MovieLens Project

Harini

26/11/2020

Introduction

The Project is related to the MovieLens Project of the HarvardX:PH125:9x Data science Capstone. Recommendation systems use ratings that users have given items to make specific recommendations. Items for which a high rating is predicted for a given user are then recommended to that user. The aim of the project is to develop a machine learning algorithm using the inputs in one subset to predict movie ratings in the validation set. Several machine learning algorithm has been used and final model is presented which shows maximum possible accuracy in prediction. This report contains problem definition, data ingestion, exploratory analysis, modeling and data analysis, results and concluding remarks. The project uses Penalized least squares approach motivated from Netfix challenges.The movie, user, year, genre are some of the features which has larger effect on errors. We will try to shrink these effects by using the proposed method to improve the accuracy.

Problem Defnition

Movie recommendation system predicts the movie rating by a user based on users past rating of movies. There are different type of biases present in the movie reviews. It can be different social, psychological, demographic variations that changes the taste of every single users for a given particular movie. However the problem can be solved by expressing major biases in mathematical equations.

Data Ingestion

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 1.3.0 --

## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
## -- Conflicts ------------------------------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

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

```
## Loading required package: caret

## Warning: package 'caret' was built under R version 4.0.3

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

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

## Warning: package 'data.table' was built under R version 4.0.3

##
## 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
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"))
```

```r
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")
```

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

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

We are going to use additional libraries:

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

Data Pre Processing

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

```r
# 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)))
```

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))
}
```

Exploratory Analysis

There are six variable "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
```

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.

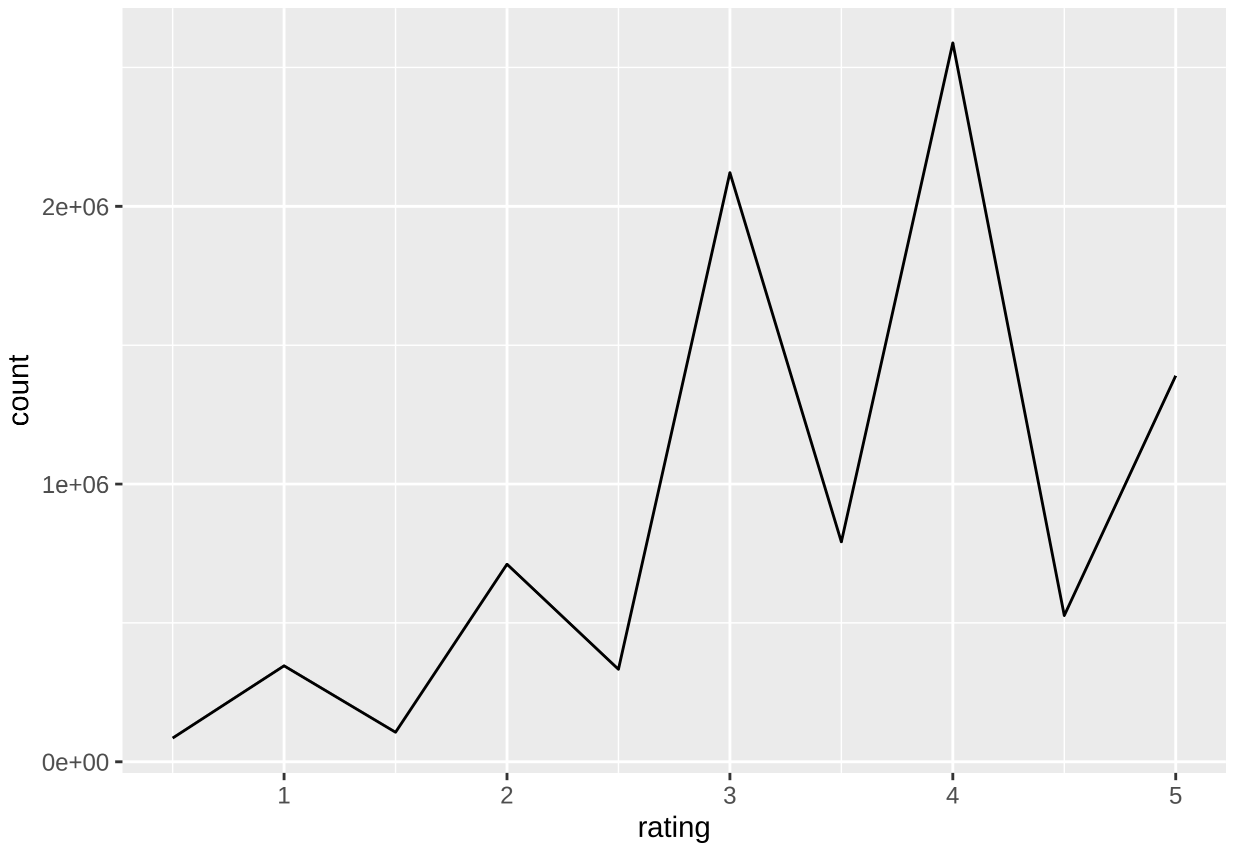
```
summary(edx$rating)
```

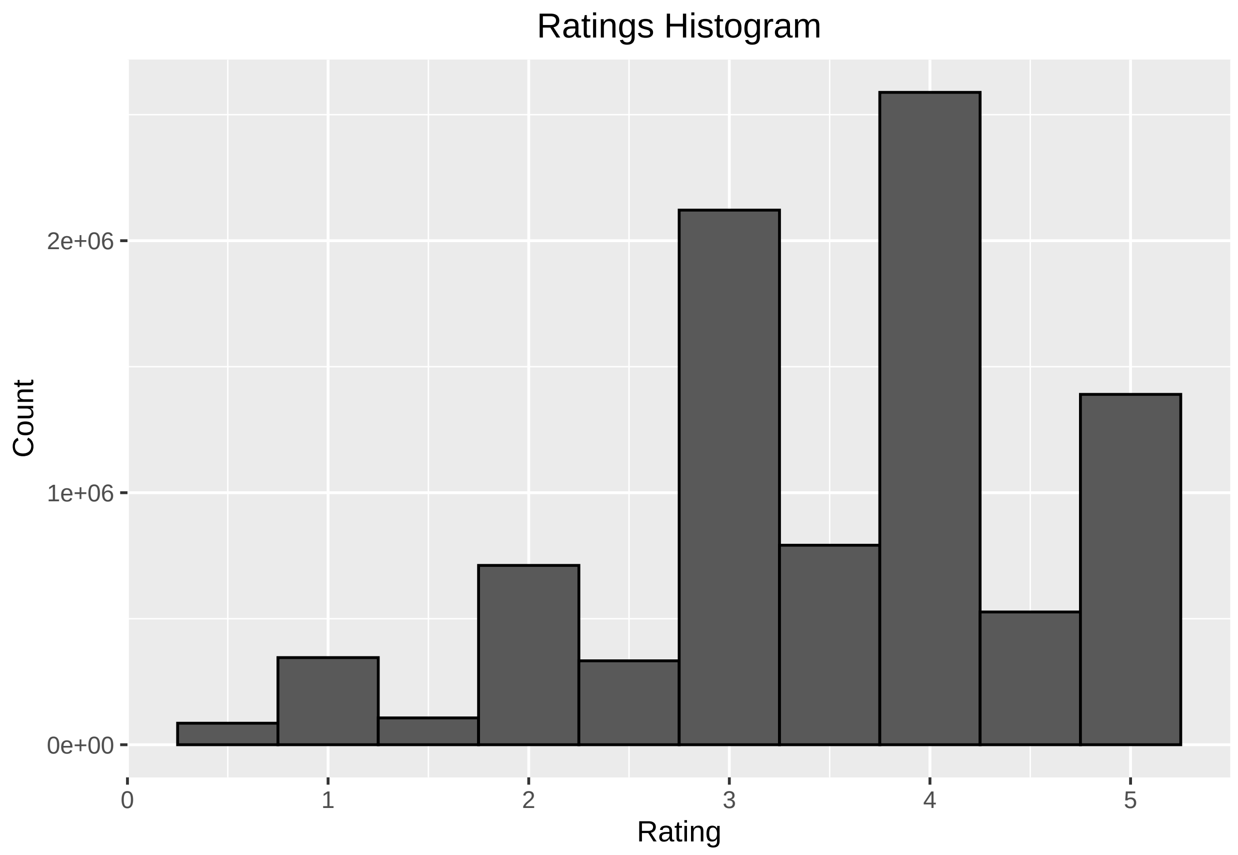
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.500   3.000   4.000   3.512   4.000   5.000
```

```
#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
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```





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))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
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
```

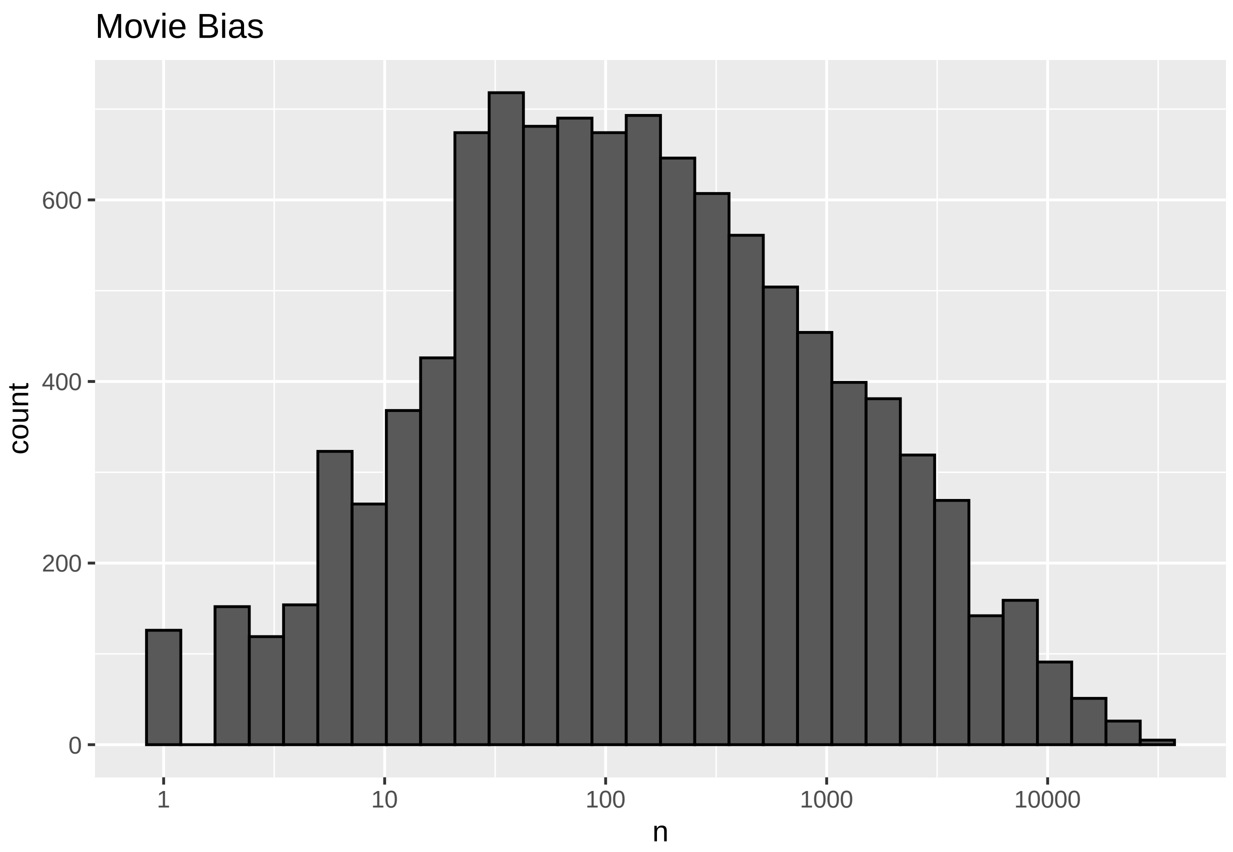
The popularity of the movie genre depends strongly on the contemporary issues. The below code shows number of movie ratings for certain genres.

```
  #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))
})
```

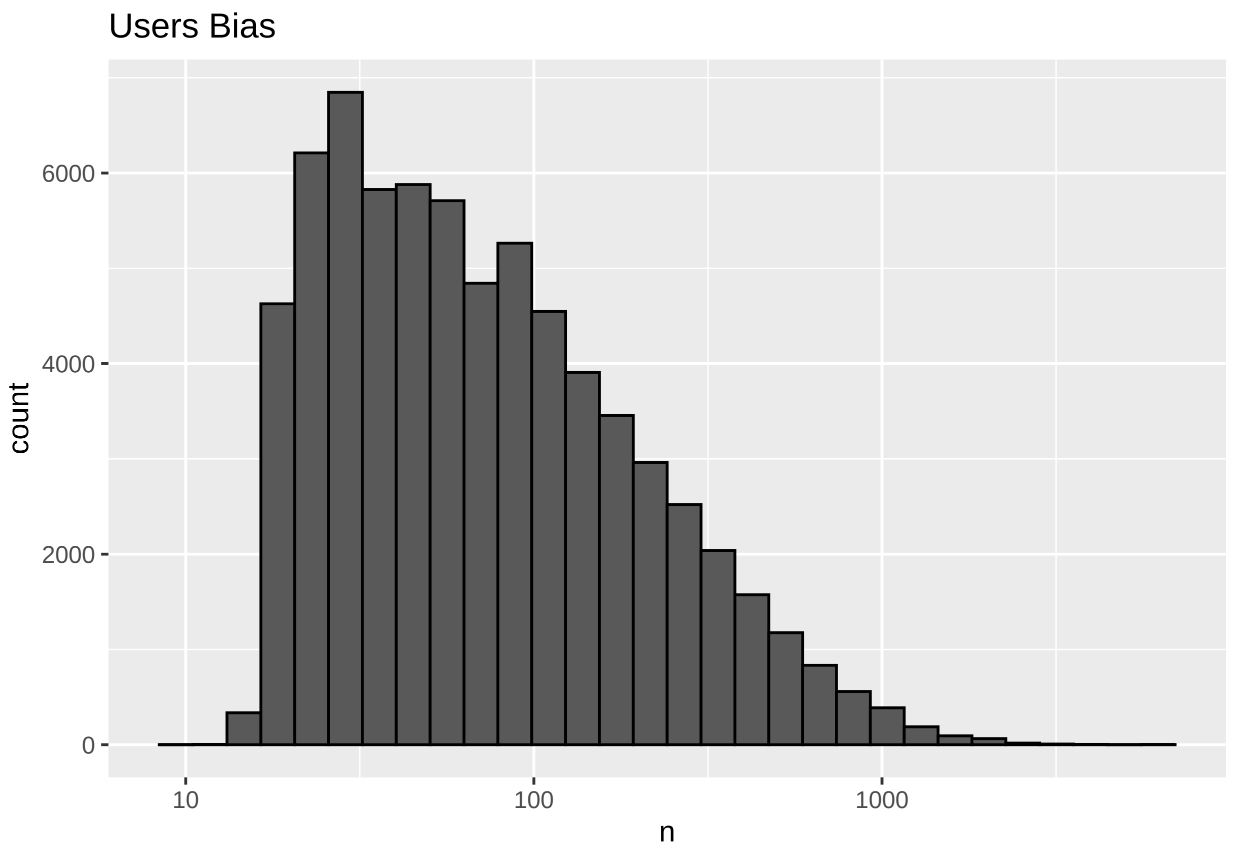
```
##    Drama   Comedy Thriller  Romance
##  3910127  3540930  2325899  1712100
```

Data Analysis Strategies

Some movies are rated more often than others (e.g. blockbusters are rated higher). This is called movie bias. The distribution of movie bias effect (b_i) is given below.



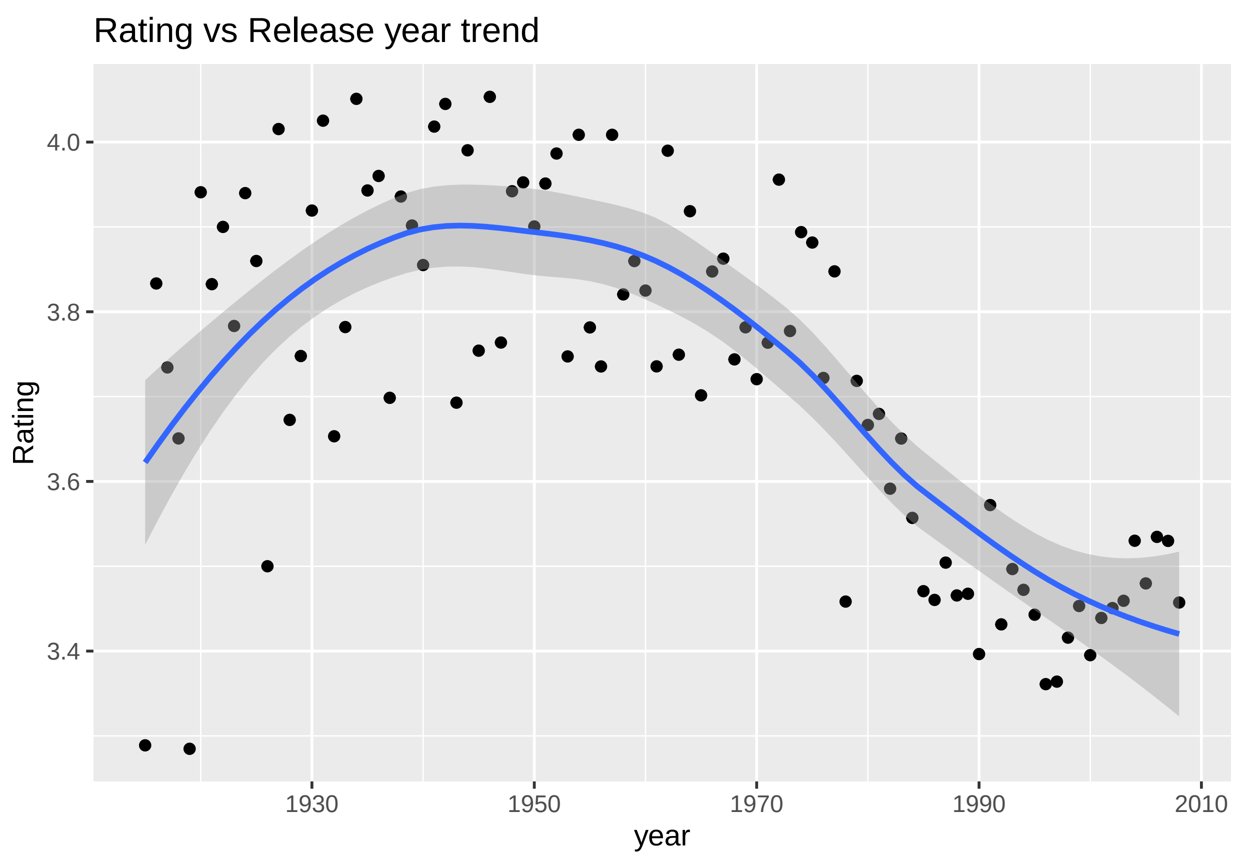
Some users are positive and some have negative reviews because of their own personal liking/disliking regardless of movie.The distribution of user bias effect (b_u) is given below.



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.The general trend shows modern users relatively rate movies lower.

```
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
```

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



From the ratings distribution which we have seen in the before module, we can observe that some movies are rated only once. This will be important for our model as very low rating numbers might results in untrustworthy estimate for our predictions.From the below table, 20 movies that were rated only once appear to be obscure, prediction of future ratings for them will be difficult.

```
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
```

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

title	rating	n_rating
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

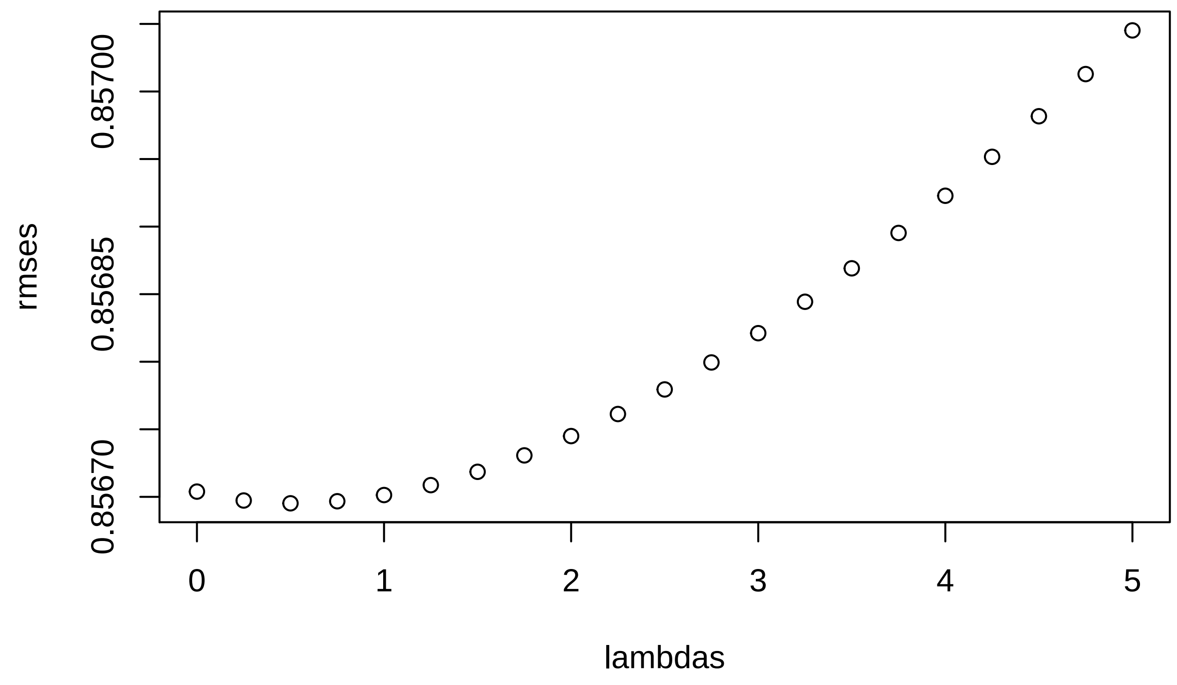
Model Creation

Regularization is a technique used to reduce the error by fitting a function appropriately on the training and avoid overfitting. The production of an analysis that corresponds too closely or exactly to a particular data set and therefore it may fail to fit additional data or predict future observations reliably is called overfitting. Regularization is a technique used for tuning the function by adding penalty term in error function. The additional term controls the excessively fluctuating function such that the coefficients will not take extreme values. The technique permits us to penalize large estimates that are formed using small sample sizes. It will constrain the total variability of the effect sizes. The challenge was to get the highest accuracy, which is measured as the number of exact matches of predicted ratings vs ratings of the validation set. After attempting different machine learning models,the most promising algorithm was the penalized least squares approach. And hence,this project uses Penalized least squares approach. 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. Lambda is a tuning parameter that will minimize the RMSE.By Using cross-validation we can find optimal value of lambda.

```r
#Model Creation
#Final Model
#Regularization using movies, users, years
##Penalized least squares Approach [minimize an equation that adds a penalty]
# lambda is a tuning parameter. Using cross-validation to find optimal value of lambda
## Please note that the below code could take some time
# b_i,b_u,b_y represent the movie,user,year effects respectively
lambdas <- seq(0, 5, 0.25)

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

  #Calculate the mean of ratings from the edx training set
  mu <- mean(edx$rating)

  #Adjust mean by movie effect and penalize low number on ratings
  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  #Adjust mean by user and movie effect and penalize low number of ratings
  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  #Adjust mean by user, movie,year effect and penalize low number of ratings
  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()+1), n_y = n())

  #Predict ratings in the training set to find optimal penalty value 'lambda'
  predicted_ratings <- edx %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred

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

```
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
```

```r
plot(lambdas, rmses) #Plot lamdas vs rmses
```



```
lambda <- lambdas[which.min(rmses)]  #To find optimal lambda
lambda
```

```
## [1] 0.5
```

Now applying the lambda value to the validation set, we can generate the predictions for the validation

```
#Apply lambda on Validation set
#Derive the mean from the training set
mu <- mean(edx$rating)
#Calculate movie effect with optimal lambda
movie_effect_reg <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
#Calculate user effect with optimal lambda
user_effect_reg <- edx %>%
  left_join(movie_effect_reg, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_i)/(n()+lambda), n_u = n())
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
#Calculate year effect with optimal lambda
year_reg_avgs <- edx %>%
  left_join(movie_effect_reg, by='movieId') %>%
  left_join(user_effect_reg, by='userId') %>%
  group_by(year) %>%
  summarize(b_y = sum(rating - mu - b_i - b_u)/(n()+lambda), n_y = n())
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
#Predict ratings on validation set
predicted_ratings <- validation %>%
  left_join(movie_effect_reg, by='movieId') %>%
  left_join(user_effect_reg, by='userId') %>%
  left_join(year_reg_avgs, by = 'year') %>%
  mutate(pred = mu + b_i + b_u + b_y) %>%
  .$pred
model_rmse <- RMSE(validation$rating,predicted_ratings)
```

Result

The RMSE value of Regularized Movie, User, Year Effect Model is given below.

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

method	RMSE
Regularized Movie, User, Year Effect Model	0.8648841

Concluding Remarks

A deeper insight into the data revealed some data point in the features have large effect on errors. So a regularization model was used to penalize such data points. The final RMSE is 0.8648 lower than the initial evaluation criteria 0.8775 which is given by the goal of the project. We can also improve the RMSE by adding other effect such as genre, age. Complex machine learning models like Neural Networks,Item Based Collaborative Filtering can also improve the results further, but hardware limitations such as RAM are the constraint.