PH125.9x - Capstone Project - MovieLens

Matthew Manasterski, MBA

16/12/2019

Contents

1 Introduction

The following report is for the first Capstone project for the course: HarvardX - PH125.9x, which is based on Netflix Challenge that was announced on October 2006, a challenge to improve its recommendation algorithm by 10% and win a million dollars as described in Chapter 33.7 of the course textbook (Irizarry, 2019). The report uses initial code provided in the course to load and partition the MovieLens data and builds off of the partial solution described by Professor Irizarry in the course textbook, in Chapters 33.7 and 33.9.

2 Overview

Recommendation systems using Machine Learning have become a standard with competing streaming services like Netflix and Amazon Prime Video, retailers like Walmart and Amazon also make a heavy use of these algorithms to recommend products based on customer reviews, purchase and browsing history to maximize their sales. In this Capstone project we demonstrate the use of Machine Learning algorithms to create recommendation engine based on 10MB version of the MovieLens dataset to generate predicted movie ratings, as with the Netflix challenge we will using the typical error loss, residual mean squared error (RMSE) on a test set to decide how much improvement was made in our algorithm. RMSE is denoted with the following formula:

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

3 Data

The Data for this Capstone project has been provided by the course and made available by GroupLens on their website. The full MovieLens dataset is 265MB and has 27,000,000 ratings by 280,000 users. To make computations easier on desktop computers 10MB subset version of this orginale dataset has been provided. And it can be downloaded here: https://grouplens.org/datasets/movielens/10m/

In this dataset each row represents a rating given by one user to one movie ratio. Each row has the following columns:
-userId: unique Id for the each user
-movieId: unique Is for the each movie
-timestamp: timestamp in POSix when the rating was given
-title: movie title ending with release year in brackets (YYYY)
-genres: genre associated with the movie
-rating: rating between 0 and 5 for the movie given by a each user

The initial code to download the dataset, load it and split it into a train and test sets have been provided by the HarvardX - PH125.9x course and is initial used for this project below.

```r
#################################
# Create edx set, validation set -  next few lines provided by edX
#################################

# Note: this process could take a couple of minutes

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

# 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")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))

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

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

# Make sure userId and movieId in validation set are also in edx set

validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set

removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

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

#### End of the code provided by edX
```

4 Data Exploration, Visualization and Analysis

4.1 Dataset Structure

Now that the dataset has been downloaded first look at the structure of the datasets using head() functon.

edx data:

```r
head(edx)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1                 Comedy|Romance
## 2          Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5        Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7        Children|Comedy|Fantasy
```

Test validation data:

```r
head(validation)
```

```
##   userId movieId rating timestamp
## 1      1     231      5 838983392
## 2      1     480      5 838983653
## 3      1     586      5 838984068
## 4      2     151      3 868246450
## 5      2     858      2 868245645
## 6      2    1544      3 868245920
##                                                      title
## 1                                     Dumb & Dumber (1994)
## 2                                     Jurassic Park (1993)
## 3                                        Home Alone (1990)
## 4                                           Rob Roy (1995)
## 5                                    Godfather, The (1972)
## 6 Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
##                                         genres
## 1                                       Comedy
## 2        Action|Adventure|Sci-Fi|Thriller
## 3                              Children|Comedy
## 4                Action|Drama|Romance|War
## 5                                  Crime|Drama
## 6 Action|Adventure|Horror|Sci-Fi|Thriller
```

Before exploring the data further check for missing values in edx, and validation sets and strip if any found.

```r
any(is.na(edx))
```

```
## [1] FALSE
```

```r
any(is.na(validation))
```

```
## [1] FALSE
```

No missing values found.

We see from the above that datasets have the same columns and structure, further exploration will be done on the edx set.

Structure of the data set.

```r
str(edx)
```

```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 8
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|A
```

Summary of the edx data structure.

```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
##
##
##
```

Number of rows and columns:

```r
nrow(edx)
```

```
## [1] 9000055
```

```r
ncol(edx)
```

```
## [1] 6
```

```r
dim(edx)
```

```
## [1] 9000055       6
```

Number of distinc movies:

```r
n_distinct(edx$movieId)
```

```
## [1] 10677
```

Number of distinct members:

```r
n_distinct(edx$userId)
```

```
## [1] 69878
```

4.2 Ratings Occurances

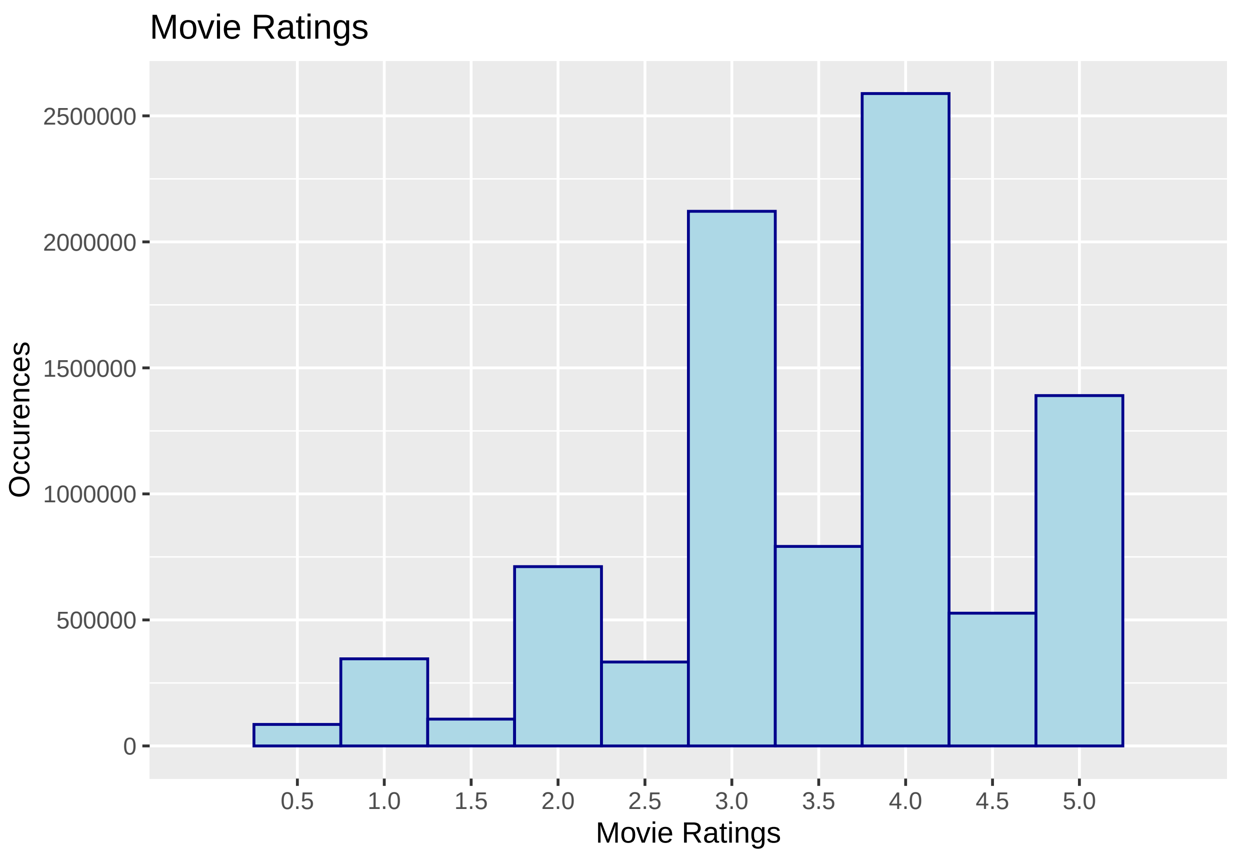
What are the five most given ratings in order from most to least?

```r
edx %>% group_by(rating) %>% summarize(count = n()) %>% top_n(5) %>%
  arrange(desc(count))
```

```
## Selecting by count
```

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

```r
ggplot(edx,aes(x=rating)) +
  geom_histogram(binwidth=0.5,color='darkblue',fill='lightblue') +
  xlab('Movie Ratings') +
  scale_x_discrete(limits = c(seq(0.5,5,0.5))) +
  ylab('Occurences') +
  scale_y_continuous(breaks = c(seq(0, 3000000, 500000))) +
  ggtitle('Movie Ratings')
```



From the above table and plot we can see that rating of 4 was used the most followed by rating of 3, 5, 3.5 and 2. In general half ratings were not used as often as full rating.

4.3 Movie effect - Explore Movie rating by occurance.

Show movies with the greatest number of ratings

```
edx %>% group_by(movieId, title) %>%
  summarize(count = n()) %>%
  arrange(desc(count))
```

```
## # A tibble: 10,677 x 3
## # Groups:   movieId [10,677]
##    movieId title                                                        count
##      <dbl> <chr>                                                        <int>
##  1     296 Pulp Fiction (1994)                                          31362
##  2     356 Forrest Gump (1994)                                          31079
##  3     593 Silence of the Lambs, The (1991)                             30382
##  4     480 Jurassic Park (1993)                                         29360
##  5     318 Shawshank Redemption, The (1994)                             28015
##  6     110 Braveheart (1995)                                            26212
##  7     457 Fugitive, The (1993)                                         25998
##  8     589 Terminator 2: Judgment Day (1991)                            25984
##  9     260 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977) 25672
## 10     150 Apollo 13 (1995)                                             24284
## # ... with 10,667 more rows
```

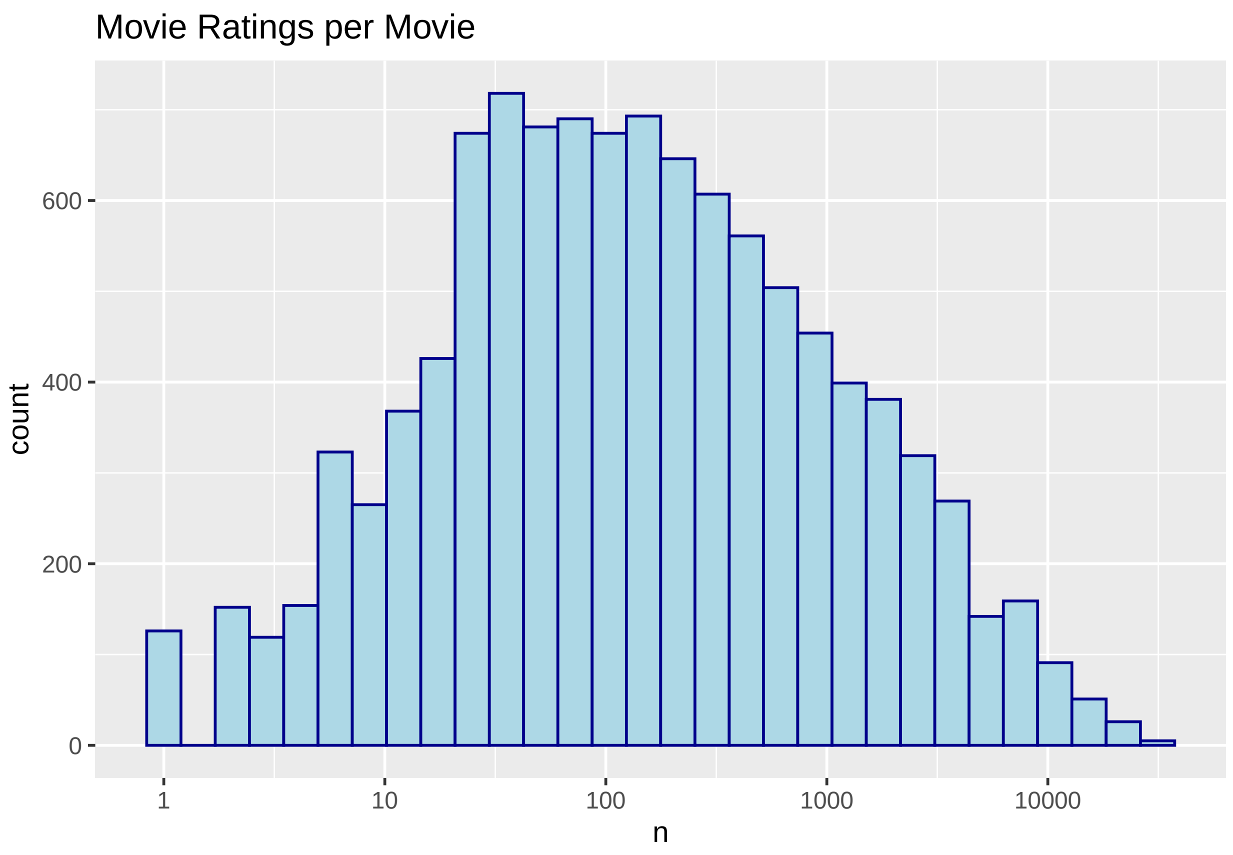
Show movies with the least number of ratings

```
edx %>% group_by(movieId, title) %>%
  summarize(count = n()) %>%
  arrange(count)
```

```
## # A tibble: 10,677 x 3
## # Groups:   movieId [10,677]
##    movieId title                                              count
##      <dbl> <chr>                                              <int>
##  1    3191 Quarry, The (1998)                                     1
##  2    3226 Hellhounds on My Trail (1999)                          1
##  3    3234 Train Ride to Hollywood (1978)                         1
##  4    3356 Condo Painting (2000)                                  1
##  5    3383 Big Fella (1937)                                       1
##  6    3561 Stacy's Knights (1982)                                 1
##  7    3583 Black Tights (1-2-3-4 ou Les Collants noirs) (1960)    1
##  8    4071 Dog Run (1996)                                         1
##  9    4075 Monkey's Tale, A (Les Château des singes) (1999)       1
## 10    4820 Won't Anybody Listen? (2000)                           1
## # ... with 10,667 more rows
```

Plot Ratings per Movie

```
edx %>% count(movieId) %>%
ggplot(aes(x=n)) +
  geom_histogram(bins=30,color='darkblue',fill='lightblue') +
  scale_x_log10() +
  ggtitle('Movie Ratings per Movie')
```



From the above tibbles and plot we can deduce that some movies have been rated multiple thousands of times while others have been only rated few times, some only once. This makes sense as blockbusters like "Pupl Fiction" and "Forrest Gump" would have millions of viewers while other smaller films would have very few viewers. This is something we will have to take into account when generating our model.

4.4 User Effect - Explore number of ratings per User

Show Users with the greatest number of ratings

```
edx %>% group_by(userId) %>%
  summarize(count = n()) %>%
  arrange(desc(count))
```

```
## # A tibble: 69,878 x 2
##    userId count
##     <int> <int>
##  1  59269  6616
##  2  67385  6360
##  3  14463  4648
##  4  68259  4036
##  5  27468  4023
##  6  19635  3771
##  7   3817  3733
##  8  63134  3371
##  9  58357  3361
## 10  27584  3142
## # ... with 69,868 more rows
```

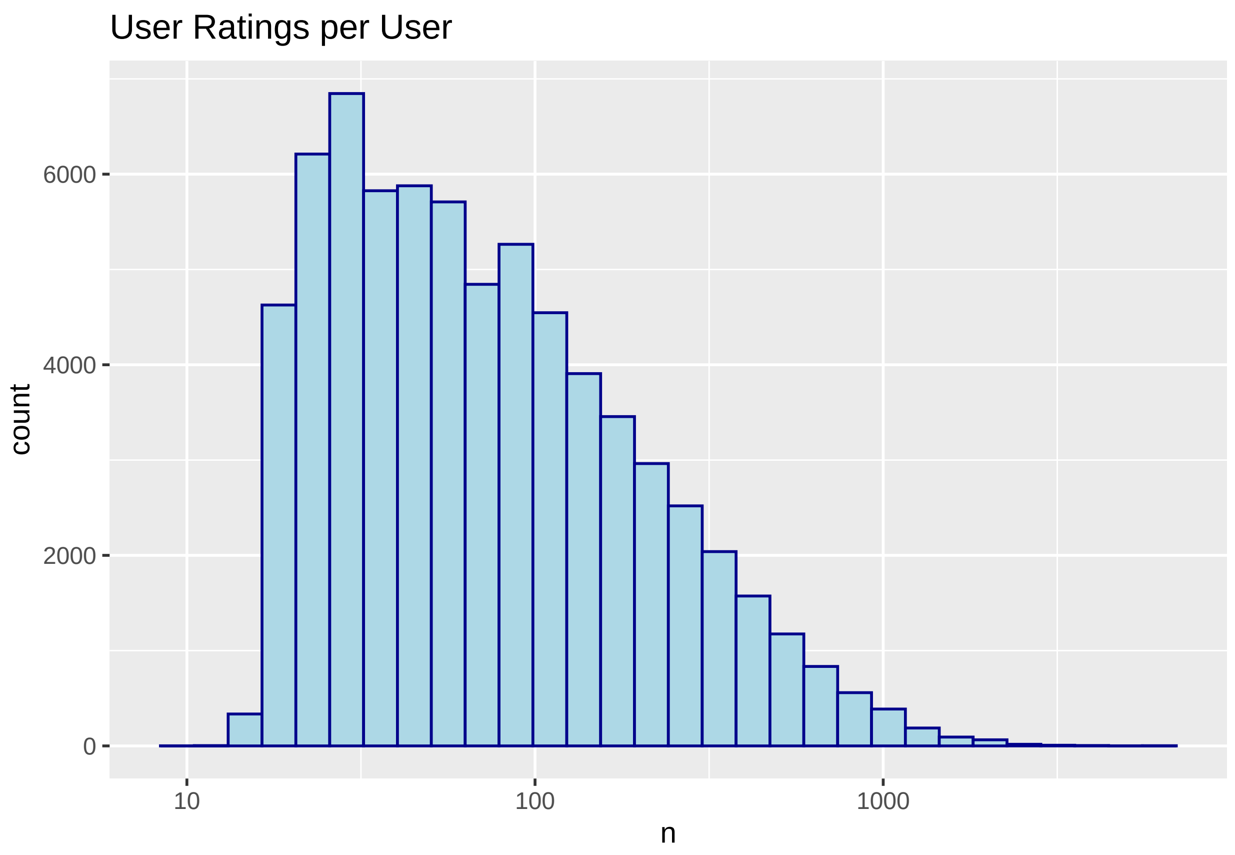
Show users with the least number of ratings

```
edx %>% group_by(userId) %>%
  summarize(count = n()) %>%
  arrange(count)
```

```
## # A tibble: 69,878 x 2
##    userId count
##     <int> <int>
##  1  62516    10
##  2  22170    12
##  3  15719    13
##  4  50608    13
##  5    901    14
##  6   1833    14
##  7   2476    14
##  8   5214    14
##  9   9689    14
## 10  10364    14
## # ... with 69,868 more rows
```

Plot User ratings per users

```
edx %>% count(userId) %>%
  ggplot(aes(x=n)) +
  geom_histogram(bins=30,color='darkblue',fill='lightblue') +
  scale_x_log10() +
  ggtitle('User Ratings per User')
```



From the above tibbles and Plot we can see some users are more active than others. We have users that have rated thousands of movies and we have users that only rated as few as 10 movies. This is something we will have to take into account when generating our model.

4.5 Date effect - Explore age at the time of rating, release date, and rating date of the movie

Timestamp is not really readable to us. Let's pull the year of the review from the timestamp and add "ratingYear" column containing it for each rating.

We will require lubridate library for that.

```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
```

Get year of the rating.

```r
edx <- mutate(edx, ratingYear = year(as_datetime(timestamp)))
```

As we see from the structure of the title in the dataset, each title ends with the release date in brackets, this let us easily seperate the release date into another column.

```r
edx <- edx %>% mutate(releaseYear = as.numeric(str_sub(title, -5, -2)))
```

We also want to calculate the age of the movie at the time of the rating. The following function helps us calculate the age of the movie at the time of the rating.

```r
calculateAgeAtRating <- function(x,y){
  return(x-y)
}
edx$ageAtRating <- mapply(calculateAgeAtRating, edx$ratingYear, edx$releaseYear)
```

Show new structure with additional three columns:

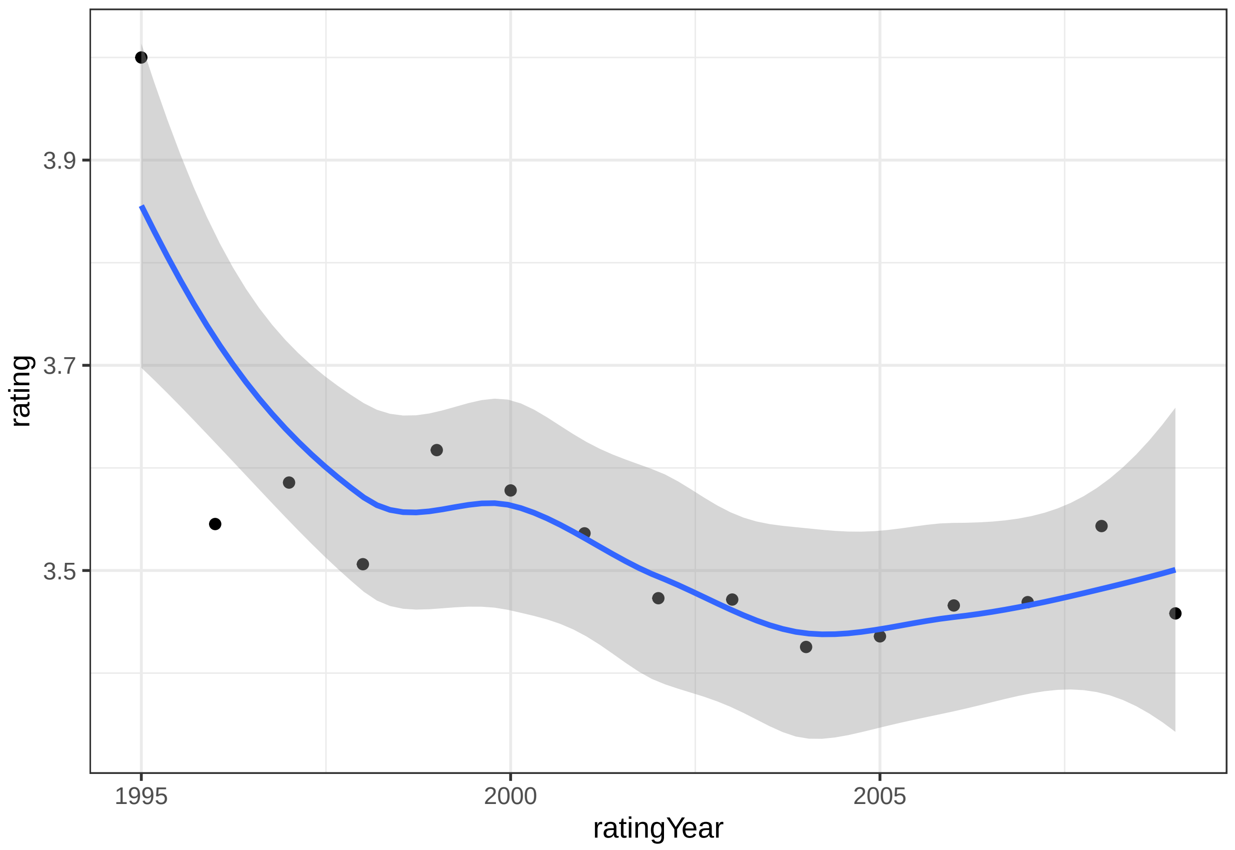
```r
head(edx)
```

```
##   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 ratingYear releaseYear ageAtRating
## 1                Comedy|Romance       1996        1992           4
## 2         Action|Crime|Thriller       1996        1995           1
## 3  Action|Drama|Sci-Fi|Thriller       1996        1995           1
## 4       Action|Adventure|Sci-Fi       1996        1994           2
## 5 Action|Adventure|Drama|Sci-Fi       1996        1994           2
## 6       Children|Comedy|Fantasy       1996        1994           2
```

Plot average rating by year movie was rated:

```r
edx %>% group_by(ratingYear) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(ratingYear,rating)) +
  geom_point() +
  geom_smooth() +
  theme_bw()
```

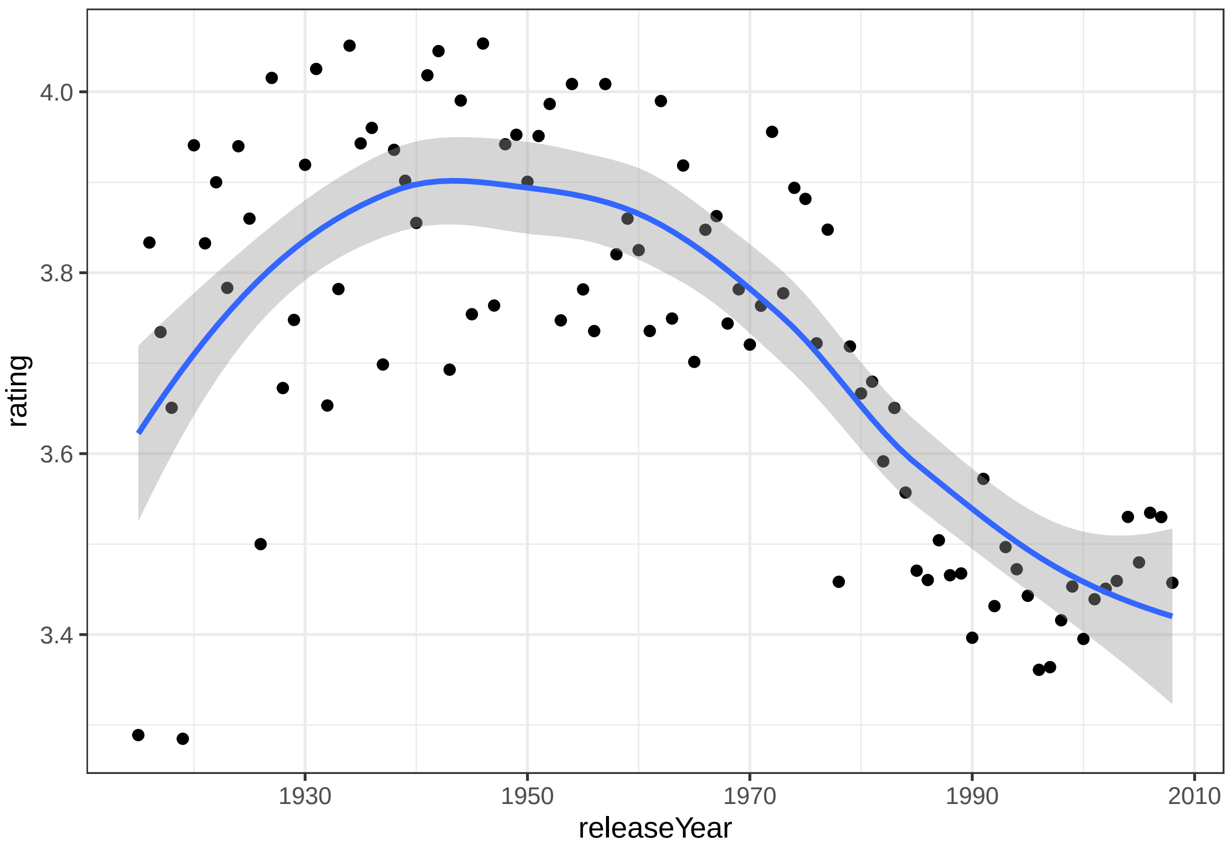
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Plot average rating by year movie was released:

```r
edx %>% group_by(releaseYear) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(releaseYear,rating)) +
  geom_point() +
  geom_smooth() +
  theme_bw()
```

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

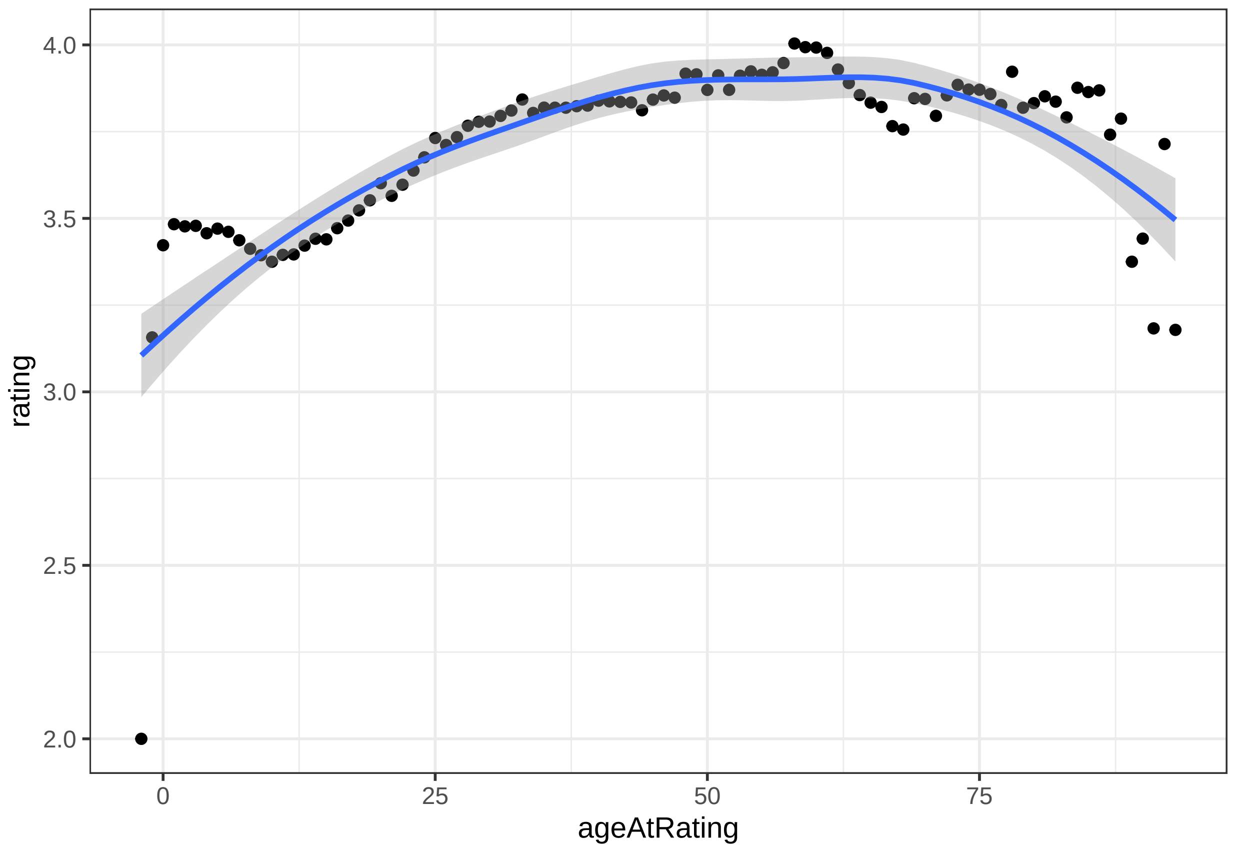


From the above plot we can see that movies rated between 1930 and 1960 have the highest average ratings, this is something we might want to consider in the future.

Plot average rating by the age of the movie when it was rated:

```
edx %>% group_by(ageAtRating) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(ageAtRating,rating)) +
  geom_point() +
  geom_smooth() +
  theme_bw()
```

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



Similar to the previous plot, this shows movies that are older, 50 to 60 years old have the highest average ratings.

5 Modeling and Results

5.1 Create a train and test sets for our modeling

Before analyzing the data and coming up with the best model we will split edx dataset to create a train and test sets for our model analysis.

```r
set.seed(1, sample.kind="Rounding")
# if using R 3.5 or earlier, use `set.seed(1)` instead
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train <- edx[-test_index,]
test_set <- edx[test_index,]

test <- test_set %>%
  semi_join(train, by = "movieId") %>%
  semi_join(train, by = "userId")

# Add rows removed from validation set back into edx set

removed_set <- anti_join(test_set, test)
train <- rbind(train, removed_set)

rm(test_set, removed_set)
```

Now we have train and test sets we can begin to develop our models.

5.2 RMSE function

As we have shown the RMSE formula in the overview section, let write out the function in R.

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

5.3 First - Simplest model

The Simplest model of prediction would be just to take the average mean movie rating for all movies regardless of the user variation. Professor Irizarry writes ‘A model that assumes the same rating for all movies and users with all the differences explained by random variation would look like this:

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

with $\epsilon_{u,i}$ independent error sample from the same distribution centered at 0 and μ the “true” rating for all movies’(Irizarry, 2019). Estimate of mu that would minimize RSME in this case would be mean movie rating.

We depict mean movie rating with mu.

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

```
## [1] 3.512456
```

Calulate RSME for mean prediction We will use the following formula

```
mean_rmse <- RMSE(test$rating, mu)
mean_rmse
```

```
## [1] 1.060054
```

We will create a new variable “change” which will track percentage change between our starting Naive mean model and our improved model as we go along.

```
# Change in % between original mean_rmse and current model.
change=0
```

We will store the results in the data frame so we can keep track of the progress and improvements we make.

```
rmse_results <- data_frame(method = "Mean movie rating model",
                           RMSE = mean_rmse,
                           Change = change )
```

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

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

method	RMSE	Change
Mean movie rating model	1.060054	0

5.4 Movie effect model

In the course textbook Professor Irizarry states that we know from experience that some movies are just generally rated higher than others (Irizarry, 2019). Blockbusters are usually ranked higher then independent films. We can improve our current model by including movie effect estimate - movie bias ‘b_i’ for all movies in a previous model. So the new model would read:

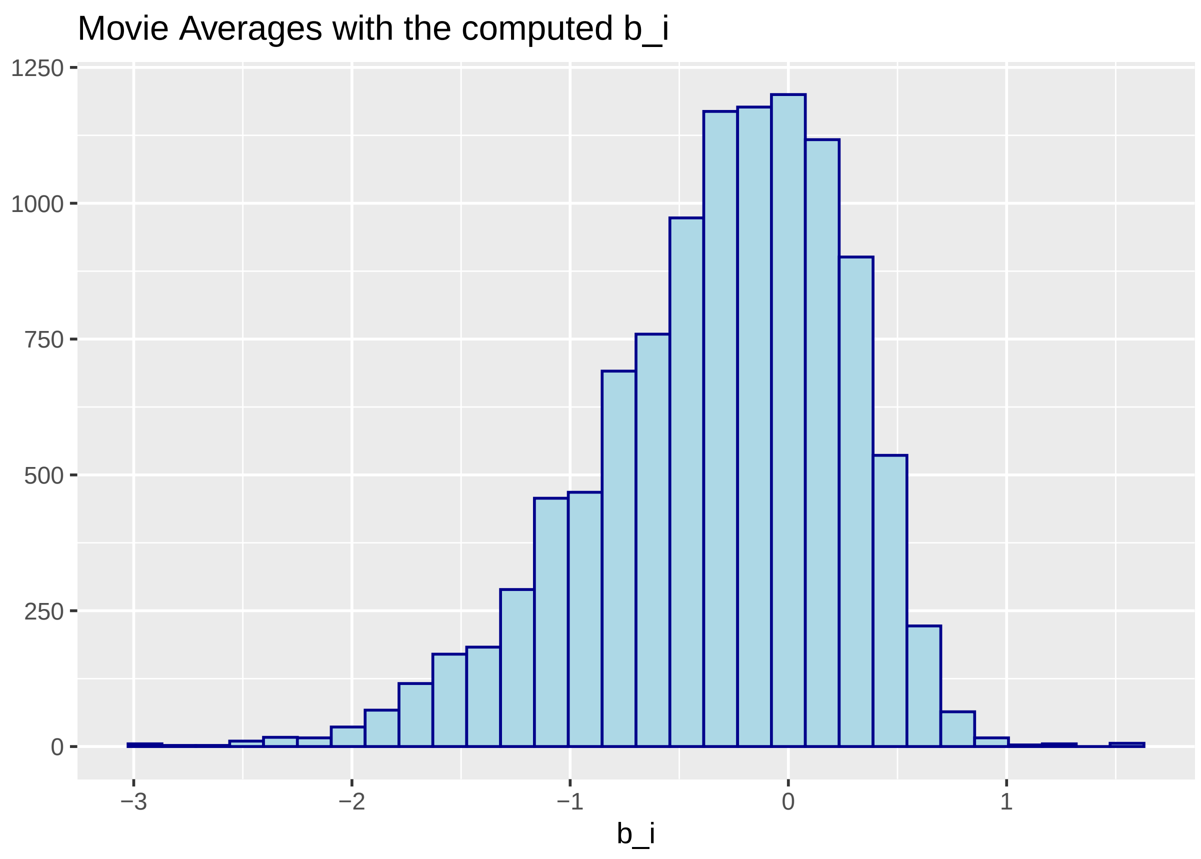
$$Y_{u,i} = \mu + b_i + \epsilon_{u,i}$$

We know that least square estimate bi is just the average of rating minus mean rating.

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

Plot with above bias

```r
movie_avgs %>% qplot(b_i, geom ="histogram", bins = 30,
                     data = ., color = I("darkblue"),
                     fill = I("lightblue"),
                     main = "Movie Averages with the computed b_i")
```



Calculate new prediced ratings with above movie bias

```r
predicted_ratings <- mu + test %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
```

Calculate rmse after modeling movie bias

```r
moviebias_rmse <- RMSE(predicted_ratings, test$rating)

moviebias_rmse
```

```
## [1] 0.9429615
```

```r
#re-calculate change
change = (((mean_rmse - moviebias_rmse)/mean_rmse) * 100)
# Update rmse_results table
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Bias model",
                                     RMSE = moviebias_rmse, Change = change))
rmse_results %>% knitr::kable()
```

method	RMSE	Change
Mean movie rating model	1.0600537	0.00000
Movie Bias model	0.9429615	11.04587

We see from the above table that we improved RMSE by 11.05% to 0.94296.

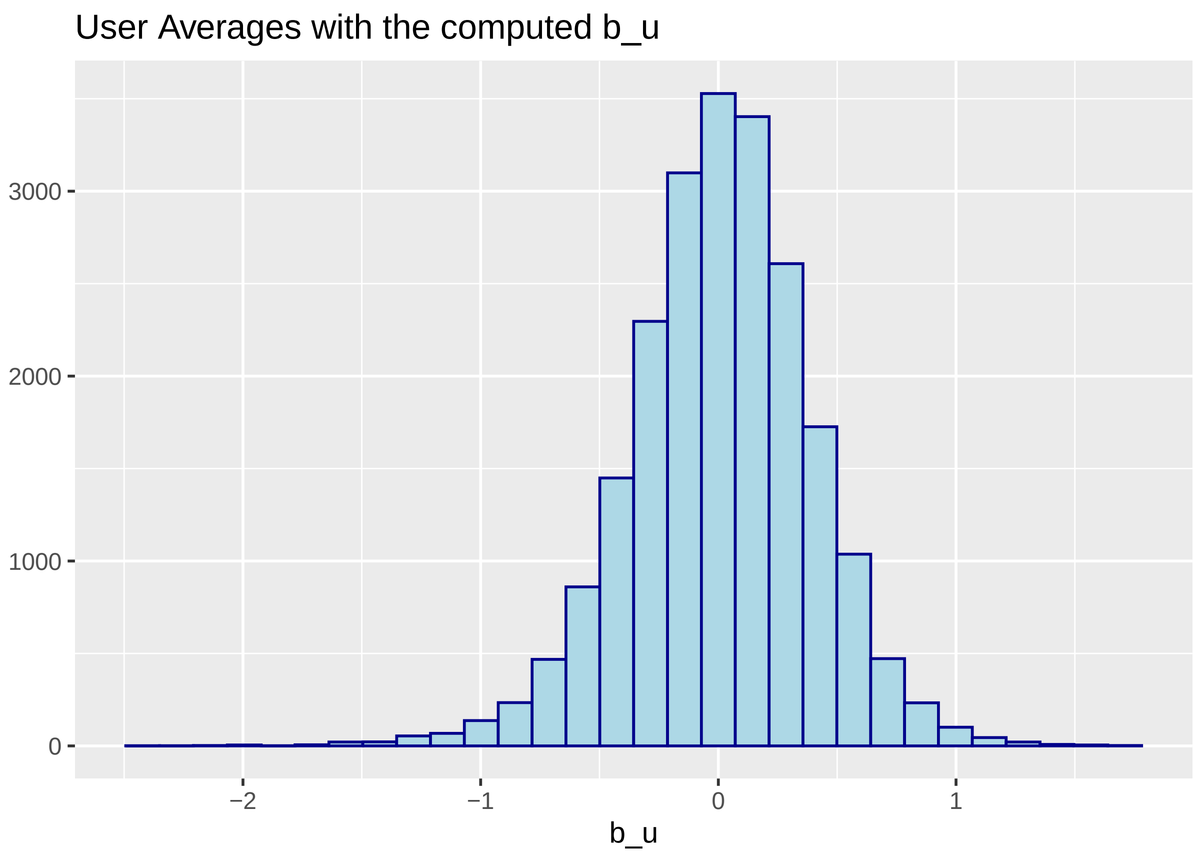
5.5 Add User effect to the model

Next we add User bias effect to Movie bias model. We know that some users are more generous with their ratings than others. We will represent user bias with b_u. And add it to our previous model which will now read:

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

We will first show the bias for users with more than 100 ratings. Plot user bias effect for users with at least 100 ratings

```r
mu <- mean(train$rating)
user_100avg<- train %>%
  na.omit() %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  filter(n() >= 100) %>%
  summarize(b_u = mean(rating - mu - b_i))
user_100avg %>% qplot(b_u, geom ="histogram", bins = 30,
                      data = ., color = I("darkblue"),
                      fill = I("lightblue"),
                      main = "User Averages with the computed b_u")
```



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

# rmse results with User bias effect and Movie bias model
predicted_ratings <- test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  na.omit() %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

user_movie_bias_rmse <- RMSE(predicted_ratings, test$rating)
user_movie_bias_rmse
```

```
## [1] 0.8646843
```

```r
#re-calculate change
change = (((mean_rmse - user_movie_bias_rmse)/mean_rmse) * 100)
# Update rmse_results table
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="User and Movie Bias model",
                                     RMSE = user_movie_bias_rmse,
                                     Change = change ))

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

method	RMSE	Change
Mean movie rating model	1.0600537	0.00000
Movie Bias model	0.9429615	11.04587
User and Movie Bias model	0.8646843	18.43014

We see from the above table that by adding user effect we improved RMSE by 18.43% from our original model to 0.86468.

5.6 Regularization Model

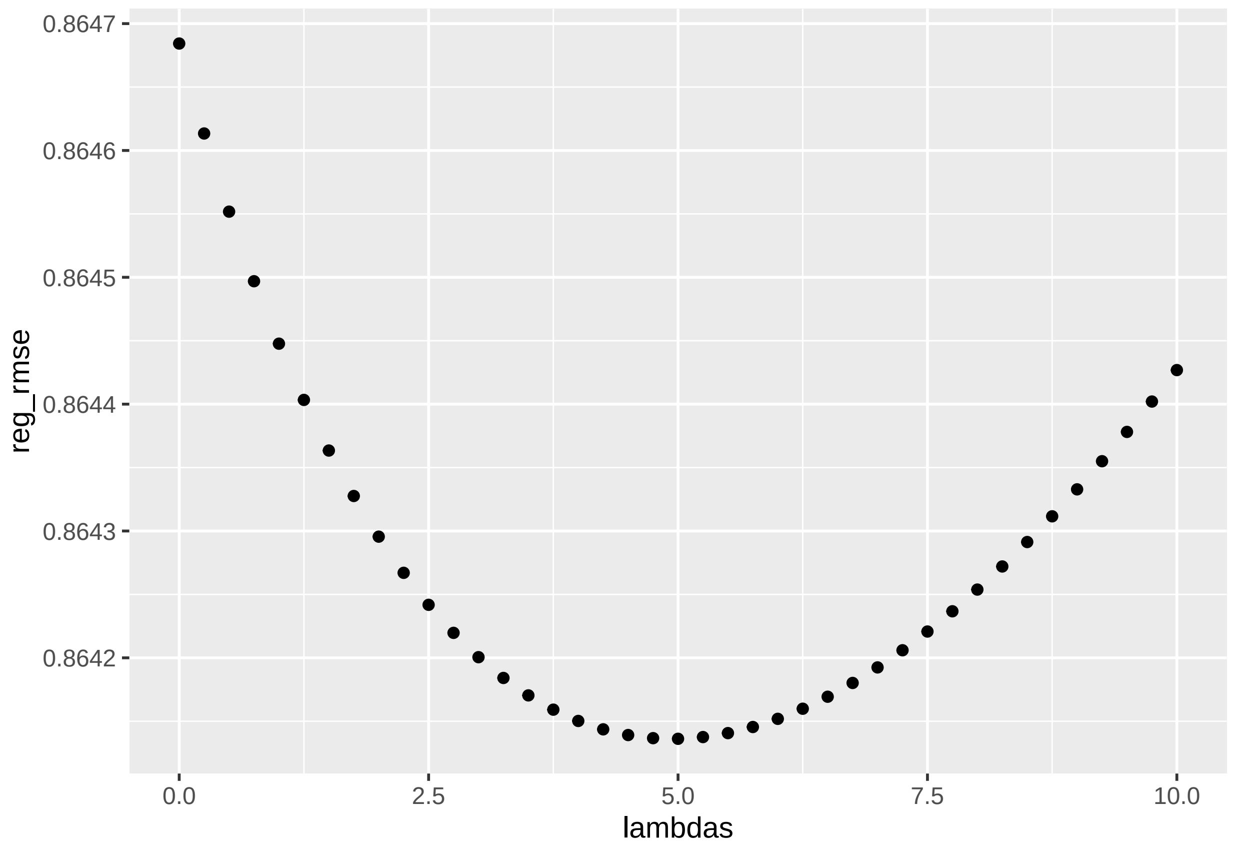
As we have already discusses there is a large variance in the number of times movies are rated, some are rated thousands of times, others very few, and some even only once. The same goes for the users, some users are very active rating almost every movie, while others only rated few movies. Next we will use regularization that will help constrain this variability. We will run a function that will help choose penalty variable lamda in the penalized regression to control the total variability of the movie and user effects.

We will set lamda from 0 to 10, increments of .25

```r
lambdas <- seq(0, 10, 0.25)

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

  mu <- mean(train$rating)

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

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

  predicted_ratings <-
    test %>%
    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, test$rating))
})

#lowest regularized rmse

reg_optimal_rmse <- min(reg_rmse)
reg_optimal_rmse
```

```
## [1] 0.8641362
```

```r
# Plot rmses vs lambdas to select the optimal lambda
qplot(lambdas, reg_rmse)
```



```r
# Find optimal lambda
lambda <- lambdas[which.min(reg_rmse)]
lambda
```

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

```r
#re-calculate change
change = (((mean_rmse - reg_optimal_rmse)/mean_rmse) * 100)
# Update rmse_results table
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized movie/user bias model",
                                     RMSE = reg_optimal_rmse,
                                     Change = change))

# print rmse_ results
rmse_results %>% knitr::kable()
```

method	RMSE	Change
Mean movie rating model	1.0600537	0.00000
Movie Bias model	0.9429615	11.04587
User and Movie Bias model	0.8646843	18.43014
Regularized movie/user bias model	0.8641362	18.48185

We see from the above table that by regulizing movie and user effect we were further able to improve our RMSE to 0.86414 improving the original Mean model by 18.48%.

We could go further and use Release Year (releaseYear) or Age at the time of rating (ageAtRating) to further try to minimize RSME, but since we achieved our RSME goal we will stop here and verify our last model "Regularized movie/user bias model" against Validation set.

6 Validation of the final model

Before our final test we should make sure edx and validation sets have the same number of variables and since we did not add the date columns to validation set and will not be using them it is a good idea to remove them from edx set.

```
# Remove the three date columns we added
edx <- select(edx, -ratingYear, -releaseYear, -ageAtRating)

# double check they were removed.
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
## 1                Comedy|Romance
## 2         Action|Crime|Thriller
## 3  Action|Drama|Sci-Fi|Thriller
## 4       Action|Adventure|Sci-Fi
## 5 Action|Adventure|Drama|Sci-Fi
## 6       Children|Comedy|Fantasy
```

Now we can run the final test with the penelty lambda we found earlier.

```
#Run a final Test with Our original Validation set
l <- lambda

mu <- mean(edx$rating)

# re-calculate initial mean RSME against edx set
mean_rmse <- RMSE(validation$rating, mu)
mean_rmse
```

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

```
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") %>%
```

```r
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

final_rmse <- RMSE(predicted_ratings, validation$rating)

# Final RMSE against Validation set with Regularized movie/user bias model
final_rmse
```

```
## [1] 0.8648177
```

```r
#re-calculate change
change = (((mean_rmse - final_rmse)/mean_rmse) * 100)


# Add Validation set to the results table
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized movie/user model- Validation",
                                     RMSE = final_rmse,
                                     Change = change))

# print rmse_ results
rmse_results %>% knitr::kable()
```

method	RMSE	Change
Mean movie rating model	1.0600537	0.00000
Movie Bias model	0.9429615	11.04587
User and Movie Bias model	0.8646843	18.43014
Regularized movie/user bias model	0.8641362	18.48185
Regularized movie/user model- Validation	0.8648177	18.50582

We see that we have achieved RMSE of 0.86482 on original edx and validation set, below our target of 0.8649, with total improvement from simple mean movie model of 18.51%.

7 Conclusion

In this Capstone project we were tasked with simulating Netflix challenge and improving on a Movie recommendation system. Netflix challenge was to improve the system by 10%. We started with the simplest model which gave every movie the same average (mean) rating. The RMSE for this system was 1.06, which means that an average error of over 1 star in most cases. We started developing our model by taking into consideration movie bias effect, this vastly improved out model as measured by RMSE of 0.94296, and improvement of 11% as shown in the table above. We then added user bias effect to our model which further improved our model with the measure of RMSE of 0.86468 beating out project target of 0.8649 and giving us an improvement of 18.43% over the original Mean Model. We then went a step further and used regularization to regularize movie and user effect lowering our RMSE slightly further to 0.86414 and overall improvement of 18.48%.

At this point we were happy with our model and verified it against the original edx and validation datasets provided. With these datasets the RMSE was 0.86481 still below the project target of 0.8649 and overall improvement of 18.51% over the simple model.

We are happy with this improvement and can trust our movie predictions.

8 References:

Irizarry, Rafael A. (2020). ntroduction to Data Science, Data Analysis and Prediction Algorithms with R. Ch 33.7, 33.9. https://rafalab.github.io/dsbook/