on RMSE. Consequently, `RMSE` is sensitive to outliers.

A comparison of the models will be based on better accuracy. The evaluation criteria for these algorithms is a RMSE expected to be lower than `0.8649`.

The function that computes the `RMSE` for vectors of ratings and their corresponding predictors will be the following:

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

MovieLens

https://movielens.org/ is a project developed by https://grouplens.org/, a research laboratory at the University of Minnesota. MovieLens provides an online movie recommender application that uses anonymously-collected data to improve recommender algorithms. To help people develop the best recommendation algorithms, MovieLens also released several data sets. In this article, latest data set will be used, which has two sizes.

The full data set consists of more than 24 million ratings across more than `40,000` movies by more than `250,000` users. The file size is kept under 1GB by using indexes instead of full string names.

Load the data

DataFrames are one of the easiest and best performing ways of manipulating data with `R`, but they require structured data in formats or sources such as `CSV.`

Download the data from MovieLens

To download the data:

7. We download the small version of the `ml-latest.zip` file from https://grouplens.org/datasets/movielens/latest/.
8. Unzip the file. The files to be used are `movies.csv` and `ratings.csv` to create `edx` and `validation` set as indicated by the code below.

Luckily, a code to download and create `edx` and `validation` set is readily available which is to be used in this 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")
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

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

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

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

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


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

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list =
FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

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

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

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

Loading important packages for this project.

Let’s load some important library packages we might need during the process:

```
if(!require(readr)) install.packages("readr")
if(!require(dplyr)) install.packages("dplyr")
if(!require(tidyr)) install.packages("tidyr")
if(!require(stringr)) install.packages("stringr")
if(!require(ggplot2)) install.packages("ggplot2")
if(!require(gridExtra)) install.packages("gridExtra")
if(!require(dslabs)) install.packages("dslabs")
if(!require(ggrepel)) install.packages("ggrepel")
if(!require(ggthemes)) install.packages("ggthemes")
```

```
library(readr)
library(dplyr)
library(tidyr)
library(stringr)
library(ggplot2)
library(gridExtra)
library(dslabs)
library(data.table)
library(ggrepel)
library(ggthemes)
```

Data Exploration: Quiz questions

This sub-section only covers the quizzes which is part of the grading of this project and it will also help to understand the structure of the dataset involved.

```
#Q1: How many rows and columns are there in the edx dataset?
dim(edx)

#Q2a: How many zeros and threes were given as ratings in the edx dataset?
#How many zeros were given as ratings in the edx dataset?
edx %>% filter(rating == 0) %>% tally()

##   n
## 1 0

#Q2b:How many threes were given as ratings in the edx dataset?
#How many threes were given as ratings in the edx dataset?
edx %>% filter(rating == 3) %>% tally()

##         n
## 1 2121240

# Q3:How many different movies are in the edx dataset?
n_distinct(edx$movieId)

## [1] 10677

#Q4: How many different users are in the edx dataset?
n_distinct(edx$userId)

## [1] 69878

#Q5: How many movie ratings are in each of the following genres in the edx dataset?
genres = c("Drama", "Comedy", "Thriller", "Romance")
sapply(genres, function(g) {
    sum(str_detect(edx$genres, g))
})

##    Drama   Comedy Thriller  Romance
##  3910127  3540930  2325899  1712100

#Q6: Which movie has the greatest number of ratings?
edx %>% group_by(movieId, title) %>%
    summarize(count = n()) %>%
    arrange(desc(count))%>%head(5)
```

```
## # A tibble: 5 x 3
## # Groups:   movieId [5]
##   movieId title                            count
##     <dbl> <chr>                            <int>
## 1     296 Pulp Fiction (1994)              31362
## 2     356 Forrest Gump (1994)              31079
## 3     593 Silence of the Lambs, The (1991) 30382
## 4     480 Jurassic Park (1993)             29360
## 5     318 Shawshank Redemption, The (1994) 28015
```

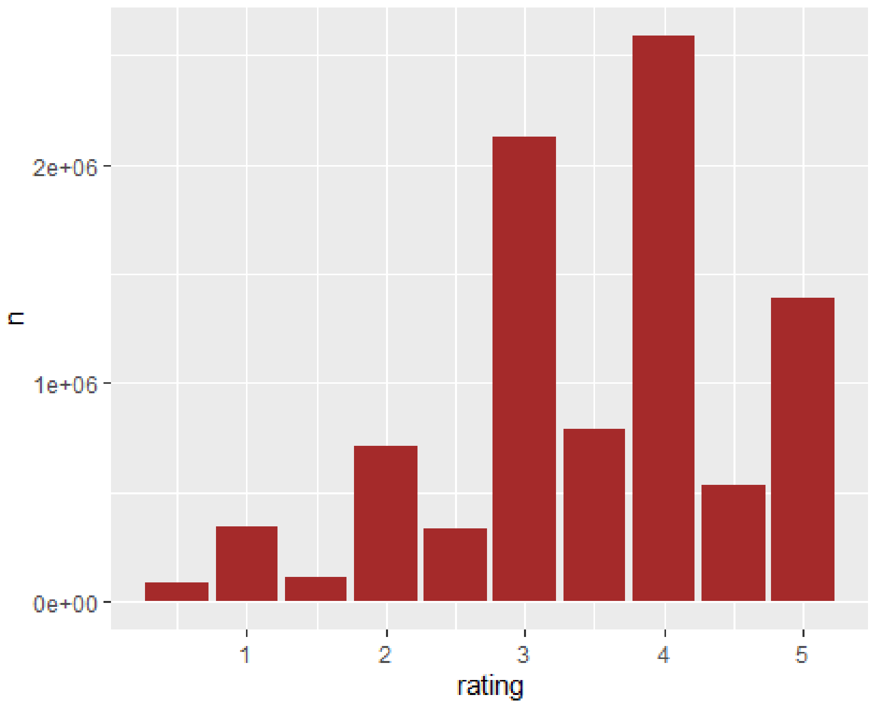
```
#Q7: What are the five most given ratings in order from most to least?
edx %>% group_by(rating) %>% summarize(count = n()) %>%
    arrange(desc(count))%>%head(5)
```

```
## # A tibble: 5 x 2
##   rating   count
##    <dbl>   <int>
## 1    4   2588430
## 2    3   2121240
## 3    5   1390114
## 4    3.5  791624
## 5    2    711422
```

```
#Q8 True or False: In general, half star ratings are less common than whole star
ratings (e.g., there are fewer ratings of 3.5 than there are ratings of 3 or 4,
etc.).
edx %>% group_by(rating)%>%
    summarize(count=n())%>%
    arrange(desc(count))%>%
    head(10)
```

```
## # 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
```

```
#confirming on the frequency of ratings given by users
edx %>% group_by(rating)%>%
    summarize(n=n())%>%
    arrange(desc(n))%>%
    ggplot()+
    #scale_y_log10() +
    geom_col(mapping = aes(x = rating, y = n), fill=I("brown"))
```



2.0 Data Exploration and Pre-processing

Data exploration (Continuation)

Edx dataset

Let have a look at the data types in our edx set and it shows that we have `int,` `num` and `chr` type of data of which it complies with the entries on each column.

```
## 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 838984474
838983653 838984885 838983707 838984596 ...
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)"
"Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-
Fi|Thriller" "Action|Adventure|Sci-Fi" ...
##  - attr(*, ".internal.selfref")=<externalptr>

## [1] 9000055       6
```

There are `9000055` rows and `6` columns in `edx` dataset.

Let have a look at the edx dataset again with a quick summary.

```
##      userId          movieId           rating        timestamp            title
genres
##  Min.   :    1   Min.   :    1   Min.   :0.50   Min.   :7.90e+08   Length:9000055
Length:9000055
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.00   1st Qu.:9.47e+08   Class
:character   Class :character
```

```
##  Median :35738   Median : 1834   Median :4.00   Median :1.04e+09   Mode
:character   Mode  :character
##  Mean   :35870   Mean   : 4122   Mean   :3.51   Mean   :1.03e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.00   3rd Qu.:1.13e+09
##  Max.   :71567   Max.   :65133   Max.   :5.00   Max.   :1.23e+09
```

Let's check if any missing values exist.

```
anyNA(edx)#checking missing values

## [1] FALSE
```

It seems there are no missing values in edx dataset.

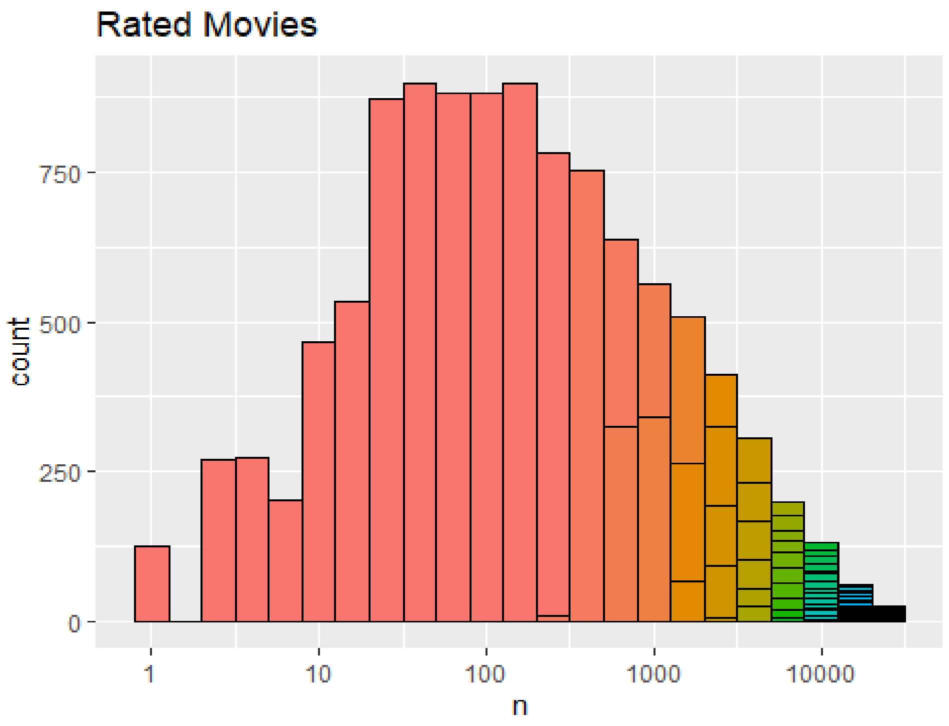
```
##   n_users n_movies
## 1   69878    10677
```

As shown above, there are approximately `70000` unique users giving ratings to more than `10000` unique movies.

Movies were given ratings ranging from `0.5` as the minimum rating to `5` as the maximum rating.

```
##  [1] 5.0 3.0 2.0 4.0 4.5 3.5 1.0 1.5 2.5 0.5
```

Some movies were given more ratings as compared to others. Let us find out which range of number of ratings were given to these movies.



We can separate the year from the movie title in order to gain better understanding of some patterns, perhaps this could also affect viewers' preferences based on the year of production which in turn influence its recommendation system.

Let's have a look at the resulting dataset:

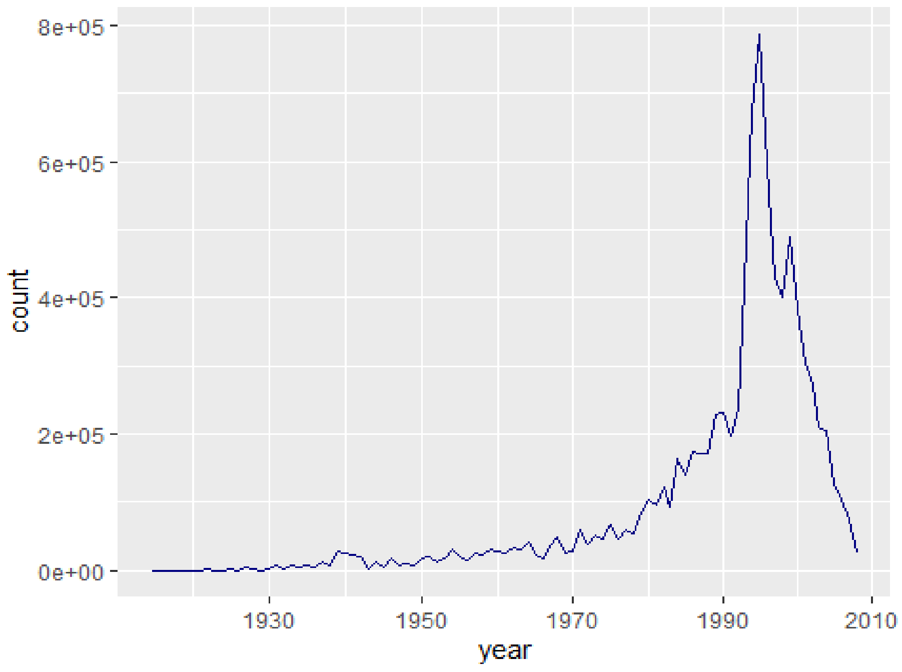
```
##    userId movieId rating timestamp                   title
genres year
## 1:      1     122      5 838985046              Boomerang
Comedy|Romance 1992
## 2:      1     185      5 838983525               Net, The
Action|Crime|Thriller 1995
## 3:      1     292      5 838983421               Outbreak   Action|Drama|Sci-
Fi|Thriller 1995
## 4:      1     316      5 838983392               Stargate
Action|Adventure|Sci-Fi 1994
## 5:      1     329      5 838983392 Star Trek: Generations
Action|Adventure|Drama|Sci-Fi 1994
```

Now the dataset clearly shows title and year of release separately.

Let us create a dataset showing release year and numbers of movies that were released in that particular year.

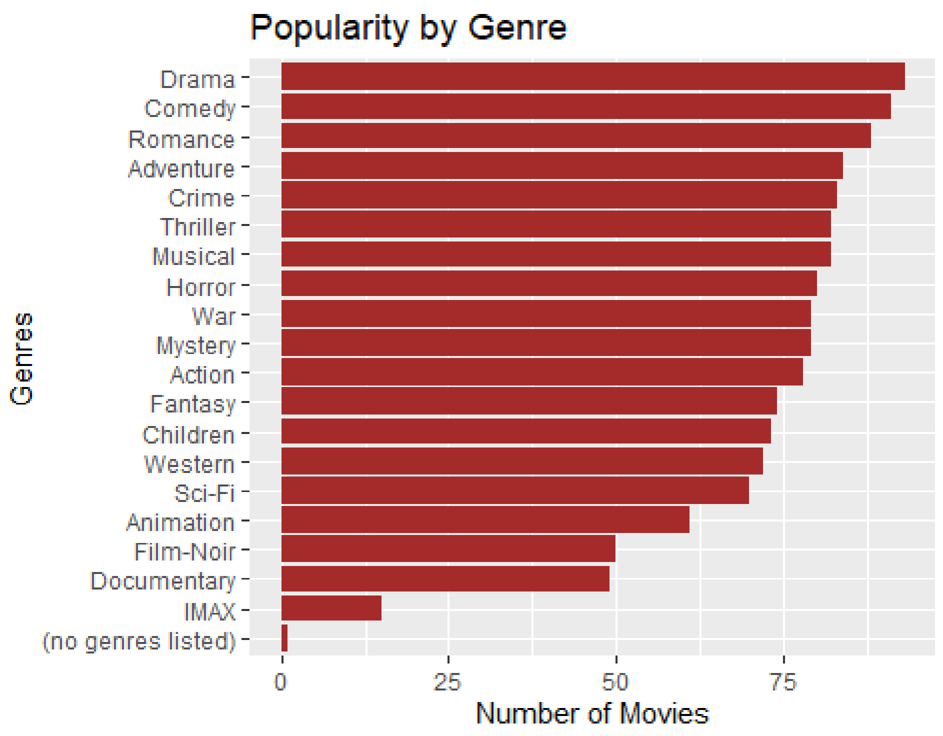
```
## # A tibble: 6 x 2
##    year count
##   <dbl> <int>
## 1  1915   180
## 2  1916    84
## 3  1917    32
## 4  1918    73
## 5  1919   158
## 6  1920   575
```

We can make a review on the above `n_movies_per_year` dataset.



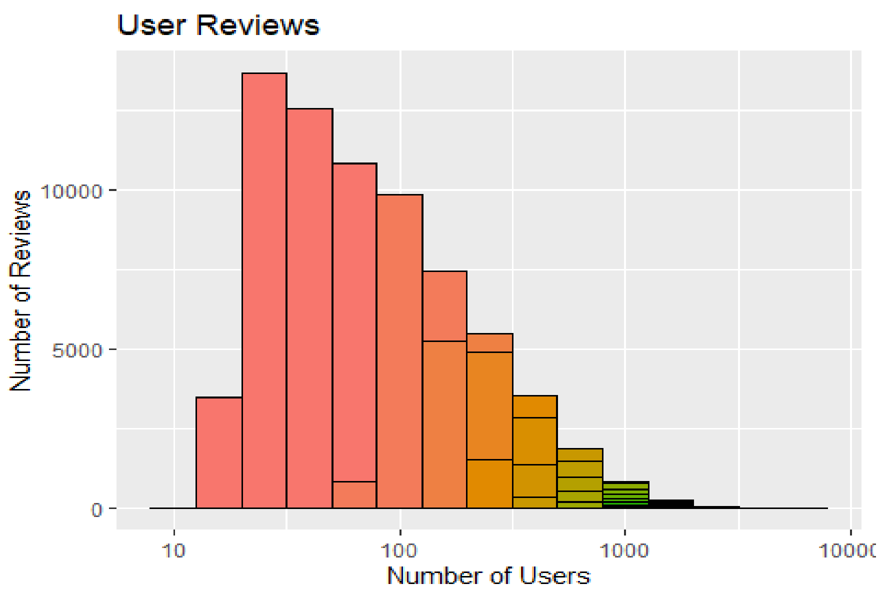
We can see that the data shows an exponential increase in movie business indicating a rapid rise from **1990** going to **2000** and started to drop thereafter.

Let us check the most popular movie genres, year by year. We can make a review on the above `genres_by_year` dataset:

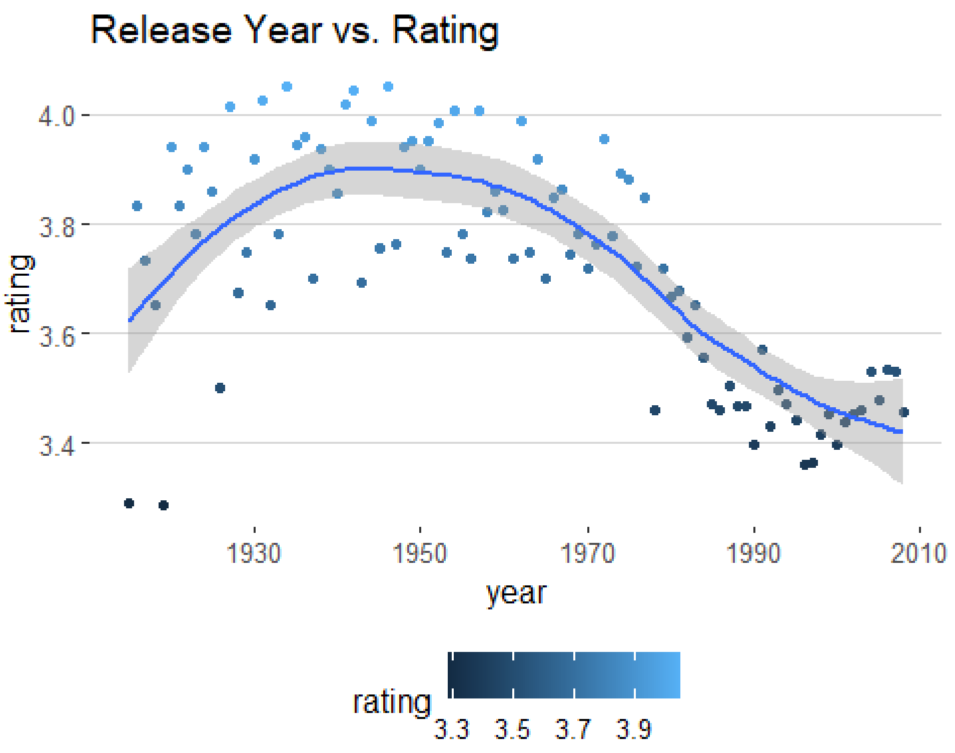


Looking at the chart above, there seems to be no much variations in terms of the movie genre against reviews.

Let's check the pattern of reviews with some plot to see the number of times each user has reviewed these movies.



We can see that most users reviewed at less than `300` movies. Let's make a review of movie ratings against the year of release:



Looking at the above chart, we can see that older movies were given higher ratings compared to newer movies. This could indicate that these old movies are more important in terms of coming up with a recommender system for Movielens dataset.

Data Partitioning

In order to build models that can be used to recommend movies to the viewers, we need to select important features needed and split the edx dataset into `train` and `test` set. For good results, we split the data into `80-20` percent ratio for `train-test` set.

```
##    userId movieId                   title                          genres year
rating
## 1:      1     122               Boomerang                  Comedy|Romance 1992
5
## 2:      1     185                Net, The           Action|Crime|Thriller 1995
5
## 3:      1     292                Outbreak    Action|Drama|Sci-Fi|Thriller 1995
5
## 4:      1     316                Stargate         Action|Adventure|Sci-Fi 1994
5
## 5:      1     329 Star Trek: Generations   Action|Adventure|Drama|Sci-Fi 1994
5
## 6:      1     355        Flintstones, The         Children|Comedy|Fantasy 1994
5
```

Now our data only include features that are important for model building phase.

We then split the dataset into train and test set.

```
set.seed(123)

#loading libraries for data partitioning process
library(caret)
library(lattice)

#creating an index to pick a sample for train set
My_Index <- createDataPartition(edxset$rating, p= 0.8, list = FALSE, times = 1)
train <- edxset[My_Index, ] #creating the train set
test <- edxset[-My_Index, ] #creating the test set

# Make sure userId, year and movieId in test set are also in train set
test <- test %>%
     semi_join(train, by = "movieId") %>%
     semi_join(train, by = "userId")%>%
     semi_join(train, by = "title")

## [1] "Dimensions of the train set are:"

## [1] 7200045       6

## [1] "Dimensions of the test set are:"

## [1] 1799974       6
```

3.0 Model Training

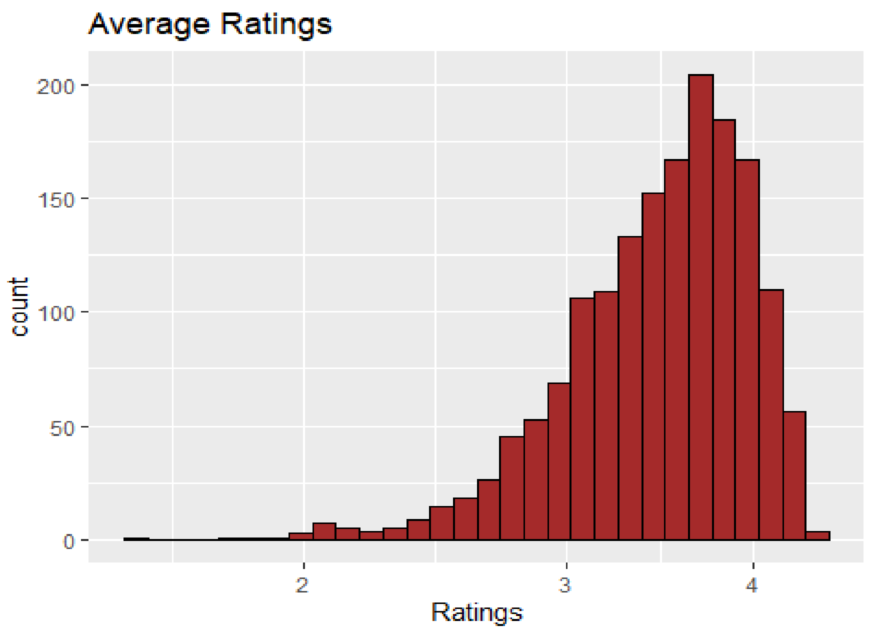
Naive-based model

Calculating the average rating of all the movies using train set. We can see that if we predict all unknown ratings with μ we obtain the following `RMSE`.

```
## [1] 3.51
```

We observe that `RMSE` for the first model indicates that movie rating is `>3.5`.

We know from experience that some movies are just generally rated higher than others. We can use data to confirm this, let's say, if we consider movies with more than `1,000` ratings, the `SE error` for the average is at most `0.05`. Yet plotting these averages we see much greater variability than `0.05`:



Movie-based model

We know from experience that some movies are just generally rated higher than others.Lets see if there will an improvement by adding the term b_i to represent the average rating for movie i.

```
#compute average rating
mu_hat <- mean(train$rating)
#compute average rating based on movies
movie_avgs <- train %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_hat))
#compute the predicted ratings on test set
predicted_ratings_movie_avg <- test %>%
  left_join(movie_avgs, by='movieId') %>%
   mutate(pred = mu_hat + b_i) %>%
   pull(pred)
```

Movie + User-effect model

```
#just the average
mu_hat <- mean(train$rating)
#compute average rating based on users
user_avgs <- train %>%
   left_join(movie_avgs, by='movieId') %>%
   group_by(userId) %>%
   summarize(b_u = mean(rating - mu_hat - b_i))
#compute the predicted ratings on test set
predicted_ratings_movie_user_avg <- test %>%
   left_join(movie_avgs, by='movieId') %>%
   left_join(user_avgs, by='userId') %>%
   mutate(pred = mu_hat + b_i + b_u) %>%
   pull(pred)
```

Movie + Title + User-effect model

```
#just the average of ratings
mu_hat <- mean(train$rating)
# Calculate the average by title
```

```r
title_avgs <- train %>%
   left_join(movie_avgs, by='movieId') %>%
   left_join(user_avgs, by='userId') %>%
   group_by(title) %>%
   summarize(b_tt = mean(rating - mu_hat - b_i - b_u))
#compute the predicted ratings on test set
predicted_ratings_movie_title_user_avg <- test %>%
   left_join(movie_avgs, by='movieId') %>%
   left_join(user_avgs, by='userId') %>%
   left_join(title_avgs, by='title') %>%
   mutate(pred = mu_hat + b_i + b_u + b_tt) %>%
   pull(pred)
```

Movie + Title + User + Regularization

To better our models above and ensure we are converging to the best solution let us add a regularization constant for our movie rating, title effect and user-specific model. The purpose of this is to penalize large estimates that come from small sample sizes. Therefore, our estimates will try its best to guess the correct rating while being punished if the movie rating, user-specific, or title effect is too large.

This method is a little more complicated than the prior two methods. That is we want to also validate how much we want to regularize the `movie-rating, title, and user-specific effects`. We will try several regularization models with our regularization constant (`lambda`) at different values. We will define the function to obtain RMSEs here and apply it in our results section:

```r
lambdas <- seq(0, 15, 0.25)
regularize <- function(l){
    #calculate just the average of ratings
     mu <- mean(train$rating)
     #calculate the average by movie
     b_i <- train %>%
          group_by(movieId) %>%
          summarize(b_i = sum(rating - mu_hat)/(n()+l))
     #calculate the average by user
     b_u <- train %>%
          left_join(b_i, by="movieId") %>%
          group_by(userId) %>%
          summarize(b_u = sum(rating - b_i - mu_hat)/(n()+l))
     #calculate the average by title
     b_tt <- train %>%
          left_join(b_i, by="movieId") %>%
          left_join(b_u, by="userId") %>%
          group_by(title) %>%
          summarize(b_tt = sum(rating - b_i - b_u - mu_hat)/(n()+l))
     #calculate prediction on test set
     predicted_ratings <- test %>%
          left_join(b_i, by = "movieId") %>%
          left_join(b_u, by = "userId") %>%
          left_join(b_tt, by = "title") %>%
          mutate(pred = mu_hat + b_i + b_u + b_tt) %>%
          .$pred

     return(RMSE(predicted_ratings, test$rating))
}
```

4.0 Model Results and Validation

In this section we will see how well our models worked to our `test` set and then we validate the best model to unseen data in the `validation` set.

Before model validation, here is the defined `RMSE function`:

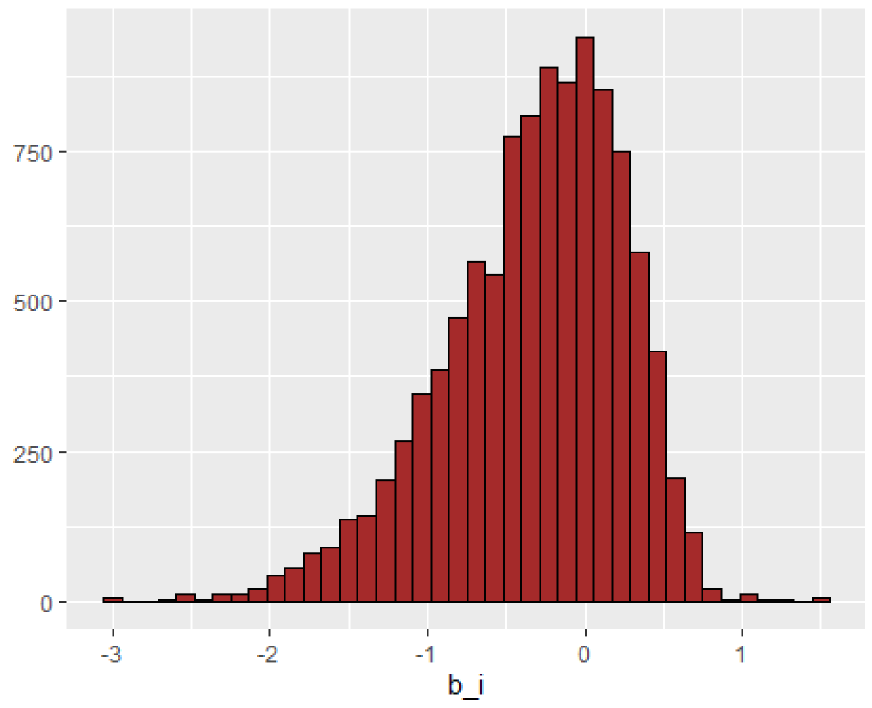
RMSE for Naive-based model

```
## [1] 1.06
```

RMSE for Movie effect model

```
## [1] 0.944
```

Based on the `RMSE` result obtained, `0.944` indicates an improvement. But can we make it much better than this? We can see that these estimates vary as indicated on the plot below:

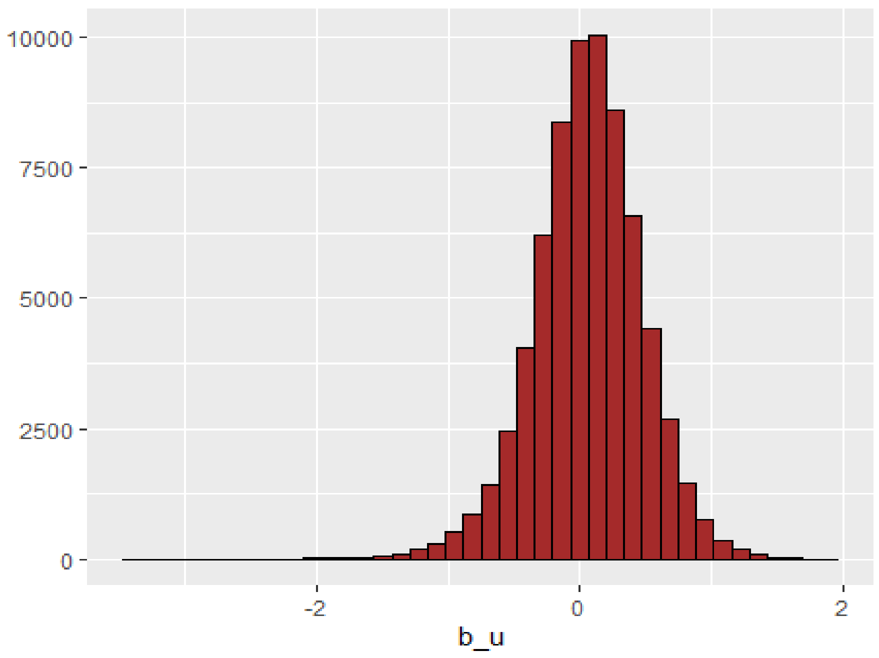


RMSE for Movie + user-specific model

```
## [1] 0.867
```

This seems to be a better model with an `RMSE` value of `0.867`.

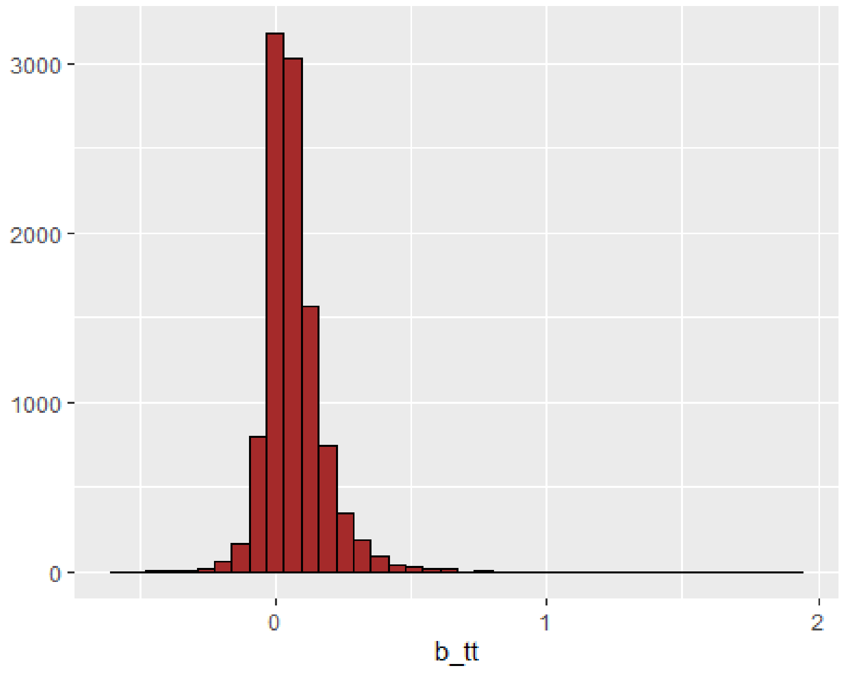
We can see that these estimates vary as indicated on the plot below:



RMSE for Movie + Title effect + user-specific model

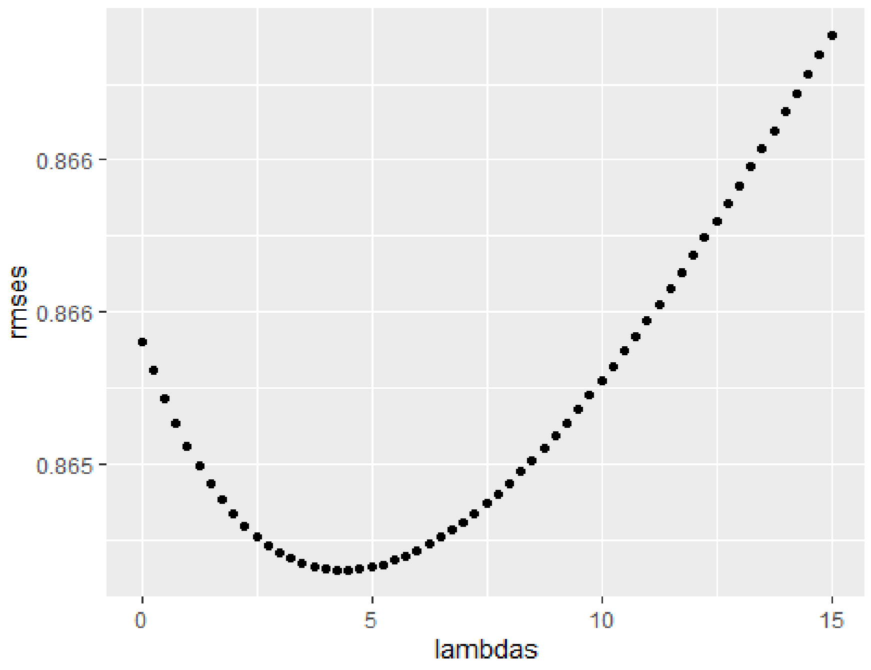
```
## [1] 0.865
```

This seems to be a better model with an `RMSE` value of `0.865`. We can see that these estimates vary as indicated on the plot below:



RMSE for Movie + user-specific + title effect + regularization model

This result is a little more complicated than the prior two methods. That is we want to also validate how much we want to regularize the movie-rating and user-specific effects. We will try several regularization models with our regularization constant (lambda) at different values:



Our minimum error is found with a `lambda` value of `4.5` with RMSE of `0.865`:

```
## [1] 4.5

## [1] 0.865
```

All Models RMSE Summary

The table below summarizes our RMSE values given our models explored:

```
## # A tibble: 5 x 2
##   Model                                RMSE
##   <chr>                               <dbl>
## 1 1. Naive-based                      1.06 
## 2 2. Movie-based                      0.944
## 3 3. Movie+User-based                 0.867
## 4 4. Movie+Title+User-based           0.865
## 5 5. Regularized Movie+Title+User-based 0.865
```

As we can see the regularization model performed best when lambda was set to `4.5` with an RMSE of `0.865`. Let's proceed to perform model validation based on this model.

Final Model Validation

Before we begin model validation, we need to separate year from the title and make sure that title in validation set are also in the train set.

```
#separating year from title and create new column named year
valid_yr = validation %>% mutate(year = as.numeric(str_extract(str_extract(title,
"[/(]\\d{4}[/)]$"), regex("\\d{4}"))),title = str_remove(title, "[/(]\\d{4}[/)]$"))

# Make sure userId and movieId in validation set are also in train set
valid_yr <- valid_yr %>%
      semi_join(train, by = "movieId") %>%
      semi_join(train, by = "userId") %>%
      semi_join(train, by = "title")
# keep this for checking RMSE on model evaluation
model_scoring <- valid_yr$rating
```

As mentioned we will train our regularized model (which takes into account (1) `average movie rating`, (2) `movie-rating`, (3) `average movie + user-rating`, and (4) `average movie + user + title-rating effects`) with a regularization parameter, lambda, of `4.5`.

```
#calculate just the average of ratings
   mu_reg <- mean(train$rating)
  #calculate the average by movie
  b_i_reg <- train %>%
          group_by(movieId) %>%
          summarize(b_i_reg = sum(rating - mu_reg)/(n()+lambda))
  #calculate the average by user
  b_u_reg <- train %>%
          left_join(b_i_reg, by="movieId") %>%
          group_by(userId) %>%
          summarize(b_u_reg = sum(rating - b_i_reg - mu_reg)/(n()+lambda))
  #calculate the average by title
  b_t_reg <- train %>%
          left_join(b_i_reg, by="movieId") %>%
          left_join(b_u_reg, by="userId") %>%
          group_by(title) %>%
          summarize(b_t_reg = sum(rating - b_i_reg - b_u_reg - mu_reg)/(n()+lambda))
  #calculate prediction on test set
  predicted_ratings_b_i_u_t <- valid_yr %>%
          left_join(b_i_reg, by = "movieId") %>%
          left_join(b_u_reg, by = "userId") %>%
          left_join(b_t_reg, by = "title") %>%
          mutate(pred = mu_hat + b_i_reg + b_u_reg + b_t_reg) %>%
          .$pred

# returning RMSE using validation set with year and title separated
rmse_validation <- RMSE(predicted_ratings_b_i_u_t, model_scoring)
rmse_validation

## [1] 0.865
```

As shown above our RMSE value against unseen data (our validation dataset) is `0.865` which meets our project objective of an `RMSE <= 0.865`.

5.0 Conclusion

In conclusion to this project, we have achieved an exceptional RMSE of `0.865` against our validation dataset. This RMSE result support the `Regularized-Movie+Title+User-effect model` which gives an RMSE of `0.865` on the test set. This is very impressive of our model since it gives a better accuracy in terms of predicting movie ratings of new dataset.

Given more time I would like to try to incorporate time factor as well as genre to see if this can also influence our model positively by improving the model accuracy.

6.0 References

All material in this project is credited to Professor Rafael Irizarry and his team at HarvardX's Data Science course. Most material was learned through his course, book, and github:

1. Irizarry, R 2021 ***Introduction to Data Science: Data Analysis and Prediction Algorithms with R*** ,github page,accessed 22 January 2021, https://rafalab.github.io/dsbook/

Another great resource was prior movie recommendation projects. Specifically, one project was referenced:

2. The work done by Brandon Rufino (how I was inspired to try data exploration techniques such as similarity measures), https://github.com/brufino/MovieRecommendationSystem/