Movielens_Project

Shane Pinto

2023-04-23

Table of Contents

Introduction and Overview

The aim of this project is to create a recommendation system used by many streaming services today like Amazon, Netflix, Spotify and Apple to try and recommend a movie, TV show, music, pod-casts etc based on user ratings. These recommendation systems help the streaming services showcase their content to better match a user with the media they enjoy with the hope that the user is enticed to stay subscribed to their network.

This project is based on the "Netflix Challenge" issued by Netflix in 2006 with the hope of improving their recommendation system by 10% and win 1 million dollars! Just like in the Netflix challenge, we will use the "Root Mean Square Error" or RMSE for short to test the findings of our systems to understand the results we achieve. The RMSE is a measure of the differences between the results obtained VS the actual results. Lower the RMSE value, the more successful the recommended system is.

We use the "movielens" data-set provided by the course to represent the sample data of users and use this data to carry out our analysis, interpret results and finally draw some conclusions.

Creating the "Test" and "Training" sets for data analysis.

```
# Create edx and final_holdout_test sets
# Note: this process could take a couple of minutes

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")

library(tidyverse)
library(caret)

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

options(timeout = 120)

dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip",
dl)

ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"),
simplify = TRUE),
                         stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")

ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"),
simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
```

```r
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

movielens <- left_join(ratings, movies, by = "movieId")

# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later

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

test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1,
list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)

## Joining with `by = join_by(userId, movieId, rating, timestamp, title,
genres)`

edx <- rbind(edx, removed)

rm(dl,ratings_file, movies_file)
```

Analysis of the dataset.

All analysis will be carried out on our test set(edx) only with the final_holdout_test set used at the end to validate our findings.

Initial analysis of the test sets reveals approx. 9000000 rows of data with 6 variables namely, “userID”,“movieID”,“rating”,“timestamp”,“title” and “genres” with all fields having valid data.

```
# Analysing the dataset.
head(edx,5)
```

```
##   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)
##                           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
```

```
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
##
##
##
```

```
# Confirming if variables have usable data
sum(is.na.data.frame(edx))
```

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

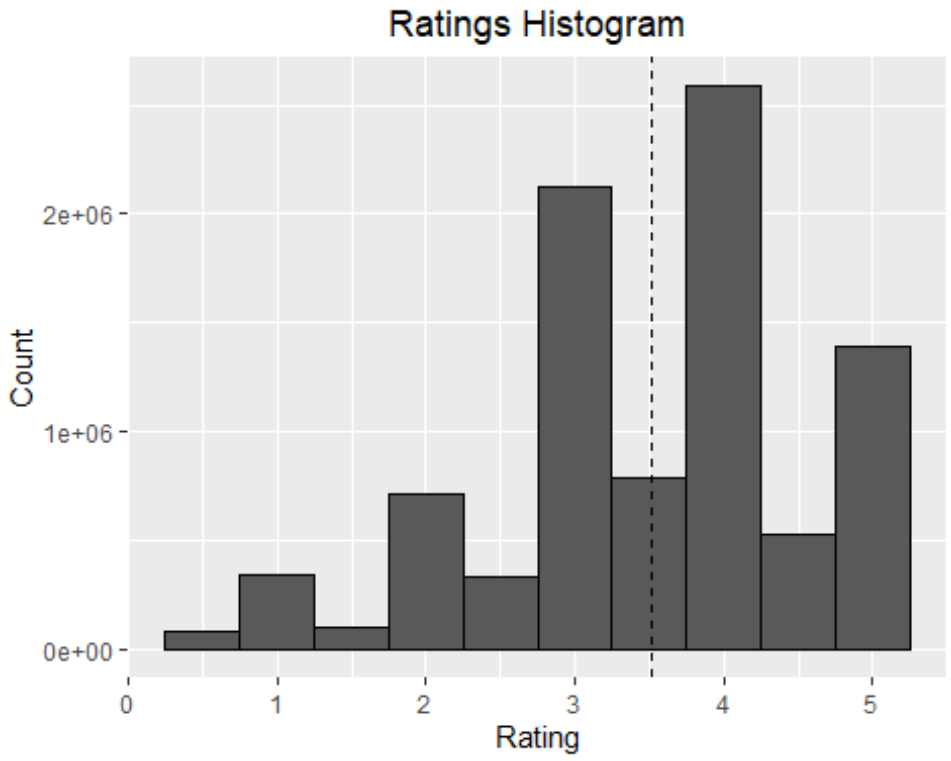
The dataset contains approx. 11000 movies, 70000 users and 800 unique genres.

```
# Review the number of distinct movies, users and genres.
edx %>% summarise(
  uniq_movies = n_distinct(movieId),
  uniq_users = n_distinct(userId),
  uniq_genres = n_distinct(genres))

##   uniq_movies uniq_users uniq_genres
## 1       10677      69878         797
```

The "ratings" variable.

We first investigate the ratings variable to understand how users have rated movies. The data reveals that a rating of "4" is the most popular and "0.5" being the least and the average rating across the dataset is around 3.5. We also notice that users prefer to rate movies with whole numbers like 1,2,3 etc rather 0.5,1.5,2.5 etc.

```
# Total count of ratings from highest to lowest
edx %>% group_by(rating) %>%
        summarise(Ratings_Count = n()) %>%
        arrange(desc(Ratings_Count))

## # A tibble: 10 × 2
##    rating Ratings_Count
##     <dbl>         <int>
##  1    4         2588430
##  2    3         2121240
##  3    5         1390114
##  4    3.5        791624
##  5    2          711422
##  6    4.5        526736
##  7    1          345679
##  8    2.5        333010
##  9    1.5        106426
## 10    0.5         85374

# Histogram of the spread of Ratings
edx %>% ggplot(aes(rating)) +
  geom_histogram(binwidth = 0.5, color = "black") +
  geom_vline(xintercept = mean(edx$rating), lty = 2) +
  xlab("Rating") +
  ylab("Count") +
  ggtitle("Ratings Histogram") +
  theme(plot.title = element_text(hjust = 0.5))
```



The "userID" variable.

Next, we try to understand the "userID" variable. This reveals that most users can rate anywhere between 10 to an excess of a 1000 movies and on an average each user rates about 130 movies.

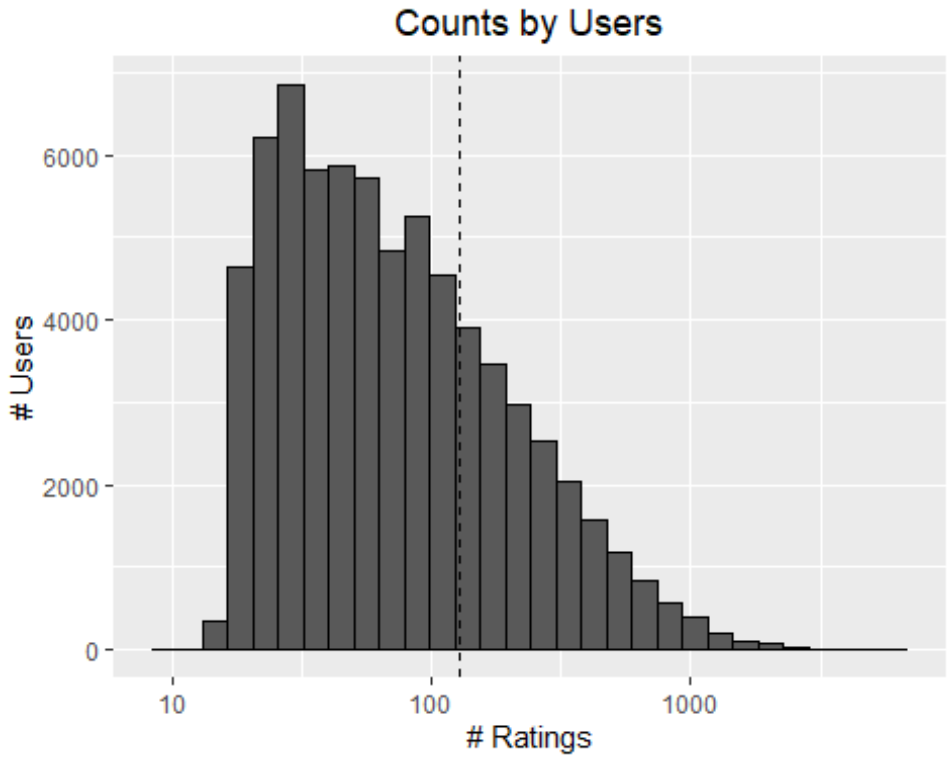
```
# View top 5 users by number of ratings
edx %>% group_by(userId) %>%
    summarise(User_Counts = n()) %>%
    arrange(desc(User_Counts)) %>%
    head(5)
```

```
## # A tibble: 5 × 2
##   userId User_Counts
##    <int>       <int>
## 1  59269        6616
## 2  67385        6360
## 3  14463        4648
## 4  68259        4036
## 5  27468        4023
```

```
# View bottom 5 users by number of ratings
edx %>% group_by(userId) %>%
    summarise(User_Counts = n()) %>%
    arrange(desc(User_Counts)) %>%
    tail(5)
```

```
## # A tibble: 5 × 2
##   userId User_Counts
##    <int>       <int>
## 1  71344          14
## 2  15719          13
## 3  50608          13
## 4  22170          12
## 5  62516          10
```

```
# Histogram of total counts of Ratings by Users
edx %>% count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(color = "black", bins=30) +
  geom_vline(xintercept = (nrow(edx)/n_distinct(edx$userId)), lty = 2) +
  scale_x_log10() +
  xlab("# Ratings") +
  ylab("# Users") +
  ggtitle("Counts by Users") +
  theme(plot.title = element_text(hjust = 0.5))
```



The "moviesID" variable.

We now investigate "moviesID" variable and notice that some movies get rated more than others. This is to be expected depending on the popularity of the cast along with big franchisees and big budget movies.

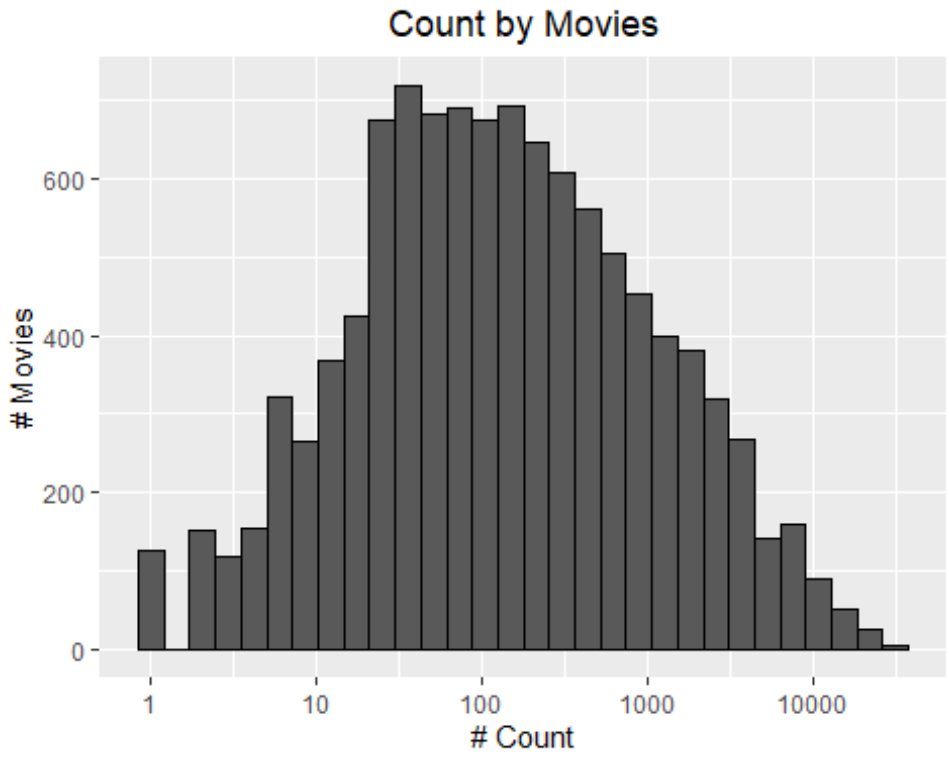
```r
# View top 5 movies by number of ratings
edx %>% group_by(title) %>%
    summarise(Movies_Count = n()) %>%
    arrange(desc(Movies_Count)) %>%
    head(5)
```

```
## # A tibble: 5 × 2
##   title                              Movies_Count
##   <chr>                                     <int>
## 1 Pulp Fiction (1994)                       31362
## 2 Forrest Gump (1994)                       31079
## 3 Silence of the Lambs, The (1991)          30382
## 4 Jurassic Park (1993)                      29360
## 5 Shawshank Redemption, The (1994)          28015
```

```r
# View bottom 5 movies by number of ratings
edx %>% group_by(title) %>%
    summarise(Movies_Count = n()) %>%
    arrange(desc(Movies_Count)) %>%
    tail(5)
```

```
## # A tibble: 5 × 2
##   title                                                          Movies_Count
##   <chr>                                                                 <int>
## 1 When Time Ran Out... (a.k.a. The Day the World Ended) (1980)              1
## 2 Where A Good Man Goes (Joi gin a long) (1999)                             1
## 3 Won't Anybody Listen? (2000)                                              1
## 4 Young Unknowns, The (2000)                                                1
## 5 Zona Zamfirova (2002)                                                     1
```

```r
# Histogram of total counts of Ratings by Movies
edx %>% count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(color = "black", bins = 30) +
  scale_x_log10() +
  xlab("# Count") +
  ylab("# Movies") +
  ggtitle("Count by Movies") +
  theme(plot.title = element_text(hjust = 0.5))
```



The "genre" variable.

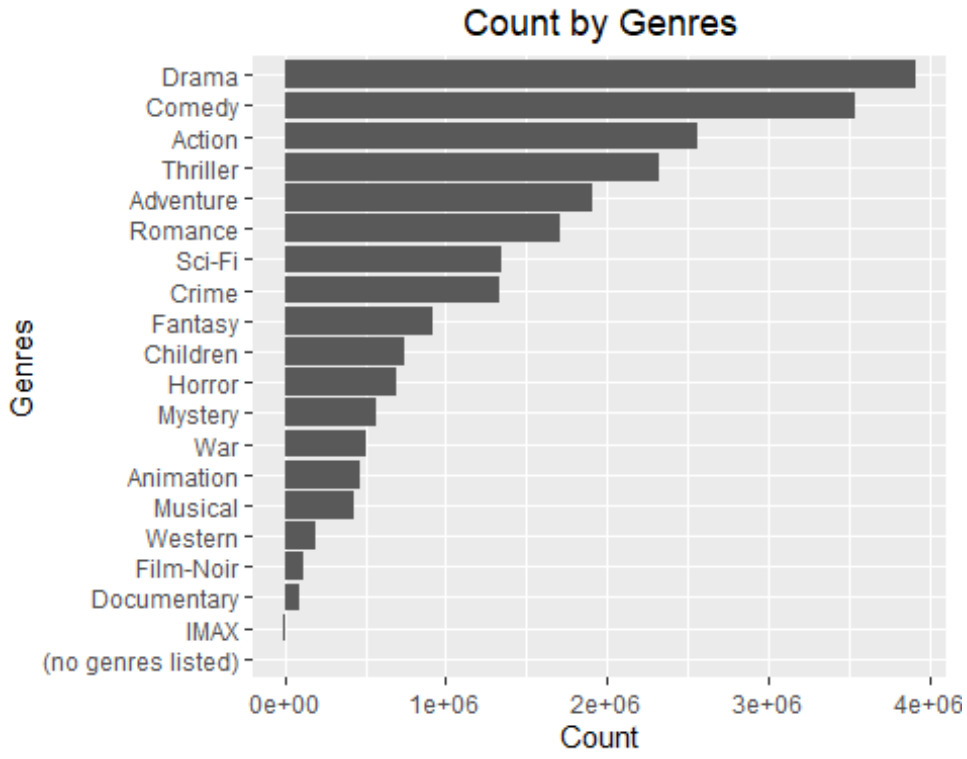
Lastly, we examine the genre variable and notice that the top 3 genres are Drama, Comedy and Action while the bottom 3 are IMAX, Documentary and Film-Noir which again follows the normal trend of movies.

```
# Viewing the most popular genres.
# Note: this process could take a couple of minutes
edx_genres <- edx %>%
        separate_rows(genres, sep = "\\|") %>%
        group_by(genres) %>%
        summarise(Count = n()) %>%
        arrange(desc(Count))
edx_genres
```

```
## # A tibble: 20 × 2
##    genres               Count
##    <chr>                <int>
##  1 Drama               3910127
##  2 Comedy              3540930
##  3 Action              2560545
##  4 Thriller            2325899
##  5 Adventure           1908892
##  6 Romance             1712100
##  7 Sci-Fi              1341183
##  8 Crime               1327715
##  9 Fantasy              925637
```

```
## 10 Children            737994
## 11 Horror              691485
## 12 Mystery             568332
## 13 War                 511147
## 14 Animation           467168
## 15 Musical             433080
## 16 Western             189394
## 17 Film-Noir           118541
## 18 Documentary          93066
## 19 IMAX                  8181
## 20 (no genres listed)       7
```

```
# Count of Ratings by Genres
edx_genres %>%
   ggplot(aes(x = Count, y = reorder(genres,Count))) +
   geom_col() +
   ylab("Genres") +
   ggtitle("Count by Genres") + theme(plot.title = element_text(hjust = 0.5))
```



Analysis, Testing and Result Interpretation

As the final_hold_out set will only be used for the final validation, we need to create new test and training sets to train and validate our algorithms.

```
# Creating test and traing sets from the edx set
set.seed(1, sample.kind = "Rounding") # if using R 3.6 or later

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

test_index <-createDataPartition(y = edx$rating, times = 1, p = 0.1, list =
FALSE)
edx_train <- edx[-test_index,]
edx_temp <- edx[test_index,]

# We need to confirm userId and movieId are in both the train and test sets
edx_test <- edx_temp %>%
  semi_join(edx_train, by = "movieId") %>%
  semi_join(edx_train, by = "userId")

# Add the Rows removed from the edx_test back into edx_train
removed <- anti_join(edx_temp, edx_test)
edx_train <- rbind(edx_train, removed)

rm(edx_temp,removed)
```

As discussed in the introduction, we will use the RMSE to test our algorithms which can defined as,

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

where N is the number of users-movie combinations, $y_{u,i}$ is the rating for for movie i by user u and $\hat{y}_{u,i}$ is our prediction.

The RMSE function can be written as follows.

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

Prediction Based on Mean Rating

This model uses the simple mean of the dataset to predict the rating for all movies, the model assumes that all differences are due to a random error.

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

where μ the expected rating and $\epsilon_{u,i}$ is the independent errors across all movies.

```
# Simple Prediction based on Mean Rating
mu <- mean(edx_train$rating)
mu

## [1] 3.512456

# Calculating RMSE using the mean
rmse_naive <- RMSE(edx_test$rating, mu)
rmse_naive

## [1] 1.060054

# Save Results to a Table
rmse_results = tibble(Method = "Naive Analysis by Mean", RMSE = rmse_naive)
rmse_results %>% knitr::kable()
```

Method	RMSE
Naive Analysis by Mean	1.060054

This prediction gives us a rather high RMSE over 1 which is far from ideal and doesn't help us in building a recommendation system. Now, using the insights we gained from the variables of this dataset we try and use them to get a better working system.

Movie Effects Model

We saw that there's a movie bias with some movies being rated higher that others and we try to incorporate this basis into our algorithm. We calculate the movie bias as the difference between the movie's mean rating vs the overall mean rating obtained earlier.

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

where b_i is the movie bias for each movie i.

```
# Model taking into account the movie effects, b_i
movie_avgs <- edx_train %>%
  group_by(movieId) %>%
  summarise(b_i = mean(rating - mu))

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

```
# Calculating RMSE using movie effects
rmse_model_movie_effects <- RMSE(predicted_ratings, edx_test$rating)
rmse_model_movie_effects

## [1] 0.9429615

# Adding RMSE results to the Table
rmse_results <- bind_rows(rmse_results,
                          tibble(Method="Movie Effects Model",
                          RMSE = rmse_model_movie_effects))
rmse_results %>% knitr::kable()
```

Method	RMSE
Naive Analysis by Mean	1.0600537
Movie Effects Model	0.9429615

The Movie Effects Model predicting the movie ratings with both biases gives an improved prediction with a lower RMSE value. Using other variables explored can we still try and achieve a lower RMSE?

Movie and User Effects Model

We now try to use out insights on the individual User's biases to help fine tune our model acknowledging that some users will rate some movies highly while others will rate the same movie lower and vice versa. We represent this user bias as b_u into our existing model.

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

where b_u is the bias for each user u.

```
# Movie and User Effects Model taking into account the user effects, b_u
user_avgs <- edx_train %>%
  left_join(movie_avgs, by="movieId") %>%
  group_by(userId) %>%
  summarise(b_u = mean(rating - mu - b_i))

predicted_ratings <- edx_test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
# Calculating RMSE using movie and user effects
rmse_model_user_effects <- RMSE(predicted_ratings, edx_test$rating)
rmse_model_user_effects

## [1] 0.8646843

# Adding RMSE results to the Table
rmse_results <- bind_rows(rmse_results,
                          tibble(Method="Movie and User Effects Model",
```

```
                                          RMSE = rmse_model_user_effects))
rmse_results %>% knitr::kable()
```

Method	RMSE
Naive Analysis by Mean	1.0600537
Movie Effects Model	0.9429615
Movie and User Effects Model	0.8646843

Incorporating the user bias into the existing model has resulted in a significant reduction of the RMSE.

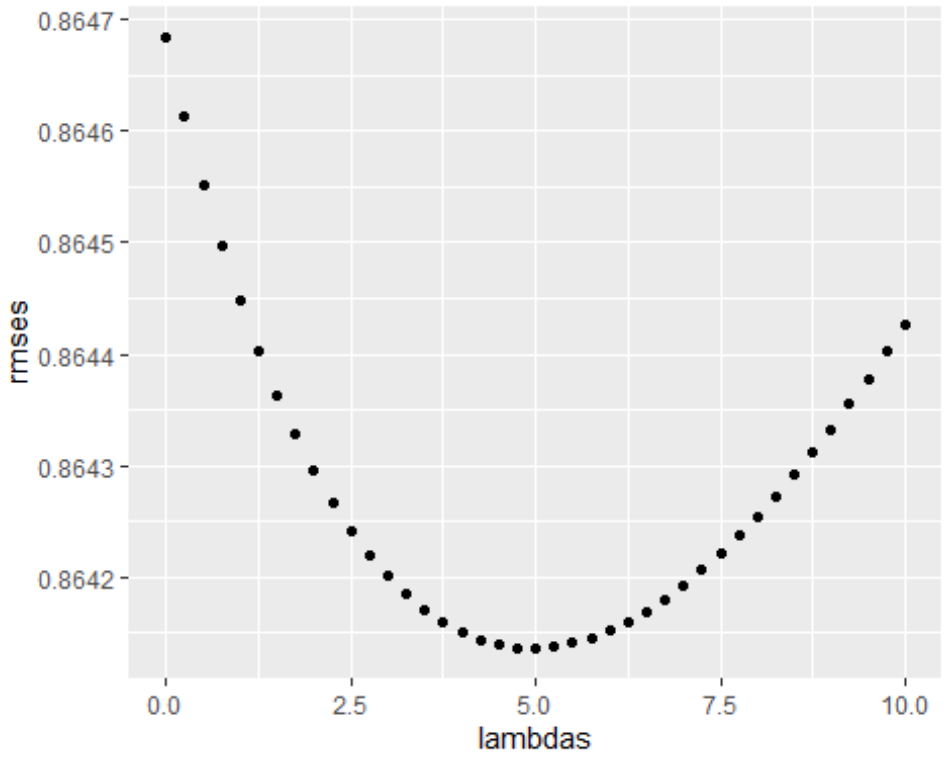
Regularisation

Regularization is used to control the variability of the effects size which can influence the prediction and skew our results. For example, a movie with 10 high ratings(above the mean of the dataset) can seem to be a better fit compared to a movie with 1000's of ratings scattered between 0-5 which is an incorrect conclusion. We uses a tuning parameter, λ, to minimise the RMSE by modifying b_i and b_u for movies with limited ratings.

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

```
# Predict via regularisation on the movie and user effect model

# Calculating RMSE using multiple values of lambda.
lambdas <- seq(0, 10, 0.25)

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

  mu <- mean(edx_train$rating)

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

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

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

  return(RMSE(predicted_ratings, edx_test$rating))
  })
```

```
rmse_regularisation <- min(rmses)
rmse_regularisation

## [1] 0.8641362

# Plot RMSE against Lambdas to find optimal lambda
tibble(lambdas,rmses) %>% ggplot(aes(lambdas,rmses)) + geom_point()
```




```
# Choose the optimal value of lambda
lambda <- lambdas[which.min(rmses)]
lambda

## [1] 5

# Adding RMSE results to the Table
rmse_results <- bind_rows(rmse_results,
                          tibble(Method="Regularised Movie and User Effects
Model",
                                 RMSE = rmse_regularisation))
rmse_results %>% knitr::kable()
```

Method	RMSE
Naive Analysis by Mean	1.0600537
Movie Effects Model	0.9429615
Movie and User Effects Model	0.8646843
Regularised Movie and User Effects Model	0.8641362

Regularisation of the Movie and User Effect model has led to an even lower RMSE value and has thus been the best prediction model we have created to use on this dataset.

Results, Discussion and Final Thoughts

The final values of the prediction models are;

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

Method	RMSE
Naive Analysis by Mean	1.0600537
Movie Effects Model	0.9429615
Movie and User Effects Model	0.8646843
Regularised Movie and User Effects Model	0.8641362

The RMSE results table tells us that the Regularised Model has yielded the lowest RMSE and hence has been the most successful while the basic Naive Mean Model has been least.

The final model optimised for the prediction is the following;

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

We tried and tested several different models to create an effective system from the data available to build a movie recommendation system. The "Regularised Movie and User Effects" Model had the most success in achieving this objective with a RMSE of 0.8641362.

While the final model performed well, there are additional biases that could be explored to further improve the accuracy of the model like year of release, genres and maybe even star casts. Further improvements could be expected by using techniques like "K-Nearest Neighbors" and "Random Forest" to this dataset.

Validating the preferred model using the final_holdout_set.

Now we use our preferred Model on the final_holdout_test set to validate the success of our findings. We compare this RMSE versus just the naive RMSE to give it some context.

```
# Prediction based on Mean Rating on the final_holdout_test set
final_mu <- mean(edx$rating)
final_mu

## [1] 3.512465

final_rmse_naive <- RMSE(final_holdout_test$rating, final_mu)
final_rmse_naive

## [1] 1.061202

# Save Results in a Table
final_rmse_results = tibble(Method = "Naive Analysis by Mean", RMSE =
final_rmse_naive)
final_rmse_results %>% knitr::kable()
```

Method	RMSE
Naive Analysis by Mean	1.061202

```
# Predict based on Regularisation Model
min_lambda <- lambda # Using the minimum lambda value from earlier

b_i <- edx %>%
    group_by(movieId) %>%
    summarise(b_i = sum(rating - final_mu)/(n() + min_lambda))

b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarise(b_u = sum(rating - b_i - final_mu)/(n() + min_lambda))

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

final_rmse_model <- RMSE(final_holdout_test$rating,predicted_ratings)
final_rmse_model

## [1] 0.8648177

# Save Results to the Table
final_rmse_results <- bind_rows(final_rmse_results,
                          tibble(Method="Regularised Movie and User Effects
Model",
```

```
                                          RMSE = final_rmse_model))
final_rmse_results %>% knitr::kable()
```

Method	RMSE
Naive Analysis by Mean	1.0612018
Regularised Movie and User Effects Model	0.8648177

Conclusion

The RMSE obtained using the Regularised model developed earlier on the final_holdout_test set has provided a satisfactory outcome and the project has delivered a working and effective algorithm which can be applied to similar situations.

References

https://rafalab.dfci.harvard.edu/dsbook/large-datasets.html#recommendation-systems
https://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/