MovieLens Project

Kunjal

12/22/2021

Contents

1 Background and Motivation overview

A recommendation system is a subclass of information filtering system that seeks to predict the "rating" or "preference" a user would give to an item. In this project the items are movies. Recommendation systems are one of the most used models in machine learning algorithms.

Many companies are using these recommendation systems in a variety of areas including movies, music, news, books, research articles, search queries, social tags, and products in general. A strong recommendation system was of such importance that in 2006, Netflix offered a million dollar prize to anyone who could improve the effectiveness of its recommendation system by 10%.

1.1 Introduction

This project is related to the MovieLens Project of the HarvardX: PH125.9x Data Science: Capstone course. For this project we will focus on create a movie recommendation system using the 10M version of MovieLens dataset, collected by GroupLens Research.

The present report start with a general idea of the project. Then the given dataset will be prepared and setup. An exploratory data analysis is carried out in order to develop a machine learning algorithm that could predict movie ratings until a final model. Results will be explained. Finally the report ends with some concluding remarks.

1.2 Aim of the project

The aim of this project is to train a machine learning algorithm using the inputs of a provided training subset to predict movie ratings in a 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.

1.3 Data Ingestion

For this project a movie rating predictor is created using the 'MovieLens' dataset. This data set can be found and downloaded here:

- [MovieLens 10M dataset] https://grouplens.org/datasets/movielens/10m/
- [MovieLens 10M dataset - zip file] http://files.grouplens.org/datasets/movielens/ml-10m.zip

The below chunk of code gives a partition of the data set for training and testing our data. It also removes the unnecessary files from the working directory.

```
############################################################
# 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  3.1.5     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.0.2     v forcats 0.5.1
```

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

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

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

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

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

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

```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
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 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)`
```

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

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

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

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

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

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

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

Algorithm development is to be carried out on the "edx" subset only, as "validation" subset will be used to test the final algorithm.

We are going to use additional libraries:

```
library(ggplot2)
library(lubridate)
```

2 Methods and Analysis

2.1 Data Pre Processing

We modify the columns to suitable formats that can be further used for analysis.

```
# Modify the year as a column in the both datasets
edx <- edx %>% mutate(year = as.numeric(str_sub(title,-5,-2)))
validation <- validation %>% mutate(year = as.numeric(str_sub(title,-5,-2)))
```

2.2 Exploratory Analysis

To get familiar with the dataset, we find that there are six variables "userId","movieID","rating","timestamp", "title","genres" in the subset. Each row represent a single rating of a user for a single movie.

```
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                            genres year
## 1:                Comedy|Romance 1992
## 2:          Action|Crime|Thriller 1995
## 3:  Action|Drama|Sci-Fi|Thriller 1995
## 4:       Action|Adventure|Sci-Fi 1994
## 5: Action|Adventure|Drama|Sci-Fi 1994
## 6:       Children|Comedy|Fantasy 1994
```

A Summary of the subset confirms that there are no missing values.

```
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              year
##  Length:9000055     Length:9000055     Min.   :1915
##  Class :character   Class :character   1st Qu.:1987
##  Mode  :character   Mode  :character   Median :1994
##                                        Mean   :1990
##                                        3rd Qu.:1998
##                                        Max.   :2008
```

The total of unique movies and users in the edx subset is given in the below chunk of code.

```
# Number of unique movies and users in the edx dataset
edx %>% summarize(n_users = n_distinct(userId), n_movies = n_distinct(movieId))
```

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

The popularity of the movie genre depends strongly on the contemporary issues. The below code shows number of movie ratings for certain genres.

```
  #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
```

A summary statistics of rating in edx subset.The 4 is the most common rating, followed by 3 and 0.5 is the least common rating.

```
summary(edx$rating)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.500   3.000   4.000   3.512   4.000   5.000
```

```
edx %>% group_by(title)%>%summarise(number = n())%>%arrange(desc(number))
```

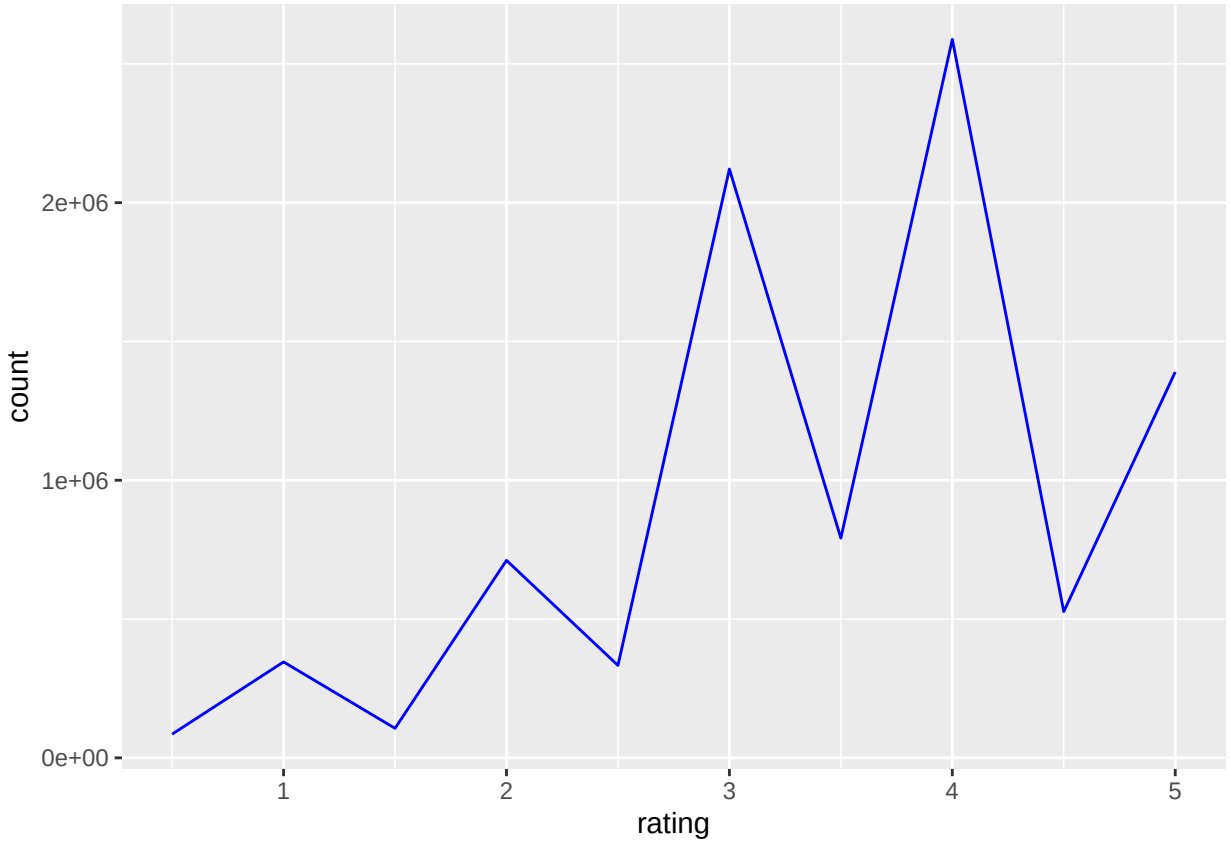
```
## # A tibble: 10,676 x 2
##    title                                                     number
##    <chr>                                                      <int>
##  1 Pulp Fiction (1994)                                        31362
##  2 Forrest Gump (1994)                                        31079
##  3 Silence of the Lambs, The (1991)                           30382
##  4 Jurassic Park (1993)                                       29360
##  5 Shawshank Redemption, The (1994)                           28015
##  6 Braveheart (1995)                                          26212
##  7 Fugitive, The (1993)                                       25998
##  8 Terminator 2: Judgment Day (1991)                          25984
##  9 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)  25672
## 10 Apollo 13 (1995)                                           24284
## # ... with 10,666 more rows
```

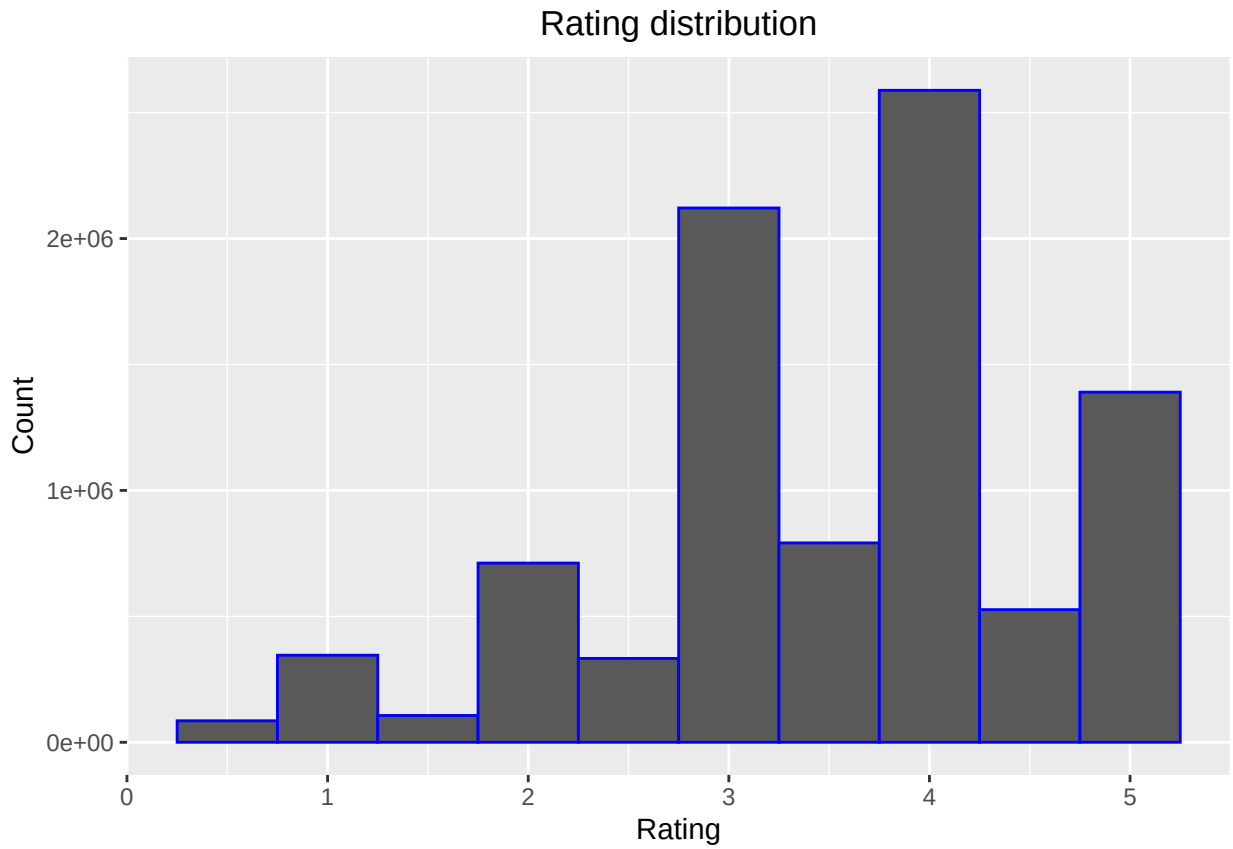
```
#The five most given ratings in order from most to least
head(sort(-table(edx$rating)),5)
```

```
##
##        4        3        5      3.5        2
## -2588430 -2121240 -1390114  -791624  -711422
```

Users have a preference to rate movies rather higher than lower as shown by the distribution of ratings below. 4 is the most common rating, followed by 3 and 5. 0.5 is the least common rating. In general, half rating are less common than whole star ratings.

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


count
2e+06
1e+06
0e+00
1
2
3
4
5
rating



From the above plot, half star ratings are less common than whole star ratings. The average rating for each year is shown in below.

```
#Average ratings of edx dataset
avg_ratings <- edx %>% group_by(year) %>% summarise(avg_rating = mean(rating))
avg_ratings
```

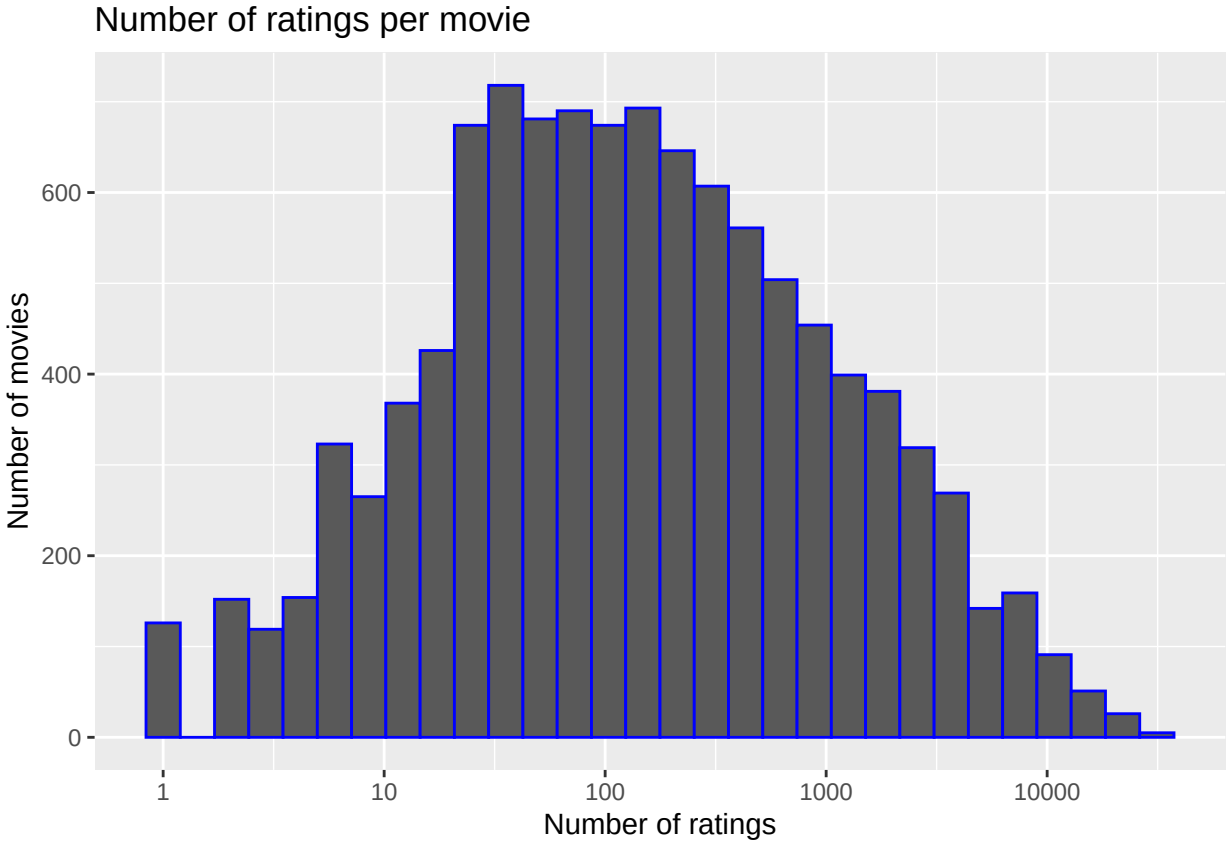
```
## # A tibble: 94 x 2
##     year avg_rating
##    <dbl>      <dbl>
##  1  1915       3.29
##  2  1916       3.83
##  3  1917       3.73
##  4  1918       3.65
##  5  1919       3.28
##  6  1920       3.94
##  7  1921       3.83
##  8  1922       3.9
##  9  1923       3.78
## 10  1924       3.94
## # ... with 84 more rows
```

2.3 Data Analysis Strategies

We can observe that some movies have been rated more often than other, while some have very few ratings and sometimes only one rating. This will be important for our model as very low rating numbers might results in untrustworthy estimate for our predictions. In fact 125 movies have been rated only once.

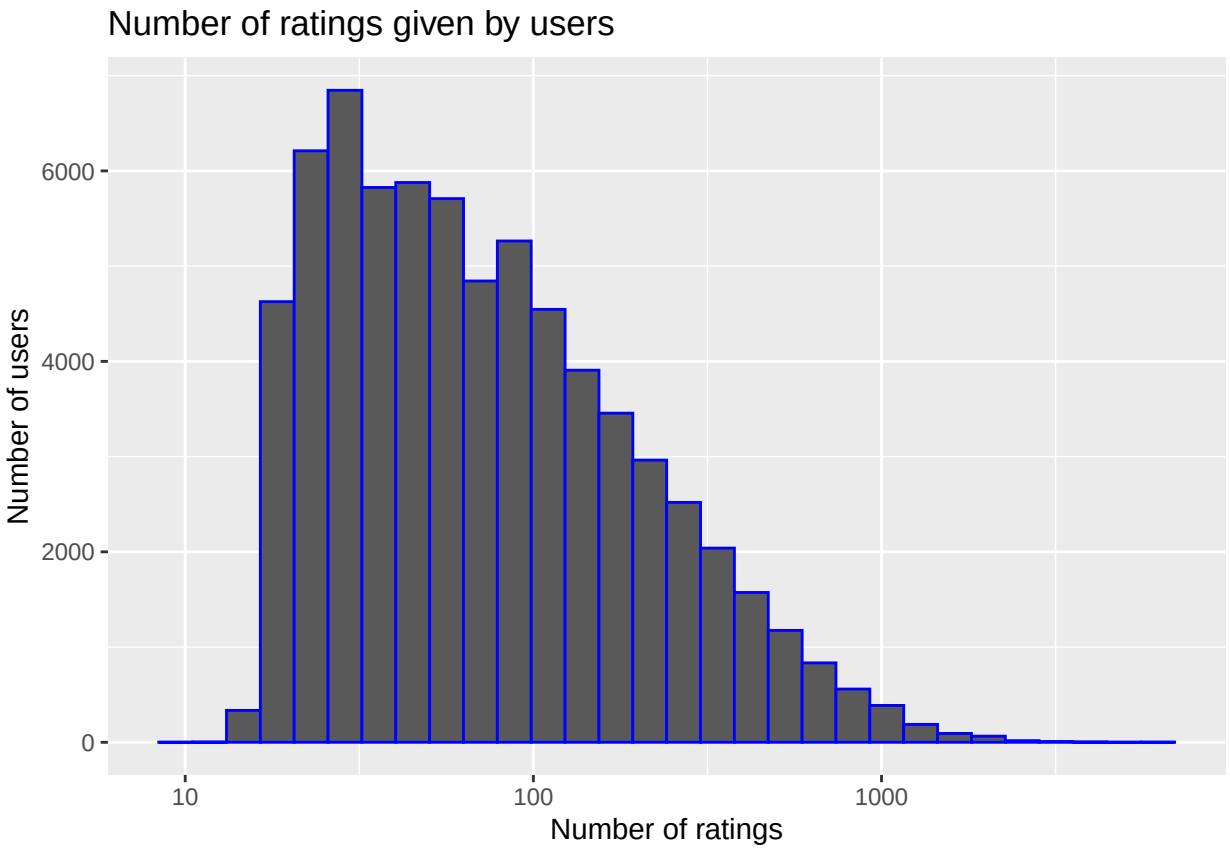
Thus regularization and a penalty term will be applied to the models in this project. Regularization are techniques used to reduce the error by fitting a function appropriately on the given training set and avoid over fitting (the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably). Regularization is a technique used for tuning the function by adding an additional penalty term in the error function. The additional term controls the excessively fluctuating function such that the coefficients don't take extreme values.

This is called movie bias. The distribution of movie bias effect (b_i) is given below.



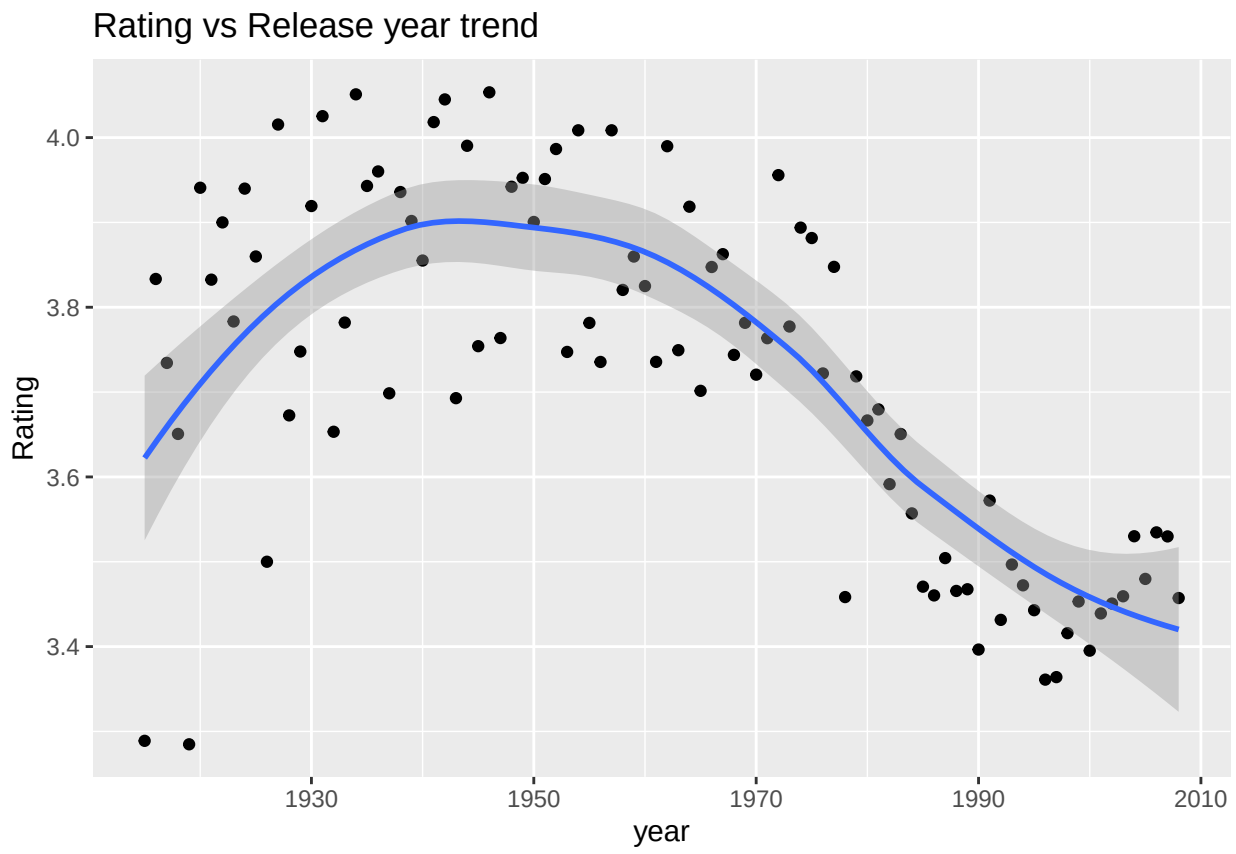
We can observe that the majority of users have rated between 30 and 100 movies. So, a user penalty term need to be included later in our models.

Some users are positive and some have negative reviews because of their own personal liking/disliking regardless of movie.The distribution of user bias effect (b_u) is given below.



Estimating the trend of rating versus release year-Year Effect.The general trend shows modern users relatively rate movies lower. The users mindset also evolve over time.This can also effect the average rating of movies over the years.The plot of year bias effect(b_y)is given below.

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

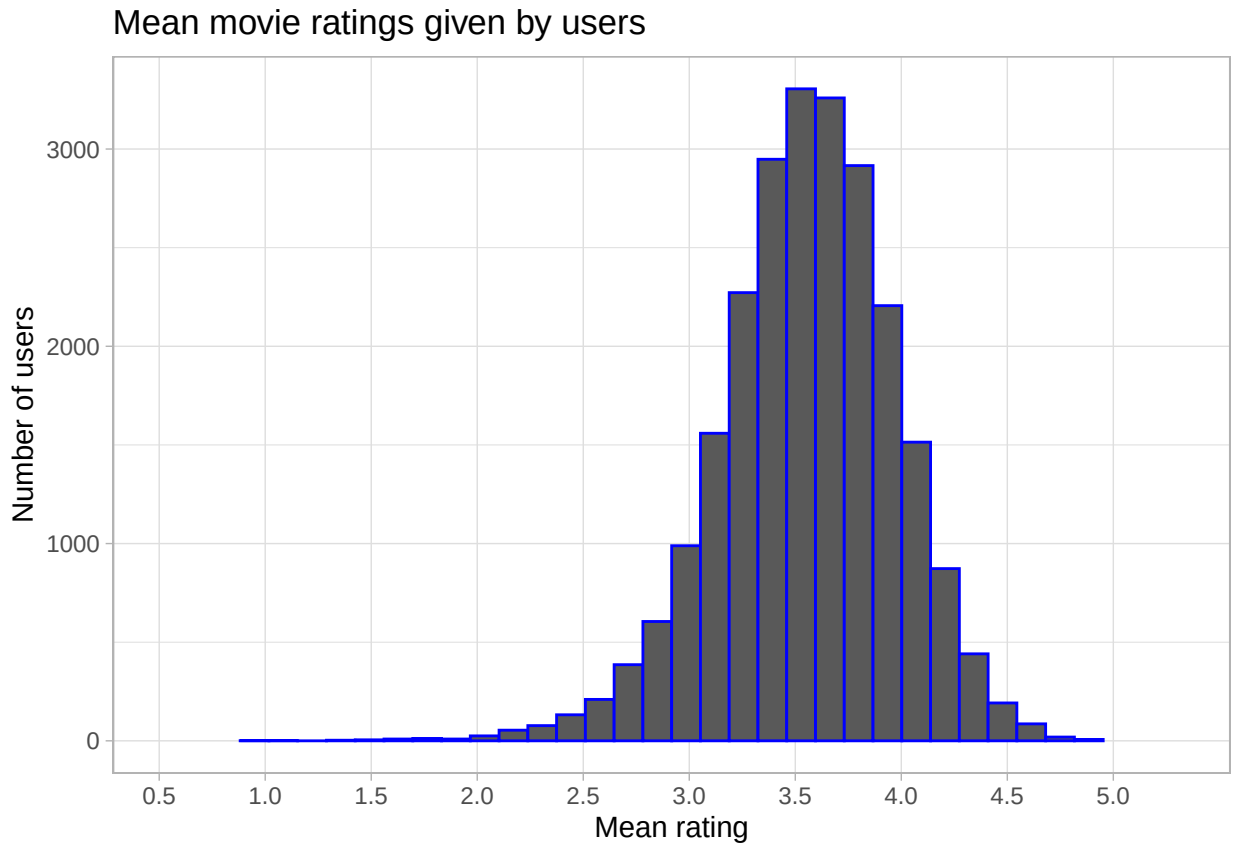


As 20 movies that were rated only once appear to be obscure, predictions of future ratings for them will be difficult.

title	rating	n_rating
1, 2, 3, Sun (Un, deuz, trois, soleil) (1993)	2.0	1
100 Feet (2008)	2.0	1
4 (2005)	2.5	1
Accused (Anklaget) (2005)	0.5	1
Ace of Hearts (2008)	2.0	1
Ace of Hearts, The (1921)	3.5	1
Adios, Sabata (Indio Black, sai che ti dico: Sei un gran figlio di. . .) (1971)	1.5	1
Africa addio (1966)	3.0	1
Aleksandra (2007)	3.0	1
Bad Blood (Mauvais sang) (1986)	4.5	1
Battle of Russia, The (Why We Fight, 5) (1943)	3.5	1
Bellissima (1951)	4.0	1
Big Fella (1937)	3.0	1
Black Tights (1-2-3-4 ou Les Collants noirs) (1960)	3.0	1
Blind Shaft (Mang jing) (2003)	2.5	1
Blue Light, The (Das Blaue Licht) (1932)	5.0	1
Borderline (1950)	3.0	1
Brothers of the Head (2005)	2.5	1
Chapayev (1934)	1.5	1
Cold Sweat (De la part des copains) (1970)	2.5	1

Furthermore, users differ vastly in how critical they are with their ratings. Some users tend to give much lower star ratings and some users tend to give higher star ratings than average. The visualization below includes only users that have rated at least 100 movies.

```
## Warning: Continuous limits supplied to discrete scale.
## Did you mean `limits = factor(...)` or `scale_*_continuous()`?
```



2.4 Modelling Approach

The value used to evaluate algorithm performance is the Root Mean Square Error(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, lower the RMSE is better than higher one. The effect of each error on RMSE is proportional to the size of the squared error; thus larger errors will have large effect on RMSE. RMSE is sensitive to outliers. The evaluation criteria for this algorithm is a RMSE expected to be lower than 0.8775.

```
#Root Mean Square Error Loss Function
#Function that computes the RMSE for vectors of ratings and their corresponding predictors
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings-predicted_ratings)^2,na.rm=T))
}
```

2.4.1 I. Average movie rating model

The first basic model predicts the same rating for all movies, so we compute the dataset's mean rating. The expected rating of the underlying data set is between 3 and 4.

We start by building the simplest possible recommender system by predicting the same rating for all movies regardless of user who give it. A model based approach assumes the same rating for all movie with all

differences explained by random variation :

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

with $\epsilon_{u,i}$ independent error sample from the same distribution centered at 0 and μ the "true" rating for all movies. This very simple model makes the assumption that all differences in movie ratings are explained by random variation alone. We know that the estimate that minimize the RMSE is the least square estimate of $Y_{u,i}$, in this case, is the average of all ratings:

The expected rating of the underlying data set is between 3 and 4.

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

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

If we predict all unknown ratings with μ or mu, we obtain the first naive RMSE:

```
naive_rmse <- RMSE(validation$rating, mu)
naive_rmse
```

```
## [1] 1.061202
```

Here, we represent results table with the first RMSE:

```
rmse_results <- data_frame(method = "Average movie rating model", RMSE = naive_rmse)
```

```
## Warning: `data_frame()` was deprecated in tibble 1.1.0.
## Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
```

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

method	RMSE
Average movie rating model	1.061202

This give us our baseline RMSE to compare with next modelling approaches.

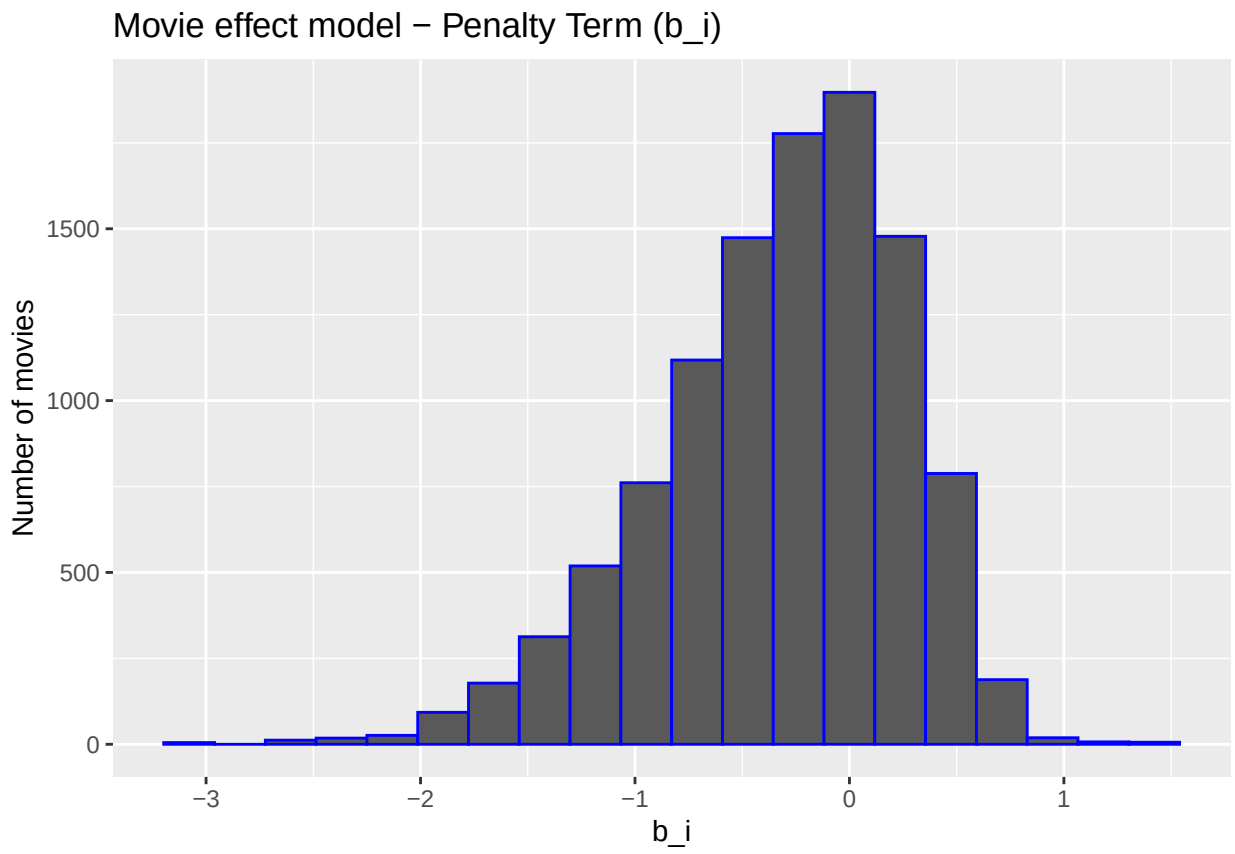
In order to do better than simply predicting the average rating, we incorporate some of insights we gained during the exploratory data analysis.

2.4.2 II. Movie effect model

To improve above model we focus on the fact that, from experience, we know that some movies are just generally rated higher than others. Higher ratings are mostly linked to popular movies among users and the opposite is true for unpopular movies. We compute the estimated deviation of each movies' mean rating from the total mean of all movies μ. The resulting variable is called "b" (as bias) for each movie "i" b_i, that represents average ranking for movie i:

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

The histogram is left skewed, implying that more movies have negative effects



This is called the penalty term movie effect.

Our prediction improve once we predict using this model.

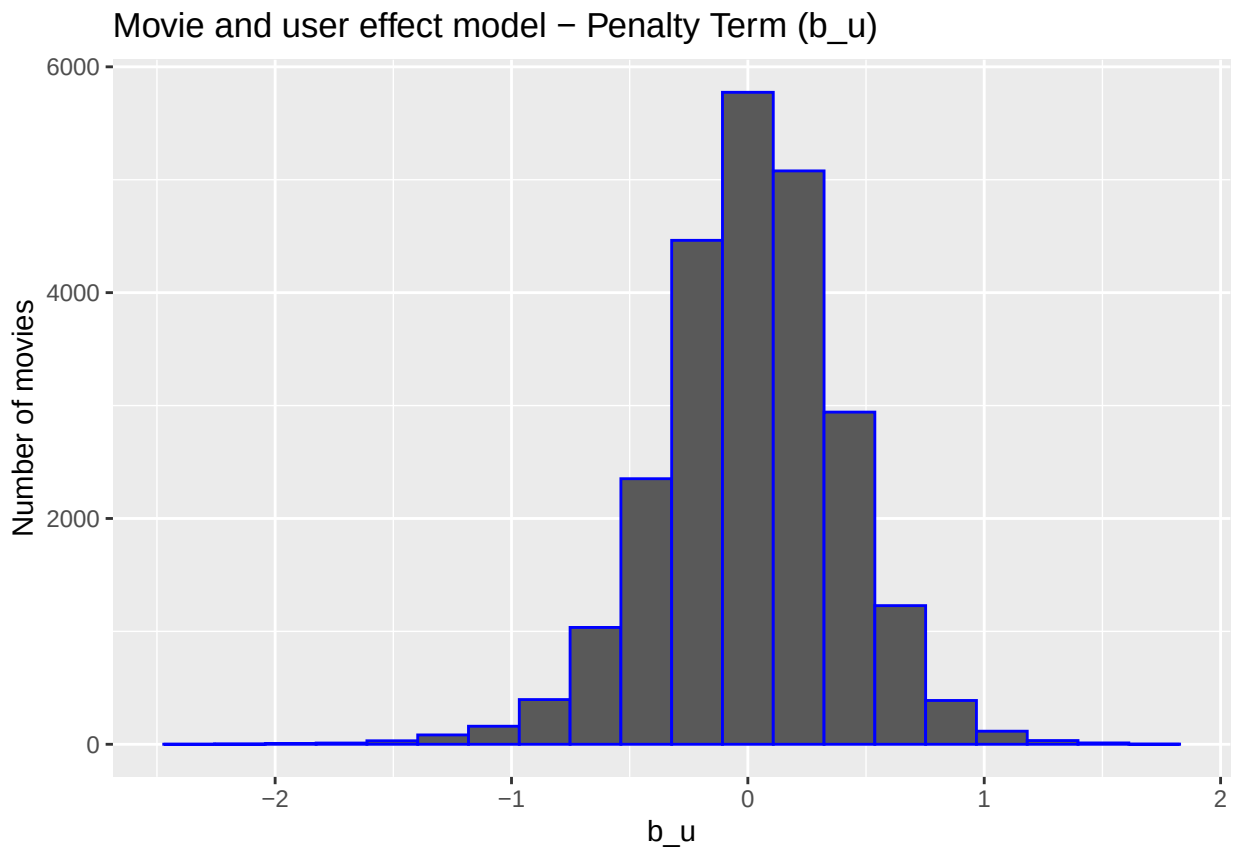
method	RMSE
Average movie rating model	1.0612018
Movie effect model - Penalty Term (b_i)	0.9439087

So we have predicted movie rating based on the fact that movies are rated differently by adding the computed b_i to μ. If an individual movie is on average rated worse that the average rating of all movies μ , we predict that it will rated lower that μ by b_i, the difference of the individual movie average from the total average.

We can see an improvement but this model does not consider the individual user rating effect.

2.4.3 III. Movie and user effect model

We compute the average rating for user μ, for those that have rated over 100 movies, said penalty term user effect. In fact users affect the ratings positively or negatively.



There is substantial variability across users as well: some users are very cranky and other love every movie. This implies that further improvement to our model my be:

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

where b_u is a user-specific effect. 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 compute an approximation by computing μ and b_i, and estimating b_u, as the average of

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

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

We can now construct predictors and see RMSE improves:

```
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 - Penalty Term (b_u)",
                                     RMSE = model_2_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Average movie rating model	1.0612018
Movie effect model - Penalty Term (b_i)	0.9439087
Movie and user effect model - Penalty Term (b_u)	0.8653488

Our rating predictions further reduced the RMSE. But we made stil mistakes on our first model (using only movies). The supposes "best " and "worst "movie were rated by few users, in most cases just one user. These movies were mostly obscure ones. This is because with a few users, we have more uncertainty. Therefore larger estimates of b_i, negative or positive, are more likely.

Large errors can increase our RMSE.

2.4.4 IV Movie, user and year effect model

Our rating predictions further reduced the RMSE. Year of the movie also effect ratings.

```
predicted_ratings <- validation%>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(year_avgs, by='year') %>%
  mutate(pred = mu + b_i + b_u + b_y) %>%
  pull(pred)
model_3_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie, User and Year effect model - Penalty Term (b_y)",
                                     RMSE = model_3_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Average movie rating model	1.0612018
Movie effect model - Penalty Term (b_i)	0.9439087
Movie and user effect model - Penalty Term (b_u)	0.8653488
Movie, User and Year effect model - Penalty Term (b_y)	0.8650043

2.4.5 V. Regularized movie, user and year effect model

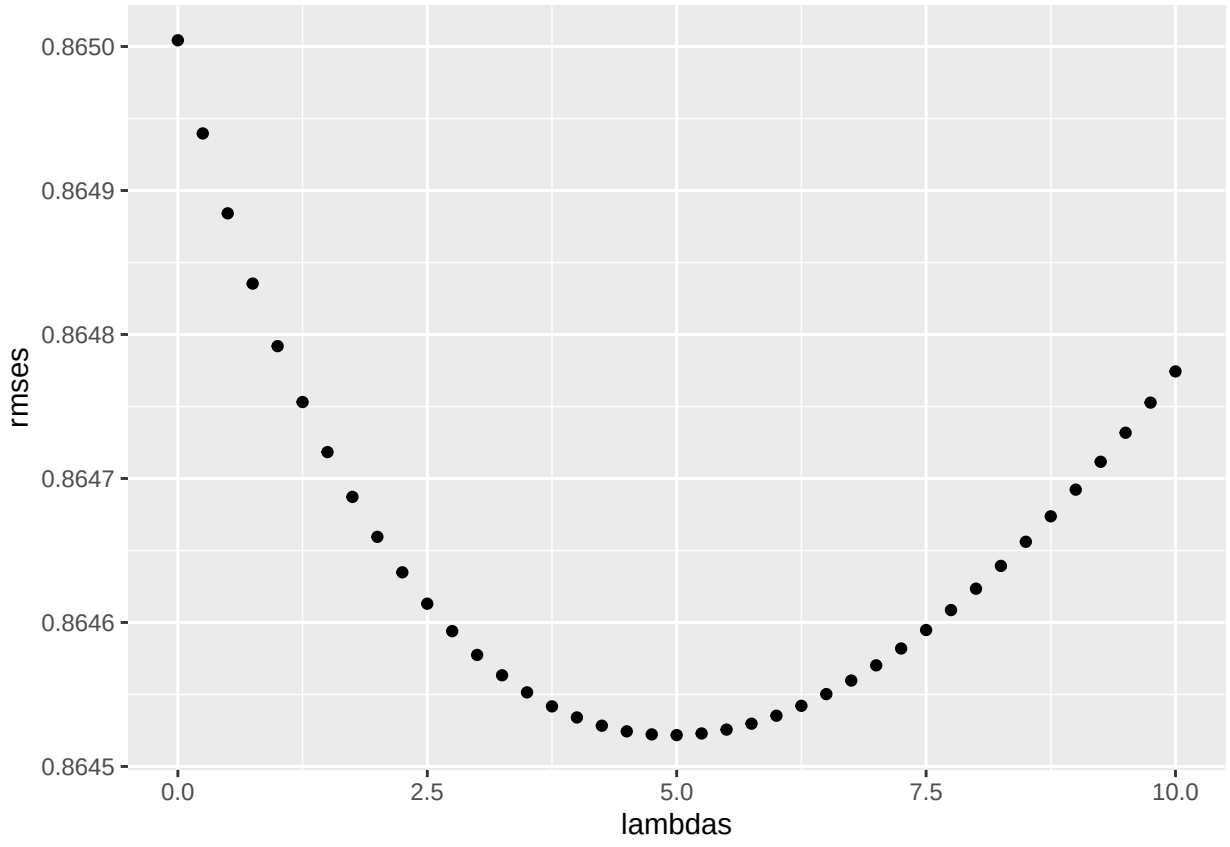
So estimates of b_i, b_u and b_y are caused by movies with very few ratings and in some users that only rated a very small number of movies, and production year Hence this can strongly influence the prediction. The use of the regularization permits to penalize these aspects. We should find the value of lambda (that is a tuning parameter) that will minimize the RMSE. This shrinks the b_i, b_u and b_y in case of small number of ratings.

```
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  - mu - b_i)/(n()+l))

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

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

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

We plot RMSE vs lambdas to select the optimal lambda

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



For the full model, the optimal lambda is:

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

```
## [1] 5
```

For the full model, the optimal lambda is: 5. A lambda of 5 was shown to be the best Tune for the train and test sets of this Regularized Model as well and therefore will be used on the validation set.

The new results will be:

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized movie, user and year effect model",
                                     RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

method	RMSE
Average movie rating model	1.0612018
Movie effect model - Penalty Term (b_i)	0.9439087
Movie and user effect model - Penalty Term (b_u)	0.8653488
Movie, User and Year effect model - Penalty Term (b_y)	0.8650043
Regularized movie, user and year effect model	0.8645218

3 Results

The RMSE values of all the represented models are the following:

method	RMSE
Average movie rating model	1.0612018
Movie effect model - Penalty Term (b_i)	0.9439087
Movie and user effect model - Penalty Term (b_u)	0.8653488
Movie, User and Year effect model - Penalty Term (b_y)	0.8650043
Regularized movie, user and year effect model	0.8645218

We therefore found the lowest value of RMSE that is 0.8645218.

The lowest RMSE using the validation set is the Final Validation Model featuring Regularized Movie, user and year Effects. It significantly improves upon the Benchmarking Model's RMSE of 1.06.

4 Discussion

This model work well if the average user doesn't rate a particularly good/popular movie with a large positive b_i, by disliking a particular movie. and if the user don't rate the movie by thier production year.

5 Conclusion

We can affirm to have built a machine learning algorithm to predict movie ratings with MovieLens dataset. The regularized model including the effect of user is characterized by the lower RMSE value and is hence the optimal model to use for the present project.

The optimal model characterised by the lowest RMSE value (0.8645218) lower than the initial evaluation criteria (0.8775) given by the goal of the present project.

6 Appendix - Enviroment

```
print("Operating System:")
```

```
## [1] "Operating System:"
```

```
version
```

```
##                _
## platform       x86_64-w64-mingw32
## arch           x86_64
## os             mingw32
## system         x86_64, mingw32
## status
## major          4
## minor          1.1
## year           2021
## month          08
## day            10
## svn rev        80725
## language       R
## version.string R version 4.1.1 (2021-08-10)
## nickname       Kick Things
```