MovieLens Project

ltja

10/23/2020

INTRODUCTION

This project, a HarvardX: PH125.9x Capstone: Data Science course requirement, showcases a movie recommendation system I created using the MovieLens dataset.

Just like any other recommendation systems out there, MovieLens' recommendation system and this project's recommendation system rely on the ratings given by users to specific movies. Ratings are given in a scale of 1 to 5, with 1 being the worst and 5 being the best. Rating data is used by companies that sell a variety of products to determine a user's preference and determine the type of products a user may like.

The recommendation system developed in this project is a trained model that is deemed accurate by yielding the least amount of error (RMSE) when the ratings it predicts are tested against actual ratings.

Dataset Description

The original MovieLens data set contains tens of millions of ratings, collected by MovieLens, a movie recommendation system, in a 5-star rating system, across tens of thousands of movies. The original data set itself was created by 283228 users (who were selected at random) between January 09, 1995 and September 26, 2018. Further description of the original MovieLens data set can be found here.

However, for this project, as outlined in the instructions, the 10M version of the MovieLens dataset was used. This version offers a stable benchmark dataset (unlike the original dataset which continuously is being developed). It has 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users. Further description of the 10M dataset can be found here.

Project Goal

The goal of the project is to come up with my own recommendation system, a model selected as the one that produces the most accurate (least RMSE) ratings when trained using the training set and tested against the test set. The chosen model was then tested against the validation set, and the RMSE derived is the final result.

To meet the project's requirement, I started off with the models recommended by the textbook, and **built upon it by adding the genre bias into the model and by regularizing the modified model**.

Steps

The following key steps were performed to explore the data, develop models, and pick the final model:

1. Ran the code provided by the instructor to download the MovieLens data and split it into edx and validation sets.
2. Split the edx dataset into train and test sets; the validation set would not be used until the very end. Then, explored all of the datasets (in terms of dimension, variables presented per column and row, etc.) to gain an understanding of them.

3. Adopted the recommendation system outlined in the textbook (chapter 33.4) as the starting point, i.e., the basic model with movie and user biases taken into account. Trained the models and recorded each model's RMSE. Only the train and test sets were used here.
4. Built a new recommendation systems by adding a feature (i.e., adding the genre bias consideration into the basic model). Trained the model and recorded the model's RMSE. Only the train and test sets were used here.
5. Further built the model by applying an analysis technique (regularization) to the model built in step 4. Trained the model and recorded the model's RMSE. Only the train and test sets were used here.
6. From the exploration steps above, picked the model that yielded the lowest RMSE ("Final Model"). Ran the model using the edx set against the validation set.
7. Print out the final RMSE. End of analysis The final RMSE was derived from the difference between the predicted edx set ratings and the actual validation set ratings. The final model picked was the Regularized Model that has Movie + User + Genre Effects taken into consideration. Final RMSE recorded against the validation set is **0.8644514**.

METHODS / ANALYSIS

The following section outlines the step-by-step process and techniques used to explore the data and insights gained from the exploration, develop models by adding features to make them more accurate, and choose the final model that was tested against the validation set. The code that was used, the results, and any charts that help with data visualization will be presented below. (Note: When code is provided by the instructor or duplicated from the textbook, it will be indicated as so.)

Step 1. Download the dataset

Download the MovieLens data from the source by running the code provided by the instructor. Once the data have been downloaded, read the file, and shape it into a dataframe with column names such as movieId, title, genre, etc. Afterwards, split the MovieLens dataset into edx and validation sets.

The code is as the following.

```r
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")

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

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")

# for 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))
```

```r
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)`
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)
```

Step 2. Prepare and explore the dataset.

Since the validation set is not to be used until the very end, we need to further split the edx dataset into train and test sets. The models will be developed by training the algorithm on the train dataset and their accuracy will be checked against the test dataset.

The test set will be set up to be 10% of edx, and the train set will be 90% of edx. (The 10/90 split is somewhat arbitrary, as there has been different literature advocating for various different ratios.)

Here is the code.

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

# Make sure userId and movieId in test set are also in train set
test_set <- temporary %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

# Add rows removed from test set back into train set
removed2 <- anti_join(temporary, test_set)
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
train_set <- rbind(train_set, removed2)
```

Now that the training and test sets are set up, I explored the datasets to understand them by obtaining information such as, what are the column headers, what is the class of the datasets, what are their dimensions? The following code gave me the picture I was looking for. This type of simple check can also pick up anything that glaringly was wrong (e.g., if the edx and validation sets have different column headers, if the train and test sets have different column headers, if the split is not as what was scripted, etc.)

```r
dim(edx)
## [1] 9000055       6
dim(validation)
## [1] 999999      6
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
class(edx)
## [1] "data.frame"
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

dim(train_set)
## [1] 8100065       6
dim(test_set)
## [1] 899990      6
```

Step 3. Set up the Basic + Movie + User Effects Model

For this project, I adopted the recommendation system outlined in the textbook (chapter 33.7) as the starting point. The code that I added myself to build on the provided code will be noted in the following sections.

The basic model and the subsequent built up models are based on Loss Function, where the goal is to have a model that yields the lowest Residual Mean Squared Error (RMSE) against a test set. The RMSE is defined in the following formula:

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

In the formula above,

$$y_{u,i}$$

is the rating for movie i by user u. The predicted rating is denoted by

$$\hat{y}_{u,i}$$

In the basic model, it is assumed that all movies have the same rating, regardless of the user, and the differences would be explained by random variations. The algorithm looks like this:

$$Y_{u,i} = \mu + \epsilon_{u,i}$$

In the basic model, the RMSE will simply be the difference between the mu_hat and the ratings in the test set. The code to derive the RMSE is below. Please note that the following code is used on the training set against the test set, and validation set is not used at all.

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

# Basic Model RMSE
mu_hat <- mean(train_set$rating)
mu_hat
## [1] 3.512456

naive_rmse <- RMSE(test_set$rating, mu_hat)
naive_rmse
## [1] 1.060054

model_0 <- tibble(method = "Basic", RMSE = naive_rmse)

rmse_results <- model_0

print.data.frame(rmse_results)
##   method     RMSE
## 1  Basic 1.060054
```

The Basic Model yields RMSE of **1.060054** which can be improved.

The textbook went a step further and shows how the Basic Model can be improved by taking into account the fact that different movie is rated differently. This is done by adding the following term to note movie biases:

$$b_i$$

The algorithm becomes:

$$Y_{u,i} = \mu + b_i + \epsilon_{u,i}$$

And the code to figure out the RMSE's of the Basic + Movie Effects Model is below.

```r
# Basic Model + Movie Effects RMSE
mu <- mean(train_set$rating)

movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
movie_avgs
## # A tibble: 10,677 x 2
##    movieId     b_i
##      <dbl>   <dbl>
##  1       1  0.415
##  2       2 -0.306
##  3       3 -0.361
##  4       4 -0.637
##  5       5 -0.442
##  6       6  0.302
##  7       7 -0.152
##  8       8 -0.397
##  9       9 -0.514
## 10      10 -0.0856
## # ... with 10,667 more rows

predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

model_1 <- RMSE(predicted_ratings, test_set$rating)

rmse_results <- bind_rows(rmse_results,
                data_frame(method="Basic+Movie", RMSE = model_1 ))

print.data.frame(rmse_results)
```

```
##        method      RMSE
## 1       Basic 1.0600537
## 2 Basic+Movie 0.9429615
```

The Basic + Movie Effects Model yields RMSE of **0.9429615**. Can it be improved?

After considering the movie effects, the textbook went a step further and shows how the Basic + Movie Effect Model can be improved by taking into account the fact that users also have their own biases that will affect the prediction. So, the user bias should be added into the algorithm:

$$b_u$$

The algorithm becomes:

$$Y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

And the code to figure out the RMSE's of the Basic + Movie + User Effects Model is below.

```
# Basic Model + Movie Effects + User Effects RMSE

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

predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

model_2 <- RMSE(predicted_ratings, test_set$rating)

rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Basic+Movie+User", RMSE = model_2))

print.data.frame(rmse_results)
##             method      RMSE
## 1            Basic 1.0600537
## 2      Basic+Movie 0.9429615
## 3 Basic+Movie+User 0.8646843
```

The Basic + Movie + User Effects Model yields RMSE of **0.8646843**. Can it be improved further?

Step 4. Add the Genre Effects to the Basic Model

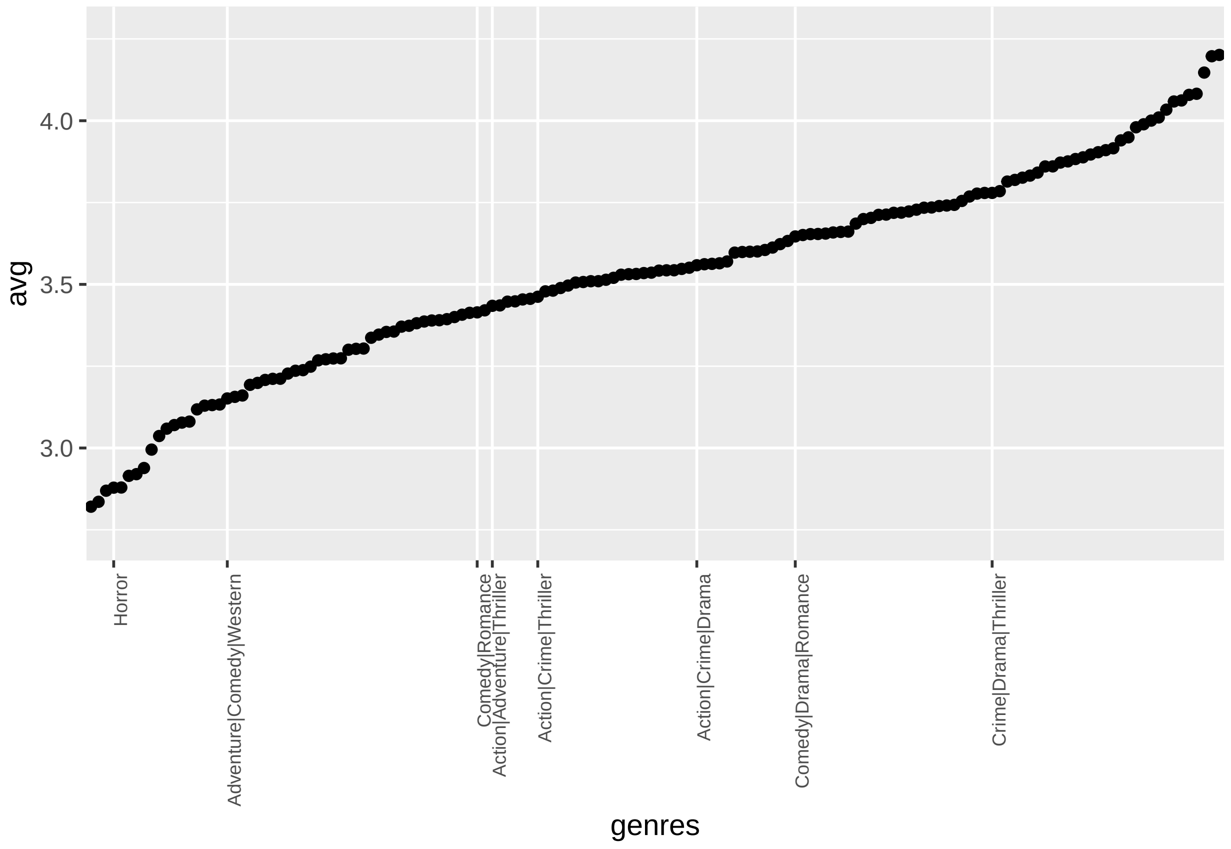
To meet the project requirement, I added a feature to the Basic + Movie + User Effects model by adding the Genre Effects.

Before I chose what feature to add, I looked at the datasets and saw that a rating could be influenced by so many more factors, such as the year the movie was released (did movies get more rating in a certain year?), the genre of the movie (maybe fans of specific genres were more generous with the rating?), the timestamp of the movie (does the day or time a movie is released make a difference?), number of ratings (would the number of rating a movie gets affect the average rating of that movie?), and so on and so forth.

However, due to time constraint, I decided to just add one feature: Genre. To be clear, what I decided to look at is the bias imposed by the different categories of genre, instead of the bias of each genre, noting that Comedy and Comedy|Action are two categories with different biases and Comedy|Action will not be split into two genres.

But, before I moved ahead, I would like to at least see whether Genre categories have substantial effects on the rating. I used the following code to generate the plot for categories of genres that have 10,000 or more ratings:

```
train_set %>% group_by(genres) %>%
  summarize(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
  filter(n >= 10000) %>%
  mutate(genres = reorder(genres, avg)) %>%
  ggplot(aes(x = genres, y = avg, ymin = avg - 10*se, ymax = avg + 10*se)) +
  geom_point() +
  scale_x_discrete(breaks = train_set$genres[c(T,rep(F,1000000))]) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1, size=7))
```



From the above, it shows that categories of genres have effects on the ratings, so I moved ahead and added that into the model. (Please note that to avoid crowding on the x-axis, only select labels were printed on the plot above.)

The algorithm that includes genre (HarvardX PH125.8xData Science: Machine Learning) is per the following:

$$Y_{u,i} = \mu + b_i + b_u + \sum_{k=1}^{K} x_{u,i}\beta_k + \epsilon_{u,i}$$

However, the algorithm above takes each genre, instead of each category, into consideration.

As such, the algorithm that I'll use instead is:

$$Y_{u,i} = \mu + b_i + b_u + b_g + \epsilon_{u,i}$$

The following is the code that I built to take categories' biases into the model. Please note that only the train and test sets were used here:

```
genre_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u))

predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by= 'genres') %>%
  mutate(pred = mu + b_i + b_u +b_g) %>%
  pull(pred)

model_3 <- RMSE(predicted_ratings, test_set$rating)

rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Basic+Movie+User+Genre", RMSE = model_3))

print.data.frame(rmse_results)
##                  method      RMSE
## 1                 Basic 1.0600537
## 2           Basic+Movie 0.9429615
## 3      Basic+Movie+User 0.8646843
## 4 Basic+Movie+User+Genre 0.8643241
```

Adding the category of genre effects into consideration improves the RMSE to **0.8643241**. What else can be done to improve it further?

Step 5. Regularize the Basic + Movie + User + Genre Effects Model

One technique I added into the model is regularization, which basically penalizes the data points that have high variability.

In order to regularize the Basic + Movie + User + Genre Effects Model developed above, first derive the best lambda using cross-validation over a sequence of lambdas. Please note that only the train and test sets are used to get the best lambda. The code is below.

```
# Find the most effective lambda for the model

lambdas <- seq(0, 10, 0.25)

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

  mu <- mean(train_set$rating)

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

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

  b_g <- train_set %>%
    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 <-
    test_set %>%
    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) %>%
    pull(pred)

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

rmses
##  [1] 0.8643241 0.8642562 0.8641973 0.8641451 0.8640984 0.8640565 0.8640189
##  [8] 0.8639852 0.8639553 0.8639288 0.8639055 0.8638852 0.8638678 0.8638531
## [15] 0.8638409 0.8638312 0.8638238 0.8638185 0.8638153 0.8638141 0.8638148
## [22] 0.8638172 0.8638214 0.8638272 0.8638346 0.8638434 0.8638537 0.8638654
## [29] 0.8638783 0.8638925 0.8639079 0.8639245 0.8639421 0.8639609 0.8639806
## [36] 0.8640013 0.8640229 0.8640455 0.8640689 0.8640931 0.8641181

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

lambda
## [1] 4.75
```

From the above, we can see that the best lambda is 4.75. The lambda is then used to regularize the Basic + Movie + User + Genre Effects Model and get its RMSE. Please note that the RMSE is derived using the train and test sets. The code is below.

```
lambda <- 4.75

mu <- mean(train_set$rating)

movie_reg_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())

user_reg_avgs <- train_set %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_i)/(n()+lambda), n_u = n())

genre_reg_avgs <- train_set %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  left_join(user_reg_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu - b_i - b_u)/(n()+lambda), n_g = n())

predicted_ratings <- test_set %>%
  left_join(movie_reg_avgs, by="movieId") %>%
  left_join(user_reg_avgs, by="userId") %>%
  left_join(genre_reg_avgs, by="genres") %>%
  mutate(pred = mu + b_i + b_u +b_g) %>%
  pull(pred)

model_5 <- RMSE(predicted_ratings, test_set$rating)

rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Basic+Movie+User+Genre", RMSE = model_5))

print.data.frame(rmse_results)
##                               method      RMSE
## 1                              Basic 1.0600537
## 2                       Basic+Movie 0.9429615
## 3                  Basic+Movie+User 0.8646843
## 4            Basic+Movie+User+Genre 0.8643241
## 5 Regularized Basic+Movie+User+Genre 0.8638141
```

The regularization over the modified model yields RMSE of **0.8638141**, which is the lowest out of all of the models reviewed so far. So, this is the model that will be tested against the validation set.

Step 6. Pick a final model to be run agains the validation set

The Regularized Basic + Movie + User + Genre Effects Model will now be run on edx set and tested against the validation set. We'll also use the lambda derived using cross validation between training and test sets, since both sets make up edx. Here is the code:

```
lambda <- 4.75

mu <- mean(edx$rating)

movie_reg_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())

user_reg_avgs <- edx %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_i)/(n()+lambda), n_u = n())

genre_reg_avgs <- edx %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  left_join(user_reg_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu - b_i - b_u)/(n()+lambda), n_g = n())

predicted_ratings <- validation %>%
  left_join(movie_reg_avgs, by="movieId") %>%
  left_join(user_reg_avgs, by="userId") %>%
  left_join(genre_reg_avgs, by="genres") %>%
  mutate(pred = mu + b_i + b_u +b_g) %>%
  pull(pred)

model_6 <- RMSE(predicted_ratings, validation$rating)
model_6
## [1] 0.8644514

rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Final Model and Validation", RMSE = model_6))

print.data.frame(rmse_results)
##                               method      RMSE
## 1                              Basic 1.0600537
## 2                        Basic+Movie 0.9429615
## 3                   Basic+Movie+User 0.8646843
## 4             Basic+Movie+User+Genre 0.8643241
## 5 Regularized Basic+Movie+User+Genre 0.8638141
## 6         Final Model and Validation 0.8644514
```

Step 7. Print the final RMSE. End of Analysis.

The final model picked was the Regularized Model that has Movie + User + Genre Effects taken into consideration. The final RMSE was derived from the difference between the predicted edx set ratings and the actual validation set ratings. Final RMSE recorded against the validation set is **0.8644514**.

RESULTS

As the models and their results were analyzed and reviewed above, this section presents a summary of each model's RMSE.

```
print.data.frame(rmse_results)
##                               method      RMSE
## 1                              Basic 1.0600537
## 2                        Basic+Movie 0.9429615
## 3                   Basic+Movie+User 0.8646843
## 4             Basic+Movie+User+Genre 0.8643241
## 5 Regularized Basic+Movie+User+Genre 0.8638141
## 6         Final Model and Validation 0.8644514
```

The Final Model RMSE result of **0.8644514** is below the lowest treshold for this project outlined by the instructor. Therefore, considered fairly accurate. (Side note: Considering that the Netflix challenge winner achieved a score of 0.8563 on the complete dataset, the RMSE derived using the Loss function is in the vicinity of accuracy.)

CONCLUSION

To summarize the report, the project's goal of developing a system that yields a model with low RMSE (i.e., accurate rating) was achieved by adding features to the model - Features such as taking *biases* (movie effects, user effects, and genre effects) into consideration and *regularization* to reduce variability.

Limitation and future work: The analysis does not take into consideration the beta of each specific genre, and other possible sources of biases (such as year of releases). The analysis also adopted the simplified approach of cross-validation. As such, were this project to be revisited, there are a number of avenues to further explore and improve the RMSE by reviewing additional sources of biases and expanding the cross validation technique by using k-fold cross validation, in addition to exploring the numerous other techniques available out there (matrix factorization, neighborhood models, and so on and so forth).

With that said, the accuracy level derived by the Final Model is sufficiently accurate for the dataset.

REFERENCES

1. Irizarry, Rafael A., "Introduction to Data Science," online version https://rafalab.github.io/dsbook/
2. Chen, Edwin, "Winning the Netflix Prize: A Summary," http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/
3. MovieLens Dataset, https://grouplens.org/datasets/movielens/10m/