Movie Recommendation System | Machine Learning

HarvardX: PH125.9x Data Science Capstone

Russell Bayes | October 07, 2021

Introduction

The objective of this project is to build a movie recommendation system using machine learning. As part of the project structure of the data , visualize it and then progressively build a number of different models with the aim of achieving a RMSE lower than 0.86490.

MovieLens Data set

GroupLens collected and provided the data. GroupLens which is research lab at the University of Minnesota that specializes in recommender systems, online communities, mobile and ubiquitous technologies, digital libraries and local geographic information systems. GroupLens have collected millions of movie reviews and offer these data sets openly to the data science community. For this project the 10M version will be used. It contains 10 million ratings on 10,000 movies by 72,000 users. It was released in 2009.

Data Loading

This code was supplied as part of the project.

```
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")

library(tidyverse)
library(caret)
library(data.table)
library(tinytex)
library(knitr,knitLatex)

# 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")

# if using R 4.0 or later:
movies <- as.data.table(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)
```

Exploratory Data Analysis

We start by analyzing the data structure. There are 9,000,055 observations and 6 columns. Each observation represents a rating given by one user for one movie. Columns include userId, movieId, rating, timestamp, title and genres. Timestamps represent seconds since midnight UTC January 1, 1970.

```
## 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|A
##  - attr(*, ".internal.selfref")=<externalptr>
```

The summary function provides a statistical summary of the data.

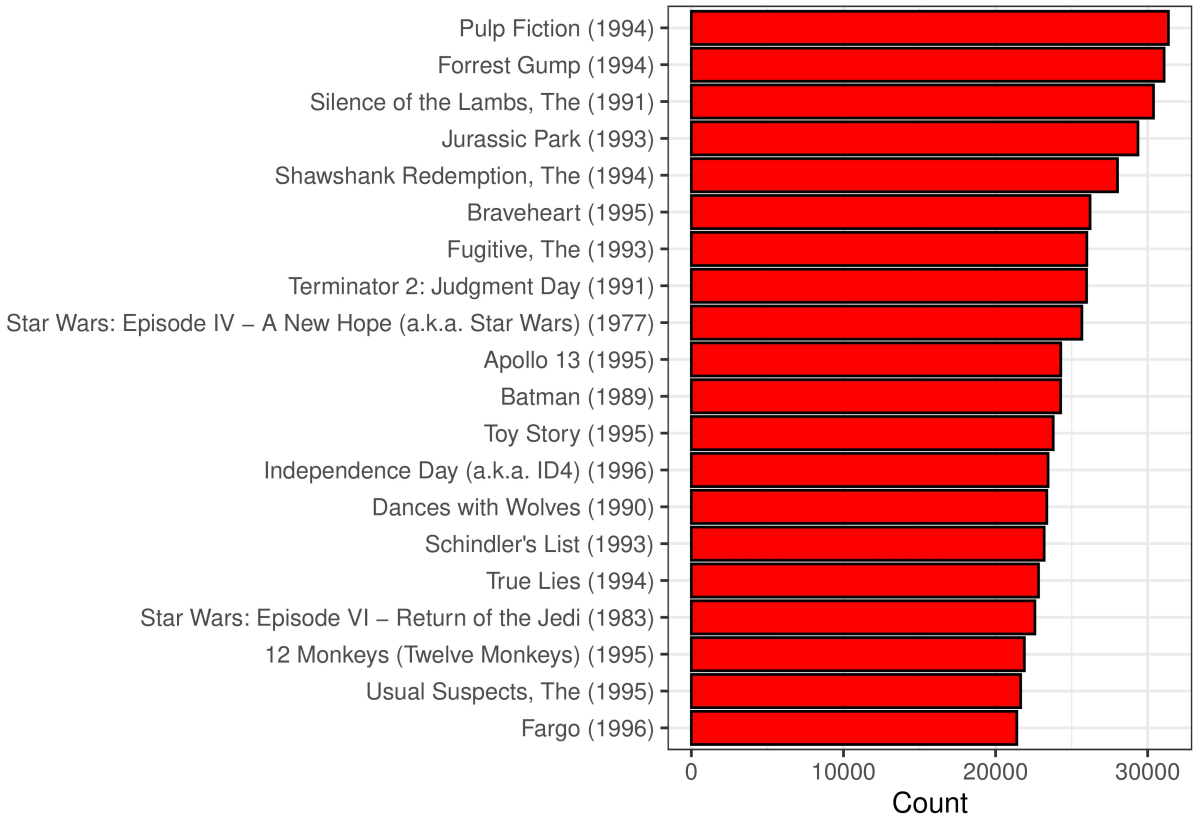
```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 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
##  Length:9000055
##  Class :character
##  Mode  :character
##
##
##
```

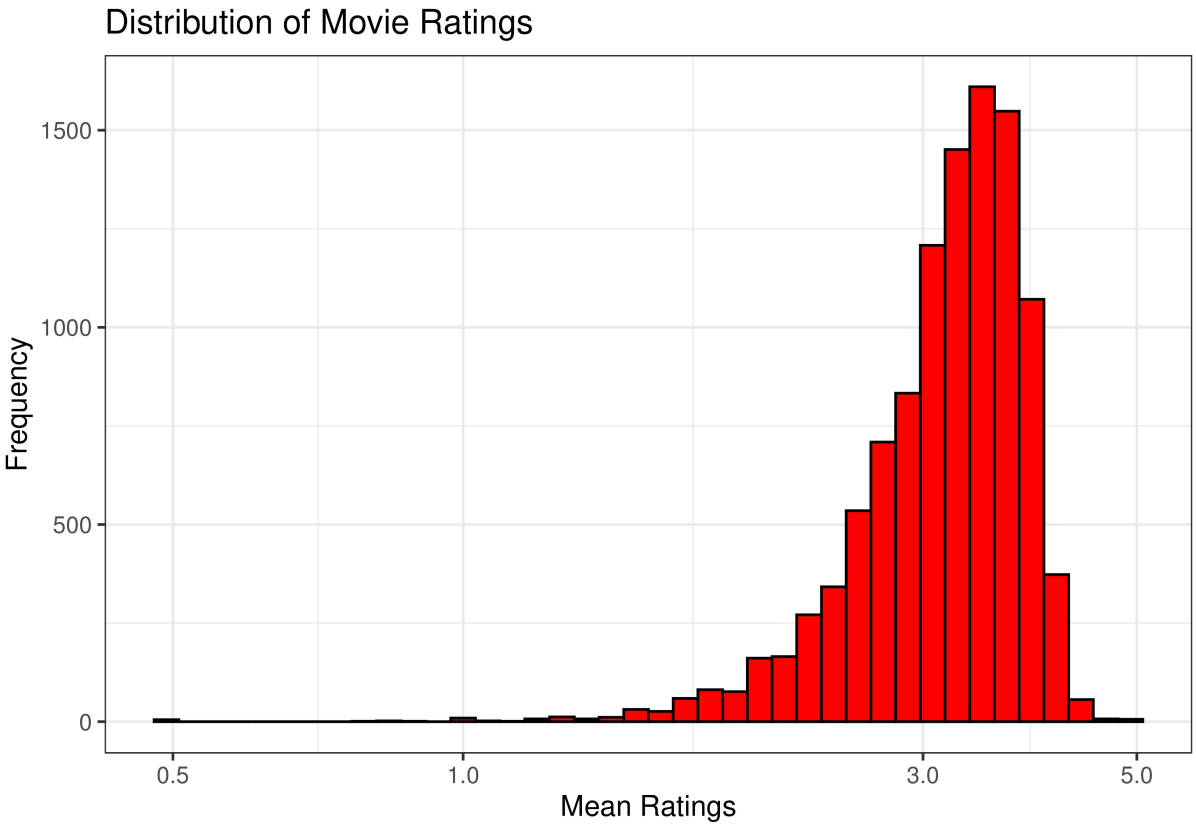
There are 69,878 unique users and 10,677 movies.

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

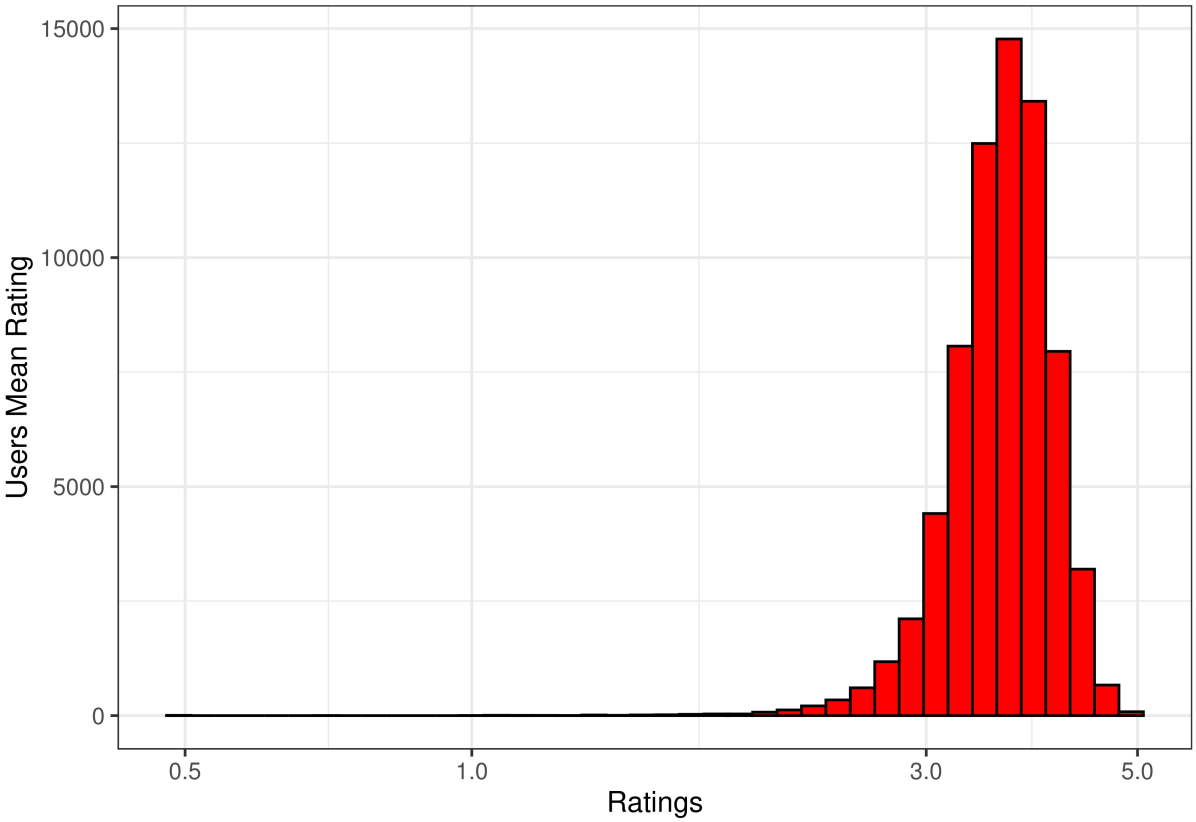
The most rated movies in this data set.



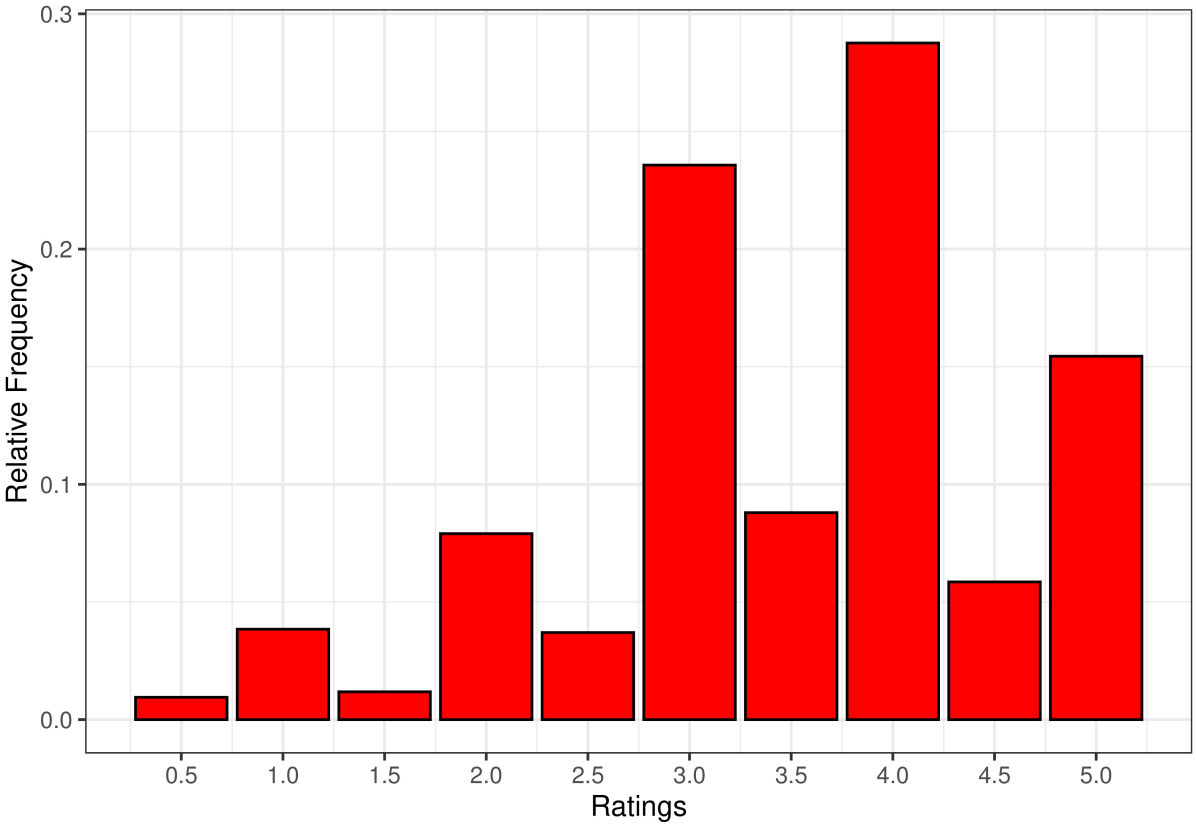
The distribution of the mean ratings by movieId.



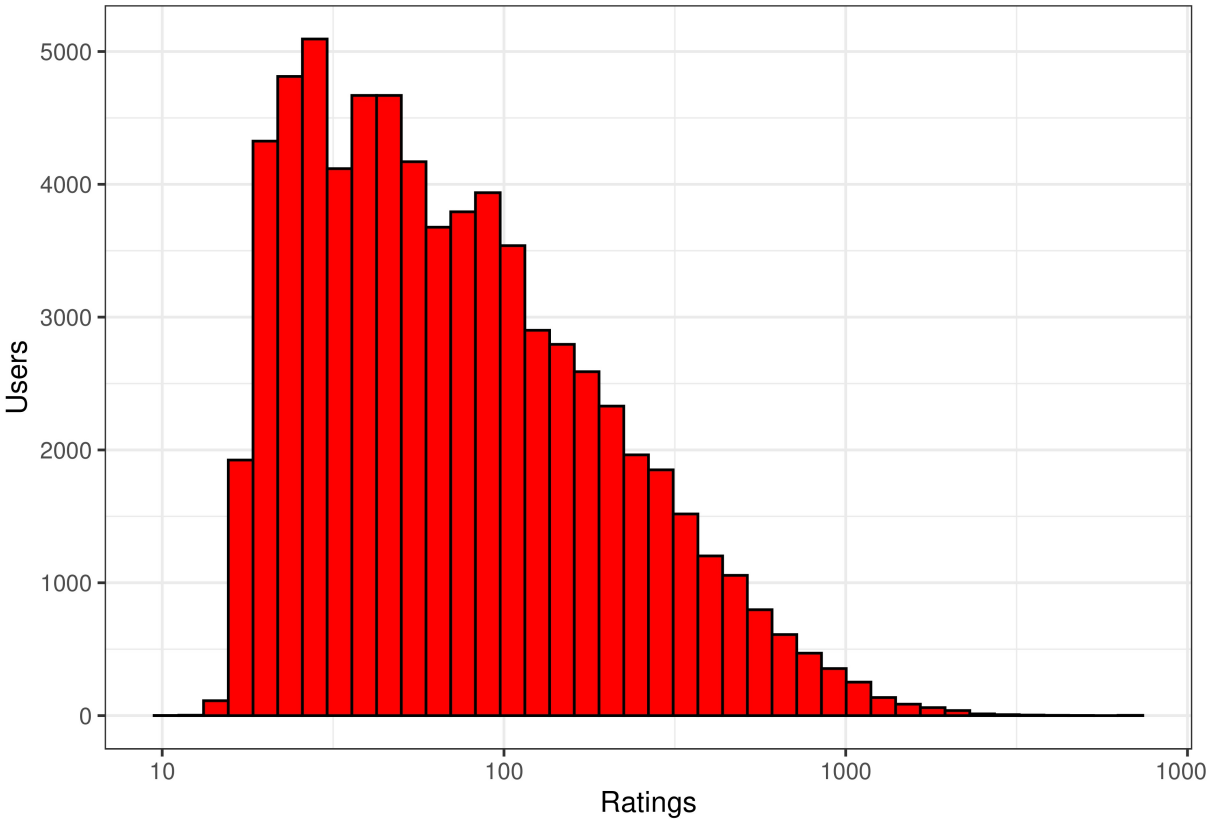
The distribution mean movie rating VS users shows that some user are more generous with there rating that others.



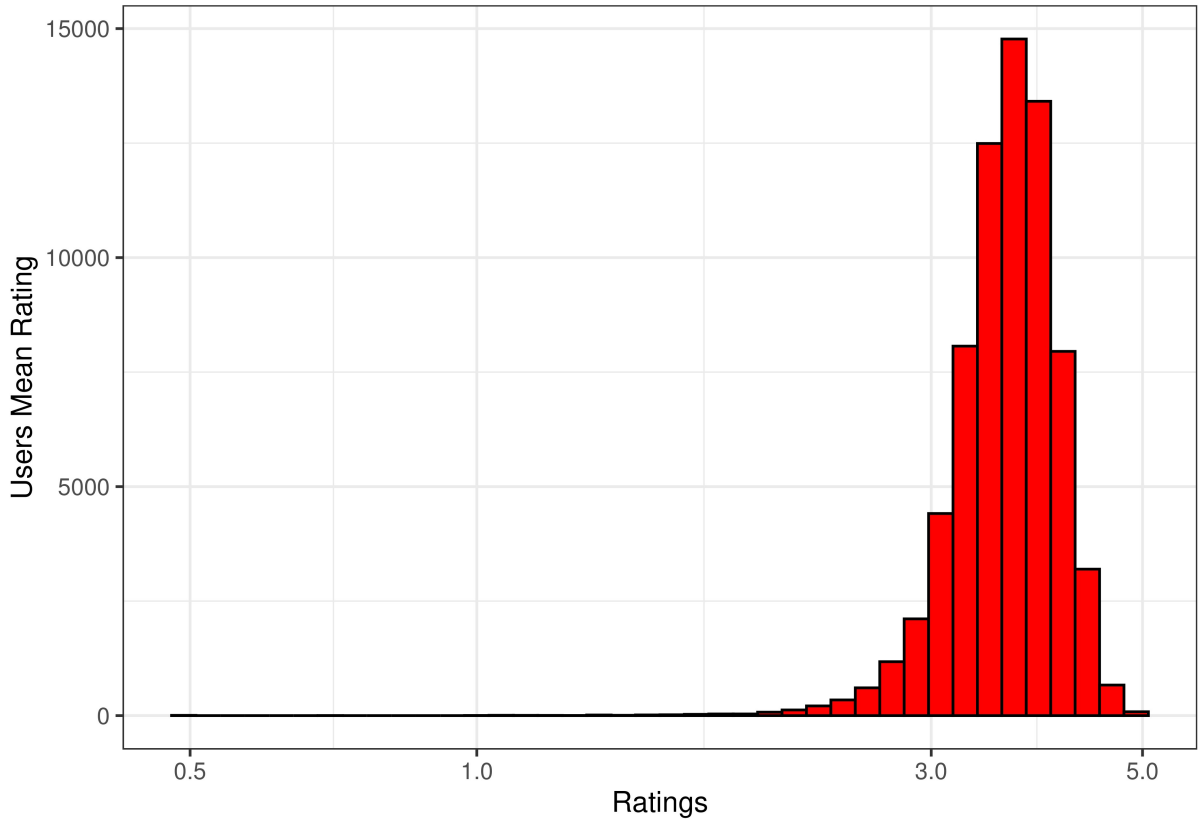
The distribution of the movie ratings shows a range of 0.5 to 5.



The number of ratings VS users shows a right skew in its distribution.



The mean of ratings VS users shows that some user are more generous with there rating that others.



Training and Testing of the Algorithm

The 10M data set is divided into two sets: edx and validation. The former is further split into two where the algorithm will be built and tested. Its final accuracy will then be tested using the validation set.

```
# Create train and test sets
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_set <- edx[-test_index,]
temp <- edx[test_index,]

# Matching userId and movieId in both train and test sets
test_set <- temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

# Adding back rows into train set
removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)

rm(test_index, temp, removed)
```

Approach and Evaluation

During the course of the project a number of models will be assessed. The accuracy of the models will be evaluated using the residual mean squared error (RMSE):

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

N is defined as the number of user/movie combinations, $y_{u,i}$ as the rating for movie i by user u with the prediction as $\hat{y}_{u,i}$. The RMSE is a commonly used loss function that simply measures the differences between predicted and observed values. It can be interpreted similarly to a standard deviation. For this exercise if the number is larger than 1 it means our typical error is larger than one star. The project objective is to produce a finalized model that produces an error below 0.8649 on the validation data set. Accuracries will be compared across all models using the code below.

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

Models

Model - Linear Model: Mean

This model uses the simplest approach possible which is to take the mean rating for all movies in the training set and applies this mean as the prediction for the training set.

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

$\epsilon_{u,i}$ is defined as the independent sample errors and μ the true rating for all movies. Statistical theory tells us that the estimate that minimizes the RMSE is the least RMSE is the squares estimate of μ. For this exercise it is the mean of all ratings.

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

If we apply this as our prediction we get the resulting RMSE:

Method	RMSE
Model - Random Prediction	1.060054

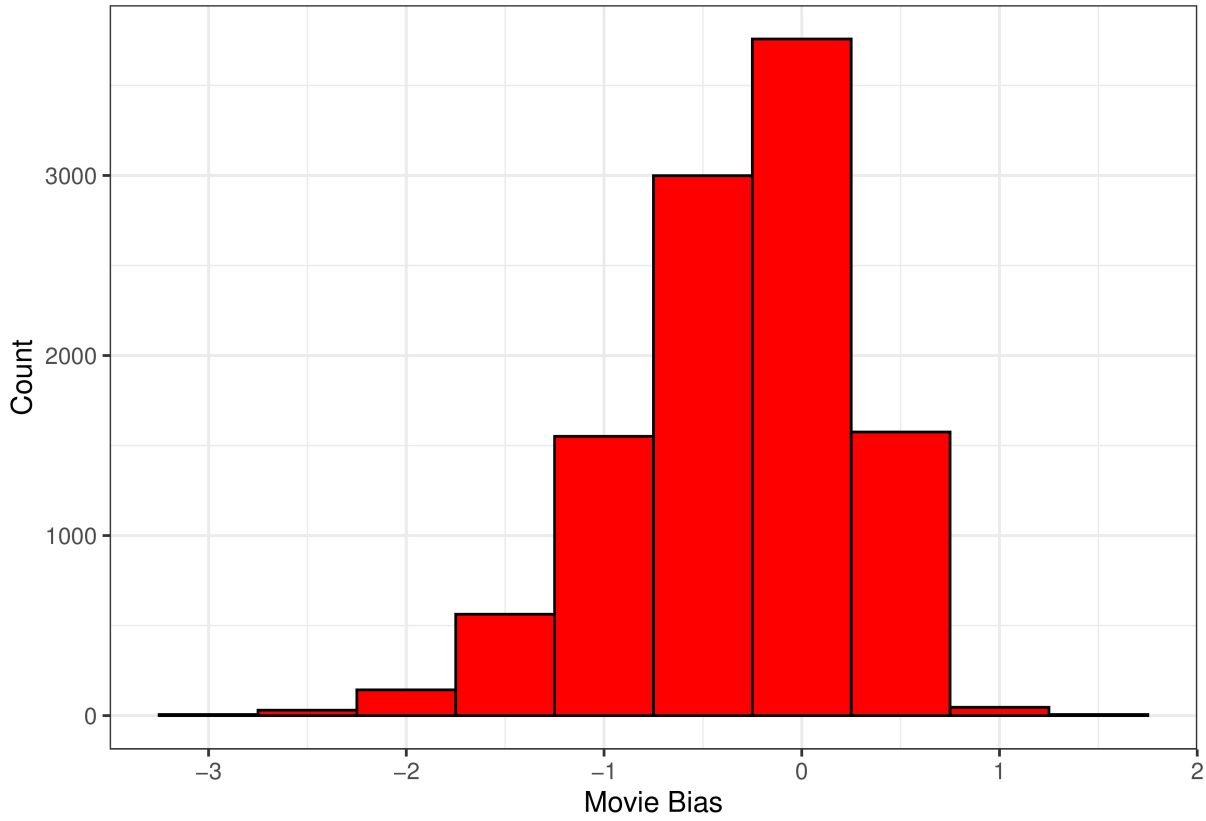
Our error is just larger than 1. This simple model allows use to create a baseline RMSE on which to build and check our other models.

Model - Linear Model: Mean + Movie bias

From the data exploration it can be seen that some movies are more popular than others and receive higher ratings. This insight provide us a way improve on the previous mode by taking into account movie bias. We shall add the term b_i to reflect this. It is the average of $Y_{u,i} - \hat{\mu}$ for each movie i.

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

We can see this bias through this distribution. The mean is at 0 so a b_i of 1.5 reflects a 5 star rating.



Here is the impact of adding this bias to our model by running a RMSE test:

Method	RMSE
Model - Random Prediction	1.0600537
Model - Linear Model: Mean + Movie bias	0.9429615

Model - Linear Model: Mean + Movie bias + Users bias

Bias can be found in users as well. Some tend to rate more positively and others negatively. We can add this effect to the model as b_u.

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

We can estimate $\hat{b}_u$ as the average of $y_{u,i} - \hat{\mu} - \hat{b}_i$.

```
## # A tibble: 6 x 2
##   userId    b_u
##    <int>  <dbl>
## 1      1  1.67
## 2      2 -0.434
## 3      3  0.268
## 4      4  0.698
## 5      5  0.100
## 6      6  0.336
```

A RMSE test will show how much we can reduce our typical error by accounting for User bias:

Method	RMSE
Model - Random Prediction	1.0600537
Model - Linear Model: Mean + Movie bias	0.9429615
Model: Mean + movie bias + user effect	0.8646843

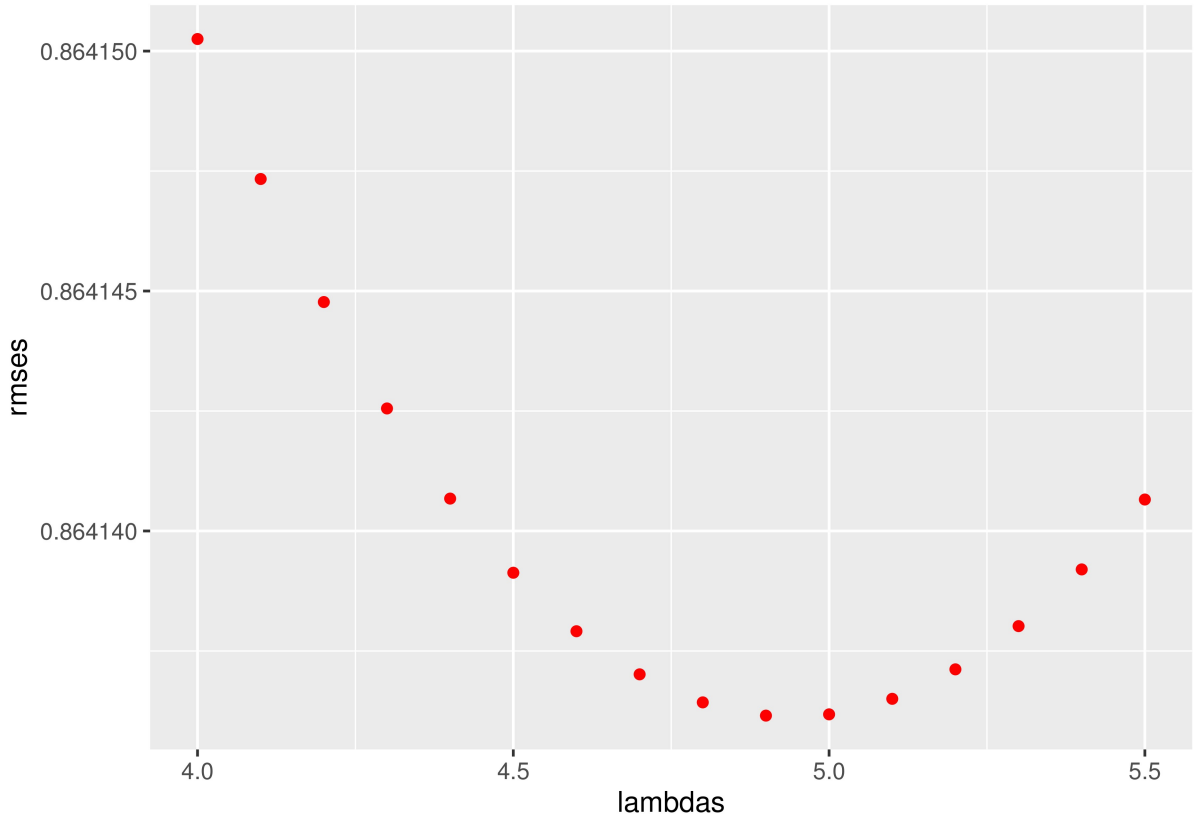
By simply factoring in movie and user biases we've managed to lower our error to ~ 0.86. Which is very close to the project target but it unlikely we will achieve that RMSE on the validation data which is the objective for the project so it is worth investigating other models to continue to reduce the RMSE to as low as it can.

Model - Regularization

The linear models provided a good estimation the ratings. From the data exploration we can see that there is some noise in the data which we need to try account for in order to reduce our RMSE further still. From our data exploration we found that there were a number of ratings for obscure or niche movies by a very small number of users. This is what is meant by noise and this adds variability and can cause our prediction to be less accurate in turn this increases the RMSE. What we shall do next is apply regularization to penalize large estimates formed by small sample sizes to reduce this effect. The optimal penalty to use, λ, can be found using cross-validation and applied to our model.

```
lambdas <- seq(4, 5.5, 0.1)
rmses <- sapply(lambdas, function(x){
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu_hat)/(n()+x))
  b_u <- train_set %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu_hat)/(n()+x))
  predicted_ratings <- test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu_hat + b_i + b_u) %>%
    pull(pred)
  return(RMSE(predicted_ratings, test_set$rating))
})
```

What we can see from the plot is a range of lambdas VS RMSE. The optimal value for lambdas is we where lowest RMSE is achieved.




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

RMSE result:

Method	RMSE
Model - Random Prediction	1.0600537
Model - Linear Model: Mean + Movie bias	0.9429615
Model: Mean + movie bias + user effect	0.8646843
Model - Regularization + Movie & User Bias	0.8641362

We can see the regularization has managed to still further reduce the RMSE value we obtained from our other models, but there are a number of package and tools that use very different methods to predict rating. So lets have a look at some and see if we get better result from some of the other methods.

Model - Matrix Factorization with recosystem

Recommender systems use historical data to make predictions. One of the most popular approaches in achieving this is collaborative filtering. It is based on historical behavior by its users. So far we have approached a data set that features sparsity and biases with models that account for these effects with decent accuracy. To get better results we turn to a more advanced method called matrix factorization. Our user data is processed as a large and sparse matrix, then decomposed into two smaller dimensional matrices with latent features and less sparsity. To make the process more efficient the recosystem package will be used. For more information on it click here. We start by converting data into the recosystem format, find the best tuning parameters, train and finally test it.

```r
if (!require(recosystem))
  install.packages("recosystem", repos = "http://cran.us.r-project.org")
library(recosystem)
set.seed(123, sample.kind="Rounding")

#Select only data that will be need to trai and test our model
train_set_factorization <- train_set %>% select(movieId, userId, rating)
test_set_factorization <- test_set %>% select(movieId, userId, rating)

#Convert the data.frame into a matrix to be feed to the Recosystem
train_set_factorization <- as.matrix(train_set_factorization)
test_set_factorization <- as.matrix(test_set_factorization)

#Write of the files so that we don't have to hold the Matrix in memory
write.table(train_set_factorization, file = "MFtrainingset.txt", sep = " ", row.names = FALSE,col.names
write.table(test_set_factorization, file = "MFtestset.txt", sep = " ", row.names = FALSE, col.names = F

#Let R know where the files are stor
training_reco <- data_file("MFtrainingset.txt")
test_reco <- data_file("MFtestset.txt")


r = Reco()

#Set parameters by which we shall attempt to tune our model
opts = r$tune(training_reco,
              opts = list(dim = c(20),
              lrate = c(0.1, 0.2),
              costp_l1 = 0,
              costq_l1 = 0,
              nthread = 4,
              niter = 30))

r$train(training_reco, opts = c(opts$min, nthread = 4, niter = 30))

#We write predictions to a tempfile on HDisk
stored_prediction = tempfile()
r$predict(test_reco, out_file(stored_prediction))

#load the test set rating data
real_ratings <- read.table("MFtestset.txt", header = FALSE, sep = " ")$V3

#Assign predicted values to a object in the memory
results_reco <- scan(stored_prediction)
```

With the algorithm trained we now test it to see the resulting RMSE:

Method	RMSE
Model - Random Prediction	1.0600537
Model - Linear Model: Mean + Movie bias	0.9429615
Model: Mean + movie bias + user effect	0.8646843
Model - Regularization + Movie & User Bias	0.8641362

Method	RMSE
Model - Matrix Factorization with recosystem	0.7884723

There has been a significant improve in the RMSE we achieved from this model compared with the other other we generated. This model provides a RMSE comfortably below project objective and so should be able to obtain an RMSE below the project target for the validation data set.

Final Validation

Now that we have down selected to a model using matrix factorization that should be able to meet the project goals. The final step is to train our model using the edx set and then test its accuracy on the validation set:

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

#Select only data that will be need to trai and test our model
edx_factorization <- edx %>% select(movieId, userId, rating)
validation_factorization <- validation %>% select(movieId, userId, rating)

#Convert the data.frame into a matrix to be feed to the Recosystem
edx_factorization <- as.matrix(edx_factorization)
validation_factorization <- as.matrix(validation_factorization)

#Write of the files so that we don't have to hold the Matrix in memory
write.table(edx_factorization, file = "MF_edx_set.txt", sep = " ", row.names = FALSE,col.names = FALSE)
write.table(validation_factorization, file = "MF_validation_set.txt", sep = " ", row.names = FALSE, col

#Let R know where the files are stor
edx_reco <- data_file("MF_edx_set.txt")
validation_reco <- data_file("MF_validation_set.txt")


r = Reco()

#Set parameters by which we shall attempt to tune our model
opts = r$tune(edx_reco,
              opts = list(dim = c(20),
              lrate = c(0.1, 0.2),
              costp_l1 = 0,
              costq_l1 = 0,
              nthread = 4,
              niter = 30))

r$train(edx_reco, opts = c(opts$min, nthread = 4, niter = 30))

#We write predictions to a tempfile on HDisk
Final_prediction = tempfile()
r$predict(validation_reco, out_file(Final_prediction))

#load the test set rating data
real_ratings <- read.table("MF_validation_set.txt", header = FALSE, sep = " ")$V3
```

```
#Assign predicted values to a object in the memory
final_reco <- scan(Final_prediction)
```

Method	RMSE
Model - Random Prediction	1.0600537
Model - Linear Model: Mean + Movie bias	0.9429615
Model: Mean + movie bias + user effect	0.8646843
Model - Regularization + Movie & User Bias	0.8641362
Model - Matrix Factorization with recosystem	0.7884723
Final validation: Matrix factorization using recosystem	0.7863785

Conclusion

Our final RMSE is 0.786. Significantly below our target of 0.8649. We built and tested several models and achieved our best accuracy using matrix factorization which was simplified through the recosystem package. There is still much potential to reduce the RMSE further, if i were continuing this work i would look at how Genres could be used to improve results, as well as the release year of the movie.

I would also investigate the potential to cluster users I a hope to find user who have similar preference and use that information to further improve upon the results obtained.