MovieLens Capstone

Avery Clark

January 1, 2020

Executive Summary

In this analysis, I used machine learning methods to build prediction models designed to predict what a user will rate a movie as a foundation for a recommendation system.

In this section I'll describe the dataset and summarize the goal of the project and key steps that were performed.

I analyzed the MovieLens 10M database and used it to attempt to build a machine learning algorithm that can predict what movies users would like to watch with high accuracy. These predictions will be trained on one dataset and tested on a separate dataset, where they will hopefully come very close to predicting how many stars (on a 0.5 to 5 star scale) a user will rate a movie.

To win the grand prize of $1 million from the Netflix challenge, a participating team had to get to a residual mean square error (RMSE) of about 0.857.

You can read more about it here: http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/

My goal was to build a prediction model with a RMSE of less than 0.8649. I surpassed that goal.

The RMSE (Residual Mean Square Error) is the standard deviation of the prediction errors (residuals).

In other words, it's the average of how far the predictions deviate from what they are trying to predict.

The lower the RMSE, the more accurate the algorithm's predictions are.

I split the data into a training set (90% of data) to train the prediction models and a testing set (10% of data) to test the accuracy of the prediction model.

After running five prediction models, the lowest Residual Mean Square Error (RMSE) obtained was 0.8644501, which accomplishes the goal of reaching lower than 0.8649.

The most effective prediction model was "Regularized Movie + User + Genre Effect Model", where I used the biases per movie, per user, and per genre of the reviews in the training set and then regularized (or "rubber-banded") the results, penalizing biases of movies/users/genres with low review counts by pulling them toward the dataset average.

This report contains four sections: Executive Summary, Analysis, Results, and Conclusion.

Executive Summary describes the dataset and summarizes the goal of the project and key steps that were performed.

Analysis explains the process and techniques used, such as data cleaning, data exploration and visualization, any insights gained, and the modeling approach.

Results presents the modeling results and discusses the model performance.

Conclusion gives a brief summary of the report, its limitations and future work.

Thank you for taking the time to look at this report. I hope that you will run this code by stepping through (by pressing Ctrl + Enter) as I'm explaining it.

Analysis

I'd like to start this analysis off by asking: How important is it that your next recommendation be something you really like? Netflix thought it was so important that they happily offered $1 million for a 10% increase in the accuracy of their recommendations.

You can read more about it here: http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/

Below, I'll be analyzing the MovieLens 10M database and attempt to build a machine learning algorithm that can predict what movies users would like to watch with high accuracy. These predictions will be trained on one dataset and tested on a separate dataset, where they will hopefully come very close to predicting how many stars (on a 0.5 to 5 star scale) a user will rate a movie.

To win the grand prize of $1 million from the Netflix challenge, a participating team had to get to a residual mean square error (RMSE) of about 0.857.

My goal is to build a prediction model with a RMSE of less than 0.8649.

The RMSE (Residual Mean Square Error) is the standard deviation of the prediction errors (residuals).

In other words, it's the average of how far the predictions deviate from what they are trying to predict.

The lower the RMSE, the more accurate the algorithm's predictions are.

This was run using RStudio Version 1.1.463 and R version 3.6.2 (Dark and Stormy Night) from https://www.r-project.org/

In this section, I'll explain the process and techniques used, such as data cleaning, data exploration and visualization, any insights gained, and the modeling approach. You'll see these models in action in the Results section.

90% of the data was designated for training the prediction model and 10% of the data was reserved for testing the accuracy of that model's predictions.

A simple way of thinking about this is that the model (or algorithm) will learn about the data by taking in different factors and will make a prediction of what star rating (on a 0.5 to 5 star scale) a user will rank a movie based on those factors. Different approaches will have the model/algorithm using the factors given to it in different ways to make predictions.

The model/algorithm decides to predict a review rating "Y" based on factors "A", "B", and "C" (or more). Then the model/algorithm is exposed to the testing dataset to see if what it predicts as the review rating "Y" (based on the factors in the new dataset "A", "B", and "C") is actually that accurate or not. Then from the results we can compute our RMSE (Residual Mean Square Error). This is how we test the model's accuracy.

The RMSE (Residual Mean Square Error) is the standard deviation of the prediction errors (residuals). In simpler terms, it's the average of how far the predictions deviate from what they are trying to predict (how far off the mark our model's predictions are). The lower the RMSE, the more accurate the model's predictions are.

I hope that you will step through the code with me as I explain it.

You can run all of the code by clicking Run. You can run it line by line by pressing Ctrl + Enter on your keyboard. You can also highlight a section of code and run just that by clicking Run or pressing Ctrl + Enter on your keyboard.

Let's dig in!

First, we'll build our training set (edx set) and our validation set from the MovieLens 10M dataset by splitting the data up randomly.

The training set (edx set) will hold about 90% of our data, and the validation set will hold about 10%.

We will use the training set to train our machine learning algorithm and we will test its accuracy on the validation set.

To begin this, we'll install the packages that will give us the tools to analyze the data. Notice the if statements mean the packages will not install if you have them already.

Note: this might take a couple of minutes.

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

```
## Loading required package: caret

## Loading required package: lattice

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 3.6.2
```

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

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

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

```
## Loading required package: dotwhisker

## Warning: package 'dotwhisker' was built under R version 3.6.2
```

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

```
## Loading required package: tidyverse

## -- Attaching packages ---------------------------------------------------------------------------- t

## v tibble  2.1.3     v purrr   0.3.3
## v tidyr   1.0.0     v dplyr   0.8.3
## v readr   1.3.1     v stringr 1.4.0
## v tibble  2.1.3     v forcats 0.4.0

## -- Conflicts ------------------------------------------------------------------------------- tidyvers
## x dplyr::between()   masks data.table::between()
## x dplyr::filter()    masks stats::filter()
## x dplyr::first()     masks data.table::first()
## x dplyr::lag()       masks stats::lag()
## x dplyr::last()      masks data.table::last()
## x purrr::lift()      masks caret::lift()
## x purrr::transpose() masks data.table::transpose()
```

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

```
## Loading required package: rmarkdown

## Warning: package 'rmarkdown' was built under R version 3.6.2
```

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

```
## Loading required package: knitr
```

```
library(caret)
library(data.table)
library(dotwhisker)
library(tidyverse)
library(rmarkdown)
library(knitr)
```

```r
wd <- getwd()

# Uncomment and run the next
# line to see your working directory:
# wd

setwd(wd)

# You can change this by editing the file path instead
# of using "wd".


# Now we'll download the data.

# 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)

# Now we'll convert the data into the columns we will use.

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")
```

Now we'll split up the data into our training set and our validation set.

The training set will be 90% of the MovieLens data and the validation set will be 10%.

We'll set a random seed so the results can be reproduced by anyone else running this code.

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

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

```r
# 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 values in validation line
# up to the IDs contained in the edx set.

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

```r
# Add rows removed from validation set back into edx set.

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

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

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

# This will clean up memory by deleting things we no
# longer need.

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

And now that we've built our training set (edx set) and our validation set, let's look at the data we're working with.

First let's check if we have any missing data.

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

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

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

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

```r
# Now let's probe the data a little more.

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
##
##
##
```

```r
dim(edx)
```

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

```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
```

```r
tibble(edx)
```

```
## # A tibble: 9,000,055 x 1
##    edx$userId $movieId $rating $timestamp $title           $genres
##         <int>    <dbl>   <dbl>      <int> <chr>            <chr>
##  1          1      122       5  838985046 Boomerang (1992) Comedy|Romance
##  2          1      185       5  838983525 Net, The (1995)  Action|Crime|Th~
##  3          1      292       5  838983421 Outbreak (1995)  Action|Drama|Sc~
##  4          1      316       5  838983392 Stargate (1994)  Action|Adventur~
##  5          1      329       5  838983392 Star Trek: Gene~ Action|Adventur~
##  6          1      355       5  838984474 Flintstones, Th~ Children|Comedy~
##  7          1      356       5  838983653 Forrest Gump (1~ Comedy|Drama|Ro~
##  8          1      362       5  838984885 Jungle Book, Th~ Adventure|Child~
##  9          1      364       5  838983707 Lion King, The ~ Adventure|Anima~
## 10          1      370       5  838984596 Naked Gun 33 1/~ Action|Comedy
## # ... with 9,000,045 more rows
```

```r
# The training set has 9,000,055 entries (or rows)
# of movie reviews and 6 columns of details.

summary(validation)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18096   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.467e+08
##  Median :35768   Median : 1827   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4108   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53621   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:999999      Length:999999
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

```r
dim(validation)
```

```
## [1] 999999      6
```

```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
```

```r
tibble(validation)
```

```
## # A tibble: 999,999 x 1
##    validation$userId $movieId $rating $timestamp $title        $genres
##                <int>    <dbl>   <dbl>      <int> <chr>         <chr>
##  1                 1      231     5    838983392 Dumb & Dumbe~ Comedy
##  2                 1      480     5    838983653 Jurassic Par~ Action|Adve~
##  3                 1      586     5    838984068 Home Alone (~ Children|Co~
##  4                 2      151     3    868246450 Rob Roy (199~ Action|Dram~
##  5                 2      858     2    868245645 Godfather, T~ Crime|Drama
##  6                 2     1544     3    868245920 Lost World: ~ Action|Adve~
##  7                 3      590     3.5 1136075494 Dances with ~ Adventure|D~
##  8                 3     4995     4.5 1133571200 Beautiful Mi~ Drama|Myste~
##  9                 4       34     5    844416936 Babe (1995)   Children|Co~
## 10                 4      432     3    844417070 City Slicker~ Adventure|C~
## # ... with 999,989 more rows
```

As we can see, the validation set has 999,999 entries and the same 6 columns.

Thankfully, this data is already pretty clean. I won't have to go through a lot of effort looking for ways to fix errors or NAs from bad data entry or missing data.

Take note that the entries in the validation set ARE NOT in the training set (edx set), and vice versa. No entries are shared between the sets.

We'll train our model/algorithm on the training set and test its accuracy on the validation set.

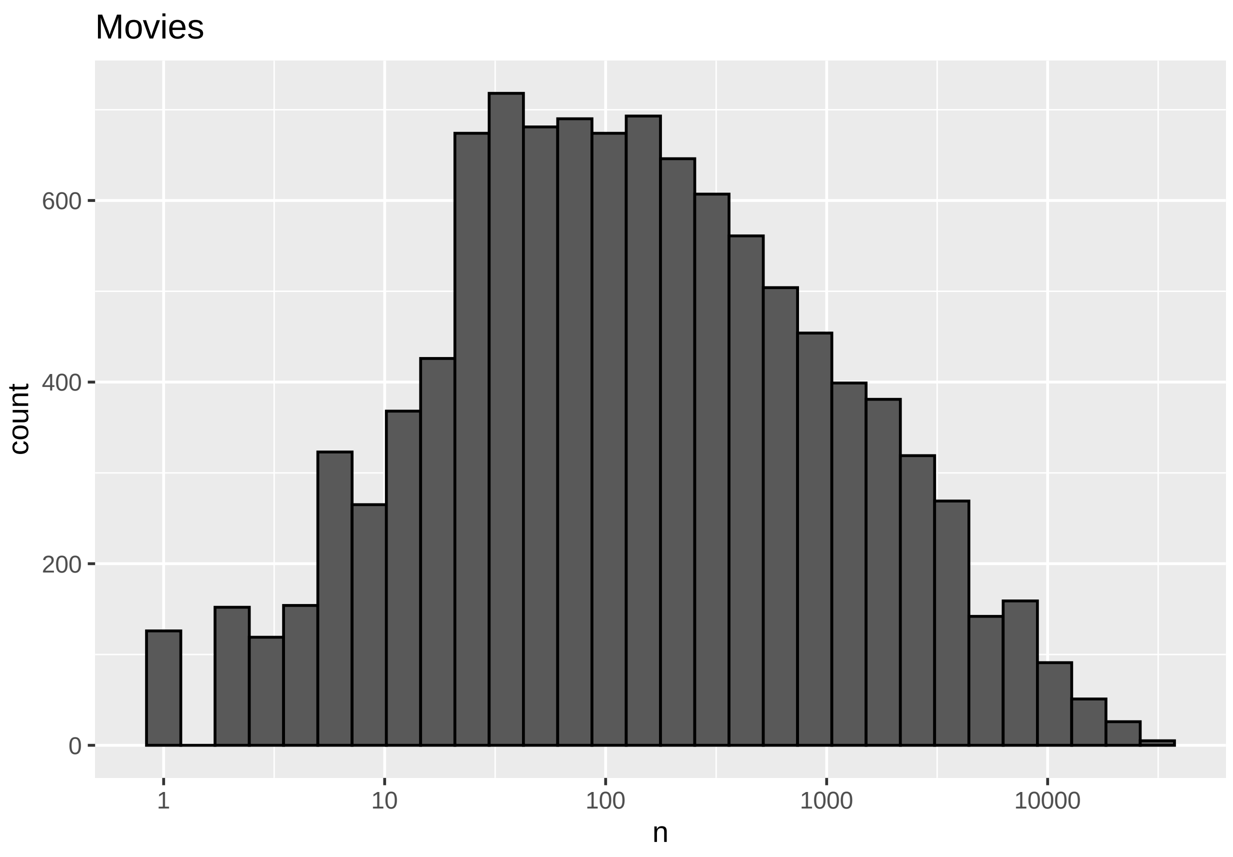
Let's see how many unique users and movies are in the training dataset.

```r
edx %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

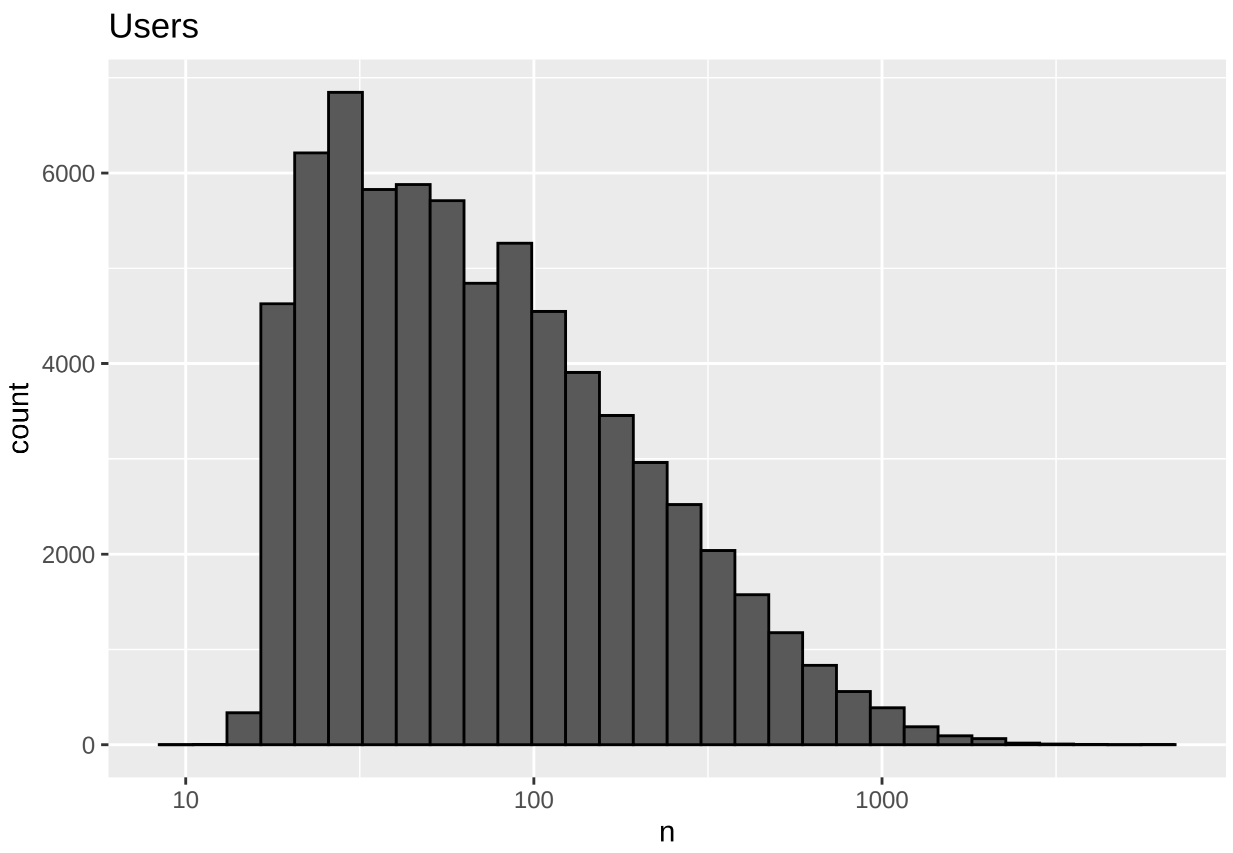
```
##   n_users n_movies
## 1   69878    10677
```

```r
# We have 69878 users and 10677 movies.


# Some distributions may give us a better understanding
# of the reviews/ratings.
# Let's see the distribution of how many ratings the movies receive.
```

```r
edx %>%
  dplyr::count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Movies")
```



```r
# From the graph we can see that the most common number
# of ratings received for movies in this dataset is roughly
# around 100.


# Let's see the distribution of how many ratings users give.

edx %>%
  dplyr::count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Users")
```



From the graph we can see that the most common number of ratings given by users is around 20 to 150 ratings.

Let's split up the genres into separate columns per genre (such as gAction for Action) and assign a 1 in each column for reviews that are of a movie in that genre and assign a 0 if not.

This could provide more accuracy later on if the effectsof genres are significant enough.

genreList <- as.factor(edx$genres) levels(genreList)

The two lines above gave me a quick way to see what genres are in the data.

Let's make columns to check for each genre.

Keep in mind we are merely making it easier for the computer to understand what genres are involved in each review.

```r
edx$gAction <- ifelse(grepl("Action", edx$genres), 1, 0)
edx$gAdvent <- ifelse(grepl("Adventure", edx$genres), 1, 0)
edx$gAnim <- ifelse(grepl("Animation", edx$genres), 1, 0)
edx$gChild <- ifelse(grepl("Children", edx$genres), 1, 0)
edx$gComedy <- ifelse(grepl("Comedy", edx$genres), 1, 0)
edx$gFantasy <- ifelse(grepl("Fantasy", edx$genres), 1, 0)
edx$gSciFi <- ifelse(grepl("Sci-Fi", edx$genres), 1, 0)
edx$gImax <- ifelse(grepl("IMAX", edx$genres), 1, 0)
edx$gDrama <- ifelse(grepl("Drama", edx$genres), 1, 0)
edx$gHorror <- ifelse(grepl("Horror", edx$genres), 1, 0)
edx$gMyst <- ifelse(grepl("Mystery", edx$genres), 1, 0)
edx$gThrill <- ifelse(grepl("Thriller", edx$genres), 1, 0)
edx$gCrime <- ifelse(grepl("Crime", edx$genres), 1, 0)
```

```r
edx$gRom <- ifelse(grepl("Romance", edx$genres), 1, 0)
edx$gWar <- ifelse(grepl("War", edx$genres), 1, 0)
edx$gWest <- ifelse(grepl("Western", edx$genres), 1, 0)
edx$gMusic <- ifelse(grepl("Musical", edx$genres), 1, 0)
edx$gDocu <- ifelse(grepl("Documentary", edx$genres), 1, 0)
edx$gFilmN <- ifelse(grepl("Film-Noir", edx$genres), 1, 0)
tibble(edx)
```

```
## # A tibble: 9,000,055 x 1
##    edx$userId $movieId $rating $timestamp $title $genres $gAction $gAdvent
##         <int>    <dbl>   <dbl>      <int> <chr>  <chr>      <dbl>    <dbl>
##  1          1      122       5  838985046 Boome~ Comedy~        0        0
##  2          1      185       5  838983525 Net, ~ Action~        1        0
##  3          1      292       5  838983421 Outbr~ Action~        1        0
##  4          1      316       5  838983392 Starg~ Action~        1        1
##  5          1      329       5  838983392 Star ~ Action~        1        1
##  6          1      355       5  838984474 Flint~ Childr~        0        0
##  7          1      356       5  838983653 Forre~ Comedy~        0        0
##  8          1      362       5  838984885 Jungl~ Advent~        0        1
##  9          1      364       5  838983707 Lion ~ Advent~        0        1
## 10          1      370       5  838984596 Naked~ Action~        1        0
## # ... with 9,000,045 more rows, and 17 more variables: $gAnim <dbl>,
## #   $gChild <dbl>, $gComedy <dbl>, $gFantasy <dbl>, $gSciFi <dbl>,
## #   $gImax <dbl>, $gDrama <dbl>, $gHorror <dbl>, $gMyst <dbl>,
## #   $gThrill <dbl>, $gCrime <dbl>, $gRom <dbl>, $gWar <dbl>, $gWest <dbl>,
## #   $gMusic <dbl>, $gDocu <dbl>, $gFilmN <dbl>
```

Let’s run a multiple regression analysis on the training set to see if genre affects users’ movie ratings.

This will statistically predict the movies’ ratings based on the whether the movie belongs to a particular genre or not.

```r
genreFit <- lm(rating ~ gAction + gAdvent + gAnim + gChild + gComedy + gFantasy + gSciFi + gImax + gDra
summary(genreFit)
```

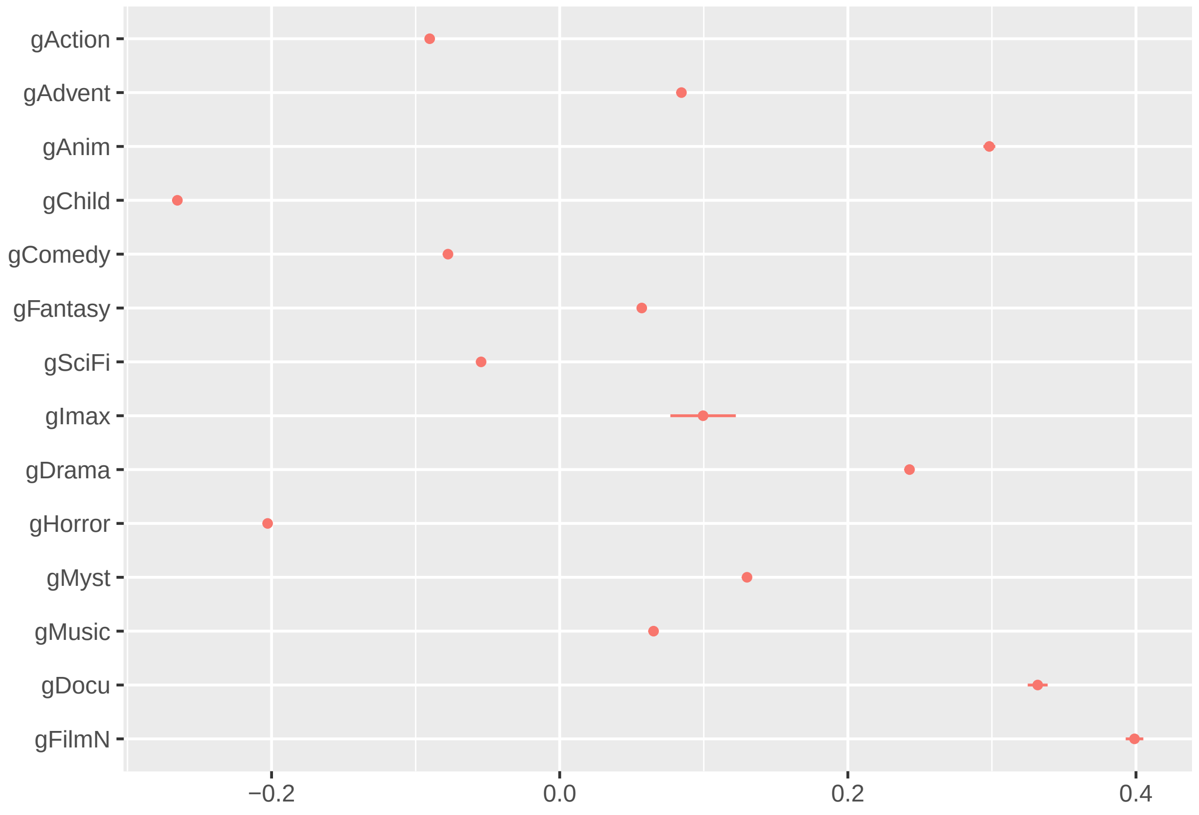
```
##
## Call:
## lm(formula = rating ~ gAction + gAdvent + gAnim + gChild + gComedy +
##     gFantasy + gSciFi + gImax + gDrama + gHorror + gMyst + gMusic +
##     gDocu + gFilmN, data = edx)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.7780 -0.6145  0.1779  0.6410  2.0394
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.4492515  0.0009500 3630.68   <2e-16 ***
## gAction     -0.0902482  0.0009543  -94.56   <2e-16 ***
## gAdvent      0.0844932  0.0009937   85.03   <2e-16 ***
## gAnim        0.2982093  0.0021112  141.25   <2e-16 ***
## gChild      -0.2653720  0.0017890 -148.34   <2e-16 ***
## gComedy     -0.0775575  0.0008334  -93.06   <2e-16 ***
## gFantasy     0.0569598  0.0012437   45.80   <2e-16 ***
## gSciFi      -0.0545653  0.0010758  -50.72   <2e-16 ***
## gImax        0.0995142  0.0115853    8.59   <2e-16 ***
## gDrama       0.2428138  0.0008504  285.53   <2e-16 ***
## gHorror     -0.2027223  0.0013877 -146.08   <2e-16 ***
## gMyst        0.1300042  0.0014831   87.66   <2e-16 ***
## gMusic       0.0651392  0.0017871   36.45   <2e-16 ***
## gDocu        0.3317775  0.0035342   93.88   <2e-16 ***
## gFilmN       0.3989744  0.0031232  127.74   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.044 on 9000040 degrees of freedom
## Multiple R-squared:  0.03061,    Adjusted R-squared:  0.03061
## F-statistic: 2.03e+04 on 14 and 9000040 DF,  p-value: < 2.2e-16
```

It turns out that all genres have a statistically significant effect on the movie rating.

You can see this by looking at the three stars "***" beside each coefficient, which shows that the p-value is small enough for the coefficient to have a high significance.

Here is a graph of the estimated effects of each genre on the movie rating.

```
dwplot(genreFit)
```



```
genreFit
```

```
##
## Call:
## lm(formula = rating ~ gAction + gAdvent + gAnim + gChild + gComedy +
##     gFantasy + gSciFi + gImax + gDrama + gHorror + gMyst + gMusic +
##     gDocu + gFilmN, data = edx)
##
## Coefficients:
## (Intercept)      gAction      gAdvent        gAnim       gChild
##     3.44925     -0.09025      0.08449      0.29821     -0.26537
##     gComedy     gFantasy       gSciFi        gImax       gDrama
##    -0.07756      0.05696     -0.05457      0.09951      0.24281
##     gHorror        gMyst       gMusic        gDocu       gFilmN
##    -0.20272      0.13000      0.06514      0.33178      0.39897
```

```r
modcoef <- summary(genreFit)[["coefficients"]]
modcoef[order(modcoef[ , 1]), ]
```

```
##                Estimate   Std. Error     t value      Pr(>|t|)
## gChild      -0.26537203 0.0017889551 -148.339122  0.000000e+00
## gHorror     -0.20272235 0.0013877165 -146.083402  0.000000e+00
## gAction     -0.09024821 0.0009543470  -94.565411  0.000000e+00
## gComedy     -0.07755751 0.0008334233  -93.058967  0.000000e+00
## gSciFi      -0.05456533 0.0010757715  -50.722040  0.000000e+00
## gFantasy     0.05695984 0.0012436947   45.798891  0.000000e+00
## gMusic       0.06513923 0.0017871201   36.449275 7.419213e-291
## gAdvent      0.08449315 0.0009936885   85.029818  0.000000e+00
## gImax        0.09951418 0.0115852871    8.589703  8.720766e-18
## gMyst        0.13000417 0.0014830673   87.658983  0.000000e+00
## gDrama       0.24281382 0.0008503866  285.533436  0.000000e+00
## gAnim        0.29820929 0.0021111580  141.253895  0.000000e+00
## gDocu        0.33177755 0.0035342157   93.875864  0.000000e+00
## gFilmN       0.39897444 0.0031232406  127.743742  0.000000e+00
## (Intercept)  3.44925153 0.0009500294 3630.678649  0.000000e+00
```

If we look closely at the results, we can see that movies of the Film-Noir genre are expected to score an estimated 0.399 points higher in ratings than the intercept (3.449).

Genres with the most positive effect are Film-Noir (0.399), Documentary (0.332), and Animation (0.298).

We also see that movies of the Children genre are expected to score an estimated 0.265 points lower in ratings than the intercept (3.449).

Genres with the most negative effect are Children (-0.265), Horror (-0.203), and Action (-0.090).

Our modeling approach will be to start off with some simple models and gradually add complexity in the hopes of reaching a lower RMSE (greater accuracy) and eventually accomplishing the goal of an RMSE of less than 0.8649.

The models we will be building are:

1. "Only the Average/Naive Approach Model", predicting that all reviews will be equal to the average of all reviews in the testing set (as a baseline model for testing subsequent model accuracy).
2. "Movie Effect Model", predicting based on previous reviews for that movie (movie bias).
3. "Movie + User Effects Model", adding user bias to the previous model.
4. "Regularized Movie + User Effect Model", regularizing the previous model because averages of biases of movies and users with few reviews are too extreme to be accurately predicted and need to be pulled in (or "rubber-banded") closer to the dataset average the fewer reviews there are.
5. "Regularized Movie + User + Genre Effect Model", adding genre bias to the previous model and regularizing (or "rubber-banding") all three biases.

Let's look at the results of these models in the next section.

Results

Now I'll present the modeling results and discuss the model performance as you step through the code with me.

You can run all of the code by clicking Run. You can run it line by line by pressing Ctrl + Enter on your keyboard. You can also highlight a section of code and run just that by clicking Run or pressing Ctrl + Enter on your keyboard.

Now let's build some models and run some tests to get the RMSE (Residual Mean Square Error) of each one.

A RMSE shows us how far off the mark our model's predictions are, which tells us how accurate it is.

A simple way of thinking about this is that the model (or algorithm) will learn about the data by taking in different factors and make a prediction of what star rating (on a 0.5 to 5 star scale) a user will rank a movie based on those factors. Different approaches will have the model/algorithm using the factors given to it to make predictions in different ways.

The model/algorithm decides to predict a review rating "Y" based on factors "A", "B", and "C" (or more). Then the model/algorithm is exposed to the testing dataset to see if what it predicts as the review rating "Y" based on the factors in the new dataset "A", "B", and "C" is actually that accurate or not. Then from the results we can compute our RMSE (Residual Mean Square Error). This is how we test the model's accuracy.

The RMSE is the standard deviation of the prediction errors (residuals).

In other words, it's the average of how far the predictions deviate from what they are trying to predict.

The lower the RMSE, the more accurate the algorithm's predictions are.

Our goal is to build an algorithm with as much accuracy as possible, so we want to have as low of a RMSE as we can get.

For this analysis, we'll aim for a RMSE lower than 0.8649.

```r
# Here is our RMSE formula:

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

To explain this step by step, we subtract the predicted ratings from the true ratings, then we square them all (to turn negative results into positive results for ease of use since we are measuring deviation/distance), then we get the average of all of those results, then we get the square root of that average.

This shows us the average of how far off the predictions are from what they are trying to predict. That's our RMSE.

If our RMSE is larger than 1, it means our rating prediction is on average more than one star off the mark (on a 0.5 to 5 star scale), which isn't very good.

Note: Users can rate movies in 0.5 increments but most of the ratings are 1, 2, 3, 4, or 5 stars.

Now let's try to build a prediction model with a RMSE of less than 0.8649.

For our first prediction model, we'll start with a very simple approach. Let's get the average of all the ratings.

```r
mu_hat <- mean(edx$rating)
mu_hat
```

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

3.512 is the average of all the ratings in the training set.

If we build a "naive" prediction model, predicting the ratings of movies in the validation set using only the average of ratings in the training set, we get a RMSE of 1.061.

```
onlyAverage_rmse <- RMSE(validation$rating, mu_hat)
onlyAverage_rmse
```

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

A RMSE of 1.061 is not very accurate. It means our predictions are about 1 star off (on a 0.5 to 5 star scale) on average from the actual rating users give in the validation set.

Let's put this model into a list and start off our list of attempts:

```
rmseTestResultsList <- tibble(method = "Only the Average/Naive Approach Model", RMSE = onlyAverage_rmse
rmseTestResultsList %>% knitr::kable()
```

method	RMSE
Only the Average/Naive Approach Model	1.061202

For our second prediction model, we'll predict what a user will rate a movie based on the average rating of that movie. First we get the average of all ratings again. We'll call this "mu".

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

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

Then we'll use mu to get our "b_i", which represents the average rating for movie "i" (you could also refer to this as the bias of ratings for movie "i").

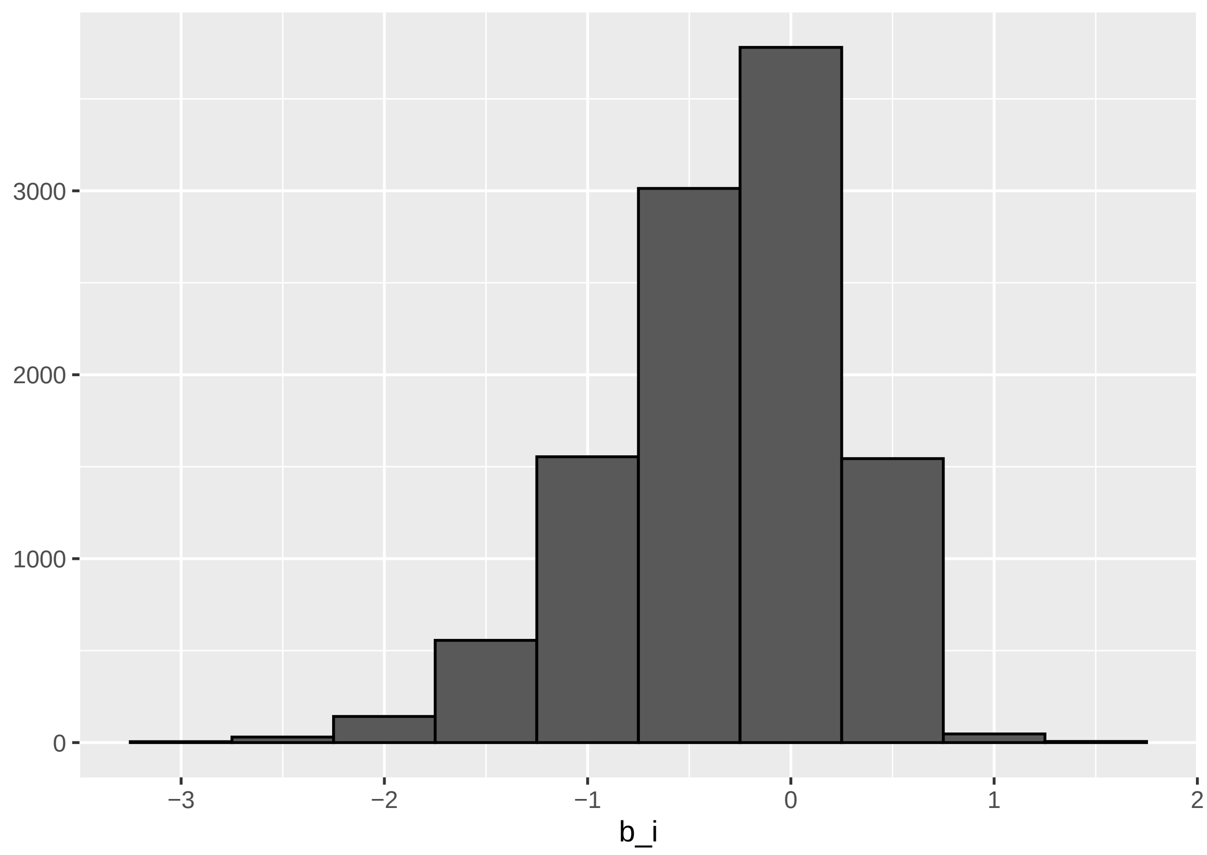
The letter "b" (short for bias) represents the coefficient estimates of our predictors. In this model we'll only have one "bias" predictor, which is movie bias.

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

This plot shows us these "movie biases" per movie. Zero represents the dataset average on this plot.

Notice some movies score far below or above the dataset average of 3.512.

```
movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = ., color = I("black"))
```

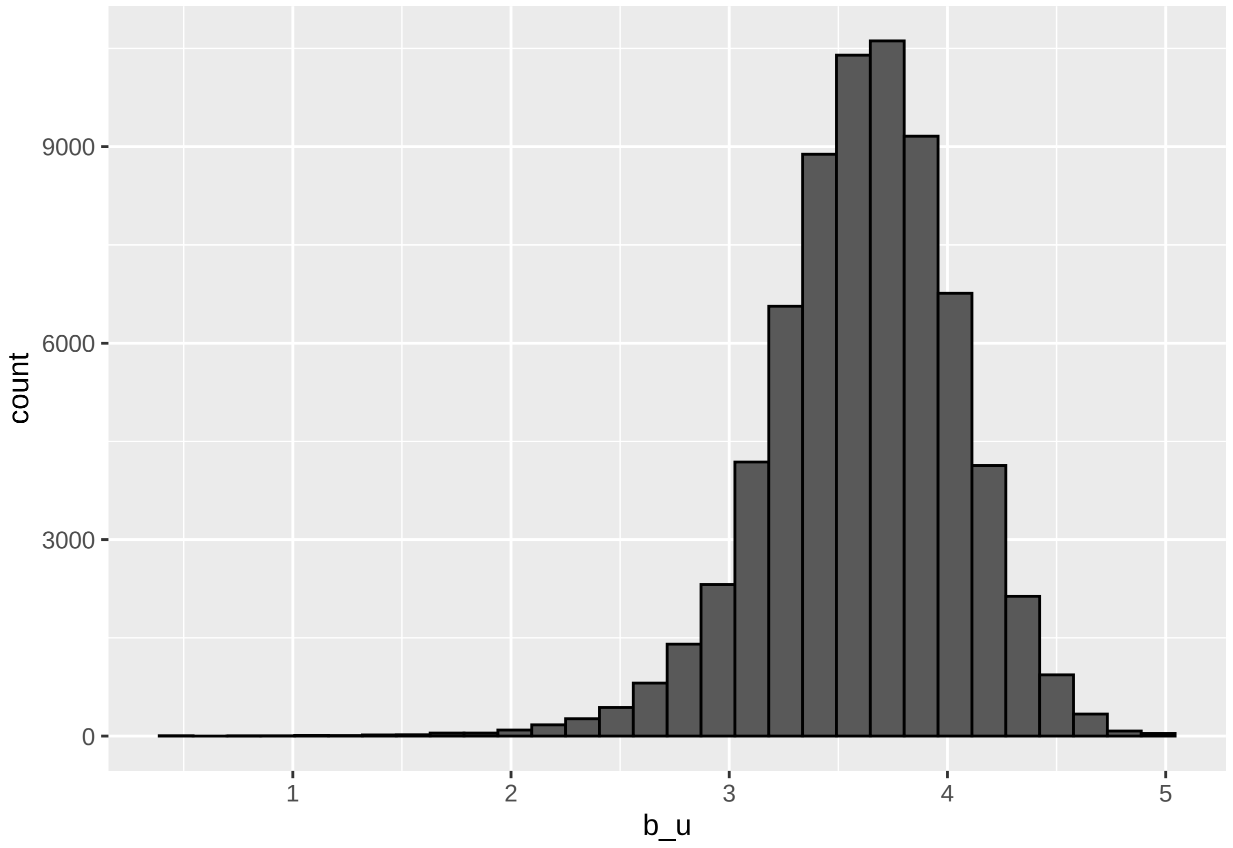


```r
# Let's see how much the predictions improve with this
# movie-based model.
predicted_ratings <- mu + validation %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i

movieEffect_rmse <- RMSE(predicted_ratings, validation$rating)
rmseTestResultsList <- bind_rows(rmseTestResultsList,
                                 tibble(method="Movie Effect Model",
                                        RMSE = movieEffect_rmse ))

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

method	RMSE
Only the Average/Naive Approach Model	1.0612018
Movie Effect Model	0.9439087

```r
# In comparison to the "Only Average" model, we've made
# a slight improvement but we're still
# nowhere near the goal of an RMSE below 0.8649.
```

```r
# From this plot we can see that different users have
# different averages of their ratings ("b_u" is user bias).
edx %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black")
```



```
# Let's try to improve our accuracy by creating a model
# based on user bias (users' previous ratings).
user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

# Now we'll predict the validation set again after
# combining our movie-based approach with our new user-based approach.
predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred


userEffect_rmse <- RMSE(predicted_ratings, validation$rating)
rmseTestResultsList <- bind_rows(rmseTestResultsList,
                                 tibble(method="Movie + User Effects Model",
                                        RMSE = userEffect_rmse ))
rmseTestResultsList %>% knitr::kable()
```

method	RMSE
Only the Average/Naive Approach Model	1.0612018
Movie Effect Model	0.9439087
Movie + User Effects Model	0.8653488

The combined approach gave us an RMSE of 0.8653! That's almost less than our goal of 0.8649! This is great! We're getting close!

If we keep studying the data we might notice that many of the highest-ranking and lowest-ranking movies do not have many reviews.

```r
movie_titles <- movielens %>%
  select(movieId, title) %>%
  distinct()

edx %>% dplyr::count(movieId) %>%
  left_join(movie_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i, n) %>%
  slice(1:10) %>%
  knitr::kable()
```

```
## Joining, by = "movieId"
```

title	b_i	n
Hellhounds on My Trail (1999)	1.487535	1
Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)	1.487535	2
Shadows of Forgotten Ancestors (1964)	1.487535	1
Fighting Elegy (Kenka erejii) (1966)	1.487535	1
Sun Alley (Sonnenallee) (1999)	1.487535	1
Blue Light, The (Das Blaue Licht) (1932)	1.487535	1
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.237535	4
Human Condition II, The (Ningen no joken II) (1959)	1.237535	4
Human Condition III, The (Ningen no joken III) (1961)	1.237535	4
Constantine's Sword (2007)	1.237535	2

```r
edx %>% dplyr::count(movieId) %>%
  left_join(movie_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  select(title, b_i, n) %>%
  slice(1:10) %>%
  knitr::kable()
```

```
## Joining, by = "movieId"
```

title	b_i	n
Besotted (2001)	-3.012465	2
Hi-Line, The (1999)	-3.012465	1
Accused (Anklaget) (2005)	-3.012465	1
Confessions of a Superhero (2007)	-3.012465	1

title	b_i	n
War of the Worlds 2: The Next Wave (2008)	-3.012465	2
SuperBabies: Baby Geniuses 2 (2004)	-2.717822	56
Hip Hop Witch, Da (2000)	-2.691037	14
Disaster Movie (2008)	-2.653090	32
From Justin to Kelly (2003)	-2.610455	199
Criminals (1996)	-2.512465	2

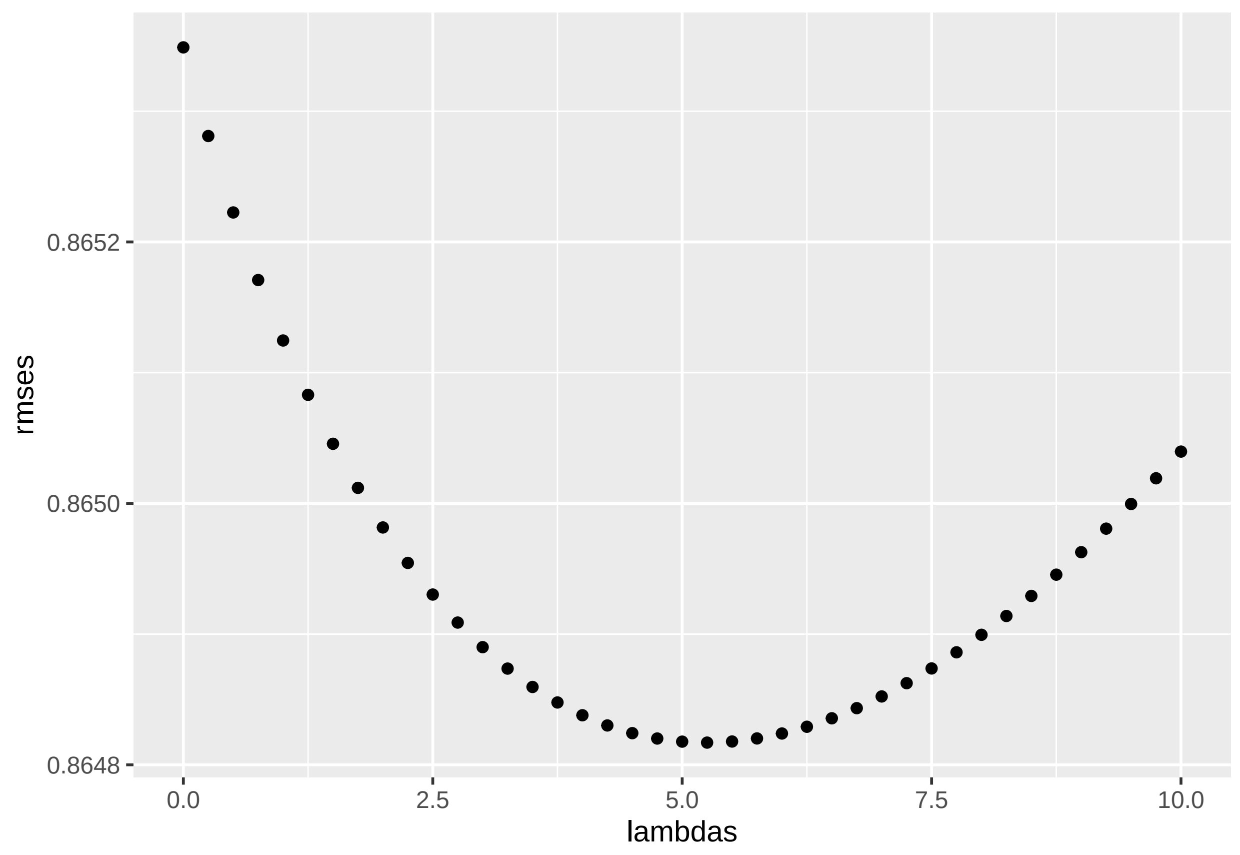
Many of the highest and lowest-ranking movies have less than five reviews. This is because small number of reviews the movie received were very high or low. This causes higher variability in their rankings and may make our prediction model innacurate. Because averages of biases of movies and users with few reviews are too extreme to be accurately predicted, they need to be pulled in (or "rubber-banded") closer to the dataset average the fewer reviews there are.

In order to smooth out the problems caused by the variability of the rankings of these movies with low numbers of reviews, we can improve our model by using regularization.

With regularization our predictions will not be as swayed by the noise and outliers of movies and users with only a few extreme reviews because they are penalized.

Regularizing our model will bring the predictions for movies with less ratings closer to the average of all ratings in the dataset, but will not affect movies with many ratings as much.

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

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

  mu <- mean(edx$rating)

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

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

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

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

qplot(lambdas, rmses)
```



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

```
## [1] 5.25
```

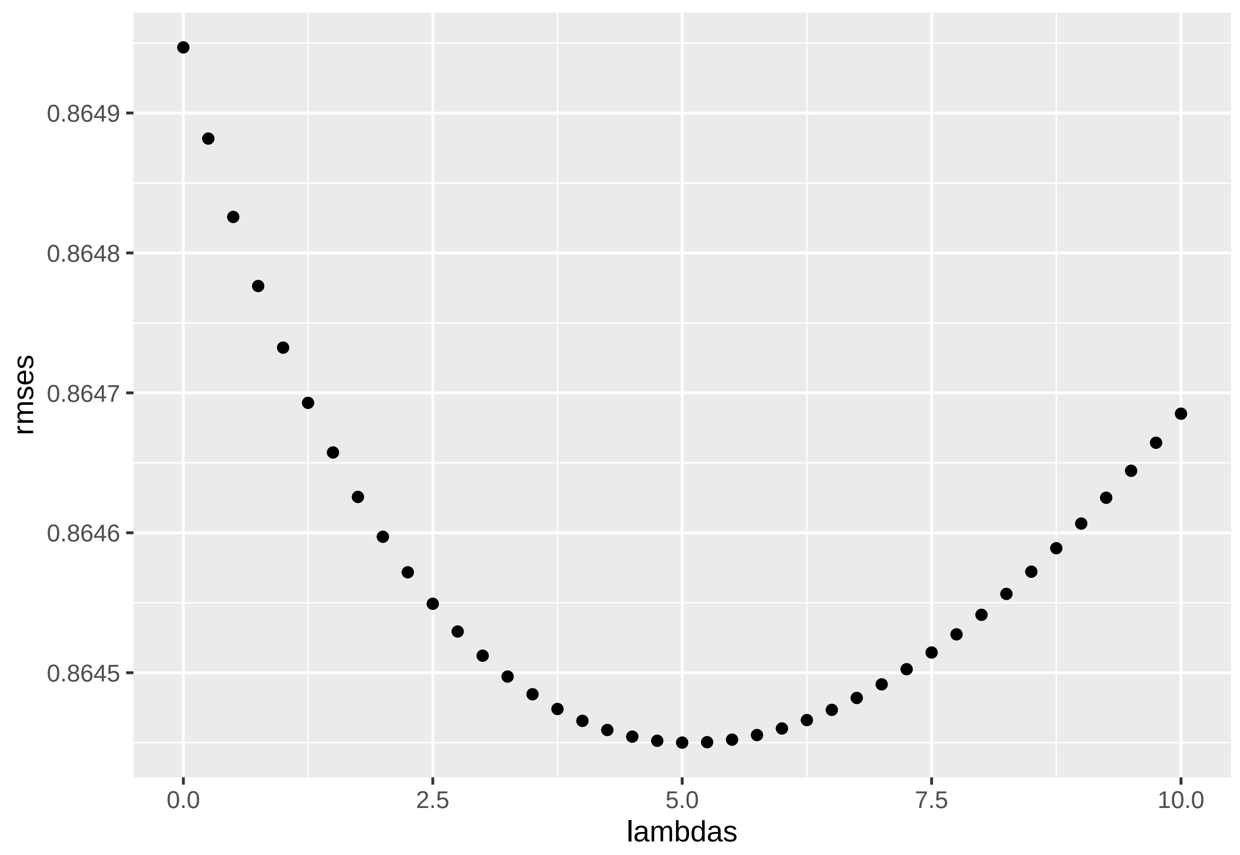
```
# The lambda that gives us the most accuracy (smallest RMSE) is 5.25.

rmseTestResultsList <- bind_rows(rmseTestResultsList,
                                 tibble(method="Regularized Movie + User Effect Model",
                                        RMSE = min(rmses)))
rmseTestResultsList %>% knitr::kable()
```

method	RMSE
Only the Average/Naive Approach Model	1.0612018
Movie Effect Model	0.9439087
Movie + User Effects Model	0.8653488
Regularized Movie + User Effect Model	0.8648170

This combined and regularized approach gave us an RMSE of 0.8648! That's less than our goal of 0.8649! We did it!

As we saw earlier in the multiple regression analysis, genres had a statistically significant effect on ratings. So let's see if we can get an even lower RMSE by adding the effects of the different genres to our regularized movie and user effect prediction algorithm.

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

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

  mu <- mean(edx$rating)

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

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

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

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

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

qplot(lambdas, rmses)
```



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

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

The lambda that gives us the most accuracy (smallest RMSE) is 5.

The higher the lambda, the less significance is held by the coefficients, giving them less and less effect on the prediction algorithm. Some variables may play no role in the model at all if the lambda is high enough.

Lambda is a tuning parameter for our model. This approach tests different values of lambda and picks the one with the lowest RMSE.

```r
rmseTestResultsList <- bind_rows(rmseTestResultsList,
                                 tibble(method="Regularized Movie + User + Genre Effect Model",
                                        RMSE = min(rmses)))


# Let's see our results:
rmseTestResultsList %>% knitr::kable()
```

method	RMSE
Only the Average/Naive Approach Model	1.0612018
Movie Effect Model	0.9439087
Movie + User Effects Model	0.8653488
Regularized Movie + User Effect Model	0.8648170
Regularized Movie + User + Genre Effect Model	0.8644501

This regularized test of the movie, user, and genre effects combined gives us a RMSE of 0.86445, which is even more accurate.

```r
minRMSE <- min(rmses)
cat(minRMSE, "is the RMSE of the Regularized Movie + User + Genre Effect Model,
    which accomplishes our goal of reaching a RMSE of less than 0.8649.")
```

```
## 0.8644501 is the RMSE of the Regularized Movie + User + Genre Effect Model,
##     which accomplishes our goal of reaching a RMSE of less than 0.8649.
```

Conclusion

In this section I'll give a brief summary of the report, its limitations and future work.

I split the data into a training set (90% of data) to train the prediction models and a testing set (10% of data) to test the accuracy of the prediction model.

After running five prediction models, the lowest Residual Mean Square Error (RMSE) obtained was 0.8644501, which accomplishes the goal of reaching lower than 0.8649.

The most effective prediction model was "Regularized Movie + User + Genre Effect Model", where I used the biases per movie, per user, and per genre of the reviews in the training set and then regularized (or "rubber-banded") the results, penalizing biases of movies/users/genres with low review counts by pulling them toward the dataset average.

I feel as though my report has some limitations. I could have taken more modeling approaches, such as Naive Bayes Classification and Matrix Decomposition, to potentially reach a lower RMSE.

I keep thinking about how, in the Netflix challenge, to win the grand prize of $1 million a participating team had to reach a RMSE of about 0.857.

I would like to improve this analysis in the future by finding some prediction model approaches that will give me a RMSE of less than 0.857.

Thank you for reading my report. I hope you enjoyed it.

- Avery Clark