Movielens Recommender System

Skyler Shapiro

12/31/2020

Contents

1. Introduction

This project aimed to create a movie recommender system using the Movielens 10M data set as part of the capstone project for the HarvardX Data Science Professional Certificate program. The Movielens 10M dataset includes 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users (https://grouplens.org/datasets/movielens/). I trained a machine learning algorithm to predict movie ratings using a subset of the Movielens dataset. I tested the recommender system by predicting the movie ratings of the other subset of the dataset and compared the predicted ratings to the actual ratings.

First, the dataset was partitioned into the “edx” and “validation” sets. The “edx” dataset was further partitioned for training and testing, which allowed for cross validation when constructing the model.I used Root Mean-Squared Error (RMSE) as the measure of prediction accuracy.

First, I constructed linear models with movie, user, age, and genre effects using the training set and evaluated on the testing set. Then I constructed a regularized model which used penalized least-squares regression using the training set and evaluated using the testing set. Regularization constrains the total variability of the effect sizes by penalizing large estimates that come from small sample sizes. Lastly, I tested my final regularized model on the validation set (final holdout set) to assess model performance.

The code can be found on github here: (https://github.com/skylershapiro/Movielens).

2. Methods, Exploration, and Analysis

2.1 Download Data

Download and format the MovieLens 10M dataset:https://grouplens.org/datasets/movielens/10m/

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

Here is a sample of the Movielens 10M dataset.

Table 1: First 6 Rows of edx Dataset

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

2.2 Load libraries

Load in required libraries.

```
library(tidyverse)
library(caret)
library(data.table)
library(lubridate)
```

```r
library(bannerCommenter)
```

2.3 Generate and Processes the Data

2.3.1 Generate Datasets

Partition the Movielens 10M dataset into "edx" (90%) for model construction and "validation" (10%) for evaluation. Save datasets as "mlens" and "val".

```r
# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in validation set are also in edx set
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

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

# Save Data
mlens <- edx
val <- validation
```

2.3.2 Process and Clean Datasets

Separate the year of release from "title", extract the year from "timestamp", and define column "age" as difference between the year of movie release and year of rating.

Table 2: First 6 Rows of mlens Dataset

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

```r
# Separate year from title
mlens <- mlens %>% mutate(movie_year = as.numeric(str_sub(title,-5,-2)))
val <- val %>% mutate(movie_year = as.numeric(str_sub(title,-5,-2)))

# Convert timestamp column to date format
mlens <- mutate(mlens, date = as_datetime(timestamp), rating_year = year(date))
val <- mutate(val, date = as_datetime(timestamp), rating_year = year(date))

# Create Age by subtracting movie_year from rating_year
mlens <- mutate(mlens, age = rating_year - movie_year)
```

```r
val <- mutate(val, age = rating_year - movie_year)
```

The data should now look like this.

Table 3: Processed Columns in mlens Dataset

title	genres	movie_year	rating_year	age
Boomerang (1992)	Comedy|Romance	1992	1996	4
Net, The (1995)	Action|Crime|Thriller	1995	1996	1
Outbreak (1995)	Action|Drama|Sci-Fi|Thriller	1995	1996	1
Stargate (1994)	Action|Adventure|Sci-Fi	1994	1996	2
Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi	1994	1996	2
Flintstones, The (1994)	Children|Comedy|Fantasy	1994	1996	2

2.3.3 Create training and testing datasets

Further partition “mlens” for training (train_set, 80%) and testing (test_set, 20%).

```r
# Partition mlens dataset
test_index <- createDataPartition(y = mlens$rating, times = 1,
                                  p = 0.2, list = FALSE)
train_set <- mlens[-test_index,]
test_set <- mlens[test_index,]
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

2.4 Data Exploration

First, let us check the dimensions of the mlens dataset.

Table 4: Number of Ratings and Variables

ratings	variables
9000055	10

The dataset contains the following variables

```
##  [1] "userId"      "movieId"     "rating"      "timestamp"   "title"
##  [6] "genres"      "movie_year"  "date"        "rating_year" "age"
```

Finally, we will look at the total number of users and movies.

Table 5: Number of Unique Users and Movies

n_users	n_movies
69878	10677

2.4.1 Inspect Ratings

Ratings Summary

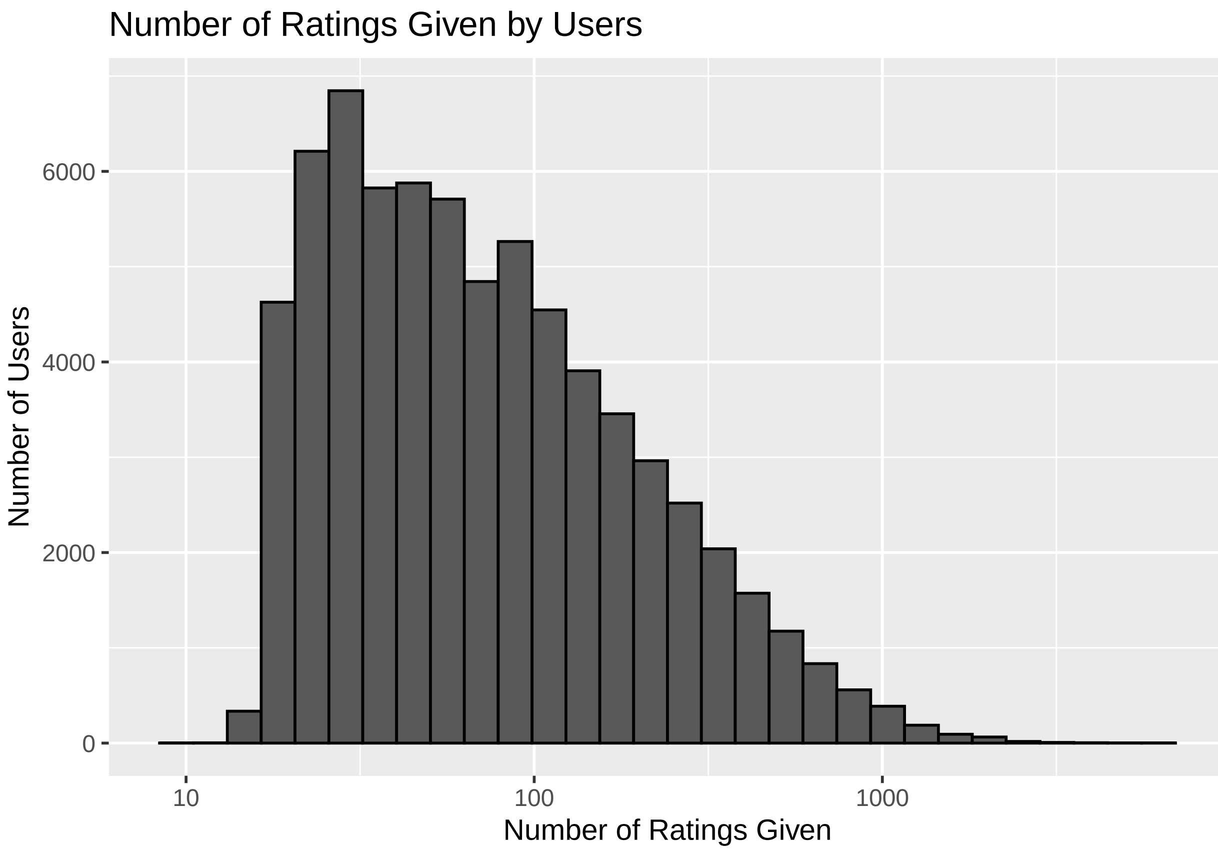
Review ratings and check for out-of-range values (ratings should be between 0.5 and 5.0 according to Movielens data documentation). As we can see, the ratings fall between 0.5 and 5.0 as expected. Additionally, we can see that 75% of the ratings fall between 3.0 and 5.0 with an overall mean rating of 3.512.

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

Visualizing the Data

First let us look at the distribution of the number of ratings given per user. We can see from the summary that the median number of ratings per user is 62 and the maximum number of ratings per user is 6616. This indicates the distribution is strongly skewed right (as seen in the figure below).

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    10.0    32.0    62.0   128.8   141.0  6616.0
```



We can look at the number ratings per movie. From the summary below, we can see that some movies are rated much more often than others.

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     1.0    30.0   122.0   842.9   565.0 31362.0
```

Here we can see the most and least often rated movies

Table 6: Most Rated Movies

movieId	title	count
296	Pulp Fiction (1994)	31362
356	Forrest Gump (1994)	31079
593	Silence of the Lambs, The (1991)	30382
480	Jurassic Park (1993)	29360
318	Shawshank Redemption, The (1994)	28015

Table 7: Least Rated Movies

movieId	title	count
64953	Dirty Dozen, The: The Fatal Mission (1988)	1
64976	Hexed (1993)	1
65006	Impulse (2008)	1
65011	Zona Zamfirova (2002)	1
65025	Double Dynamite (1951)	1
65027	Death Kiss, The (1933)	1

Lastly we can inspect the number of ratings per genre.

Table 8: Number of Ratings per Genre

genres	count
Drama	3910127
Comedy	3540930
Action	2560545
Thriller	2325899
Adventure	1908892
Romance	1712100
Sci-Fi	1341183
Crime	1327715
Fantasy	925637
Children	737994
Horror	691485
Mystery	568332
War	511147
Animation	467168
Musical	433080
Western	189394
Film-Noir	118541
Documentary	93066
IMAX	8181
(no genres listed)	7

2.5 Modeling Approach

2.5.1 Defining the Evaluation Method

First, we will define how our model will be evaluated using Root Mean-Squared Error (RMSE) of the true compared with predicted ratings. Our goal is to minimize the RMSE of our final model.

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

2.5.2 Model 1: Naive Model

First, we create a naive model to establish a baseline for comparison. Our naive model predicts the mean value of the distribution of ratings for every movie.

```
# Calculate Mean Rating
mu_hat <- mean(train_set$rating)
```

```
mu_hat
```

```
## [1] 3.512574
```

```
# Build Model and Calculate RMSE
naive_rmse <- RMSE(test_set$rating, mu_hat)
predictions <- rep(mu_hat, nrow(test_set))
```

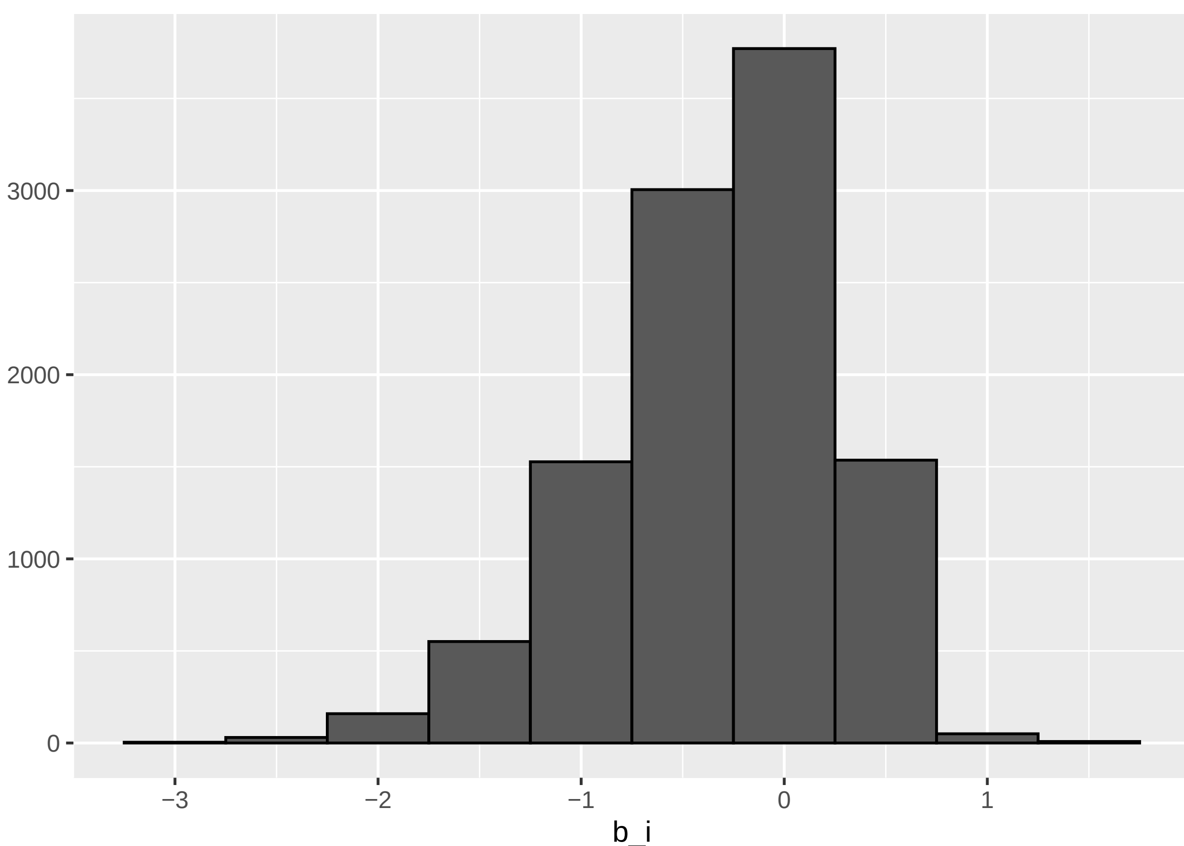
2.5.3 Model 2: Movie Effect Model

The distribution ratings can vary by movie. Now in addition to the overall mean, we will take into account movie effect.

```
# Define Mu (3.512)
mu <- mean(train_set$rating)

# Build Model
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# Predict Ratings
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i
```

Exploring Movie Effect

First let us look at a histogram of movie effects. We can see the plot is slightly skewed left.



Now we can look at the best and worst rated movies according to our movie effects model. From the tables we can see that most of the movies are obscure and rated very few times.

Table 9: Best Rated Movies according to Movie Effect Model

title	b_i	n
Hellhounds on My Trail (1999)	1.487426	1
Shanghai Express (1932)	1.487426	1
Satan's Tango (Sátántangó) (1994)	1.487426	2
Fighting Elegy (Kenka erejii) (1966)	1.487426	1
Sun Alley (Sonnenallee) (1999)	1.487426	1
Bullfighter and the Lady (1951)	1.487426	1
Blue Light, The (Das Blaue Licht) (1932)	1.487426	1
Human Condition II, The (Ningen no joken II) (1959)	1.320760	3
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.237426	4
Life of Oharu, The (Saikaku ichidai onna) (1952)	1.237426	2

Table 10: Worst Rated Movies according to Movie Effect Model

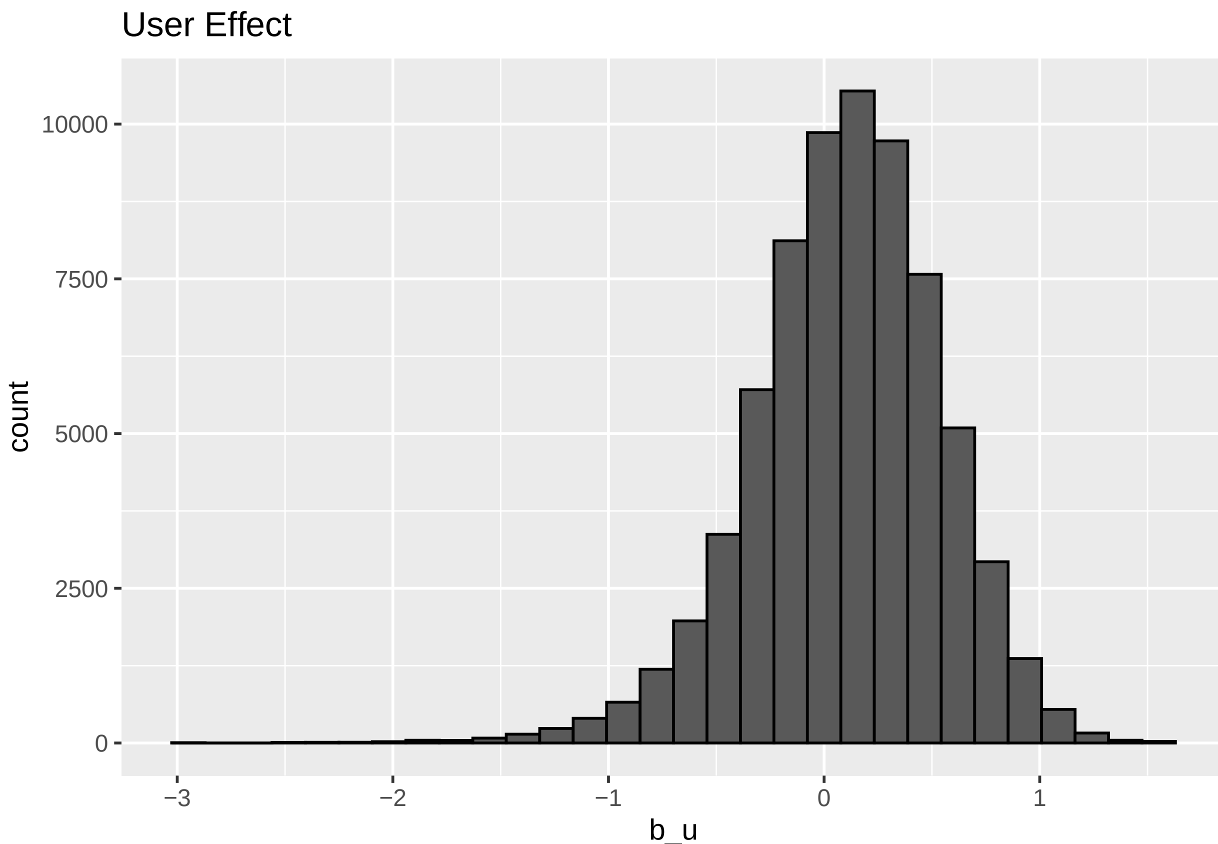
title	b_i	n
Besotted (2001)	-3.012573	2
Grief (1993)	-3.012573	1
Confessions of a Superhero (2007)	-3.012573	1
War of the Worlds 2: The Next Wave (2008)	-3.012573	1
Disaster Movie (2008)	-2.729240	30
SuperBabies: Baby Geniuses 2 (2004)	-2.712573	40
From Justin to Kelly (2003)	-2.652325	161
Hip Hop Witch, Da (2000)	-2.612573	10
Criminals (1996)	-2.512573	2
Mountain Eagle, The (1926)	-2.512573	1

2.5.4 Model 3: Movie and User Effects Model

Next we will incorporate user effect into our existing movie effect model.

```r
# Calculate mu
mu <- mean(train_set$rating)

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

# Predict ratings
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
```

We can see the distribution of user effect modeled in the histogram below.



2.5.5 Model 4: Movie, User and Age Effects Model

Next we will incorporate age effects into our model. Age is defined as the difference between the year of the movie release and the year of the rating given.

```
# Calculate mu
mu <- mean(train_set$rating)

# Build model
age_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(age) %>%
  summarize(b_a = mean(rating - mu - b_i - b_u))

#Predict Ratings
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(age_avgs, by='age') %>%
  mutate(pred = mu + b_i + b_u + b_a) %>%
  .$pred
```

2.5.6 Model 5: Movie, User, Age, and Genre Effects Model

Finally, we will incorporate genre effect into our existing movie, user, and age effects model.

```r
# Calculate mu
mu <- mean(train_set$rating)

# Build model
genre_avgs <- train_set %>%
  left_join(movie_avgs,by="movieId") %>%
  left_join(user_avgs,by="userId") %>%
  left_join(age_avgs,by='age') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u - b_a))

# Predict ratings
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(age_avgs, by='age') %>%
  left_join(genre_avgs, by='genres') %>%
  mutate(pred = mu + b_i + b_u + b_a + b_g) %>%
  .$pred
```

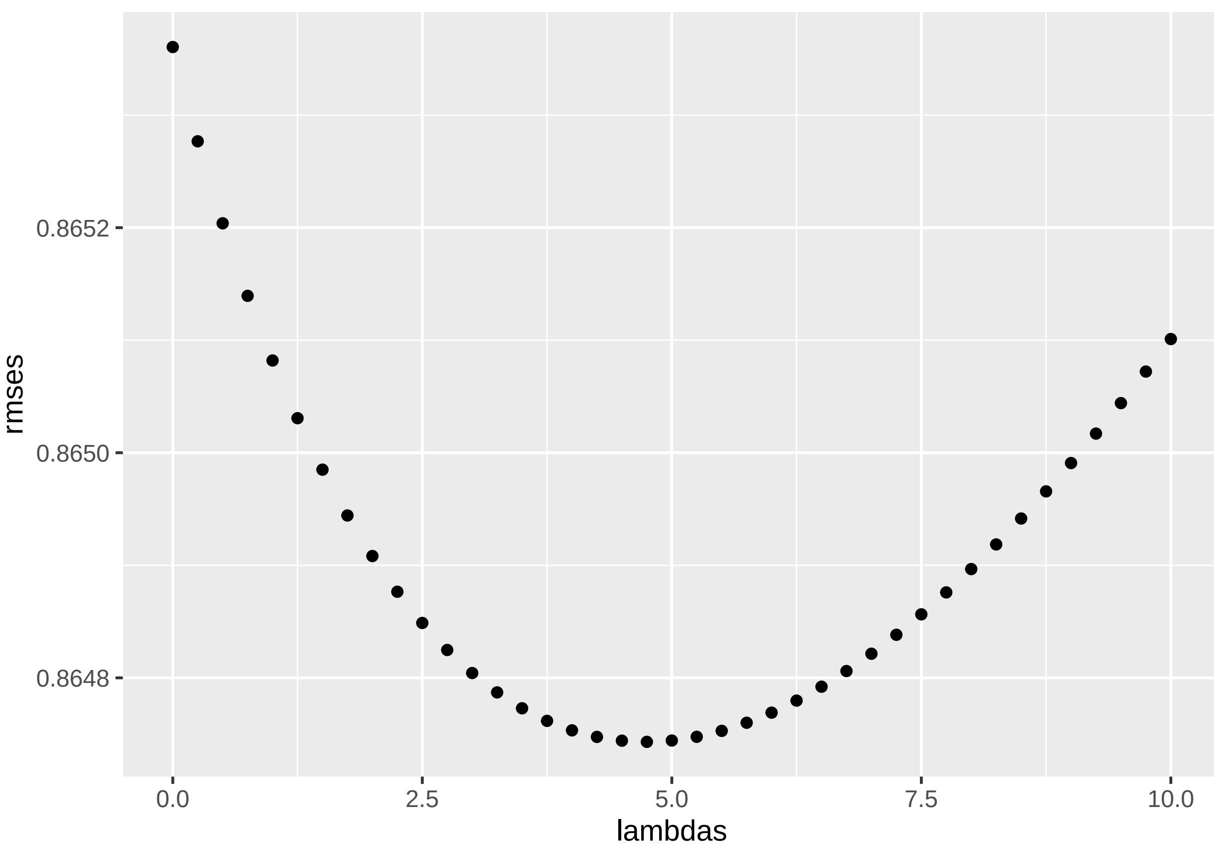
2.6 Regularizing Model using Penalized Least-Squares Regression

Now, we regularized our movie, user, age, and genre effects model using penalized least-squares regression. First we fit models with a range of tuning parameter lambdas and calculate the RMSE.

```r
# Choose range of tuning parameter lambda to optimize model
lambdas <- seq(0, 10, 0.25)

# rmses stores RMSE from each model
rmses <- sapply(lambdas, function(l){

  # Calculate mu
  mu <- mean(train_set$rating)

  # Regularize movie effect
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

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

  # Regularize age effect
  b_a <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by='userId') %>%
    group_by(age) %>%
    summarize(b_a = sum(rating - b_u - b_i - mu)/(n()+l))

  # Regularize genre effect
    b_g <- train_set %>%
      left_join(b_i, by="movieId") %>%
```

```
      left_join(b_u, by='userId') %>%
      left_join(b_a, by='age') %>%
    group_by(genres) %>%
      summarize(b_g = sum(rating - b_a - b_u - b_i - mu)/(n()+l))

    # Predict ratings
  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_a, by='age') %>%
    left_join(b_g, by='genres') %>%
    mutate(pred = mu + b_i + b_u + b_a + b_g) %>%
    pull(pred)

  # Calculate RMSE
  return(RMSE(predicted_ratings, test_set$rating))
})
```

Then we plotted our RMSE values versus lambda. We can see from the graph below that lambda is at a minimum just below 5.0.



Next we calculated the lambda which minimized RMSE.

```
# Calculate lambda which minimizes RMSE
lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 4.75
```

2.7 Testing Model on Validation Set

Finally, we tested our regularized model with the optimized lambda on the validation set.

```
# Calculate RMSE on model with optimized lambda
final_rmse <- sapply(lambda, function(l){
 # Calculate mu
   mu <- mean(mlens$rating)

   # Regularized movie effect
   b_i <- mlens %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

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

   # Regularized age effect
   b_a <- mlens %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by='userId') %>%
    group_by(age) %>%
    summarize(b_a = sum(rating - b_u - b_i - mu)/(n()+l))

    # Regularized genre effect
   b_g <- mlens %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by='userId') %>%
    left_join(b_a, by='age') %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_a - b_u - b_i - mu)/(n()+l))

   # Predict ratings for validation set
  predicted_ratings <- val %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_a, by='age') %>%
    left_join(b_g, by='genres') %>%
    mutate(pred = mu + b_i + b_u + b_a + b_g) %>%
    pull(pred)

  return(RMSE(val$rating, predicted_ratings))

})
```

3. Results

The table below shows the RMSE of each model. We can see that incorporating the effects of movie, user, age, and genre each decreased the RMSE of the model, indicating improved accuracy. Implementing regularization on the model further improved accuracy. When we applied our final regularized model to the validation set the RMSE was 0.8639820.

Table 11: RMSE Results

Method	RMSE
1: Naive Model	1.0607045
2: Movie Effect Model	0.9437144
3: Movie + User Effect Model	0.8661625
4: Movie + User + Age Effect Model	0.8656935
5: Movie + User + Age + Genre Effect Model	0.8653605
6: Regularized Movie + User + Age + Genre Effect Model	0.8647431
VALIDATION SET: Regularized Movie + User + Age + Genre Effect Model	0.8639820

4. Conclusion

In this project, we built a movie recommender system using the Movielens 10M dataset. Our final model included movie, user, age, and genre effects as well as penalized least-squares regression. The model performed well on the validation set.

One limitation was the way genre was modeled. Because each movie's genre was comprised of multiple genre categories, each unique combination of genre categories was counted as its own genre. In future work, we could model each genre category independently to try to extract more information from this variable.

In future work, implementing matrix factorization and Principal Components Analysis could potentially reduce the RMSE even further and therefore improve the accuracy of predictions.