HARVARDX-PH125.9x Data Science Capstone Project: Movie recommendation system using MovieLens Dataset

Deepika Dittakavi

5/23/2020

Introduction

Recommendation systems use ratings that users give the products to make particular recommendations. When users buy products from different companies like Amazon, Walmart, Target etc., they are allowed to rate the different products they have purchased. The companies are then able to collect massive datasets that can be used to predict which rating a specific user will give to a specific product. Products for which a high rating is predicted for a given user are then recommended to that user.

Reccomendation systems are very useful for service providers and customers. They have proven to improve the decision making process of customers and thereby enhance revenues. Today, in an environment of online-shopping, they are an effective means of selling products. The reviews given by users serve as marketing for the products and also help the service providers to improve the quality of the products.

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. Movies that are expected to get a higher rating are then recommended to that user.

The goal of this project is to create a movie recommendation system using the MovieLens dataset by utilizing all the skills that were taught throughout the courses in the Data Science Certificate Program by HarvardX.

Movielens Dataset

The GroupLens Research lab has collected and made available rating data sets from the MovieLens web site. The datasets were collected over various periods of time. GroupLens generated their own database, of which one set is the MovieLens 10M movie ratings version, with about 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users. This project uses the MovieLens 10M dataset for the analysis and prediction.

Data Analysis and Methods

Data Collection

The MovieLens dataset can be found at https://grouplens.org/datasets/movielens/10m/ . A code has been provided to collect the data from the above mentioned website. The data was also cleaned up to tidy format and split to create an edx dataset and validation dataset.

The edx dataset has been provided to develop the algorithm and the validation dataset for final prediction of the movie ratings. To develop and train the algorithm the edx dataset was further split to create train and test datasets.

```
################################
# Create edx set, validation set
################################

# Note: this process could take a couple of minutes

if(!require(tidyverse)) install.packages("tidyverse",
                                         repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse
```

```
## -- Attaching packages ---------------------------------------------------------------------------
```

```
## v ggplot2 3.3.0     v purrr   0.3.4
## v tibble  3.0.1     v dplyr   0.8.5
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
```

```
## Warning: package 'tibble' was built under R version 3.6.2
```

```
## Warning: package 'purrr' was built under R version 3.6.2
```

```
## -- Conflicts ------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caret
```

```
## Loading required package: lattice
```

```
## Warning: package 'lattice' was built under R version 3.6.2
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```r
if(!require(data.table)) install.packages("data.table",
                                          repos = "http://cran.us.r-project.org")
```

```
## Loading required package: data.table

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

## The following object is masked from 'package:purrr':
##
##     transpose
```

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

dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))

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

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding")
```

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

```r
# if using R 3.5 or earlier, use `set.seed(1)` instead
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)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

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

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


#############################################
# Create training and test sets from edx set  #
#############################################

# Used 20% for test and 80% for train
set.seed(123, sample.kind="Rounding")
```

```
## Warning in set.seed(123, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
t_index <- createDataPartition(y = edx$rating, times = 1,
                               p = 0.2, list = FALSE)
train_set <- edx[-t_index,]
temp <- edx[t_index,]

# Keep movies and users that have a match in the train set
test_set <- temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

# Add rows removed from test set back into train set
removed <- anti_join(temp, test_set)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

```
train_set <- rbind(train_set, removed)
rm(t_index, temp, removed)
```

Data Exploration

First, it is important to explore the data to see what format it is in and if it needs any further cleaning. Understanding the data and different parameters or predictors is crucial as it will determine what model will be appropriate.

To familiarize with the data the following excercises were performed.

```
# First few lines of both datasets
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
```

```
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(edx)
```

```
## [1] 9000055       6
```

```
dim(validation)
```

```
## [1] 999999      6
```

Clearly the data is in the tidy format and does not need any changes to it. Checked for any NA values.

```
sum(is.na(edx))
```

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

```
sum(is.na(validation))
```

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

The datasets consists of 6 columns:

- userId: unique ID for each user.
- movieId: unique ID for each movie.
- title: Name of the movie.
- rating: Specifies the rating given by "userId" to "movieId". It can be anywhere between 0-5. (the algorithm will predict this for validation set)
- timestamp: Specifies the time when the rating was given, it is in epoch format, that means it is the total seconds from January 1st, 1970 at UTC to the time of rating.
- genres: list containing all the applicable genres for the movie. They are joined with the "|" symbol.

According to the dimensions, there are 9000055 rows in the edx dataset and 999999 rows in the validation set, where each row is a rating.

To get unique users and movies in the edx dataset:

```
edx%>% summarize(unique_users=n_distinct(userId), unique_movies=n_distinct(movieId))
```

```
##   unique_users unique_movies
## 1        69878         10677
```

Rating

The rating column of the dataset is what the machine learning algorithm predicts, based on all other features as needed. The rating is a number between 0.5 and 5 inclusive. So for the purposes of our algorithm, the rating is the outcome Y, which depends on each movie and user.

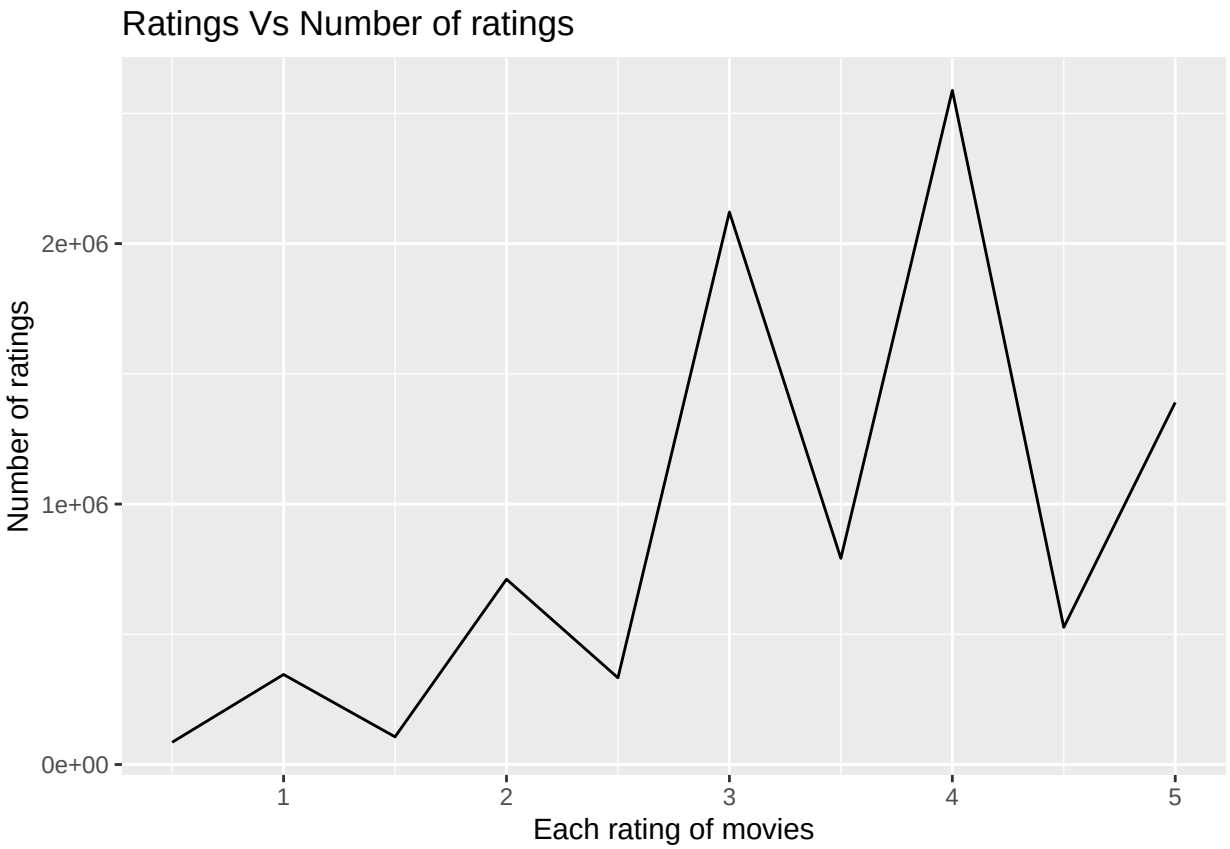
Creating a list of the number of ratings received for each rating:

```
edx%>%group_by(rating)%>%summarize(n=n())%>%arrange(desc(n))
```

```
## # A tibble: 10 x 2
##    rating       n
##     <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
```

Plot of the ratings Vs no. of ratings

```
edx %>%
  group_by(rating) %>% summarize(count = n()) %>%
  ggplot(aes(x = rating, y = count)) +
  geom_line()+
  ggtitle("Ratings Vs Number of ratings")+xlab("Each rating of movies")+
  ylab("Number of ratings")
```

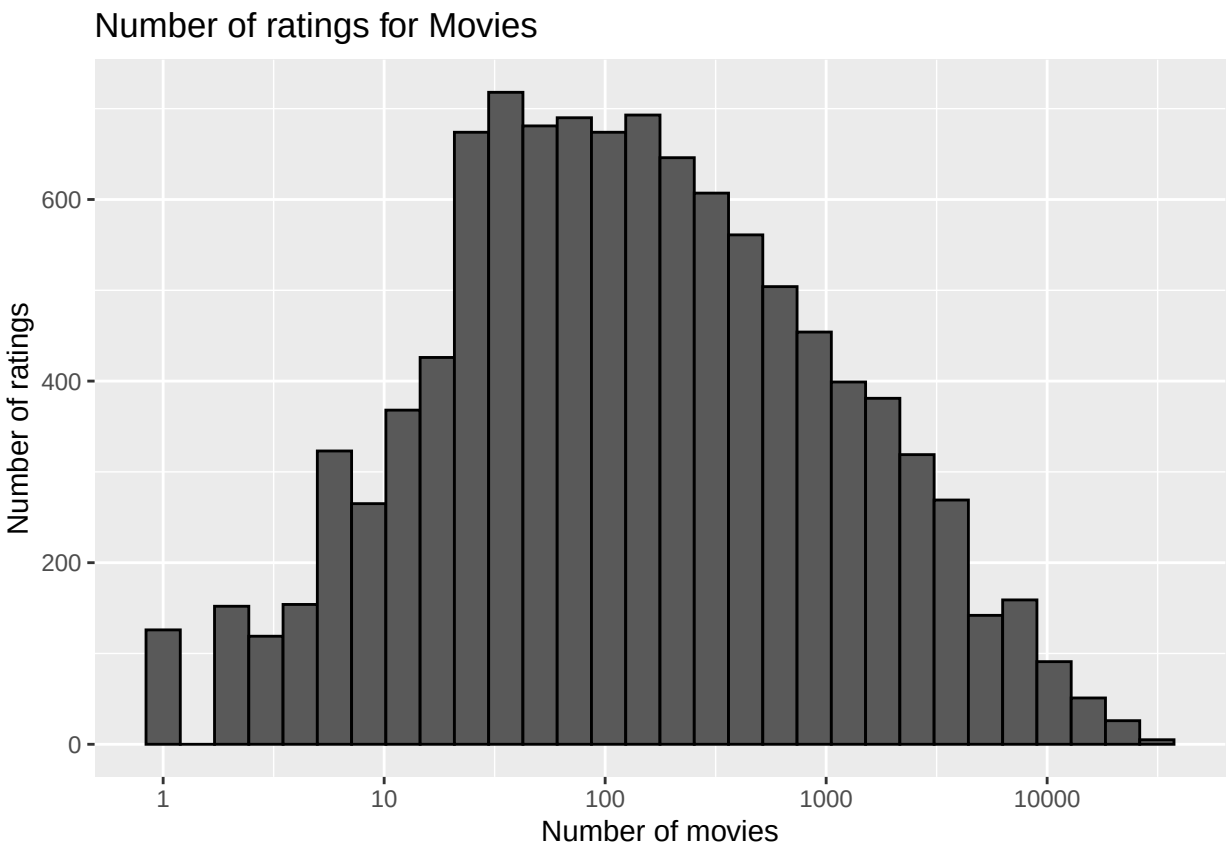


This plot shows that users tend to rate movies higher rather than lower and 4 being the most popular rating.

Movies

In the below plot it can be seen that some movies get rated more than others. This should not be a surprise as it is known that blockbuster movies are watched by many and artsy movies are watched by a few.

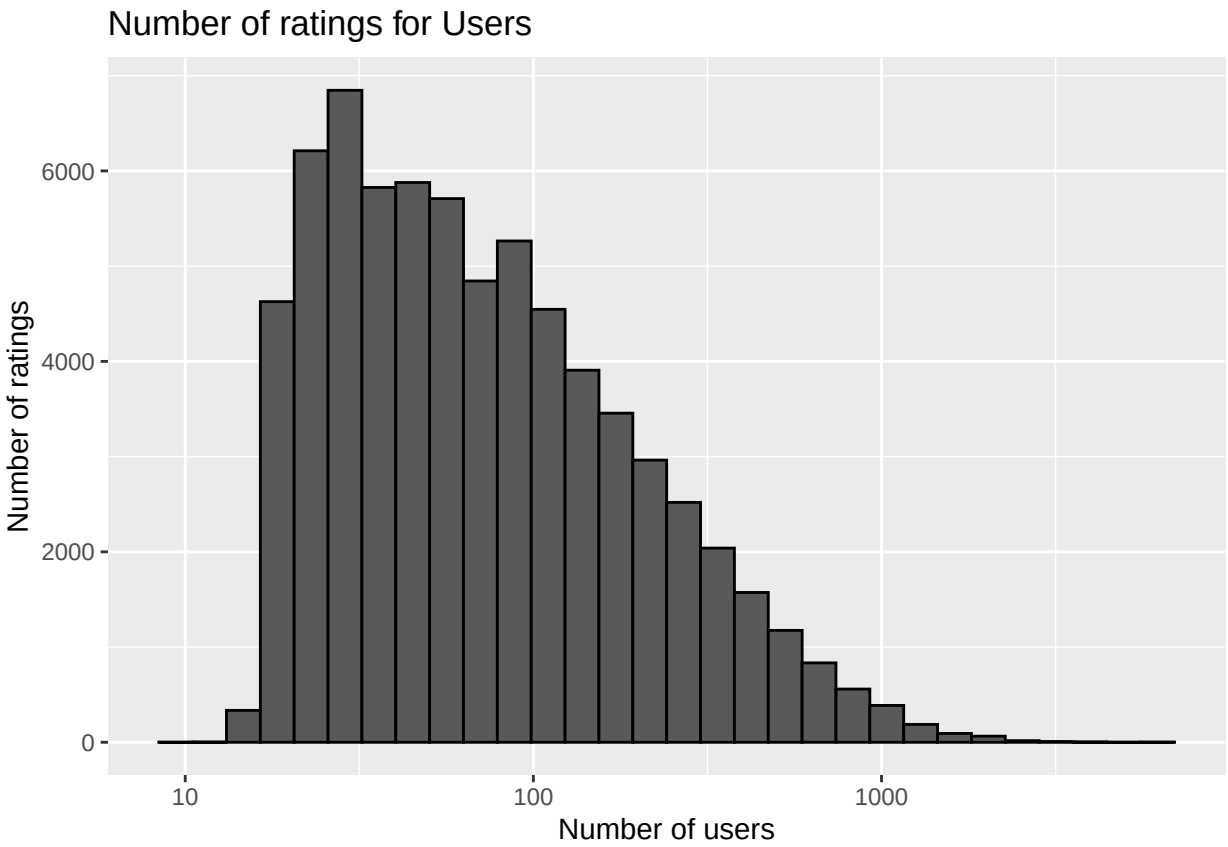
```
edx%>% count(movieId) %>%
  ggplot(aes(n))+
  geom_histogram(bins = 30, color="black")+
  scale_x_log10()+
  ggtitle("Number of ratings for Movies")+xlab("Number of movies")+
  ylab("Number of ratings")
```



Users

The below plot shows that some users have more number of ratings. This just means that some of these users are more active at rating movies than the others.

```
edx%>% count(userId) %>%
  ggplot(aes(n))+
  geom_histogram(bins = 30, color="black")+
  scale_x_log10()+
  ggtitle("Number of ratings for Users")+xlab("Number of users")+
  ylab("Number of ratings")
```



Data Splitting

There are two datasets, edx to train the model and validation to report results of the model. However, to train and fit a model a test set is needed. It is known that the model will be inaccurate to use validation set for training purposes. Therefore the edx dataset is further split into train and test datasets. The train dataset can then be used to train the model and the test dataset can be used to measure and compare different models that will be evaluated.

Using a similar split, as between edx and validation datasets, the train and test datasets are created with 90% and 10% split as shown below.

```
# Create test and train sets from the edx set that was generated
# Used 10% for test and 90% for train
set.seed(123, sample.kind="Rounding")
```

```
## Warning in set.seed(123, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

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

# Remove movies that do not include users and movies in the test set but not in train set
test_set <- temp %>%
```

```
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

# Add rows removed from test set back into train set
removed <- anti_join(temp, test_set)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

```
train_set <- rbind(train_set, removed)
```

Data Modeling method and Analysis

Output measure Method

The goal of the machine learning algorithm here is to predict the rating of a movie given by a user. Since the output Y is a number the best way to measure the accuracy of the prediction algorithm is to measure the RMSE(root-mean-square-error) on the test set.

RMSE can be generally called the loss function and can be defined as:

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

where N is the number of movies, $y_{u,i}$ is the true rating and $\hat{y}_{u,i}$ is the predicted rating for each user u and movie i.

```
# Define RMSE function to call it as needed
RMSE <- function(y, yhat){
  sqrt(mean((y - yhat)^2))
}
```

RMSE is similar to standard deviation, it is the error made when predicting a movie rating. So if the number is greater than 1 it would mean that the rating is off by 1 star which is not great. The lower the RMSE the better the prediction. The goal of this project is to get lower RMSE, ideally < 0.86490.

Guessing Model

One of the most easiest and simplest model of prediction is to just make a random guess of the rating. The RMSE for this method is expected to be higher because there is no reason to predict a particular rating. It is not realistic to use this model, however it is included here to just show how worse the prediction can get.

```
set.seed(123, sample.kind="Rounding")
```

```
## Warning in set.seed(123, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

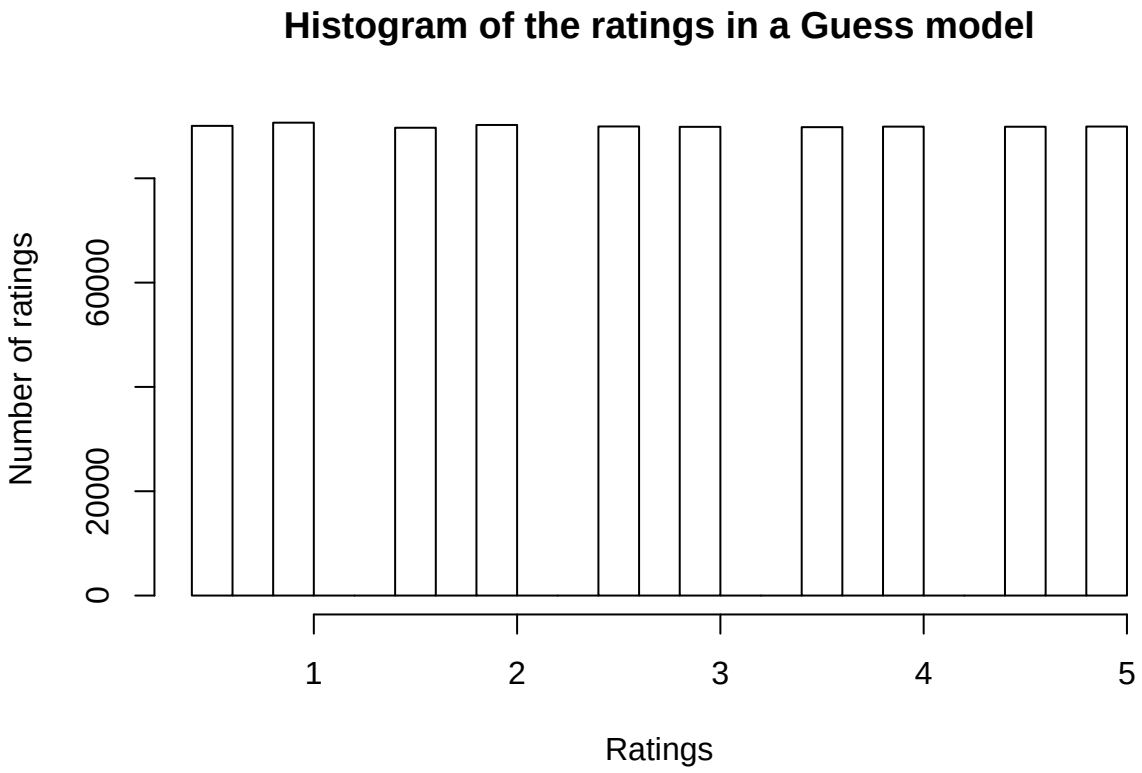
```
guess_rating<-sample(c(0.5,1,1.5,2,2.5,3,3.5,4,4.5,5),
                     length(test_set$rating), replace=TRUE)
# randomly selects a number between 0.5 and 5
guess_rmse<-RMSE(test_set$rating, guess_rating)

# create a summary to keep adding RMSE values for each model
rmse_summary<-data.frame(method = "A guess model", RMSE = guess_rmse)
rmse_summary %>% knitr::kable()
```

method	RMSE
A guess model	1.94324

Results:

The RMSE is very high, the value indicates that the prediction is off by almost two stars. The high error can be explained by the below plot which shows the distribution of the prediction. The plot shows that each rating was sampled equal number of times and from a plot shown in a previous section we know that very low ratings and very high ratings had lower number of ratings. So that means this model was predicting many ratings incorrectly and also with high error.

```
hist(guess_rating, xlab="Ratings", ylab="Number of ratings",
     main="Histogram of the ratings in a Guess model")
```



An Average Model

Since the guessing model had high RMSE it is known that the errors need to be minimized. One way to do this is by predicting that all users will give the same rating regardless of movies. The initial prediction is just the average of all ratings.

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

where $Y_{u,i}$ is the prediction, $\epsilon_{u,i}$ is the independent errors sampled from the same distribution centered at 0, and μ is the mean of the observed data (the "true" rating for all movies). Any value other than the mean, increases the RMSE, so this is a good initial estimation. We know that the estimate that minimizes the RMSE is the least squares estimate of μ and, in this case, is the average of all ratings.

```
mu_train<-mean(train_set$rating)
mu_rating<-rep(mu_train, length(test_set$rating))
```

```
mu_rmse<-RMSE(test_set$rating, mu_rating)
rmse_summary<-rbind(rmse_summary, data.frame(method = "An average model", RMSE = mu_rmse))
rmse_summary %>% knitr::kable()
```

method	RMSE
A guess model	1.94324
An average model	1.06062

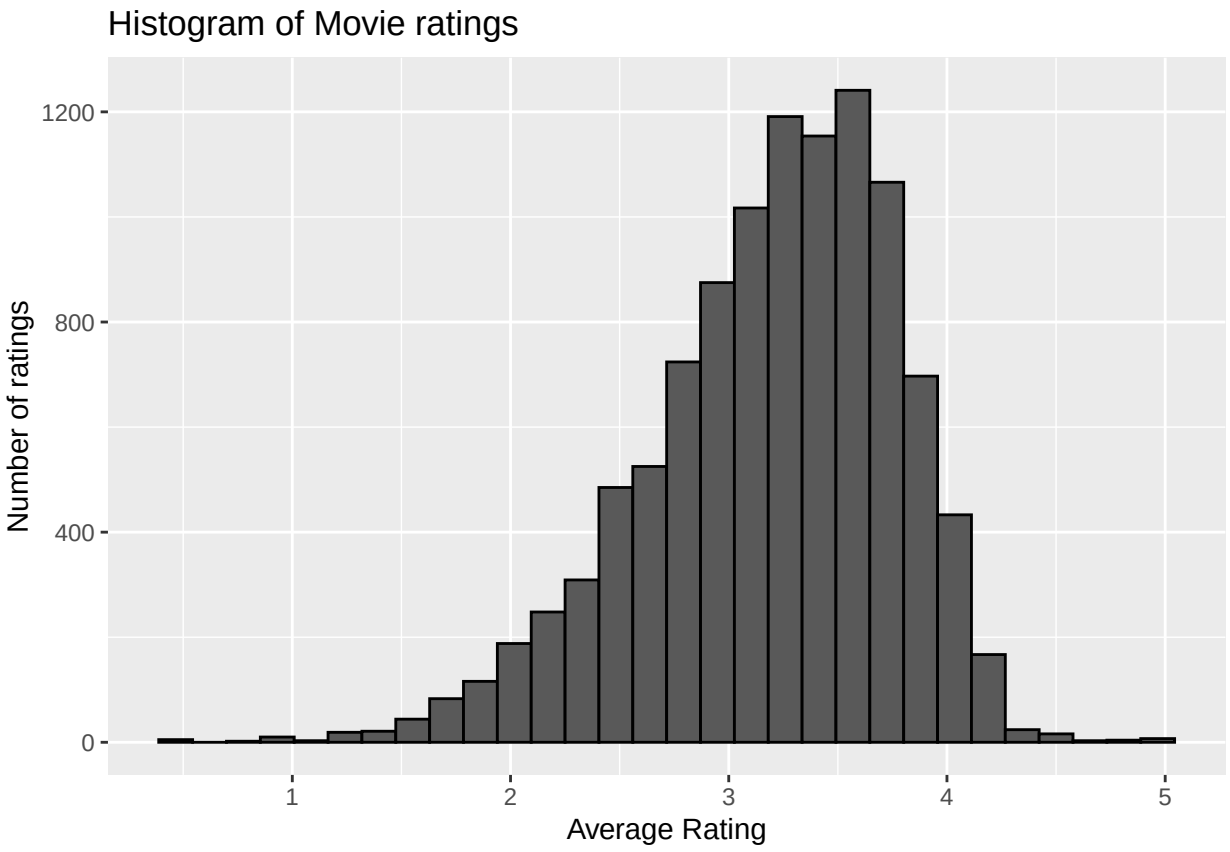
Results:

The RMSE has improved, but is still off by 1 star rating. By using further models this number can get better. It should be noted here that if any other rating value other than the mean value was used then the RMSE would have been higher.

The Movie effect Model

In a previous graph, it was shown that some movies are just generally rated higher than others. Higher ratings are given to popular movies and lower ratings to the not so popular ones. Below is the histogram showing the distribution for the training dataset.

```
train_set%>%
  group_by(movieId) %>%
  summarize(mov_avg = mean(rating))%>%
  ggplot(aes(mov_avg))+geom_histogram(color="black", bins=30)+
  ggtitle("Histogram of Movie ratings")+xlab("Average Rating")+ylab("Number of ratings")
```



Here are the top ten and bottom ten rated movies:

```
# Top 10 rated movies based on avg by movie
train_set%>%
  group_by(movieId) %>%
  summarize(title=first(title), count=n(), mov_avg = mean(rating))%>%
  arrange(desc(mov_avg))%>%
  slice(1:10)
```

```
## # A tibble: 10 x 4
##    movieId title                                                  count mov_avg
##      <dbl> <chr>                                                  <int>   <dbl>
##  1    3226 Hellhounds on My Trail (1999)                             1    5
##  2   33264 Satan's Tango (Sátántangó) (1994)                         2    5
##  3   42783 Shadows of Forgotten Ancestors (1964)                     1    5
##  4   51209 Fighting Elegy (Kenka erejii) (1966)                      1    5
##  5   53355 Sun Alley (Sonnenallee) (1999)                            1    5
##  6   64275 Blue Light, The (Das Blaue Licht) (1932)                  1    5
##  7    4454 More (1998)                                              6    4.92
##  8   26048 Human Condition II, The (Ningen no joken II) (1959)       3    4.83
##  9    5194 Who's Singin' Over There? (a.k.a. Who Sings Over There~   4    4.75
## 10   26073 Human Condition III, The (Ningen no joken III) (1961)     4    4.75
```

```
# Bottom 10 rated movies based on avg by movie
train_set%>%
  group_by(movieId) %>%
```

```
  summarize(title=first(title), count=n(), mov_avg = mean(rating))%>%
  arrange(mov_avg)%>%
  slice(1:10)
```

```
## # A tibble: 10 x 4
##    movieId title                                        count mov_avg
##      <dbl> <chr>                                        <int>   <dbl>
##  1    5805 Besotted (2001)                                2   0.5
##  2    8394 Hi-Line, The (1999)                            1   0.5
##  3   61768 Accused (Anklaget) (2005)                      1   0.5
##  4   63828 Confessions of a Superhero (2007)              1   0.5
##  5   64999 War of the Worlds 2: The Next Wave (2008)      2   0.5
##  6    8859 SuperBabies: Baby Geniuses 2 (2004)           51   0.824
##  7    7282 Hip Hop Witch, Da (2000)                      13   0.846
##  8   61348 Disaster Movie (2008)                         28   0.893
##  9    6483 From Justin to Kelly (2003)                  178   0.899
## 10     604 Criminals (1996)                               2   1
```

Since it is confirmed by data, the prediction model can be augmented by inlcuding a term b_m to represent average ranking for movie i. This effect is also called bias, hence b_m is effect of movies. This can be denoted by:

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

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

By using least square to estimate b_m:

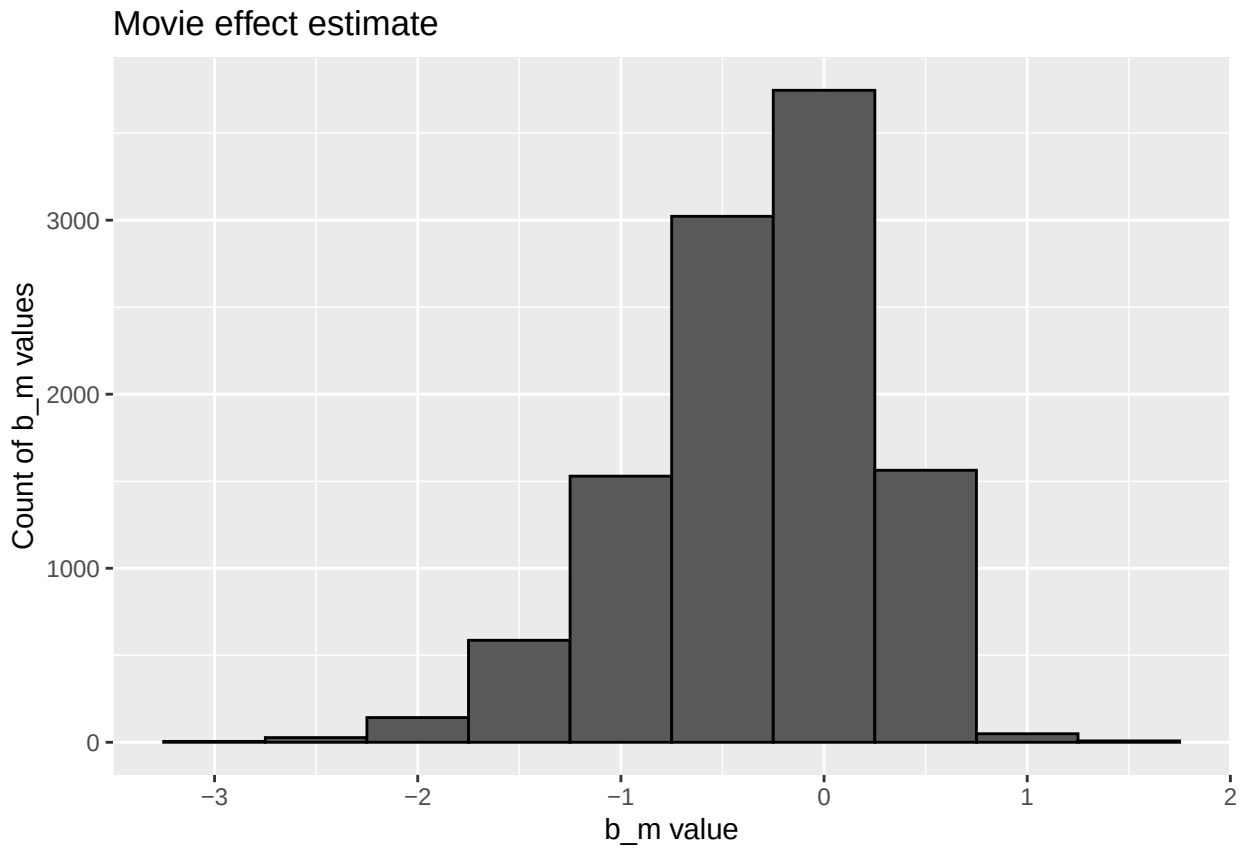
$$\hat{b_m} = mean(Y_{u,i} - \hat{\mu})$$

for each movie i. So we can compute it in the following way:

```
#Calculate the movie effect or bias
movie_bias <- train_set %>%
  group_by(movieId) %>%
  summarize(b_m = mean(rating - mu_train))
```

It can be seen that these estimates have a lot of variation. Note that for a perfect rating b_m needs to be about 1.5 as average rating mu is about 3.5.

```
#Plot of the movie effect b_m
movie_bias %>%
  qplot(b_m, geom ="histogram", bins = 10, data = ., color = I("black"))+
  ggtitle("Movie effect estimate")+
  xlab("b_m value")+
  ylab("Count of b_m values")
```



The model can be predicted and RMSE computed as shown here.

```
#predicting b_m for test_set
b_m_hat<-test_set%>%left_join(movie_bias, by='movieId')%>%.$b_m

# rating prediction and rmse of movie effect
mov_bias_rating<-mu_train+b_m_hat
mov_bias_rmse<-RMSE(test_set$rating, mov_bias_rating)
rmse_summary<-rbind(rmse_summary,
                    data.frame(method = "Movie Bias model", RMSE = mov_bias_rmse))
rmse_summary %>% knitr::kable()
```

method	RMSE
A guess model	1.9432396
An average model	1.0606198
Movie Bias model	0.9432105

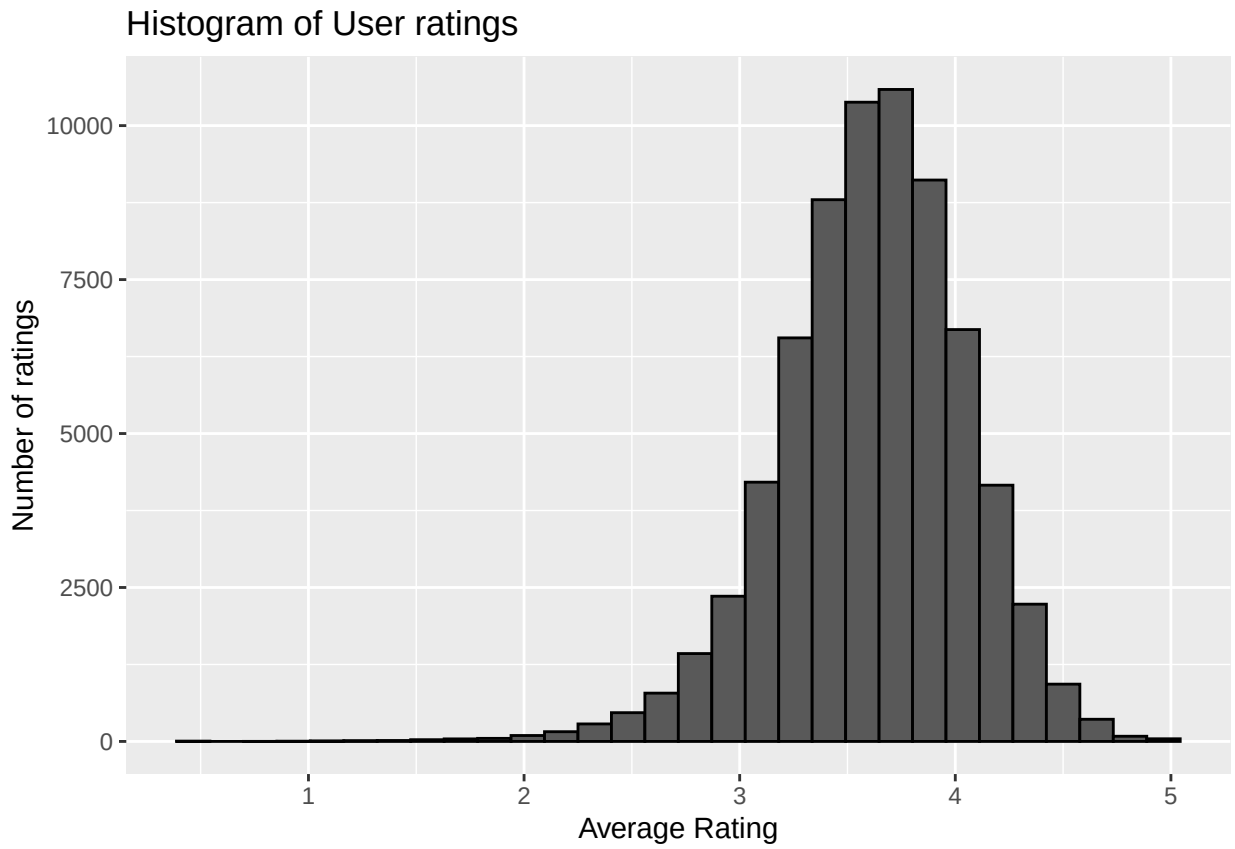
Results:

So with movie effect model the RMSE has improved to 0.94321. But can still be better.

The User effect Model

Similar to the movie effect, there is also user effect. Some users generally give higher ratings than others. Below is a histogram showing the distribution of user ratings of the dataset. It clearly shows that some users are not very interested and others love every movie.

```
train_set%>%
  group_by(userId) %>%
  summarize(u_avg = mean(rating))%>%
  ggplot(aes(u_avg))+geom_histogram(color="black", bins=30)+
  ggtitle("Histogram of User ratings")+xlab("Average Rating")+ylab("Number of ratings")
```



Here are the ten top and bottom ratings by users:

```
# Users with top ratings based on avg by user
train_set%>%
  group_by(userId) %>%
  summarize(count=n(), u_avg = mean(rating))%>%
  arrange(desc(u_avg))%>%
  slice(1:10)
```

```
## # A tibble: 10 x 3
##    userId count u_avg
##     <int> <int> <dbl>
##  1      1    17     5
##  2   7984    16     5
```

```
##  3  11884    16     5
##  4  13027    22     5
##  5  13513    16     5
##  6  13524    19     5
##  7  15575    26     5
##  8  18591    18     5
##  9  18965    44     5
## 10  22045    18     5
```

```
# Bottom 10 rated movies based on avg by user
train_set%>%
  group_by(userId) %>%
  summarize(count=n(), u_avg = mean(rating))%>%
  arrange(u_avg)%>%
  slice(1:10)
```

```
## # A tibble: 10 x 3
##    userId count u_avg
##     <int> <int> <dbl>
##  1  13496    17 0.5
##  2  48146    23 0.5
##  3  49862    16 0.5
##  4  62815    17 0.5
##  5  63381    17 0.5
##  6   6322    15 0.733
##  7   3457    17 1
##  8  24176   124 1
##  9  24490    16 1
## 10  28416    25 1
```

We can further augment our prediction model by inlcuding a term b_u to represent effect of user u. This can be denoted by:

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

where $Y_{u,i}$ is the prediction, $\epsilon_{u,i}$ is the independent error, and μ the mean rating for all movies, and b_m is the bias for each movie and b_u is the bias for each user. So if a not very interested user (-ve b_u) rates a great movie (+ve b_m) the effects counter each other and give us a better prediction.

By using least square to estimate b_m:

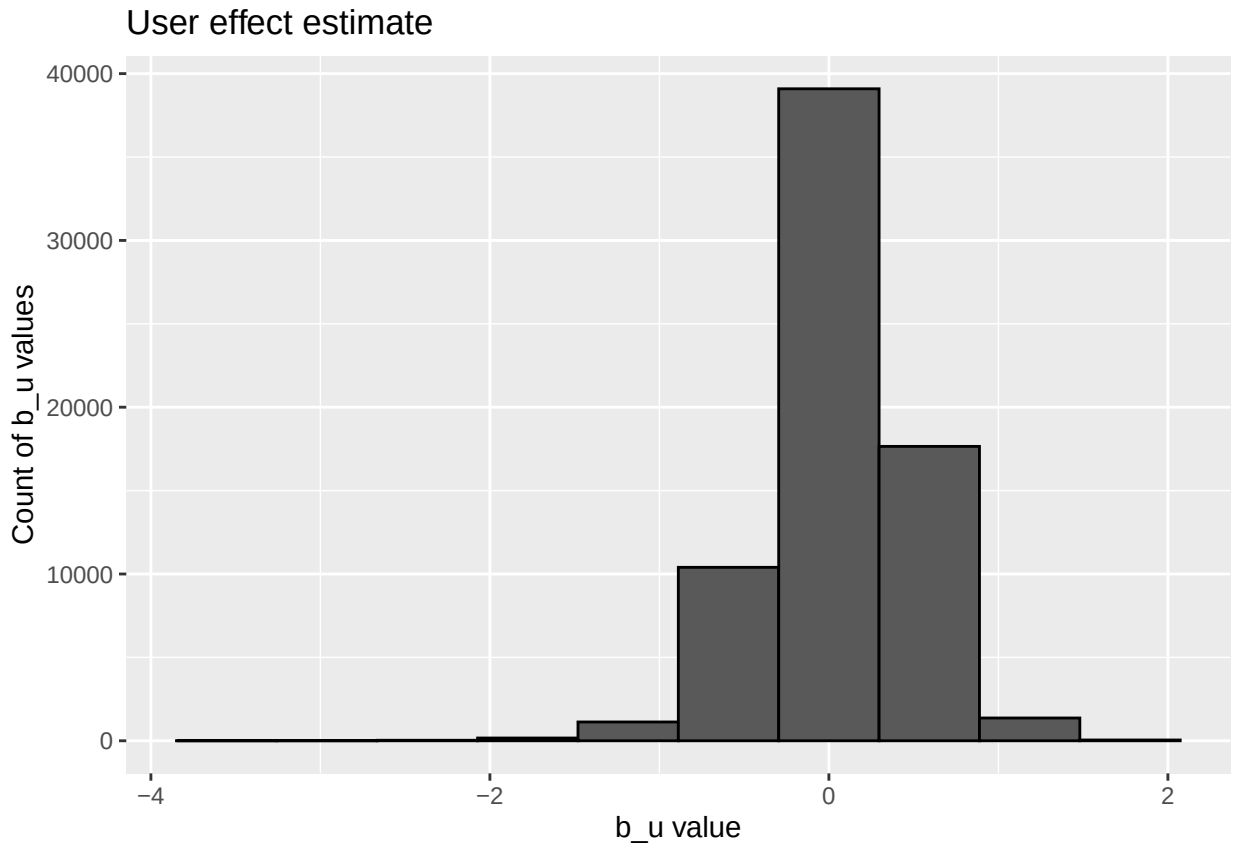
$$\hat{b_u} = mean(Y_{u,i} - b_m - \hat{\mu})$$

for each movie i. It can be computed in the following way:

```
#Calculate the user effect or bias
user_bias <- train_set %>%
  left_join(movie_bias, by='movieId')%>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_train - b_m))
```

Clearly, it can be seen that these estimates have a lot of variation in the b_u estimate as shown below.

```
#Plot of the user effect b_u
user_bias %>%
  qplot(b_u, geom ="histogram", bins = 10, data = ., color = I("black"))+
  ggtitle("User effect estimate")+
  xlab("b_u value")+
  ylab("Count of b_u values")
```



The model can be predicted and RMSE computed as shown here.

```
#predicting b_u for test_set
u_bias_rating<-test_set%>%
  left_join(movie_bias, by='movieId')%>%
  left_join(user_bias, by='userId')%>%
  mutate(b_u_hat=mu_train+b_m+b_u)%>%.$b_u_hat

# rating prediction and rmse of movie effect
u_bias_rmse<-RMSE(test_set$rating, u_bias_rating)
rmse_summary<-rbind(rmse_summary,
                    data.frame(method = "Movie and User Bias model", RMSE = u_bias_rmse))
rmse_summary %>% knitr::kable()
```

method	RMSE
A guess model	1.9432396
An average model	1.0606198

method	RMSE
Movie Bias model	0.9432105
Movie and User Bias model	0.8650391

Results:

By combining the movie effect model and the user effect model, the RMSE has improved to 0.86504. But can still be better.

The Regularization model

Even though there was large variation between movie ratings, the improvement observed by using the Movie effect model was not very significant. To see why that happened, observe the maximum residuals between prediction and true rating obtained for a few movies.

```
test_set %>% left_join(movie_bias, by='movieId') %>%
  mutate(residual = rating - (mu_train + b_m)) %>%
  arrange(desc(abs(residual))) %>%
  slice(1:10) %>%
  pull(title)
```

```
##  [1] "Shawshank Redemption, The (1994)" "Godfather, The (1972)"
##  [3] "Godfather, The (1972)"            "Godfather, The (1972)"
##  [5] "Godfather, The (1972)"            "Godfather, The (1972)"
##  [7] "Godfather, The (1972)"            "Usual Suspects, The (1995)"
##  [9] "Schindler's List (1993)"          "Schindler's List (1993)"
```

These movies have very large residuals, they seem to be blockbusters that received very poor ratings. Now lets look at the top 5 best and worst movies based on $\hat{b_m}$

```
#List of movie names
movie_titles<-edx%>% select(movieId, title)%>% distinct()

#movies with 5 best and worst estimates of b_m along with the rating count
train_set%>%count(movieId)%>%left_join(movie_bias)%>%
  left_join(movie_titles, by='movieId')%>%
  arrange(desc(b_m))%>%
  select(title, b_m, n)%>% slice(1:5)
```

```
## Joining, by = "movieId"
```

```
## # A tibble: 5 x 3
##   title                                  b_m     n
##   <chr>                                <dbl> <int>
## 1 Hellhounds on My Trail (1999)         1.49     1
## 2 Satan's Tango (Sátántangó) (1994)     1.49     2
## 3 Shadows of Forgotten Ancestors (1964) 1.49     1
## 4 Fighting Elegy (Kenka erejii) (1966)  1.49     1
## 5 Sun Alley (Sonnenallee) (1999)        1.49     1
```

```
train_set%>%count(movieId)%>%left_join(movie_bias)%>%
  left_join(movie_titles, by='movieId')%>%
  arrange(b_m)%>%
  select(title, b_m, n)%>% slice(1:5)
```

```
## Joining, by = "movieId"
```

```
## # A tibble: 5 x 3
##   title                                    b_m     n
##   <chr>                                  <dbl> <int>
## 1 Besotted (2001)                        -3.01     2
## 2 Hi-Line, The (1999)                    -3.01     1
## 3 Accused (Anklaget) (2005)              -3.01     1
## 4 Confessions of a Superhero (2007)      -3.01     1
## 5 War of the Worlds 2: The Next Wave (2008) -3.01     2
```

They all seem quite obscure movies and were rated by very few users (1 or 2). These movies with very few user ratings have more uncertainity for prediction. Therefore large estimates for b_m are more likely to result in a higher RMSE. To overcome this problem, regularization can be used.

Regularization constraints the total variablity and allows to penalize the large estimates formed using small sample sizes. To estimate the b's the below equation containing the penalty term is minimized.

$$\frac{1}{N}\sum_{u,i}(y_{u,i}-\mu-b_m)^2+\lambda\sum_i b_m^2$$

The first term is mean squared error and the second is the penalty term that gets larger when many b's are large.

The value of b that minimizes this equation is given by:

$$\hat{b}_m(\lambda)=\frac{1}{\lambda+n_i}\sum_{n_i}(Y_{u,i}-\hat{\mu})$$

where the sum is done for number of ratings b for movie i.

When n_i is low λ dominates and shrinks the estimate and when n_i is large λ is practically ignored. The larger λ is, the more penalty will shrink. It is a tuning parameter and needs to be selected without using the test set.

To compute the regularized estimate, start with $\lambda = 3$ (needs to be fine tuned later)

```
lambda<-3
movie_regn <- train_set %>%
  group_by(movieId) %>%
  summarize(b_reg_m = sum(rating - mu_train)/(n()+lambda), n_i = n())
```

To see the effect of the shrinking observe the top 10 best and worst movies based on the estimates:

```
#top 10 best movies based on b_m
train_set %>%
  count(movieId) %>%
  left_join(movie_regn, by = "movieId") %>%
  left_join(movie_titles, by = "movieId") %>%
```

```
  arrange(desc(b_reg_m)) %>%
  slice(1:10) %>%
  pull(title)
```

```
##  [1] "Shawshank Redemption, The (1994)"
##  [2] "More (1998)"
##  [3] "Godfather, The (1972)"
##  [4] "Usual Suspects, The (1995)"
##  [5] "Schindler's List (1993)"
##  [6] "Casablanca (1942)"
##  [7] "Rear Window (1954)"
##  [8] "Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)"
##  [9] "Third Man, The (1949)"
## [10] "Double Indemnity (1944)"
```

```
#top 10 worst movies based on b_m
train_set %>%
  count(movieId) %>%
  left_join(movie_regn, by = "movieId") %>%
  left_join(movie_titles, by = "movieId") %>%
  arrange(b_reg_m) %>%
  slice(1:10) %>%
  pull(title)
```

```
##  [1] "From Justin to Kelly (2003)"
##  [2] "SuperBabies: Baby Geniuses 2 (2004)"
##  [3] "Pokémon Heroes (2003)"
##  [4] "Disaster Movie (2008)"
##  [5] "Pokemon 4 Ever (a.k.a. Pokémon 4: The Movie) (2002)"
##  [6] "Glitter (2001)"
##  [7] "Barney's Great Adventure (1998)"
##  [8] "Carnosaur 3: Primal Species (1996)"
##  [9] "Gigli (2003)"
## [10] "Yu-Gi-Oh! (2004)"
```

Clearly, these make much more sense. The final RMSE can be calculated with the test set.

```
#predicting b_m for test_set
b_reg_m_hat<-test_set%>%left_join(movie_regn, by='movieId')%>%.$b_reg_m

# rating prediction and rmse of movie effect
mov_regn_rating<-mu_train+b_reg_m_hat
mov_regn_rmse<-RMSE(test_set$rating, mov_regn_rating)
rmse_summary<-rbind(rmse_summary,
                    data.frame(method = "Regularization Movie Bias model",
                               RMSE =mov_regn_rmse))
rmse_summary %>% knitr::kable()
```

method	RMSE
A guess model	1.9432396
An average model	1.0606198

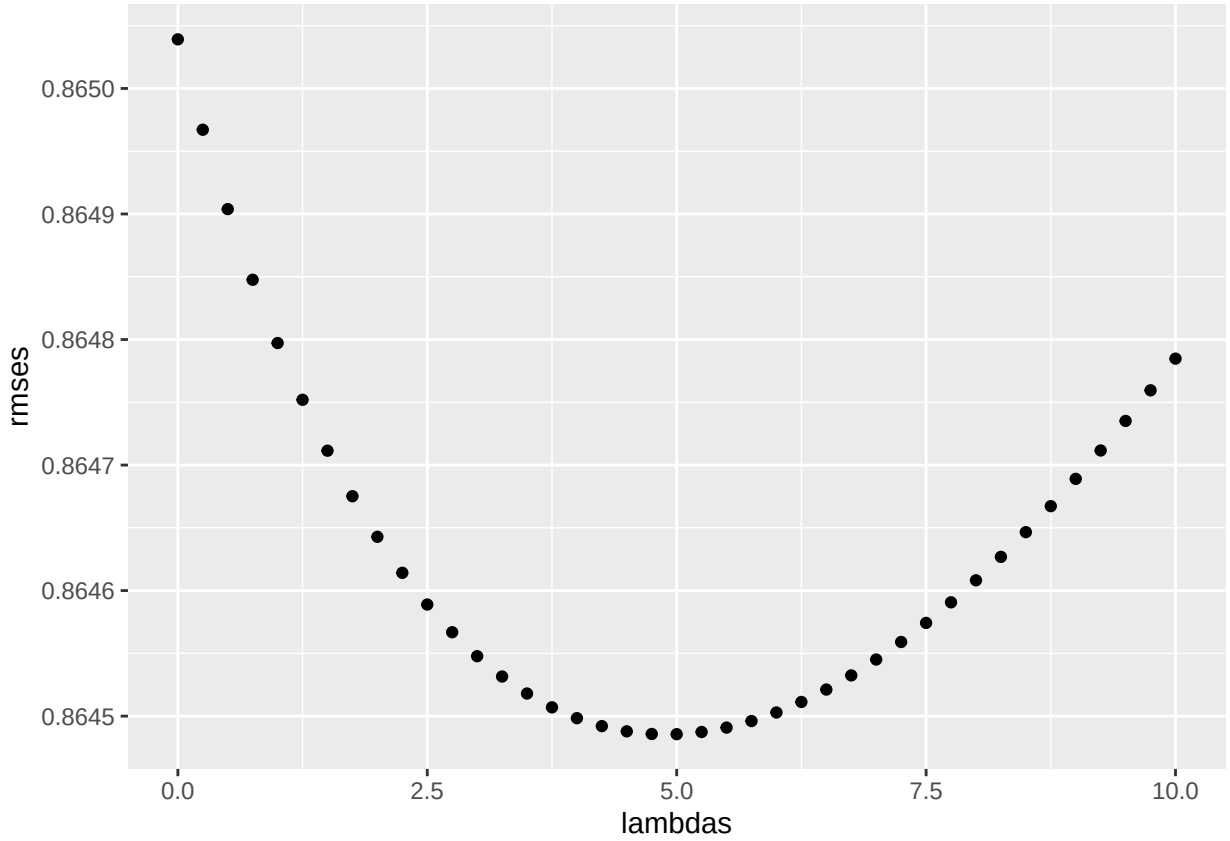
method	RMSE
Movie Bias model	0.9432105
Movie and User Bias model	0.8650391
Regularization Movie Bias model	0.9431897

The regularization model shows improvement and looks promising to extend it to model the user effects as well. Also, λ can be fine tuned using cross-validation.

For regularization the below equation is minimised:

$$\frac{1}{N}\sum_{u,i}(y_{u,i}-\mu-b_m-b_u)^2+\lambda(\sum_i b_m^2+\sum_u b_u^2)$$

```
#Fine tune model to get optimum lambda and extend to model user efefcts
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  mu<- mean(train_set$rating)
  b_m <- train_set %>%
    group_by(movieId) %>%
    summarize(b_m = sum(rating - mu)/(n()+l))
  b_u <- train_set %>%
    left_join(b_m, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_m - mu)/(n()+l))
  ratings_hat <-
    test_set %>%
    left_join(b_m, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_m + b_u) %>%
    .$pred
  return(RMSE(ratings_hat, test_set$rating))
})

#Plot of lambdas Vs rmse
qplot(lambdas, rmses)
```



This plot shows the optimum value of λ as 5 for best RMSE results. The final RMSE can be calculated by using this λ :

```
#Choosing the optimum lambda value
lambda <- lambdas[which.min(rmses)]
lambda
```

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

```
#Choosing the minimum RMSE
mov_user_regn_rmse<-min(rmses)
rmse_summary<-rbind(rmse_summary,
                    data.frame(method = "Regularization Movie and User Bias model",
                               RMSE = mov_user_regn_rmse))
rmse_summary %>% knitr::kable()
```

method	RMSE
A guess model	1.9432396
An average model	1.0606198
Movie Bias model	0.9432105
Movie and User Bias model	0.8650391
Regularization Movie Bias model	0.9431897
Regularization Movie and User Bias model	0.8644856

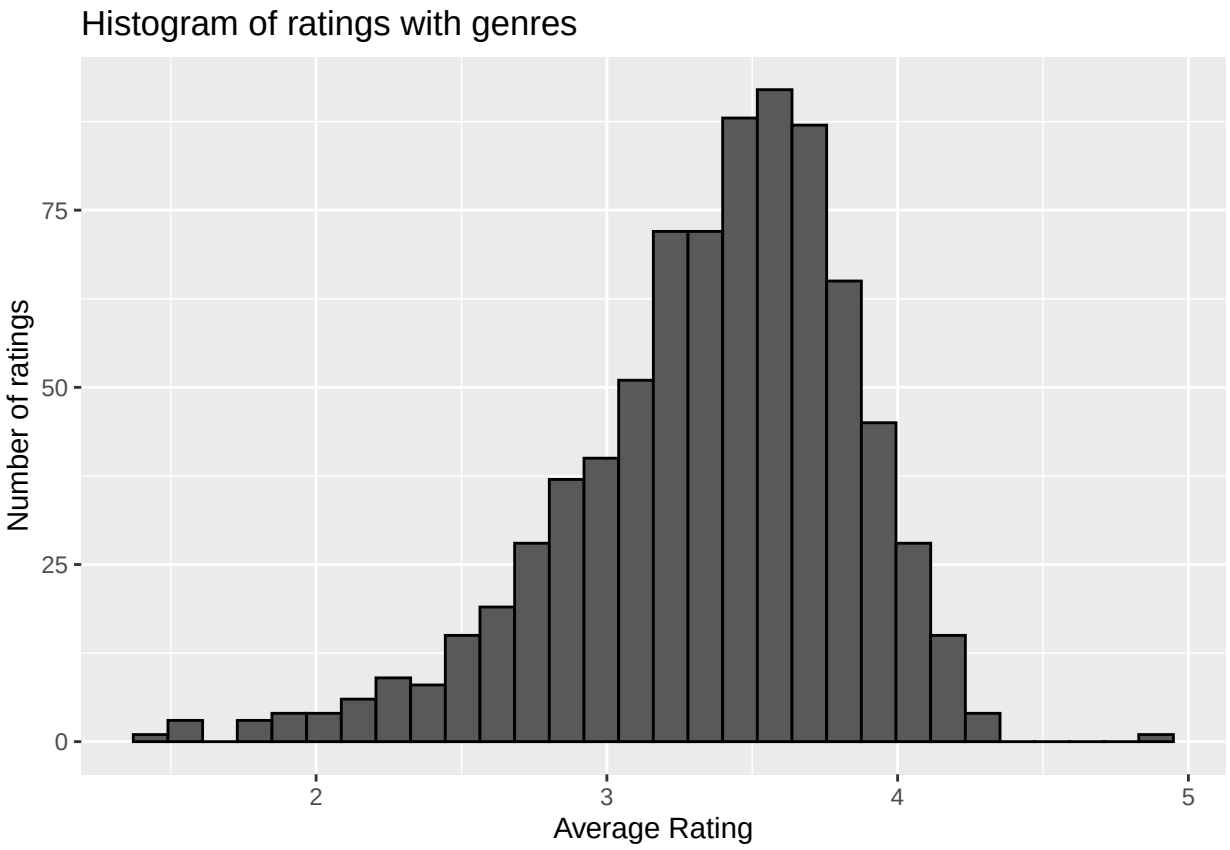
Result:

Clearly it can be seen that the RMSE with regularization has improved to 0.86449, which is lower than the ideal goal of 0.86490. However, more models can be evaluated to further bring it down.

The Genre effect Model

So far, the movie effects and user effects have been modeled. However, it can also be seen that ratings vary very much with the genre of the movie.

```
train_set %>% group_by(genres) %>%
  summarize(g_avg = mean(rating))%>%filter(n()>=100)%>%
  ggplot(aes(g_avg))+geom_histogram(color="black", bins=30)+
  ggtitle("Histogram of ratings with genres")+
  xlab("Average Rating")+ylab("Number of ratings")
```



The genres in this model are treated as a combination of all genres related to the movie and have not been analyzed by separating into individual genres. For example, if a movie has a genre "Romance|Comedy" it is treated as a new genre "romantic comedy" and not separated into romance and comedy for the sake of simplicity.

It can be clearly seen that some genres have lower ratings and some have higher ratings. Below is the list of top 10 best genres and worst genres.

```
#Top 10 best genres
train_set %>% group_by(genres) %>%
```

```
  summarize(g_avg = mean(rating))%>%
  arrange(desc(g_avg))%>%
  slice(1:10)
```

```
## # A tibble: 10 x 2
##    genres                                    g_avg
##    <chr>                                     <dbl>
##  1 Animation|IMAX|Sci-Fi                      4.92
##  2 Drama|Film-Noir|Romance                    4.30
##  3 Action|Crime|Drama|IMAX                    4.30
##  4 Animation|Children|Comedy|Crime            4.27
##  5 Film-Noir|Mystery                          4.24
##  6 Film-Noir|Romance|Thriller                 4.22
##  7 Crime|Film-Noir|Mystery                    4.22
##  8 Crime|Film-Noir|Thriller                   4.21
##  9 Crime|Mystery|Thriller                     4.20
## 10 Action|Adventure|Comedy|Fantasy|Romance    4.20
```

```
#Top 10 worst genres
train_set %>% group_by(genres) %>%
  summarize(g_avg = mean(rating))%>%
  arrange(g_avg)%>%
  slice(1:10)
```

```
## # A tibble: 10 x 2
##    genres                                      g_avg
##    <chr>                                       <dbl>
##  1 Documentary|Horror                           1.46
##  2 Action|Animation|Comedy|Horror               1.5
##  3 Comedy|Film-Noir|Thriller                    1.5
##  4 Action|Horror|Mystery|Thriller               1.61
##  5 Action|Drama|Horror|Sci-Fi                   1.75
##  6 Adventure|Drama|Horror|Sci-Fi|Thriller       1.81
##  7 Action|Horror|Mystery|Sci-Fi                 1.82
##  8 Action|Children|Comedy                       1.88
##  9 Action|Adventure|Children                    1.91
## 10 Adventure|Animation|Children|Fantasy|Sci-Fi  1.94
```

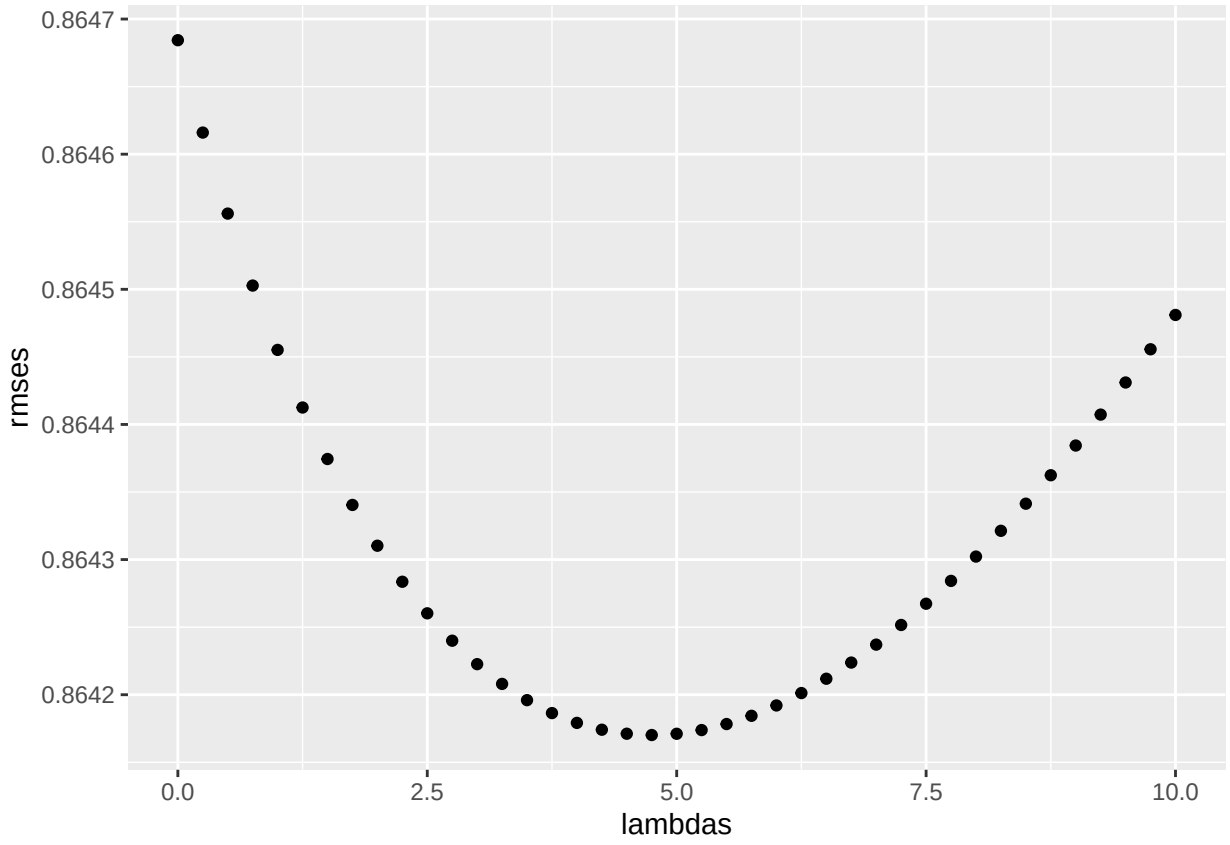
Extending regularization to model the genre effect the equation can be modified to:

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

The model can be created using:

```
# fine tuning of lambda for genre effect
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  mu<- mean(train_set$rating)
  b_m <- train_set %>%
    group_by(movieId) %>%
    summarize(b_m = sum(rating - mu)/(n()+l))
  b_u <- train_set %>%
```

```
    left_join(b_m, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_m - mu)/(n()+l))
  b_g <- train_set %>%
    left_join(b_m, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_m - b_u - mu)/(n()+l))
  ratings_hat <-
    test_set %>%
    left_join(b_m, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_m + b_u + b_g) %>%
    .$pred
  return(RMSE(ratings_hat, test_set$rating))
})

#Plot of lambdas Vs rmse
qplot(lambdas, rmses)
```



This plot shows the optimum value of λ is 4.75 for best RMSE results. The final RMSE can be calculated.

```
#Choosing the optimum lambda value
lambda <- lambdas[which.min(rmses)]
lambda
```

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

```
#Choosing the minimum RMSE
mov_user_gen_regn_rmse<-min(rmses)
rmse_summary<-rbind(rmse_summary,
                    data.frame(method = "Regularization Movie, User and Genre Bias model",
                               RMSE = mov_user_gen_regn_rmse))
rmse_summary %>% knitr::kable()
```

method	RMSE
A guess model	1.9432396
An average model	1.0606198
Movie Bias model	0.9432105
Movie and User Bias model	0.8650391
Regularization Movie Bias model	0.9431897
Regularization Movie and User Bias model	0.8644856
Regularization Movie, User and Genre Bias model	0.8641702

Result:

A comparison of the table shows that RMSE with regualrization for genre effect has further improved to 0.86417 and is still less than the ideal goal of 0.86490. The improvement of adding the genre model is not very significant and the reason for this could be that the genres were essentially combined. A more appropriate model would be to seperate into each genre and also use Matrix Factorization for predicting. However, in the interest of time it is not pursued currently.

Final Results

Based on the analysis of various models in the previous section the best model for predicted ratings is the one with Regularization to model the movie, user and genre effects.

This table shows the incremental improvement of the models by measuring the performance on the test set created from the edx dataset.

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

method	RMSE
A guess model	1.9432396
An average model	1.0606198
Movie Bias model	0.9432105
Movie and User Bias model	0.8650391
Regularization Movie Bias model	0.9431897
Regularization Movie and User Bias model	0.8644856
Regularization Movie, User and Genre Bias model	0.8641702

Applying the model to the validation set would mean that the different parameters be obtained using the train set and not using the validation set.

```
# Final model: use parameters from edx set
l<-lambda
# use the lambda obtained above to apply on the validation set for predicting output
mu<- mean(train_set$rating)
b_m <- train_set %>%
  group_by(movieId) %>%
  summarize(b_m = sum(rating - mu)/(n()+l))
b_u <- train_set %>%
  left_join(b_m, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_m - mu)/(n()+l))
b_g <- train_set %>%
  left_join(b_m, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - b_u - b_m - mu)/(n()+l))
```

The different b's have been obtained, hence, applying them to the validation dataset:

```
# Final model application: use validation set to predict final ratings
final_rating <- validation %>%
  left_join(b_m, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  mutate(pred = mu + b_m + b_u + b_g) %>%
  .$pred

rmse_validation <- RMSE(final_rating, validation$rating)
```

```
# Final predicted ratings can be found in final_rating

#Final RMSE to be reported
rmse_validation
```

```
## [1] 0.864852
```

The final predicted rating can be found in the vector final_rating. It can be seen that the RMSE value is 0.864852 which is lower than the ideal RMSE for this project. Also, observe that this value is higher than the corresponding RMSE obtained from the training set and that is expected because the parameters are fine tuned for train set.

Final RMSE reported for validation set: 0.864852

Conclusion

The movie recommendation system has been successfully modeled using machine learning algorithms and the final RMSE has been reported.

Modeling started with a guessing model that gave RMSE 1.9432 and it got better with the average model to 1.0606. By using movie and user effects the RMSE was brought down to 0.8650. Since some features have large effect on residuals, regularization was used and also added genre effects which further brought down the RMSE to 0.86417. Having found an optimum model, it was then applied to the validation set and found that the final RMSE is 0.864852.

Finally, for future work, matrix factorization method could be evaluated to further improve the prediction results.

References

Rafael A. Irizarry. (2020). Introduction to Data Science: Data Analysis and Prediction Algorithms with R.

https://grouplens.org/datasets/movielens/

https://www.rdocumentation.org/

https://rmarkdown.rstudio.com/