MovieLens Rating Prediction Project

Nabeel Khan

21-May-2020

Contents

1 Introduction

This report is part of the 'HarvardX: PH125.9x Data Science: Capstone' course. In this report, we discuss various approaches to develop a recommender system using the acquired data science skills.

1.1 Background

The recommender systems predict the interests of users and recommend product items that are quite likely interesting for them[1, 2]. The recommender systems use state of the art machine learning algorithms to make precise predictions. The recommender systems are commonly used by Netflix, YouTube, Spotify, Amazon, or Ebay.

One of the famous success stories of the recommender system is Netflix competition [3]. In 2006, Netflix announced the awarding of one million dollar prize to a team, which can come up with the best filtering algorithm to predict user ratings for movies based on previous ratings. It took three years for a team to win the competition. The winning team improved the accuracy of the Netflix recommender system by 10%.

1.2 Aim of Project

This project aims to develop a recommender system using a model to predict movie ratings based on suitable predictors.

2 Dataset and Evaluation Metric

Netflix dataset is not available online. So, we use the 10M version of MoiveLens dataset in this project [4]. The dataset is generated by the GroupLens, which is a research lab in the Department of Computer Science and Engineering at the University of Minnesota. The dataset contains 10 million ratings. We will split the data into training and validation sets to develop our model.

To evaluate the performance of the model, we will use Root Mean Square Error (RMSE) [5] as defined in Equation 1. The RMSE is a standard method to measure the error of a model in predicting quantitative data.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2} \tag{1}$$

Where $\hat{y}_i$ refers to the predicted values by the model, y_i refers to the actual values, and n refers to the total number of observations. The RMSE is a commonly used metric for analyzing the performance of models, but it can produce biased results in the presence of a large number of outliers or noise in the data. In this project, the RMSE will indicate how close model predictions are to the actual ratings in the validation set.

2.1 Download Data

We download data from the website. Then, we split data into a training set referred to as *edx*, and test set referred to as *validation*. 10% of the data is used for validation, and 90% is used for training.

- The model is trained using *edx* dataset, and we compute RMSE using *validation* dataset.

```r
################################
#  Install packages (if not installed)
################################
# Note: this process could take a couple of minutes
repos_path<- "http://cran.us.r-project.org"
if(!require(tidyverse)) install.packages("tidyverse", repos =repos_path)
```

```
## Loading required package: tidyverse

## -- Attaching packages ---------------------------------------------------------------------- tidyvers

## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

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

```r
if(!require(caret)) install.packages("caret", repos = repos_path)
```

```
## Loading required package: caret

## Warning: package 'caret' was built under R version 3.6.3

## Loading required package: lattice
```

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

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

```r
if(!require(data.table)) install.packages("data.table", repos =repos_path)
```

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

## Warning: package 'data.table' was built under R version 3.6.3

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

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

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

```r
if(!require(lubridate)) install.packages("lubridate", repos = repos_path)
```

```
## Loading required package: 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 object is masked from 'package:base':
##
##     date
```

```r
if(!require(dplyr)) install.packages("dplyr", repos = repos_path)
if(!require(sjmisc)) install.packages("dplyr", repos = repos_path)
```

```
## Loading required package: sjmisc

## Warning: package 'sjmisc' was built under R version 3.6.3

##
## Attaching package: 'sjmisc'

## The following object is masked from 'package:purrr':
##
##     is_empty

## The following object is masked from 'package:tidyr':
##
##     replace_na

## The following object is masked from 'package:tibble':
##
##     add_case
```

```
################################
# Load libraries
################################
library(lubridate)
library(tidyverse)
library(dplyr)
library(lubridate)
library(sjmisc)
```

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

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

ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))), col.names = c
movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
   colnames(movies) <- c("movieId", "title", "genres")
   movies <- as.data.frame(movies) %>% mutate(movieId =   as.numeric(levels(movieId))[movieId], title =

################################
# Creating edx and validation sets
################################
movielens <- left_join(ratings, movies, by = "movieId")


# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind = "Rounding")
```

```
## 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)

 # Removing the objects from environment as no longer required
```

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

3 Data Exploration

In this section, we explore the dataset to get familiar with it. We also perform some data wrangling. Data wrangling is the process of cleaning, structuring, or enriching raw data into the desired format to make better decisions and get meaningful insights.

The dataset contains six variables namely `userID",`movieID“, `rating",`timestamp”, `title'", and`genres“. The “timestamp” indicates the date on which a user recorded his reviews for a particular movie. Each row in the dataset represents a single rating of a user for a single movie.

```r
head(edx)
```

	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
4	1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
5	1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
6	1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi
7	1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy

There are around nine million rows in *edx* dataset, which is approximately 90% of the whole data, as mentioned before. The summary of *edx* dataset shows that there are no null values in the training dataset.

```r
sprintf("Edx Dataset - Rows = %d  | Columns = %d",nrow(edx),ncol(edx))
```

```
## [1] "Edx Dataset - Rows = 9000055  | Columns = 6"
```

```r
summary(edx)
```

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

3.1 Data Wrangling

We find a couple of issues with the *edx* dataset :

1. The *“title”* of each movie has its premiere year appended to it.
2. The *“timestamp”* contains the number of seconds that have elapsed since January 1, 1970., which is hard to interpret.

To resolve these issues, we perform the following operations on both training and validation datasets:

1. Create a new column *"premiereYr"*, which will store the premier year of the movie. We remove the year information from *"title"* feature.
2. Modify the *"timestamp"* feature so that it contains the information in date format, which is easy to interpret.

We can see the impact of the above operations on our dataset by looking at the summary of the dataset.

```r
########################################
# EDX Dataset
########################################
# Extracting premiere date from movie title
edx <-edx %>% extract(title, c("title", "premiereYr"),
                      regex = "^(.*) \\(([0-9 ##\\-]*)\\)$")
# Converting to integer format from char
edx$premiereYr <- as.numeric(edx$premiereYr)

# Converting timestamp to date format
edx$timestamp <- as.Date(as_datetime((edx$timestamp), origin="1970-01-01"))

########################################
# Validation Dataset
########################################
# Extracting premiere date from movie title
validation <-validation %>% extract(title, c("title", "premiereYr"),
                                    regex = "^(.*) \\(([0-9 ##\\-]*)\\)$")
# Converting to integer format from char
validation$premiereYr <- as.numeric(validation$premiereYr)

# Converting timestamp to date format
validation$timestamp <- as.Date(as_datetime((validation$timestamp), origin="1970-01-01"))

head(edx)
```

	userId	movieId	rating	timestamp	title	premiereYr	genres
1	1	122	5	1996-08-02	Boomerang	1992	Comedy|Romance
2	1	185	5	1996-08-02	Net, The	1995	Action|Crime|Thriller
4	1	292	5	1996-08-02	Outbreak	1995	Action|Drama|Sci-Fi|Thriller
5	1	316	5	1996-08-02	Stargate	1994	Action|Adventure|Sci-Fi
6	1	329	5	1996-08-02	Star Trek: Generations	1994	Action|Adventure|Drama|Sci-Fi
7	1	355	5	1996-08-02	Flintstones, The	1994	Children|Comedy|Fantasy

4 Data Analysis

In the previous section, we understood the structure of the data and performed a couple of data wrangling operations. In this section, we extract insights about all features of the *edx* dataset.

4.1 Users

The dataset has around 69800 unique users.

```r
# Finding unique users
edx %>% summarize(users = n_distinct(userId))
```

users
69878

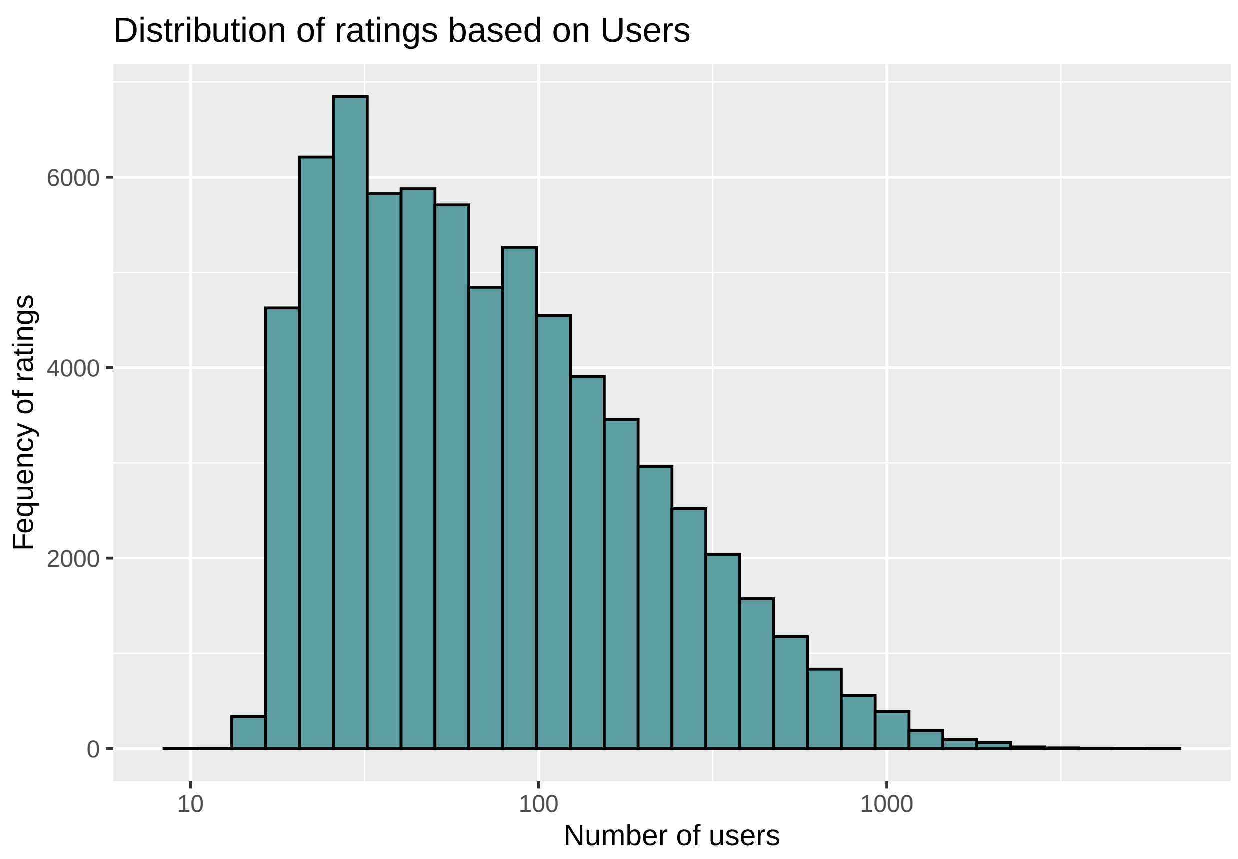
Each user can rate more than one movie. On average, each user rates around 129 movies. If we look at the frequency of ratings for users, we find out that the distribution is skewed to the right. Most of the users rate a few movies, while some rate movies in thousands. To handle such variability, we can include a user penalty term in our model (if needed) to improve the performance. The process of adding a penalty term to error function is known as **Regularization** [6].

Regularization allows penalizing large estimates that are formed using small sample sizes and avoid overfitting of data. Overfitting occurs because the model tries too hard to capture the noise in the training dataset. The basic idea of regularization is to constrain the total variability of the effect sizes.

```r
# Calculate the average number of ratings for users
edx %>% count(userId) %>% summarise(mean_rating=mean(n))
```

mean_rating
128.7967

```r
# Plotting frequency of ratings per user
edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30,  fill='cadetblue', color='black') +
  scale_x_log10() +
  xlab("Number of users") +
  ylab("Fequency of ratings") +
  ggtitle("Distribution of ratings based on Users")
```



4.2 Movies

The dataset has around 10400 unique movies.

```
# Finding unique movies
edx %>% summarize(movies = n_distinct(title))
```

movies
10407

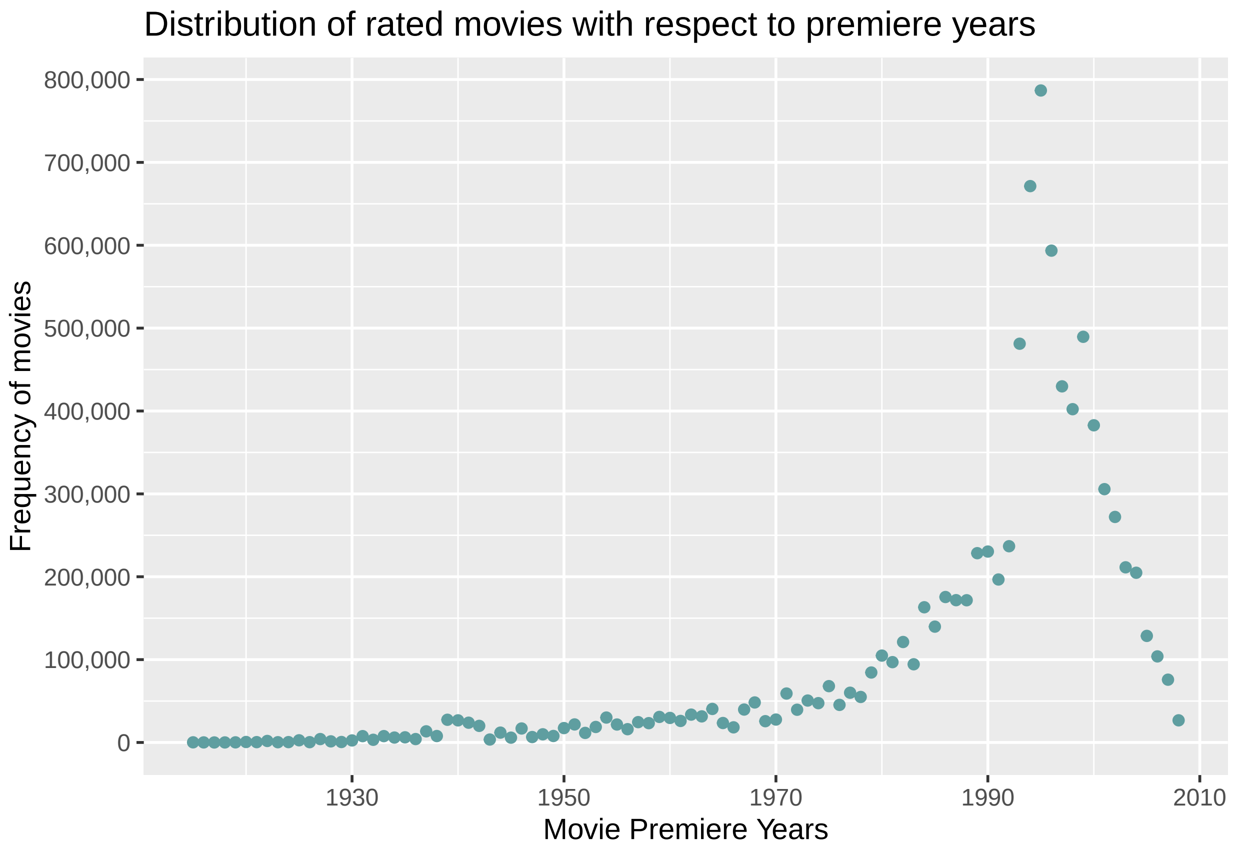
The release years for movies in *edx* dataset range from 1915 to 2008. We can see an increase in the number of rated movies for the years 1980 to 1995. After 1995, the frequency of rated movies starts decreasing—one of the possible reasons is that the dataset has not included many movies for other years due to insufficient rating information or the number of users providing the rating information dropped.

```
library(scales)
```

```
##
## Attaching package: 'scales'

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

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

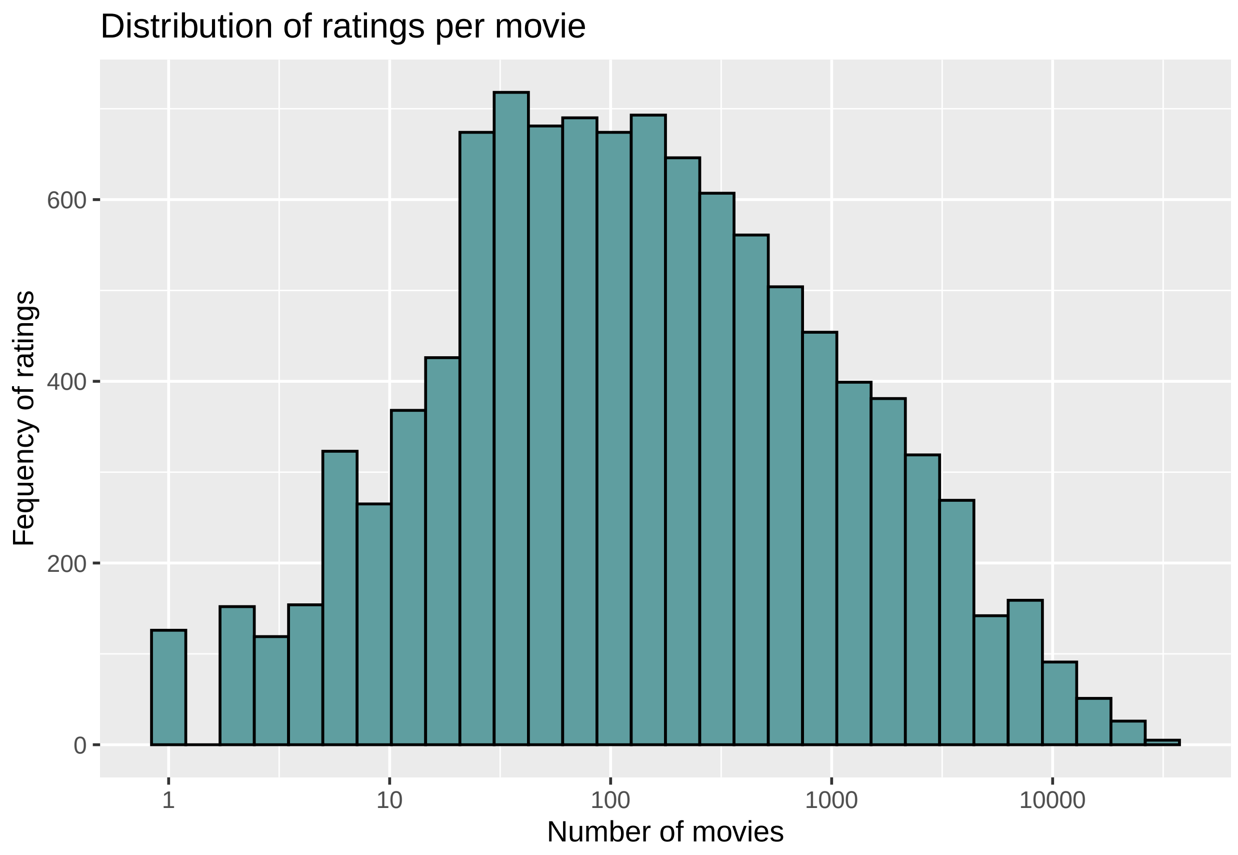
```
# Plotting frequency of rated moves over the years
# We use premiere years of movies as the range of years
edx %>%
  select(movieId, premiereYr) %>% # select columns we need
  group_by(premiereYr) %>% # group by year
  summarise(count = n())  %>% # count movies per year
  arrange(premiereYr)%>%
  ggplot(aes(x = premiereYr, y = count)) +
  scale_y_log10() +
  scale_y_continuous(breaks = c(seq(0, 800000, 100000)), labels=comma)+
  labs(x="Movie Premiere Years", y="Frequency of movies") +
  geom_point(color="cadetblue") +
  ggtitle("Distribution of rated movies with respect to premiere years")
```

```
## Scale for 'y' is already present. Adding another scale for 'y', which
## will replace the existing scale.
```



Most movies get a decent number of ratings from the users. Although some movies have a less number of ratings. This indicates that movie information can also be used as penalty term in our model (if required).

```
# Distribution or freqeuncy of ratings for all movies
edx %>% group_by(movieId) %>% summarize(n = n()) %>%
  ggplot(aes(n)) + geom_histogram(bins=30, fill = 'cadetblue', color = 'black') +
  scale_x_log10() +
  xlab("Number of movies") +
  ylab("Fequency of ratings") +
  ggtitle("Distribution of ratings per movie")
```



4.3 Ratings

As we discussed in Section that the release years of movies in the dataset range from 1915 to 2008. The dataset also indicates when users recorded the ratings for each movie. A close look suggests that rating data covers a reasonable period from years 1995 to 2009 i.e., 14 years.

```
# Analysing when ratings data was added by users
years<-format(edx$timestamp, format="%Y")
sprintf("Rating year (minimum) =%s", min(years))
```

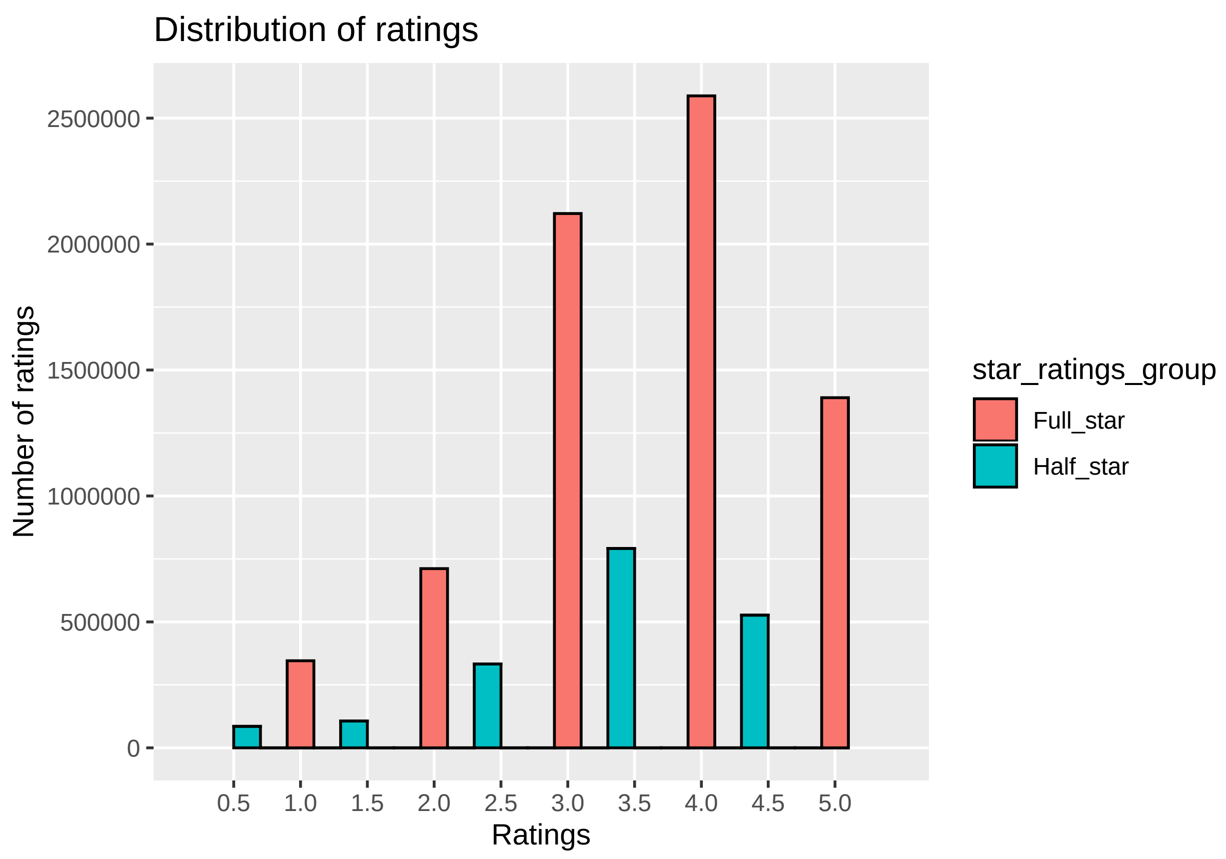
```
## [1] "Rating year (minimum) =1995"
```

```
sprintf("Rating year (maximu) =%s", max(years))
```

```
## [1] "Rating year (maximu) =2009"
```

The movies can have either half-star or full-stars ratings. The distribution indicates that most movies get full star ratings. We observe that there is a tendency that users generally rate movies 3 or 4.

```
# Generating groups of data for half star and full star ratings
star_ratings_group <-  ifelse((edx$rating == 1 |edx$rating == 2 | edx$rating == 3 |
                  edx$rating == 4 | edx$rating == 5) ,
                   "Full_star",
                   "Half_star")

# Plotting the distribution of ratings in terms of half and full star
ratings_distribution <- data.frame(edx$rating, star_ratings_group)
ggplot(ratings_distribution, aes(x= edx.rating, fill = star_ratings_group)) +
  geom_histogram( binwidth = 0.2,color = 'black') +
  scale_x_discrete(limits = c(seq(0.5,5,0.5))) +
  scale_y_continuous(breaks = c(seq(0, 3000000, 500000)))+
  labs(x="Ratings", y="Number of ratings") +
  ggtitle("Distribution of ratings")
```



We now explore the top 10 and bottom 10 movies based on the total number of ratings. Top 10 movies are well-known and many users rate them as expected. Contrary, the bottom 10 movies appear to be obscure, and only a few users rate them. Therefore, the predictions of future ratings for such movies will be difficult. There are 125 movies in the dataset with a single user rating. This information is critical because a low rating numbers can result in overfitting for our model.

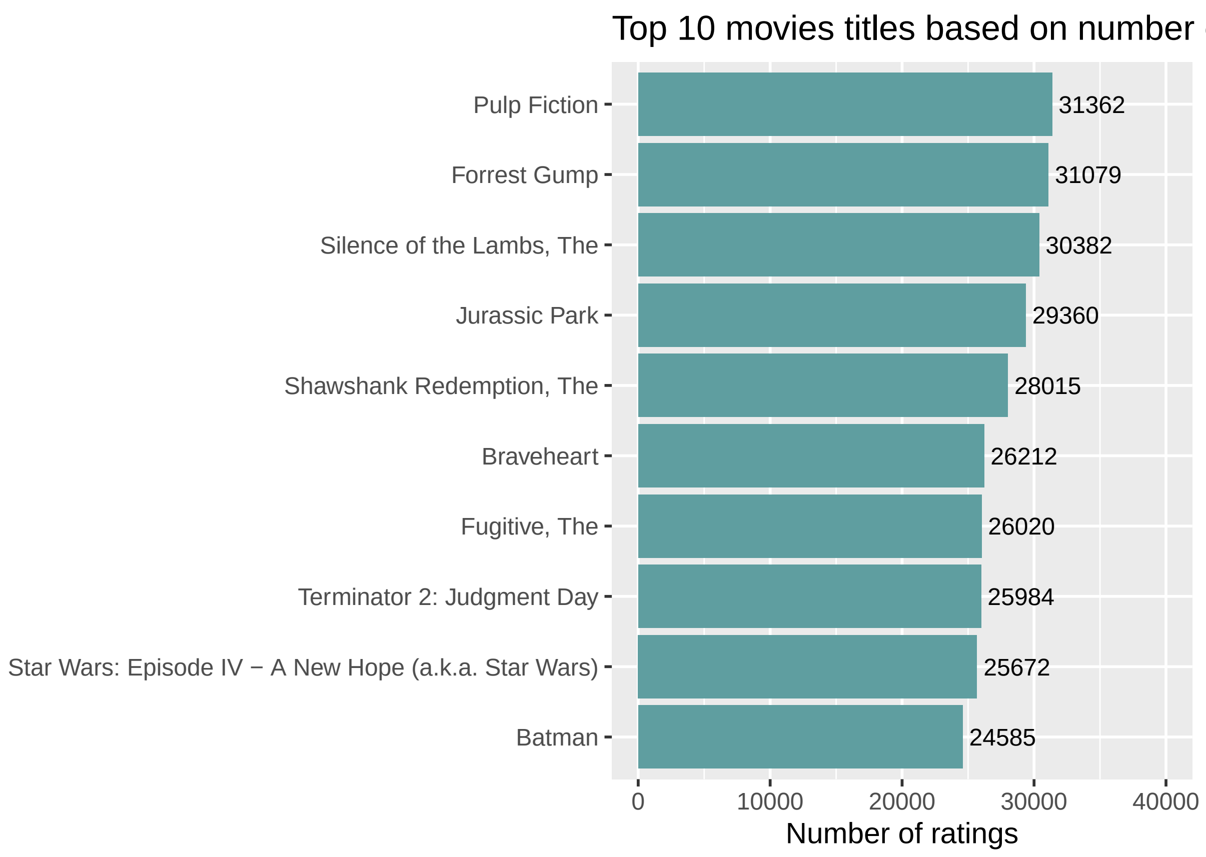
```
movies_count<-(edx %>% group_by(title) %>%summarize(count=n()))
sprintf("Movies with a single rating = %d",sum(movies_count$count==1))
```

```
## [1] "Movies with a single rating = 125"
```

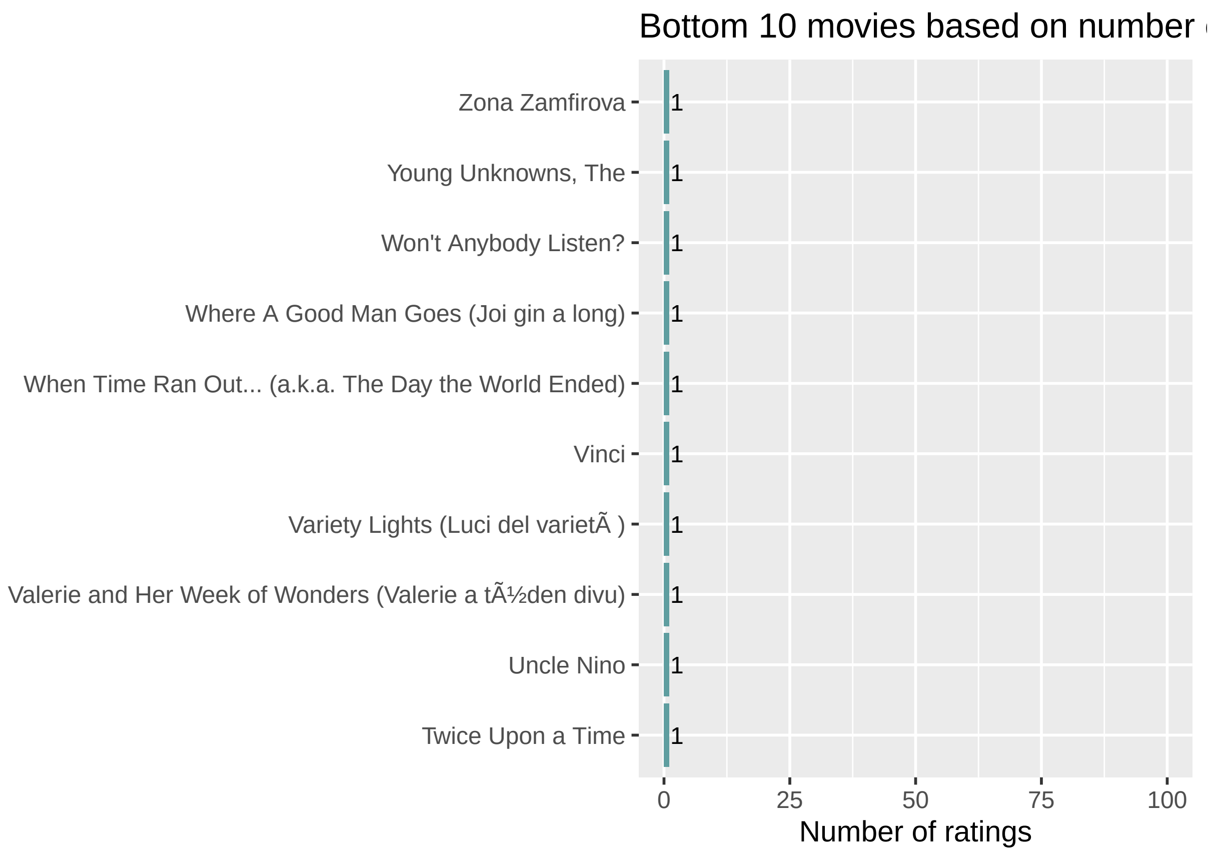
```
top_movies <- edx %>%
  group_by(title) %>%
  summarize(count=n()) %>%
  top_n(10) %>%
  arrange(desc(count))
```

```
## Selecting by count
```

```
top_movies %>%
  ggplot(aes(x=reorder(title, count), y=count)) +
  geom_bar(stat='identity', fill="cadetblue") + coord_flip(y=c(0, 40000)) +
  labs(x="", y="Number of ratings") +
  geom_text(aes(label= count), hjust=-0.1, size=3) +
  labs(title="Top 10 movies titles based on number of ratings" )
```



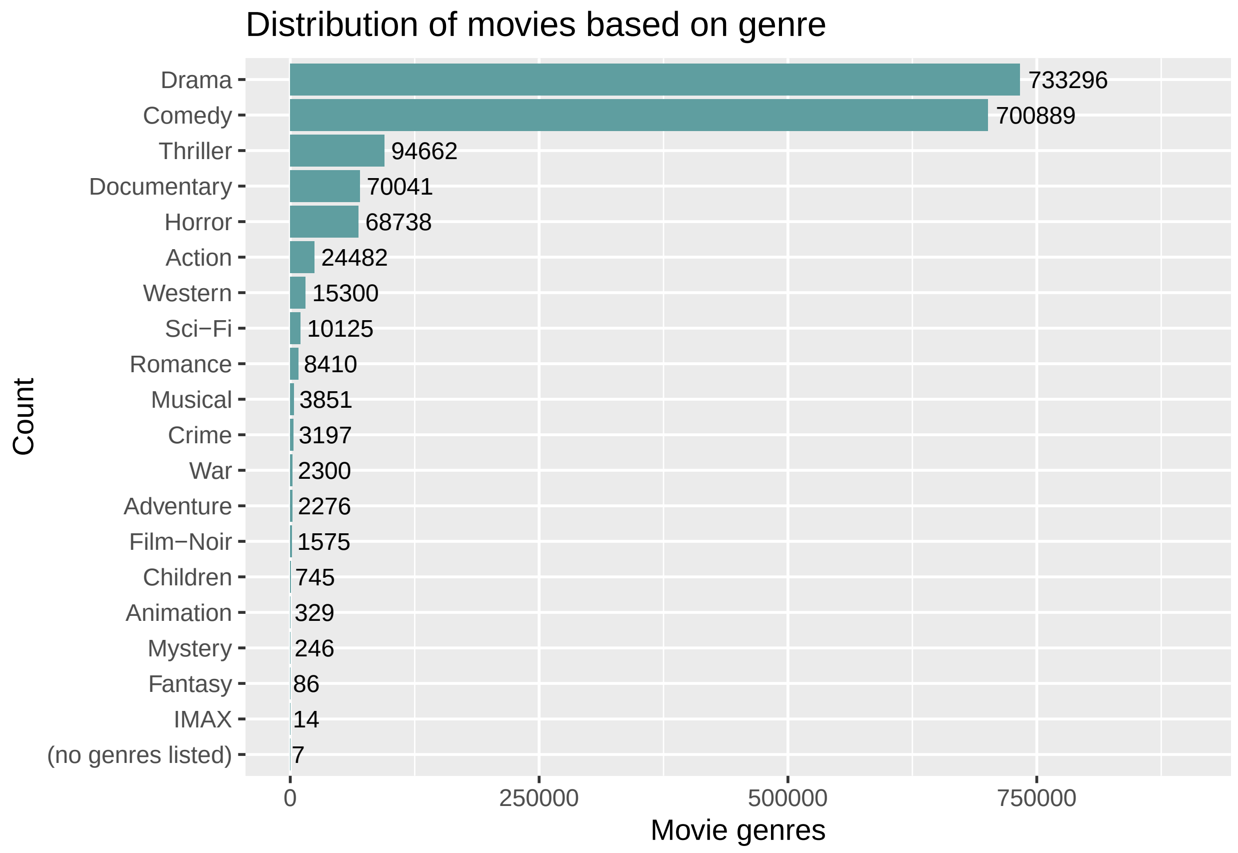
```
bottom_movies <- edx %>%
  group_by(title) %>%
  summarize(count=n()) %>%
  arrange(desc(count))

bottom_movies %>% tail(10) %>%
  ggplot(aes(x=reorder(title, count), y=count)) +
  geom_bar(stat='identity', fill="cadetblue") + coord_flip(y=c(0, 100)) +
  labs(x="", y="Number of ratings") +
  geom_text(aes(label= count), hjust=-0.1, size=3) +
  labs(title="Bottom 10 movies based on number of ratings")
```



4.4 Generes

Each movie can have more than one genre. A close analysis indicates that most movies in the dataset belong to `Comedy" and`Drama" genres. A very few movies belong to the "fantasy" genre. Therefore, we can also use this feature as a penalty term in our model (if required). Moreover, there are some movies with untitled genres. We can remove them if needed.

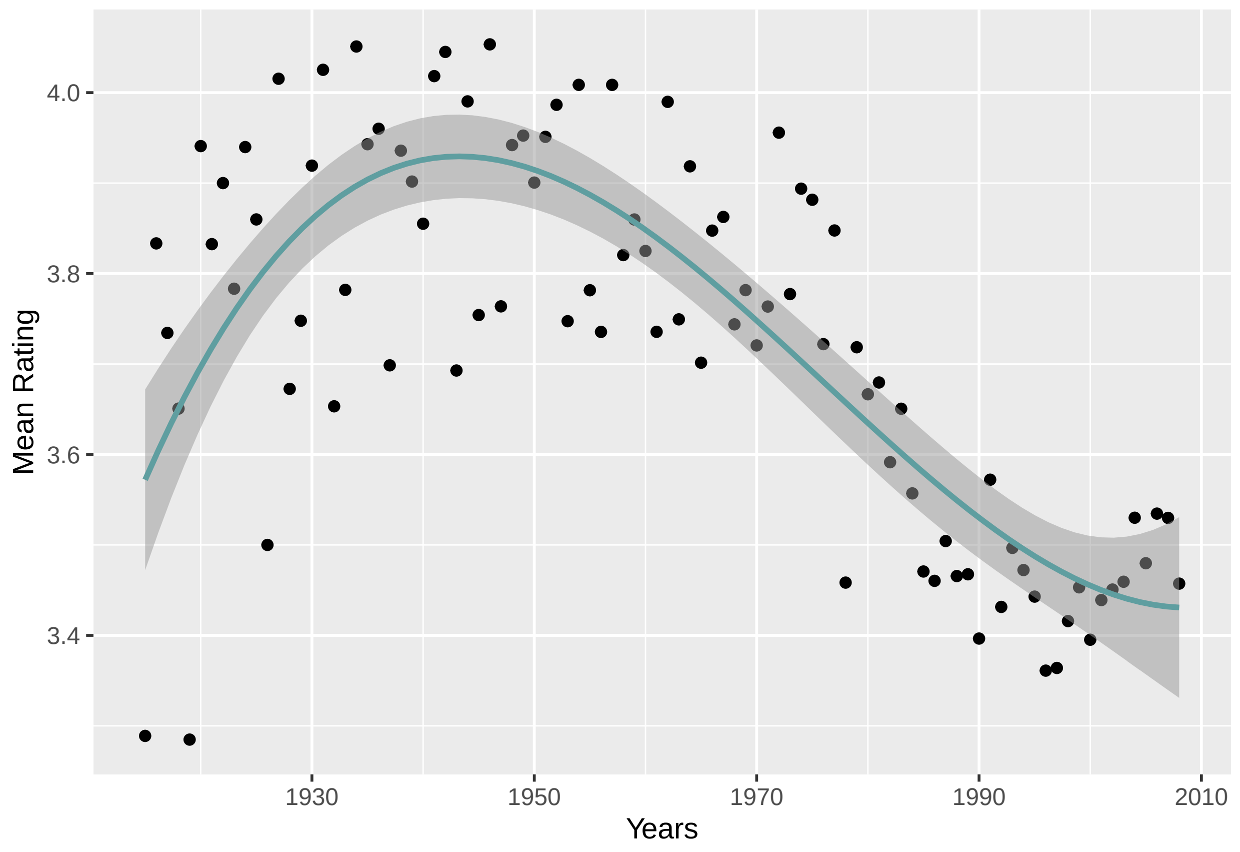
```
# Extracting all unique combination of genres
unique_genres_combination<-edx%>% group_by(genres)%>%summarize(count = n())

# Extract all possible genres (which exist in the dataset)
movie_genres <- unique(unlist(strsplit(unique(edx$genres), split='|', fixed=TRUE)))
movie_genres <- as.data.frame(movie_genres)
movie_genres<-movie_genres %>% add_column(count = 0)

# Looping through to find the freqeucny or distribution of each genre
for(i in 1:dim(movie_genres)[1]) {
  for(j in 1:dim(unique_genres_combination)[1])
  {
    match = as.character(unique_genres_combination[j,1])
    count = as.numeric(unique_genres_combination[j,2])
    # See if genre category exists in the combination
    if (str_contains(movie_genres[i,1],match))
    {
      movie_genres[i,2]<-movie_genres[i,2]+count
    }
  }
}

# Plot distrobution of movie genres
movie_genres%>%
  ggplot(aes(x=reorder(movie_genres, count), y=count)) +
  geom_bar(stat='identity', fill="cadetblue") + coord_flip(y=c(0, 900000)) +
  labs(x="Count", y="Movie genres") +
  geom_text(aes(label= count), hjust=-0.1, size=3) +
  labs(title="Distribution of movies based on genre" )
```



4.5 Premier Year

We analyse the rating trends of users over the years. Interestingly, we find out that mean ratings have dropped over the years. This indicates that users are not rating more movies in recent years.

```
edx %>% group_by(premiereYr) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(premiereYr, rating)) +
  geom_point() +
  labs(x="Years", y="Mean Rating")+
  geom_smooth(alpha=0.5, color='cadetblue',method = lm,formula = y ~ splines::bs(x, 3))
```



5 Methods

We will use RMSE to estimate the quality of our model. We define $y_{u,i}$ as the rating for the movie i and donate prediction by $\hat{y}_{u,i}$ as shown below:

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

For our problem, as the rating for movie

Where N is the number of user/movie combinations. Using Equation 2, we write the following function to compute the RMSE:

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

The lower RMSE better is the model. The evaluation criteria for this project is that the RMSE of the proposed model should be less than 0.8649.

```r
#Initiate RMSE results to compare various models
rmse_results <- data_frame(method = "Target RSME", RMSE = 0.86490)
```

```
## Warning: `data_frame()` is deprecated, use `tibble()`.
## This warning is displayed once per session.
```

```
rmse_results
```

method	RMSE
Target RSME	0.8649

5.1 Average Movie Rating Model

We start with the simplest model by predicting the same ratings for all movies regardless of users or frequency of ratings. So, we compute the dataset's mean rating:

$$Y_{u,i} = \mu + \epsilon_{u,i} \tag{3}$$

Where $\epsilon_{u,i}$ is the independent error sample from the same distribution centered at 0 and μ the true rating for all movies. This model uses the assumption that differences in movie ratings are explainable by random variation alone. The expected value (μ) of the data is around 3.5. We make predictions using μ, and the resulting RMSE of the model is 1.06 on validation data.

```
# Calculating mean of ratings
mean_rating <- mean(edx$rating)
mean_rating
```

```
## [1] 3.512465
```

```
# Making predictions using mean rating
mean_model_rmse <- RMSE(validation$rating, mean_rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Average Rating Model",
                                     RMSE = mean_model_rmse ))
rmse_results
```

method	RMSE
Target RSME	0.864900
Average Rating Model	1.061202

5.2 Model with movie effect

The RMSE of our previous model is very high compared to the target RMSE. To improve the RMSE of our model, we have to include extra parameters in our model. We recall from Section **??** that different movies are rated differently. The popular movies are rated much higher than unpopular movies. We modify the previous model by adding the term 'b_i" to represent the average ranking for movie i as follows:

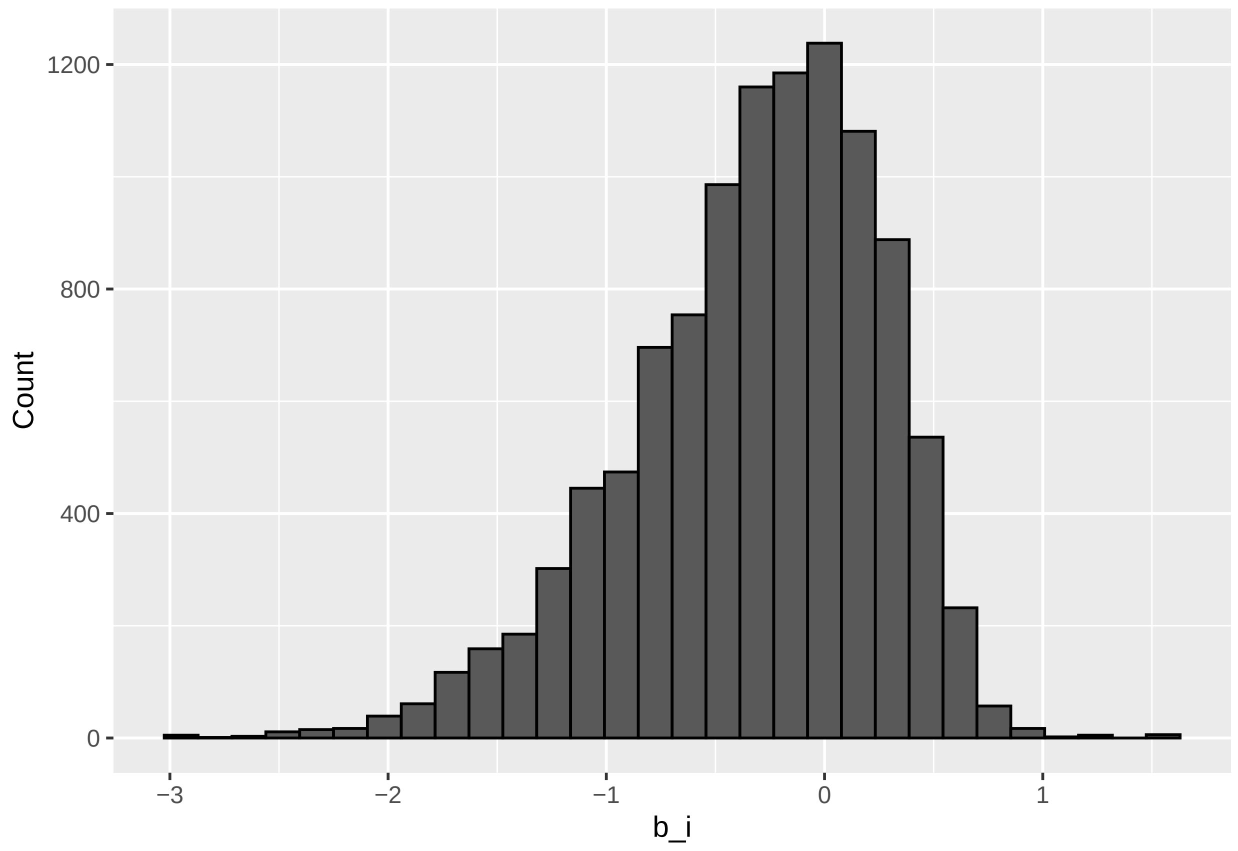
$$Y_{u,i} = \mu + b_i + \epsilon_{u,i} \tag{4}$$

The least-squares estimate b_i is just the average of $Y_{u,i} - \mu$ for each movie i. We can compute it as follows:

```
# Computing average ranking for movie "i"
movie_averages <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mean_rating))
```

The histogram of average rankings for movies is not symmetric and is skewed to the left (towards negative rating effect). This is known as the penalty term movie effect and supports the idea of using it in our model.

```r
# Plotting average ranking estimates for movies
movie_averages%>%
  ggplot(aes(b_i)) +
  labs(x="b_i", y="Count")+
  geom_histogram(bins = 30, color = "black")
```



If one movie is on average rated better than the average rating of all movies μ, then we predict that it will be rated higher than μ by b_i.By including the movie effect, the RMSE of the model improves to 0.9439 on validation data.

```r
# Making predictions using new model
predicted_ratings <- mean_rating +  validation %>%
  left_join(movie_averages, by='movieId') %>%
  pull(b_i)

# Calculating RMSE
model_movie_effects_rmse <- RMSE(predicted_ratings, validation$rating)

# Appending results to RMSE table
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie effect model",
                                     RMSE = model_movie_effects_rmse ))
rmse_results
```

method	RMSE
Target RSME	0.8649000
Average Rating Model	1.0612018

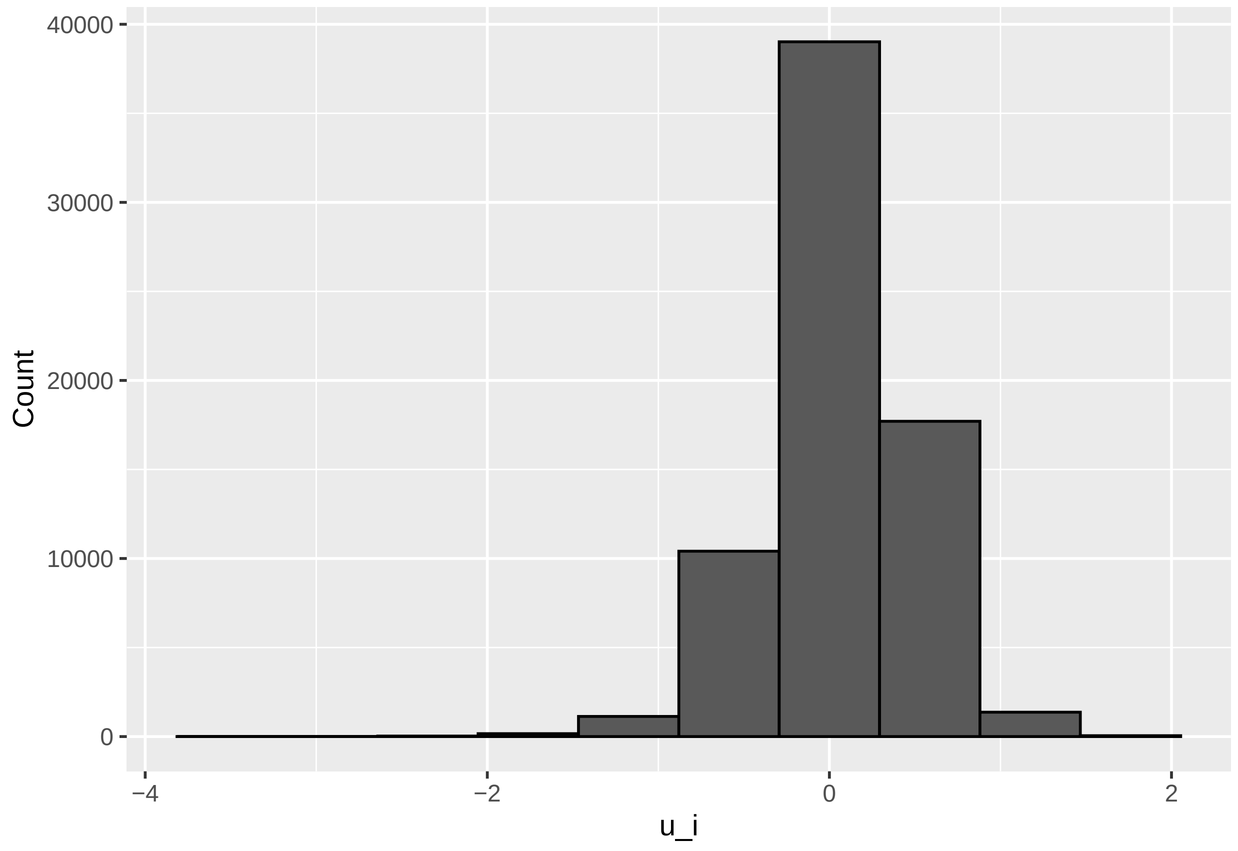
method	RMSE
Movie effect model	0.9439087

6 Model with user effects

We know that some users are more active in rating than others. We compute the average rating for user u, who have rated over 100 movies. We notice a substantial variability across users. Some users are very cranky (like few movies), while others love every movie. Therefore, we can further improve our previous model by adding user-specific effect (b_u):

$$Y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i} \tag{5}$$

```r
# Calculating average rating for user u
user_averages <- edx %>%
  left_join(movie_averages, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mean_rating - b_i))

# Plotting average rating for users
user_averages %>%
  ggplot(aes(b_u)) +
  labs(x="u_i", y="Count")+
  geom_histogram(bins = 10, color = "black")
```



If a cranky user (negative b_u) rates a great movie (positive b_i), the effects counter each other. And we may be able to correctly predict that this user gave this great movie a 3 rather than a 5. By using the new model for prediction on the validation dataset, the RMSE improves to 0.8653.

```r
# Making predictions using new model
predicted_ratings <- validation%>%
  left_join(movie_averages, by='movieId') %>%
  left_join(user_averages, by='userId') %>%
  mutate(pred = mean_rating + b_i + b_u) %>%
  pull(pred)

# Calculating RMSE on validation dataset
model_user_effect_rmse <- RMSE(predicted_ratings, validation$rating)

# Appending results to RMSE table
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie and user effect model",
                                     RMSE = model_user_effect_rmse))
rmse_results
```

method	RMSE
Target RSME	0.8649000
Average Rating Model	1.0612018
Movie effect model	0.9439087
Movie and user effect model	0.8653488

6.1 Regularization using movie and user effects

6.2 sec:reg

We made mistakes on our first model when we used the effect of movies only.

1. There are users who have rated very few movies (less than 30 movies). On the other hand,
2. Some movies are rated very few times (as low as 1).

These are noisy estimates that result in overfitting of the model. Because the model tries hard to fit the noisy estimates, such large errors are likely to increase the RMSE and affect the predictions.

To solve this problem, we use Regularization. The Regularization permits to penalize large estimates that come from small sample sizes. The general idea of penalized regression is to control the total variability of the movie effects. Instead of minimizing the least squares equation, we minimize an equation that adds a penalty that gets larger when many b_i are large:

$$\frac{1}{N}\sum_{u,i}(y_{u,i}-\mu-b_i)^2+\lambda\sum_i b_i^2 \tag{6}$$

The values of b_i that minimise the above equation are :

$$\hat{b}_i(\lambda)=\frac{1}{\lambda+n_i}\sum_{u=1}^{n_i}(Y_{u,i}-\hat{\mu}) \tag{7}$$

Where n_i is the number of ratings made for movie i. This approach will have our desired effect because when our sample size n is very large, then we will get a stable estimate, and the penalty λ is effectively ignored.

However, with a small sample size, the estimate $\hat{b}_i(\lambda)$ will shrink towards zero. We can use regularization to estimate of user effects as well as shown below:

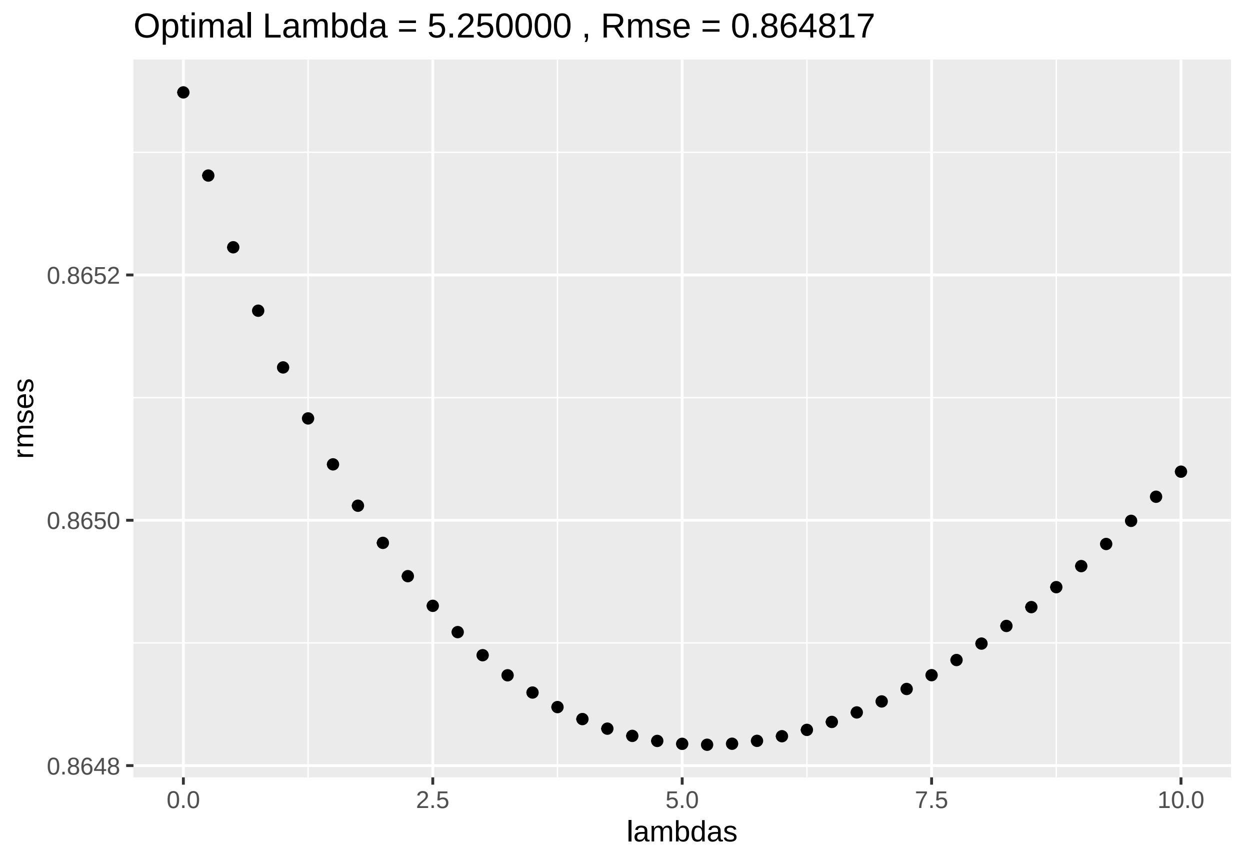
$$\frac{1}{N}\sum_{u,i}(y_{u,i}-\mu-b_i-b_u)^2+\lambda\left(\sum_i b_i^2+\sum_u b_u^2\right) \tag{8}$$

We can see that it is important to find an optimal value of our tuning parameter λ which will minimise the value of RMSE. We find optimal λ giving us the best RMSE score as follow.

```
# Possible Lambda values
lambdas <- seq(0, 10, 0.25)

# Calculating mean of model
mean_rating <- mean(edx$rating)
rmses <- sapply(lambdas, function(l){
  # Penalised movie effect
  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mean_rating)/(n()+l))
  # Penalised user effect
  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mean_rating)/(n()+l))
  # Predicting new ratings
  predicted_ratings <-
    validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mean_rating + b_i + b_u) %>%
    pull(pred)

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

The optimal lambda value is 5.25, which gives us an RMSE of 0.864817.

```
lambda <- lambdas[which.min(rmses)]
qplot(lambdas, rmses)+
  ggtitle(sprintf("Optimal Lambda = %f , Rmse = %f",lambda,min(rmses)))
```



```r
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized movie and user effect model",
                                     RMSE = min(rmses)))
```

7 Results

In Section 5, we presented and discuss various models. We started with a very simple model by predicting the mean regardless of the user. Then, we realised that some movies are rated less compared to popular movies. Moreover, some users love every movie and give a higher rating to the movies. This led us to use movie and user effects with our model to improve the performance. But small estimates can result in overfitting of our model. Then, we use regularisation to produce a model with the lowest RMSE.

```r
rmse_results
```

method	RMSE
Target RSME	0.8649000
Average Rating Model	1.0612018
Movie effect model	0.9439087
Movie and user effect model	0.8653488
Regularized movie and user effect model	0.8648170

The RMSE of our best model is 0.8648170, which is lower than the **Target RMSE**. This is also a significant improvement compared to the first simple model i.e., around 18.5 % decrease in RMSE.

8 Conclusion

In this report, we developed recommendation algorithms to predict movie ratings using Movielens data. There is a lot of variability in our data, such as popular movies are rated by many users, obscure movies have a very low rating, some users like every movie, and some users give more negative ratings. These factors prompted us to use average ratings of movies and users with our model. We noticed a significant improvement to the RMSE compared to our first model. Then, we used regularization to handle the overfitting of our model and gained a slight increase in the RMSE score.

8.1 Future Work

We did not use the genre's information in our model. Maybe we can use that in a future model by penalizing movies belonging to less common genres. Similarly, we can compute the release year, which has the lowest number of rated movies. Then, we can ignore movies from that year in our model. But, we have to see the impact of these changes on the performance of our model.

References

[1] Baptiste Rocca -Introduction to recommender systems https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada

[2] Resnick, P., Varian, H. *Recommender Systems.* CACM 40(3), 56–58 (1997)

[3] Netflix Prize https://en.wikipedia.org/wiki/Netflix_Prize

[4] MovieLens 10M Dataset https://grouplens.org/datasets/movielens/10m

[5] James Moody - What does RMSE really mean? https://towardsdatascience.com/what-does-rmse-really-mean-806b65f2e48e

[6] Rafael Irizarry - Introduction to Data Science https://rafalab.github.io/dsbook