Minimizing RMSE with MovieLens Dataset

Mark RIchards

4/30/2019

Introduction

This paper will outline the process of building a reccomendation 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 ratings. The goal is to reach a RMSE of 0.87750 or lower.

Overview

We will start by building a training and a validation set by partitioning the data. The training set will contain 90% of the available data while the validation set contains the remaining 10%. We will then perform some basic invesigation of the training set to see what information is available. from there we will start testing different alogrithms to find which methods perfom best.

It should be noted that much of the code will contain "If" statements that load results of calcualtions if the RDS file contianing them is avialable. This is due to limitations of computing power. The main code chunks are being run on a server and this report is being written on a laptop. These files will be made avaialbe on github (size limitations providing).

Analysis

We will start by loading the nessessary 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 valiation test sets. This code will check to see if the resulting data sets are available. If not it will downlaod the dataset and build the training and validations sets.

```
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 files do exist
  edx<-readRDS("edx.rds") #reads the generated edx training set
  validation<-readRDS("validation.rds") #reads the generated test set
  rm(test,train) #removes the file checks
}
```

For convieniance lets convert the date stamps of reviews to just the year

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

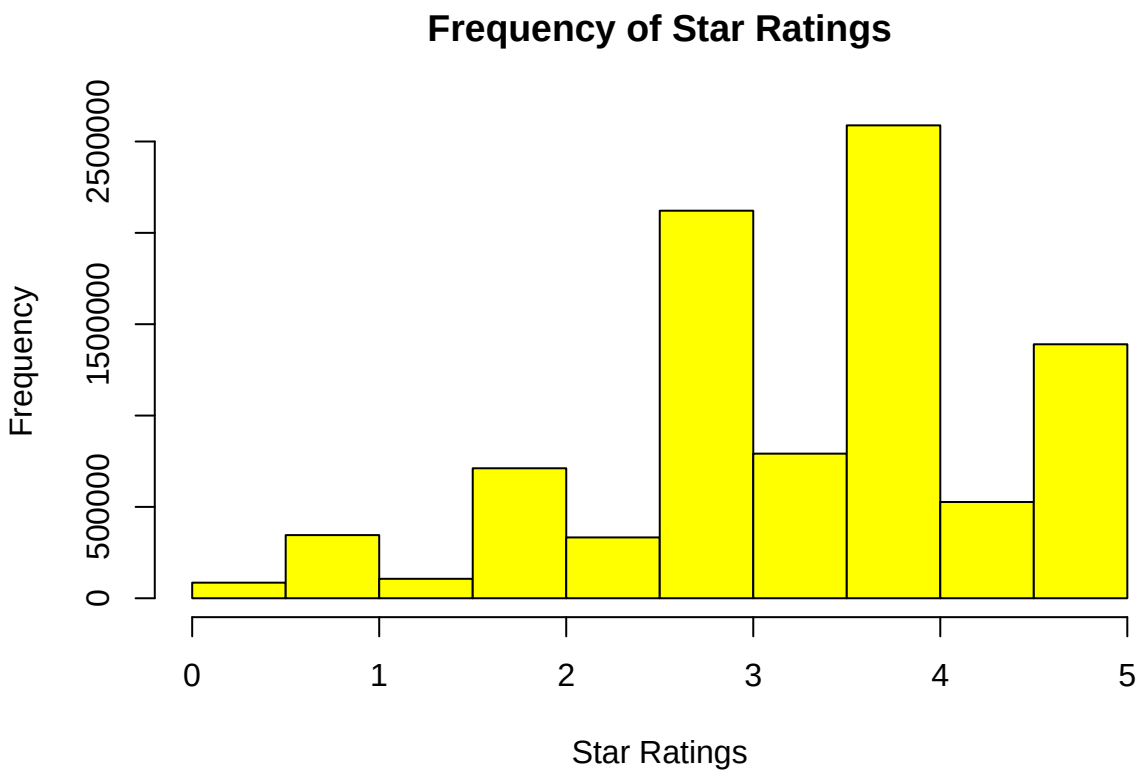
Now we can start looking at some basic information about the dataset. We might want to know how large it is. How many movies are in it, or how many users are in it.

```
print("Training set size: ")
dim(edx) #shows the dimentions of the data set
print(paste("Number of Movies in dataset: ", n_distinct(edx$movieId))) #number of movies
print(paste("Number of Users in dataset: ",n_distinct(edx$userId))) #numbr of users
```

From this we can see that the trainign set contains 9,000,055 reviews with 6 variables. Inside this dat there are 10,677 differnet movies that have reviews. There are also 69,878 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 follwing 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
```



Here 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.

```
mean(edx$rating)
```

Here we see that the average rating is 3.512 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))
}
```

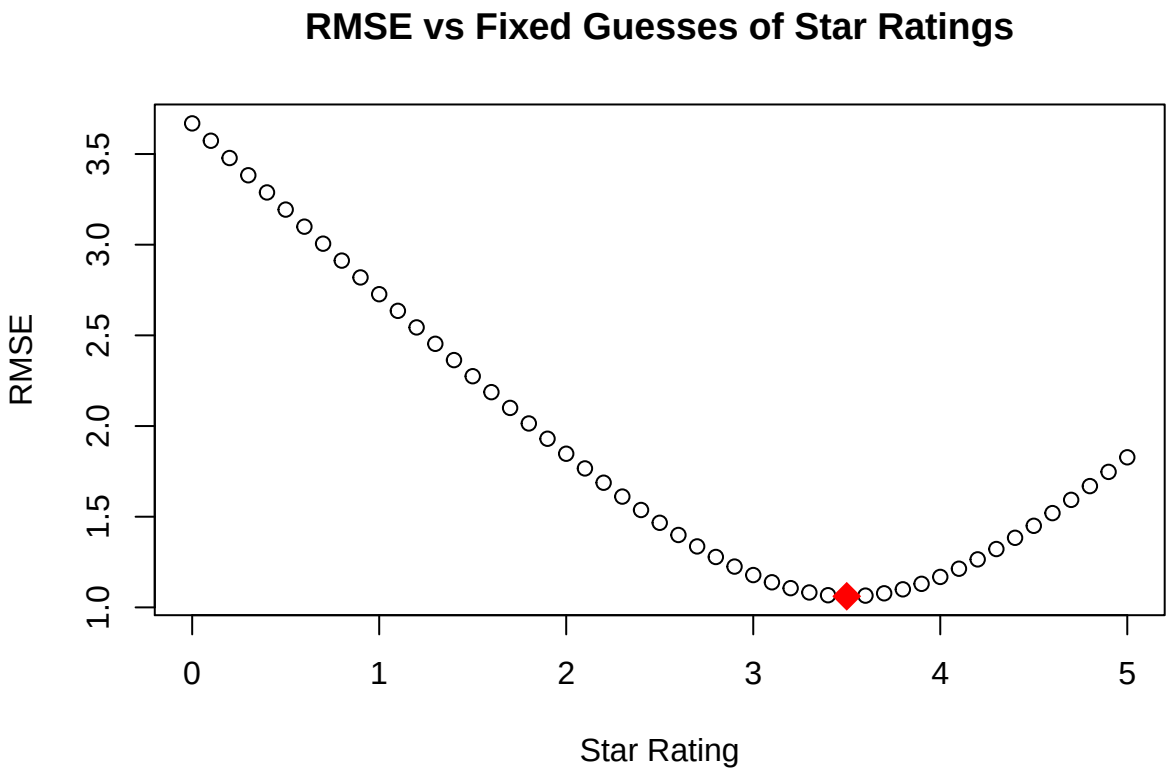
with this function in place we can now test our prediction of using the average star rating for all unknown ratings. To do this we set the true_rating for the RMSE funtion to the validation set ratings and our predicted rating to the average for the trainign set.

```
RMSE(validation$rating,mean(edx$rating))
```

The resulting RMSE is 1.0612. 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 set is to see if there is any fixed star rating value that can lower the RMSE. We woudl expect the average to be the lowest but to be sure we can use teh following code to check. It will build a list all values between 0 stars and 5 stars and calcualte the RMSE then plot the reults 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
```

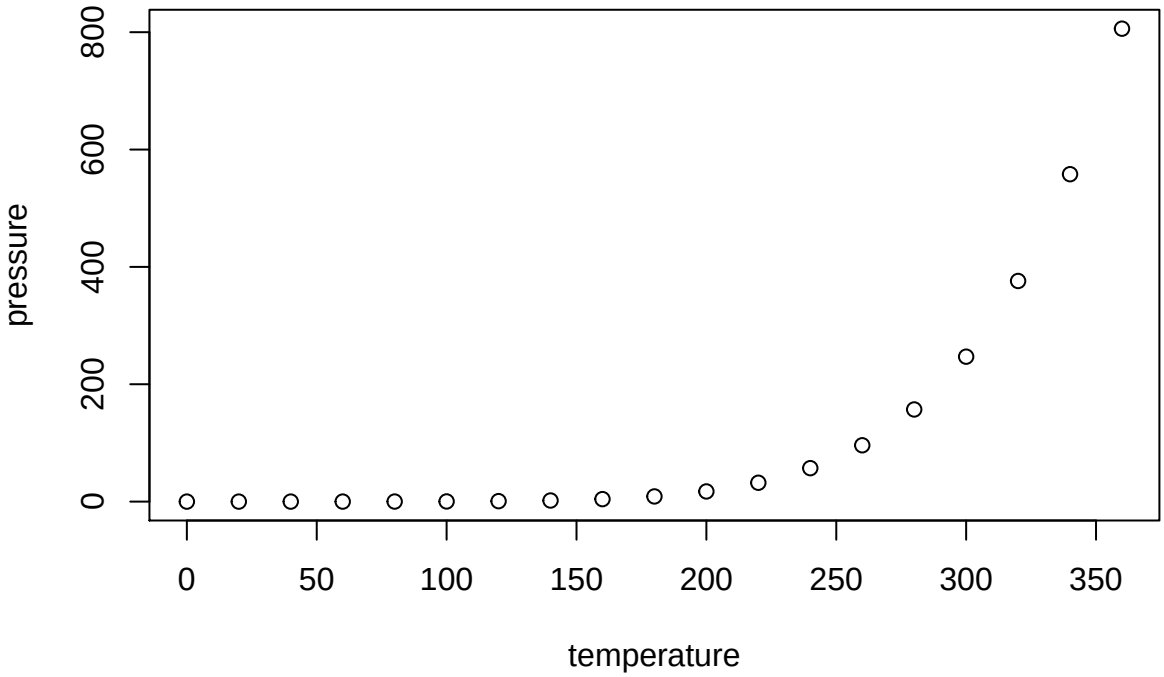
```
}
plot(x=seq(0,5,0.1), y=fixed,main = "RMSE vs Fixed Guesses of Star Ratings",
     ylab = "RMSE", xlab = "Star Rating") #build plot of results
points(y=min(fixed), x=(which.min(fixed)/10)-0.1,col="Red",
       pch=18,bg="Red", cex=2) #add red point at lowest RMSE value
```



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 implies we should investigate other aspects of the data and other methods for making predictions.

Including Plots

You can also embed plots, for example:



Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.