Minimizing RMSE with MovieLens Dataset

Mark Richards

4/30/2019

Introduction

This paper will outline the process of building a recommendation system for the MovieLens data set. It will cover several methods for building predictions and measure their performance using the root mean squared error (RMSE) of the predictions vs a know set of rating. The goal is to reach a RMSE of 0.87750 or lower.

In order to test different prediction methods we will start by building training and validation data sets. These will be created by partitioning the available MovieLens 10M data set. The training set will contain 90% of the available data while the validation set contains the remaining 10%. We will then perform some basic investigation of the training set to see what information is available. from there we will start testing different algorithms to find which methods perform best. Ultimately we hope to reach an RMSE of 0.87750 or lower.

Note: Much of the code will contain "If" statements that load results of calculations if the RDS file containing them is available. This is due to limited computing power. The main code chunks are being run on an instance or R server and this report is being written on a laptop. The files loaded will be made available on GitHub providing they are not above the current file size limit of 100MB.

Analysis

We will start by loading the necessary packages for the analysis. These include the Tidyverse, Caret, and several others.

```
library(tidyverse)
library(caret)
library(ggplot2)
library(dplyr)
library(lubridate)
```

Next we will build the training and validation data sets. This code will check to see if the resulting data sets are available in the working directory as RDS files. If not it will download the MovieLens data and build the training and validations sets. The code to download the MoveLens 10M data set and build the training and vlidation partitions is provided by Harvard as part of their capstone course for a profesional certification in Data Science.

```
test<-file_test("-f","validation.rds") #checks to see if test set data exists
train<-file_test("-f","edx.rds") #checks to see if training set data exists

if (test & train == FALSE) { #run code block if one or both files are not available
  dl <- tempfile() # Following code chunk provided by Harvard EdX Capstone Course.
  download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

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

  movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
  colnames(movies) <- c("movieId", "title", "genres")
  movies <- as.data.frame(movies) %>%
```

```
    mutate(movieId = as.numeric(levels(movieId))[movieId],
                                            title = as.character(title),
                                            genres = as.character(genres))

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

  # Validation set will be 10% of MovieLens data

  set.seed(1)
  test_index <- createDataPartition(y = movielens$rating,
                                    times = 1, p = 0.1, list = FALSE)
  edx <- movielens[-test_index,]
  temp <- movielens[test_index,]
  rm(test,train) #removes the file checks
}else { #run if both files exist
  edx<-readRDS("edx.rds") #reads the generated edx training set
  validation<-readRDS("validation.rds") #reads the generated validation test set
  rm(test,train) #removes the file checks
}
```

For convenience lets convert the date stamps of reviews to just the year the review was submitted

```
edx$timestamp<-date(as_datetime(edx$timestamp)) #convert timestamp to years
```

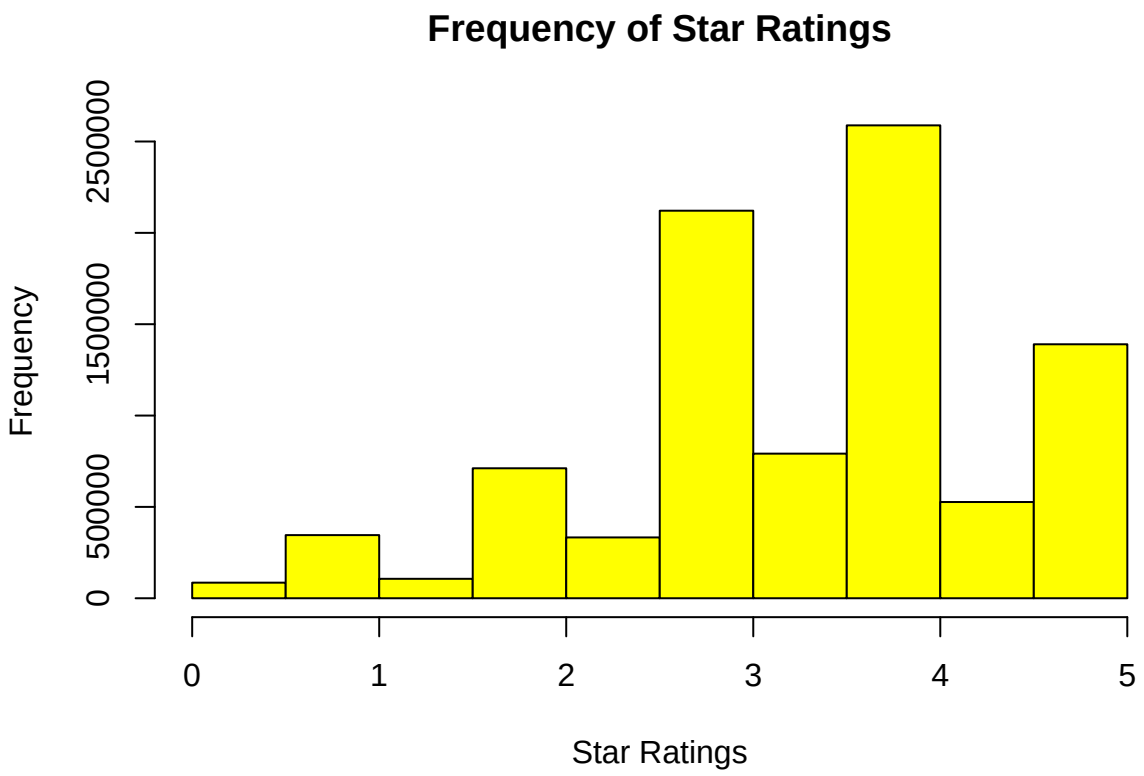
Now we can start looking at some basic information about the data set. We might want to know how large it is, how many movies are in it, or how many users are in it. We can check all of these things with a few simple lines of code.

```
edxdim<-dim(edx) #dimentions of the data set, name allows inline insertion with text
NumMovies<-n_distinct(edx$movieId) #number of movies
NumUsers<-n_distinct(edx$userId) #numbr of users
```

From this we can see that the training set contains 9000055 reviews with 6 variables. Inside this data there are 10677 different movies that have reviews. There are also 69878 users who provided ratings for those movies.

It might be useful to look at how many stars are generally given in the training set. We can build a plot showing the frequency of the different star ratings with the following code.

```
hist(x=edx$rating, main = "Frequency of Star Ratings", xlab = "Star Ratings",
     col ="yellow", breaks = seq(0,5,0.5)) #plot of star ratings in training set
```



From this graph we can see that 3 and 4 stars are the most common ratings given. To get an exact value we can take the average of all of the ratings in the training set with the following code.

```
trainMean<-mean(edx$rating) #average for all of the ratings in the training set
```

Here we see that the average rating is 3.5124652 stars. This fits with the plot shown above. Now what if we were to just use this average as our guess for the ratings in the validation set. What would the RMSE be?

To determine that we need to define the RMSE. We will build the RMSE as a function so we can call it and feed it our results. the following code will create a function for this. It calculates the error by subtracting the true rating from our predicted value. Then squaring the results and taking the average. This resulting average is then rooted to return the RMSE.

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

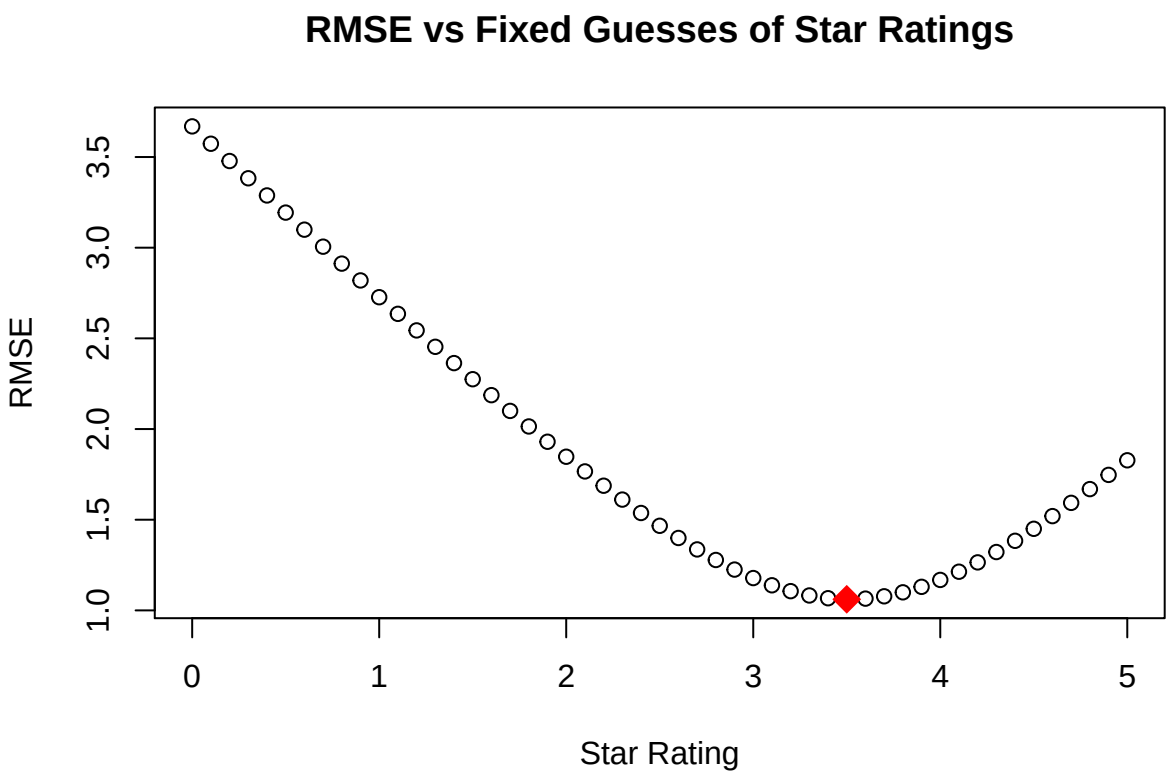
with this function in place we can now test our prediction of using the average star rating for all unknown ratings. To do this we set the true_rating for the RMSE function to the validation set ratings and our predicted rating to the average for the training set.

```
FixedAvg<-RMSE(validation$rating,mean(edx$rating))
```

The resulting RMSE is 1.0612018 This is a good baseline to measure the rest of our models against but it does not come close to the goal value of 0.87750. The next logical step is to see if there is any fixed star rating value that can lower the RMSE. We would expect the average to be the lowest but to be sure we can use the following code to check. It will build a list all values between 0 stars and 5 stars and calculate the RMSE then plot the results to find the lowest value we can achieve.

```
fixed<-integer() #initialize list
for (i in seq(0,5,0.1)){ #iterate over values 0 to 5
  fixed<-append(fixed,RMSE(validation$rating,i)) #calculate RMSE
```

```
}
plot(x=seq(0,5,0.1), y=fixed,main = "RMSE vs Fixed Guesses of Star Ratings",
     ylab = "RMSE", xlab = "Star Rating") #build plot of results
points(y=min(fixed), x=(which.min(fixed)/10)-0.1,col="Red",
       pch=18,bg="Red", cex=2) #add red point at lowest RMSE value
```



From the plot we can see that the star value of 3.5 gives us the lowest possible RMSE of 1.06 by guessing a fixed value. This confirms the inital thought that using the average would provide the best RMSE.

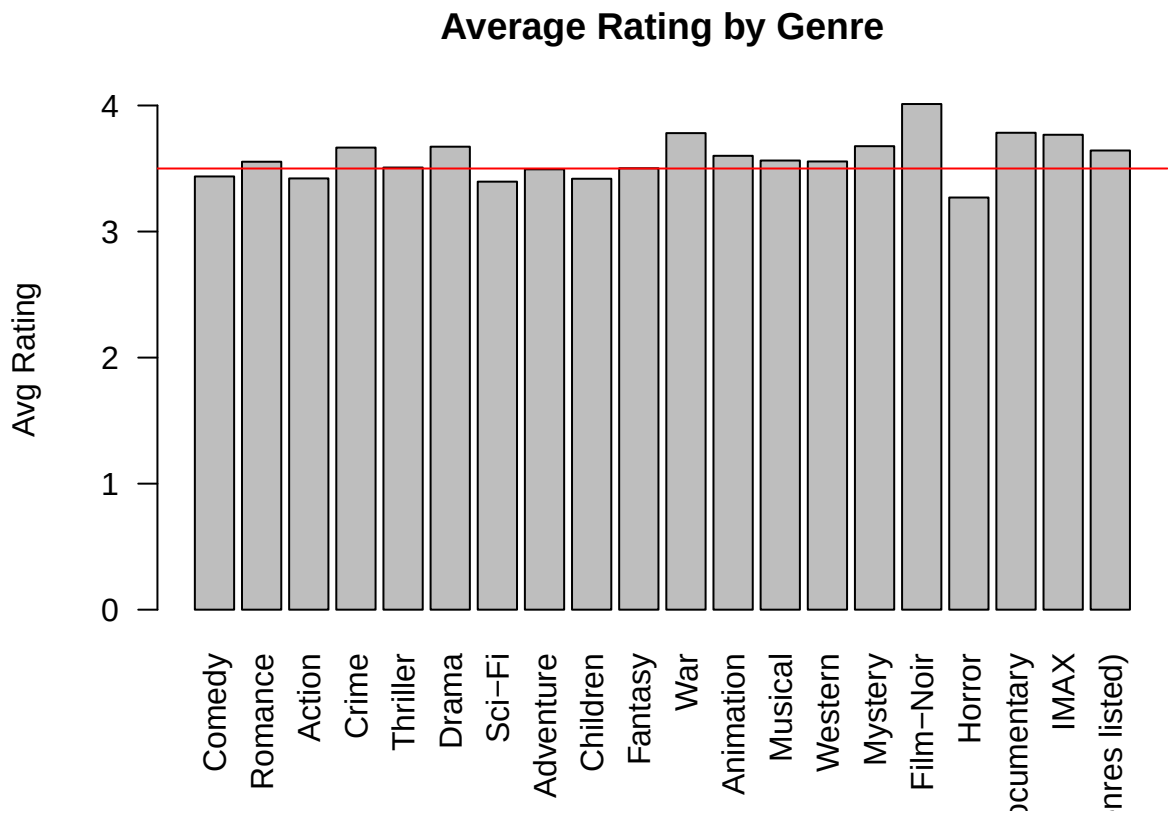
Continuing in this direction it might be worthwhile to investigate if different genres have different average ratings. Perhaps setting the prediction based on the genre average would yield better results. The current data set provides the reviews with a list of genres tagged to each entry as a single string. We can separate these out and create rows for each review and each genre using the code below. This operation can take some time to complete so it will check to see if an RDS of the data is available first. If it is, it will simply load the file rather than run the whole process.

```
genre<-file_test("-f","genres.rds") #checks to see if test set data exists
if (genre == FALSE) {
  genres<-separate_rows(edx, genres, sep = "[|]") #create a df showing 1 genre tag per row
  rm(genre)
}else {
  genres<-readRDS("genres.rds") #reads the generated edx training set
  rm(genre)
}
```

Now that the data has been separated we can take a look at the averages for each of the genres with this code. This will create a dataframe showing each genre and the average star rating for that genre. It will also add a red line at the overall average of 3.5 stars.

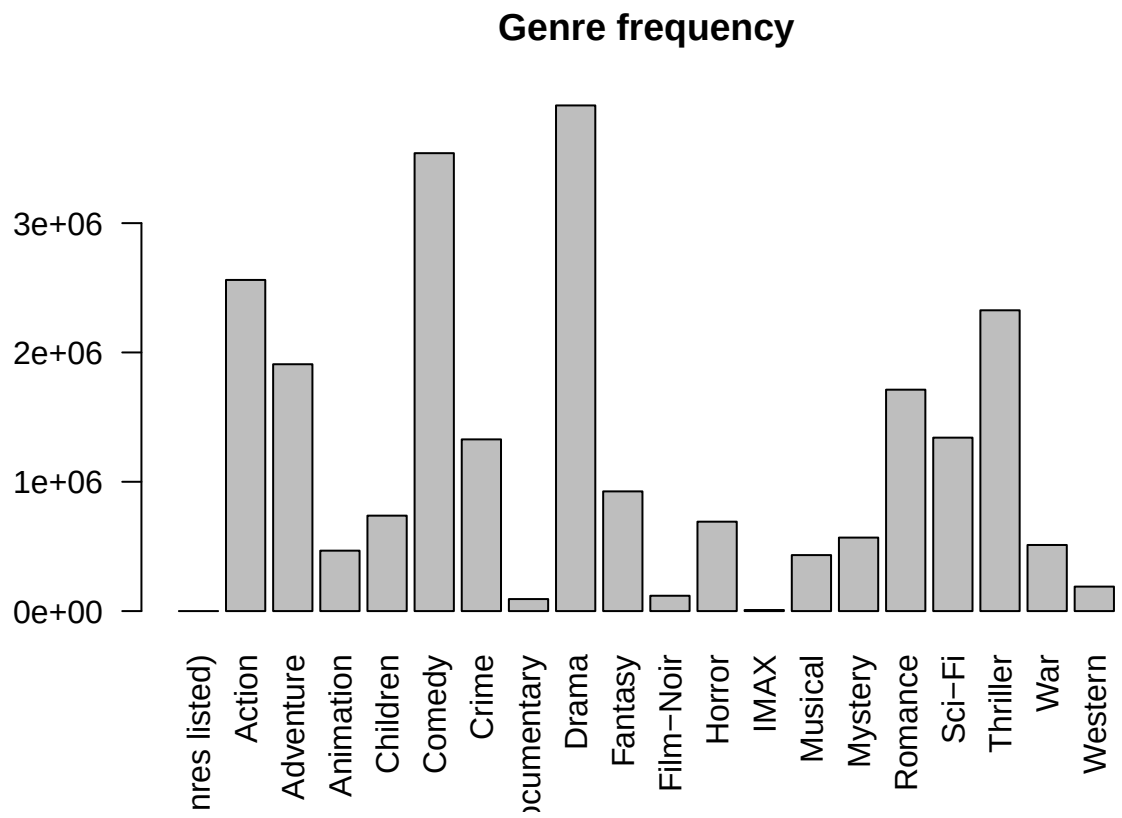
```
genreList<-data.frame(genres=unique(genres$genres), avg=0)
for (i in seq(1,length(genreList$genres),1)){
```

```
  inter<-genres %>% filter(genres == genreList$genres[i])
  genreList$avg[i]<-mean(inter$rating)
}
rm(inter)
barplot(names.arg =genreList$genres, height =genreList$avg, las=2,
        main = "Average Rating by Genre", ylab = "Avg Rating")
abline(a=3.5,b=0, col="Red")
```



From this graph we can see that the ratings center around the overall average. None of them vary too much with the largest deviations being Film-Noir and Horror. If these movies make up a dispropritanely large numer of the reviews in the training set then it might be worthwhile to adjust for genes. To find the number of movies in the data we can use the following code.

```
genres<-data.frame(table(genres$genres))
barplot(height = genres$Freq, names.arg = genres$Var1, main = "Genre frequency", las=2)
```

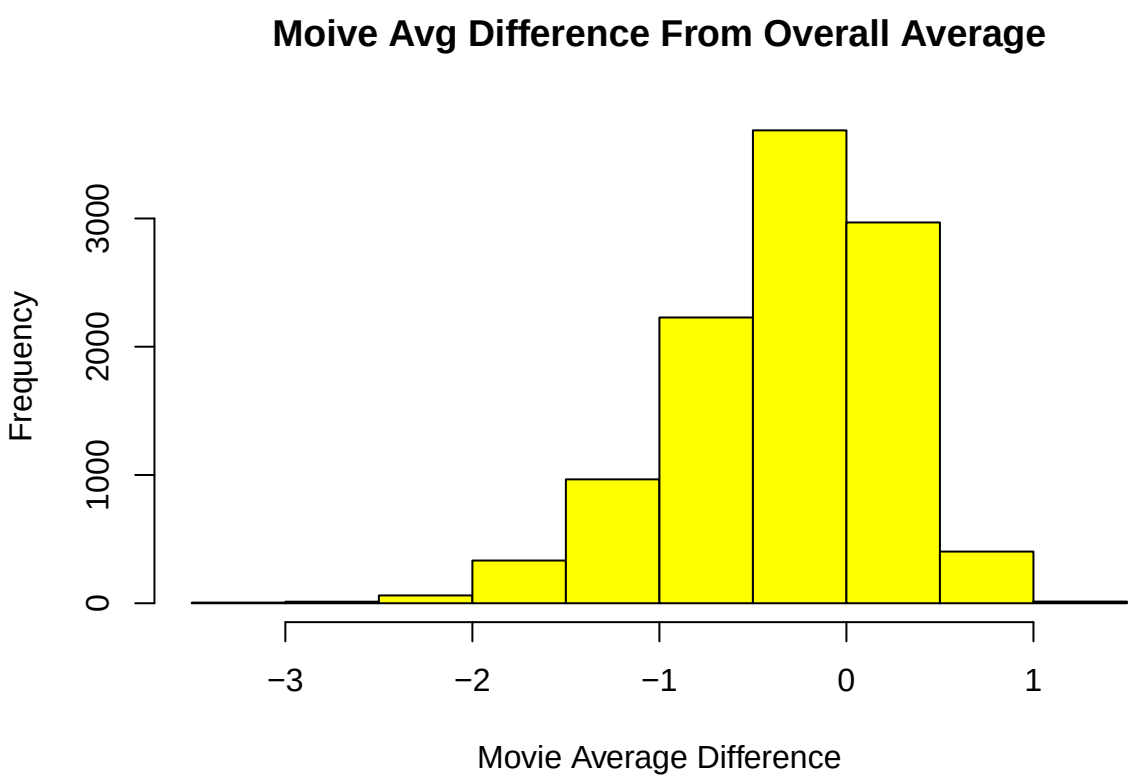


This plot shows that Film-Noir and Horror actually make up a very small percentage of the data. So it would not be much benifit to adjust for genres. This implies we should investigate other aspects of the data and other methods for making predictions.

Movies

while the genre averages may not provide much insight it could be that the information isnt granular enough. Here we are going to look at the movies themselves and see if indivigual movies deviate from the average of 3.5 stars. Logically it woudl make sense that some movies are generally considered good and would be rated above the average. It also seems quite probable that there are moves that are generally bad and will be rated below the average. We will use the following code to generate the averages for each indivigual movie. It will then generate a plot showing how frequency of the difference from the overall average.

```
movieAvg <- edx %>% group_by(movieId) %>% summarize(movieAvg = mean(rating - trainMean))
hist(movieAvg$movieAvg, main = "Moive Avg Difference From Overall Average",
     xlab = "Movie Average Difference", col="Yellow")
```



From this graph we can see that most movies fall within the 2.5 to 4.5 star range, with a majority of those falling below the average of 3.5 stars. Now we will use these movie averages and see if including them in our model can lower our RMSE. The following code builds a set of predictions by adding or subtracting the difference from the overall average for each movie. It then passes those predictions to the RMSE fucntion to calcuate our score.

```
moviePredict <- trainMean + validation %>% left_join(movieAvg, by='movieId') %>%
  .$movieAvg

movieRMSE <- RMSE(moviePredict,validation$rating)
```

This incluson gives us an RMSE of 0.9439087 which is below our previous score of 1.0612018. There are still improvments that need to be made if we intend to reach our goal of 0.8775.

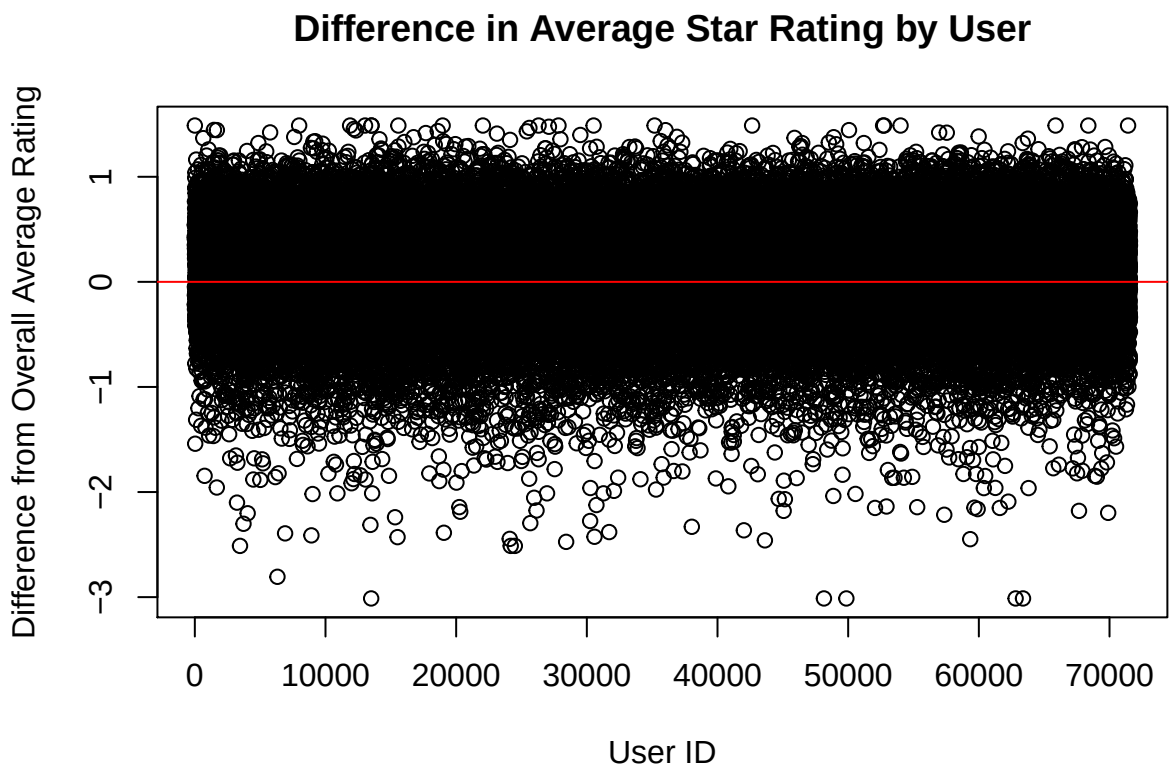
Users

In a sysem where the goal is to predict a users taste in movies it may be benificial to invesitgate users and the ratings they have provided. We will start by building averages for each user. It may be the case that some users score movies low over all and others score them high overall. To examine if this is worthwhile we can build a plot of users and their average ratings and see how they deviate from the overall average. The following code will calcuate this average for users with more than 100 ratings and then plot the difference in their average ratings with a red line idicating the overall average.

```
userEffect<-edx %>% group_by(userId) %>% summarize(userEffect = mean(rating- trainMean)) %>%
  filter(n()>=100)

plot(y=userEffect$userEffect, x=userEffect$userId,
     main = "Difference in Average Star Rating by User",
```

```
     ylab = "Difference from Overall Average Rating", xlab = "User ID")
abline(a=0,b=0, col="Red")
```



We can see there are a significant number of users who deviate from the average with there ratings. so now lets build a model with this uder effect acounted for. The following code will take the average rating users give and add or subtract it from the overall average and then calculate the RMSE.

```
userPredict <- trainMean + validation %>% left_join(userEffect, by='userId') %>%
  .$userEffect
userRMSE <- RMSE(userPredict,validation$rating)
```

Adjusting for users average rating gives and RMSE of 0.978336 which is again below the result of simply seting the average as our estimate which gave an RMSE of 1.0612018. It however is not as good as adjusting for the average score based on movie. which gave an RMSE of 0.9439087.

Given that both of these methods have given better results than just averaging across all the data we can try combining them into one algorithm to see if together than can provide even better results. The following code will take the movie effect and the user effect into account and generate an RMSE score.

```
usermovie <- validation %>%
  left_join(movieAvg, by='movieId') %>%
  left_join(userEffect, by='userId') %>%
  mutate(pred = trainMean + movieAvg + userEffect) %>%
  .$pred

combinRMSE <- RMSE(usermovie, validation$rating)
```

When both of these methods are combined we get tantalizingly close to our target. The resulting RMSE is 0.8850398 which is just 0.0075398 away from where we want to be.