Project: MovieLens Ratings System

Shane Pinto

2023-04-01

Table of Contents

1 Introduction and Overview 2
1.1 Creating the working Dataset 2
1.2 Analysing the created datasets 4
2 Analysis and Result Interpretation 10
2.1 Prediction Based on Mean Rating 10
2.2 Movie Effects Model 11
2.3 Movie and User Effects Model 11
2.4 Regularisation 12
3 Results and Discussion 15
4 Conclusion 15
5 Validating our preferred model using the final_holdout_set. 16
6 Conclusion and Final Thoughts 17

1 Introduction and Overview

Companies that sell a large selection of products to a large selection of customers and permit these customers to rate their products, like Amazon, are able to collect massive datasets that can be used to predict what rating a particular user will give a specific item. Items for which a high rating is predicted for a given user are then recommended to that user.

The aim of the Movie Lens Ratings System project is to develop and train a recommendation machine learning algorithm to predict a movie recommendation by using historical ratings of users to movies in the dataset. Netflix uses a recommendation system to predict how many stars a user will give a specific movie. One star suggests it is not a good movie, whereas five stars suggests it is an excellent movie. Here, we provide the basics of how these recommendations are made motivated by some of the approaches taken by the winners of the Netflix challenges.

To test the success of our recommendation system we will use the Residual Mean Square Error (RMSE) to evaluate the accuracy of the algorithm. RMSE is one of the preferred methods that measure the differences between values predicted by a model and the values observed, a lower RMSE is better than a higher one.

We can interpret RMSE similarly to a standard deviation, it is the typical error we make when predicting a movie rating. If this number is larger than 1, it means our typical error is larger than one star, which is not good. Our aim is therefore to obtain a RMSE below 1, this can be written as a function that computes the RMSE for vectors of ratings and their corresponding predictors

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

This report will present an overview of the data, analysis, results and a conclusion.

1.1 Creating the working Dataset

Data is downloaded and set up as per instruction from the MovieLens Capstone Project dataset.

```r
# Create edx and final_holdout_test sets

# 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)
```

```r
# 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")

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
```

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

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

As the final_holdout_test set will only be used for testing the final algorithm at the end, we need to create separate training and test sets to be used for training, developing and selecting our algorithms.

```r
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,]

# Again, 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)
```

1.2 Analysing the created datasets

An introductory review of the dataset is performed in order to familiarise ourselves. The edx_train dataset contains rows corresponding to an users rating of a movie. The set contains the variables; “userId”, “movieId”, “rating”, “timestamp”,“title”, “genres”.

```r
# Summarise edx
head(edx_train, 5)

##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 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
## 4  Action|Drama|Sci-Fi|Thriller
## 5       Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7       Children|Comedy|Fantasy
```

Summarising the dataset reveals a well formatted set with no missing values. Movies are rated between 0.5 and 5.0, with ~8100000 rows in total.

```
summary(edx_train)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18127   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35732   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4120   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:8100065     Length:8100065
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

The dataset contains ~10,700 unique movies, ~70,000 unique users and ~800 unique genres, and a mean movie rating of ~3.5 out of 5.

```
# Movies, Users and Genres in Database
edx_train %>% summarise(
  uniq_movies = n_distinct(movieId),
  uniq_users = n_distinct(userId),
  uniq_genres = n_distinct(genres))
```

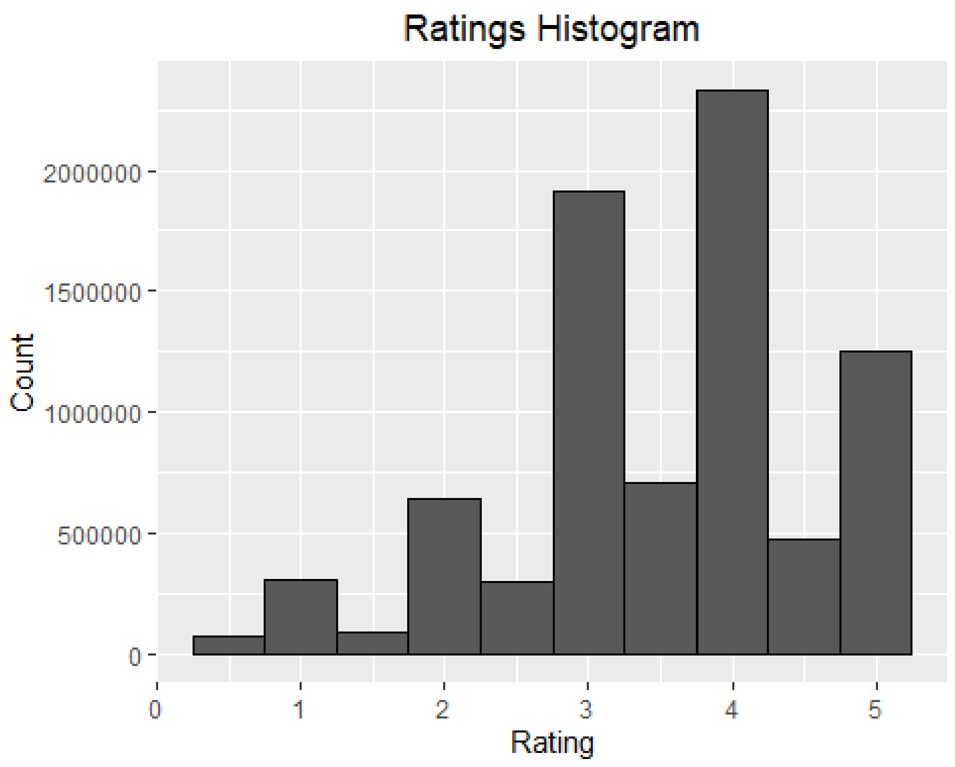
```
##   uniq_movies uniq_users uniq_genres
## 1       10677      69878         797
```

```
# Ratings Mean
rating_mean <- mean(edx_train$rating)
rating_mean
```

```
## [1] 3.512456
```

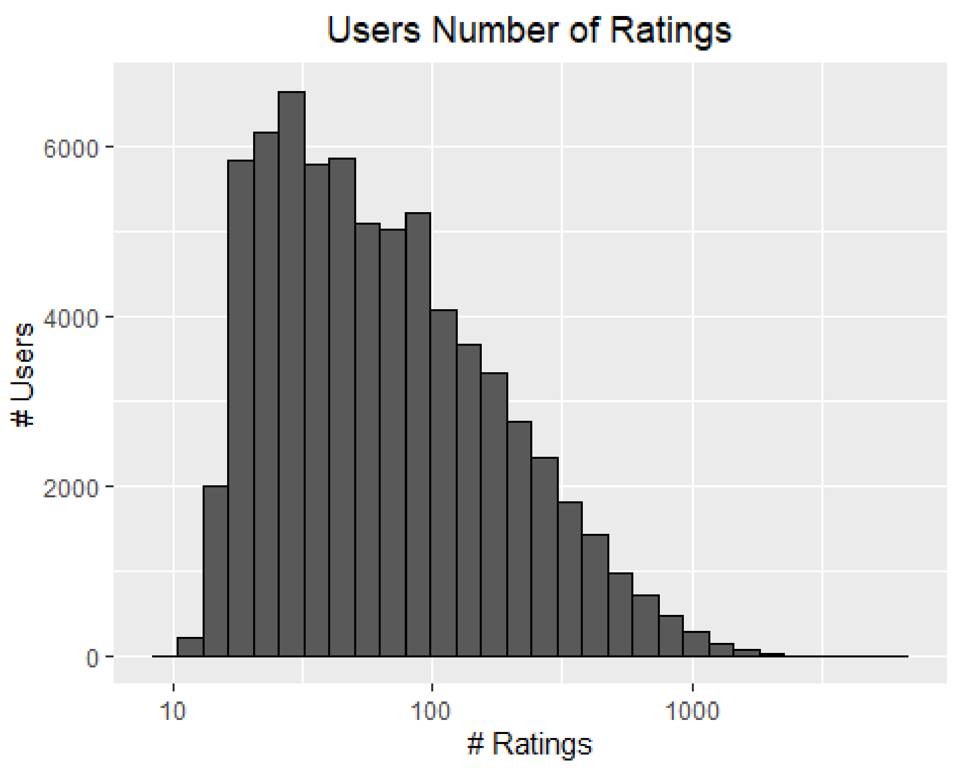
A histogram of the dataset mapping ratings and counts reveals that the highest rating was 4.0 while the lowest rating was 0.5.Overall, we also notice that users tend to rate movies with whole star ratings like 1,2,3 etc rather than half star ratings like 0.5,1.5,2.5 etc.

```
# Ratings Histogram
edx_train %>%
  ggplot(aes(rating)) +
  geom_histogram(binwidth = 0.5, color = "black") +
  xlab("Rating") +
  ylab("Count") +
  ggtitle("Ratings Histogram") +
  theme(plot.title = element_text(hjust = 0.5))
```



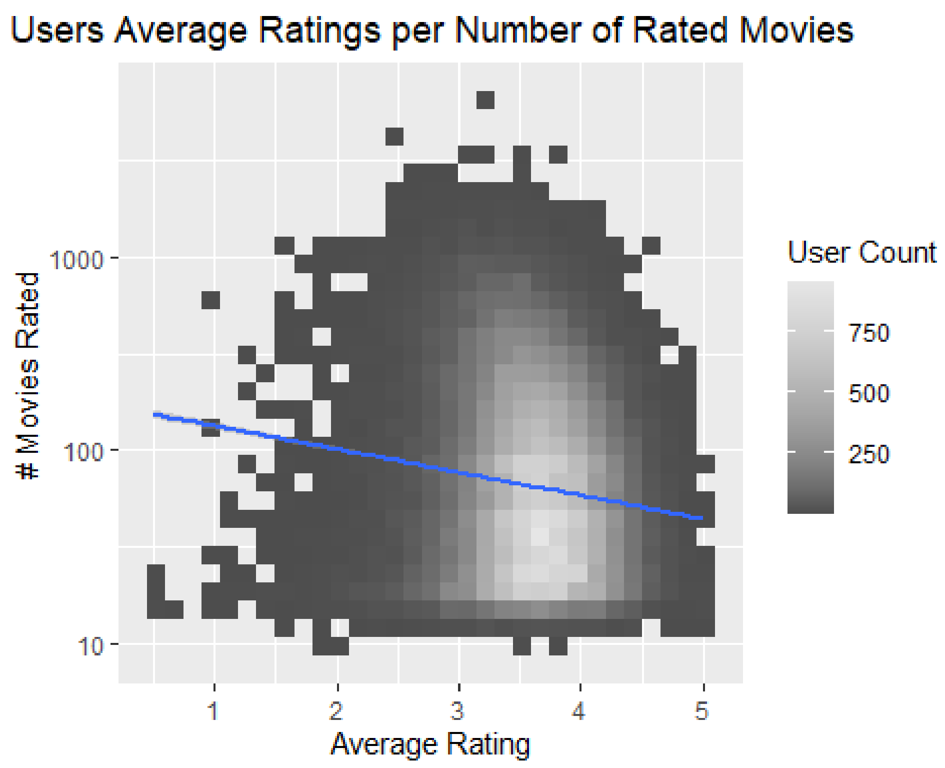
The majority of users rate between 10 and 100 movies, whilst some may rate over 1,000. Including a variable in the model to account for number of ratings should be considered to account for this.

```
# Ratings Users - Number of Ratings
edx_train %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(color = "black", bins=30) +
  scale_x_log10() +
  xlab("# Ratings") +
  ylab("# Users") +
  ggtitle("Users Number of Ratings") +
  theme(plot.title = element_text(hjust = 0.5))
```



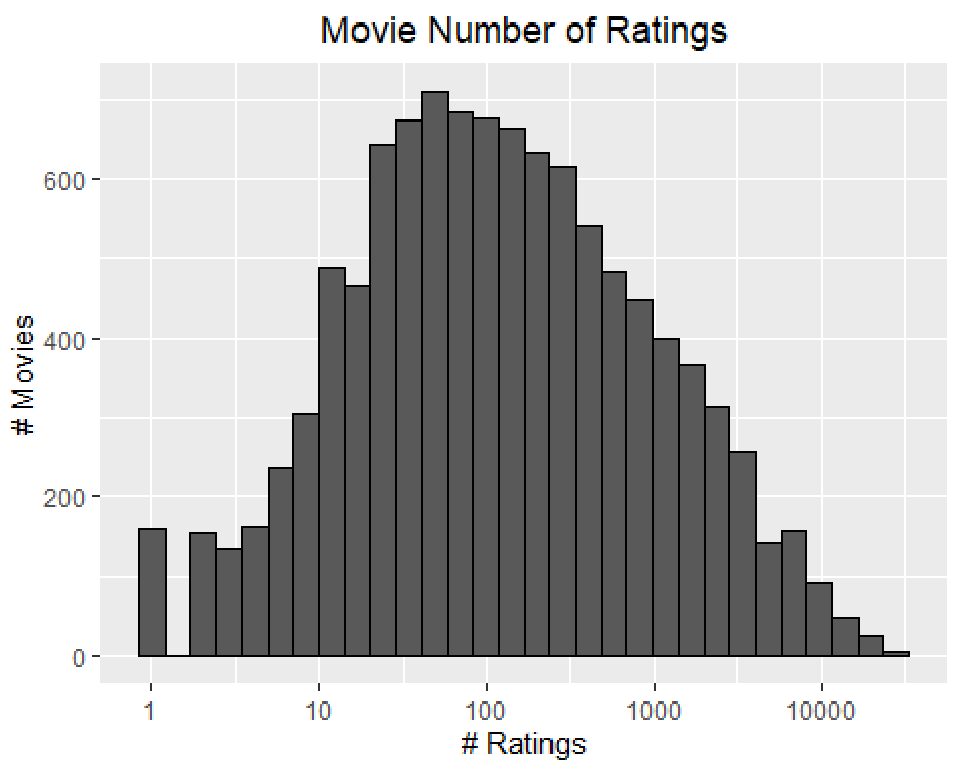
A heatmap showing average movie rating against number of movies rated is plotted. The most common movies ratings and number of movies rated are highlighted. This occurs for a rating between 3.5 and 4.0 and between 10 and 100 movies. A linear curve was fitted to the data to show the overall trajectory of the ratings with number of movies rated, the more ratings a user gives results in a lower mean rating.

```
# Ratings Users - Mean by Number with Curve Fitted
edx_train %>%
  group_by(userId) %>%
  summarise(mu_user = mean(rating), number = n()) %>%
  ggplot(aes(x = mu_user, y = number)) +
  geom_bin2d( ) +
  scale_fill_gradientn(colors = grey.colors(10)) +
  labs(fill="User Count") +
  scale_y_log10() +
  geom_smooth(method = lm) +
  ggtitle("Users Average Ratings per Number of Rated Movies") +
  xlab("Average Rating") +
  ylab("# Movies Rated") +
  theme(plot.title = element_text(hjust = 0.5))

## `geom_smooth()` using formula = 'y ~ x'
```



The number of ratings for each movie are shown below in the histogram. A number of outlier movies have been rated less than 10 times which will make predicting future ratings more difficult.

```r
# Ratings Movies - Number of Ratings
edx_train %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(color = "black", bins=30) +
  scale_x_log10() +
  xlab("# Ratings") +
  ylab("# Movies") +
  ggtitle("Movie Number of Ratings") +
  theme(plot.title = element_text(hjust = 0.5))
```


Movie Number of Ratings
# Movies
0
200
400
600
# Ratings
1
10
100
1000
10000

2 Analysis and Result Interpretation

The Residual Mean Square Error (RMSE) is the error function to that will measure accuracy and quantify the typical error we make when predicting the movie rating. An RMSE greater than or equal to 1 means our typical error is larger than almost a star, which is not good.

RMSE defined;

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

where; N is the number of users, movie ratings, and the sum incorporating the total combinations.

2.1 Prediction Based on Mean Rating

This simple prediction model uses the 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 $Y_{u,i}$ is the prediction, $\epsilon_{u,i}$ is the independent error, and μ the expected "true" rating for all movies.

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

## [1] 3.512456

rmse_naive <- RMSE(edx_test$rating, mu)
rmse_naive

## [1] 1.060054

# Save Results in Data Frame
rmse_results = data_frame(method = "Naive Analysis by Mean", RMSE =
rmse_naive)

## Warning: `data_frame()` was deprecated in tibble 1.1.0.
## ℹ Please use `tibble()` instead.

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

method	RMSE
Naive Analysis by Mean	1.060054

Predicting the mean gives us a rather high RMSE over 1 which is far from ideal and doesn't help us in building a recommendation system. Using our earlier insights of the dataset allows us a for more advanced analysis and rating predictions with hope if achieving a smaller error.

2.2 Movie Effects Model

The Movie Effects Model calculates a bias term for each movie based on the difference between the movies mean rating and the overall mean rating.

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

where $Y_{u,i}$ is the prediction, $\epsilon_{u,i}$ is the independent error, and μ the mean rating for all movies, and b_i is the bias for each movie i.

```
# Simple 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)

rmse_model_movie_effects <- RMSE(predicted_ratings, edx_test$rating)
rmse_model_movie_effects

## [1] 0.9429615

rmse_results <- bind_rows(rmse_results,
                          data_frame(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 b_i and the mean μ gives an improved prediction with a lower RMSE value. But can we do better by accounting for more variables and biasis affecting the overall RMSE.

2.3 Movie and User Effects Model

The next step is to incorporate the individual User Effects, b_u, in to the model, acknowledging that each user's inherent bias to mark all films higher or lower.

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

where $Y_{u,i}$ is the prediction, $\epsilon_{u,i}$ is the independent error, and μ the mean rating for all movies, b_i is the bias for each movie i, and 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)

rmse_model_user_effects <- RMSE(predicted_ratings, edx_test$rating)
rmse_model_user_effects

## [1] 0.8646843

rmse_results <- bind_rows(rmse_results,
                          data_frame(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

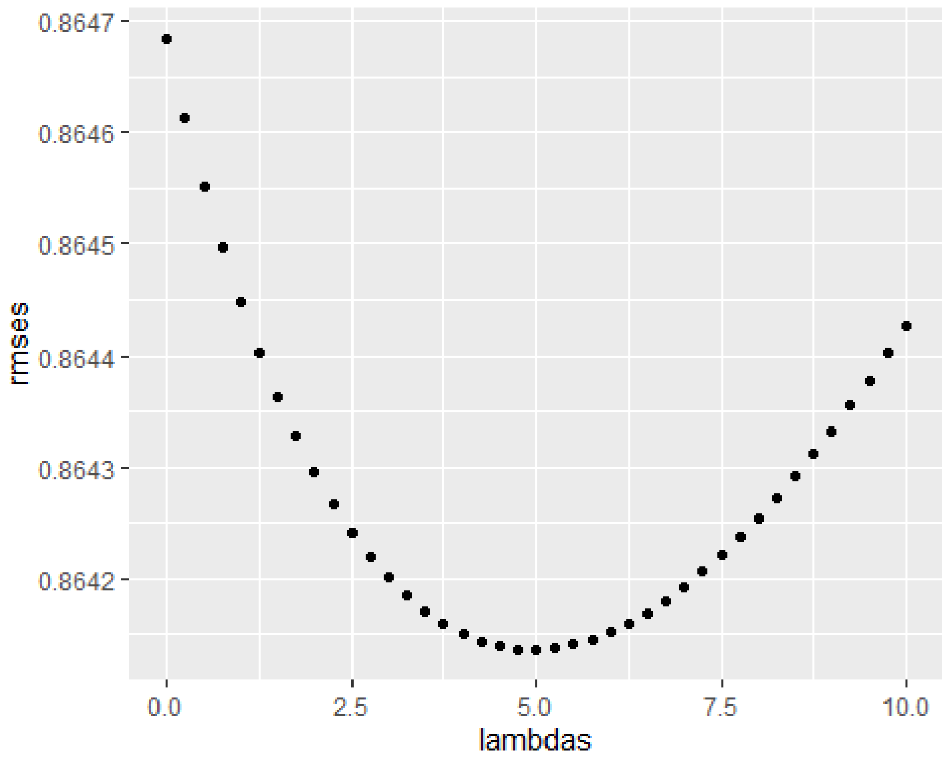
Incorporating the user bias into the model resulted in a further reduced RMSE and is a marked improvement as compared to the previous models.

2.4 Regularisation

Regularisation allows for reduced errors caused by movies with few ratings which can influence the prediction and skew the error metric. The method uses a tuning parameter, λ, to minimise the RMSE. 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, movie and user effect model
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
qplot(lambdas, rmses)

## Warning: `qplot()` was deprecated in ggplot2 3.4.0.
```



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

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

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(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 a Movie and User Effect model has led to a lowest RMSE of the prediction methods for the MovieLens ratings system.

3 Results and Discussion

The final values of the prediction models are shown below;

```
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 Movie and User Effects Model has been the yielded the lowest RMSE and hence has been the most successful at achieving our goal. The basic Naive Mean Model has been least successful and really is not very helpful at achieving our goals.

The final model optimised for the prediction is the following;

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

4 Conclusion

A machine learning algorithm to predict the ratings from the Movie Lens dataset was constructed. The optimal model incorporated the effects of both user and movie biases and these variables were regularised to incorporate movies will a low number of ratings.

The aim of the project was to develop an algorithm with RMSE lower than 1 which was achieved by using the Movie and User Effects along with Regularisation.

The lowest value of RMSE obtained via this model was 0.8641362.

5 Validating our preferred model using the final_holdout_set.

Now we use the Regularised Movie and User Effects Model algorithm on the final_holdout_test set to validate the success of our findings. We compare the RMSE obtained versus just the naive RMSE to give it some context.

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

## [1] 3.512465

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

## [1] 1.061202

## Save Results in Data Frame
final_rmse_results = data_frame(method = "Naive Analysis by Mean", RMSE =
final_rmse_naive)
final_rmse_results %>% knitr::kable()
```

method	RMSE
Naive Analysis by Mean	1.061202

```
# Predict via regularisation, movie and user effect model
# We use the minimum lambda value from the earlier models on the
final_holdout_set as it was shown to be the best tune.
min_lambda <- lambda

mu <- mean(edx$rating)

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

predicted_ratings <- final_holdout_test %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = 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 in Data Frame
final_rmse_results <- bind_rows(final_rmse_results,
                            data_frame(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

6 Conclusion and Final Thoughts

The selected model has a RMSE of 0.8648242 using the final_holdout_test set and has therefore achieved the goal we had set. This model significantly improves upon the benchmarking model's Naive RMSE of ~1.06 and can be suggested as a recommended model.

While the final model, which was comprised of User Effects, Movie Effects and with Regularization performed well, there are additional biases that could be explored to further improve the accuracy of the model like year of release, genre, sub genres etc. More advanced models featuring Random Forests and those based on Singular Value Decomposition (made famous by Simon Funk during the Netflix Challenge) generate higher levels of accuracy, though they are also far more computationally intensive. It is likely that future iterations of models involving K-Nearest-Neighbours and Collaborative Filtering with cosine similarity promise to continue making strides and improve the overall user experience for streamers everywhere.