HarvardX Capstone Project Using the Movielens Data

Alev Cieslinski

2022-02-15

I. Introduction and Project Background:

In October 2006, Netflix announced a contest, "The Netflix Prize". The competition quickly became popular among computer and data scientists, tech-savvy engineers and tech circles. In order to win the grand prize of $1,000,000, a participating team had to improve the RMSE by 10% and achieve 0.8572 or lower.

The purpose of this study is to utilize the movielens dataset and develop a series of machine learning models in R and achieve an RMSE lower than the target of 0.8572. The goal is to show a consistent decline in RMSE by employing a variety of methods.

II. Data Pull and Preparation

The initial code was provided in the Capstone course material. I added some additional code to create additional attributes for each of the data sets that I created f

```
#Movielens Data. This project will utilize the following dataset:
#https://grouplens.org/datasets/movielens/10m/.  The initial code was provide
d by the course
#materials and we used to do the data pull as shown below:

##########################################################
# Create edx set, validation set (final hold-out test set)
##########################################################

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

## Loading required package: tidyverse

## ── Attaching packages ─────────────────────────────────────── tidyverse 1.
3.1 ──

## ✓ ggplot2 3.3.5     ✓ purrr   0.3.4
## ✓ tibble  3.1.6     ✓ dplyr   1.0.8
## ✓ tidyr   1.2.0     ✓ stringr 1.4.0
## ✓ readr   2.1.2     ✓ forcats 0.5.1

## ── Conflicts ────────────────────────────────────────── tidyverse_conflict
s() ──
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

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

## Loading required package: caret

## Loading required package: lattice

##
## Attaching package: 'caret'

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

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")

movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))
```

I created two variables, 1) the movie rating year (rating_year) and 2) the movie release year (movie_year):

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

movielens <- mutate(movielens, rating_year= year(as.Date(as.POSIXct(timestamp
,
                                                            origin = "1970-01
-01"))))


#create movie year

movie_year <- stringi::stri_extract(movielens$title, regex = "(\\d{4})", comm
ents = TRUE) %>%
  as.numeric()
movielens <- movielens %>% mutate(movie_year = movie_year) %>% select(-timest
amp)
```

Let's see whether we find outliers in our dataset:

```
summary(movielens)

##      userId         movieId          rating          title
##  Min.   :    1   Min.   :    1   Min.   :0.500   Length:10000054
##  1st Qu.:18123   1st Qu.:  648   1st Qu.:3.000   Class :character
##  Median :35740   Median : 1834   Median :4.000   Mode  :character
##  Mean   :35870   Mean   : 4120   Mean   :3.512
##  3rd Qu.:53608   3rd Qu.: 3624   3rd Qu.:4.000
##  Max.   :71567   Max.   :65133   Max.   :5.000
##     genres           rating_year     movie_year
##  Length:10000054    Min.   :1995   Min.   :1000
##  Class :character   1st Qu.:2000   1st Qu.:1987
##  Mode  :character   Median :2002   Median :1994
##                     Mean   :2002   Mean   :1991
##                     3rd Qu.:2005   3rd Qu.:1998
##                     Max.   :2009   Max.   :9000

Based on the information provided for the dataset, i.e., being updated most r
ecently in 2018, we can find outlying values with the code below and correct
for them:

movielens %>% filter(movie_year > 2018) %>%
  group_by(movieId, title, movie_year) %>%
  summarize(n = n())

## `summarise()` has grouped output by 'movieId', 'title'. You can override u
sing
## the `.groups` argument.
```

```
## # A tibble: 6 × 4
## # Groups:   movieId, title [6]
##   movieId title                                         movie_year     n
##     <dbl> <chr>                                              <dbl> <int>
## 1     671 Mystery Science Theater 3000: The Movie (1996)       3000  3620
## 2    2308 Detroit 9000 (1973)                                  9000    24
## 3    4159 3000 Miles to Graceland (2001)                       3000   788
## 4    5310 Transylvania 6-5000 (1985)                           5000   218
## 5    8864 Mr. 3000 (2004)                                      3000   163
## 6   27266 2046 (2004)                                          2046   472
```

```
movielens %>% filter(movie_year < 1900) %>%
  group_by(movieId, title, movie_year) %>%
  summarize(n = n())
```

```
## `summarise()` has grouped output by 'movieId', 'title'. You can override u
sing
## the `.groups` argument.
```

```
## # A tibble: 8 × 4
## # Groups:   movieId, title [8]
##   movieId title                                                movie_year
n
##     <dbl> <chr>                                                     <dbl>
<int>
## 1    1422 Murder at 1600 (1997)                                      1600
1742
## 2    4311 Bloody Angels (1732 Høtten: Marerittet Har et Postnu…       1732
9
## 3    5472 1776 (1972)                                                1776
205
## 4    6290 House of 1000 Corpses (2003)                               1000
406
## 5    6645 THX 1138 (1971)                                            1138
525
## 6    8198 1000 Eyes of Dr. Mabuse, The (Tausend Augen des Dr. …       1000
30
## 7    8905 1492: Conquest of Paradise (1992)                          1492
152
## 8   53953 1408 (2007)                                                1408
520
```

```
#Correct movie dates dates

movielens[movielens$movieId == "27266", "movie_year"] <- 2004
movielens[movielens$movieId == "671", "movie_year"] <- 1996
movielens[movielens$movieId == "2308", "movie_year"] <- 1973
movielens[movielens$movieId == "4159", "movie_year"] <- 2001
movielens[movielens$movieId == "5310", "movie_year"] <- 1985
movielens[movielens$movieId == "8864", "movie_year"] <- 2004
movielens[movielens$movieId == "1422", "movie_year"] <- 1997
```

```
movielens[movielens$movieId == "4311", "movie_year"] <- 1998
movielens[movielens$movieId == "5472", "movie_year"] <- 1972
movielens[movielens$movieId == "6290", "movie_year"] <- 2003
movielens[movielens$movieId == "6645", "movie_year"] <- 1971
movielens[movielens$movieId == "8198", "movie_year"] <- 1960
movielens[movielens$movieId == "8905", "movie_year"] <- 1992
movielens[movielens$movieId == "53953", "movie_year"] <- 2007
```

I calculated the age of the movie based on current year to see if there's a relationship between movie age and rating distributions.

```
#Calculate the age of movie

movielens <-movielens %>% mutate(movie_age = 2022 - movie_year)

#Create the validation set that will be 10% of MovieLens data.

set.seed(1, sample.kind="Rounding")

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

test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, l
ist = 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", "title", "genres",
## "rating_year", "movie_year", "movie_age")

train <- rbind(edx, removed)

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

set.seed(1, sample.kind="Rounding")

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

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

Since validation data will be used at the very end to validate the model, we further create a train_data and test_data from the train dataset.

```r
#Matching userId and movieId in both train and test sets

test_data <- temp %>%
  semi_join(train_data, by = "movieId") %>%
  semi_join(train_data, by = "userId")

# Adding back rows into train set

removed <- anti_join(temp, test_data)
```

```
## Joining, by = c("userId", "movieId", "rating", "title", "genres",
## "rating_year", "movie_year", "movie_age")
```

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

rm(test_index, temp, removed)

summary(train)
```

```
##      userId         movieId          rating          title
##  Min.   :    1   Min.   :    1   Min.   :0.500   Length:9000055
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   Class :character
##  Median :35738   Median : 1834   Median :4.000   Mode  :character
##  Mean   :35870   Mean   : 4122   Mean   :3.512
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000
##  Max.   :71567   Max.   :65133   Max.   :5.000
##     genres            rating_year     movie_year     movie_age
##  Length:9000055     Min.   :1995   Min.   :1900   Min.   : 12.00
##  Class :character   1st Qu.:2000   1st Qu.:1987   1st Qu.: 24.00
##  Mode  :character   Median :2002   Median :1994   Median : 28.00
##                     Mean   :2002   Mean   :1990   Mean   : 31.73
##                     3rd Qu.:2005   3rd Qu.:1998   3rd Qu.: 35.00
##                     Max.   :2009   Max.   :2010   Max.   :122.00
```

```
head(train)

##    userId movieId rating                         title
## 1:      1     122      5              Boomerang (1992)
## 2:      1     185      5               Net, The (1995)
## 3:      1     292      5               Outbreak (1995)
## 4:      1     316      5               Stargate (1994)
## 5:      1     329      5 Star Trek: Generations (1994)
## 6:      1     355      5       Flintstones, The (1994)
##                           genres rating_year movie_year movie_age
## 1:                Comedy|Romance        1996       1992        30
## 2:         Action|Crime|Thriller        1996       1995        27
## 3:  Action|Drama|Sci-Fi|Thriller        1996       1995        27
## 4:       Action|Adventure|Sci-Fi        1996       1994        28
## 5: Action|Adventure|Drama|Sci-Fi        1996       1994        28
## 6:       Children|Comedy|Fantasy        1996       1994        28
```

III. Exploratory Data Analysis

EDA is utilized to further investigate the data set in an effort to discover patterns, spot anomalies and better formulate our hypotheses about the data, i.e., relationships between variable distributions such as movie age and ratings, etc.

First, we look at distinct values of users, movies, and genres.

```
#Check for unique values of users, movies, genres

train %>% summarize(unique_users = length(unique(userId)),
                   unique_movies = length(unique(movieId)),
                   unique_genres = length(unique(genres)))

##   unique_users unique_movies unique_genres
## 1        69878         10677           797

#Analysis of Rating
#Distribution and Avg. Rating by Genre
```

We can look at the distribution by genre with the following code:

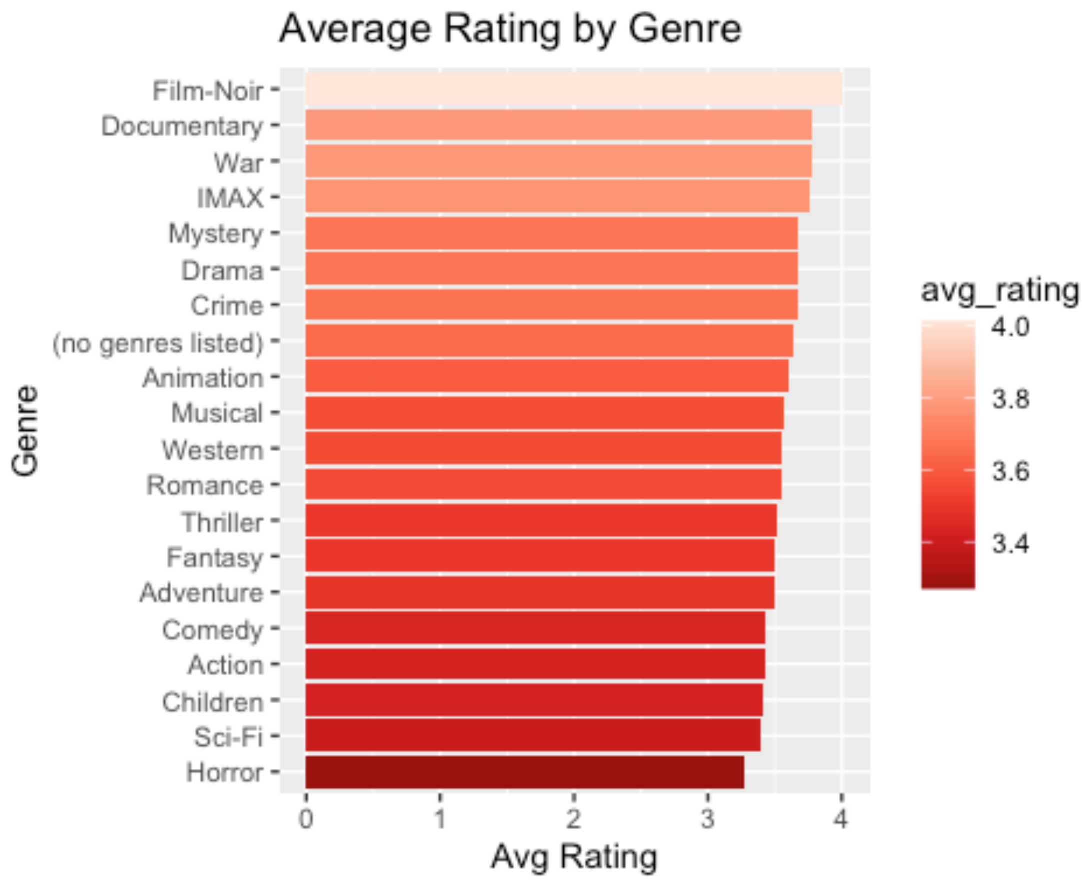
```
train %>% separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarize(count = n(), avg_rating = mean(rating)) %>%
  arrange(desc(count))

## # A tibble: 20 × 3
##    genres               count avg_rating
##    <chr>                <int>      <dbl>
##  1 Drama              3910127       3.67
##  2 Comedy             3540930       3.44
##  3 Action             2560545       3.42
```

```
##   4 Thriller            2325899       3.51
##   5 Adventure           1908892       3.49
##   6 Romance             1712100       3.55
##   7 Sci-Fi              1341183       3.40
##   8 Crime               1327715       3.67
##   9 Fantasy              925637       3.50
##  10 Children             737994       3.42
##  11 Horror               691485       3.27
##  12 Mystery              568332       3.68
##  13 War                  511147       3.78
##  14 Animation            467168       3.60
##  15 Musical              433080       3.56
##  16 Western              189394       3.56
##  17 Film-Noir            118541       4.01
##  18 Documentary           93066       3.78
##  19 IMAX                   8181       3.77
##  20 (no genres listed)        7       3.64
```

```
#Avg.rating by genre visually
```

There seems to be slight differences for rating, on average by genre and we can see this visually after running the code below:

```
train %>% separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarize(avg_rating = mean(rating)) %>%
  arrange(desc(avg_rating)) %>%
  ggplot(aes(reorder(genres, avg_rating), avg_rating, fill= avg_rating)) +
  geom_bar(stat = "identity") + coord_flip() +
  scale_fill_distiller(palette = "Reds") + labs(y = "Avg Rating", x = "Genre"
) +
  ggtitle("Average Rating by Genre")
```

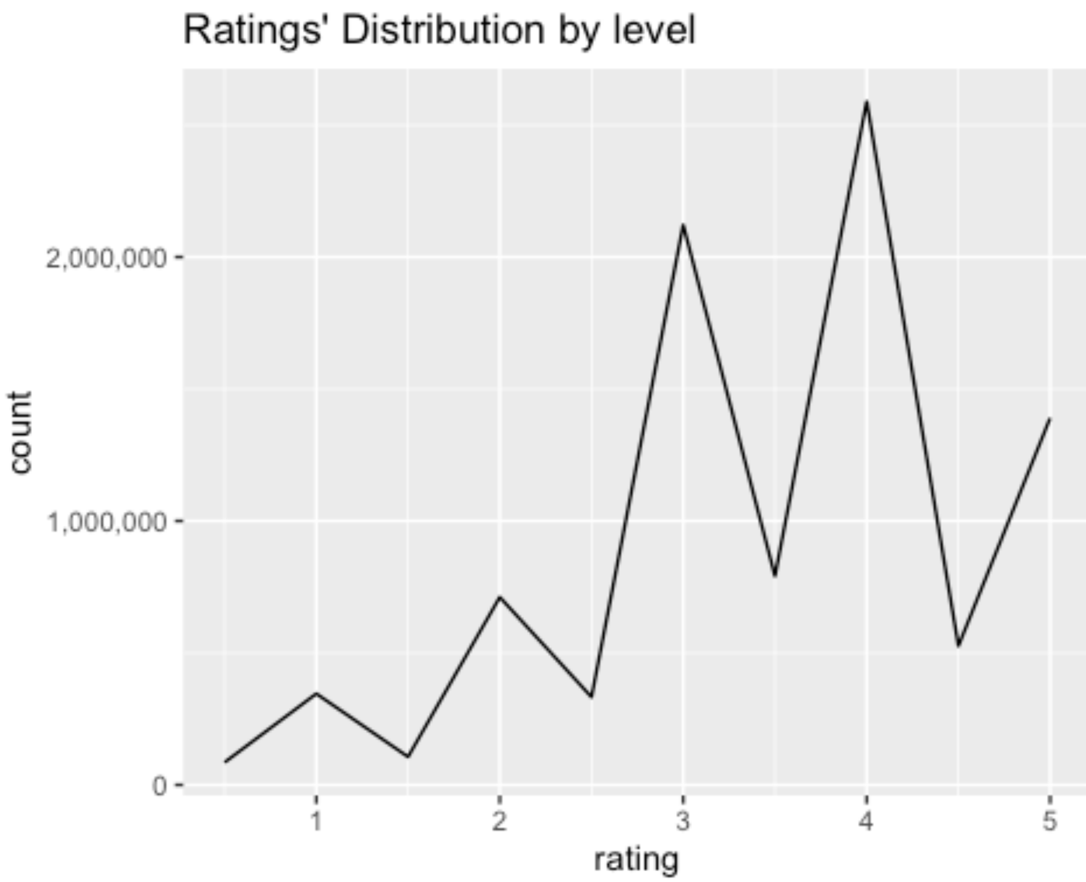



```
#Analysis of ratings: compare n of ratings by level
```

As we can see from the table and chart below, the majority of ratings cluster between 2.5 and 4.5.

```
train %>% group_by(rating) %>% summarize(count = n())

## # A tibble: 10 × 2
##    rating   count
##     <dbl>   <int>
##  1    0.5   85374
##  2    1    345679
##  3    1.5  106426
##  4    2    711422
##  5    2.5  333010
##  6    3   2121240
##  7    3.5  791624
##  8    4   2588430
##  9    4.5  526736
## 10    5   1390114
```

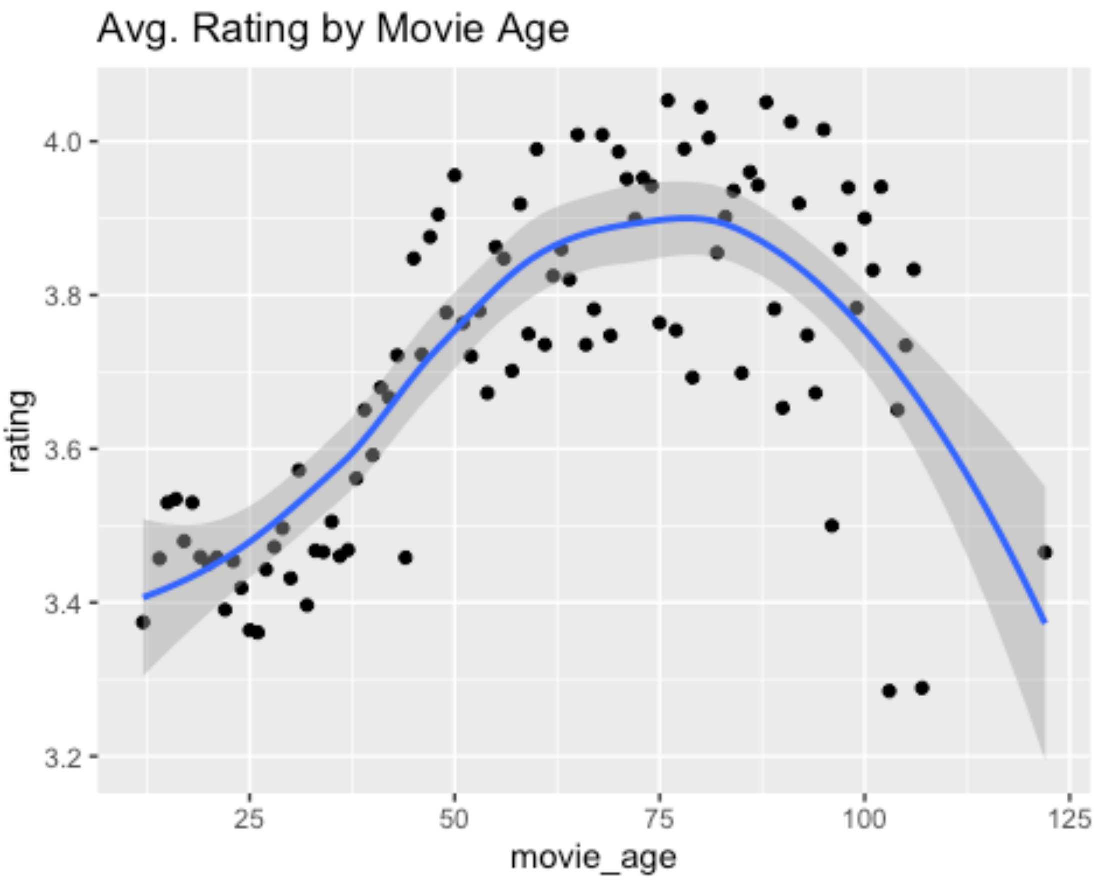
```r
#Ratings' distribution by level visually
  train %>%
  group_by(rating) %>%
  summarize(count = n()) %>%
  ggplot(aes(x = rating, y = count)) +
  scale_y_continuous(labels = scales::comma) +
  geom_line() +
  ggtitle("Ratings' Distribution by level")
```




```r
#Relationship between movie age and ratings
```

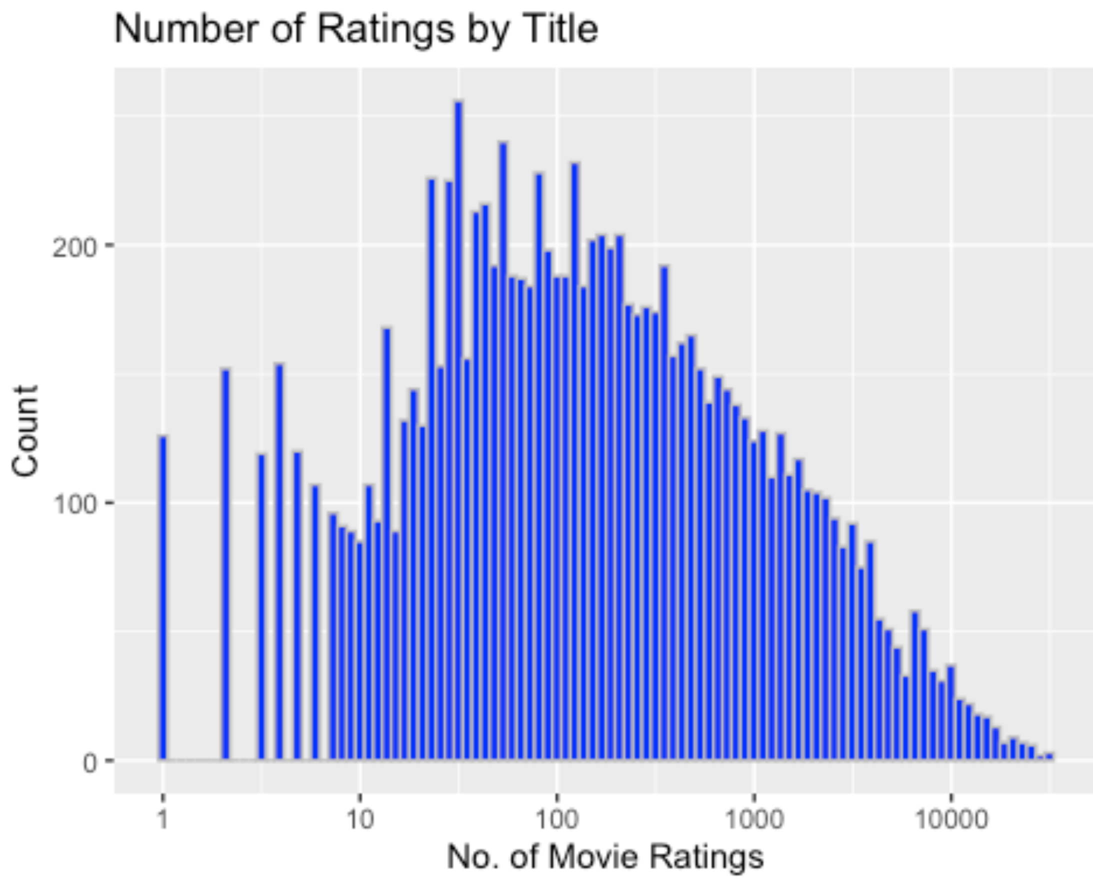
Looking at the distribution of movie age, we see a positive relationship between movie rating and the age of movies. This positive correlation turns negative when the age of the movie is ~75 plus years.

```r
train %>% group_by(movie_age) %>%
    summarize(rating = mean(rating)) %>%
    ggplot(aes(movie_age, rating)) +
    geom_point() +
    geom_smooth() +
  ggtitle("Avg. Rating by Movie Age")

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



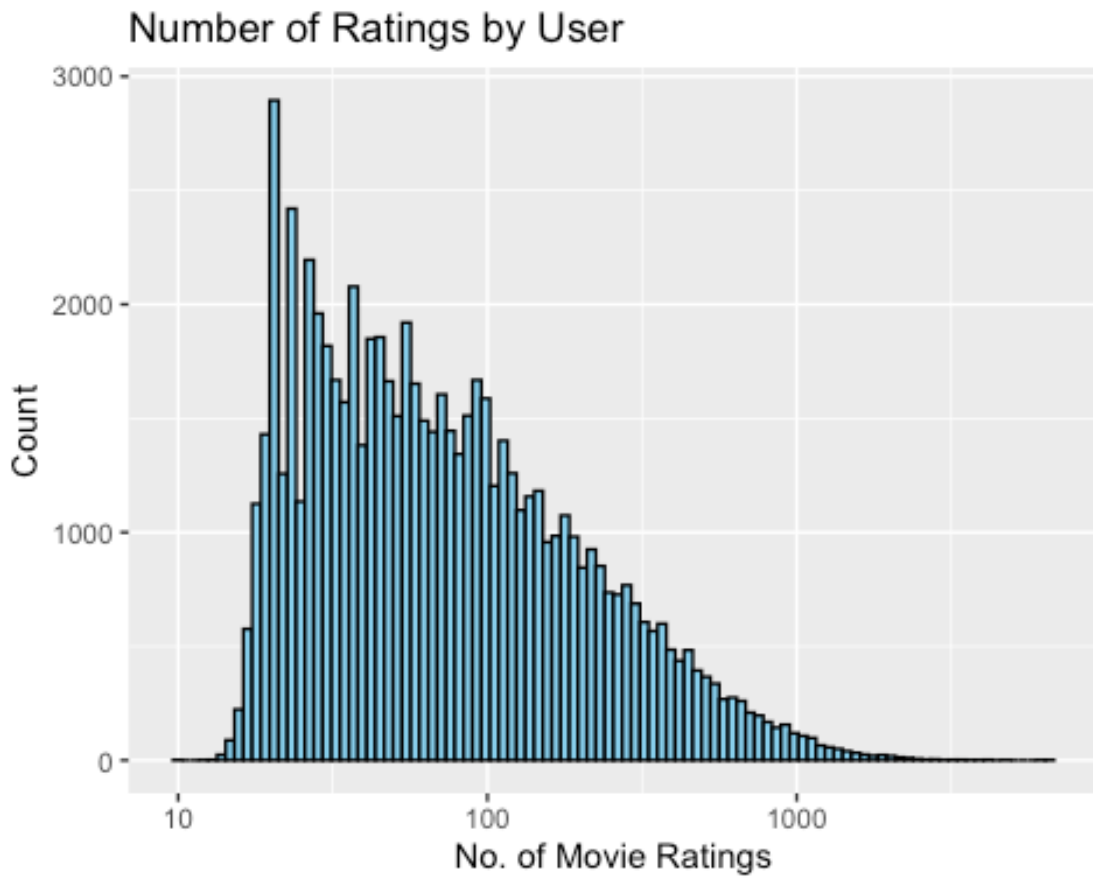
Next, we can look at the distribution of ratings by movie titles. We see from the chart below that some movies got a lot of ratings and some only got a few.

```
#Distribution of ratings by title

  train %>% group_by(title) %>% summarize(count = n()) %>%
    ggplot(aes(count)) + geom_histogram(fill = "blue", color = "grey", bins =
100) +
    labs(x = "No. of Movie Ratings", y = "Count") +
    scale_x_continuous(trans="log10") +
  ggtitle("Number of Ratings by Title")
```



The distribution of ratings by users shows that some users rated a lot of movies and some rated only a few. The distribution is skewed to the right.

```
#Distribution of ratings by users

train %>% group_by(userId) %>% summarize(count = n()) %>%
    ggplot(aes(count)) + geom_histogram(fill = "skyblue", color = "black", bi
ns = 100) +
    labs(x = "No. of Movie Ratings", y = "Count") +
    scale_x_log10() +
  ggtitle("Number of Ratings by User")
```

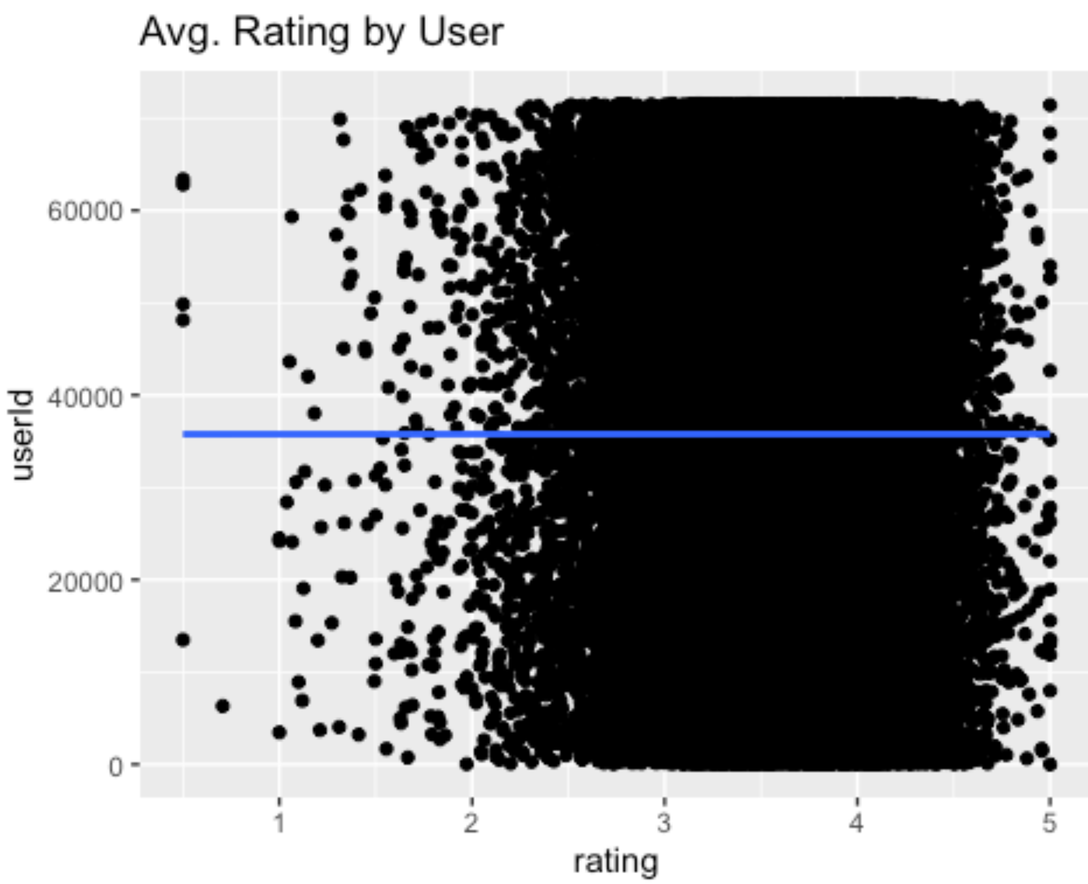



```
#By user, distribution of avg. rating
```

On average, we see ratings to be between 2.5 and 4.5, consistent with the summary metrics.

```
user_rating <- train %>%
  group_by(userId) %>%
  summarize(count = n(), rating = mean(rating)) %>%
  arrange(desc(count))

summary(user_rating)

##      userId          count            rating     
##  Min.   :    1   Min.   :  10.0   Min.   :0.500  
##  1st Qu.:17943   1st Qu.:  32.0   1st Qu.:3.357  
##  Median :35798   Median :  62.0   Median :3.635  
##  Mean   :35782   Mean   : 128.8   Mean   :3.614  
##  3rd Qu.:53620   3rd Qu.: 141.0   3rd Qu.:3.903  
##  Max.   :71567   Max.   :6616.0   Max.   :5.000  

train %>% group_by(userId) %>% summarize(rating = mean(rating)) %>%
  ggplot(aes(rating,userId)) +
  geom_point() +
```

```
  geom_smooth() +
ggtitle("Avg. Rating by User")

## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



IV. Modeling Methodology

As was stated at the beginning of this study, Netflix competition was based on lowering the value of the RMSE a.k.a. the Root Mean Square Error.

In simple terms, it measures the square root of the expected difference between the estimator and the parameter. It can be depicted as:

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

I first created the RMSE function in R and then ran the simplest version of the model, which is the average value of rating in the train data set.

```
#Modeling

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

mu <- mean(train_data$rating)
mu

## [1] 3.512456

naive_rmse <- RMSE(test_data$rating, mu)
naive_rmse

## [1] 1.060054

rmse_results <- tibble(method = "Just the average", RMSE = naive_rmse)
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.060054

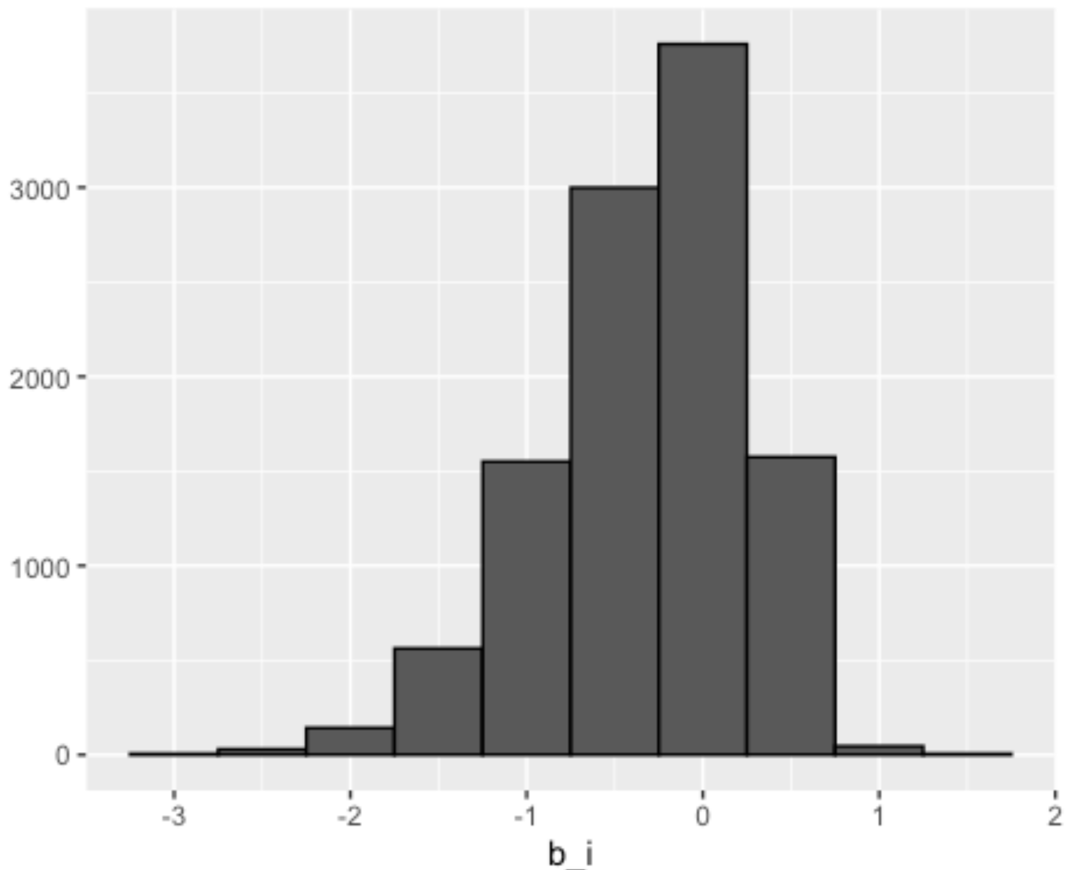
As a second step, I researched the movie bias to improve my prediction. The movie bias stemmed from the fact that there were some movies that were not rated by a lot of users and their outlying values could bias the estimates. I was able to see an improvement of ~12% in the RMSE value. The RMSE went down from 1.06 to 0.94.

```
#effect of movie ratings
# calculate RMSE of movie ranking effect

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

predicted_ratings <- test_data %>%
  left_join(movie_avgs, by='movieId') %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)
RMSE(predicted_ratings, test_data$rating)

## [1] 0.9429615

movie_bias <- RMSE(predicted_ratings, test_data$rating)
rmse_results <- bind_rows(rmse_results, tibble(method = "movie bias added", R
MSE = movie_bias))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0600537
movie bias added	0.9429615

As seen in the plot below, estimates vary when we take into account the movie effect.

```
qplot(b_i, data = movie_avgs, bins = 10, color = I("black"))
```



As a third model, I looked at the user effect since some users rated a lot of movies and some rated only a few. Controling for the user bias, I was able to get the RMSE down to 0.86, another ~9% improvement.

```
#effect of users

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

predicted_ratings <- test_data %>%
  left_join(movie_avgs, by='movieId') %>%
```

```
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
user_bias <- RMSE(predicted_ratings, test_data$rating)
rmse_results <- bind_rows(rmse_results, tibble(method = "movie and user bias
added", RMSE = user_bias))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0600537
movie bias added	0.9429615
movie and user bias added	0.8646843

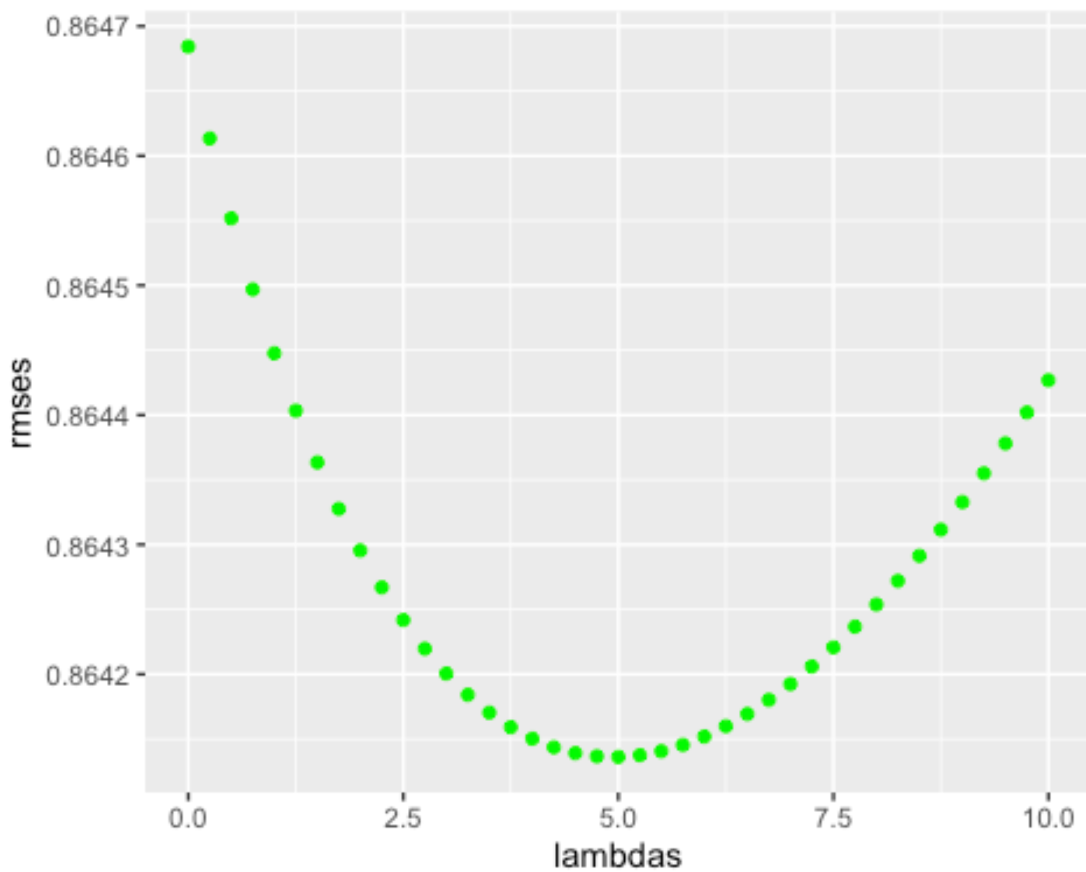
As a next model, I studied the Regularization Method. Regularization is used to avoid over-fitting due to the effect of large errors. This method reduces the impact of the magnitude of the independent variables by adding a penalty term to estimates on small sample sizes. For example, some movies were rated by only a few users. Therefore, larger estimates of b_i are likely to occur. Large errors can increase RMSE and we can use the regularization method to correct for this.

```
#Regularization method

lambdas <- seq(0,10,0.25)
rmses <- sapply(lambdas, function(a){
  mu <- mean(train_data$rating)

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

  b_u <- train_data %>%
    left_join(b_i, by='movieId') %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n() + a))

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

  return(RMSE(predicted_ratings, test_data$rating))
})

rmse_results <- bind_rows(rmse_results, tibble(method = "regularization metho
d", RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0600537
movie bias added	0.9429615
movie and user bias added	0.8646843
regularization method	0.8641362

Employing the regularization method gave us another slight improvement and the RMSE value ended up at 0.8641. Next, continue to add more variables such as the movie year and genre in this method to see if we can improve the estimates further.

```
qplot(lambdas, rmses, color = I("green"))
```




```
lambdas[which.min(rmses)]
## [1] 5
```

Regularization with additional attributes is shown below:

```
#Regularization with additional attributes

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

```r
rmses <- sapply(lambdas, function(a){
  mu <- mean(train_data$rating)

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

  b_u <- train_data %>%
    left_join(b_i, by='movieId') %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n() + a))

  b_y <- train_data %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    group_by(movie_year) %>%
    summarize(b_y = sum(rating - mu - b_i - b_u)/(n() + a))

  b_g <- train_data %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    left_join(b_y, by = 'movie_year') %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - b_i - b_u - b_y)/(n() + a))

  predicted_ratings <- test_data %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_y, by = 'movie_year') %>%
    left_join(b_g, by = 'genres') %>%
    mutate(pred = mu + b_i +  b_u + b_y + b_g) %>% .$pred

  return(RMSE(predicted_ratings, test_data$rating))
})

rmse_results <- bind_rows(rmse_results,
tibble(method = "regularization method enhanced with year and genre", RMSE =
min(rmses)))
rmse_results %>% knitr::kable()
```

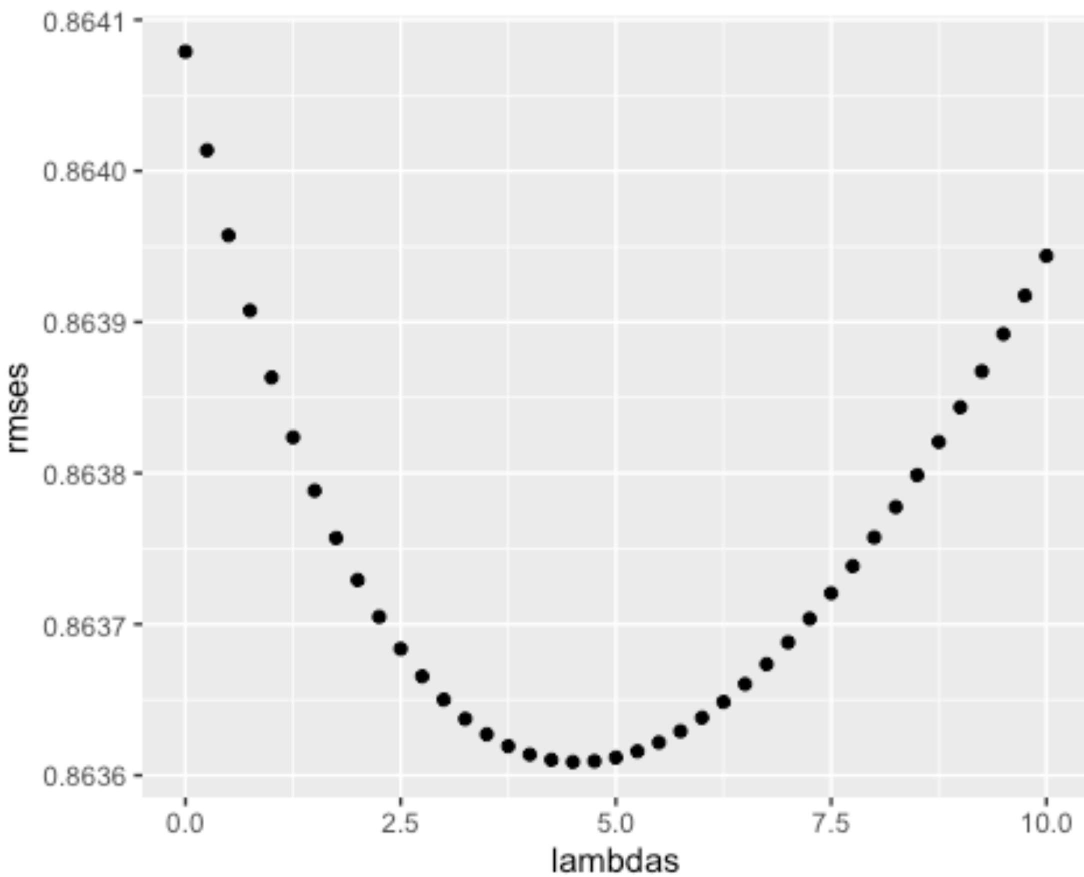
method	RMSE
Just the average	1.0600537
movie bias added	0.9429615
movie and user bias added	0.8646843
regularization method	0.8641362
regularization method enhanced with year and genre	0.8636089

V. Validation and Conclusion

Including the movie year and genre reduced the RMSE to 0.8636. We further validate this result using the validation dataset and achieve an RMSE of 0.8388.

While I was able to reduce the RMSE by employing a series of methods, I am surprised by the validation result being even lower. Usually, validation results are less favorable than training results. I will continue to explore this question in my next project for this course.

```r
# Compute new predictions using the optimal lambda
qplot(lambdas, rmses)
```




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

#Validation results

mu <- mean(validation$rating)

b_title <- validation %>%
  group_by(movieId) %>%
```

```
  summarize(b_title = sum(rating - mu)/(n() + a))

b_user <- validation %>%
  left_join(b_title, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_user = sum(rating - b_title - mu)/(n() + a))

predicted_ratings <- validation %>%
  left_join(b_title, by = "movieId") %>%
  left_join(b_user, by = "userId") %>%
  mutate(pred = mu + b_title +  b_user) %>% .$pred

RMSE(predicted_ratings, validation$rating)

## [1] 0.8388471
```

REFERENCES:

1. https://rafalab.github.io/dsbook/large-datasets.html
2. https://rmarkdown.rstudio.com/articles_intro.html
3. https://grouplens.org/datasets/movielens/10m/