Minimizing RMSE with MovieLens Dataset

Mark Richards

5/3/2019

Introduction

This paper will outline the process of building a recommendation system for the MovieLens 10M data set. It will cover several methods for building predictions and measure their performance using the root mean squared error (RMSE) of the predictions. The goal is to reach a RMSE of 0.87750 or lower.

Analysis

In order to test the prediction methods, we will start by building a training and a validation data set. These sets will be created by partitioning the available MovieLens 10M data. The training set will contain 90% of the available data while the validation set contains the remaining 10%. This gives us a large sample to train our models on and a small set to test their accuracy with.

Note: Many of the code blocks will contain IF statements that load results of calculations if there is an RDS file available that contains the results. Due to limited computing power on the laptop this report is being written on. Sections that were computationally intensive were offloaded to a server running R and the results were saved to RDS files. Loading them in allows the laptop to bypass this while still providing the necessary code. Any of the intermediate files that fall below the 100MB limit will be available on GitHub

The first step is to load the necessary packages for performing the analysis. These include the Tidyverse, Caret, and several others.

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

Once the necessary packages are loaded, we need to build the training and validation data sets. The base code to perform this is provided by Harvard as part of the Capstone course for its Professional Certification in Data Science. Since this process is computationally intensive it includes a check for the resulting files in the working directory. If it finds both the training and the validation set it skips running the code and simply loads the RDS files. If it cannot find one or both of the sets it will download the dataset and build the partitions.

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

if (test & train == FALSE) { #Run if one/both files are not available
  dl <- tempfile() # Code 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) #remove file checks
}else { #run if both files exist
  edx<-readRDS("edx.rds") #read training set
  validation<-readRDS("validation.rds") #read validation set
  rm(test,train) #remove 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 time to years
```

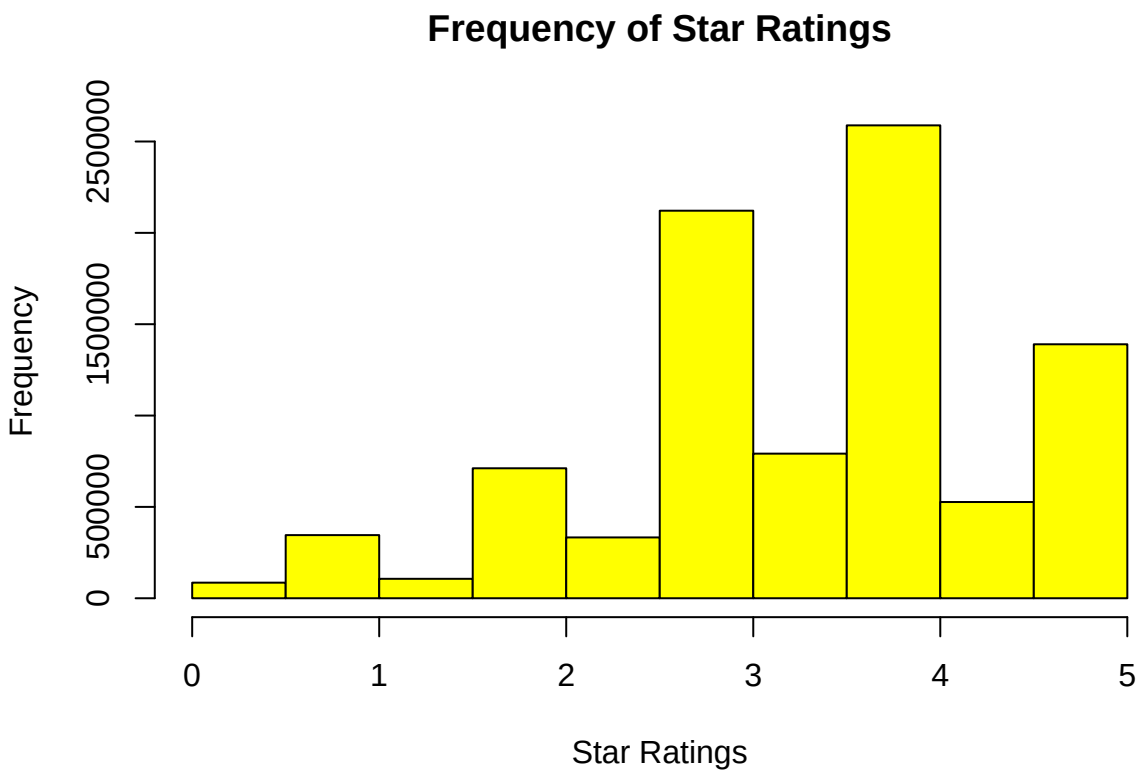
To start let's look at some basic information about the training data set. How large is it? How many unique movies are in it? How many unique users are in it? We can use the code below to examine these parts of the data.

```
edxdim<-dim(edx) #dimentions of the data set
NumMovies<-n_distinct(edx$movieId) #number of movies
NumUsers<-n_distinct(edx$userId) #number of users
```

It appears that the training set contains 9000055 reviews with 6 variables. There are 10677 different movies as well as 69878 unique users that provided ratings.

Now that we know a little about the data set itself, we can start looking at the ratings themselves. A good starting point is to find out what the average rating given is across the entire training set. The code below will build a histogram of the ratings in the training set.

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



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 ratings
```

The average rating across the entire training set is 3.5124652 stars. This number seems to confirm the plot shown above. We would expect the average to fall between the 3 and 4 star range.

Now that we know the average rating, we can start investigating the RMSE. First, we need to define the calculation of the RMSE. The calculation is going to be the difference created by subtracting the true rating from our predicted rating. The difference is then squared and averaged. This average is then rooted to return the RMSE. Now since we will be using this calculation for testing all of our methods we will save some time by building it as a user defined function in R.

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

Now that the function is ready, we can feed it our first model. This is going to be simply using the average of the training set as the predicted rating for all of the movies in the validation set.

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

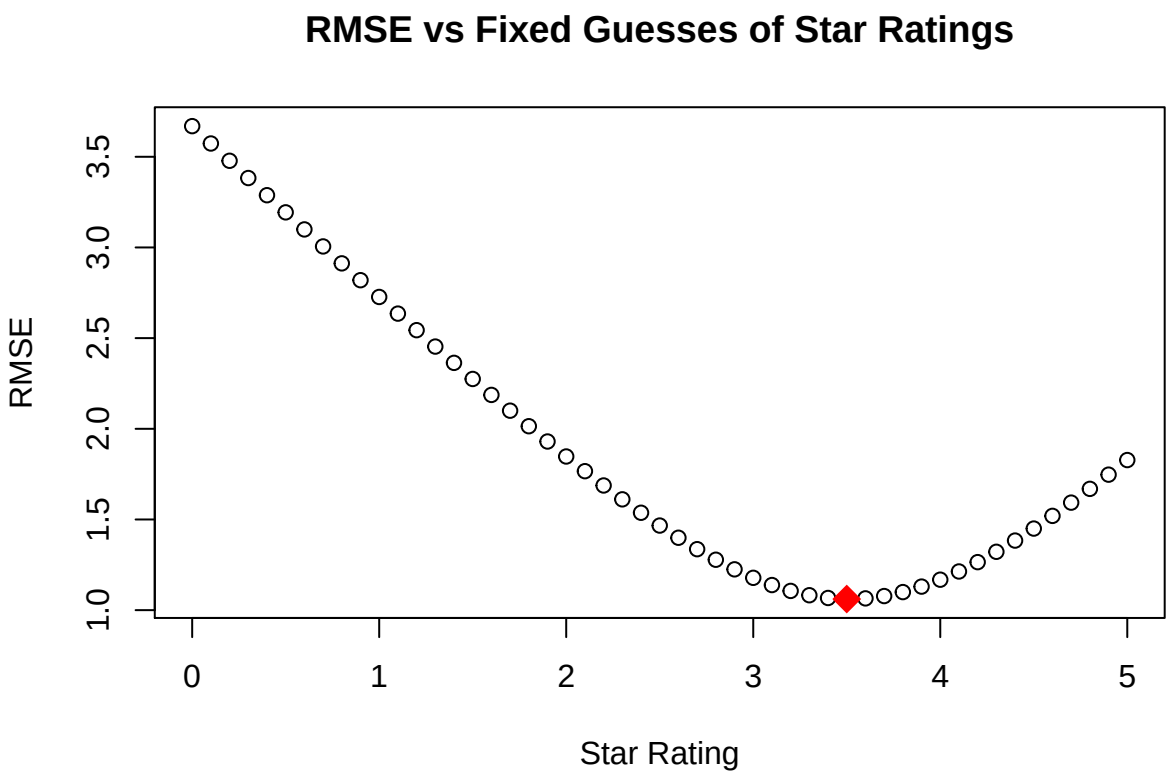
After running this code, we get an RMSE of 1.0612018 This is a good baseline to measure the rest of our models against, but it does not come close to our goal value of 0.87750.

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 for all of them and 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
```



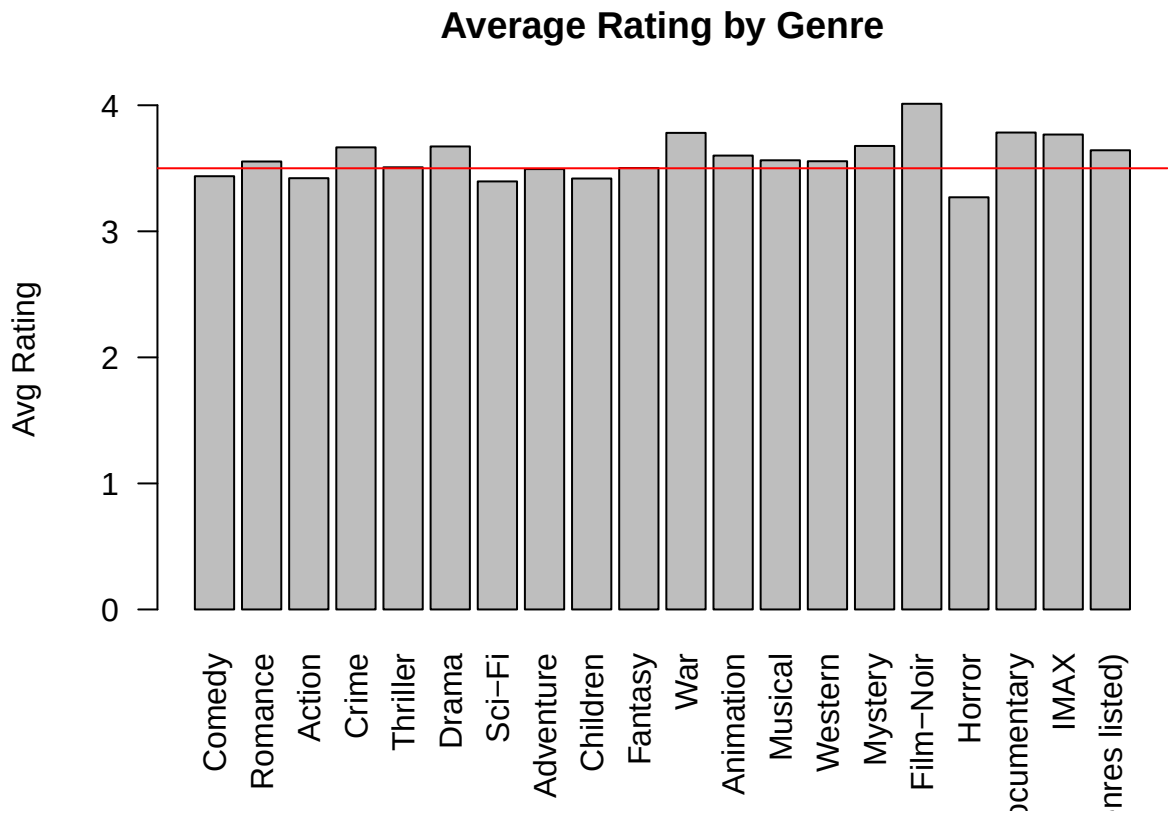
The plot shows that the star value of 3.5 gives us the lowest possible RMSE that can be achieved by guessing a fixed value. It also confirms our inital thought that using the average would provide the lowest RMSE.

If we can't just guess a set value that gives us the RMSE that we desire, we need to explore the features of the data. As a starting point we will take a look at the genres and see if there are any significant differences in rating between the genres. To do this we need to separate the genre tags into individual rows. The following code will separate these genre tags by the pipe delimiter. Again, this process is computationally intensive so the code will first check to see if the resulting data frame is available as an RDS before committing to rerunning the code.

```
genre<-file_test("-f","genres.rds") #checks data exists
if (genre == FALSE) {
  genres<-separate_rows(edx, genres, sep = "[|]") #df showing 1 genre per row
  rm(genre)
}else {
  genres<-readRDS("genres.rds") #read generated set
  rm(genre)
}
```

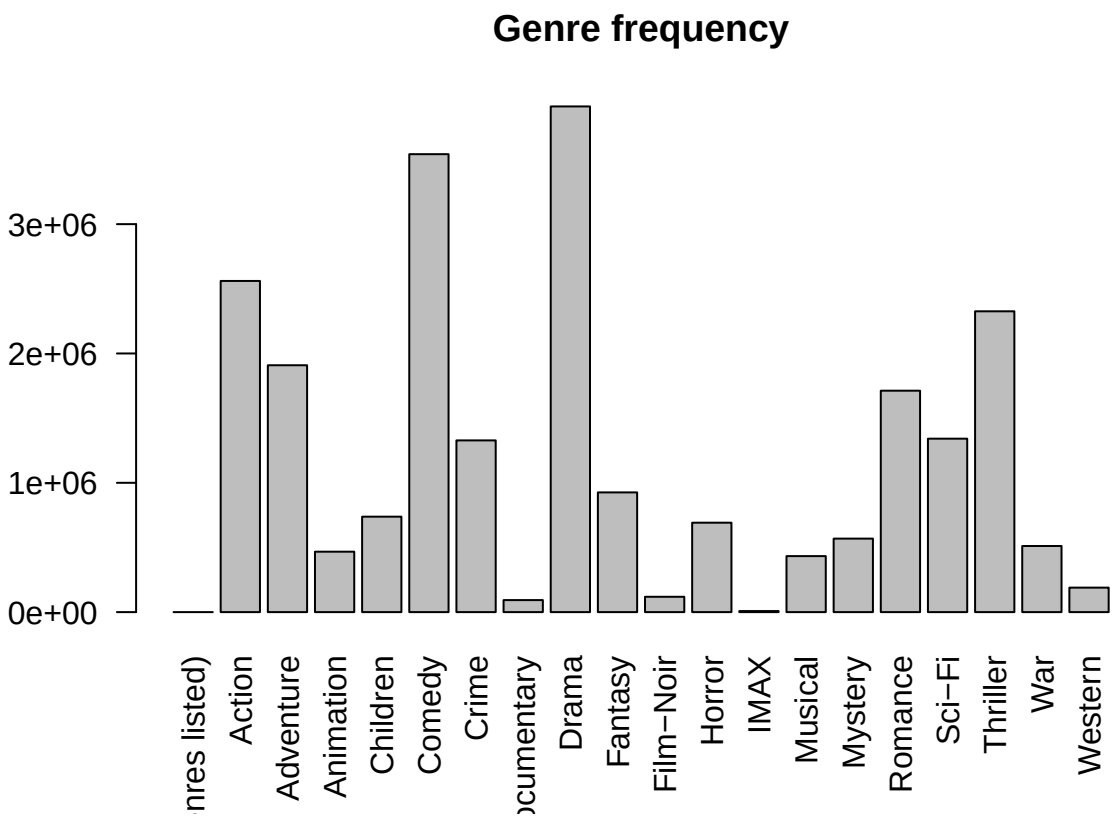
Now that we have separated the genres out, we can calculate the average for each genre. The code will create a data frame each genre that shows the name and the average star rating for that genre. Once all the averages are calculated it creates a bar chart showing the average for the different genres. It also adds 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")
```



The graph seems to indicate that the genre ratings center around the overall average. With none of them vary too much. The largest deviations are the genres of Film-Noir and Horror. If these genres make up a disproportionally large number of the reviews in the training set, then it might be worthwhile to adjust for genes. To find the number of movies that are in each genre 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)
```

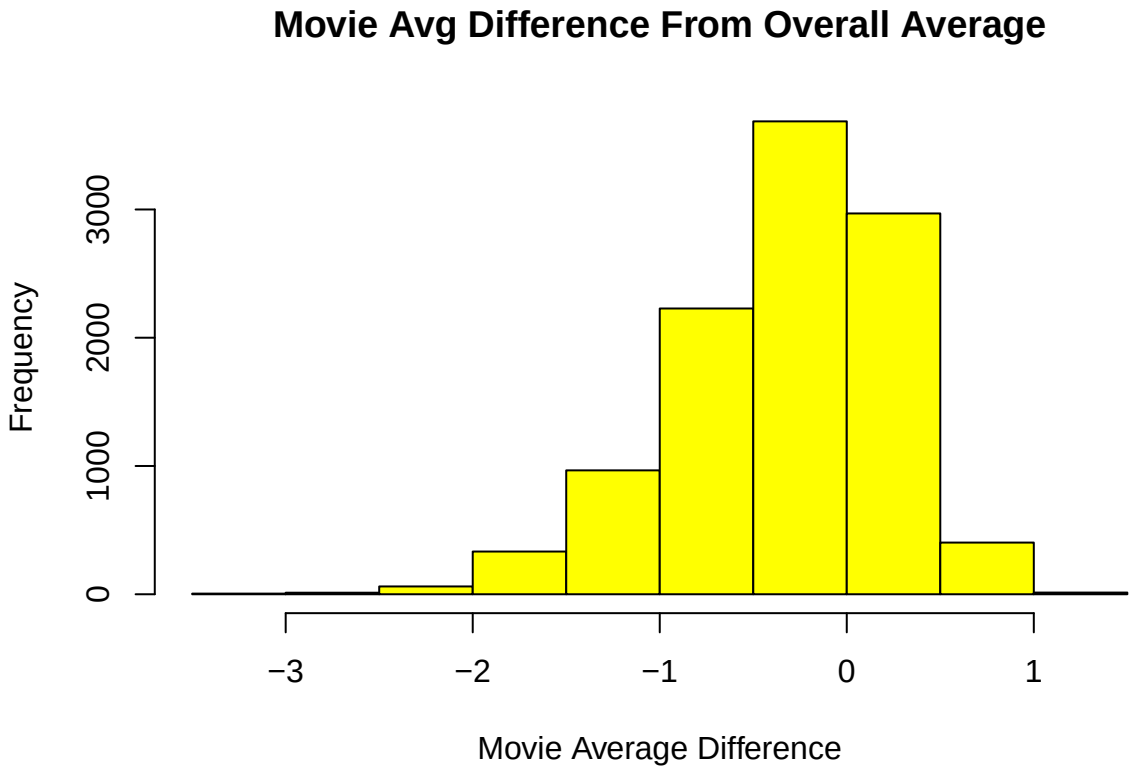


This plot shows that Film-Noir and Horror only make up a very small proportion of the data. So, it would not be much benefit to adjust for genres. This implies we should investigate other aspects of the data and other methods for making predictions.

Movie Effects

While the genre averages may not provide much insight, it could be that the information isn't granular enough. Here we are going to look at the movies themselves and see if individual movies deviate significantly from the average of 3.5 stars. Logically it would make sense that some movies are generally considered good and would be rated above the average. It also seems quite reasonable that there are moves which are generally considered bad and will be rated below the average. The following code will generate the averages for each indivigual movie. Then create a plot showing the severity and the frequency of the differences.

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



The graph shows that most movies fall within the 2.5 to 4.5 star range, with a majority of ratings falling below the average.

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 calculate our score.

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

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

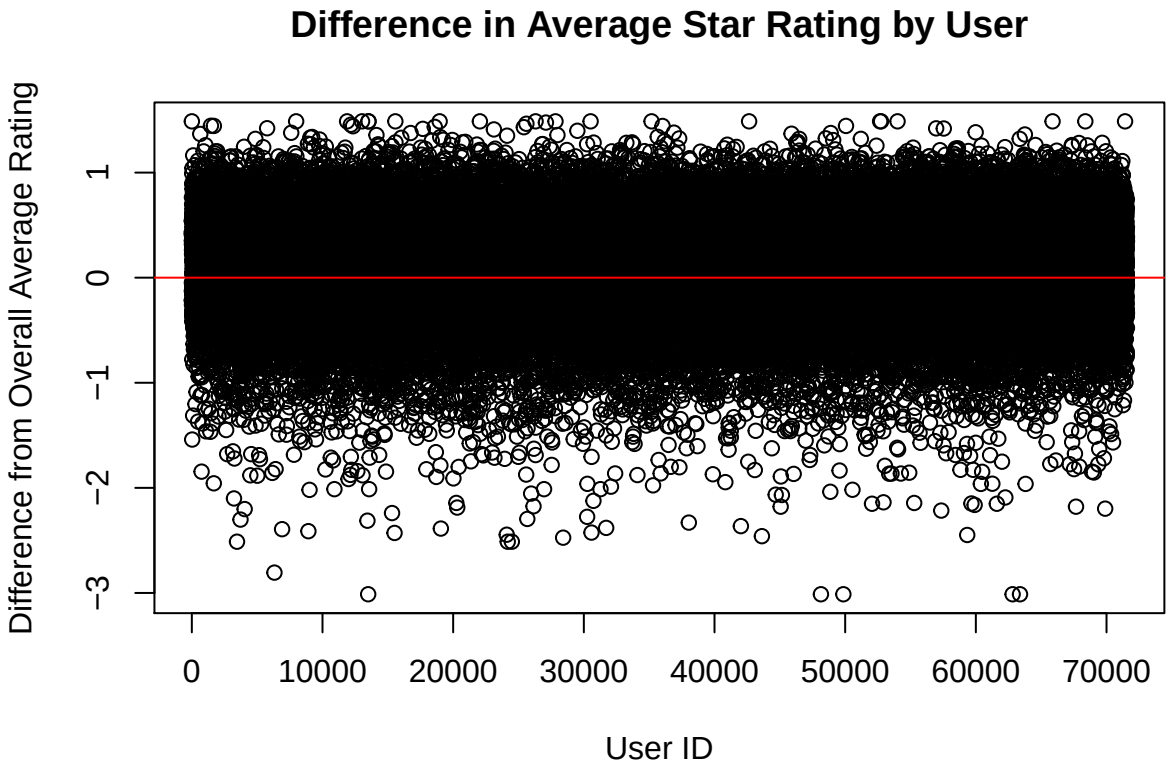
This model gives us an RMSE of 0.9439087 which is below our previous score of 1.0612018 where we simply guessed a fixed value for every movie. There are still improvments that need to be made if we intend to reach our goal of 0.87750 but we seem to be moving in the right direction again.

User Effects

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. Similar to how we built averages for each movie. We will now build averages for each individual user. It may be the case that some users score movies low over all and others score them high overall. 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 let's build a model that accounts for this user effect. 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 a users average rating gives and RMSE of 0.978336 which is also below the RMSE achieved by simply suing a fixed star value. It however is not as good as adjusting for the average score based on movie which gave an RMSE of 0.9439087.

Since both of these methods have given better results than just averaging across the data we can try combining them into one algorithm to see if together they 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 methods are combined, we get extremely close to our target. The resulting RMSE is 0.8850398 which is just 0.0075398 away from where we want to be. We just need a small push to improve our results.

Regularization

If we are confident that our previous methods are good what can we do to imporove them? It might be the case that a small number of ratings are pushing or pulling a prediction farther than it should. We can include a term in our model to adjust any large variations back down. Essentially any values that we are unsure of

due to low number of ratings get coerced back towards the mean. The following code builds the framework for including this penalty. It incoperates the previous mode and simply adds weights to any values that are too far out of line. Here we use three as a starting point.

```
lambda<-3
movieRegular <- edx %>% group_by(movieId) %>%
  summarize(movieRegular = sum(rating - trainMean)/(n()+lambda))
userRegular <- edx %>%
  left_join(movieRegular, by="movieId") %>% group_by(userId) %>%
  summarize(userRegular = sum(rating - movieRegular - trainMean)/(n()+lambda))
predicted_ratings <- validation %>%
  left_join(movieRegular, by = "movieId") %>%
  left_join(userRegular, by = "userId") %>%
  mutate(pred = trainMean + movieRegular + userRegular) %>% .$pred
regularized<-RMSE(predicted_ratings, validation$rating)
```

Adding this penalty to our model does improve our RMSE and lowers it to 0.86489 It appears that corralling stray results is a viable method for improving a model.

results

By calculating the average of the training set we were able to build a baseline RMSE. We were then able to leverage the average ratings that each individual movie receives to lower the RMSE. The same process was applied to the individual users to account for their preferences. We then employed regularization to adjust any results that had a low number of ratings back towards the mean. This gave us the final model seen in the analysis section which resulted in an RMSE of 0.86489 Which is comfortably below our stated goal of 0.87750. A Table summarizing the results is included below.

```
Results<-data.frame(Method=c("Overall Average","Movie Effects",
                             "User Effects","Combined Movie/User",
                             "Regularized Movie/User"),
                    RMSE=c(FixedAvg,movieRMSE,userRMSE,
                           combinRMSE,regularized))
Results
```

```
##                    Method      RMSE
## 1        Overall Average 1.0612018
## 2          Movie Effects 0.9439087
## 3           User Effects 0.9783360
## 4    Combined Movie/User 0.8850398
## 5 Regularized Movie/User 0.8648900
```

Conclusion

We tried several methods for minimzing the RMSE from simply using the average of all ratings to adjusting for movies generally getting higher or lower ratings. As well as adjusting for the fact that some users give higher or lower ratings overall. Finally we combined all of these and added a penalty for any predicted ratings that were uncertain. This was able to take us from an inital RMSE of 1.06 to an RMSE of 0.86489. There is more to explore in this data set and more advanced techniques but this is were we end our analysis.