Movie Recommendation System | Machine Learning Project

Francisco Arzadon

2022-09-26

I. Introduction

Movie streaming sites has been famous on the general public nowadays, one of the features that these sites should have is an algorithm to recommend users possible movies that they would want to watch. This project aims to create the most effective system to predict movie preferences based on the users' and movies' characteristics. This involves the formation of machine learning algorithm that will assign values to movie entries with missing ratings To create this system, a dataset called movielens is used, specifically the 10M version of this data which contains 10 million observations. The goal is to reach a Root Mean Square Error (RMSE) less than 0.86490 (RMSE < 0.86490) in predicting ratings using data on users, movies, and genres.

Certain Machine Learning techniques are applied, such as the mean as prediction (i.e., Just the Average Method), linear regression and regularization. Based on the results from each method, the recommendation is to combine linear regression with controls on users, movies, and genres then adjust using the regularization method. Including control variables address the bias, and then the regularization method reduces the effect of outliers. With this system, the calculated RMSE using the validation set is equal to 0.8648402.

This report includes the detailed explanation of the methodology, followed by the results from the models and concluding remarks.

II. Methodology

To explain the process to form the algorithm, this section first provides the information on cleaning the data to form the "edx" dataset and the validation dataset. The data distribution is then characterized. Lastly, the codes used for the methods are included. R Studio is used for the data processing, and the following sections contain the actual R script.

1. **Library Initialization and Data wrangling**

This part of the code installs, and loads packages that will be used in the data analysis, and machine learning methods. This also includes downloading of the 10 million observations included in the movielens dataset. The data is also saved into a dataset variable with the naming and correcting datatype for specific columns; the dataset is named "movielens".

Further, the edx dataset contains 90% of a random sample from the movielens dataset. The edx dataset is mainly used to train the algorithm. But to officially test the model, the remaining 10% from the movielens dataset (those that were not included in the edx dataset) were stored as the validation set.

From the edx dataset, the train set and test set are then generated. The train set contains 90% of the edx dataset, while the test set contains 10%.

Lastly, A function called "RMSE" is generated to ease the computation for each model.

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

# Note: this process could take a couple of minutes

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

#initializing packages needed
library(tidyverse)
library(caret)
library(data.table)

#initialize dl for file address of future downloaded temp file
dl <- tempfile()

#downloading MovieLens 10M dataset and saving it in dl variable
download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

#data wrangling from ratings.dat from the downloaded file, with column names userId, movieId, rating, a
ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

#data wrangling from movies.dat from the downloaded file, with column names movieId, title, and genres
movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")


#converting movies to a dataframe, making movieId as numeric, title as character, and genres as charact
movies <- as.tibble(movies) %>% mutate(movieId = as.numeric(movieId),
                                       title = as.character(title),
                                       genres = as.character(genres))
```

```
## Warning: `as.tibble()` was deprecated in tibble 2.0.0.
## Please use `as_tibble()` instead.
## The signature and semantics have changed, see `?as_tibble`.
```

```r
#combining data from movies, and data from ratings by movieId
movielens <- left_join(ratings, movies, by = "movieId")

# Validation set will be 10% of MovieLens data
#setup for creating validation set that is comprised of 10% of 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
```

```r
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
#creation of edx dataset
```

```r
edx <- movielens[-test_index,]
#creation of temp dataset for validation
temp <- movielens[test_index,]

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

# Add rows removed from validation set back into edx set
#create edx dataset combining existing and the removed data
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

#remove variables not needed in training
rm(dl, ratings, movies, test_index, temp, movielens, removed)


#creation of train set and test set
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
```

```r
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
#creation of train_set dataset used for training
train_set <- edx[-test_index,]
#creation of temp dataset for test_set
temp <- edx[test_index,]

# Make sure userId and movieId in test set are also in train set
#finish creation of test_set
test_set <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from test_set set back into train_set
#create train_set dataset used for training combining existing and the removed data
removed <- anti_join(temp, validation)
train_set <- rbind(train_set, removed)

#remove variables not needed in training
rm(test_index, temp, removed)

#RMSE function to test RMSE comparing true ratings from predicted ratings
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

2. **Distribution of Data**

After running the codes for data cleaning, the edx dataset contains 9 million observations and 6 columns. Column names are:

- userId - an identification variable for users;
- movieId - and identification variable for movies;
- rating - a categorical variable to indicate the existing rating of users per movie, with values from 0.5 to 5.0 (and itervals of 0.5);
- timestamp - a variable indicating the period of rating of each user per movie;
- title - a categorical variable to indicate movie titles, and;
- genres - a categorical variable related to the genre/s of each movie.

Each observation consists of the rating of a specific movie, title, movie id, user id, genre, and timestamp. The data type for each column is presented in the following output:

```
str(edx)
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Ad
##  - attr(*, ".internal.selfref")=<externalptr>
```

This output shows the distinct number of users, movies, and genres in the edx dataset, which is 69,878 users, 10,677 movies, and 797 genres:

```
edx %>% summarize(user_count = n_distinct(userId), movie_count = n_distinct(movieId), genre_count = n_di
```

```
##   user_count movie_count genre_count
## 1      69878       10677         797
```

The following charts provide information on the distribution of data by cecrtain categories.

Figure 1 provides the ratings and count for each rating category. This shows that the rating of 4 has the highest frequency. Further, ratings with "0.5" (i.e., 1.5, 2.5, 3.5, 4.5) seem to have less frequency compared to their integer counterpart

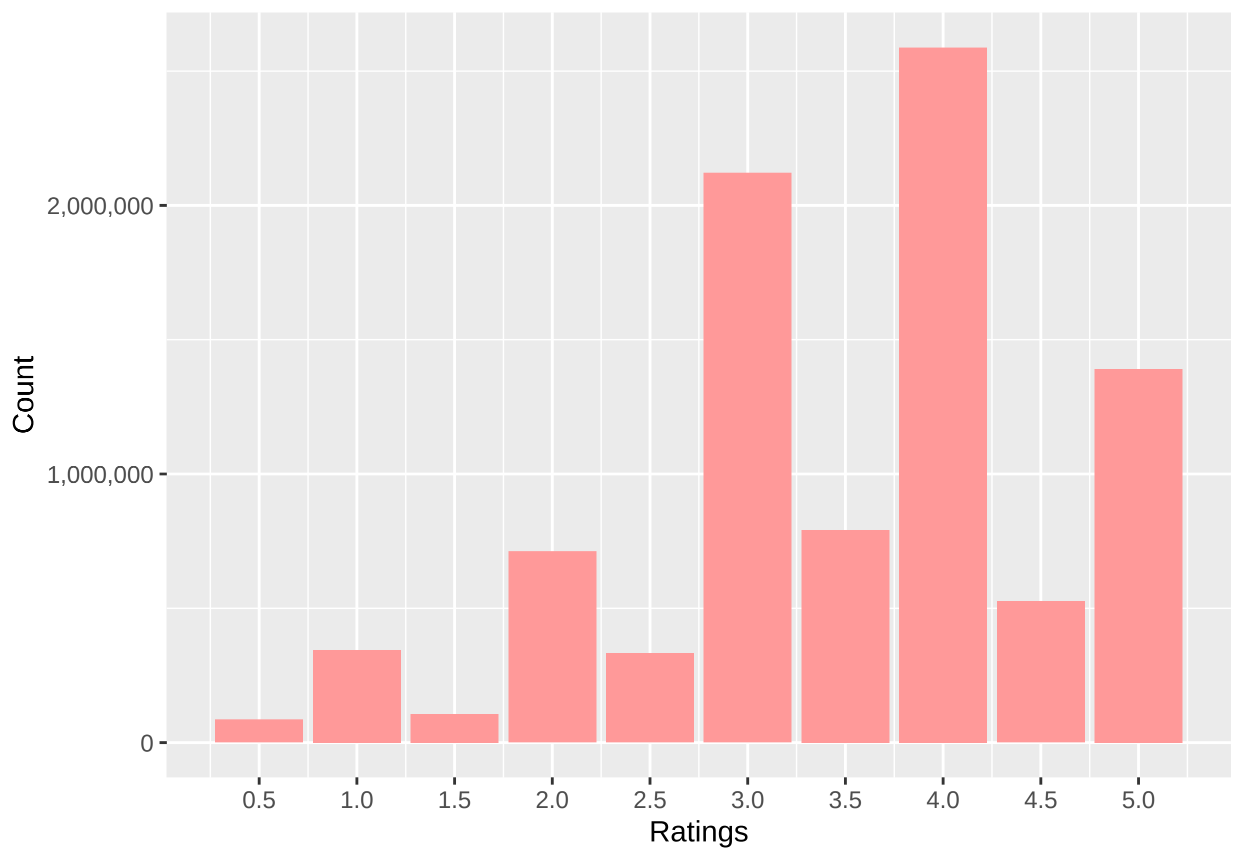

figure 1. Total observations per rating

Figure 2 shows the top 20 movies with their equivalent total of rating entries. Further, Figure 3 and 4 provide observations with very little count per movie title and rating. These figures help in determining the possible outliers of the dataset.

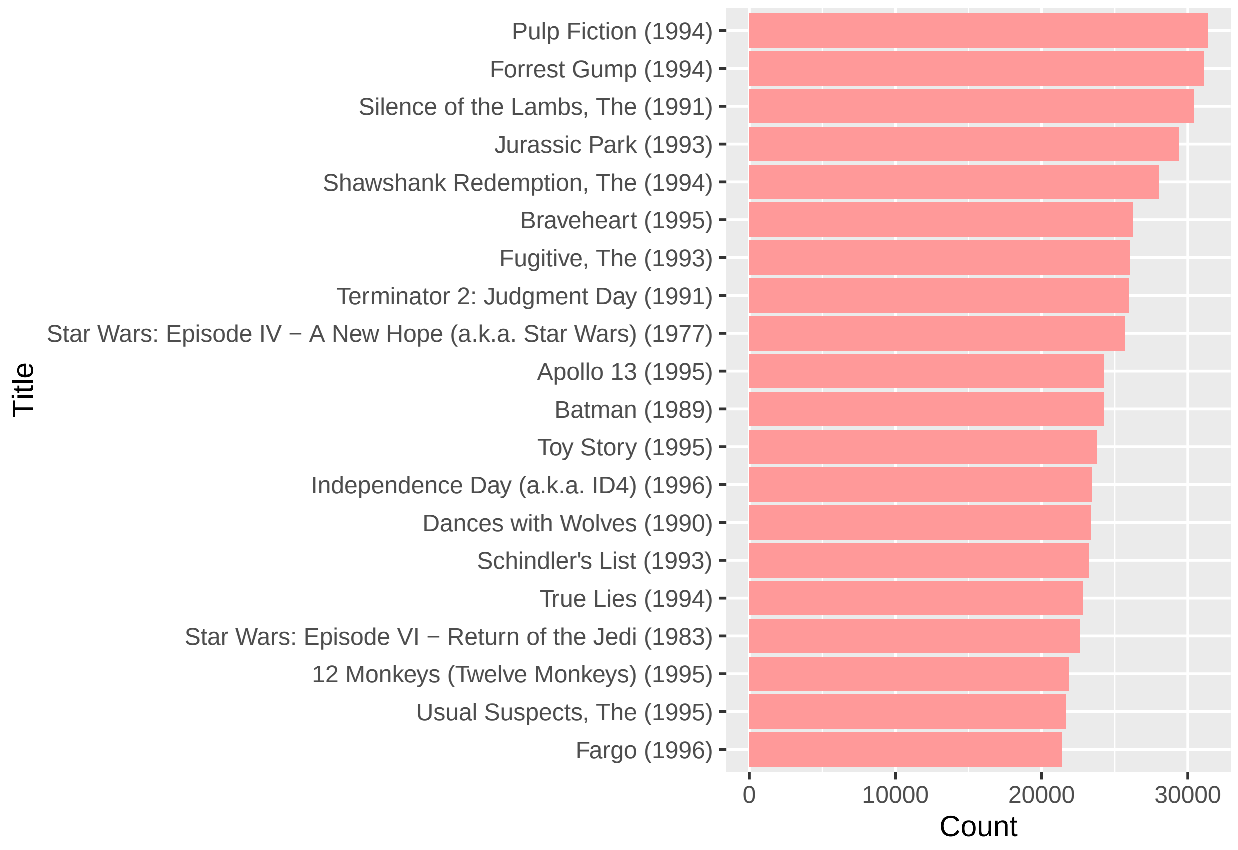

figure 2. Total of rating entries per movie title of the top 20 movies

```
## # A tibble: 17 x 3
##    title                                                avgrating count
##    <chr>                                                    <dbl> <int>
##  1 Accused (Anklaget) (2005)                                0.5       1
##  2 Besotted (2001)                                          0.5       2
##  3 Confessions of a Superhero (2007)                        0.5       1
##  4 Hi-Line, The (1999)                                      0.5       1
##  5 War of the Worlds 2: The Next Wave (2008)                0.5       2
##  6 SuperBabies: Baby Geniuses 2 (2004)                      0.795    56
##  7 Hip Hop Witch, Da (2000)                                 0.821    14
##  8 Disaster Movie (2008)                                    0.859    32
##  9 From Justin to Kelly (2003)                              0.902   199
## 10 Criminals (1996)                                         1         2
## 11 Dischord (2001)                                          1         1
## 12 Dog Run (1996)                                           1         1
## 13 Monkey's Tale, A (Les ChÃ¢teau des singes) (1999)        1         1
## 14 Mountain Eagle, The (1926)                               1         2
## 15 Relative Strangers (2006)                                1         1
## 16 Stacy's Knights (1982)                                   1         1
## 17 When Time Ran Out... (a.k.a. The Day the World Ended) (1980)     1         1
```

figure 3. Showing the movies with top ratings and equivalent total rating entries

```
## # A tibble: 10 x 3
##    title                                                         avgra~1 count
##    <chr>                                                           <dbl> <int>
##  1 Constantine's Sword (2007)                                       4.75     2
##  2 Human Condition II, The (Ningen no joken II) (1959)              4.75     4
##  3 Human Condition III, The (Ningen no joken III) (1961)            4.75     4
##  4 Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to~  4.75     4
##  5 Blue Light, The (Das Blaue Licht) (1932)                         5        1
##  6 Fighting Elegy (Kenka erejii) (1966)                             5        1
##  7 Hellhounds on My Trail (1999)                                    5        1
##  8 Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)                              5        2
##  9 Shadows of Forgotten Ancestors (1964)                            5        1
## 10 Sun Alley (Sonnenallee) (1999)                                   5        1
## # ... with abbreviated variable name 1: avgrating
```

figure 4. Showing the movies with the lowest rating and equivalent total rating entries

3. **Models**

Prior to arriving at the final recommended algorithm, the following models are tested:

1. Just Everything Model -a mean prediction model;
2. Movie Mean Effect Model - a linear regression with control on movies model;
3. Movie and User Mean Effect Model - a linear regression with control on movies, and users model;
4. Movie, User, and Genre Mean Effect Model - a linear regression with control on movies, users, and genres model
5. Movie, User and Genre Effects with Regularized Movies Model - a linear regression with control on movies, users, and genres, combined with regularized of movies model

The next section provides the results of the data processing for each model. By recommended algorithm is based on the model with the highest RMSE.

III. Results

In order to get the efficiency of each model, the train set from the edx dataset is used to run the algorithms. Results are provided for each model.

At the end, the recommended model is the combination of linear regression (with control on movies, users, and genres) and regularization (with control on movies). The final RMSE is computed using the validation set.

- **Model 1 - Just Everything Model**

  For model 1 the approach used is to predict by using only the mean of all ratings present in the dataset. The variable used is m_all for the mean of all the ratings, this method is then named "Just Average of Everything Model". This can still be improved because it only resulted with RMSE of 1.060056.

```r
## Model 1 ##

#average rating of all movies for all users
m_all <- mean(train_set$rating)
#m_all = mean of the ratings for all movies and all users

#rmse for average of everything
all_mean_rmse <- RMSE(test_set$rating, m_all)

#save rmse to rmse_results tibble
rmse_results <- tibble(method = "Just Average of Everything Model", RMSE = all_mean_rmse)

#display rmse results for models
rmse_results %>% knitr::kable()
```

method	RMSE
Just Average of Everything Model	1.060056

- **Model 2 - Movie Mean Effect Model**

Improving on the first model, now taking account for the movie bias for predicting ratings. Good movies are typically rated higher compared to others, the same for bad movies having low ratings. The least squares estimate is just the average of:

(rating - m_all)

then stored in movie_mean. The movie bias (m_movie) is combined with m_all it is then used to predict ratings. The method is named "Movie Mean Effect Model", this can still be improved because the RMSE for this method is 0.9417151

```r
## Model 2 ##

#average rating for each movie
movie_mean <- train_set %>%
  group_by(movieId) %>%
  summarize(m_movie = mean(rating - m_all))
#m_movie = mean of the differences between mean of the ratings for each movie and m_all (refer above)

#making rating predictions based on mean rating of everything, and each movies
predicted_ratings <- test_set %>%
  left_join(movie_mean, by = "movieId") %>%
  mutate(pred = m_all + m_movie) %>%
  .$pred


#get rmse for movie mean predicted ratings
movie_mean_rmse <- RMSE(test_set$rating, predicted_ratings)

#save rmse to rmse results
rmse_results <- bind_rows(rmse_results,
                          tibble(method = "Movie Mean Effect Model",
                                 RMSE = movie_mean_rmse ))
```

```r
#display rmse results for models so far
rmse_results %>% knitr::kable()
```

method	RMSE
Just Average of Everything Model	1.0600561
Movie Mean Effect Model	0.9417151

- **Model 3 - Movie and User Mean Effect Model**

Improving on the second model, after taking account movie bias now taking account for the user bias on predicting ratings. Some users can be more generous in giving ratings on movies, while others are more selfish in giving ratings. The least squares estimate is just the average of:

(rating – m_all – m_movie)

which is stored in variable user_mean. With the user bias (m_user) combined with m_all and m_movie it is then used to predict ratings. The method is named "Movie and User Mean Effect Model", this can still be improved because the RMSE for this method is 0.8558163.

```r
## Model 3 ##

#mean rating for each user
user_mean <- train_set %>%
  left_join(movie_mean, by = "movieId") %>%
  group_by(userId) %>%
  summarize(m_user = mean(rating - m_all - m_movie))

#predicted ratings based on mean of everything, per movie, and per user
predicted_ratings <- test_set %>%
  left_join(movie_mean, by = "movieId") %>% left_join(user_mean, by = "userId") %>%
  mutate(pred = m_all + m_movie + m_user) %>%
  .$pred

#rmse computation for the predicted ratings
user_mean_rmse <- RMSE(test_set$rating, predicted_ratings)

#saving rmse result in tibble
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie and User Mean Effect Model",
                                 RMSE = user_mean_rmse ))

#display rmse results of all models so far
rmse_results %>% knitr::kable()
```

method	RMSE
Just Average of Everything Model	1.0600561
Movie Mean Effect Model	0.9417151
Movie and User Mean Effect Model	0.8558163

- **Model 4 - Movie, User, and Genre Mean Effect Model**

Improving on the third model, after taking account movie bias, and user bias now taking account for the genre bias on predicting ratings. Some genres are rated higher compared to other genres. The least squares estimate is just the average of:

(rating – m_all – m_movie – m_user)

which is stored in variable genre_mean. Combined with m_all, m_movie, m_user, and m_genre it is then used to predict ratings. The method is named "Movie, User, and Genre Mean Effect Model", this can still be improved because the RMSE for this method is 0.8554599.

```r
## Model 4 ##

#average mean for each genre
genre_mean <- train_set %>%
  left_join(movie_mean, by = "movieId") %>% left_join(user_mean, by = "userId") %>%
  group_by(genres) %>%
  summarize(m_genre = mean(rating - m_all - m_movie - m_user))

#predicted ratings based on mean of everything, per movie, per user, and per genre
predicted_ratings <- test_set %>%
  left_join(movie_mean, by = "movieId") %>%
  left_join(user_mean, by = "userId") %>%
  left_join(genre_mean, by = "genres") %>%
  mutate(pred = m_all + m_movie + m_user + m_genre) %>%
  .$pred

#rmse computation for the predicted ratings
genre_mean_rmse <- RMSE(test_set$rating, predicted_ratings)

#saving rmse to rmse results
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie, User, and Genre Mean Effect Model",
                                 RMSE = genre_mean_rmse ))

#displaying rmse of all models so far
rmse_results %>% knitr::kable()
```

method	RMSE
Just Average of Everything Model	1.0600561
Movie Mean Effect Model	0.9417151
Movie and User Mean Effect Model	0.8558163
Movie, User, and Genre Mean Effect Model	0.8554599

- **Model 5 - Movie, User, and Genre Mean Effects with Regularized Movies Model**

As seen in figure 3 and figure 4, most of the movies with the highest and lowest ratings have few rating count. Using the previous models with regularization on movies to improve results. A function is made to find the best tuning parameter (lambda) by testing a sequence and returns the value of least RMSE. Figure 8 is showing the graph of lambdas vs RMSEs and it shows that the value of lambda with the least RMSE is 1.75. The method is named "Movie, User and Genre Effects with Regularized Movies Model" and It resulted with 0.8554392 RMSE.

```r
## Model 5 ##

#sequence of lambdas to test
lambdas <- seq(0, 10, 0.25)

#create function to check rmse for lambdas
rmses <- sapply(lambdas, function(l){

  m_all <- mean(train_set$rating)

  #movie mean with regularization
  movie_mean <- train_set %>%
    group_by(movieId) %>%
    summarise(m_movie = sum(rating - m_all)/(n()+l))

  #user mean
  user_mean <- train_set %>%
    left_join(movie_mean, by="movieId") %>%
    group_by(userId) %>%
    summarise(m_user = sum(rating - m_movie - m_all)/(n()))

  #genre mean
  genre_mean <- train_set %>%
    left_join(movie_mean, by = "movieId") %>% left_join(user_mean, by = "userId") %>%
    group_by(genres) %>%
    summarise(m_genre = sum(rating - m_movie - m_user - m_all)/(n()))


  #predicted ratings based on everything, movie mean, user mean, and genre mean with regularization
  predicted_ratings <- test_set %>%
    left_join(movie_mean, by = "movieId") %>% left_join(user_mean, by = "userId") %>% left_join(genre_me
    mutate(pred = m_all + m_movie + m_user + m_genre) %>%
    pull(pred)

  #return rmse
  return(RMSE(predicted_ratings, test_set$rating))

})
```

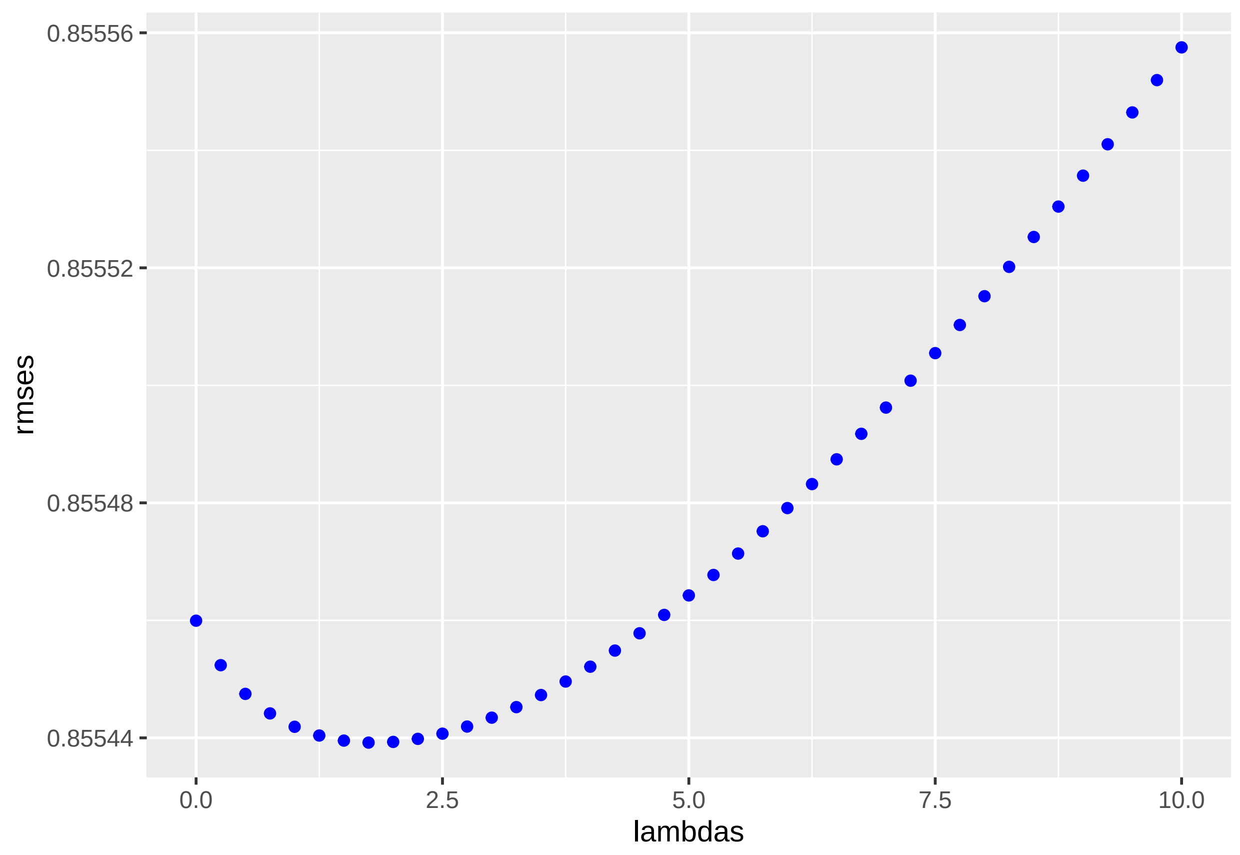

Figure 8. lambdas vs rmses graph showing the lamda used for the best RMSE result

```r
#get minimum rmse from all result of all lambdas for mean with regularization
rmse_regularisation <- min(rmses)
rmse_regularisation
```

```
## [1] 0.8554392
```

```r
#add rmse for regularized movie, user, and genre effects model to rmse_results
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie, User and Genre Effects with Regularized Movies Model",
                                 RMSE = rmse_regularisation))

#display rmse results for all models
rmse_results %>% knitr::kable()
```

method	RMSE
Just Average of Everything Model	1.0600561
Movie Mean Effect Model	0.9417151
Movie and User Mean Effect Model	0.8558163
Movie, User, and Genre Mean Effect Model	0.8554599
Movie, User and Genre Effects with Regularized Movies Model	0.8554392

- **Final Validation**

The final validation uses the fifth model, Movie, User and Genre Effects with Regularized Movies Model, the validation dataset was used to check the final RMSE and it resulted with 0.8648402, which satisfies the target value.

```
### Final Validation ###

#gettingg the lambda that was used for the least RMSE
l <- lambdas[which.min(rmses)]
m_all <- mean(edx$rating)

#movie mean with regularization
movie_mean_reg <- edx %>%
  group_by(movieId) %>%
  summarise(m_movie = sum(rating - m_all)/(n()+l))

#user mean
user_mean_reg <- edx %>%
  left_join(movie_mean_reg, by="movieId") %>%
  group_by(userId) %>%
  summarise(m_user = sum(rating - m_movie - m_all)/(n()))

#genre mean
genre_mean_reg <- edx %>%
  left_join(movie_mean_reg, by = "movieId") %>%
  left_join(user_mean_reg, by = "userId") %>%
  group_by(genres) %>%
  summarise(m_genre = sum(rating - m_movie - m_user - m_all)/(n()))

#predicted ratings based on everything, movie mean, user mean, and genre mean with regularization on mo
predicted_ratings <- validation %>%
  left_join(movie_mean_reg, by = "movieId") %>%
  left_join(user_mean_reg, by = "userId") %>%
  left_join(genre_mean_reg, by = "genres") %>%
  mutate(pred = m_all + m_movie + m_user + m_genre) %>%
  pull(pred)

final_RMSE <- RMSE(validation$rating, predicted_ratings)
#save rmse to rmse_results tibble
rmse_results_validation <- tibble(method = "Movie, User and Genre Effects with Regularized Movies Model

#final rmse
rmse_results_validation %>% knitr::kable()
```

method	RMSE
Movie, User and Genre Effects with Regularized Movies Model	0.8648402

IV. Conclusion

The previous sections provide the steps to create an algorithm for movie rating predictions based on the characteristics of users and movies. After testing a list of models, it is recommended to use the linear regression model that controls the bias of movies, users, and genres, and applies regularization using the movies indicator. This model gives an RMSE of 0.8648402, which is in the target range.

For future study, the algorithm may still be improved with the use of other movie characteristics, such as movie age, featuring actors, and directors. It is also possible to find more efficient results with the use of KNN or Random Forest methods, which can be done with better hardware. Nonetheless, the results of this project provides a good system for the prediction of movie ratings for user recommendations.