HarvardX: PH125.9x Data Science
Project Submission: Movielens Capstone Project

Santhosh Channa

5/13/2019

Contents

1 Overview

This report is part of the capstone project of the EdX course 'HarvardX: PH125.9x Data Science: Capstone'. The goal is to demonstrate that the student acquired skills with the R programming language in the field of datascience to actually solve real world problems. The task is to analyze a dataset called 'MovieLens' which contains millions of movieratings by users. The insights from this analysis are used to generate predictions of movies which are compared with the actual ratings to check the quality of the prediction algorithm.

1.1 Introduction

Recommendation systems use ratings that *users* have given to *items* to make specific recommendations. Companies that sell many products to many 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 to a specific item. Items for which a high rating is predicted for a given user are then recommended to that user.

The same could be done for other items, as movies for instance in our case. Recommendation systems are one of the most used models in machine learning algorithms. In fact the success of Netflix is said to be based on its strong recommendation system.

For this project we will focus on creating a movie recommendation system using the 10M version of MovieLens dataset, collected by GroupLens Research and made available by edX.

1.2 Aim of the project

The aim of this project is to train a machine learning algorithm that predicts user ratings *(from 0.5 to 5 stars)* using the inputs of a provided **edx** data set to predict movie ratings in a provided **validation** data set.

We will be using **RMSE** (Root Mean Square Error).

Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are; RMSE is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit. Root mean square error is commonly used in climatology, forecasting, and regression analysis to verify experimental results.

RMSE is one of the most used measure of the differences between values predicted by a model and the values observed. RMSE is a measure of accuracy, to compare forecasting errors of different models for a particular dataset, a lower RMSE is better than a higher one. The effect of each error on RMSE is proportional to the size of the squared error; thus larger errors have a disproportionately large effect on RMSE. Consequently, RMSE is sensitive to outliers.

In this project we will develop four models that will be compared using their resulting RMSE in order to assess their quality. The evaluation criteria for this algorithm is a RMSE expected to be **<= 0.87750**.

The function that computes the RMSE for vectors of ratings and their corresponding predictors will be the following:

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

Here is the R Code:

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

Finally, the best resulting model will be used to predict the movie ratings.

1.3 Dataset

MovieLens data set was downloaded from the course material and created using the code below:

```
# Install and load libraries required
if(!require(tidyverse)) install.packages("tidyverse")
if(!require(caret)) install.packages("caret")
if(!require(devtools)) install.packages("devtools")
if(!require(sqldf)) install.packages("sqldf")
library(devtools)
devtools::install_github("collectivemedia/tictoc")
library(tictoc)
library(tidyverse)
library(ggplot2)
library(ggrepel)
library(knitr)
library(dplyr)
library(caret)

# Get the edx and validataion data set from either Google Drive or One Drive
tic("Loading edx data set...")
```

```r
edx <- readRDS("data/edx.rds")
toc()
```

```
## Loading edx data set...: 15.541 sec elapsed
```

```r
tic("Loading validation data set...")
validation <-
  readRDS("data/validation.rds")
toc()
```

```
## Loading validation data set...: 1.096 sec elapsed
```

2 Data Analysis

2.1 Overview of dataset

Let us glance through the data sets we just created to make sure if both the sets has the same attributes

```r
# Information about edx data set
class(edx)
```

```
## [1] "data.frame"
```

```r
glimpse(edx)
```

```
## Observations: 9,000,055
## Variables: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 37...
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5...
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 83898339...
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (19...
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|D...
```

```r
# Information about validation data set
class(validation)
```

```
## [1] "data.frame"
```

```r
glimpse(validation)
```

```
## Observations: 999,999
## Variables: 6
## $ userId    <int> 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5...
## $ movieId   <dbl> 231, 480, 586, 151, 858, 1544, 590, 4995, 34, 432, 4...
## $ rating    <dbl> 5.0, 5.0, 5.0, 3.0, 2.0, 3.0, 3.5, 4.5, 5.0, 3.0, 3....
## $ timestamp <int> 838983392, 838983653, 838984068, 868246450, 86824564...
## $ title     <chr> "Dumb & Dumber (1994)", "Jurassic Park (1993)", "Hom...
## $ genres    <chr> "Comedy", "Action|Adventure|Sci-Fi|Thriller", "Child...
```

Summary information of edx dataset:

```r
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
##
##
##
```

2.2 Understanding the given dataset

Distinct Movies in the edx data set:

```
# using sqldf
tic("Distinct Movies in edx data set -Using sqldf")
sqldf("select count(distinct(movieId)) from edx")
```

```
##   count(distinct(movieId))
## 1                    10677
```

```
toc()
```

```
## Distinct Movies in edx data set -Using sqldf: 12.983 sec elapsed
```

```
# using R
tic("Distinct Movies in edx data set -Using R")
n_distinct(edx$movieId)
```

```
## [1] 10677
```

```
toc()
```

```
## Distinct Movies in edx data set -Using R: 0.388 sec elapsed
```

Distinct Users in the edx data set:

```
n_distinct(edx$userId)
```

```
## [1] 69878
```

Distinct Users and Movies in the edx data set:

```
tic("Distinct Users and Movies in the edx data set -R")
edx %>% summarize(unique_users = n_distinct(userId), unique_movies = n_distinct(movieId))
```

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

```
toc()
```

```
## Distinct Users and Movies in the edx data set -R: 0.568 sec elapsed
```

```
# using sqldf to acheive the same. Note SQLDF is slow
tic("Distinct Users and Movies in the edx data set -sqldf")
```

```r
sqldf("select count(distinct(userId)) unique_users,
      count(distinct(movieId)) unique_movies from edx")
```

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

```r
toc()
```

```
## Distinct Users and Movies in the edx data set -sqldf: 13.899 sec elapsed
```

Distinct Genre's in the data set:

```r
#tic("Number of Ratings per Genre...")
#edx %>% separate_rows(genres, sep = "\\|") %>%
#     group_by(genres) %>%
#     summarize(count = n()) %>%
#     arrange(desc(count))
#toc()
# I found the below approach is much faster than above code
tic("Number of Ratings per Genre. Method 2...")
tic("Step#1 - Creating edx_by_genre data set per genre")
edx_by_genre <- edx %>% separate_rows(genres, sep = "\\|")
toc()
```

```
## Step#1 - Creating edx_by_genre data set per genre: 54.075 sec elapsed
```

```r
tic("Step#2 - Distinct Genres from edx_by_genre set")
data.frame(table(edx_by_genre$genres))
```

```
##                  Var1    Freq
## 1  (no genres listed)       7
## 2              Action 2560545
## 3           Adventure 1908892
## 4           Animation  467168
## 5            Children  737994
## 6              Comedy 3540930
## 7               Crime 1327715
## 8         Documentary   93066
## 9               Drama 3910127
## 10            Fantasy  925637
## 11          Film-Noir  118541
## 12             Horror  691485
## 13               IMAX    8181
## 14            Musical  433080
## 15            Mystery  568332
## 16            Romance 1712100
## 17             Sci-Fi 1341183
## 18           Thriller 2325899
## 19                War  511147
## 20            Western  189394
```

```r
toc()
```

```
## Step#2 - Distinct Genres from edx_by_genre set: 7.026 sec elapsed
```

```r
toc()
```

```
## Number of Ratings per Genre. Method 2...: 61.104 sec elapsed
```

```r
tic("Distinct Genre's")
n_distinct(edx_by_genre$genres)
```

```
## [1] 20
```

```r
toc()
```

```
## Distinct Genre's: 0.648 sec elapsed
```

```r
# Number of Unique Movies per Genre

tic("Number of Unique Movies per Genre")
edx_by_genre %>% group_by(genres) %>%
    summarize(count = n_distinct(movieId)) %>%
    arrange(desc(count))
```

```
## # A tibble: 20 x 2
##    genres             count
##    <chr>              <int>
##  1 Drama               5336
##  2 Comedy              3703
##  3 Thriller            1705
##  4 Romance             1685
##  5 Action              1473
##  6 Crime               1117
##  7 Adventure           1025
##  8 Horror              1013
##  9 Sci-Fi               754
## 10 Fantasy              543
## 11 Children             528
## 12 War                  510
## 13 Mystery              509
## 14 Documentary          481
## 15 Musical              436
## 16 Animation            286
## 17 Western              275
## 18 Film-Noir            148
## 19 IMAX                  29
## 20 (no genres listed)     1
```

```r
toc()
```

```
## Number of Unique Movies per Genre: 2.774 sec elapsed
```

```r
# SQL Version:
# tic("SQL Version:")
# sqldf("select genres, count(distinct movieId) tot
#         from edx_by_genre group by genres order by tot desc")
# toc()
```

Which Movie has the greatest number of Ratings?

```r
tic("Which movie has the greatest number of ratings?")
edx %>% group_by(movieId, title) %>%
     summarize(count = n()) %>%
```

```
    arrange(desc(count))
```

```
## # A tibble: 10,677 x 3
## # Groups:   movieId [10,677]
##    movieId title                                                        count
##      <dbl> <chr>                                                        <int>
##  1     296 Pulp Fiction (1994)                                          31362
##  2     356 Forrest Gump (1994)                                          31079
##  3     593 Silence of the Lambs, The (1991)                             30382
##  4     480 Jurassic Park (1993)                                         29360
##  5     318 Shawshank Redemption, The (1994)                             28015
##  6     110 Braveheart (1995)                                            26212
##  7     457 Fugitive, The (1993)                                         25998
##  8     589 Terminator 2: Judgment Day (1991)                            25984
##  9     260 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (19~ 25672
## 10     150 Apollo 13 (1995)                                             24284
## # ... with 10,667 more rows
```

```
toc()
```

```
## Which movie has the greatest number of ratings?: 1.122 sec elapsed
```

2.3 Visualyzing the data

Distribution of Ratings. . .

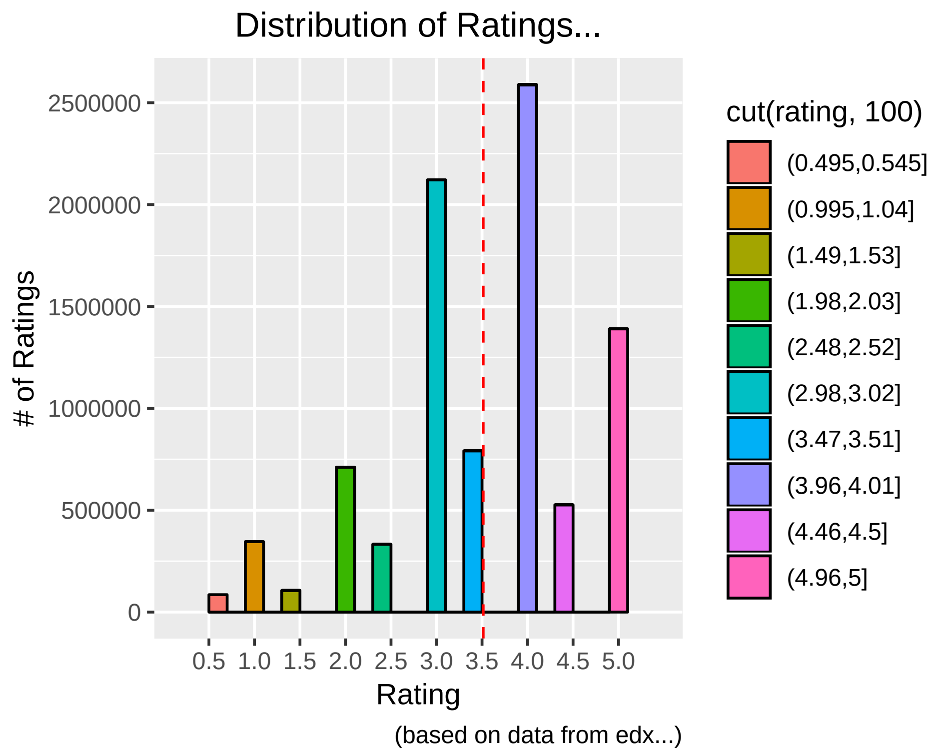
```
edx %>%
    group_by(rating) %>%
    summarize(count = n()) %>%
    select(Rating = rating, Number_of_Movies = count) %>%
    arrange(desc(Rating))
```

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

Plot the Distribution of Ratings. . .

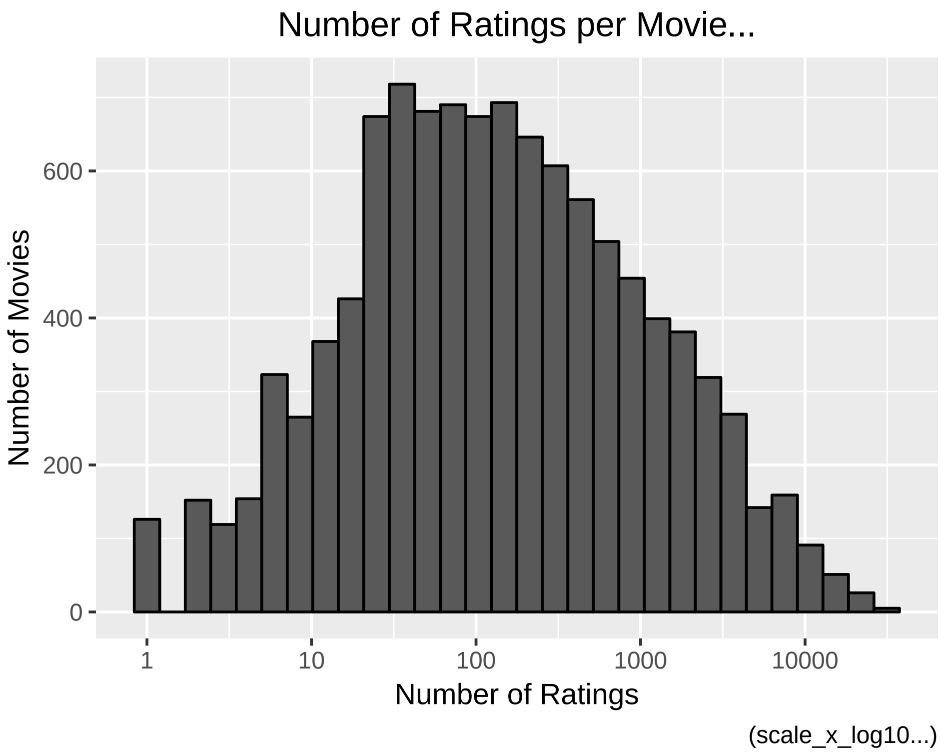
```
edx %>%
    ggplot(aes(rating, fill = cut(rating, 100))) +
    geom_histogram(binwidth = .20, color = "black") +
    scale_x_discrete(limits = c(seq(.5, 5, .5))) +
    scale_y_continuous(breaks = c(seq(0, 2500000, 500000))) +
    geom_vline(xintercept = mean(edx$rating), col = "red", linetype = "dashed") +
```

```
    ggtitle("Distribution of Ratings...") +
    theme(plot.title = element_text(hjust = 0.5)) +
    labs(x = "Rating") +
    labs(y = "# of Ratings") +
    labs(caption = "(based on data from edx...)")
```



Number of Ratings per Movie...

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



As you notice, there are quite a few movies rated very few times. Lets see the movies which has less than 10 ratings:

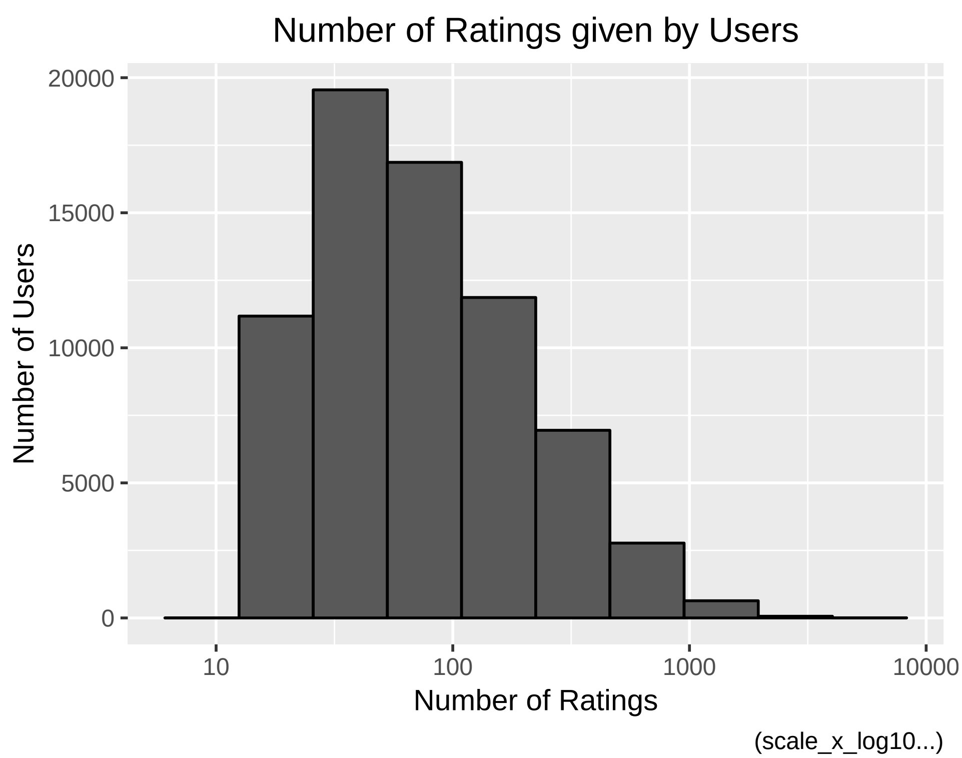
```
edx %>%
    group_by(title) %>%
    summarize(count = n()) %>%
    filter(count <= 10) %>% #change the value from 10 to 1 to see movies rated only once
    left_join(edx, by = "title") %>%
    select(Movie = title, Rating = rating, Number_Of_Ratings = count) %>%
    arrange(Number_Of_Ratings, desc(Rating)) %>%
    slice(1:20) %>%
    knitr::kable()
```

Movie	Rating	Number_Of_Ratings
Blue Light, The (Das Blaue Licht) (1932)	5.0	1
Fighting Elegy (Kenka erejii) (1966)	5.0	1
Hellhounds on My Trail (1999)	5.0	1
Shadows of Forgotten Ancestors (1964)	5.0	1
Sun Alley (Sonnenallee) (1999)	5.0	1
Bad Blood (Mauvais sang) (1986)	4.5	1
Demon Lover Diary (1980)	4.5	1
Kansas City Confidential (1952)	4.5	1
Ladrones (2007)	4.5	1
Man Named Pearl, A (2006)	4.5	1
Mickey (2003)	4.5	1
Please Vote for Me (2007)	4.5	1
Testament of Orpheus, The (Testament d'Orphée) (1960)	4.5	1
Tokyo! (2008)	4.5	1

Movie	Rating	Number_Of_Ratings
Valerie and Her Week of Wonders (Valerie a týden divu) (1970)	4.5	1
Bellissima (1951)	4.0	1
David Holzman's Diary (1967)	4.0	1
Deadly Companions, The (1961)	4.0	1
Family Game, The (Kazoku gêmu) (1983)	4.0	1
Fists in the Pocket (I Pugni in tasca) (1965)	4.0	1

Number of Ratings given by Users:

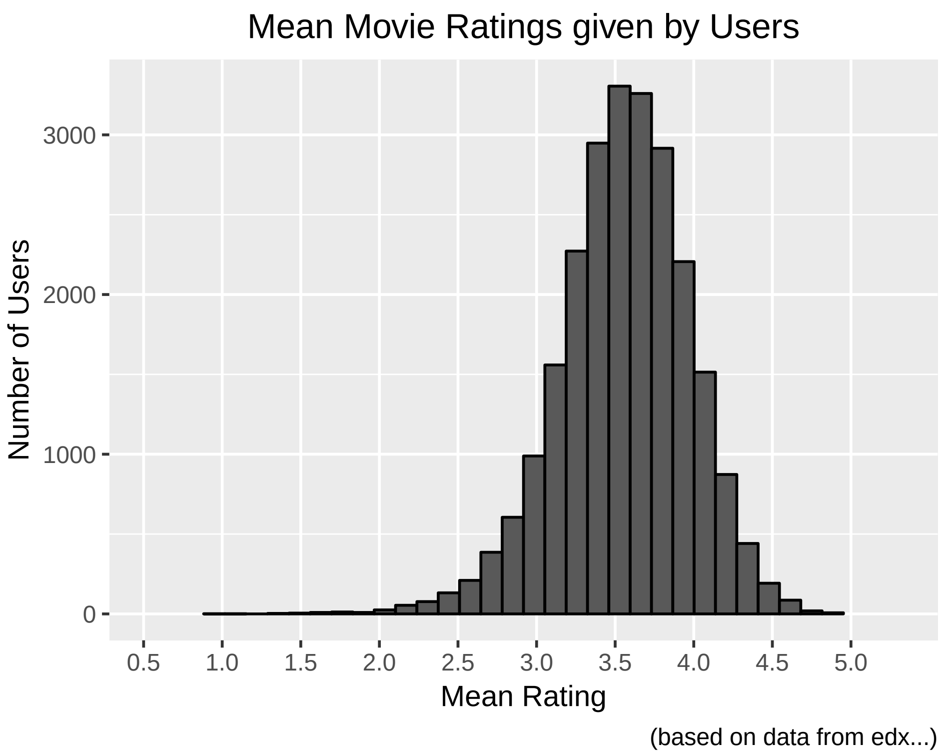
```
edx %>%
    group_by(userId) %>%
    summarize(number_of_ratings = n()) %>%
    ggplot(aes(number_of_ratings)) +
    geom_histogram(bins = 10, color = "black") +
    scale_x_log10() +
    xlab("Number of Ratings") +
    ylab("Number of Users") +
    ggtitle("Number of Ratings given by Users") +
    theme(plot.title = element_text(hjust = 0.5)) +
    labs(caption = "(scale_x_log10...)")
```



Mean Movie Ratings given by Users: The visualization below includes only users that have rated at least 100 Movies.

```
edx %>%
    group_by(userId) %>%
```

```r
    filter(n() >= 100) %>%
    summarize(b_u = mean(rating)) %>%
    ggplot(aes(b_u)) +
    geom_histogram(bins = 30, color = "black") +
    xlab("Mean Rating") +
    ylab("Number of Users") +
    ggtitle("Mean Movie Ratings given by Users") +
    scale_x_discrete(limits = c(seq(0.5,5,0.5))) +
    theme(plot.title = element_text(hjust = 0.5)) +
  labs(caption = "(based on data from edx...)")
```



2.4 Modelling Approach

2.4.1 Model#1: Average Movie Rating

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

```r
tic("Mean Rating of edx data set")
mu <- mean(edx$rating)
toc()
```

```
## Mean Rating of edx data set: 0.02 sec elapsed
```

```r
# Naive RMSE of validataion data set
tic("Naive RMSE of validataion data set")
naive_rmse <- RMSE(validation$rating, mu)
toc()
```

```
## Naive RMSE of validataion data set: 0.021 sec elapsed
```

```r
naive_rmse
```

```
## [1] 1.061202
```

```r
# Persist prediction results
tic("Persist prediction results ")
rmse_results <- data_frame(method = "Model#1: Average Movie Rating", RMSE = naive_rmse)
rmse_results %>% knitr::kable()
```

method	RMSE
Model#1: Average Movie Rating	1.061202

2.4.2 Model#2: Movie Effect

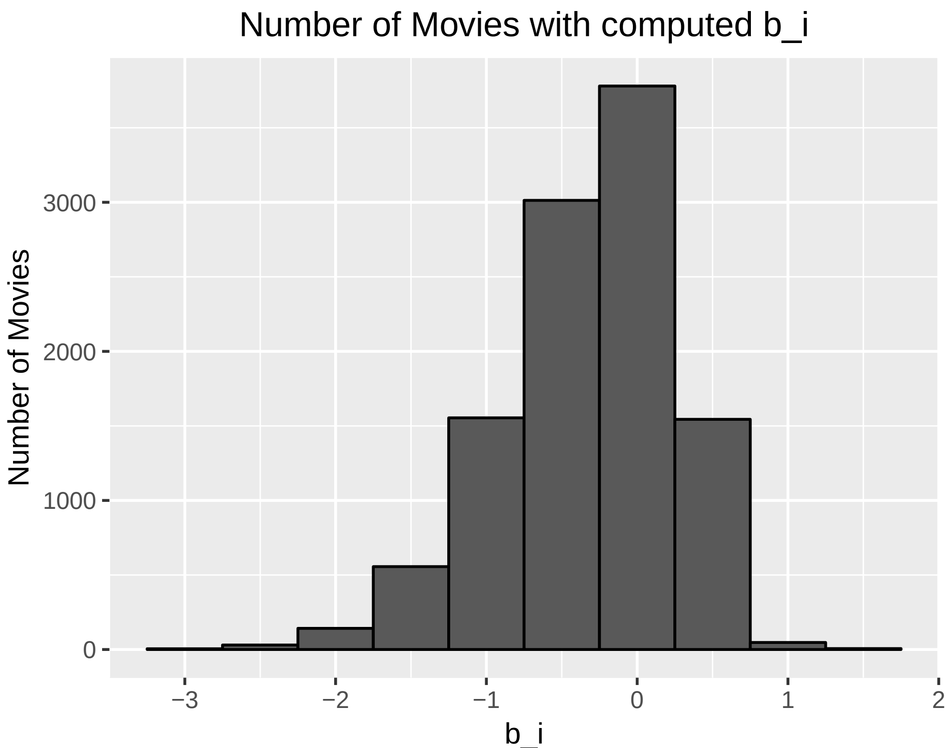
To improve above model we focus on the fact that, from experience, we know that some movies are just generally rated higher than others. Higher ratings are mostly linked to popular movies among users and the opposite is true for unpopular movies. We compute the estimated deviation of each movies' mean rating from the total mean of all movies μ. The resulting variable is called "b" (as bias) for each movie "i" b_i, that represents average ranking for movie i:

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

```r
# Substract mu from movie rating -getting b_i
tic("Substract mu from movie rating -getting b_i")
movie_avgs <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = mean(rating - mu))
toc()
```

```
## Substract mu from movie rating -getting b_i: 0.928 sec elapsed
```

```r
# Generate a plot with computed b_i
tic("Generate a plot with computed b_i")
movie_avgs %>%
    qplot(b_i, geom = "histogram", bins = 10, data = ., color = I("black"),
          ylab = "Number of Movies",
          main = "Number of Movies with computed b_i") +
    theme(plot.title = element_text(hjust = 0.5))
```




```
toc()
```

```
## Generate a plot with computed b_i: 0.094 sec elapsed
```

The histogram is left skewed in the above, implying that more movies have negative effects. This is called the penalty term movie effect. Our prediction improve once we predict using this model.

```
# Validate with validation data set
tic("Validate with validation data set")
predicted_ratings <- mu + validation %>%
    left_join(movie_avgs, by = 'movieId') %>%
    pull(b_i)
toc()
```

```
## Validate with validation data set: 0.157 sec elapsed
```

```
# Persist prediction results for Model#1 - Movie Effect Model

tic("Persist prediction results for Model#1 - Movie Effect Model")

movie_effect_rmse <- RMSE(predicted_ratings, validation$rating)

# Appending the results
rmse_results <-
    bind_rows(rmse_results,
          data_frame(method = "Model#2: Movie Effect",
                                 RMSE = movie_effect_rmse)
         )

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

method	RMSE
Model#1: Average Movie Rating	1.0612018
Model#2: Movie Effect	0.9439087

From the above, we have predicted movie rating based on the fact that movies are rated differently by adding the computed b_i to μ. If an individual movie is, on average, rated worse than the average rating of all movies μ , we see that it will be rated lower than μ by b_i, the difference of the individual movie average from the total average.

We can see an improvement in the next model by considering the individual user rating effect.

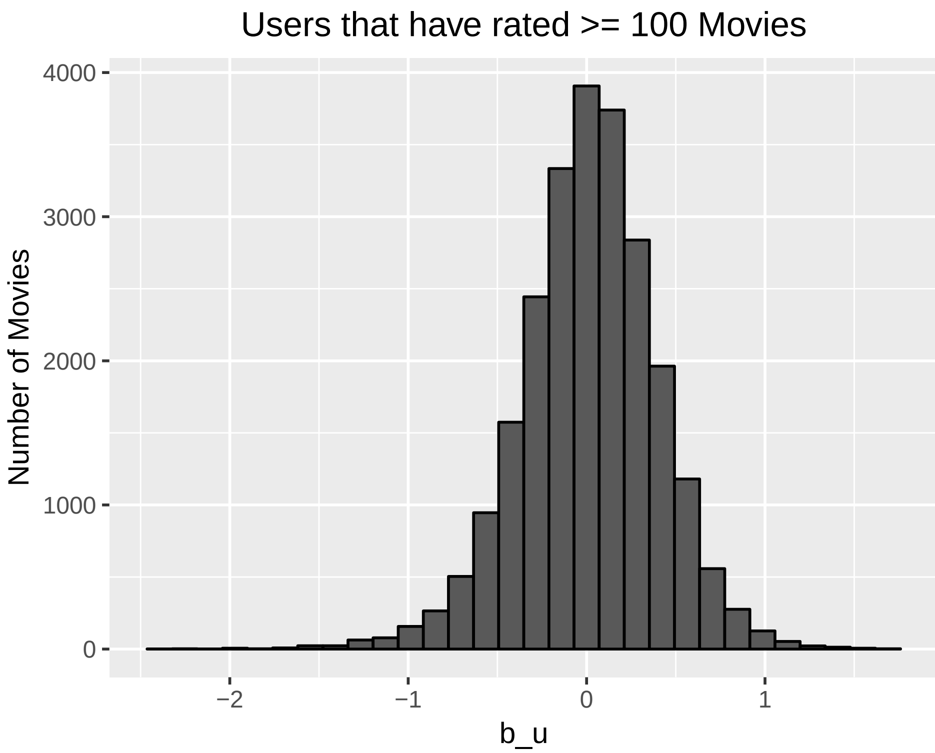
2.4.3 Model#3: Movie and User Effect

Let's compute the average rating for user μ for those that have rated over 100 movies:

```
# "Model#3: Movie and User"
# Users those have rated more than 100 movies
user_avgs <- edx %>%
    left_join(movie_avgs, by = 'movieId') %>%
    group_by(userId) %>%
    filter(n() >= 100) %>%
    summarize(b_u = mean(rating - mu - b_i))

# Plot the results

user_avgs %>%
    qplot(b_u, geom = "histogram", bins = 30, data = ., color = I("black"),
          ylab = "Number of Movies",
          main = "Users that have rated >= 100 Movies") +
    theme(plot.title = element_text(hjust = 0.5))
```



There is substantial variability across users as well: some users are very cranky and other love every movie. This implies that further improvement to our model my be:

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

where b_u is a user-specific effect. If a cranky user (negative b_u rates a great movie (positive b_i), the effects counter each other and we may be able to correctly predict that this user gave this great movie a 3 rather than a 5.

We compute an approximation by computing μ and b_i, and estimating b_u, as the average of

$$Y_{u,i} - \mu - b_i$$

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

We can now construct predictors and see how much the RMSE improves:

```
# Validate with validation data set
tic("Validate with validation data set")
predicted_ratings <- validation %>%
    left_join(movie_avgs, by = 'movieId') %>%
    left_join(user_avgs, by = 'userId') %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)
```

```r
# Persist prediction results for Model#2 - Movie and User Effect Model
movie_user_effect_rmse <- RMSE(predicted_ratings, validation$rating)

# Appending the results
rmse_results <-
    bind_rows(rmse_results,
          data_frame(method = "Model#3: Movie and User Effect",
                                     RMSE = movie_user_effect_rmse)
         )

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

method	RMSE
Model#1: Average Movie Rating	1.0612018
Model#2: Movie Effect	0.9439087
Model#3: Movie and User Effect	0.8653488

2.4.4 Model#4: Regularization: Movie and User Effect

Until now we computed standard error and constructed confidence intervals to account for different levels of uncertainty. However, when making predictions, we need one number, one prediction, not an interval. For this, we introduce the concept of regularization. Regularization permits us to penalize large estimates that are formed using small sample sizes. The general idea is to add a penalty for large values of b_i to the sum of squares equation that we minimize. So having many large b_i, make it harder to minimize. Regularization is a method used to reduce the effect of overfitting.

So estimates of b_i and b_u are caused by movies with very few ratings and in some users that only rated a very small number of movies. Hence this can strongly influence the prediction. The use of the regularization permits to penalize these aspects. We should find the value of lambda (that is a tuning parameter) that will minimize the RMSE. This shrinks the b_i and b_u in case of small number of ratings.

```r
# Using lambda tuning parameters
lambdas <- seq(0, 10, 0.25)

# Iterate for each lambda paramter and find b_i, b_u, predictions and validations

rmses <- sapply(lambdas, function(i){
  # Calculate the mean of ratings from the edx training set
    mu <- mean(edx$rating)

    # Adjust mean by movie effect and penalize low number on ratings
    # tic("Finding b_i")
    b_i <- edx %>%
        group_by(movieId) %>%
        summarize(b_i = sum(rating - mu)/(n() + i))
    # toc()

    # Ajdust mean by user and movie effect and penalize low number of ratings
    # tic("Finding b_u")
    b_u <- edx %>%
        left_join(b_i, by = "movieId") %>%
        group_by(userId) %>%
```

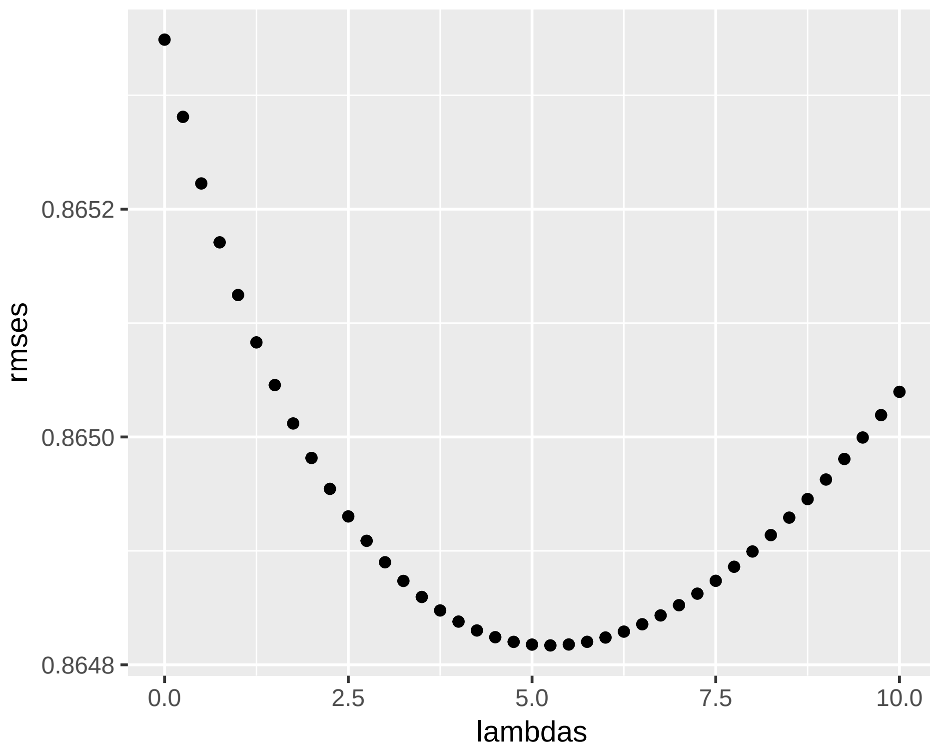
```
        summarize(b_u = sum(rating - b_i - mu)/(n() + i))
    # toc()

    # Finding Predicted_ratings
    # tic("Finding Predicted_ratings")
    predicted_ratings <- validation %>%
        left_join(b_i, by = "movieId") %>%
        left_join(b_u, by = "userId") %>%
        mutate(prediction = mu + b_i + b_u) %>%
        pull(prediction)
    # toc()

    # Return RMSE
    # tic("Return RMSE")
    return(RMSE(predicted_ratings, validation$rating))
    # toc()
})
```

Plot below shows RMSE vs lambdas to select the optimal lambda.

```
# Plot the results
qplot(lambdas, rmses)
```



Here are the optimal lambda and lowest RMSE:

```
# Which is the Optimal lambda
optimal_lambda <- lambdas[which.min(rmses)]
```

```r
# Print Optimal Lambda
optimal_lambda
```

```
## [1] 5.25
```

```r
# Print the minimum RMSE value
min(rmses)
```

```
## [1] 0.864817
```

```r
# Appending the results
rmse_results <- bind_rows(rmse_results,
    data_frame(method = "Model#4: Regularization: Movie and User Effect",
    RMSE = min(rmses)))
```

3 Final Results

Below is the list of all Model's RMSE and we can see that the Model#4 has the lowest of all.

```r
# Print the RMSE's obtained from all the Models
rmse_results %>% knitr::kable()
```

method	RMSE
Model#1: Average Movie Rating	1.0612018
Model#2: Movie Effect	0.9439087
Model#3: Movie and User Effect	0.8653488
Model#4: Regularization: Movie and User Effect	0.8648170

4 Conclusion

The regularized model including the effect of user is characterized by the lower RMSE value and is hence the optimal model to use for the present project. The optimal model characterised by the lowest RMSE value (0.8648170) lower than the initial evaluation criteria (0.8775) given by the goal for this project. We can surely improve RMSE by adding other effect such as genere, year, age of the movie etc.,

5 Envrionment Used for this Project

```r
# Show the environment used for this project
print("Envrionment Information:")
```

```
## [1] "Envrionment Information:"
```

```r
version
```

```
##                _
## platform       x86_64-apple-darwin15.6.0
## arch           x86_64
## os             darwin15.6.0
## system         x86_64, darwin15.6.0
## status
## major          3
## minor          6.0
## year           2019
## month          04
## day            26
## svn rev        76424
## language       R
## version.string R version 3.6.0 (2019-04-26)
## nickname       Planting of a Tree
```