MovieLens Capstone Project

Jarred Priester

11/21/2021

1. Overview

- 1.1 description of the dataset
- 1.2 goal of the project
- 1.3 steps to achieve the goal

2. Analysis

- 2.1 downloading the data
- 2.2 data cleaning
- 2.3 data exploration
- 2.4 visualization
- 2.5 models

3. Results

- 3.1 results
- 3.2 brief thoughts about the results

4. Conclusion

- 4.1 summary
- 4.2 limitations
- 4.3 future work

1. Overview

1.1 The dataset

In 2006 Netflix put out an award of $1 million dollars for any individual/team that could improve Netflix's movie recommendation system by 10% or more. This project will be replicating that competition by using the MovieLens dataset. This dataset consist of movies, users and how they rated the movies. Not every user has rated each movie so the dataset is quite sparse.

1.2 Goal of the project

The goal of the project is to predict how each user would rate each movie. The Netflix competition used root means squared error (RMSE) as the metric to measure accuracy of the predictions and this project will be using the same method to measure accuracy. The goal is to achieve an error of .8649 or less.

1.3 Steps to achieve the goal

In order to achieve this goal we start by downloading the data that has been provided. We will clean and explore the data. We will then split the edx set into a training and test set. We will build a model using a baseline of the average of every movie. From there we will add effects from features in the data set. Once we have trained this model we will run it on the validation set, which will get us our final RMSE.

2. Analysis

2.1 Downloading the data

data provided from Harvard for this capstone project:

```
############################################################
# 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 3.3.3     v purrr   0.3.4
## v tibble  3.1.1     v dplyr   1.0.5
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     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
```

```
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
```

```
library(tidyverse)
library(caret)
library(data.table)
library(dplyr)
library(recommenderlab)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loading required package: arules
```

```
##
## Attaching package: 'arules'
```

```
## The following object is masked from 'package:dplyr':
##
##     recode
```

```
## The following objects are masked from 'package:base':
##
##     abbreviate, write
```

```
## Loading required package: proxy
```

```
##
## Attaching package: 'proxy'
```

```
## The following object is masked from 'package:Matrix':
##
##     as.matrix

## The following objects are masked from 'package:stats':
##
##     as.dist, dist

## The following object is masked from 'package:base':
##
##     as.matrix

## Loading required package: registry

## Registered S3 methods overwritten by 'registry':
##   method               from
##   print.registry_field proxy
##   print.registry_entry proxy

##
## Attaching package: 'recommenderlab'

## The following objects are masked from 'package:caret':
##
##     MAE, RMSE
```

```
# 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
```

```
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)
```

2.2 Data cleaning

Now that we have the data downloaded. Let's take a quick look at how the dataset is structured and what features or columns we have to work with.

```
#quick preview of the data
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
```

```
#the edx dataset dimensions
dim(edx)
```

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

```
#the edx dataset feature names
names(edx)
```

```
## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

Title and year are included in the same column. We ideally would like to have the year separate to make it tidy and easier to use. So next we will abstract the year by adding a column of year movie was released.

```
edx <- edx %>%
  mutate(released = str_extract(title,"(\\d{4})"))

validation <- validation %>%
  mutate(released = str_extract(title,"(\\d{4})"))
```

the timestamp feature is in a format that is not easy to read. Let's convert the timestamp into dates

```
edx$timestamp <- as.POSIXct(edx$timestamp, origin="1970-01-01")

validation$timestamp <- as.POSIXct(validation$timestamp, origin="1970-01-01")
```

Our new timestamp gives us a readable date. Would like to have the month abstracted in order to further analyze, so let's add a column with the month of the rating

```
edx <- edx %>%
  mutate(month = months.Date(timestamp))

validation <- validation %>%
  mutate(month = months.Date(timestamp))
```

2.3 Data exploration

In this section we will be taking a look at the data and try to find some key insights:

How many zeros were given as ratings in the edx dataset?

```
sum(edx$rating == 0)
```

```
## [1] 0
```

How many threes were given as ratings in the edx dataset?

```
sum(edx$rating == 3)
```

```
## [1] 2121240
```

What percentage of the ratings were a perfect rating?

```
mean(edx$rating == 5)
```

```
## [1] 0.1544562
```

How many different movies are in the edx dataset?

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

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

How many different users are in the edx dataset?

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

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

Which movie has the greatest number of ratings?

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

```
## `summarise()` has grouped output by 'movieId'. You can override using the `.groups` argument.
```

```
## # 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
```

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

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

```
## # A tibble: 10 x 2
##    rating   count
##     <dbl>   <int>
##  1    4   2588430
##  2    3   2121240
##  3    5   1390114
##  4    3.5  791624
##  5    2    711422
```

```
##  6    4.5  526736
##  7    1    345679
##  8    2.5  333010
##  9    1.5  106426
## 10    0.5   85374
```

table of distribution of movies by release year

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

```
## # A tibble: 107 x 2
##    released  count
##    <chr>     <int>
##  1 1995     786762
##  2 1994     671298
##  3 1996     590238
##  4 1999     486936
##  5 1993     481184
##  6 1997     428185
##  7 1998     400605
##  8 2000     387196
##  9 2001     316794
## 10 2002     272127
## # ... with 97 more rows
```

table of distribution of movies by month rated

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

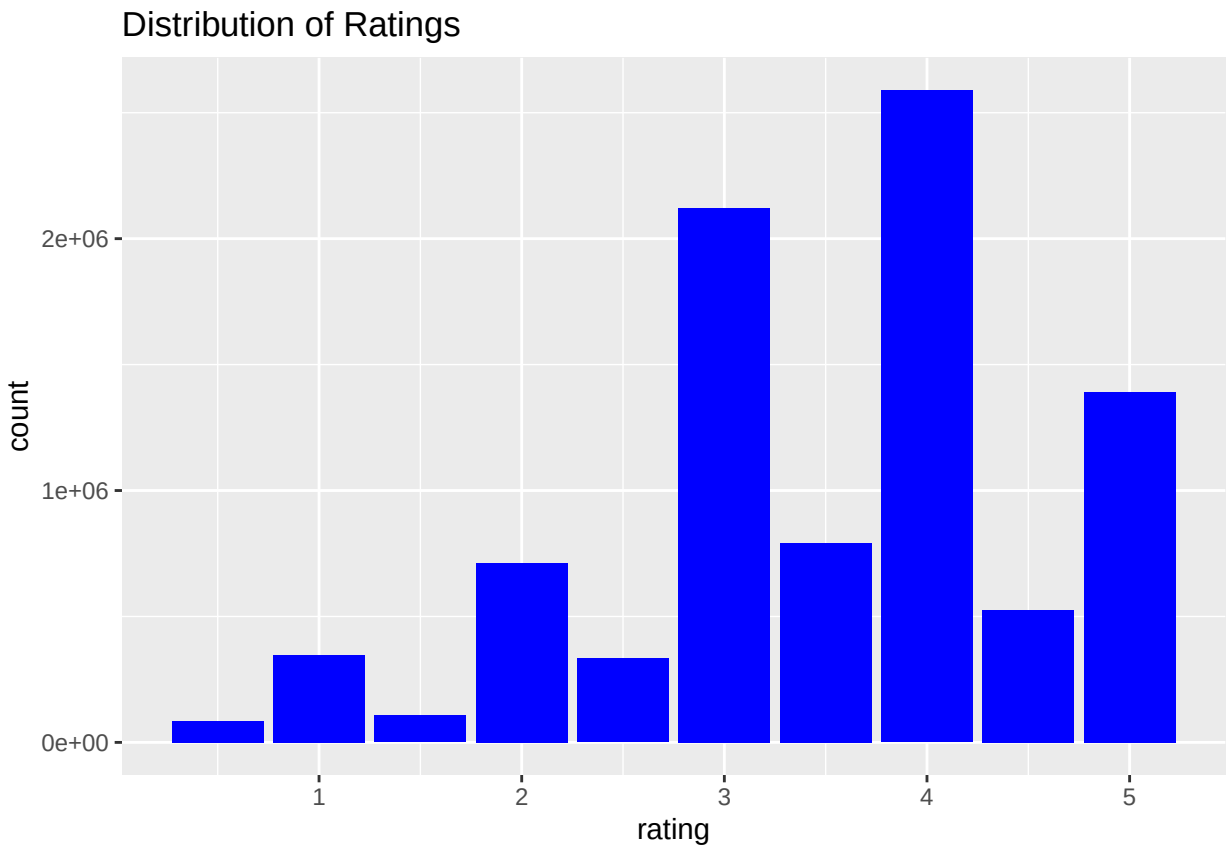
```
## # A tibble: 12 x 2
##    month      count
##    <chr>      <int>
##  1 November  978075
##  2 December  900202
##  3 October   827088
##  4 July      805620
##  5 January   753912
##  6 June      747818
##  7 March     746452
##  8 August    720414
##  9 May       667229
## 10 April     666945
## 11 February  615023
## 12 September 571277
```

2.4 Visualization

In this section we will look at a few graphs

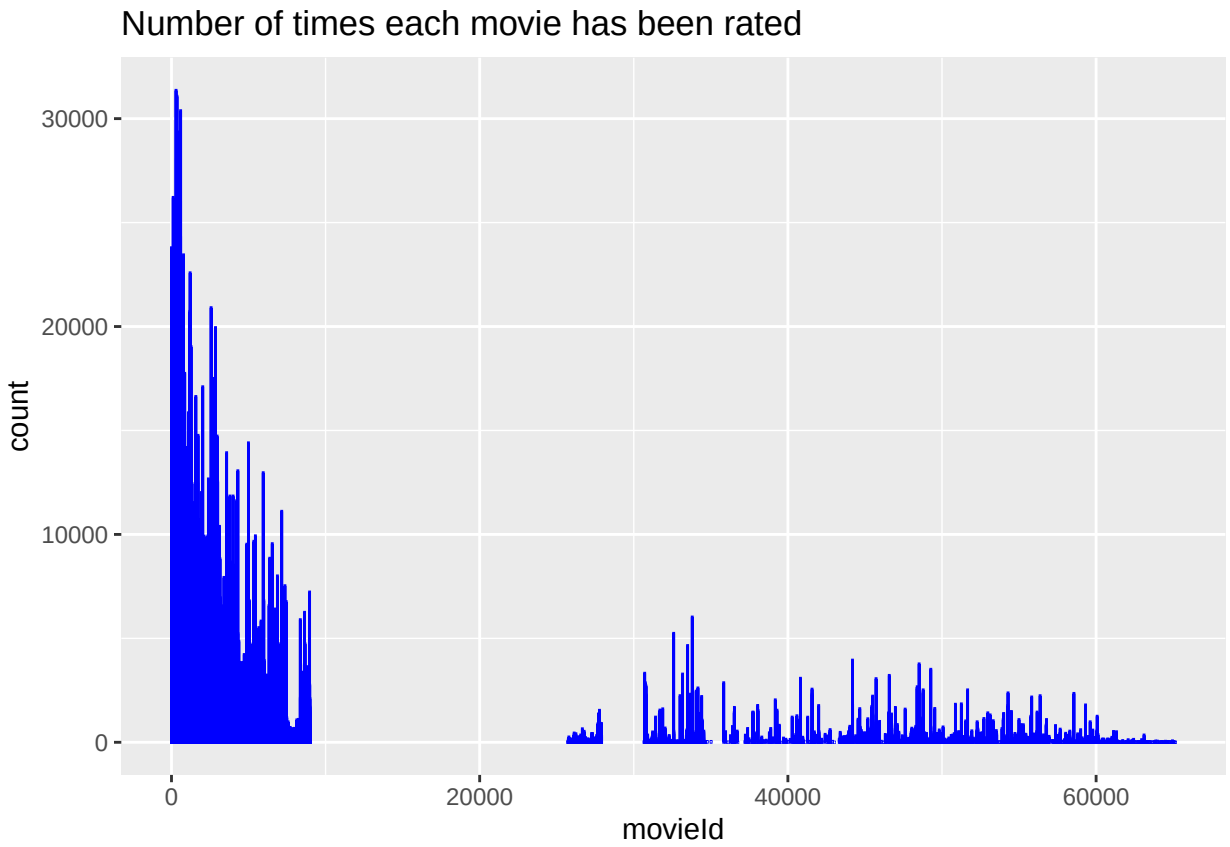
In general, half star ratings are less common than whole star ratings

```
edx %>%
  group_by(rating) %>%
  ggplot(aes(rating)) +
  geom_bar(fill = "blue") +
  ggtitle("Distribution of Ratings") +
  theme_gray()
```



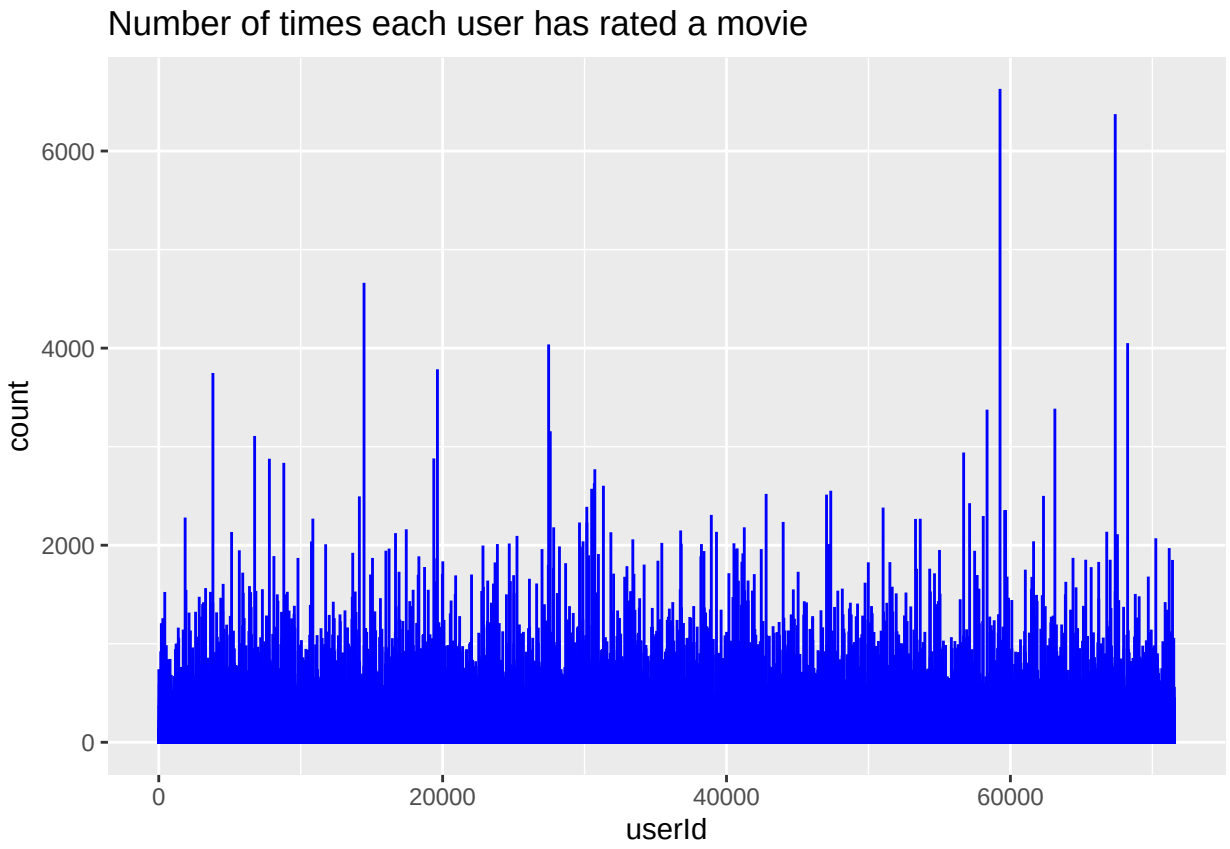
Here is a bar graph of movies and how many times they have been rated. You can see that some have been rated many times while most have very few ratings.

```
edx %>%
  ggplot(aes(movieId)) +
  geom_bar(color = "blue") +
  ggtitle("Number of times each movie has been rated") +
  theme_gray()
```



Here is a bar graph of users and how many times they have rated movies. You can see that it is a little more evenly distributed than the movies. Some users have rated a lot of movies while some users have only rated a few movies.

```
edx %>%
  ggplot(aes(userId)) +
  geom_bar(color = "blue") +
  ggtitle("Number of times each user has rated a movie") +
  theme_gray()
```



2.5 Models

Before we run any models, first we split the edx dataset into training and testing sets

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

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

```
edx_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
edx_training <- edx[-edx_index,]
edx_test <- edx[edx_index,]
```

Now let's make sure userId and movieId in validation set are also in edx_training set so that we do not generate N/As.

```
temp <- validation

temp2 <- temp %>%
  semi_join(edx_training, by = 'movieId') %>%
  semi_join(edx_training, by = 'userId')

# Add rows removed from temp set back into edx_training set
removed <- anti_join(temp, temp2)
```

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

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

rm(temp,temp2,removed)
```

Now let's make sure userId and movieId in edx_test set are also in edx_training set so that we do not generate N/As.

```
temp <- edx_test

temp2 <- temp %>%
  semi_join(edx_training, by = 'movieId') %>%
  semi_join(edx_training, by = 'userId')

# Add rows removed from temp set back into edx_training set
removed <- anti_join(temp, temp2)
```

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

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

rm(temp,temp2,removed)
```

Checking the dimensions of the training set and the test set to make sure they split 80/20

```
dim(edx_training)
```

```
## [1] 7200087       8
```

```
dim(edx_test)
```

```
## [1] 1800012       8
```

Now checking to see if there are any N/A values in the rating column

```
sum(is.na(edx_training$rating))
```

```
## [1] 0
```

```
sum(is.na(edx_test$rating))
```

```
## [1] 0
```

Residual mean squared error (RMSE) is what we will be using on the test set and the validation set to test our predictions accuracy. RMSE =

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

Here we have $y_{u,i}$ as the rating for movie i from user u and denote our prediction with with N being the number of user/movie combinations and the sum occurring over all these combinations

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

First model

We are going to take the average rating of the training set and use that as a base line to compare other models to.

$$y_{u,i} = \mu + \varepsilon_{u,i}$$

Where μ represents the average of all movies and $\varepsilon_{u,i}$ represents the random variations

```
mu_hat <- mean(edx_training$rating)

#Here we apply the rating average to the test set to get #our error lost
naive_rmse <- RMSE(edx_test$rating, mu_hat)

#creating a data frame to store the method and results
rmse_results <- data.frame(method = "average rating", RMSE = naive_rmse)

#viewing results table
rmse_results
```

```
##           method     RMSE
## 1 average rating 1.059909
```

Second model

For the second model we will add the movie effect to the first model. To get the movie effect we will generate the least squared estimate by obtaining the average of the rating minus the average rating for each movie.

$$y_{u,i} = \mu + b_i + \varepsilon_{u,i}$$

Here b_i represents the average rating for movie i.

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

#adding the movie effect to the rating average for each movie to generate our
#predictions for each movie
movie_effect <- edx_test %>%
  left_join(movie_avgs, by='movieId') %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)

#movie effect RMSE
movie_effect_rmse <- RMSE(edx_test$rating,movie_effect)

#adding movie effect RMSE to the results table
rmse_results <- rbind(rmse_results,
```

```
                        c(method = "movie effect",
                          RMSE = movie_effect_rmse))

#viewing the results table
rmse_results
```

```
##           method              RMSE
## 1 average rating  1.05990935828899
## 2   movie effect 0.943737550921437
```

Third model

For the third model we will be adding the user effect to the second model.

$$y_{u,i} = \mu + b_i + b_u + \varepsilon_{u,i}$$

Here b_u represents the average rating for user u.

```
#Here we will be generating the least squared estimate by #obtaining the average of the rating minus th
user_avgs <- edx_training %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

#adding the movie effect and user effect to the rating average
#for each movie to generate our predictions for each movie
movie_user_effect <- edx_test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

#movie and user effect RMSE
movie_user_effect_rmse <- RMSE(edx_test$rating, movie_user_effect)

##adding movie effect RMSE to the results table
rmse_results <- rbind(rmse_results,
                      c(method = "movie user effect",
                      RMSE = movie_user_effect_rmse))

#viewing the results table
rmse_results
```

```
##              method              RMSE
## 1    average rating  1.05990935828899
## 2      movie effect 0.943737550921437
## 3 movie user effect 0.865931300929012
```

Fourth model

For the fourth model we will add the year released effect to the third model.

$$y_{u,i} = \mu + b_i + b_u + b_y + \varepsilon_{u,i}$$

Here b_y represents the average rating of year y.

```
#Here we will be generating the least squared estimate by obtaining the
#average of the rating minus the average rating minus the movie effect
#minus the user effect for each movie
year_avg <- edx_training %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by="userId") %>%
  group_by(released) %>%
  summarize(b_y = mean(rating - mu - b_i - b_u))

#adding the movie, user and year effect to the rating average
#for each movie to generate our predictions for each movie
movie_user_year_effect <- edx_test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(year_avg, by='released') %>%
  mutate(pred = mu + b_i + b_u + b_y) %>%
  pull(pred)

#Movie, User and Year effect RMSE
movie_user_year_effect_rmse <- RMSE(edx_test$rating, movie_user_year_effect)

#adding movie effect RMSE to the results table
rmse_results <- rbind(rmse_results,
                      c(method = "movie user year effect",
                      RMSE = movie_user_year_effect_rmse  ))

#viewing results table
rmse_results
```

```
##                    method              RMSE
## 1         average rating  1.05990935828899
## 2           movie effect 0.943737550921437
## 3      movie user effect 0.865931300929012
## 4 movie user year effect  0.86561150368403
```

Fifth Model

For the fifth model we will be adding the month effect to the fourth model.

$$y_{u,i} = \mu + b_i + b_u + b_y + b_m + \varepsilon_{u,i}$$

Here b_m represents the average rating of month m.

```
#Here we will be generating the least squared estimate by #obtaining the average of the rating minus th
month_avg <- edx_training %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(year_avg, by='released') %>%
  group_by(month) %>%
  summarize(b_m = mean(rating - mu - b_i - b_u - b_y))
```

```
#adding the movie, user, year and month effect to the #rating average for each movie to generate our pr
movie_user_year_month_effect <- edx_test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(year_avg, by='released') %>%
  left_join(month_avg, by='month') %>%
  mutate(pred = mu + b_i + b_u + b_y + b_m) %>%
  pull(pred)

#movie, user, year, month effect RMSE
movie_user_year_month_effect_rmse <-
  RMSE(edx_test$rating,movie_user_year_month_effect)

#adding movie effect RMSE to the results table
rmse_results <- rbind(rmse_results,
                      c(method = "movie user year month effect",
                        RMSE = movie_user_year_month_effect_rmse))

#viewing results table
rmse_results
```

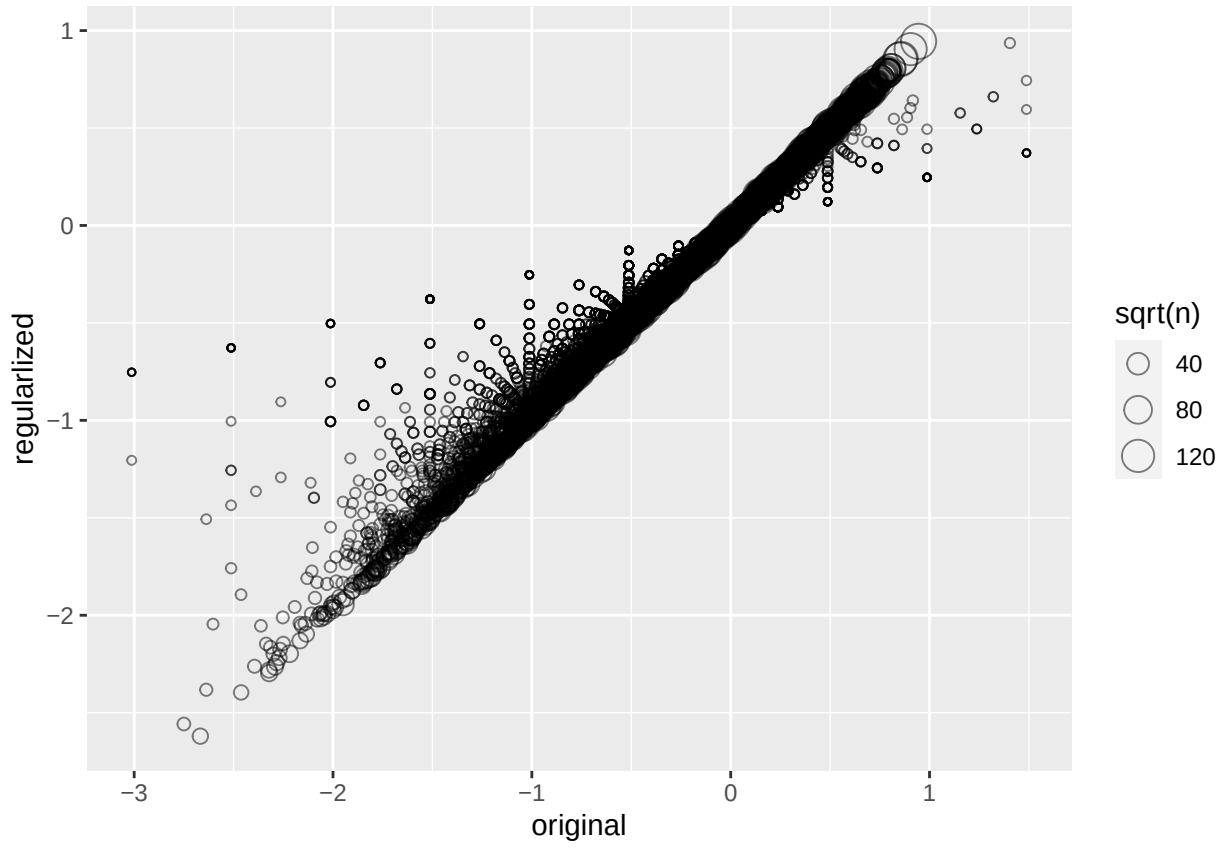
```
##                          method              RMSE
## 1                average rating  1.05990935828899
## 2                 movie effect 0.943737550921437
## 3            movie user effect 0.865931300929012
## 4       movie user year effect  0.86561150368403
## 5 movie user year month effect 0.865603862861746
```

Sixth Model

Since some movies have over a thousand ratings while other movies only have one rating, we will be regularizing the least squared estimate in order to penalize movies with few ratings

$$\sum_{u,i}(y_{u,i} - \mu - b_i)^2 + \lambda \sum_i b_i^2$$

```
#regularized least squared estimates
lambda <- 3
mu <- mean(edx_training$rating)
movie_reg_avgs <- edx_training %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())

#graph of the results of the movie effect regularized
tibble(original = movie_avgs$b_i,
       regularlized = movie_reg_avgs$b_i,
       n = movie_reg_avgs$n_i) %>%
  ggplot(aes(original, regularlized, size=sqrt(n))) +
  geom_point(shape=1, alpha=0.5)
```




```
#adding the regularized movie effect to the rating average for each movie
#to generate our predictions for each movie
movie_effect_reg <- edx_test %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)

#Movie effect regularized RMSE
movie_effect_reg_rmse <- RMSE(edx_test$rating, movie_effect_reg)

#adding regularized movie effect RMSE to the results table
rmse_results <- rbind(rmse_results,
                      c(method = "movie effect reg",
                        RMSE = movie_effect_reg_rmse))

#viewing results table
rmse_results
```

```
##                          method              RMSE
## 1                average rating  1.05990935828899
## 2                 movie effect 0.943737550921437
## 3            movie user effect 0.865931300929012
## 4       movie user year effect  0.86561150368403
## 5 movie user year month effect 0.865603862861746
## 6             movie effect reg 0.943676637247946
```

Seventh Model

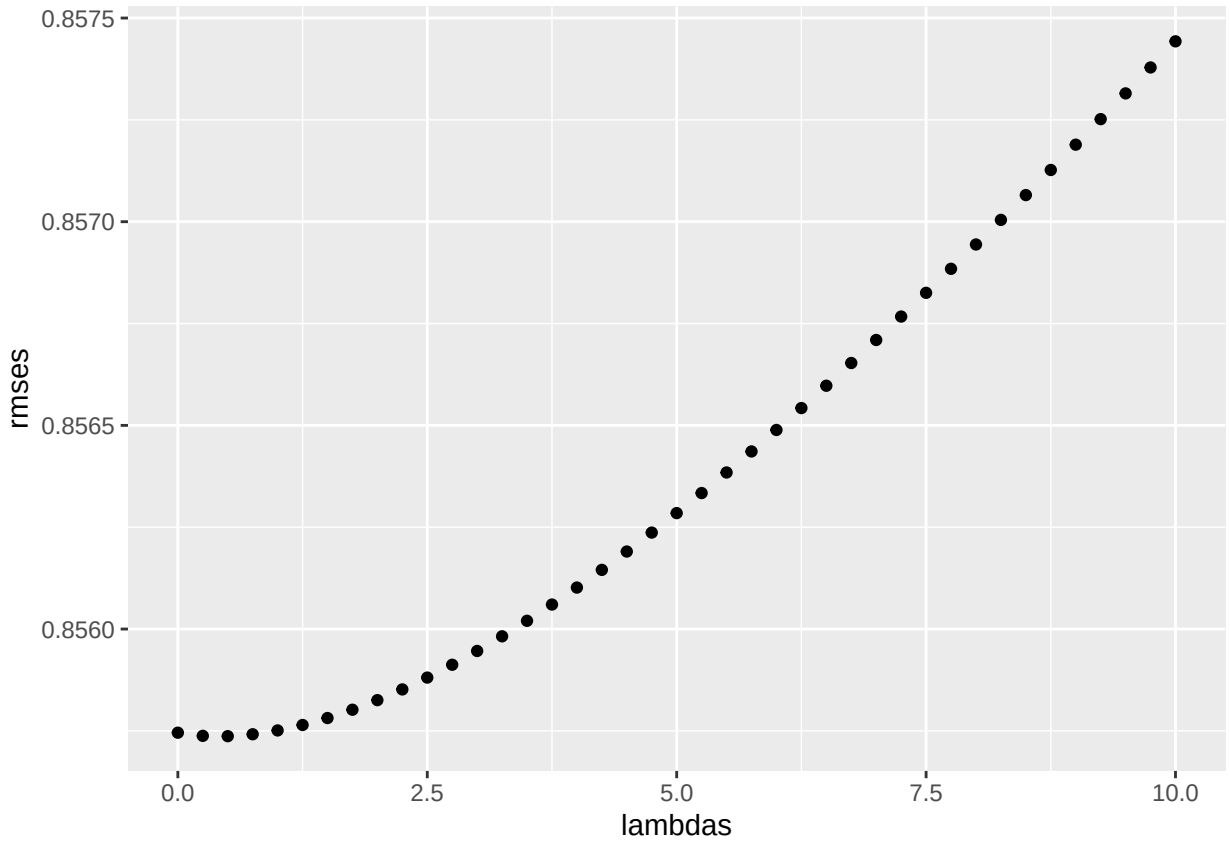
Since the lambda is a tuning parameter, we will run a function in order to find the lambada that gives us the lowest RMSE

$$\hat{b}_i(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n} (y_{u,i} - \hat{u})$$

Once we find the lambda that gives us the lowest RMSE we will apply it to the regularized model

$$\sum_{u,i} (y_{u,i} - \mu - b_i - b_u)^2 + \lambda \sum_i b_i^2$$

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

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

  mu <- mean(edx_training$rating)

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

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

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

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

#plot showing the RMSE result for each lambda
qplot(lambdas, rmses)
```




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

mu <- mean(edx_training$rating)

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

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

#Movie and user regularized effect predictions
movie_user_reg_avgs <-
  edx_test %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

#Movie and user regularized RMSE
movie_user_reg_rmse <- RMSE(edx_test$rating, movie_user_reg_avgs)

#adding the regularized movie, user effect RMSE to the results table
rmse_results <- rbind(rmse_results,
```

```
                      c(method = "movie user effect reg",
                        RMSE = movie_user_reg_rmse))

#viewing results table
rmse_results
```

```
##                          method              RMSE
## 1               average rating  1.05990935828899
## 2                 movie effect 0.943737550921437
## 3            movie user effect 0.865931300929012
## 4       movie user year effect  0.86561150368403
## 5 movie user year month effect 0.865603862861746
## 6              movie effect reg 0.943676637247946
## 7         movie user effect reg 0.865754152011791
```

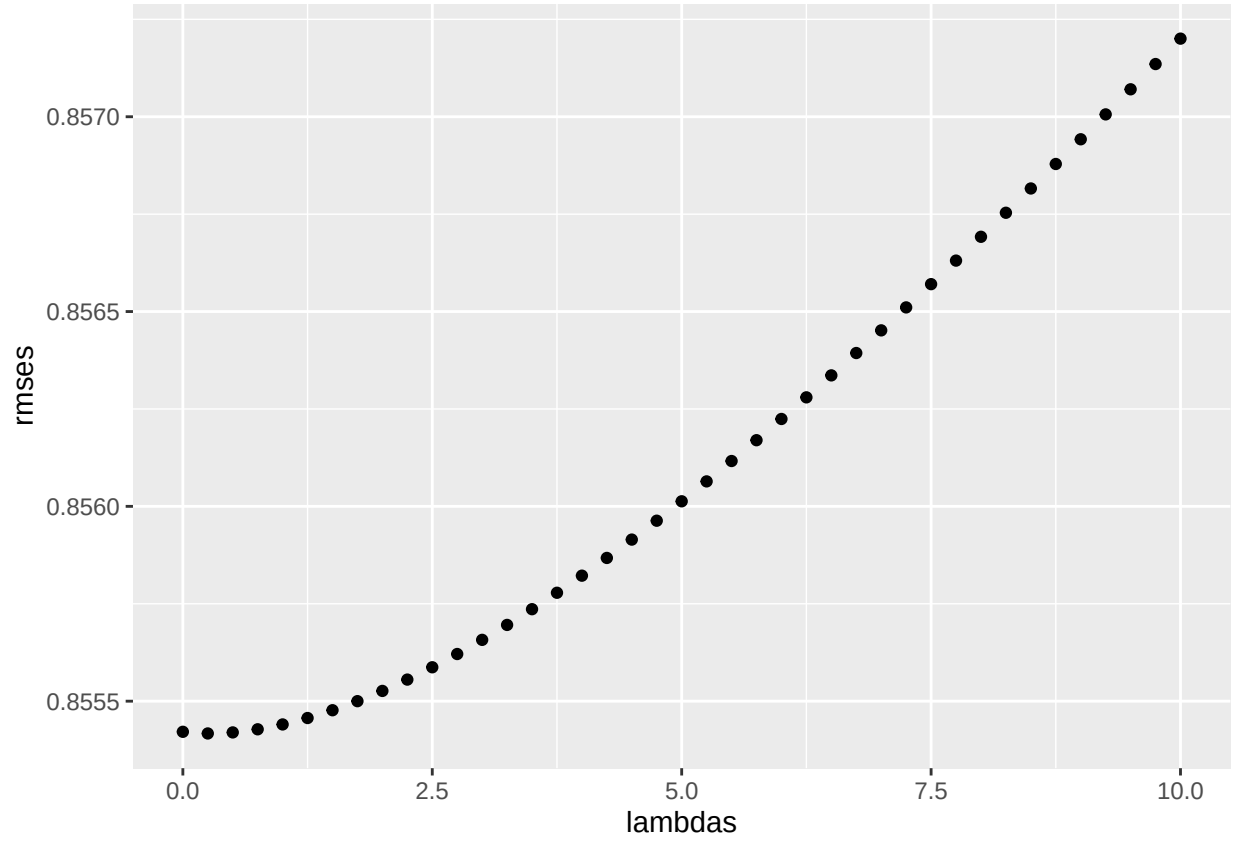
Eighth Model

We will now be running the same model as the previous model but this time we will add the year released effect to the model.

$$\sum_{u,i}(y_{u,i} - \mu - b_i - b_u - b_y)^2 + \lambda \sum_i b_i^2$$

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

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

  mu <- mean(edx_training$rating)

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

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

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

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

  return(RMSE(edx_training$rating, predicted_ratings))
```

```
})

#plot showing the RMSE result for each lambda
qplot(lambdas, rmses)
```




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

mu <- mean(edx_training$rating)

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

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

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

#movie, user, year regularized predictions
```

```
movie_user_year_reg_avgs <-
  edx_test %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_y, by = "released") %>%
  mutate(pred = mu + b_i + b_u + b_y) %>%
  pull(pred)

#movie, user, year regularized RMSE
movie_user_year_reg_rmse <- RMSE(edx_test$rating, movie_user_year_reg_avgs)

#adding the regularized movie, user, year RMSE result to the result table
rmse_results <- rbind(rmse_results,
                      c(method = "movie user year reg",
                        RMSE = movie_user_year_reg_rmse))

#viewing the results table
rmse_results
```

```
##                          method              RMSE
## 1                average rating  1.05990935828899
## 2                 movie effect 0.943737550921437
## 3            movie user effect 0.865931300929012
## 4       movie user year effect  0.86561150368403
## 5 movie user year month effect 0.865603862861746
## 6             movie effect reg 0.943676637247946
## 7        movie user effect reg 0.865754152011791
## 8          movie user year reg 0.865519596155414
```

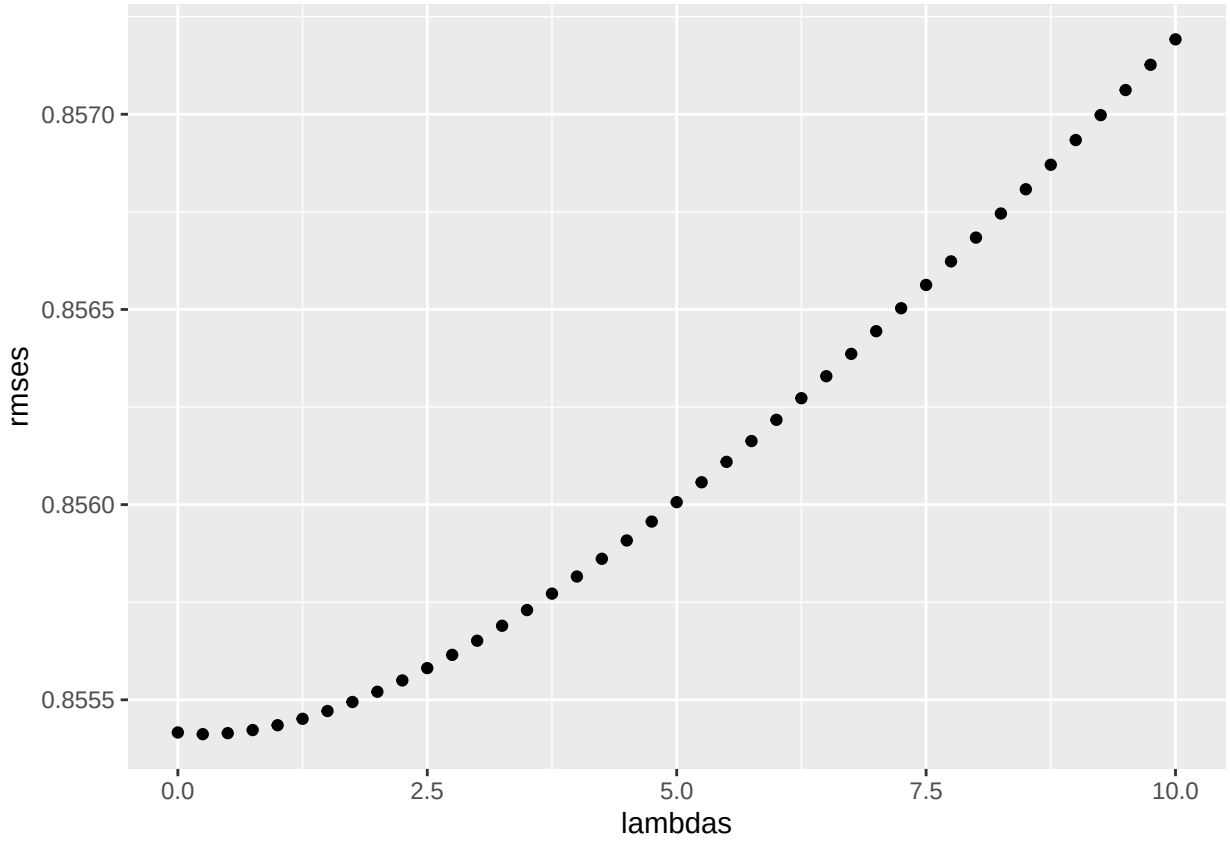
Ninth Model

We will now be running the same model as the previous model but this time we will add the month effect to the model.

$$\sum_{u,i}(y_{u,i} - \mu - b_i - b_u - b_y - b_m)^2 + \lambda \sum_i b_i^2$$

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

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

  mu <- mean(edx_training$rating)

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

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

  b_y <- edx_training %>%
```

```
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    group_by(released) %>%
    summarize(b_y = sum(rating - mu - b_i - b_u)/(n()+l))

  b_m <- edx_training %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    left_join(b_y, by='released') %>%
    group_by(month) %>%
    summarize(b_m = sum(rating - mu - b_i - b_u - b_y)/(n()+1))

  predicted_ratings <-
    edx_training %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_y, by = "released") %>%
    left_join(b_m, by = 'month') %>%
    mutate(pred = mu + b_i + b_u + b_y + b_m) %>%
    pull(pred)

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

#plot showing the RMSE result for each lambda
qplot(lambdas, rmses)
```




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

mu <- mean(edx_training$rating)

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

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

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

b_m <- edx_training %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_y, by = "released") %>%
  group_by(month) %>%
  summarize(b_m = sum(rating - mu - b_i - b_u - b_y)/(n()+l))
```

```
#movie, user, year, month regularized predictions
movie_user_year_month_reg_avgs <-
  edx_test %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_y, by = "released") %>%
  left_join(b_m, by = "month") %>%
  mutate(pred = mu + b_i + b_u + b_y + b_m) %>%
  pull(pred)

#movie, user, year, month regularized RMSE
movie_user_year_month_reg_rmse <-
  RMSE(edx_test$rating, movie_user_year_month_reg_avgs)

#adding the regularized movie, user, year, month RMSE result to the result table
rmse_results <- rbind(rmse_results,
                      c(method = "movie user year month reg",
                        RMSE = movie_user_year_month_reg_rmse))

#viewing RMSE result
rmse_results
```

```
##                          method              RMSE
## 1                average rating  1.05990935828899
## 2                 movie effect 0.943737550921437
## 3            movie user effect 0.865931300929012
## 4       movie user year effect  0.86561150368403
## 5 movie user year month effect 0.865603862861746
## 6             movie effect reg 0.943676637247946
## 7        movie user effect reg 0.865754152011791
## 8          movie user year reg 0.865519596155414
## 9    movie user year month reg 0.865511895575428
```

```
#now that we have trained and tested our models. Now we #will test it against the validation data set a
validation_pred <- validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_y, by = "released") %>%
  left_join(b_m, by = "month") %>%
  mutate(pred = mu + b_i + b_u + b_y + b_m) %>%
  pull(pred)

#validation RMSE
validation_rmse <- RMSE(validation$rating, validation_pred)

#adding the validation RMSE to the results table
rmse_results <- rbind(rmse_results,
                      c(method = "validation",
                        RMSE = validation_rmse))
```

3. Results

3.1 Final table

```
rmse_results
```

```
##                           method              RMSE
## 1                 average rating  1.05990935828899
## 2                   movie effect 0.943737550921437
## 3              movie user effect 0.865931300929012
## 4         movie user year effect  0.86561150368403
## 5  movie user year month effect 0.865603862861746
## 6                movie effect reg 0.943676637247946
## 7          movie user effect reg 0.865754152011791
## 8            movie user year reg 0.865519596155414
## 9     movie user year month reg 0.865511895575428
## 10                    validation 0.866006835748326
```

3.2 Brief thoughts about results

The final result from running the model on the validation set is a RMSE of .8660. This did not reach the original goal of .8649. In order for this approach to reach .8649 there will need to be more features added to the equations. Features that have significant amount of variability in rating averages.

4. Conclusion

4.1 Summary

In summary, we were given data from the movielens dataset. We cleaned and explored the data. Split the data into training and testing sets. Applied a baseline model of the average of every movie. From there we added an effect from the features from the dataset. We ran the best model that we had from training on the validation set that was the final hold off set, and produced a RMSE of .8660.

4.2 Limitations

The biggest limitation that I came across was that I was unable to produce a matrix of movies and users with their ratings due to not having enough memory space on my laptop. I would have liked to apply a PCA analysis and a matrix factorization using the recommenderlab library. I do think that would have a chance at reaching the goal of .8649 or less.

4.3 Future Work

I think the next step would be to use the model that we have laid out as a baseline and add predictions from a matrix factorization or another type clustering analysis that would utilize the similarities and group movies into clusters, basically creating a more accurate version of genres. Another step might be to use an ensemble of different approaches to try and optimize the RMSE.