HarvardX: PH125.9x
Data Science: Capstone - MovieLens

Sanver Gozen

May 21, 2021

Contents

1 Introduction

This is the first part of the "HarvardX: Data Science" Professional Course's Capstone Project: Movie lens.

Predictor systems are one of the most important and sought-after applications of machine learning technologies in the data businesses. For example, Netflix awarded a $1,000,000.00 dollar prize to a developer for an algorithm that increased the accuracy of Netflix's recommendation system by 10%.

In this report we present an approach to predict movie ratings with the given huge dataset, using the R Programming language. I train a linear model with our portion of the data to generate movie rating predictions, and finally calculate the Root Mean Square Error (RMSE) (RMSE) of the ratings to predict the final rating. So, we will demonstrate Machine Learning techniques for finding the smallest Root Mean Squared Error(RMSE) which means better predictions for users.

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

1.1 Libraries

First of all, selected appropriate libraries for enhanced visualization:

```
if(!require(dplyr)) install.packages("dplyr", repos = "http://cran.us.r-project.org")
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(GGally)) install.packages("GGally", repos = "http://cran.us.r-project.org")
if(!require(kableExtra)) install.packages("kableExtra", repos = "http://cran.us.r-project.org")
if(!require(Metrics)) install.packages("Metrics", repos = "http://cran.us.r-project.org")

library(dplyr) # Provides a set of tools for efficiently manipulating datasets.
library(ggplot2) # Makes it simple to create complex plots
library(tidyverse) # An opinionated collection of R packages.
library(caret) # Classification And Regression Training.
library(data.table) # For fast aggregation of large datasets.
library(GGally) # Allows to build a great scatterplot matrix.
library(kableExtra) # For better visualization of the tables.
library(Metrics) # For Machine Learning, and predictions.
```

1.2 Dataset

I use the following code to generate the datasets for the project:

```
##########################################################
# Create edx set, validation set (final hold-out test set)
##########################################################

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

dl <- tempfile()
```

```r
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")

# if using R 4.0 or later:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))

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

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

# Make sure userId and movieId in validation set are also in edx set
validation <- temp %>%
      semi_join(edx, by = "movieId") %>%
      semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

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

The MovieLens dataset is split into 2 parts which will be the **"edx"**, a training subset to train our linear model, and **"validation"** a subset will be used to test the model.

2 Analysis

2.1 Data Summary

After dividing the data set into two pieces, I needed to gain an understanding of the contents of the portions. Here I analyze the portions of the dataset, and visualize the data for better understanding.

Head of edx Subset:

```r
head(edx) %>%
  kable() %>% kable_styling(font_size = 10, position = "center",
                            latex_options = c("scale_down","HOLD_position"))
```

userId	movieId	rating	timestamp	title	genres
1	122	5	838985046	Boomerang (1992)	Comedy|Romance
1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi
1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy

Summary of the edx subset:

```
summary(edx) %>%
  kable() %>% kable_styling(font_size = 10, position = "center",
                             latex_options = c("scale_down","HOLD_position"))
```

	userId	movieId	rating	timestamp	title	genres
	Min. : 1	Min. : 1	Min. :0.500	Min. :7.897e+08	Length:9000055	Length:9000055
	1st Qu.:18124	1st Qu.: 648	1st Qu.:3.000	1st Qu.:9.468e+08	Class :character	Class :character
	Median :35738	Median : 1834	Median :4.000	Median :1.035e+09	Mode :character	Mode :character
	Mean :35870	Mean : 4122	Mean :3.512	Mean :1.033e+09	NA	NA
	3rd Qu.:53607	3rd Qu.: 3626	3rd Qu.:4.000	3rd Qu.:1.127e+09	NA	NA
	Max. :71567	Max. :65133	Max. :5.000	Max. :1.231e+09	NA	NA

The **edX** dataset is made of 6 features (columns) with a total of **9,000,055** ratings (rows).

Validation Subset:

```
head(validation) %>%
  kable() %>% kable_styling(font_size = 10, position = "center",
                             latex_options = c("scale_down","HOLD_position"))
```

userId	movieId	rating	timestamp	title	genres
1	231	5	838983392	Dumb & Dumber (1994)	Comedy
1	480	5	838983653	Jurassic Park (1993)	Action|Adventure|Sci-Fi|Thriller
1	586	5	838984068	Home Alone (1990)	Children|Comedy
2	151	3	868246450	Rob Roy (1995)	Action|Drama|Romance|War
2	858	2	868245645	Godfather, The (1972)	Crime|Drama
2	1544	3	868245920	Lost World: Jurassic Park, The (Jurassic Park 2) (1997)	Action|Adventure|Horror|Sci-Fi|Thriller

Summary of the Validation subset:

```
summary(validation) %>%
  kable() %>% kable_styling(font_size = 10, position = "center",
                             latex_options = c("scale_down","HOLD_position"))
```

	userId	movieId	rating	timestamp	title	genres
	Min. : 1	Min. : 1	Min. :0.500	Min. :7.897e+08	Length:999999	Length:999999
	1st Qu.:18096	1st Qu.: 648	1st Qu.:3.000	1st Qu.:9.467e+08	Class :character	Class :character
	Median :35768	Median : 1827	Median :4.000	Median :1.035e+09	Mode :character	Mode :character
	Mean :35870	Mean : 4108	Mean :3.512	Mean :1.033e+09	NA	NA
	3rd Qu.:53621	3rd Qu.: 3624	3rd Qu.:4.000	3rd Qu.:1.127e+09	NA	NA
	Max. :71567	Max. :65133	Max. :5.000	Max. :1.231e+09	NA	NA

The Validation subset is made of 6 features (columns) for a total of **999,999** ratings (rows). Below is the code for a check if there is invalid data:

```
anyNA(edx)
```

```
## [1] FALSE
```

There appears to be no invalid or missing data in the set.

2.2 Data Analysis

2.2.1 First Look

I gained initial understanding of the structure of the data. I now determine how many unique films, users and genres are contained in the edx data, and also calculated the average rating:

```
edx %>% summarise(
Unique_Movies = n_distinct(movieId),
Unique_Users = n_distinct(userId),
Combined_Genres = n_distinct(genres),
Average_Rating = mean(validation$rating)) %>%
  kable() %>% kable_styling(font_size = 10, position = "center",
                            latex_options = c("HOLD_position"))
```

Unique_Movies	Unique_Users	Combined_Genres	Average_Rating
10677	69878	797	3.512033

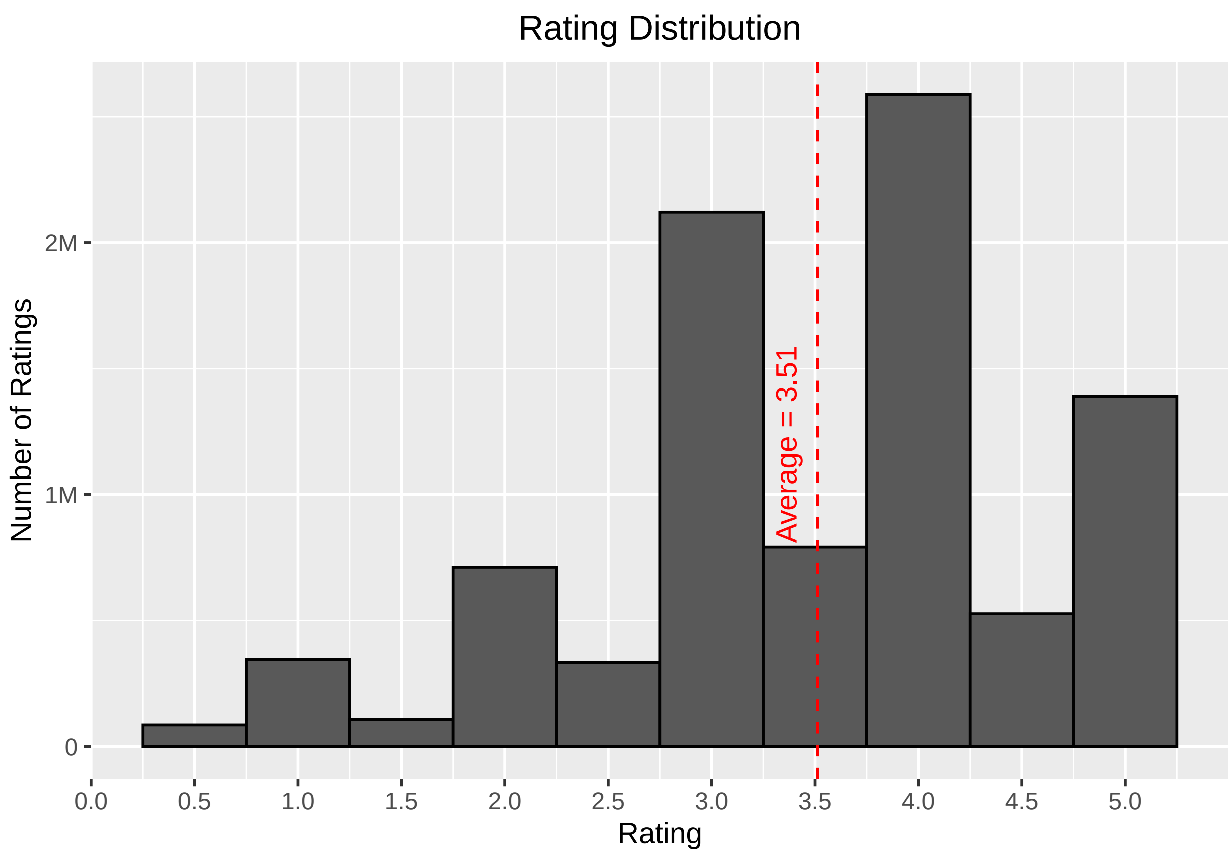
The analysis reveals almost **70,000** unique users, **800** genres and **10,700** unique movies. The movie's rating average is **3.5**.

2.2.2 Movie Effect

In this part, I attempt to search the data for details about the movies, and visualized the results. Below is an analysis of the movie rating distribution:

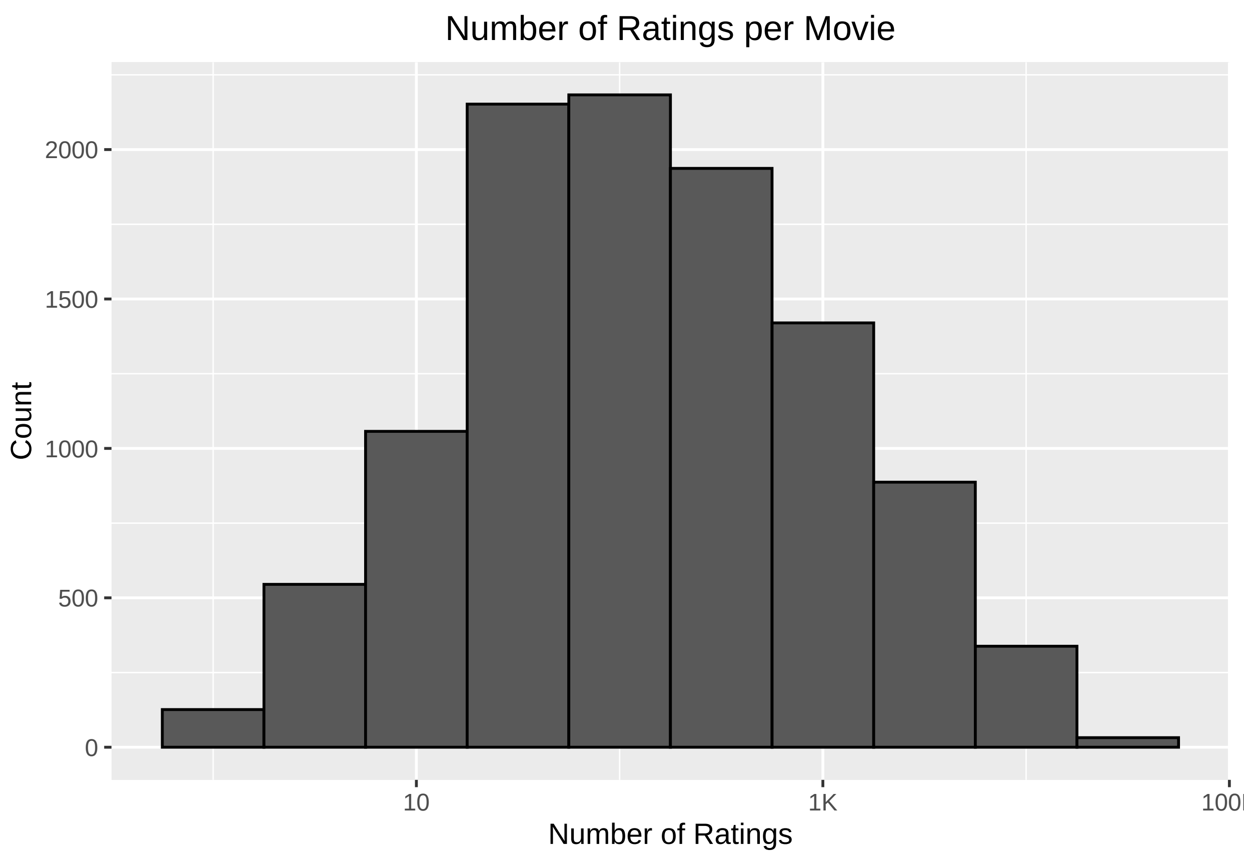
```
# Ratings distribution
edx %>%
  ggplot(aes(rating)) +
  geom_histogram(binwidth = 0.5, color = "black") +
  scale_x_continuous(breaks=seq(0, 5, by= 0.5))+
  scale_y_continuous(labels = scales::label_number_si())+
  # Draw average line
  geom_vline(xintercept = mean(edx$rating),col = "red", linetype = "dashed") +
  # Give a name to the average
  annotate("text", x = mean(edx$rating), y = 1200000, angle = 90,
           label = paste("Average =", round(mean(edx$rating),digits=2)),
           vjust = -1, colour ="red")+
  # x label
  xlab("Rating") +
  # y label
```

```r
  ylab("Number of Ratings") +
  # title
  ggtitle("Rating Distribution") +
  theme(plot.title = element_text(hjust = 0.5))
```



With the above distribution chart, I now obtain the distribution of the ratings per movie;

```r
# Number of rating per movie
edx %>% count(movieId) %>% ggplot(aes(n))+
  geom_histogram(binwidth = 0.5, color = "black")+
  xlab("Number of Ratings") +
  ylab("Count") +
  ggtitle("Number of Ratings per Movie")+
  scale_x_log10(labels = scales::label_number_si())+
  theme(plot.title = element_text(hjust = 0.5))
```



It is clearly discernible that some movies are **more popular** than others.

Now, time to create **"edx_genres"** sub data to determine unique genres:

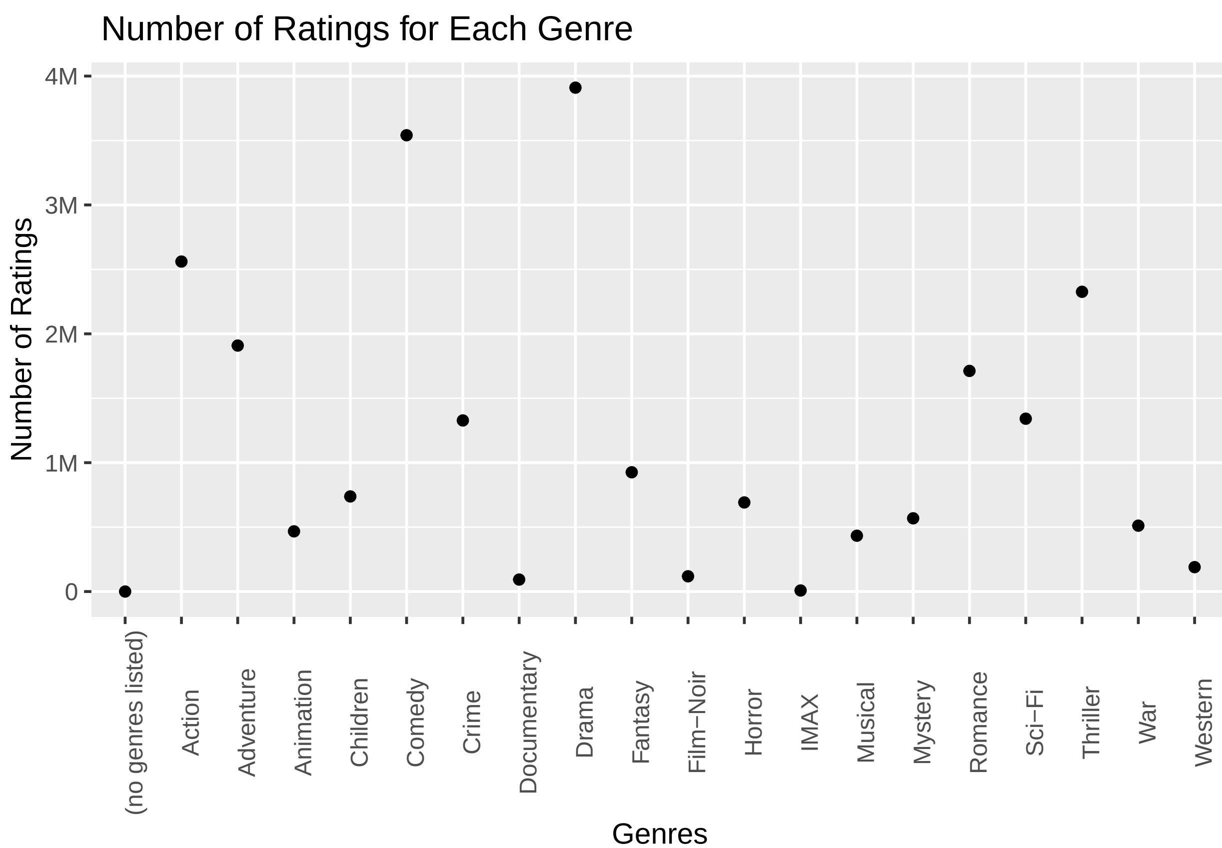
```
# Separate group of genres
edx_genres <- edx %>% separate_rows(genres, sep = "\\|")

# Unique Genres
edx_genres %>% group_by(genres) %>% summarise()
```

```
## # A tibble: 20 x 1
##    genres
##    <chr>
##  1 (no genres listed)
##  2 Action
##  3 Adventure
##  4 Animation
##  5 Children
##  6 Comedy
##  7 Crime
##  8 Documentary
##  9 Drama
## 10 Fantasy
## 11 Film-Noir
## 12 Horror
## 13 IMAX
```

```
## 14 Musical
## 15 Mystery
## 16 Romance
## 17 Sci-Fi
## 18 Thriller
## 19 War
## 20 Western
```

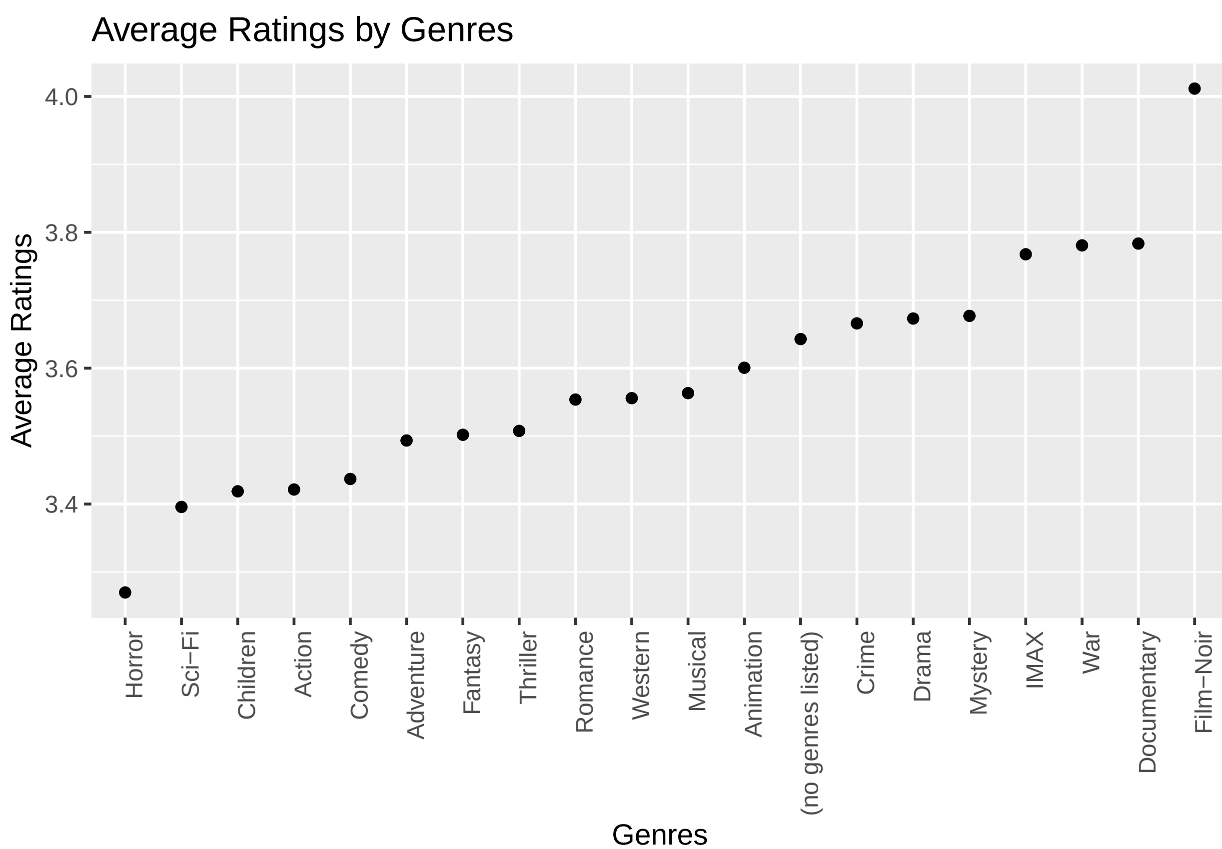
Below is the rating count of the each genre;

```
# Number of rating for each movie genres
edx_genres %>%
  group_by(genres) %>%
  summarise(count = n()) %>%
  arrange(desc(count)) %>%
  ggplot(aes(x = genres, y = count)) +
  geom_point() +
  xlab("Genres") +
  ylab("Number of Ratings") +
  scale_y_continuous(labels = scales::label_number_si())+
  labs(title = " Number of Ratings for Each Genre")+
  theme(axis.text.x  = element_text(angle= 90))
```



In the following I determine the average ratings of each genre;

```r
# Average Ratings by Genres
edx_genres %>% group_by(genres) %>%
  summarise(n = n(), avg = mean(rating)) %>%
  mutate(genres = reorder(genres, avg)) %>%
  ggplot(aes(x = genres, y = avg)) +
  geom_point() +
  xlab("Genres") +
  ylab("Average Ratings") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(title = "Average Ratings by Genres")
```



I determine the highest-rated 20 Movies, and attempt to associate the data with them;

```r
# Most Rated 20 Movies
edx %>%
  group_by(title, genres) %>%
  summarize(count=n()) %>%
  arrange(desc(count))%>% head(20)
```

```
## # A tibble: 20 x 3
## # Groups:   title [20]
##    title                                      genres                        count
##    <chr>                                      <chr>                         <int>
##  1 Pulp Fiction (1994)                        Comedy|Crime|Drama            31362
##  2 Forrest Gump (1994)                        Comedy|Drama|Romance|War      31079
```

```
##  3 Silence of the Lambs, The (1991)         Crime|Horror|Thriller         30382
##  4 Jurassic Park (1993)                     Action|Adventure|Sci-Fi|Thri~ 29360
##  5 Shawshank Redemption, The (1994)         Drama                         28015
##  6 Braveheart (1995)                        Action|Drama|War              26212
##  7 Fugitive, The (1993)                     Thriller                      25998
##  8 Terminator 2: Judgment Day (1991)        Action|Sci-Fi                 25984
##  9 Star Wars: Episode IV - A New Hope (a.k.~ Action|Adventure|Sci-Fi       25672
## 10 Apollo 13 (1995)                         Adventure|Drama               24284
## 11 Batman (1989)                            Action|Crime|Sci-Fi|Thriller  24277
## 12 Toy Story (1995)                         Adventure|Animation|Children~ 23790
## 13 Independence Day (a.k.a. ID4) (1996)     Action|Adventure|Sci-Fi|War   23449
## 14 Dances with Wolves (1990)                Adventure|Drama|Western       23367
## 15 Schindler's List (1993)                  Drama|War                     23193
## 16 True Lies (1994)                         Action|Adventure|Comedy|Roma~ 22823
## 17 Star Wars: Episode VI - Return of the Je~ Action|Adventure|Sci-Fi       22584
## 18 12 Monkeys (Twelve Monkeys) (1995)       Sci-Fi|Thriller               21891
## 19 Usual Suspects, The (1995)               Crime|Mystery|Thriller        21648
## 20 Fargo (1996)                             Comedy|Crime|Drama|Thriller   21395
```

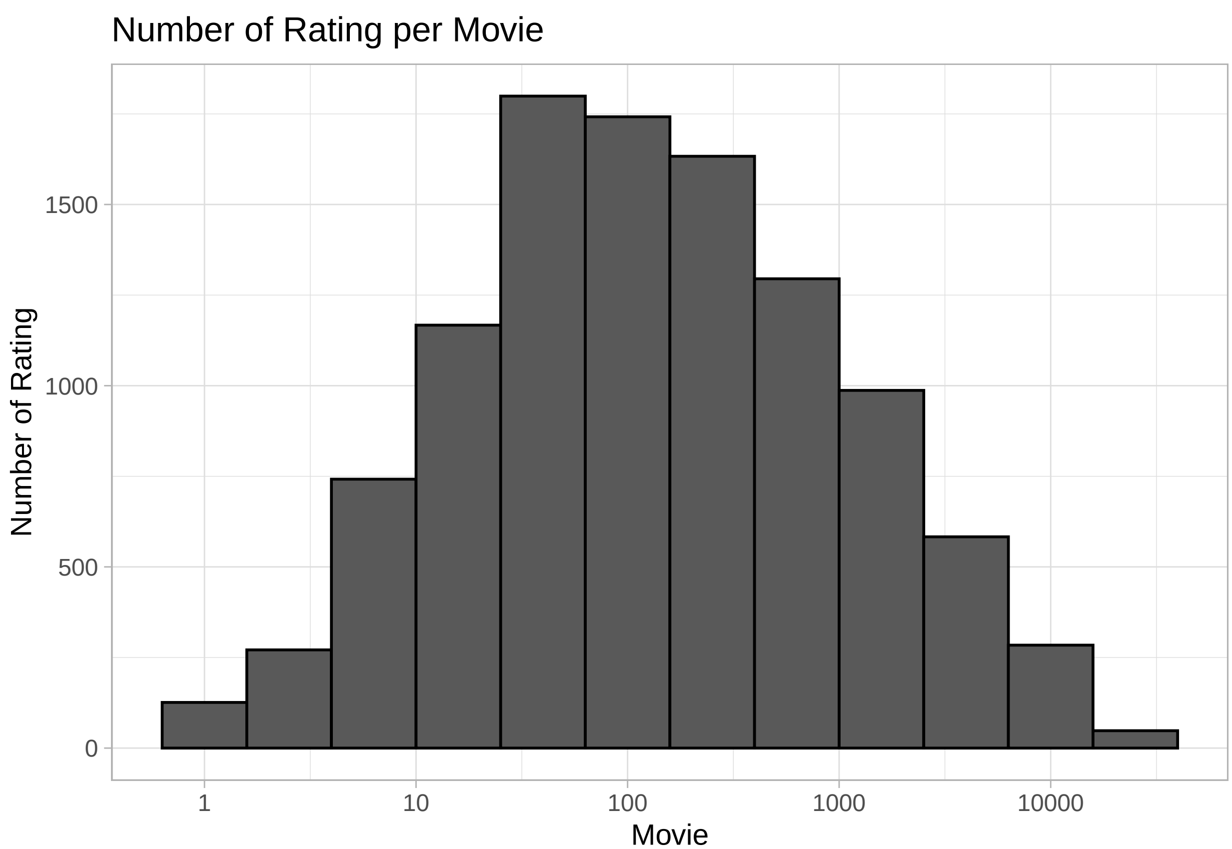
Below the highest rated genres:

```
# Most rated Genres
edx_genres %>%group_by(genres) %>%
  summarise(avg = mean(rating),count=n()) %>% arrange(desc(count))
```

```
## # A tibble: 20 x 3
##    genres               avg   count
##    <chr>              <dbl>   <int>
##  1 Drama               3.67 3910127
##  2 Comedy              3.44 3540930
##  3 Action              3.42 2560545
##  4 Thriller            3.51 2325899
##  5 Adventure           3.49 1908892
##  6 Romance             3.55 1712100
##  7 Sci-Fi              3.40 1341183
##  8 Crime               3.67 1327715
##  9 Fantasy             3.50  925637
## 10 Children            3.42  737994
## 11 Horror              3.27  691485
## 12 Mystery             3.68  568332
## 13 War                 3.78  511147
## 14 Animation           3.60  467168
## 15 Musical             3.56  433080
## 16 Western             3.56  189394
## 17 Film-Noir           4.01  118541
## 18 Documentary         3.78   93066
## 19 IMAX                3.77    8181
## 20 (no genres listed)  3.64       7
```

Finally the number of rating per movie is determined:

```r
# Number of Rating per Movie
edx %>% count(movieId) %>% ggplot(aes(n))+
  geom_histogram(binwidth = 0.4, color = "black")+
  scale_x_log10()+
  xlab("Movie") +
  ylab("Number of Rating") +
  ggtitle("Number of Rating per Movie") +
  theme_light()
```



The above analysis show that there are some genres and movies rated higher than others, indicating (the expected) differences in popularity. **Drama**, **Comedy**, **Action**, **Thriller** and **Adventure** genres mostly belong to top movies in the **Top20** list. This is the so-called **"Movie Effect"**. I use this later for the calculation of the intended predictions.

2.2.3 User Effect

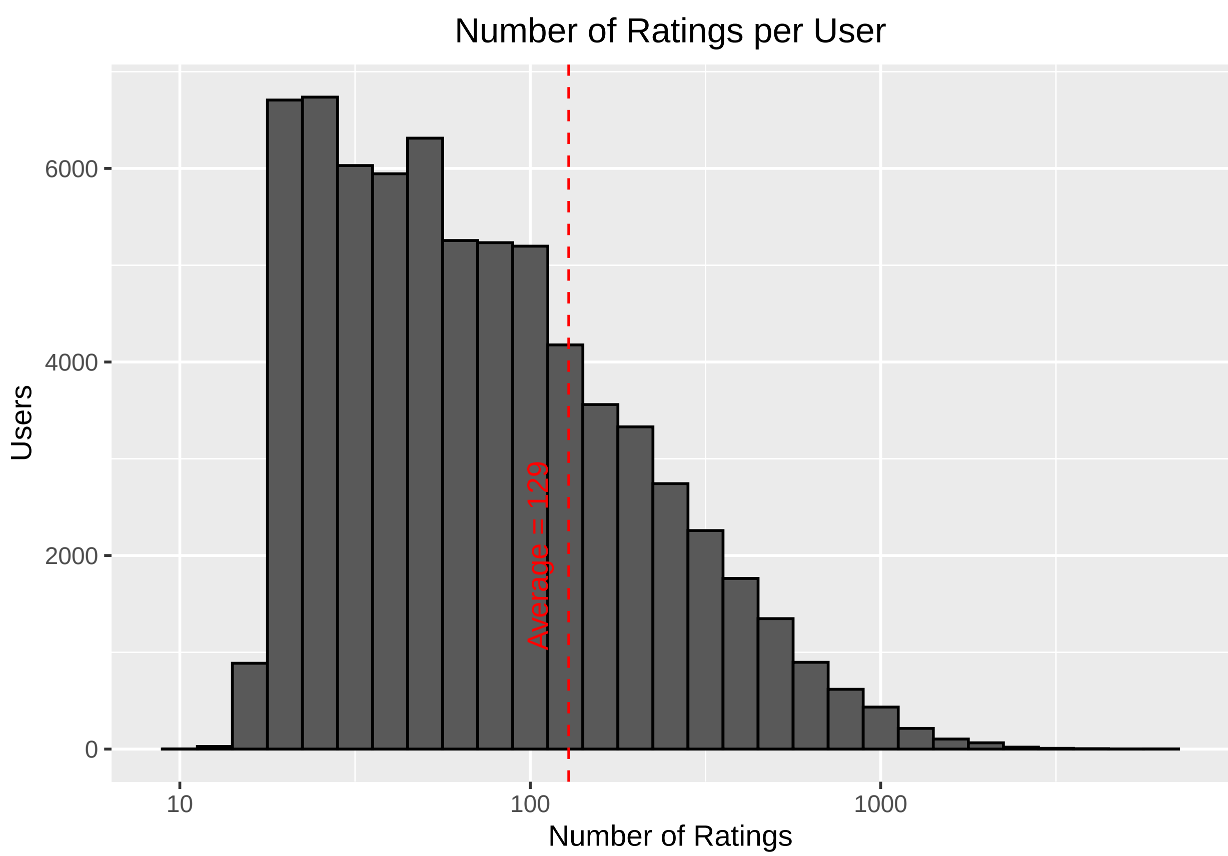
In this part, I try to search and visualized the data to obtain information about the users.

The number of the ratings per user and the rounded average number of ratings;

```r
# Mean Number of Rates
mean_number_of_rates <- edx %>% count(userId) %>% summarize(Average = mean(n))
mean_number_of_rates
```

```
##    Average
## 1 128.7967
```

```r
# Number of rating per user
edx %>% count(userId) %>% ggplot(aes(n))+
  geom_histogram(binwidth = 0.1, color = "black")+
  # Draw average line
  geom_vline(xintercept = mean_number_of_rates$Average,col = "red",
             linetype = "dashed") +
  # Give a name to the average
  annotate("text", x = mean_number_of_rates$Average, y = 2000, angle = 90,
           label = paste("Average =",round(mean_number_of_rates$Average)),
           vjust = -1, colour ="red")+
  xlab("Number of Ratings") +
  ylab("Users") +
  ggtitle("Number of Ratings per User")+
  scale_x_log10()+
  theme(plot.title = element_text(hjust = 0.5))
```

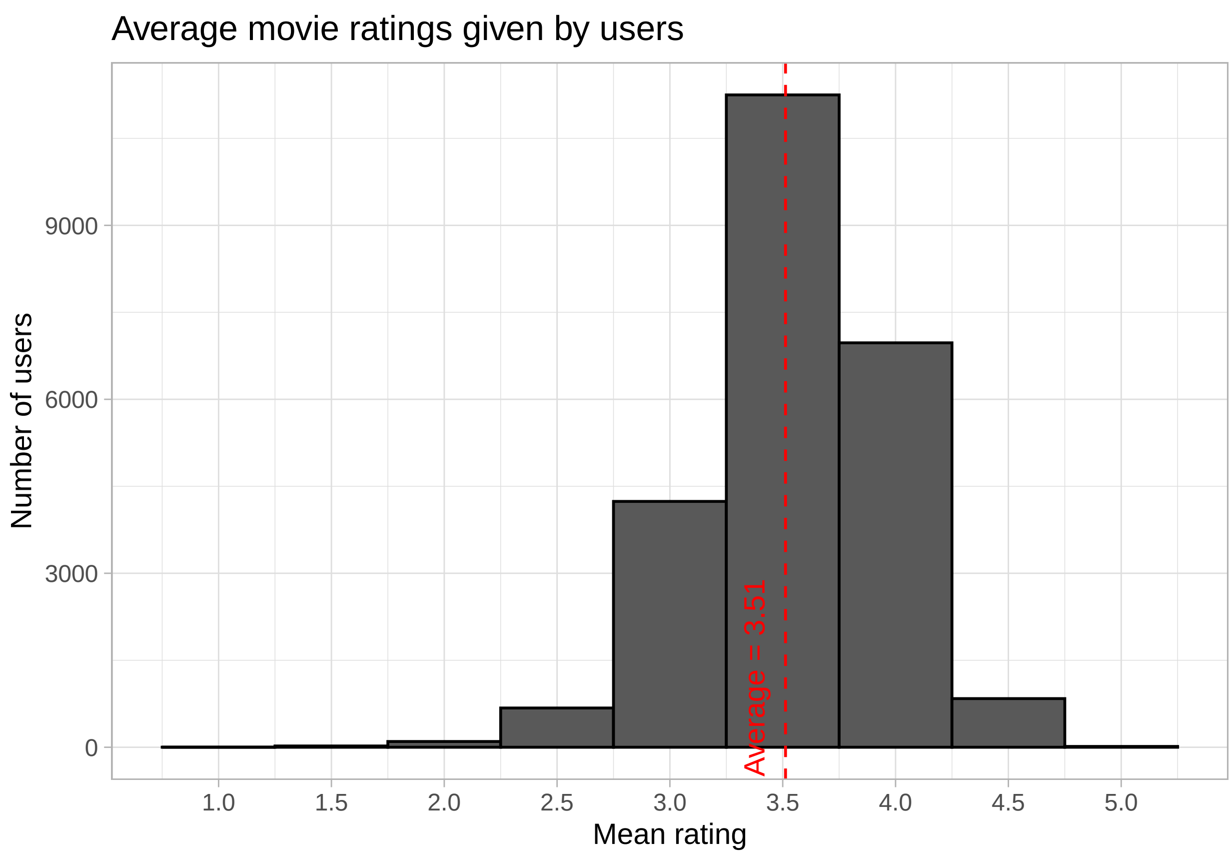


The above chart shows clearly that some USERS are more active than others. Some of the users rated more than **1000** movies, while the average activity of the users is characterized by **129** ratings.

I obtain the average rating count given by users, filter the users who provided at least **100** rating;

```r
# Mean movie rating per User
edx %>%
  group_by(userId) %>%
  filter(n() >= 100) %>%
  summarize(average = mean(rating)) %>%
```

```
  ggplot(aes(average)) +
  geom_histogram(binwidth = 0.5, color = "black") +
  scale_x_continuous(breaks=seq(0, 5, by= 0.5))+
  # Draw average line
  geom_vline(xintercept = mean(edx$rating),col = "red", linetype = "dashed") +
  # Give a name to the average
  annotate("text", x = mean(edx$rating), y = 1200, angle = 90,
           label = paste("Average =", round(mean(edx$rating),digits=2)),
           vjust = -1, colour ="red")+
  xlab("Mean rating") +
  ylab("Number of users") +
  ggtitle("Average movie ratings given by users") +
  theme_light()
```



Below I search the data for users who submitted a rating of **1** or lower for a total of more than **100** movies:

```
# Unique users rated 1 or below to at least 100 movie
edx %>%
  group_by(userId) %>%
  filter(n() >= 100) %>% filter(rating <= 1) %>%
  summarize() %>% nrow
```

```
## [1] 21040
```

It is apparent from the above chart that users different widely from each other, some are give predominantly low, others high r. Most of the user's average is **3.5** rating, but around **21,040** of them rated **1** to at least

100 movies. I assume therefore that some users will affect the rating algorithms greater than others, due to their rate characteristics. I term this the **User Effect**, and use it later when calculating the prediction.

2.3 Modeling

The Root Mean Square Error (RMSE) is the square root of the mean of the square of all of the error contributions. It tells how concentrated the data is around the line of best fit. The use of RMSE is very common, and it is considered an excellent general-purpose error metric for numerical predictions.

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

where y are the observations, $\hat{y}$ predicted values of a variable, and N the number of observations available for analysis. RMSE is a good measure of accuracy, but only to compare prediction errors of different models or model configurations for a particular variable and not between variables, as it is scale-dependent.

I use the actual ratings for prediction.

2.3.1 Train and Test Sets

In machine Learning algorithms, we can train our model with a portion of our data, and test our algorithm to confirm the model with the remaining portion of the data. I thus can tell whether the model is a good predictor or not.

First of all, I need to split **edx** data into 2 different subsets by 20% for the **test** subset and 80% for the **training** subset. With the **train** subset, I train the linear model, followed by testing it with the **test** subset to judge the prediction quality. For the RMSE prediction, the a lower RMSE indicates higher prediction quality.

```
# Split edx data to train and test data by portion of %20-%80
set.seed(2105, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2,
                                  list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]
```

I now have obtained **test** and **train** subsets.

2.3.2 Basic Model: Average Movie Rating

The simplest linear model is using average movie ratings for all movies. We assume that all errors of our predictions are random. There will not be anything except movie rating average.

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

Where $Y_{u,i}$ is the prediction and μ is the average rating for all movies. $\epsilon_{u,i}$ will be zero due to the error difference.

```
# Set an option for decimals
options(digits = 10)
# Use train subset for train our model
mu <- mean(train_set$rating)
# Use test subset to test our model
basic_model_rmse <- rmse(test_set$rating,mu)
```

Below is the first model result obtained. Table shows an overview over the model results, for easy comparison.

```
# Write the result shown in table
all_rmse <- data.frame(Linear_Model = "Basic Model", RMSE = basic_model_rmse)
all_rmse %>%
  kable() %>% kable_styling(font_size = 12, position = "center",
                            latex_options = c("HOLD_position"))
```

Linear_Model	RMSE
Basic Model	1.060135826

2.3.3 Movie Effect Model

As mentioned in 2.2.2 section, some movies are more active than the others. We know that this will cause some error in our model. So, in this model we will calculate a bias term (estimate deviation) for the difference between mean each movie rating and all average movie ratings.

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

Where μ is the average movie rating, $\mu + b_i$ is our movie effect for each (i) movie and $Y_{u,i}$ will be our predictions.

```
# Calculate the difference between each movie and all movie ratings.
movie_average <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
# test our model
predicted_ratings <- mu + test_set %>%
  left_join(movie_average, by='movieId') %>%
  pull(b_i)
movie_effect_model <- RMSE(predicted_ratings, test_set$rating, na.rm = TRUE)
# Write the result shown in table
all_rmse <- bind_rows(all_rmse, data.frame(Linear_Model = "Movie Effect Model",
                       RMSE = movie_effect_model))
all_rmse %>%
  kable() %>% kable_styling(font_size = 12, position = "center",
                            latex_options = c("HOLD_position"))
```

Linear_Model	RMSE
Basic Model	1.0601358259
Movie Effect Model	0.9432725123

2.3.4 User Effect Model

Similar to the above treated case, an error needs to be taken into consideration for the user effect mentioned in section 2.3.4. Some users are more popular than the others. In the model below, I calculate the user effect b_u.

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

In the following I investigate how it effects the result:

```
# Calculate the difference between each user average and all users average.
user_average <- train_set %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu))
# test our model
predicted_ratings <- mu + test_set %>%
  left_join(user_average, by='userId') %>%
  pull(b_u)
user_effect_model <- RMSE(predicted_ratings, test_set$rating, na.rm = TRUE)
# Write the result shown in table
all_rmse <- bind_rows(all_rmse, data.frame(Linear_Model = "User Effect Model",
                                           RMSE = user_effect_model))
all_rmse %>%
  kable() %>% kable_styling(font_size = 12, position = "center",
                            latex_options = c("HOLD_position"))
```

Linear_Model	RMSE
Basic Model	1.0601358259
Movie Effect Model	0.9432725123
User Effect Model	0.9786981866

2.3.5 Movie and User Effect Model

I have now determined the difference between the user effect and the movie effect, which is combined as follows.

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

```
# I calculate the model by combining user and movie effect
user_average <- train_set %>%
  left_join(movie_average, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
# test our model
predicted_ratings <- test_set %>%
  left_join(movie_average, by='movieId') %>%
  left_join(user_average, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
movie_user_model <- RMSE(predicted_ratings, test_set$rating, na.rm = TRUE)
```

```r
# Write the result shown in table
all_rmse <- bind_rows(all_rmse,
                      data.frame(Linear_Model = "Movie and User Effect Model",
                                 RMSE = movie_user_model))
all_rmse %>%
  kable() %>% kable_styling(font_size = 12, position = "center",
                           latex_options = c("HOLD_position"))
```

Linear_Model	RMSE
Basic Model	1.0601358259
Movie Effect Model	0.9432725123
User Effect Model	0.9786981866
Movie and User Effect Model	0.8659151103

2.3.6 Regularized Movie and User Effect Model

Regularization is the process of adding information in order to solve an ill-posed problem or to prevent overfitting. Overfitting occurs when a model "learns" the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the mode.

Therefore, movies and users with few ratings can affect the standard deviation of our model. Large errors can increase our RMSE which we do not want. This needs to be taken into account by tuning the model. First we need to determine our tuning parameter which we call lambda (λ). For that purpose, the RMSE is repeatedly calculated and lambda (λ) is determined. Finding the smallest RMSE leads to an optimal lambda (λ), which is evaluated.

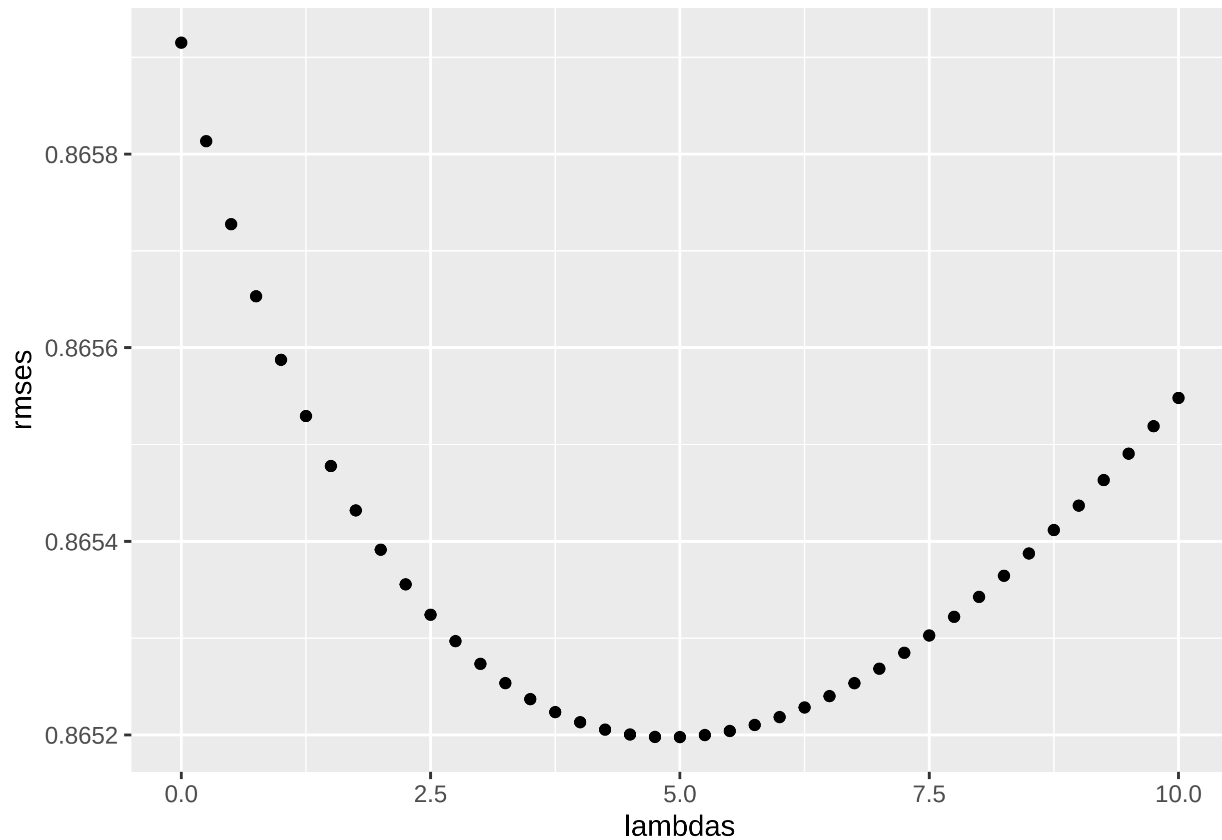
$$Y_{u,i} = \mu + b_{i,n,\lambda} + b_{u,n,\lambda} + \epsilon_{u,i}$$

```r
# Determine best lambda from a sequence
lambdas <- seq(0, 10, 0.25)
# Calculate best lambda
rmses <- sapply(lambdas, function(l){

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

  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
# Run our previous model with lambda
  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)
```

```r
  return(RMSE(predicted_ratings, test_set$rating, na.rm = TRUE))
})
```

We will see what we have calculated:

```r
# See all lambdas by RMSEs
qplot(lambdas, rmses)
```




```r
# Select the best lambda
min_lambda <- lambdas[which.min(rmses)]
min_lambda
```

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

As we see above, our best lambda is around 5.

Now time to select best RMSE:

```r
# Select the best RMSE
rmse_regularization <- min(rmses)
# Write the result shown in table
all_rmse <- bind_rows(all_rmse,
           data.frame(Linear_Model = "Regularized Movie and User Effect Model",
                                 RMSE = rmse_regularization))
```

```
all_rmse %>%
  kable() %>% kable_styling(font_size = 12, position = "center",
                            latex_options = c("HOLD_position"))
```

Linear_Model	RMSE
Basic Model	1.0601358259
Movie Effect Model	0.9432725123
User Effect Model	0.9786981866
Movie and User Effect Model	0.8659151103
Regularized Movie and User Effect Model	0.8651977670

2.3.7 Validation of the Model

Earlier, as shown in section introduction, a code section was designed to split the data into **edx** subset and **validitaion** subset. With it the best model which gives us the **smallest RMSE**. In the following, the regularized RMSE is obtained with our **validation** subset.

```
# Calculate regularized movie effect
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+min_lambda))
# Calculate regularized user effect
b_u <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+min_lambda))
# Calculate the predictions on validation set based on these above terms
predicted_ratings <- validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
# output RMSE of our final model
validation_model <- RMSE(predicted_ratings, validation$rating)
# Write the result shown in table xy
all_rmse <- bind_rows(all_rmse,
                      data.frame(Linear_Model = "Validation",
                                 RMSE = validation_model))
all_rmse %>%
  kable() %>% kable_styling(font_size = 12, position = "center",
                            latex_options = c("HOLD_position"))
```

Linear_Model	RMSE
Basic Model	1.0601358259
Movie Effect Model	0.9432725123
User Effect Model	0.9786981866
Movie and User Effect Model	0.8659151103
Regularized Movie and User Effect Model	0.8651977670
Validation	0.8648177556

3 Results

A large data set was selected, searched and visualized, and training and prediction sets were defined. Several models were tested. User and movie effect errors considered, and associated these with the respective models. A machine learning model was obtained which generated the smallest **RMSE** for the given dataset, satisfying the provided specification:

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

3.1 Discussion

It could be beneficial to consider a time effect on the ratings for the last model. Although I have achieve the target specification without this provision, adding time effects as a bias would potentially improve our last model.

4 Conclusion

The target of **the Data Science: Capstone - MovieLens project** was to find an algorithm gives you lower than **0.87750** RMSE, which was achieved by our last model above and the result is **0.8651977670**.

This shows us, the Linear Regression model with regularized effects on users and movie is give you an opportunity to make a good recommender systems to predict ratings.