MovieLens Report

Donal Medina

August 18, 2021

Introduction

For this Project we will create a movie recommendation system using the MovieLens dataset (10M version). The first step will be download and generate the dataset, split in two subsets, an **edx** set for training and test and a **validation** set for evaluate the final algorithm, with a relation 90% y 10% of DataLens dataset respectively.
The second step will be explore the dataset, develop and train a machine learning algorithm using the inputs in one subset of edx set to predict movie ratings. We will train different models and will evaluate the performance in the test set.
The final model will be compared to the true ratings in the validation set using RMSE(residual mean squared error) to evaluate how close our predictions are.

Analysis

Install and load Packages

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

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

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.0     v dplyr   1.0.5
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_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
```

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

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

```
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
```

```
library(tidyverse)
library(caret)
library(data.table)
```

Data Loading

Create edx set, validation set (final hold-out test set).
MovieLens 10M dataset can found: https://grouplens.org/datasets/movielens/10m/
The Dataset can be download: http://files.grouplens.org/datasets/movielens/ml-10m.zip

```
# Note: this process could take a couple of minutes
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")

# 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")
```

Now we split the MovieLens dataset into Training (edx) and Validation (validation) sets. The Validation set will be 10% of MovieLens data.
We need to make sure userId and movieId in validation set are also in edx set.

```
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
```

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

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

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)
```

Data Exploration

First we check for any NA value.

```
anyNA(edx)
```

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

General overview of dataset:

```
str(edx)
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Ad
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
head(edx) %>% knitr::kable()
```

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

After loading the data set we start by looking at the data structure and type we can see that there is 9000055 observations and six variables: userId,movieId,rating,timestamp,title,genres.

```
summary(edx) %>% knitr::kable()
```

	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

userId	movieId	rating	timestamp	title	genres
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

We can see the average of rating is 3.512, 71567 user rating and 65133 movie rating.
Now look for the distinct number of user and movies:

```
edx %>% summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   69878    10677
```

We can see the edx dataset has 10677 distinct movies and 69878 distinct users.

Top 5 movies with most ratings:

```
edx %>% group_by(title) %>%
  summarize(count = n()) %>%
  top_n(5) %>%
  arrange(desc(count)) %>%
  knitr::kable()
```

```
## Selecting by count
```

title	count
Pulp Fiction (1994)	31362
Forrest Gump (1994)	31079
Silence of the Lambs, The (1991)	30382
Jurassic Park (1993)	29360
Shawshank Redemption, The (1994)	28015

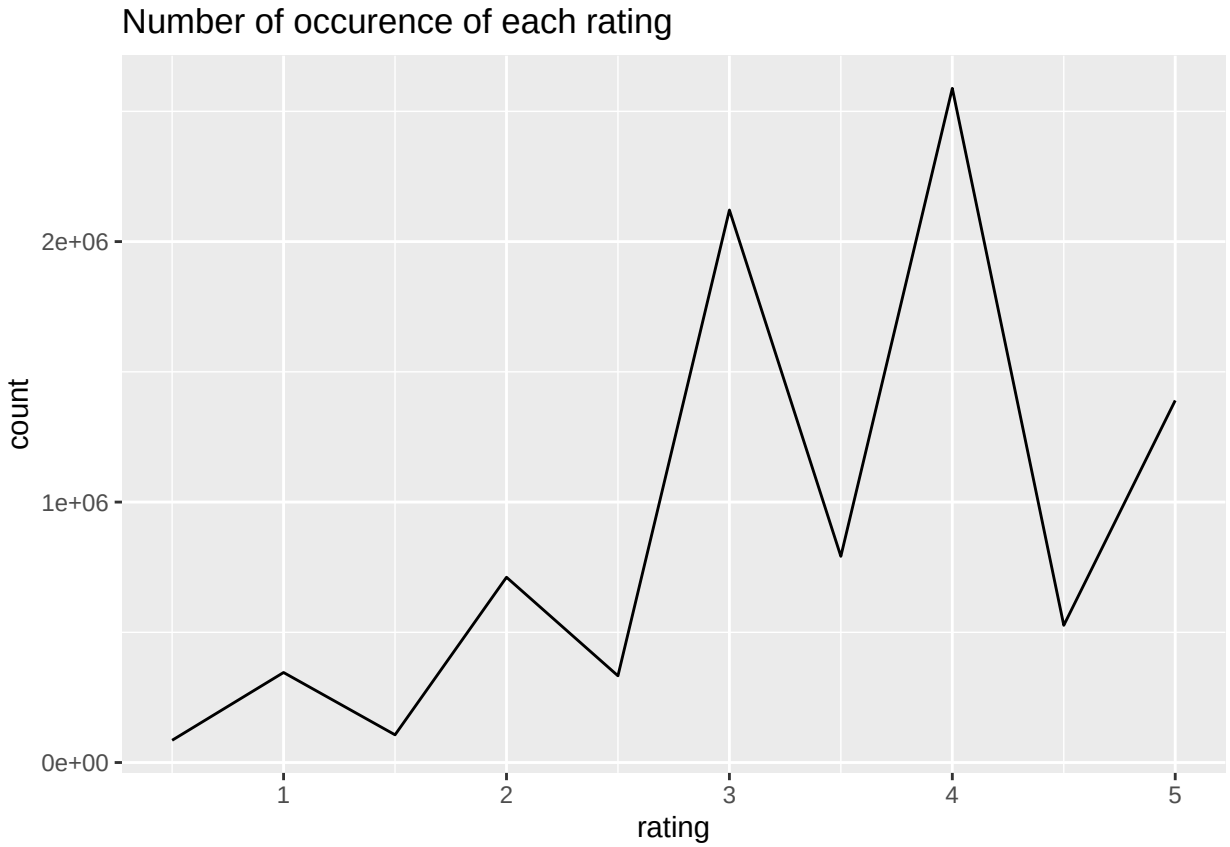
Let's look for the most ratings.

```
edx %>% group_by(rating) %>%
  summarize(count = n()) %>%
  top_n(5) %>%
  arrange(desc(count))  %>%
  knitr::kable()
```

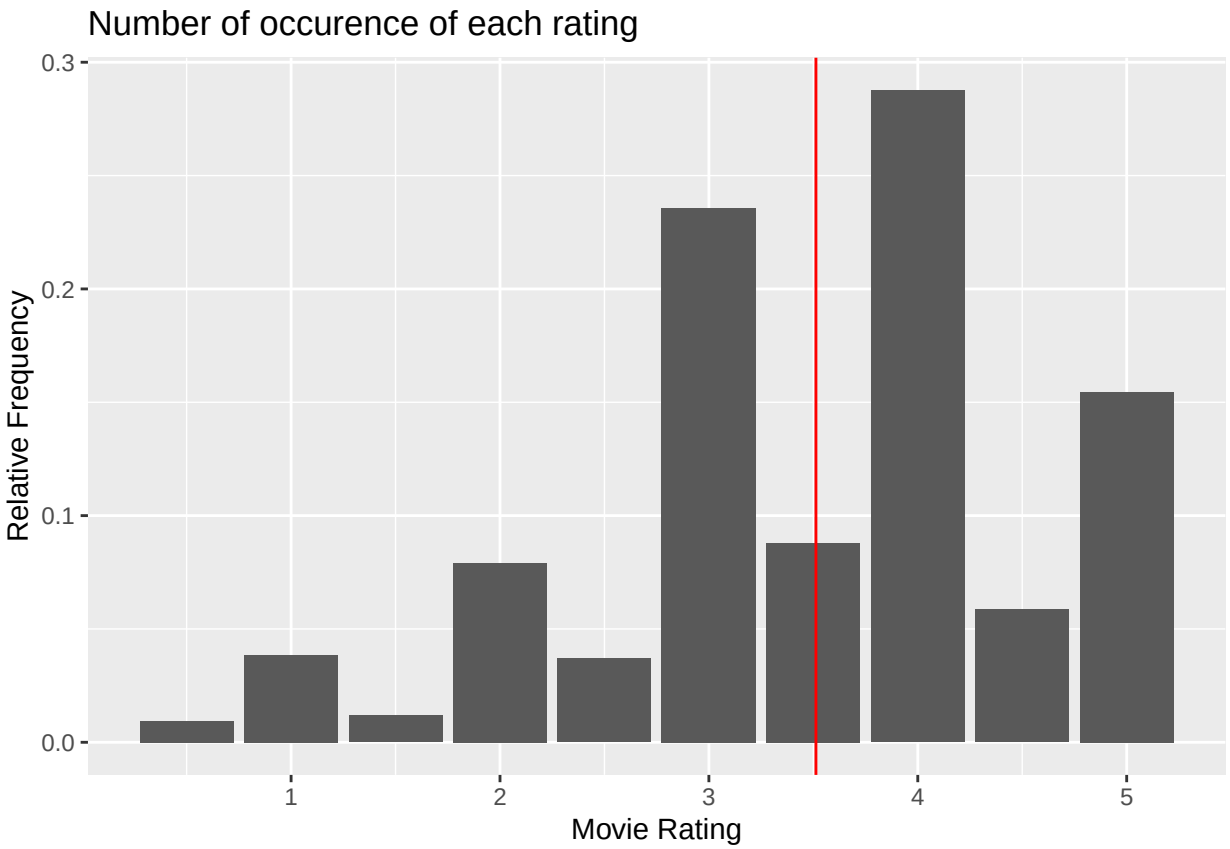
```
## Selecting by count
```

rating	count
4.0	2588430
3.0	2121240
5.0	1390114
3.5	791624
2.0	711422

Number of occurence of each rating:

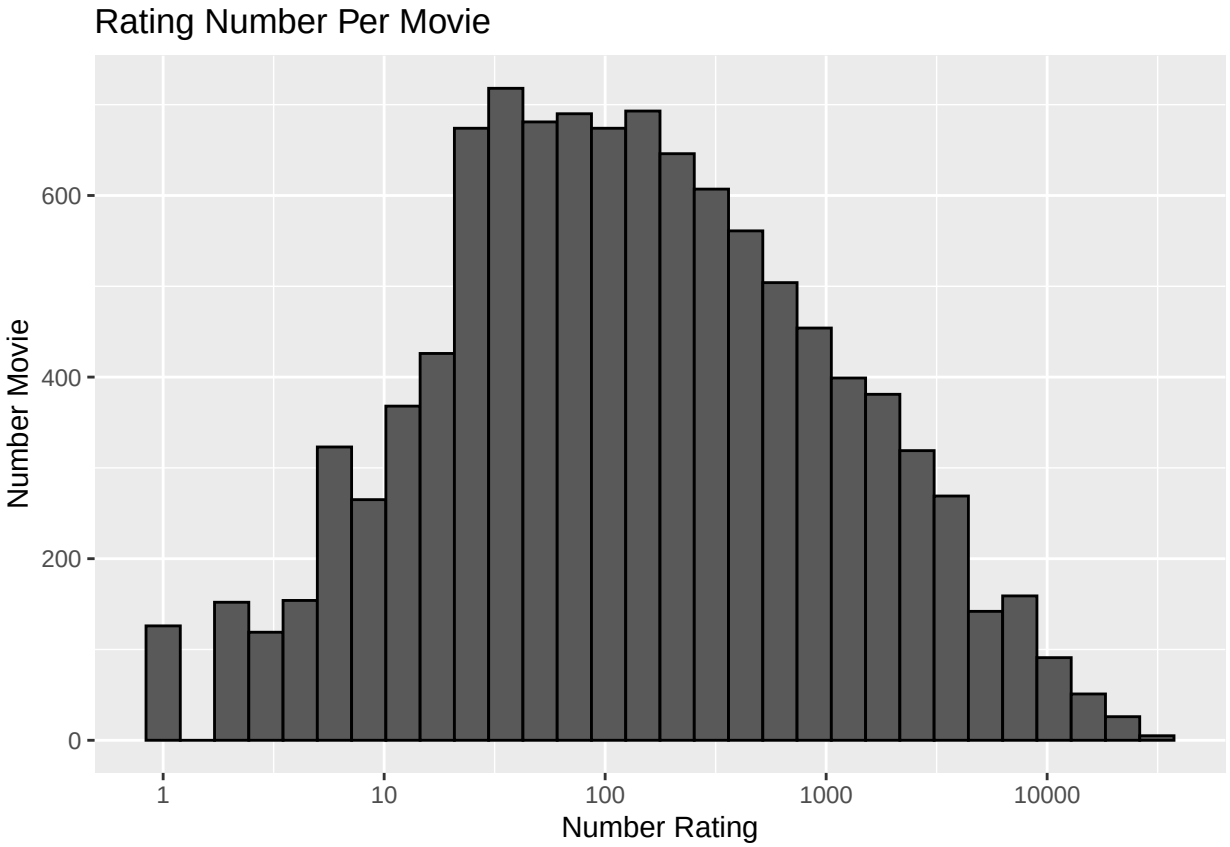


Rating distribution:



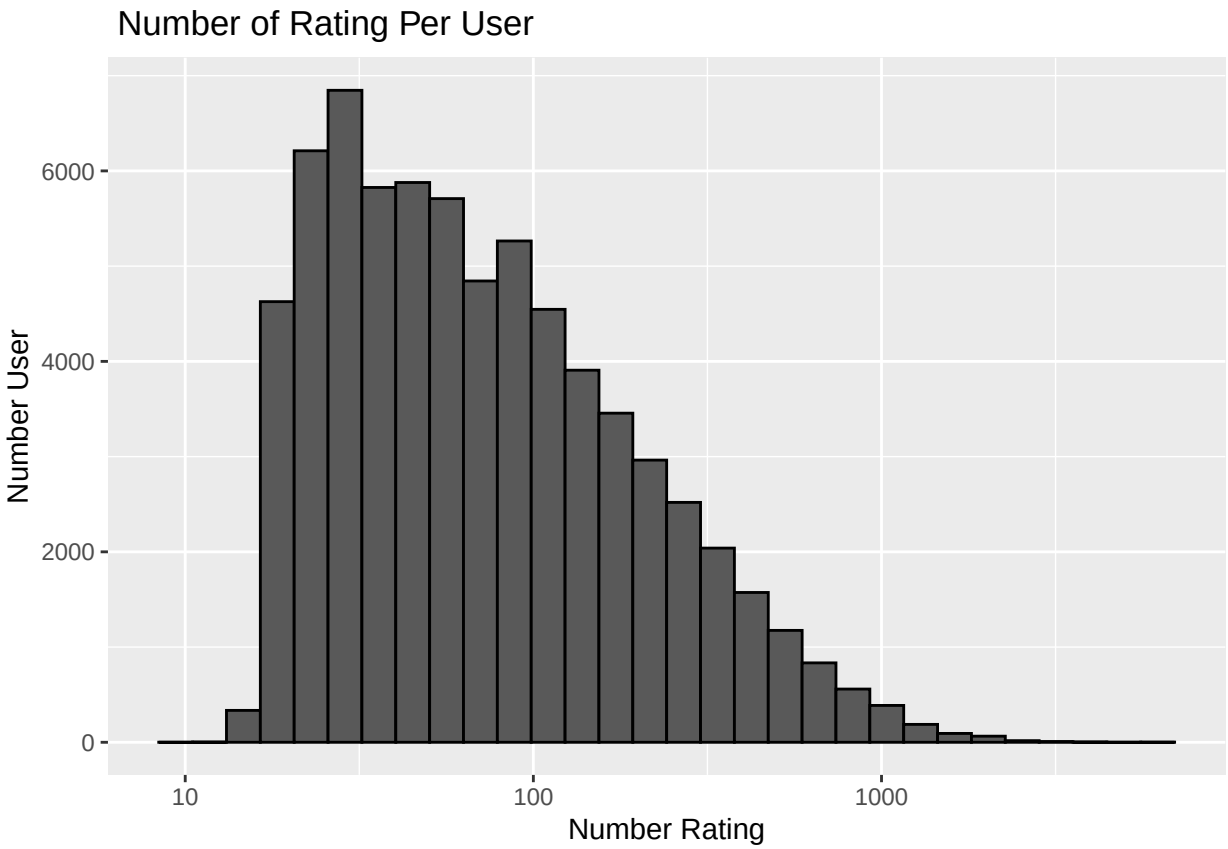
We can see that the most common rating is 4, and the least common is 0. There is a pattern where it is more common for users to give positive ratings. There is also a pattern where half ratings are less common compared to whole ratings.
A red line of the overall average rating is also plotted here as a reference.

Number of rating per movies:



Most movies have below 1000 reviews, while some have more than 10000.

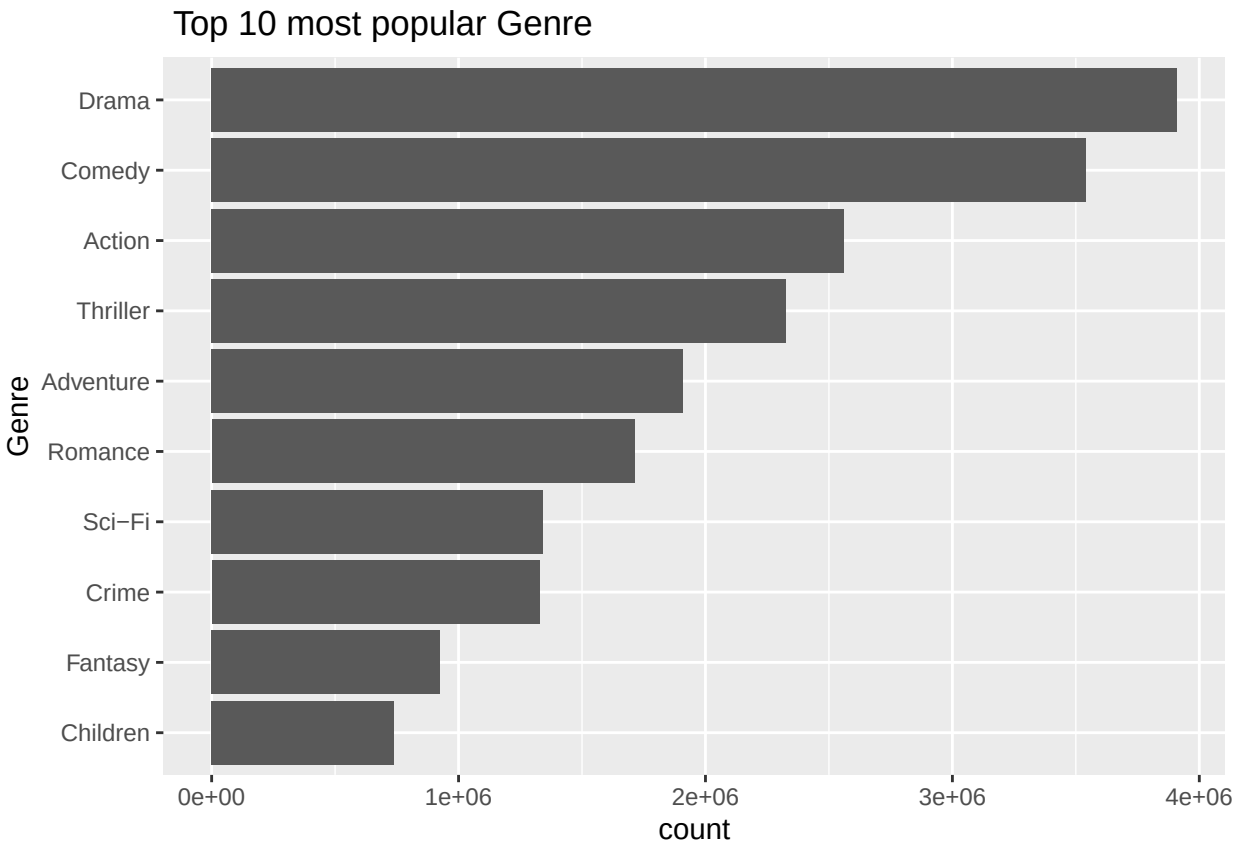
Number of rating per user:



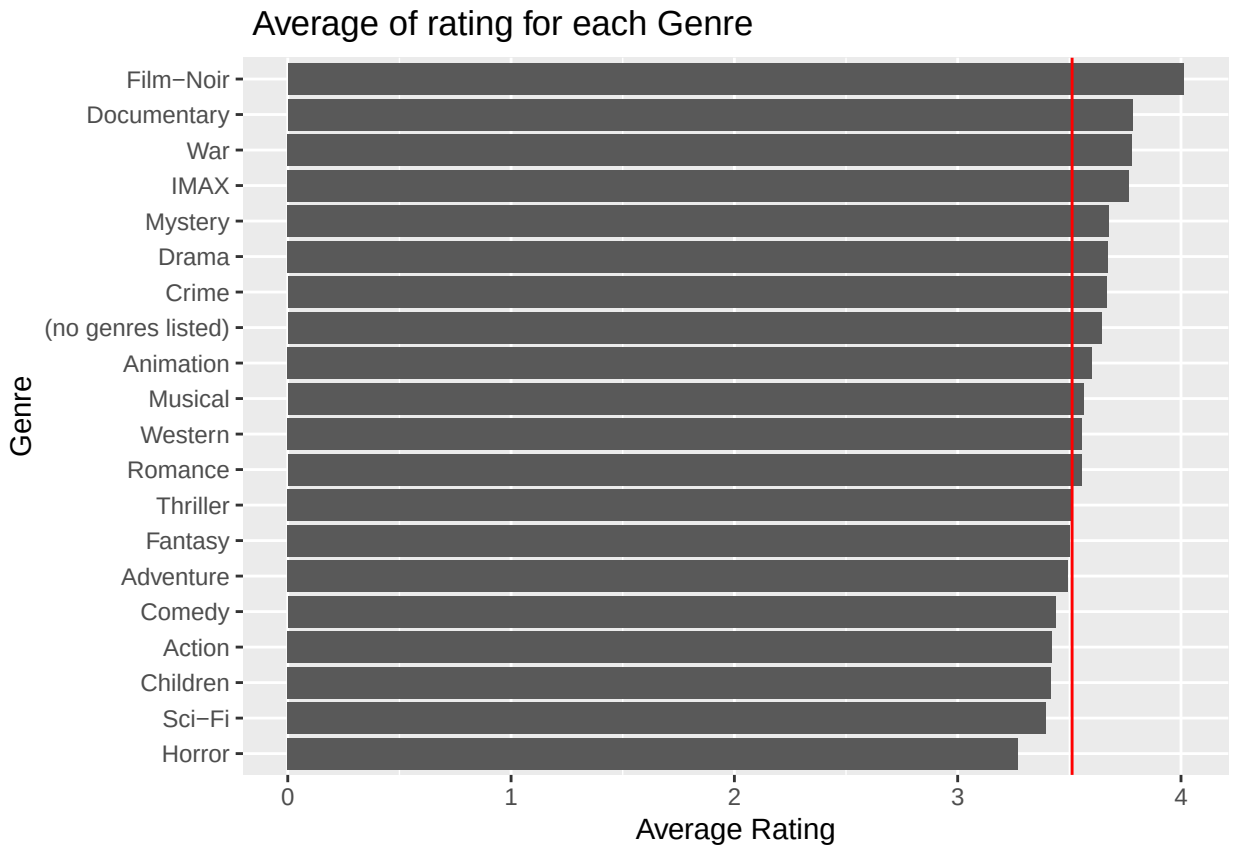
Most users wrote less than 100 reviews, while some wrote more than 1000.

Now we look for genre and rating relationship, the top 10 most popular genre.

```
## Selecting by count
```



Then we plot the average of rating for each movie genres and the overall average rating with a red line as reference.



Now we find that some genres tend to have higher ratings than the average and some tend to have lower ratings. However, overall, the genres effect seems to be rather minor.

Data Wrangling

Set seed to 1.

```
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
```

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

Split the edx data set to train set(80%) and test set(20%).

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

To make sure we do not include users and movies in the test set that do not appear in the training set, we remove these entries using the semi_join function.

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

Modeling approach

Based on the eploration of the data, we will build a baseline prediction model, we will consider the movie effect, user effect and regulirazation of user and movie effect, we will evaluate these models and choose the

best to go with.

The movie and user effect model describes the rating Y_u_i = mu + b_i + b_u + error, while mu is the average of all ratings, b_i and b_u are item bias and user bias, respectively. To avoid over-training caused by estimates from small sample sizes, we will use regularization to add penalties to shrink the estimates from small samples sizes, using 10-fold cross validation.

Results

RMSE (residual mean squared error) calculation funtion

```
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2, na.rm = TRUE))
}
```

First model: Just the average

We will use average ratings for all movies regardless of user.
We know that the estimate that minimizes the RMSE is the least squares estimate of mu and, in this case, is the average of all ratings:

```
mu <- mean(train_set$rating)
mu
```

```
## [1] 3.512482
```

If we predict all unknown ratings with mu_hat we obtain the following RMSE:

```
naive_rmse <- RMSE( mu,test_set$rating)
naive_rmse
```

```
## [1] 1.059904
```

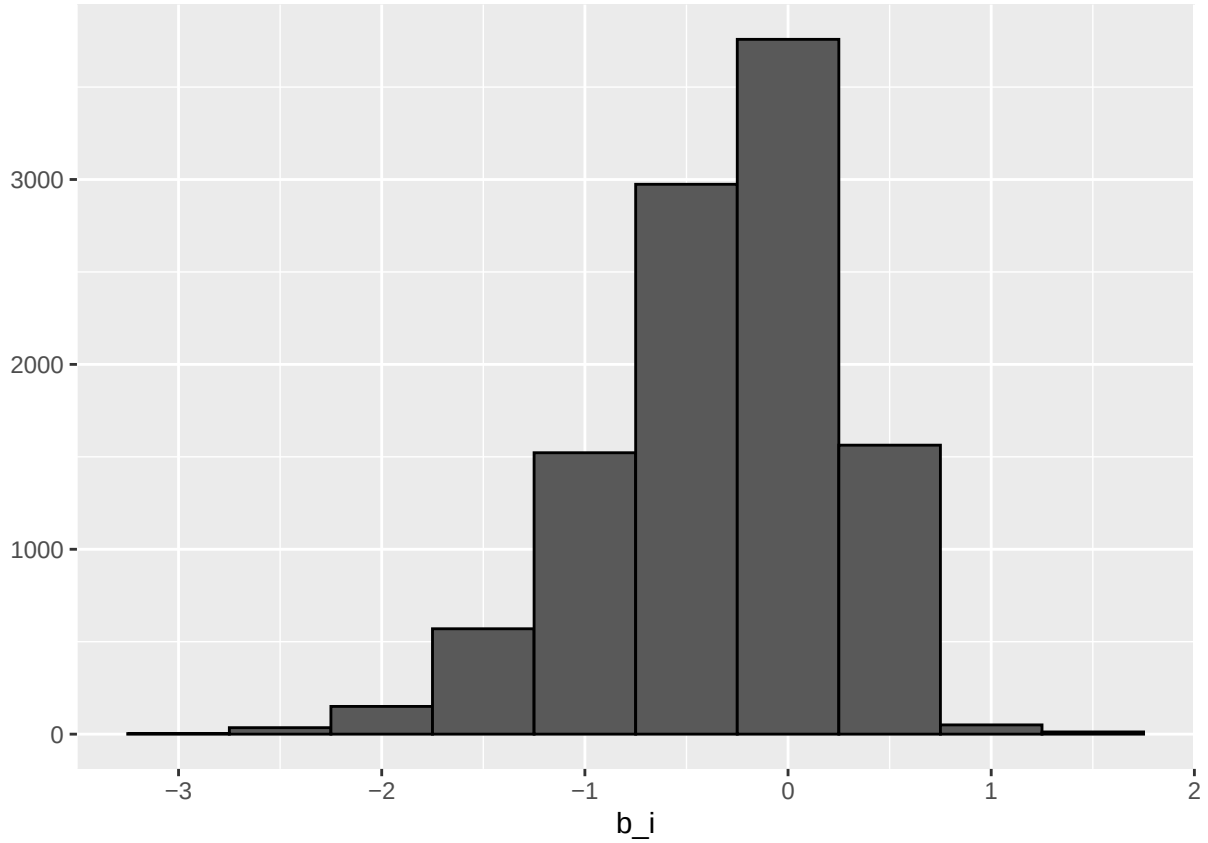
The result RMSE is a little more than 1, which means on average the prediction is off by about 1, which is not very good.

Second model: Movie Effect

As we saw on the exploratory analysis some are rated more than other We can augment our previous model by adding the term b_i to represent average ranking for movie i Y_u,i = mu + b_i + error_u,i.
We can again use least squared to estimate the movie effect.

```
mu <- mean(train_set$rating)
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

We can see that variability in the estimate as plotted here:



Predict ratings and apply RMSE to predicted ratigns:

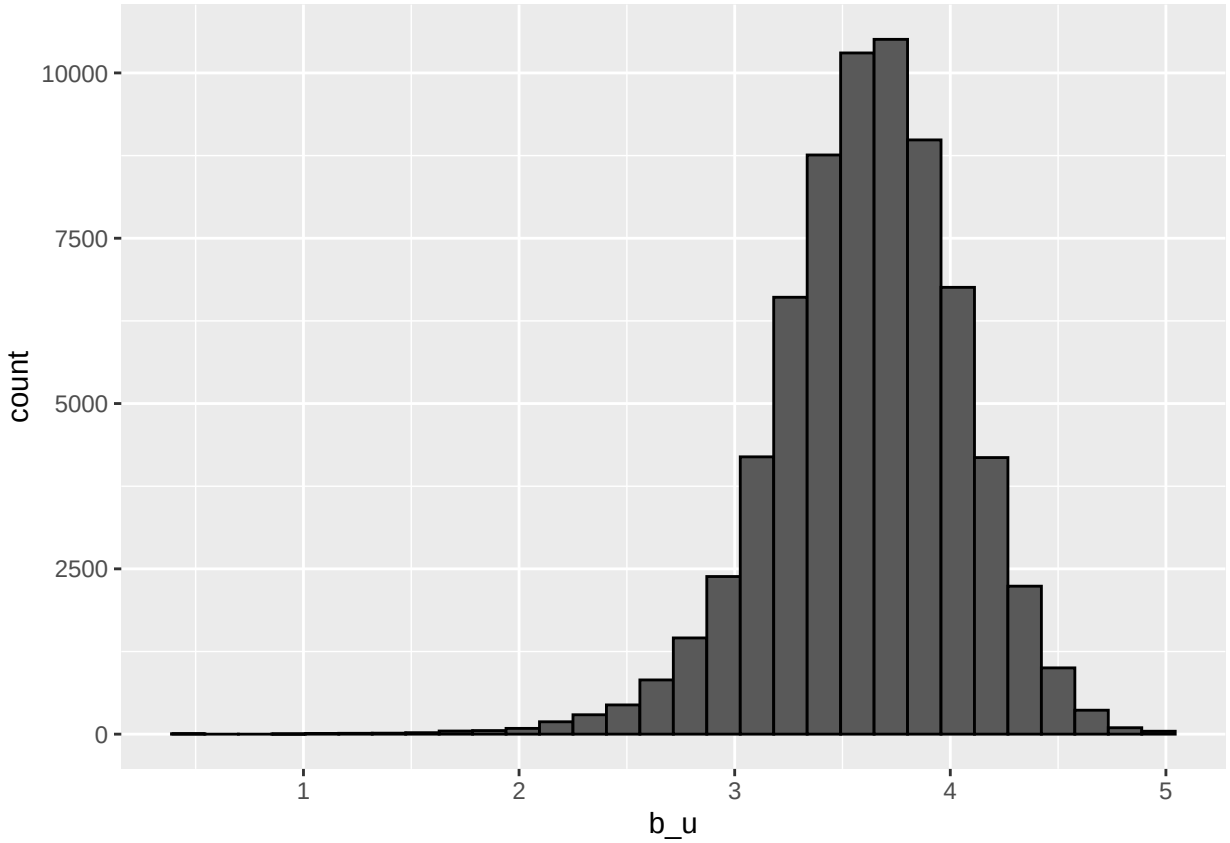
```
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
model_rmse_2 <- RMSE(predicted_ratings, test_set$rating)
model_rmse_2
```

```
## [1] 0.9437429
```

As we can see is better than naive rmse.

Third model: Movie + User Effect

Similar to the movie effect, intrinsic features of a given user could also affect the ratings of a movie. Let's compute the user U for , for those who ratedover 100 movies.



Notice that there is substantial variability across users as well: some users are very cranky and others love every movie.
This implies that a further improvement to our model may be Y_u,i = mu + b_i + b_u + error_u,i.
We could fit this model by using use the lm() function but it would be very slow.
We now further add the bias of user (b_u) to the movie effect model.

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

Now let's see how RMSE improved this time.

```
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

model_rmse_3 <- RMSE(predicted_ratings, test_set$rating)
model_rmse_3
```

```
## [1] 0.865932
```

Adding the user effect dramatically increased the RMSE to lower than 0.9.

Fourth model: Regularization of both movie and user effects

A machine learning model could be over-trained if some estimates were from a very small sample size. Regularization technique should be used to take into account the number of ratings made for a specific movie, by adding a larger penalty to estimates from smaller samples. To do this, a parameter lambda will be used. Cross validation within the test set can be performed to optimize this parameter before being applied to the validation set.

Perform cross validation to determine the parameter lambda

To train one single lambda value for both movie and user effects, We use 10-fold cross validation here within only the edx set.
Split the data into 10 parts.

```
set.seed(1, sample.kind = "Rounding")
```

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

```
cv_splits <- createFolds(edx$rating, k=10, returnTrain =TRUE)
```

Define a matrix to store the results of cross validation:

```
lambdas <- seq(0, 8, 0.1)
rmses <- matrix(nrow=10,ncol=length(lambdas))
```

Perform 10-fold cross validation to determine the optimal lambda, it will take several minutes.

```
for(k in 1:10) {
  train_set_k <- edx[cv_splits[[k]],]
  test_set_k <- edx[-cv_splits[[k]],]

  # Make sure userId and movieId in test set are also in the train set
  test_final <- test_set_k %>%
    semi_join(train_set_k, by = "movieId") %>%
    semi_join(train_set_k, by = "userId")

  # Add rows removed from validation set back into edx set
  removed <- anti_join(test_set_k, test_final)
  train_final <- rbind(train_set_k, removed)

  mu <- mean(train_final$rating)

  rmses[k,] <- sapply(lambdas, function(l){
    b_i <- train_final %>%
      group_by(movieId) %>%
      summarize(b_i = sum(rating - mu)/(n()+l))
    b_u <- train_final %>%
      left_join(b_i, by="movieId") %>%
      group_by(userId) %>%
      summarize(b_u = sum(rating - b_i - mu)/(n()+l))
    predicted_ratings <-
      test_final %>%
      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_final$rating))
  })
}
```

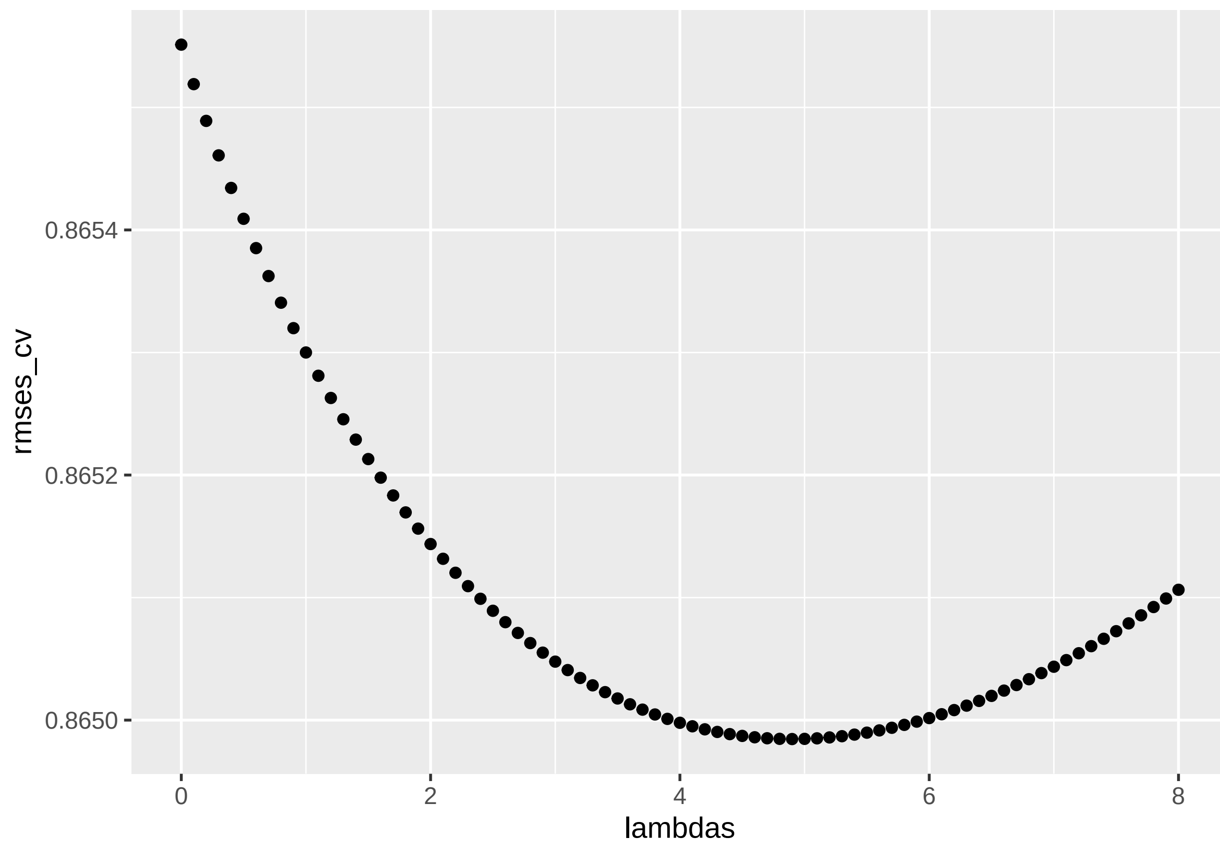
```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
## Joining, by = c("userId", "movieId", "rating", "timestamp",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

Perform the colum average of rmses:

```r
rmses_cv <- colMeans(rmses)
```

Plot the rmses cross validation vs lambdas:

```r
qplot(lambdas,rmses_cv)
```



Get the minimal RMSE as the optimized lambda for model:

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

```
## [1] 4.9
```

The optimized lammda is 4.9.

Model generation and prediction

Now we use this parameter lambda to predict the validation dataset and evaluate the RMSE.

```
mu <- mean(edx$rating)
movie_avg_reg <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))
user_avg_reg <- edx %>%
  left_join(movie_avg_reg, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))
predicted_ratings_4 <-
  test_set %>%
  left_join(movie_avg_reg, by = "movieId") %>%
  left_join(user_avg_reg, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
model_rmse_4 <- RMSE(predicted_ratings_4, test_set$rating)
model_rmse_4
```

```
## [1] 0.8567695
```

With the regularized of movie and user affect, the RMSE it is lower than 0.86.
Our fourth model is the best over the previous ones and therefore our final model to test with the **validation** data set.

Now We will create a tibble for the RMSE result to store all the result from each method to compare.

```
rmse_results <- tibble(method = c("Just the average","Movie Effect Model","Movie + User Effects Model",
                       , RMSE = c(naive_rmse,model_rmse_2,model_rmse_3,model_rmse_4))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0599043
Movie Effect Model	0.9437429
Movie + User Effects Model	0.8659320
Regularized Movie + User Effect Model	0.8567695

RMSE of the validation set

```
valid_pred_rating <- validation %>%
  left_join(movie_avg_reg, by = "movieId") %>%
  left_join(user_avg_reg, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

model_final <- RMSE(validation$rating, valid_pred_rating)
model_final
```

```
## [1] 0.8648185
```

The RMSE returned by testing our algorithm on the validation it is **0.86481**.
Now We will add the validation result to the tibble as show bellow:

```
rmse_results <- bind_rows( rmse_results,
                           tibble(method = "Validation Results" , RMSE = model_final))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0599043
Movie Effect Model	0.9437429
Movie + User Effects Model	0.8659320
Regularized Movie + User Effect Model	0.8567695
Validation Results	0.8648185

Conclusion

I have developed a naive approach, movie effect, movie + user effect and regularized movie + user effect, the best RMSE given by the fourth model.
Another approach to consider is the age and genres bias, the matrix factorization thecnique could improve the perform, or even the ensemble method should be considered en the future to apply on the MovieLens dataset.