MovieLens Rating Prediction Project

Nabeel Khan

16-May-2020

Contents

1 Introduction

This report is part of the 'HarvardX: PH125.9x Data Science: Capstone' course. The goal of this work is to use the acquired data science skills to develop a recomender system.

1.1 Background

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

One of the famous success story of the recommender system is the Netflix competition[3]. In 2006, Netflix announced to award 1 milloin dollar prize to a team, which can come up with the best filtering algorithm to predict user ratings for movies based on previous ratings. The winners were announced in 2009 and the winning team improved the accuracy of Netfliix system system by 10%.

1.2 Aim of Project

The aim of this project is to dveleop a recomender system using machine learning to predict movie ratings based on suitable predictors.

2 Dataset and Evaluation Metric

Netflix dataset is not available online. So, we will use 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 validaiton sets to develop the machine lelarning algorithm.

The performace of the model will be evaluated with the Root Mean Square Error (RMSE) [5]. The RMSE is a standard method to measure the error of a model in predicting quantitative data and is defined in Equation 1.

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

$\hat{y}_i$ refers to the predicted values by the model, y_i refers to the actual values, and n refers to the total number of oservations. The RMSE is one of the very useful metrics to analyse the performance of models, but it can produce biased results in the presence of large number of outliers. In this project, the RMSE will indicate how close model predictions are to the actual ratings in the validation set.

2.1 Download Data

After downloading data from the website, we split it into training set referred to as *edx* and test set referred to as *validation*. The *edx* data consist of 90% of the data, while 10% of data is used for validation.

- The model will be trained using *edx* dataset and *validation* dataset will be used to compute the RMSE.

```
################################
#  Install packages (if not installed)
################################
# 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 1.3.0 --

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

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

```
## Loading required package: caret

## Loading required package: lattice

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

```r
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
```

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

```
## 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 = "http://cran.us.r-project.org")


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

```r
#################################
# 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 no longer required
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 also perform some data wrangling. Data wrangling is the process of cleaning, structuring or enriching raw data into a desired format to make better decisions and get more meaningful insights.

We look at the first six rows of *edx* dataset. We can see that dataset contains six variables namely `userID'',`movieID”‘, `rating'',`timestamp’‘, `title'', and`genres”. Each row represents a single rating of a user for a single movie.

```
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

	userId	movieId	rating	timestamp	title	genres
7	1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy

There are around nine million rows in *edx* dataset, which is around 90% of the whole data.

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

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

The summary of *edx* dataset indicates that there are no null values. So, we don't need to perform any data cleaning.

```
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

The *edx* dataset has a couple of issues:

1. The *"title"* of each movie has its premier year appended to it.
2. The *"timestamp"* contains the date on which user added his rating. But, the column shows the number of seconds that have elapsed since January 1, 1970.

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 strip out the year information from *"title"* column.
2. Modify the *"timestamp"* so that it contains the information in date format, which is easy to interprete.

The summary of *edx* dataset shows that we have successfully performed data wrangling operations.

```
########################################
# EDX Dataset
########################################
# Extracting premier date from movie title
edx <-edx %>% extract(title, c("title", "premiereYr"),
                      regex = "^(.*) \\(([0-9 ##\\-]*)\\)$")
edx$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 premier date from movie title
validation <-validation %>% extract(title, c("title", "premiereYr"),
                                    regex = "^(.*) \\(([0-9 ##\\-]*)\\)$")
validation$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 operations. In this section, we extract insights about all features of the *edx* dataset.

4.1 Users

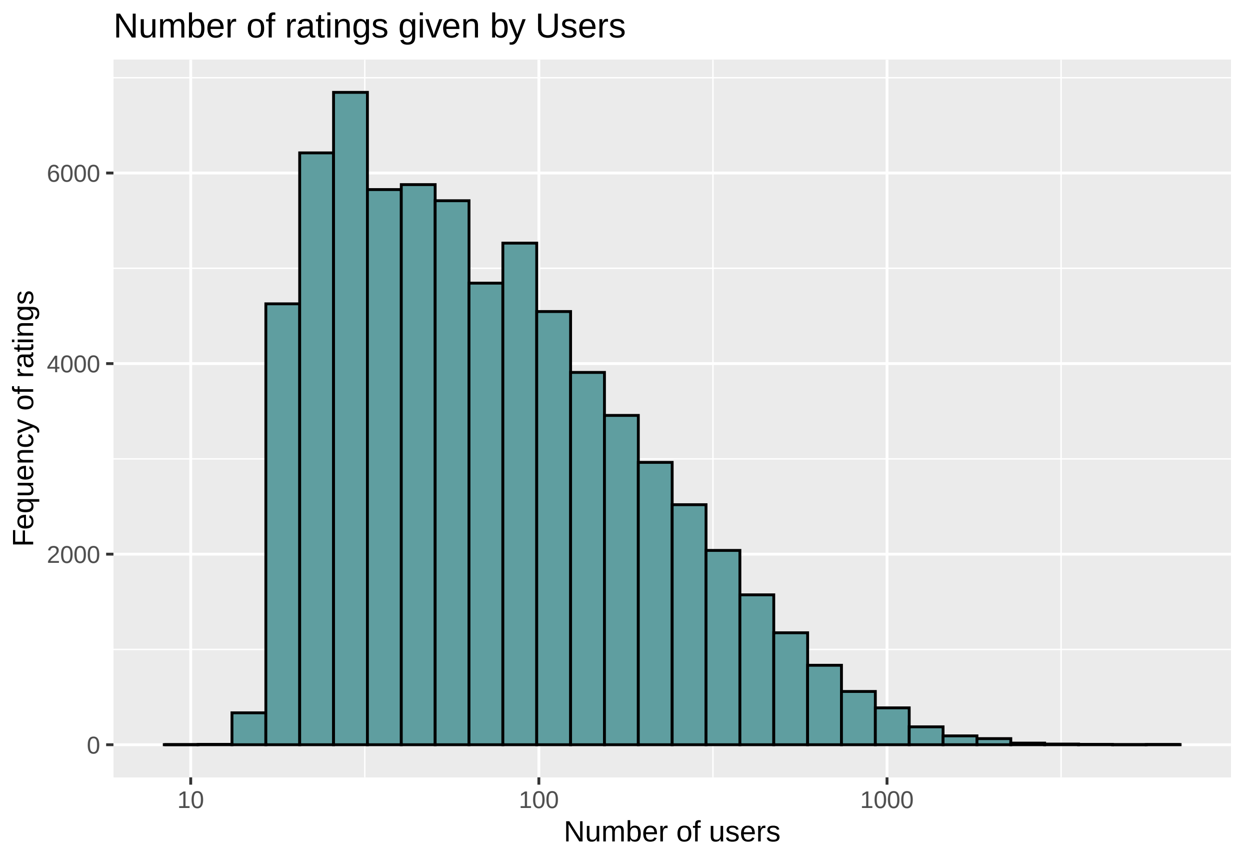
The dataset has around 69800 unique users. Each user can rate more than one movie. If we look at the frequency of ratings for users, we can see that the distribution is rightly skewed. Most of the users rate few movies as shown in the graph. So, we can include a user penalty term in our model (if needed) to improve the RMSE. The process of adding penatly terms to error functions is known as **Regularization** [6].

Regularization permits to penalize large estimates that are formed using small sample sizes and avoids overfitting of data. The overfitting occurs because model is trying too hard to capture the noise in the training dataset. The basic idea behind regularization is to constrain the total variability of the effect sizes.

```
edx %>%
summarize(users = n_distinct(userId))
```

users
69878

```
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("Number of ratings given by Users")
```



4.2 Movies

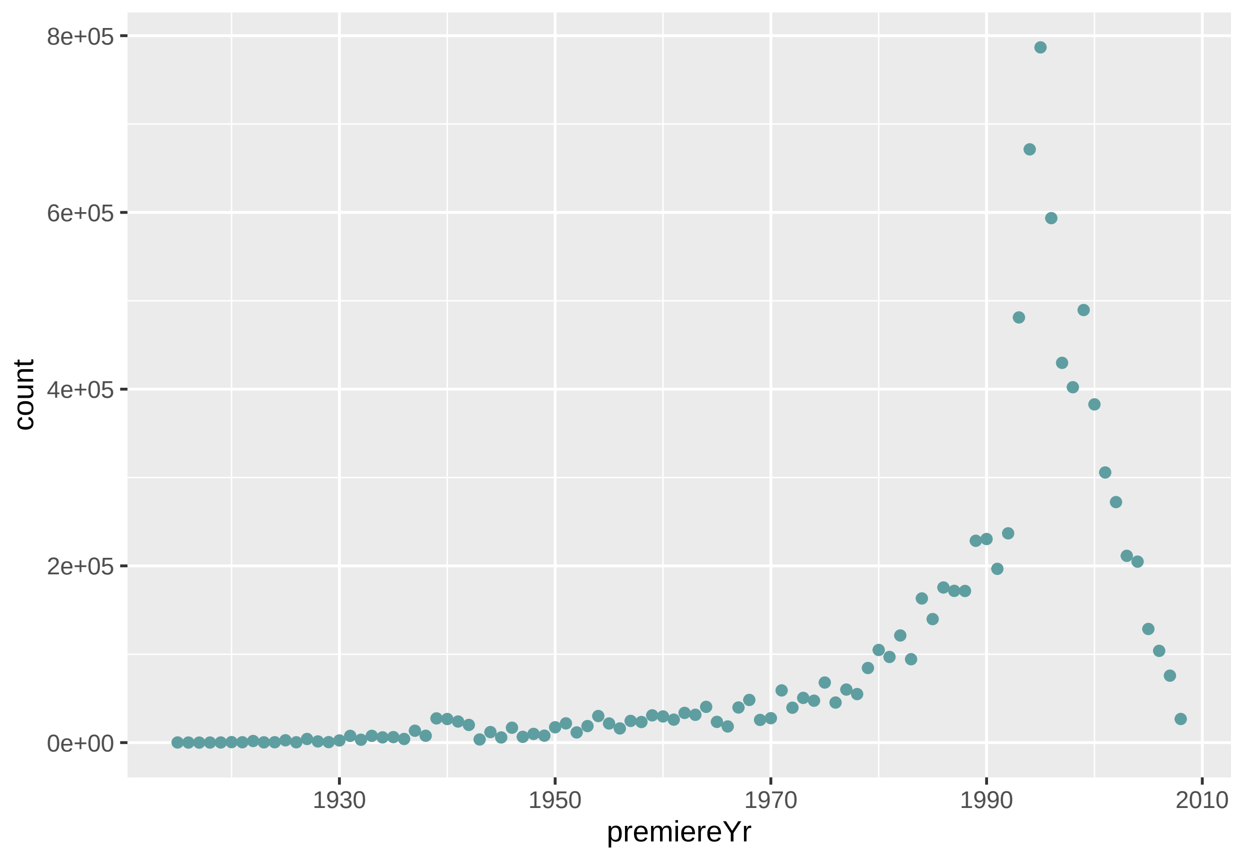
The dataset has around 10400 unique movies.

```r
edx %>%
summarize(users = n_distinct(title))
```

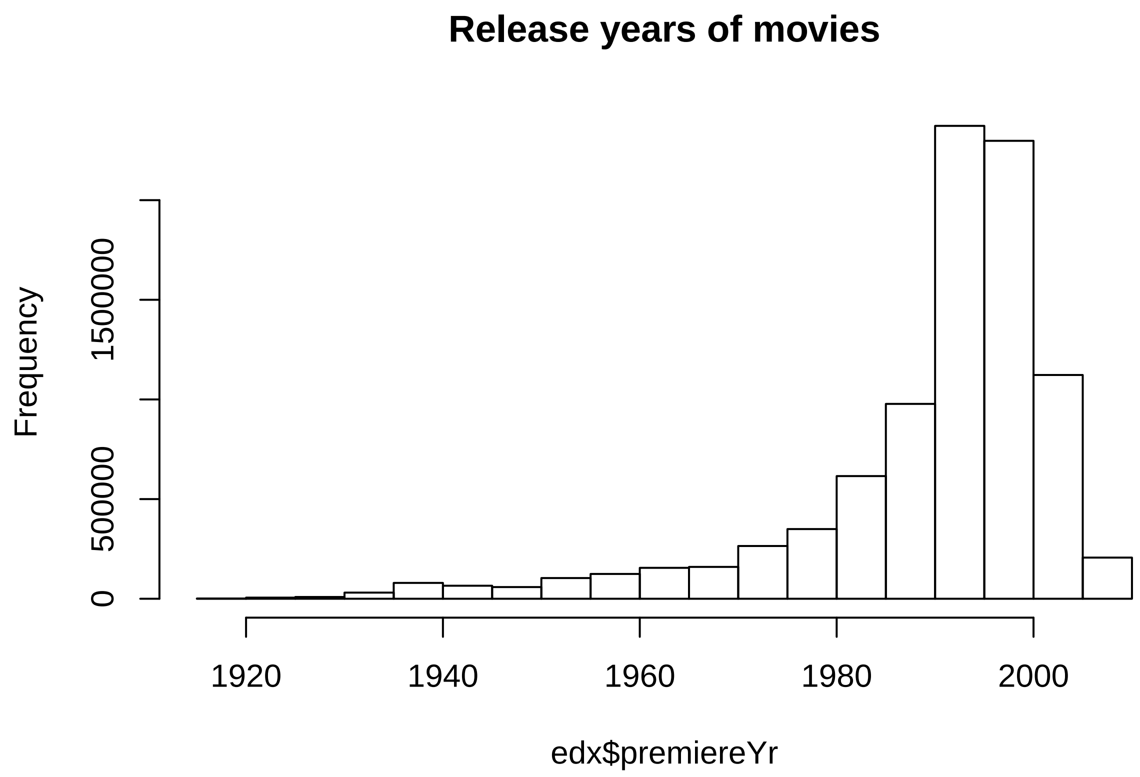
users
10407

The release years for movies in *edx* range 1915 to 2008. Most movies belong to recent years ranging from 1980 to 2000. Though, it would be nice to have more movies from old years to make model more robust.

```r
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)) +
  geom_point(color="cadetblue")
```

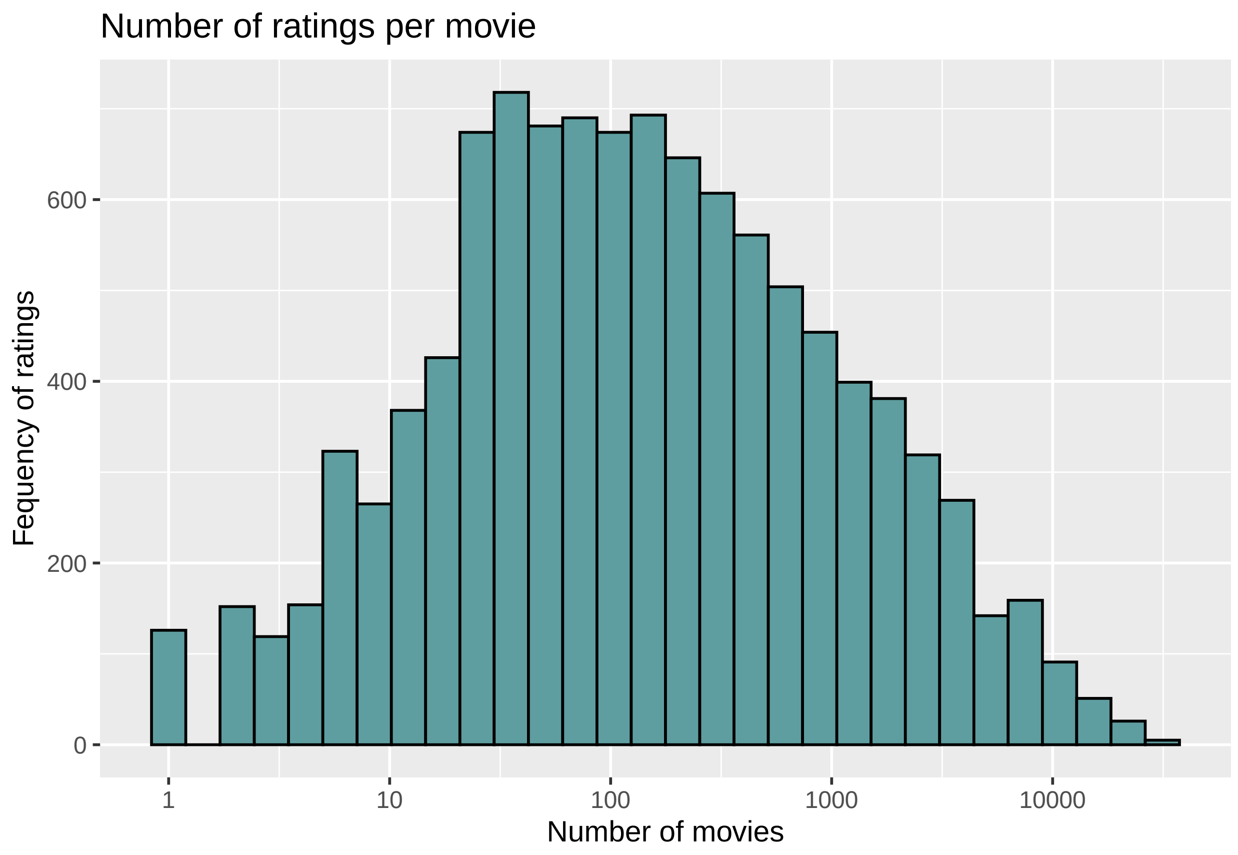


```
hist(edx$premiereYr, main="Release years of movies")
```



The distribution of ratings for movies indicate that most movies get a decent number of ratings. Although some movies have less number fof ratings.This indicates that movie information can also be used as penalty term (if required).

```
#Distribution of Movie Ratings
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("Number of ratings per movie")
```



4.3 Ratings

We observe that ratings data is collected over a span of around 14 years. All movies have at-least one rating. We also observe that there is a tendency that users generally rate movies 3 or 4.

```
years<-format(edx$timestamp, format="%Y")
as.numeric(max(years))-as.numeric(min(years))
```

```
## [1] 14
```

The movies can have half or full stars ratings. The distribution indicates that most movies get full star ratings.

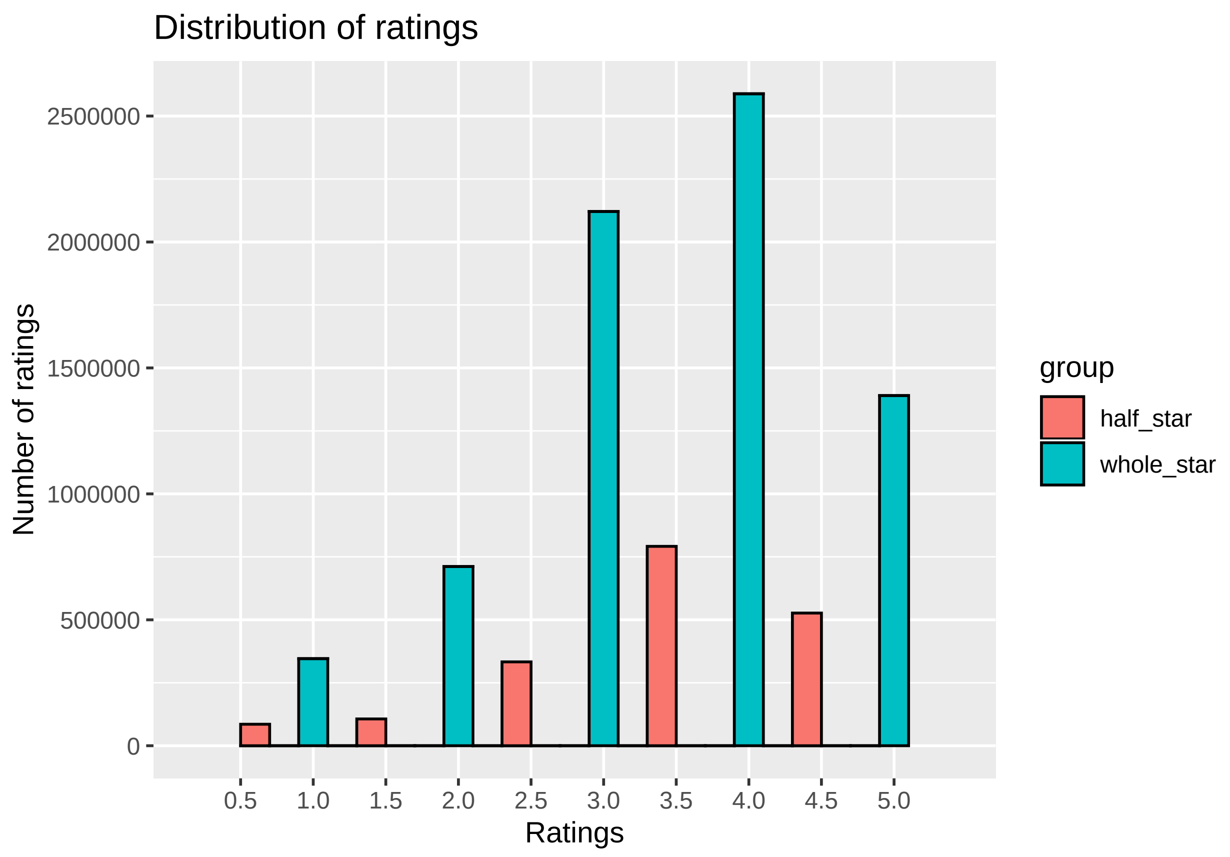
```
print("Frequencies of ratings")
```

```
## [1] "Frequencies of ratings"
```

```
sort(table(edx$rating))
```

```
##
##     0.5     1.5     2.5       1     4.5       2     3.5       5       3
##   85374  106426  333010  345679  526736  711422  791624 1390114 2121240
##       4
## 2588430
```

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

# Plotting the distribution of ratings
ratings_distribution <- data.frame(edx$rating, group)
ggplot(ratings_distribution, aes(x= edx.rating, fill = 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")
```



Now, we explore the top 10 and bottom 10 movies based on total number of ratings. We see that some movies are rated more often than the others. For instance, 120 movies are rated by one user only.

Top 10 movies are well-known and are rated by many users. But, the bottom 10 movies appear to be obscure and are only rated by one user. Therefore, the predictions of future ratings for such movies will be difficult.

This information is important for our model because a very low rating numbers can result in overfitting for our predictions. Therefore, the use of regularization by including this information as a penalty term in our model (if needed) can improve the RMSE of our model.

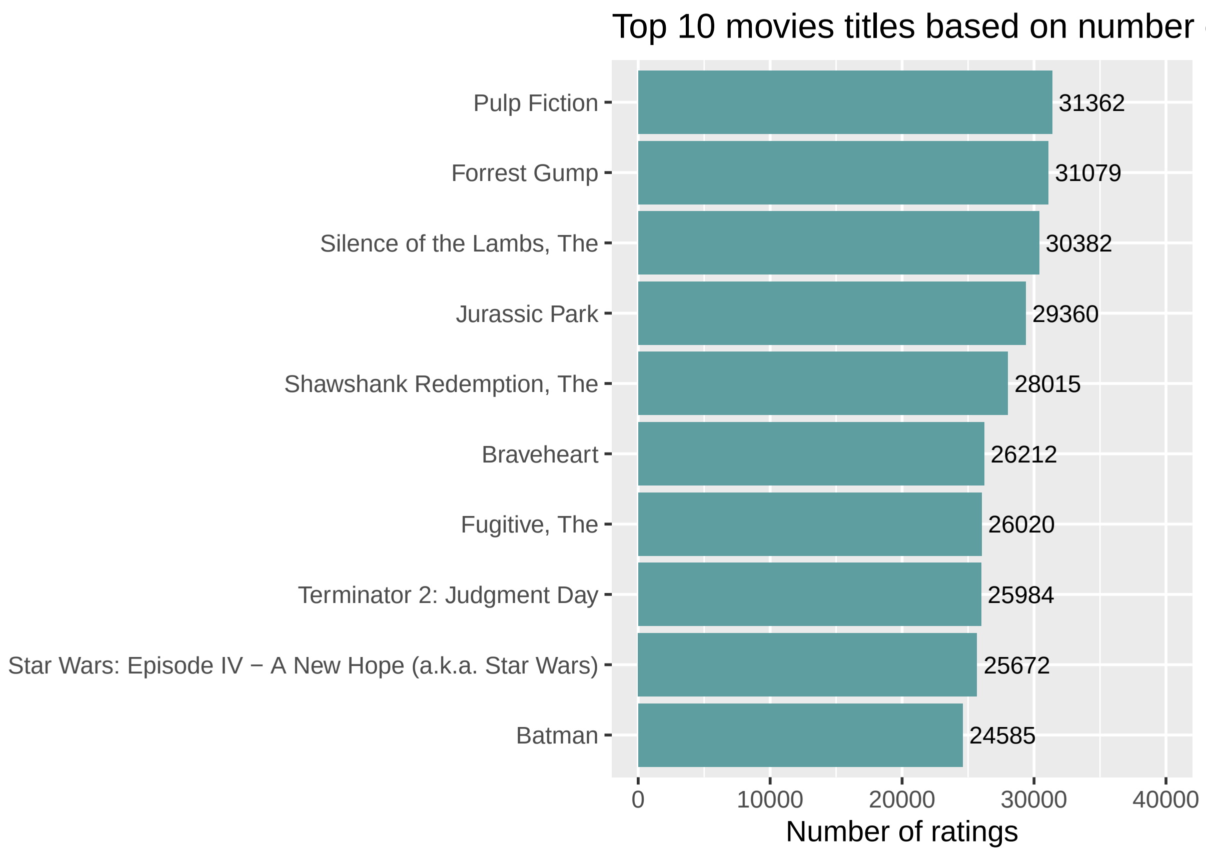
```
movies_count<-(edx %>%
  group_by(title) %>%summarize(count=n()))


sprintf("Movies with a single rating = %d",sum(movies_count$count==1))
```

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

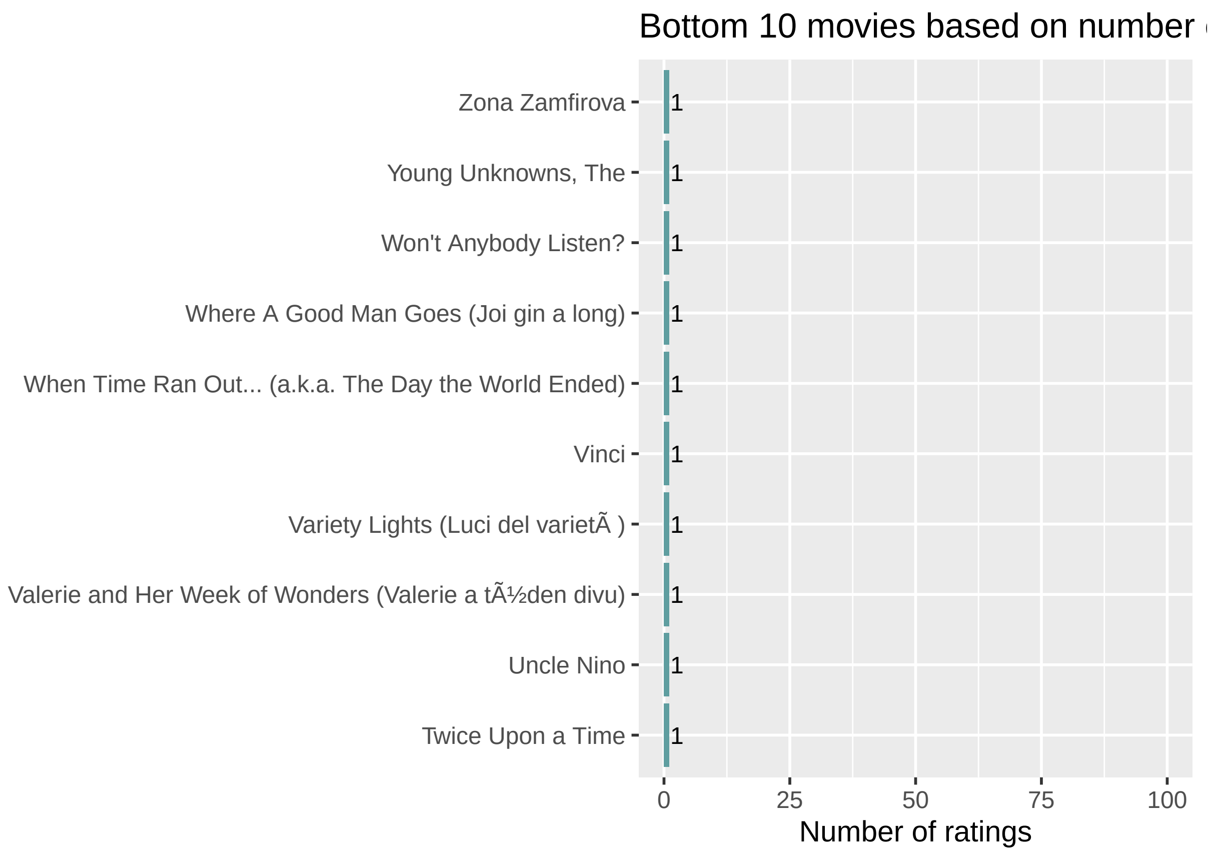
```r
top_movies <- edx %>%
  group_by(title) %>%
  summarize(count=n()) %>%
  top_n(10) %>%
  arrange(desc(count))
```

```
## Selecting by count
```

```r
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" )
```



```r
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

Most movies belong to Drama and Comedy generes.A very few movies belong to horror genere. Though, this information does not add much compared to previous features. Therefore, we will not use "genere" feature for our model building.

```
generes <- c("Drama","Comedy","Thriller","Romance","Sci-Fi","Adventure","War","Action","Horror")
sapply(generes, function(x)
{
 sprintf("%s = %d",x,sum(grepl(x,edx$genres)))
}
)
```

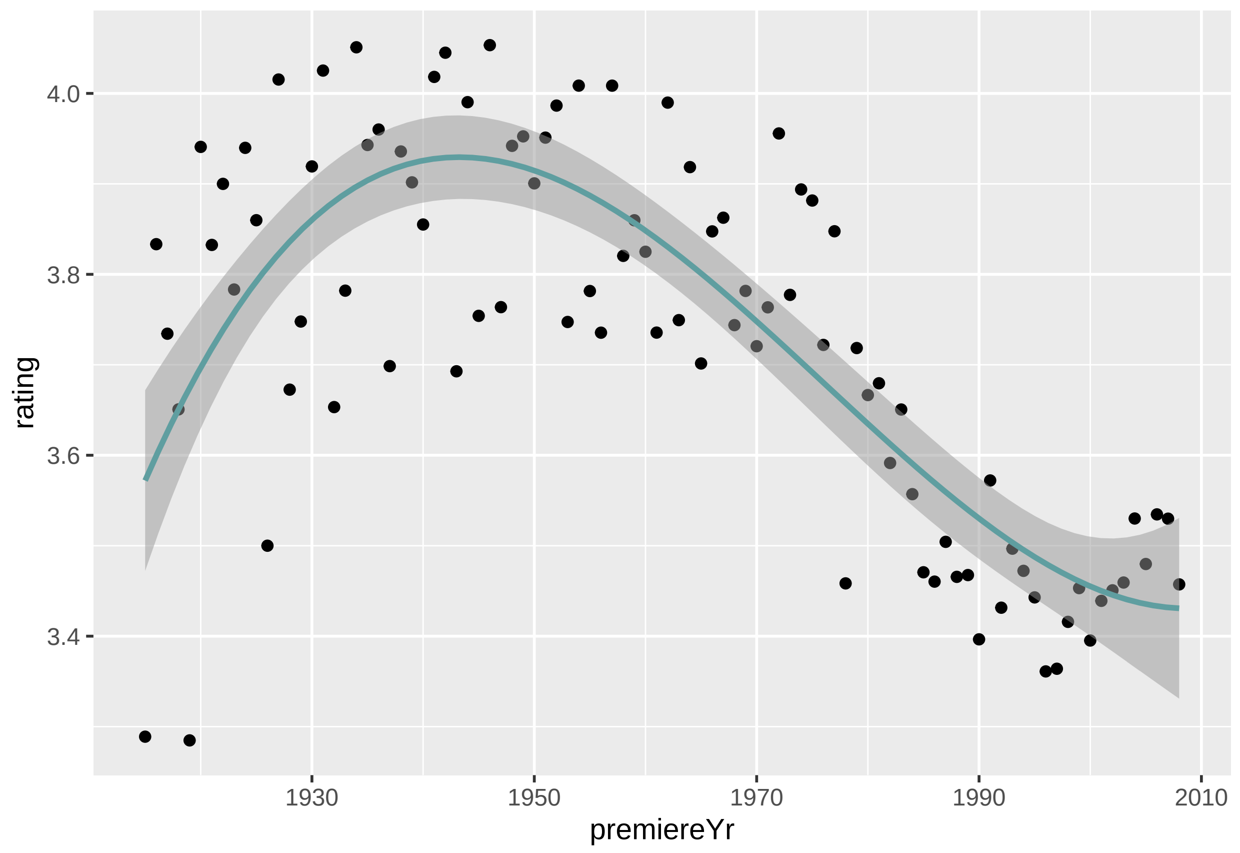
```
##                 Drama                Comedy              Thriller
##     "Drama = 3910127"    "Comedy = 3540930"  "Thriller = 2325899"
##               Romance                Sci-Fi             Adventure
##   "Romance = 1712100"    "Sci-Fi = 1341183" "Adventure = 1908892"
##                   War                Action                Horror
##        "War = 511147"    "Action = 2560545"     "Horror = 691485"
```

4.5 Premier Year

We analyse the rating trends of users over the years. Interestingly, we can see 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)) %>%
```

```r
ggplot(aes(premiereYr, rating)) +
geom_point() +
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. Using equation 1, we write the following function to compute the RMSE between actual ratings and predicted ratings::

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

The lower RMSE is better.The evaluation criteria for this project is that 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 = 0.8649)
```

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

```r
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 freqeuncy of ratings. In other words, we compute the dataset's mean raitng as follows:

$$Y_{u,i} = \mu + \epsilon_{u,i} \tag{2}$$

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 is based on the assumption that differences in movie ratings can be explained by random variation alone. The expected value of the data is around 3.5 and we obtain a RMSE of 1.06.

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

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

We make predictions using μ and obtain our firt RMSE score of 1.06.

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

method	RMSE
Target RSME	0.864900
Average Rating Model	1.061202

In order to improve the RMSE of our model, we have to include extra parameters in our model. We can use some of the insights obtained in previous section, such as user effect.

5.2 Movie Effect Model

We recall from Section **??** that different movies are rated differently. The popular movies are rated much higher than unpopular movies. We can modify our previous model by adding the term 'b_i" to represent average ranking for movie i as follows:

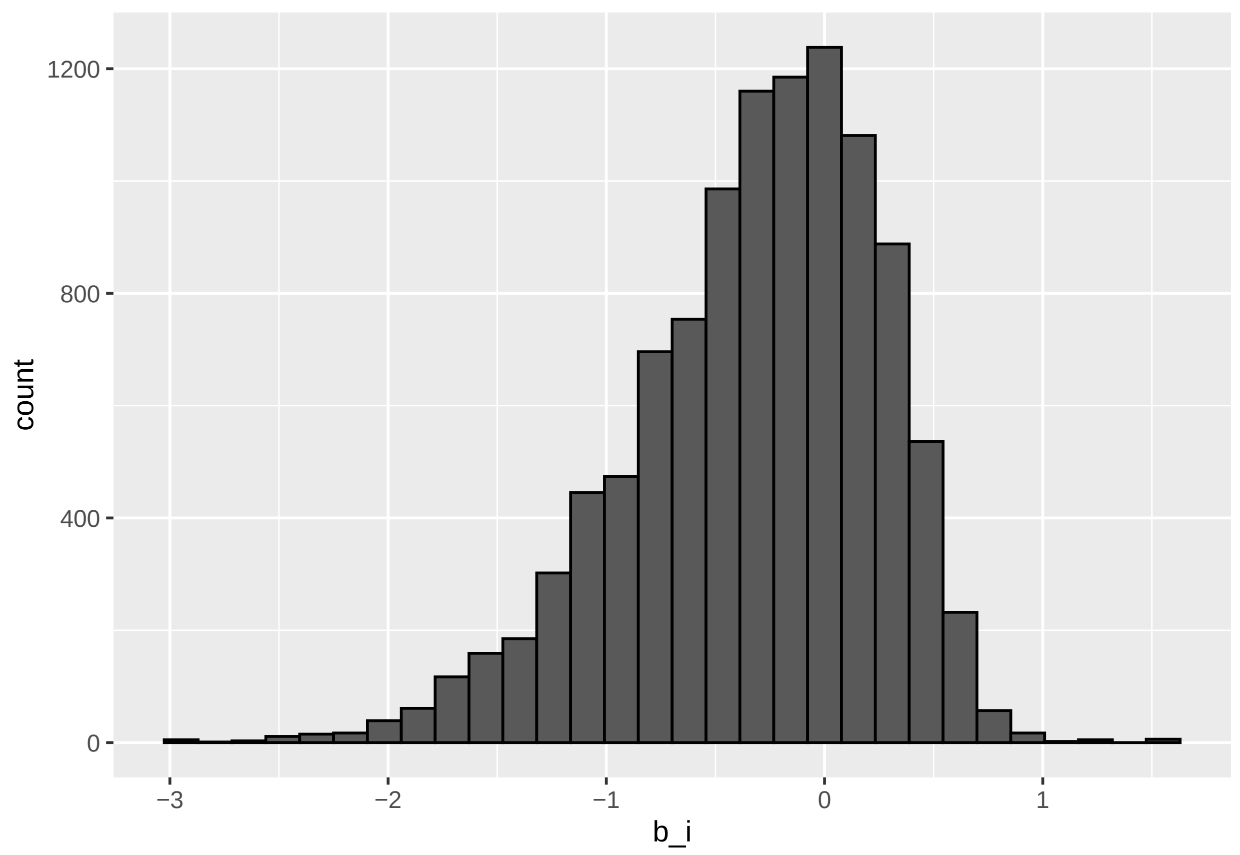
$$Y_{u,i} = \mu + b_i + \epsilon_{u,i} \tag{3}$$

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

```r
mu <- mean(edx$rating)
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

The histogram is skewed to the left, which implies that most movies have negative effects. This is known as penalty term movie effect.

```r
movie_avgs%>%
ggplot(aes(b_i)) +
geom_histogram(bins = 30, color = "black")
```



If one movie is on average rated worse than the average rating of all movies μ, then we predict that it will rated lower that μ by b_i.By including the movie effect, our model's RMSE improves to 0.9437 on validation data.

```r
predicted_ratings <- mu +  validation %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
model_1_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie effect model",
                                     RMSE = model_1_rmse ))
rmse_results
```

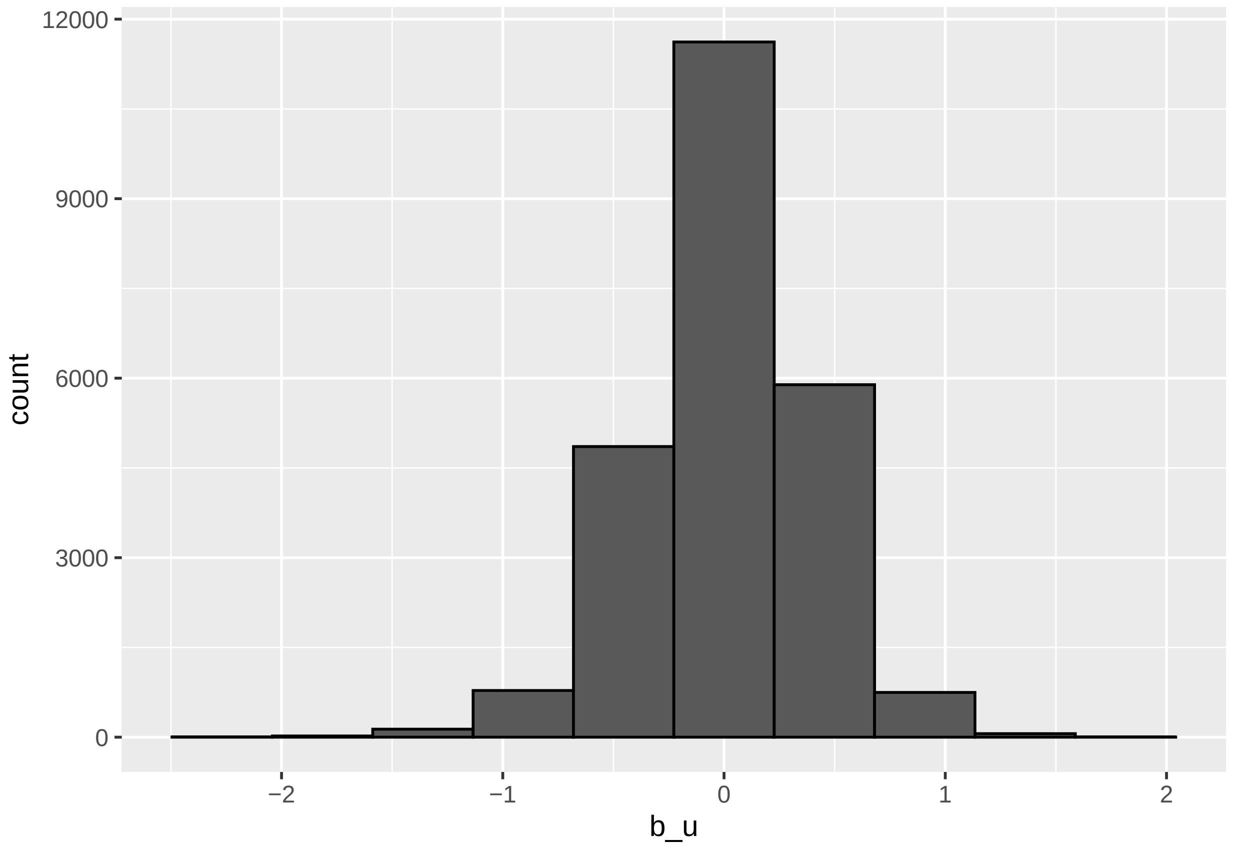
method	RMSE
Target RSME	0.8649000
Average Rating Model	1.0612018
Movie effect model	0.9439087

6 User Effects

We know that some users are more active than others.We compute the average rating for user μ, for those that have rated over 50 movies. There is also substantial variability acorss users. Some users are very cranky (like few movies), and other users love every movie. Therefore, we can further improve our previous model by adding user-specific effect (b_u):

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

```
edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  filter(n() >= 100) %>%
  summarize(b_u = mean(rating - mu - b_i)) %>%
  ggplot(aes(b_u)) +
  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. We estimate b_u as the average of

$$Y_{u,i} - \mu - b_i$$

as shown below.

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

By using new model for prediction, the RMSE improves to 0.865.

```
predicted_ratings <- validation%>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
model_2_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie and user effect model",
                                     RMSE = model_2_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 Moives and Users

6.2 sec:reg

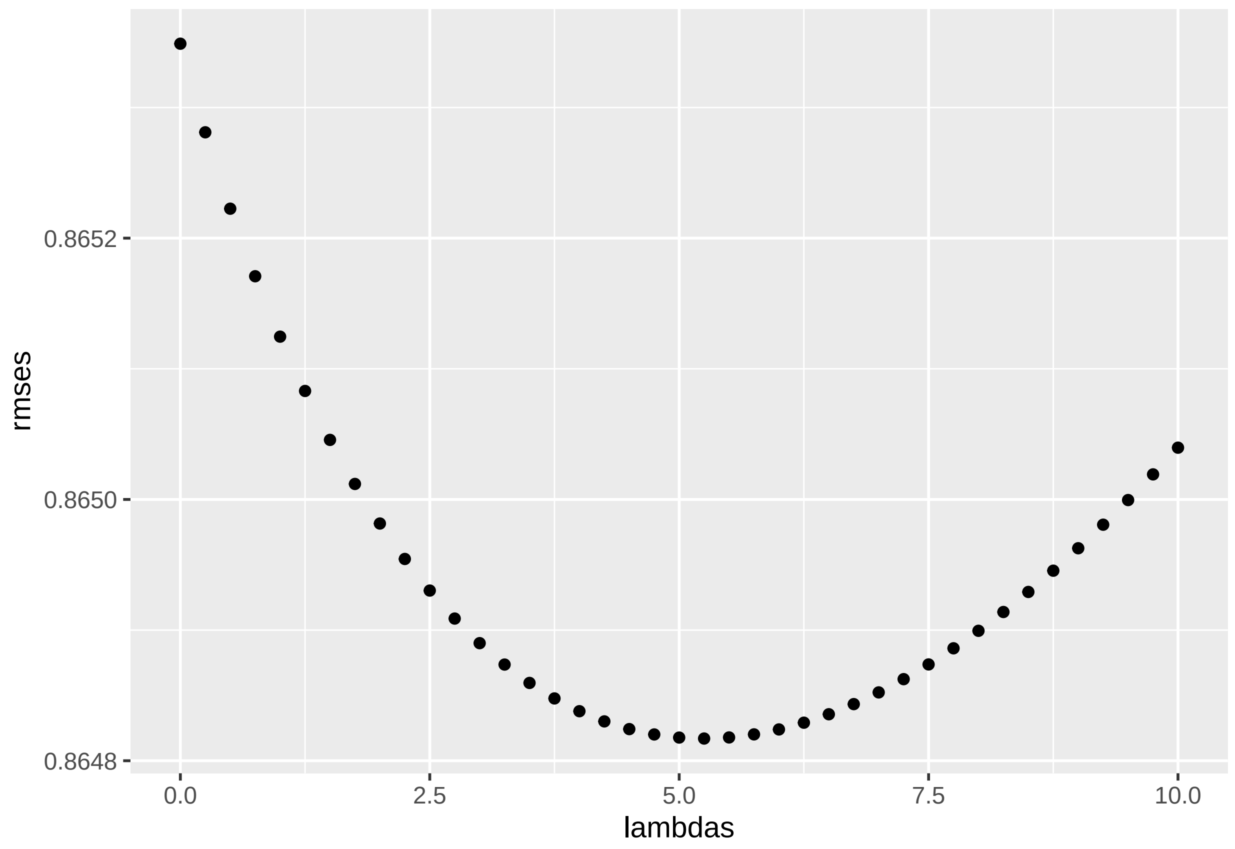
There is still room for improvement as we stil made mistakes on our first model (using only movies). There are users who have rated very few movies (less than 30 movies). On the other hand, some movies are rated very few times (say 1 or 2). These are basically noisy estimates that result in overfitting of the model. Such large errors are likely increase the RMSE and affect our predictions.

To tackle this problem, we use the concept of regularization. The regularization permits to penalize large estimates that come from small sample sizes. First, we find the value of lambda (that is a tuning parameter) that will minimize the RMSE.

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

  mu <- mean(edx$rating)

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

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

  predicted_ratings <-
    validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

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

We can see that the optimal lambda value is 5.25, which gives us a RMSE of 0.8649.

```r
qplot(lambdas, rmses)
```



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

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

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

7 Results

We can analyse the RMSEs for the various models trained in the previouse section. We started with a very basic approach of predicting the mean ratingregardless of the user. Then, we added movie and user effects to the simple model. Finally, we experimented regularization with movie and user effects. For regularization, we found the optimal value of turing parameter lambda.The resultant RMSEs are as follows:

```
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

We can see that the RMSE of our final model is 0.864817. This is a significant improvement (decrease in RMSE of around 18.5%) compared to the first simple model.

8 Conclusion

The simplest model which predicts the same rating (mean rating) for all movies regardless of user gave anRMSE above 1. If RMSE is larger than 1, it means our typical error is larger than one star, which is notgood. By taking into consideration the movie effect the RMSE went down to 0.9439087 which was a greatimprovement. RMSE further went down to 0.8653488 after modelling the user effect in the previous model.However, the lowest RMSE i.e, 0.8648201 was achieved by regularizing the movie and user effect, whichpenalized larger estimates of ratings from relatively small sample size of users.Subsequently, regularizing the year and genres effects can further improve the residual mean squared errorbut regularizing the genres is computationally very expensive since it requires separating multiple genres formany movies into multiple observations for a given movie i having single genre in genres column.

In this report we described a way to build up the recommendation algorithm to predict movie ratings using Movielens data set step by step. Four predictors were used in our algorithm included: (i) the baseline as average rating of each movie, (ii) the specific-effect by user, (iii) the specific-effect by aging time, (iv) the specific-effect by genres. Two main predictors have highest impact to the results are the average rating of each movie and the specific-effect by user. The final RMSE from our algorithm is 0.8645. This result achieved our project goals and the initial criteria of the course HarvardX - PH125.9X Data Science: Capstone project - All learners (RMSE < 0.8649).Because our algorithm is based on the average rating of each movie, therefore if a movie have very less number of rating, this algorithm is limited to provide an accuracy results. A better result is received only when the number of rating of a movie and/or the number of rating given by a user is increased high enough.However this is an opportunity to improve our algorithm in the future by including the number of rating of a movie and the number of rating given by a user to our algorithm. Furthermore, other machine learning techniques such as Regularization, Penalized Least Squares, Matrix Factorization could also improve the results further.Another limitation during this project time is the hardware, especially RAM as the main constraint, and the speed of normal laptop.

```
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

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