Movie Lens Report

By Philip J Brown - phil@pjb3.com

6/14/2019

Part 1) EXECUTIVE SUMMARY

HarvardX Data Science Capstone Class edx.org course: PH125.9x (2T2018)

- Student: Philip Brown
- email: Phil@pjb3.com
- github: https://github.com/pjbMit

MovieLens.org is a non-commercial website that helps users find movies that they may like. The data collected by this website is available on line to researchers for use in projects.

The data set used in this project is the MovieLens 10M Dataset which was released in 2009 and according to the MovieLens website contains 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users.

The goal of the project is to utilize data analysis and modelling skills to create a movie ratings prediction engine based on the data provided. Given the information about a user and a movie, the engine should try to predict what that user would rate a particular movie.

Optimize the model to produce a low the Root Mean Square Error (RMSE), which is defined as

$$RMSE = \sqrt{\Sigma_{i=1}^{n} \frac{(actual_i - predicted_i)^2}{N}}$$

For the highest grade, the model should produce a RMSE <= 0.87750.

After running the processing steps provided in the assignment and reviewing the data, I created a script in R Code to pre-process the data, then predict movie ratings using a basic naive model based on the the following prediction model:

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

with $\epsilon_{u,i}$ independent errors sampled from the same distribution centered at 0 and μ the "true" rating for all movies, which is estimated by

$$\hat{\mu} = mean(trainingDataRatings)$$

The next models built were a movie effect model, a movie effect plus user effect model, a regularized movie effect model, and then a regularized movie effect plus user effect model. With these models, **were are able to acheive the following RMSE values**:

method	RMSE
Just the average	1.0612018
Movie Effect Model	0.9439087
Movie + User Effects Model	0.8653488
Regularized Movie Effect Model	0.9438538
Regularized Movie + User Effect Model	0.8648170

We then went on to further explore the data, looking at variability in the predictors and trying different models and methods such as cross-validation to see if we could further improve the model, but found that none of the efforts improved upon the best result shown above.

In addition to this **executive summary**, this report also a **methods and analysis section**, a **results section** and a **conclusion section**.

Additionally, key files for this project have been uploaded and stored on my git hub page at github.com/pjbMit/movieLensProject. The three main files for this project are listed below, and can be viewed on git hub – The file names are also links:

- Movie_Lens_R_Script.Rmd
- Movie_Lens_Report.Rmd
- Movie_Lens_Report.pdf

Lastly, *you can view the R Script as an HTML page in a web browser by examining the following file – the name is also a link:*

- Movie_Lens_R_Script.nb.html

Part 2) METHODS AND ANALYSIS

The project was created in the RStudio environment using Rstudio Version 1.1.442 on a Macintosh; Intel Mac OS X 10_14_5

R version 3.5.1 (2018-07-02)
nickname Feather Spray

See the README.Rmd or README.html files for more information on the environment and setup. For questions, email me: pjbMit@pjb3.com , and I'll gladly respond promptly.

All code was written in R and executed in RStudio.

Here are the methods and techniques used.

- Data was downloaded as a zip file from the web, and read into a data frame in R.
- The data was separated into a testing and a training set.
- The data was processed and left-joins were used to ensure that all users and all movies in the test set were also in the training set.
- The movie data includes a ***genre*** field, which is a pipe ("|") delimited text field that that classifies each movie as belonging to one or more of a list of named genres. This field was parsed to create one column for each genere, with a value of 1 if the movie is part of that genre, and a value of 0 if the movie is not.
- Next we visualized some descriptive and summary information about the dataset.
- Next, the naive model, the movie effect model and the movie effect and user effect models were created and scored.
- We then create a Regularized Movie Effect Model, which was tuned to optimize the parameter lambda.
- We followed up with a Regularized Movie Effect model with user effect added in, which produced excellent results.
- After these models were evaluated, we looked at variability, and attempted to add genre to models using several standard models available through the **caret package** and applied techniques such as cross-validation. While we examined these models, and made multiple attempts to improve the results, none of the techniques tried improved upon the best results that were previously used.

– General approach:

For many approaches, I will first try on a very small data set, just to get the code working, then re-run on a medium sized data set, and then when I am satisfied, run on the full edx training set.

Similarly, initially I will NOT do full cross-validation, but once I have code working, I will try it. (Update – with 10 million rows, cross-validation takes too long to compute on my laptop.)

Additionally, being sensitive to computation times, I wrote code and used global variables to enable saving objects containing intermediate results as files on the local file system. By changing the values of these logial variable from TRUE to FALSE, or vice-versa, I am able to re-run code, skipping some of the lenghty processing steps and loading the correct results into objects in my environment by reading in pre-processed saved data from files on my local file system.

–set up

Set up libraries and enable multi-core processing for some of the operations used by the caret package. Because I have an 8 core processor, for calcuations that can utilize the parallel processing features, the script runs ***substantially** faster.

First set up a few logical variables in the global environment, then install packages and load libraries if and as needed.

The global variables serve as flags to skip portins of the code that are time-consuming, and to sometimes load saved objects from files instead of computing the result. This allows the script to be run to create a PDF report, or to be re-run without expedning the time needed for lengthy downloads or time consuming processing tasks.

```
#Also add another variable that lets me skip or alter how this code runs included in the report,
#as opposed to running in this script.

runningInScript <-TRUE # TRUE/FALSE value to skip portions of the script,or load from files if running
runningInScript <-FALSE  # uncomment this line when pasting this code to the Report Rmd file.

loadFromFile <- TRUE
#loadFromFile <- TRUE  ## Comment out this line if we don't want to reload the ojbects from files

load_edx_and_validation <- FALSE
#load_edx_and_validation <- TRUE ## Comment out this line if we don't want to reload the ojbects from f

# To make computations faster, when developing the code,
# set a flag to FALSE to make this script only use a small subset of the training data.
# Later, once everything works, re-run the code with this value set to TRUE to use the full training se

inDevelopment <- FALSE # TRUE/FALSE value to only use a subset of data during development
#inDevelopment <- TRUE  # To use the full training set, comment out this line and re-run the code.

myTrainFileName <- "~/movieLensMyTrain.rds"
myTestFileName  <- "~/movieLensMyTest.rds"
edxFile <- "~/edxFile.rds"
validationFile <- "~/validationFile.rds"
lambdasFile <- "movieLensLambdasFile.rds"
rmsesFile <- "movieLensRmsesFile.rds"
#fileName <- "movieLensSetup.rds"


# PJB - Changed repos to a parameter so that I can use a compatible repo on my mac.
isMac <- TRUE
#isMac <- FALSE  ##Set to false if you are NOT running on a Mac.

ifelse(isMac,repos <<- "https://cran.revolutionanalytics.com/" , repos <<- "http://cran.us.r-project.org
```

```
## [1] "https://cran.revolutionanalytics.com/"
```

```r
#Load libraries, installing as necessary
if(!require(tidyverse)) install.packages("tidyverse", repos = repos)
```

```
## Loading required package: tidyverse

## -- Attaching packages -----------------------------------------------------------------------------

## v ggplot2 3.1.1     v purrr   0.3.2
## v tibble  2.1.3     v dplyr   0.8.1
## v tidyr   0.8.3     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## Warning: package 'ggplot2' was built under R version 3.5.2

## Warning: package 'tibble' was built under R version 3.5.2

## Warning: package 'tidyr' was built under R version 3.5.2

## Warning: package 'purrr' was built under R version 3.5.2

## Warning: package 'dplyr' was built under R version 3.5.2

## Warning: package 'stringr' was built under R version 3.5.2

## Warning: package 'forcats' was built under R version 3.5.2

## -- Conflicts ----------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
if(!require(caret)) install.packages("caret", repos = repos)
```

```
## Loading required package: caret

## Warning: package 'caret' was built under R version 3.5.2

## Loading required package: lattice

##
## Attaching package: 'caret'

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

```r
if(!require(mlbench)) install.packages("mlbench", repos = repos)
```

```
## Loading required package: mlbench
```

```r
# Multicore processing package for caret.
if(!require(doMC)) install.packages("doMC", repos = repos)
```

```
## Loading required package: doMC

## Loading required package: foreach

##
## Attaching package: 'foreach'

## The following objects are masked from 'package:purrr':
##
##     accumulate, when

## Loading required package: iterators
## Loading required package: parallel
```

```
registerDoMC(cores=8)

#For ctree model.  See https://rpubs.com/chengjiun/52658
if(!require(party)) install.packages("party", repos = repos)
```

```
## Loading required package: party

## Warning: package 'party' was built under R version 3.5.2

## Loading required package: grid

## Loading required package: mvtnorm

## Warning: package 'mvtnorm' was built under R version 3.5.2

## Loading required package: modeltools

## Loading required package: stats4

## Loading required package: strucchange

## Loading required package: zoo

## Warning: package 'zoo' was built under R version 3.5.2

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

## Loading required package: sandwich

## Warning: package 'sandwich' was built under R version 3.5.2

##
## Attaching package: 'strucchange'

## The following object is masked from 'package:stringr':
##
##     boundary
```

```
if(!require(randomForest)) install.packages("randomForest", repos = repos)
```

```
## Loading required package: randomForest

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
if(!require(inTrees)) install.packages("inTrees", repos = repos) #Used by rfRules model
```

```
## Loading required package: inTrees
```

```r
if(!require(xgboost)) install.packages("xgboost", repos = repos)
```

```
## Loading required package: xgboost

## Warning: package 'xgboost' was built under R version 3.5.2

##
## Attaching package: 'xgboost'

## The following object is masked from 'package:dplyr':
##
##     slice
```

```r
if(!require(pROC)) install.packages("pROC", repos = repos)  #provides the roc function.
```

```
## Loading required package: pROC

## Warning: package 'pROC' was built under R version 3.5.2

## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
if(!require(matrixStats)) install.packages("matrixStats", repos = repos)  #provides the roc function.
```

```
## Loading required package: matrixStats

##
## Attaching package: 'matrixStats'

## The following object is masked from 'package:dplyr':
##
##     count
```

```r
#if(!require(Amelia)) install.packages("Amelia", repos = repos) #provides missmap visualization
if(!require(corrplot)) install.packages("corrplot", repos = repos)  #provides corrplot visualizarion
```

```
## Loading required package: corrplot

## corrplot 0.84 loaded
```

```r
#Function that if passed save==TRUE with save the object to the file.
#  otherwise returns without doing anything
#  This function was  created to save intermediate results to the file system,
#  because for some reason RStudio crashed repeatedly when I tried to run the whole script.
#  using code of the form:
#        objectName <- load(file)
#  if RStudio crashes, you can reload the objects already processed, and then
#  continue the scrit from there.
saveWork <- function(file,object,save){
    if(save){
        saveRDS(object,file)
    }
}
```

Data is downloaded from https://grouplens.org/datasets/movielens/10m/

and then a subset is selected by running this code which is given in the assignment's instructions:

Data setup

I needed to make two basic changes to the script that was provided: First) I needed to use a different repository to load the libraries, because repo specified didn't work on my mac. Second) This R Code crashed RStudio repeatedly, so I created a function and added code to save objects as files, so that I could save intermediate results into a file, and then reload them if and as necessary if R crashed. With these two work-arounds, I was able to load and process all of the data without incident.

```r
#
# Create edx set and validation set
#

##Start by defining two functions that can restore previously saved results from the local filesystem.


#Function to load the edx object from a saved file.
#This allows faster re-runs of the code, by avoiding downloading and re-processing
#the data through each development iteration.
restoreEdx <- function(loadFromFile){
    if(loadFromFile){
        print("loading edx object from filesystem.")
        edxFile <- "~/edxFile.rds"
        edx <<- readRDS(edxFile)
    }
}

#Function to load the validation object from a saved file.
#This allows faster re-runs of the code, by avoiding downloading and re-processing
#the data through each development iteration.
restoreValidation <- function(loadFromFile){
    if(loadFromFile){
        print("loading validation object from filesystem.")
        validationFile <- "~/validationFile.rds"
        validation <<- readRDS(validationFile)
    }
}


# Note: this process could take a couple of minutes
#Download the 10M row movie lens data set, and process the data.
#If we have already processed the data, then if the boolean flags are
#set appropriately, save time by skipping the download and data processing,
#and simply load pre-calculated results that were saved to the local file system.
if(runningInScript && loadFromFile == FALSE){
    #runningInScript
    print("runningInScript is TRUE, and loadFromFile is FALSE")
    # Create a bunch of file names in my directory to save intermediate results to files
    t1 <- "~/movieLens1.rds"
    t2 <- "~/movieLens2.rds"
    t3 <- "~/movieLens3.rds"
    t4 <- "~/movieLens4.rds"
    t5 <- "~/movieLens5.rds"
    t6 <- "~/movieLens6.rds"
```

```r
t7 <- "~/movieLens7.rds"
t8 <- "~/movieLens8.rds"
t9 <- "~/movieLens9.rds"
t10 <- "~/movieLens10.rds"


# 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)

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

## Set saveProgess to TRUE to save objects as intermediate results to files
## set it to FALSE not to save the intermediate results to files.
## This is used because RSTUDIO crashed repeatedly when I ran the script,
## so this allowed me to save intermediate results into files, and reload them later
## to continue processing.

saveProgress <- TRUE
saveProgress <- FALSE  # Set to false, and intermediate files WILL NOT be saved.

saveWork(t1,ratings,saveProgress)

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))
saveWork(t2,movies,saveProgress)

movielens <- left_join(ratings, movies, by = "movieId")
saveWork(t3,movielens,saveProgress)

# Validation set will be 10% of MovieLens data

set.seed(1) # if using R 3.6.0: set.seed(1, sample.kind = "Rounding")
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

saveWork(t4,test_index,saveProgress)
saveWork(t5,edx,saveProgress)
saveWork(t6,temp,saveProgress)

# 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")
    saveWork(t7,validation,saveProgress)

    # Add rows removed from validation set back into edx set

    removed <- anti_join(temp, validation)
    saveWork(t8,removed,saveProgress)

    edx <<- rbind(edx, removed)


    rm(dl, ratings, movies, test_index, temp, movielens, removed)

    #Save these two objects into files, so that I can easily
    #recreate the object using code similar to:
    #    objectName <- load(file)
    #without having to download and process the data again.

    saveProgress <- TRUE
    saveWork(edxFile,edx,saveProgress)
    saveWork(validationFile,validation,saveProgress)

    #Clean up my environment by removing old variables.
    rm(t1,t2,t3,t4,t5,t6,t7,t8,t9,t10)
}

if((runningInScript && loadFromFile == TRUE) || (!runningInScript)){
    restoreEdx(loadFromFile)
    restoreValidation(loadFromFile)
}
```

```
## [1] "loading edx object from filesystem."
## [1] "loading validation object from filesystem."
```

```
if(exists("edx")) glimpse(edx)
```

```
## Observations: 9,000,055
## Variables: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 37...
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5...
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 83898339...
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (19...
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|D...
```

```
if(exists("validation")) glimpse(validation)
```

```
## Observations: 999,999
## Variables: 6
## $ userId    <int> 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5...
## $ movieId   <dbl> 231, 480, 586, 151, 858, 1544, 590, 4995, 34, 432, 4...
## $ rating    <dbl> 5.0, 5.0, 5.0, 3.0, 2.0, 3.0, 3.5, 4.5, 5.0, 3.0, 3....
## $ timestamp <int> 838983392, 838983653, 838984068, 868246450, 86824564...
## $ title     <chr> "Dumb & Dumber (1994)", "Jurassic Park (1993)", "Hom...
## $ genres    <chr> "Comedy", "Action|Adventure|Sci-Fi|Thriller", "Child...
```

Now that the edx object and the validation object are both saved on the file system, so as a short-cut I can reload those objects from disk to run my code, rather than having to download and process the raw data multiple times during development.

```r
#Defines many of the functions that the script calls


#Now define some utility functions and functions used to preProcess the data.

# Function that calls createDataPartition to return a subset of the data frame.
# Created so that I can develop the code on a subset of the training data,
# so that the code will run faster during development
getSubset <- function(df,y,percent){
   #rows <- length(res[,1])
   # n <- as.integer(rows * percent)
   set.seed(1967) # For repeatability
   subset_index <- createDataPartition(y = y, times = 1, p = percent, list = FALSE)
   res <- df[subset_index,]
   res
}

# Function to process the data and return only the unique generes.
# Pares out the genere separator "|"
getAllGenres <- function(edxData) {
    edxData %>% separate_rows(genres, sep = "\\|") %>%
    group_by(genres) %>%
    summarize(count = n()) %>%
    arrange(desc(count)) %>%
    select(genres)
}

#Function extractGenresData is used to convert each genre into its own column
#The function accepts then name of a single movie genere, and returns a vector of 1's and 0's
#indicating whether or not that genere name is part of the genres string from that movie
extractGenresData <- function(oneGenre,genresByMovie){
    res <- grepl(oneGenre, genresByMovie$genres, fixed = TRUE)
    sapply(res, as.numeric) ## Convert all logical values to numeric
}

## Function myPreProcess preprocesses the data.
## It Accepts the training data frame, and returns a new data frame
## containing the preProcessed data
myPreProcess <- function (data) {

allGenres <- getAllGenres(data)   # gets all generes
allGenresStr <- allGenres$genres  # A vector with one entry for each unique genere
rm(allGenres)
genresByMovie <- data %>% select(genres) # string containing multiple generes with | separator
oneRow <- extractGenresData("Comedy",genresByMovie)  #Get one row, just to test "extracting" one genre
glimpse(oneRow)

# For each movie, Extract genres into T/F columns
genreDf <- as_data_frame(sapply(allGenresStr,extractGenresData,genresByMovie))
rm(allGenresStr)
dim(genreDf)  #Just to see the value

#Now convert certain fields to factors, since they are intended as labels, and not meaningful numeric v
factors <- data  %>%
transmute(userId=userId,
          movieId=movieId,
          titleId=as.factor(title),
          genres=as.factor(genres))
dim(factors) #just to see the value
#tmp  <- cbind(factors,genreDf,tibble(test=data$test))

# Get the movie data with factors and genres,
# and add the rating, which is the random variable we want to predict, as the last column
movieDataWithGenre <<- as_data_frame(cbind(factors,genreDf,tibble(test=data$test))) %>%
                    mutate(rating=data$rating)
#rm(tmp)
#as_data_frame(movieDataWithGenre)#And return the now preProcessed data frame that is the result.
movieDataWithGenre#And return the now preProcessed data frame that is the result.
}

#
# Function getData is used to run the code on a subset of the data
# during development, so that the code runs faster.
# This function will either return the data pased,
# or  a subset of the data based on the value of returnSubSet  second parameter
#
getData <- function (data,returnSubSet) {
 myData <- data
 if(returnSubSet)
   { #Only use a Subset if in development
    #percent <- 0.001
    percent <- 0.01
    dataSubset <- getSubset(data,edx$rating,percent)
    myData <- dataSubset

    }
myData  #Return the data
}

#Function calculates RMSE, substituting zero as the error for NA rows
myRMSE <- function(true_ratings, predicted_ratings){
    diff <- true_ratings - predicted_ratings
    #Replace NA with zero
    diff[is.na(diff)]<-0
    sqrt(mean((diff)^2))
  }
```

PreProcess the data. During development, only preprocess a subset of the data so that code runs quickly. Run on the full set of data once development is completed, and it works.

```
##Preprocess and get either the training data, or a subset it, depending on the value of inDevlopment.
## If certain booleans are set, then load myTest and myTrain from pre-saved files those objects don't a
print("PreProcessing")
```

```
## [1] "PreProcessing"
```

```
if(runningInScript) {
    #RunningInSCript
    print("RunningInScript")
    data <- getData(edx,inDevelopment)  # Get the training data, or a subset of it.
    glimpse(data)

    d1 <- data %>% mutate(test=FALSE)
    d2 <- validation %>% mutate(test=TRUE)
    data <- rbind(d1,d2)

    print("Starting to pre-process data...")
    movieDataWithGenre <<-myPreProcess(data) # Preprocess and get subsets of the data
    dim(as.matrix(movieDataWithGenre))

    print("pre-processing completed.")
    glimpse(movieDataWithGenre)
    myTrain <<- movieDataWithGenre %>% filter(test==FALSE) %>% select (-test,-genres)
    myTest  <<- movieDataWithGenre %>% filter(test==TRUE)  %>% select (-test,-genres)

    ## Save processed train and test data to files,
    ## so that they can be reloaded later, instead of created from scratch.
    saveWork(myTrainFileName,myTrain,TRUE)
    saveWork(myTestFileName,myTest,TRUE)

    rm(data,d1,d2,movieDataWithGenre)
}

if(!runningInScript){
    #!runningInScript
    # print("NOT RunningInScript")
    #Load myTrain object from a file if boolean flags are set, and if the object doesn't already exist.
    #Load myTest  object from a file if boolean flags are set, and if the object doesn't already exist.
    if(loadFromFile){
        # Note use of special assignment operator which scopes the variable in the global environment.
        if(! exists("myTrain")) {
            print("Loading myTrain from file.")
            myTrain <<- readRDS(myTrainFileName)
        }
        if(! exists("myTest")){
            print("Loading myTest from file.")
            myTest  <<- readRDS(myTestFileName)
        }
    }
}
```

```
## [1] "Loading myTrain from file."
## [1] "Loading myTest from file."
```

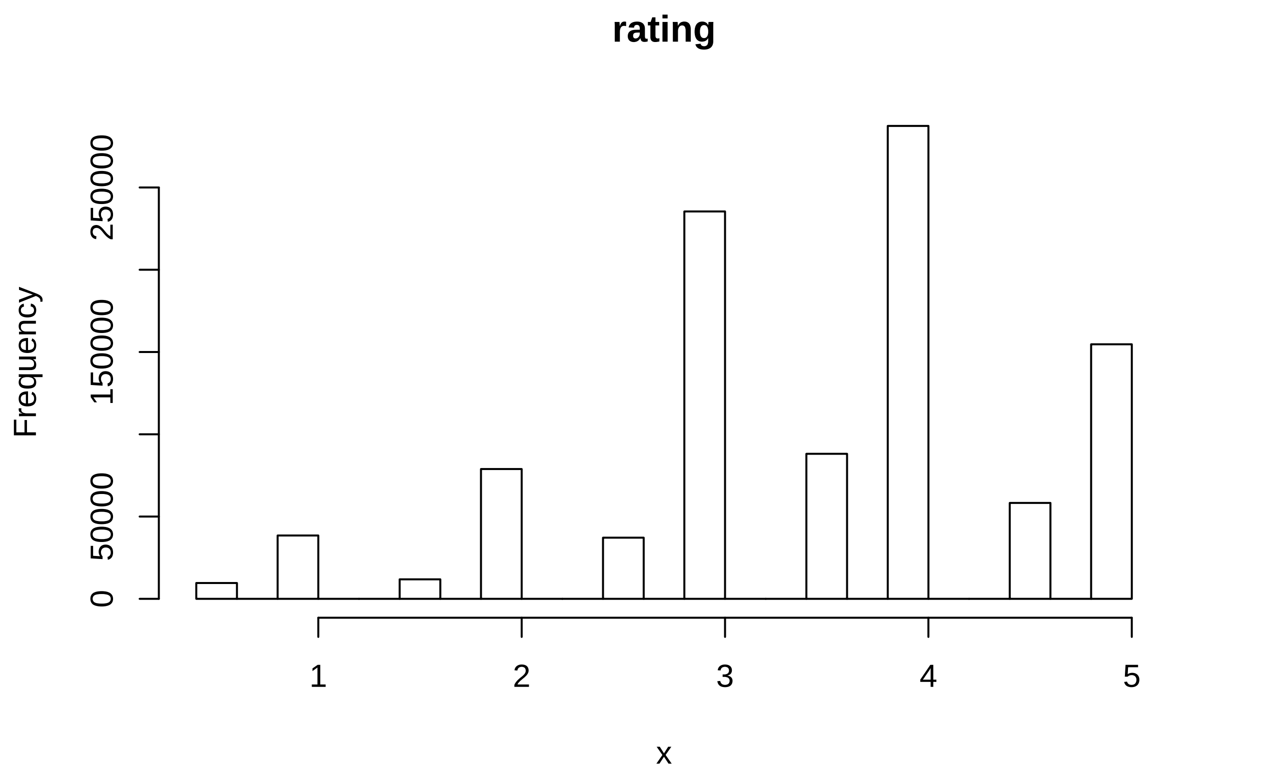
Here we show a few summary visualizations of the data.

```
#cor(myTrain[,6],myTrain[,2])

#Histogram of ratings
x <- myTest[,"rating"]$rating
```

```r
n <-"rating" #names(myTest)[24]
is.numeric(x)
```

```
## [1] TRUE
```

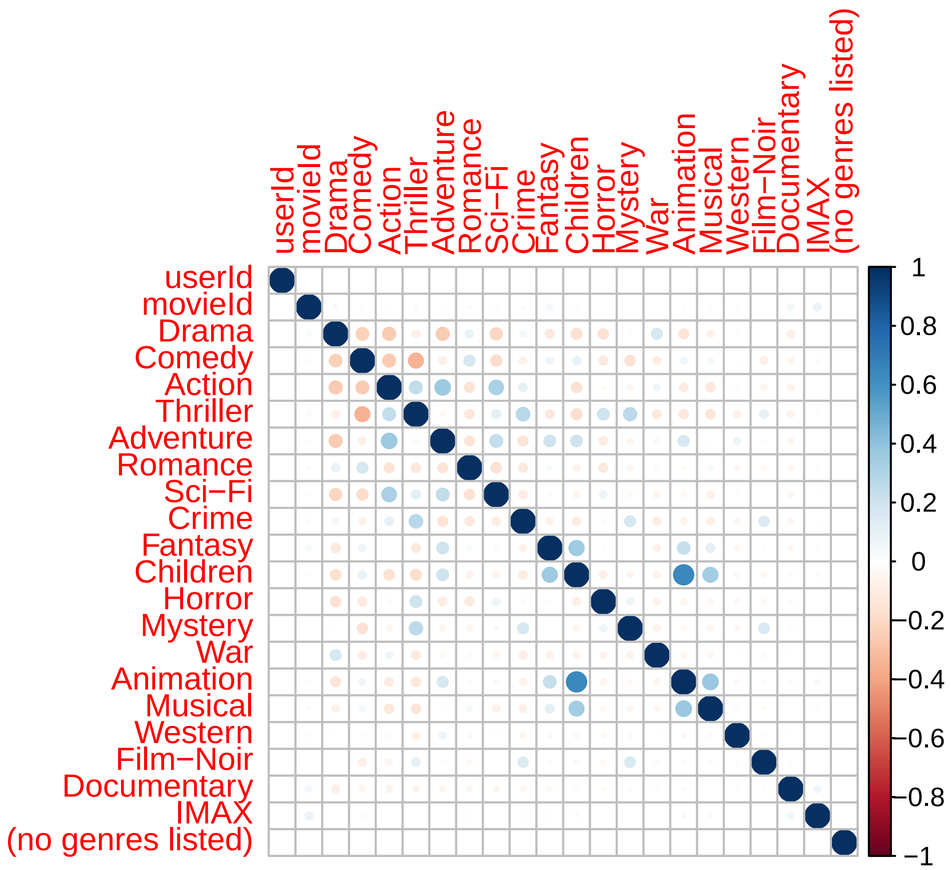
```r
hist(x, main=n)
```



```r
#Note: using myTest for the histogram instead of myTrain,
#because this data set is 1/10th as big, and for the purposes of
#histogram trends should have a very similar pattern to the Test data set which has 9 million observati

##Shows a lot of 4's, and not many half-star ratings.
```

This histogram shows that 4 stars is the most popular rating, followed by 3 stars. It also shows that half-star ratings are less common than whole-star ratings.

```r
#Correlations
t <- myTrain %>% select (-titleId)
correlations <- cor(t[,1:22]) #calculate correlations
corrplot(correlations, method="circle") # create correlation plot
```



```
##Shows that there are a few popular movie groups
## such as Children Animation, and Action Adventure.
## Also shows that you rarely see movies such as Comedy Thrillers, or Action Dramas
```

This correlation plot shows that certain genre's have a positive correlation with each other, while other's have a negative correlation. This shows that Shows that there are a few popular movie groups such as Children Animation, and Action Adventure, where the two genre's are frequently both assiged to the same movie. The negative correlations show that you rarely see movies on certain combinations of genres such as Comedy Thrillers, or Action Dramas.

Next we move on to building a predictive model. First look at the most naive approach… and notice that 3.51 is the mu_hat that has the lowest RMSE.

```
# Using model Y_u_i = U + E_u_i
mu_hat <- mean(myTrain$rating)
mu_hat
```

```
## [1] 3.512465
```

```
naive_rmse <- myRMSE(myTest$rating, mu_hat)

#Now create a results table, so that we can compare different approaches
rmse_results <- tibble(method = "Just the average", RMSE = naive_rmse)
rmse_results
```

```
## # A tibble: 1 x 2
##   method            RMSE
##   <chr>            <dbl>
## 1 Just the average  1.06
```

```r
# See how any other value increase or RMSE
#errors<-sapply(seq(2.5,4,.01),function(mu_hat) RMSE(myTest$rating, mu_hat))
#plot(seq(2.5,4,.01),errors)

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

#movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = ., color = I("black"))

predicted_ratings <- mu + myTest %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
predicted_ratings[is.na(predicted_ratings)]<-mu  ## use the mu_hat for any NA values
#predicted_ratings <- tibble(rating=predicted_ratings) ## Get predicted ratings

model_1_rmse <- myRMSE(myTest$rating, predicted_ratings)

rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie Effect Model",
                                 RMSE = model_1_rmse))
rmse_results
```

```
## # A tibble: 2 x 2
##   method              RMSE
##   <chr>              <dbl>
## 1 Just the average   1.06
## 2 Movie Effect Model 0.944
```

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

sum(user_avgs$b_u[is.na(user_avgs$b_u)])  #Check for NA's
```

```
## [1] 0
```

```r
predicted_ratings <- myTest %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

model_2_rmse <- myRMSE(myTest$rating,predicted_ratings)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie + User Effects Model",
                                 RMSE = model_2_rmse))

#Now We'll add regularization
lambda <- 3
mu <- mean(myTrain$rating)

movie_reg_avgs <- myTrain %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())

predicted_ratings <- myTest %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)

model_3_rmse <- myRMSE(myTest$rating,predicted_ratings)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Regularized Movie Effect Model",
                                     RMSE = model_3_rmse))
rm(edx,validation,user_avgs,movie_avgs,movie_reg_avgs) #free memory
rmse_results
```

```
## # A tibble: 4 x 2
##   method                          RMSE
##   <chr>                          <dbl>
## 1 Just the average               1.06
## 2 Movie Effect Model             0.944
## 3 Movie + User Effects Model     0.865
## 4 Regularized Movie Effect Model 0.944
```

Part 3) RESULTS

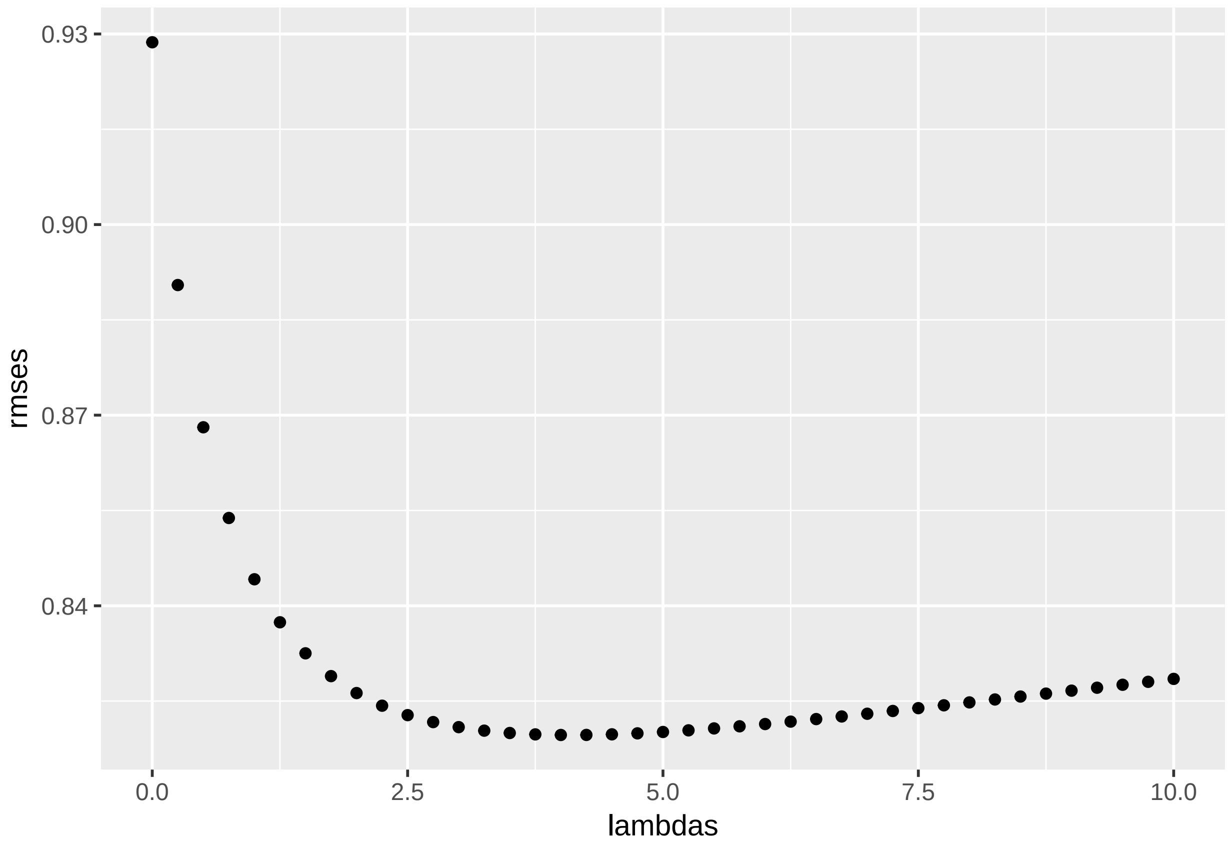
Here's a link back to the Summary section of this script.

The model results were promising, as can be seen by the output from rmse_results.

Next we ran the code below to tune the lambda parameter used by regularizatio model.

(See the output below which shows best results obtained.)

```
# The estimates that minimize this can be found similarly to what we did above.
# Here we use cross-validation to pick a  lambda
if(runningInScript){
    #RunningInScript
    print("RunningInScript is TRUE")
    lambdas <<- seq(0, 10, 0.25)

    rmses <<- sapply(lambdas, function(l){
          mu <<- mean(myTrain$rating)

          b_i <<- myTrain %>%
            group_by(movieId) %>%
            summarize(b_i = sum(rating - mu)/(n()+l))

          b_u <<- myTrain %>%
            left_join(b_i, by="movieId") %>%
            group_by(userId) %>%
            summarize(b_u = sum(rating - b_i - mu)/(n()+l))

          predicted_ratings <<-
            myTest %>%
            left_join(b_i, by = "movieId") %>%
            left_join(b_u, by = "userId") %>%
            mutate(pred = mu + b_i + b_u) %>%
            pull(pred)

          sum(is.na(predicted_ratings))
          rmse <- myRMSE(myTest$rating,predicted_ratings)
          return(rmse)
        })

    ## Save processed train and test data to files,
    ## so that they can be reloaded later, instead of created from scratch.
    saveWork(lambdasFile,lambdas,TRUE)
    saveWork(rmsesFile,rmses,TRUE)
}

if(!runningInScript){
    lambdas <<- readRDS(lambdasFile)
    rmses <<- readRDS(rmsesFile)
}

qplot(lambdas, rmses)
```



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

```
## [1] 4
```

```
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Regularized Movie + User Effect Model",
                                 RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0612018
Movie Effect Model	0.9439087
Movie + User Effects Model	0.8653488
Regularized Movie Effect Model	0.9438538
Regularized Movie + User Effect Model	0.8196661

As the resuts show, we achieved satisfactory results from the tuned regularized movie effect model with user affect added. At this point, with a best RMSE value of 0.8648170, we have achieved our goal.

The code and comments below are to show further exploration and further analysis that was done on the data to try to improve the result. After substantial analysis, and trying multiple techniques, including attempting to apply the genre data to improve results, I determined that the tuned regularized movie effect with user effect model produced the best results.

Nonethe less, for the purposes of documenting the efforts, and demonstrating some the data visualation techniques, I have included the analysis and selected outputs below.

Additional examination, and trying different models.

```
#Perform additional analysis & testing to learn about the data
#and to see if certain approaches improve model performance.

#Determine factors that contribute the most variablity, and show a heatmap of a random sample of rows,
#from highest variable factor to least variable.

    #runningInScript
    print("runningInScript is TRUE")
```

```
## [1] "runningInScript is TRUE"
```

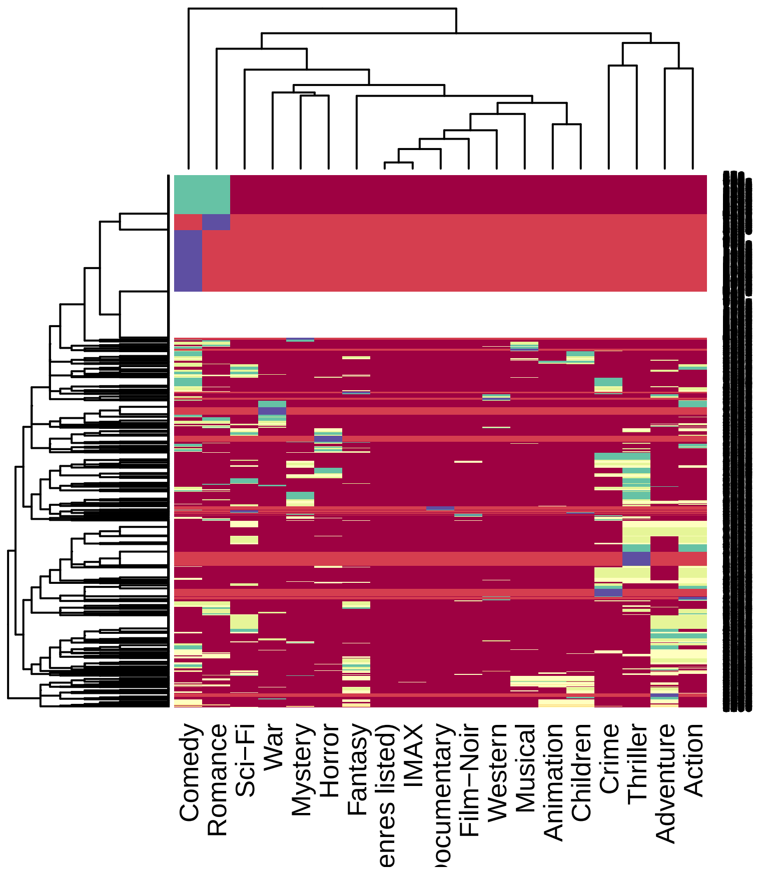
```
    x<-as.matrix(myTrain[5:23])
    sds <- colSds(x, na.rm = TRUE)
    o <- order(sds, decreasing = TRUE)[1:19]
    dim(x)
```

```
## [1] 9000055      19
```

```
    #Draw a heatmap, using a subset of the data
    x2 <- as.matrix(getSubset(x[,o],myTrain$rating,0.0005))
    dim(x2)
```

```
## [1] 4502   19
```

```
    sds2 <- colSds(x2, na.rm = TRUE)
    o2 <- order(sds2, decreasing = TRUE)[1:19]
    heatmap(x2[,o2], col = RColorBrewer::brewer.pal(11, "Spectral"))
```



The heat map above shows that most of the variation in the ratings is within the five genres of Romance, Comedy, Thriller, Horror and Sci-Fi.

```
o2
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 14 13 15 16 17 18 19
```

```
sds2
```

```
##  [1] 0.48801459 0.44733853 0.43603318 0.40681148 0.39846974 0.35419086
##  [7] 0.35083837 0.29878525 0.26919299 0.25919809 0.25361899 0.24032336
## [13] 0.21137968 0.21374487 0.14521169 0.11084637 0.08907449 0.02979768
## [19] 0.00000000
```

```
apply(x2,MARGIN=2,mean)
```

```
##            Comedy            Action          Thriller
##       0.390937361       0.276543758       0.255219902
##         Adventure           Romance            Sci-Fi
##       0.209240338       0.197912039       0.147045757
##             Crime           Fantasy          Children
##       0.143713905       0.099067081       0.078631719
##            Horror           Mystery               War
##       0.072412261       0.069080409       0.061528210
##         Animation           Musical           Western
##       0.046868059       0.047978676       0.021545980
##         Film-Noir       Documentary              IMAX
##       0.012438916       0.007996446       0.000888494
## (no genres listed)
##       0.000000000
```

```r
colnames(x2)
```

```
##  [1] "Comedy"             "Action"             "Thriller"
##  [4] "Adventure"          "Romance"            "Sci-Fi"
##  [7] "Crime"              "Fantasy"            "Children"
## [10] "Horror"             "Mystery"            "War"
## [13] "Animation"          "Musical"            "Western"
## [16] "Film-Noir"          "Documentary"        "IMAX"
## [19] "(no genres listed)"
```

```r
top <- x2[1,2:5]
cols <- names(top)
#These are the attributes that add the most variability to the data
cols
```

```
## [1] "Action"    "Thriller"  "Adventure" "Romance"
```

```r
colNames <- names(myTrain)[1:7]

#ReOrder columns, removing titleID, and putting ratings as the first column
tmp2 = names(myTrain)[1]
tmp3 = names(myTrain)[2]
colNames[1] <- "rating"  #Remove titleId, and replace it with "rating"
colNames[2] <- tmp2
colNames[3] <- tmp3

#Show order of columns
names(myTrain[colNames])
```

```
## [1] "rating"   "userId"   "movieId"  "Drama"    "Comedy"   "Action"
## [7] "Thriller"
```

```r
# See list of models available in caret
#names(getModelInfo())

# prepare training scheme  ##number and repeats set lower for faster executiion

#Alter myTrain to have fewer rows because training on 10MM rows takes wayyy too long!
myTrainSubset <- getSubset(myTrain[colNames],myTrain$rating,0.01)
#trainControl <- trainControl(method="repeatedcv", number=5, repeats=2)

trainControl <- trainControl(method="LGOCV",
                             number=4,
                             returnData = FALSE,
                             trim = TRUE,
                             allowParallel = TRUE
                                 )

metric <- "RMSE"

# CART
set.seed(7)
fit.model1 <- train(rating ~ ., data =myTrainSubset, method="rpart", metric = metric, maximize = FA
    trControl=trainControl)
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info =
## trainInfo, : There were missing values in resampled performance measures.
```

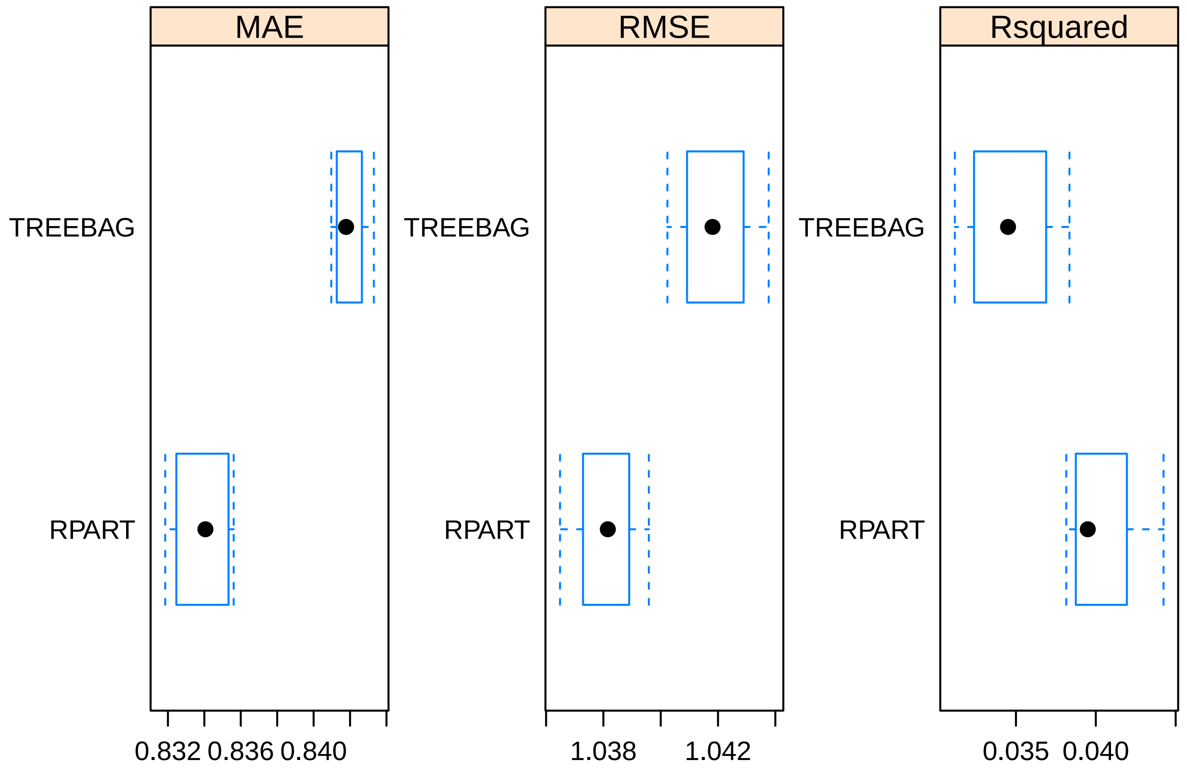
```
# Second Model
set.seed(7)
fit.model2 <- train(rating~., data = myTrainSubset,
                    method = "treebag",
                    metric = metric,
                    maximize = FALSE,
                    na.action = na.omit,
                    trControl=trainControl)

# collect resamples
results <<- resamples(list(RPART=fit.model1, TREEBAG=fit.model2))

# summarize differences between models
summary(results)
```

```
##
## Call:
## summary.resamples(object = results)
##
## Models: RPART, TREEBAG
## Number of resamples: 4
##
## MAE
##              Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## RPART   0.8318518 0.8327679 0.8340585 0.8338949 0.8351856 0.8356108    0
## TREEBAG 0.8409684 0.8414214 0.8417816 0.8419602 0.8423204 0.8433090    0
##
## RMSE
##             Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## RPART   1.036487 1.037691 1.038154 1.038095 1.038559 1.039586    0
## TREEBAG 1.040234 1.041260 1.041809 1.041905 1.042455 1.043769    0
##
## Rsquared
##               Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
## RPART   0.03815035 0.03904558 0.03949655 0.04034593 0.04079690 0.04424028
## TREEBAG 0.03117708 0.03297833 0.03450554 0.03463423 0.03616144 0.03834877
##         NA's
## RPART      0
## TREEBAG    0
```

```
# box and whisker plots to compare models
par(1,2)
```

```
## [[1]]
## NULL
##
## [[2]]
## NULL
```

```
scales <<- list(x=list(relation="free"), y=list(relation="free"))
bwplot(results, scales=scales)
```



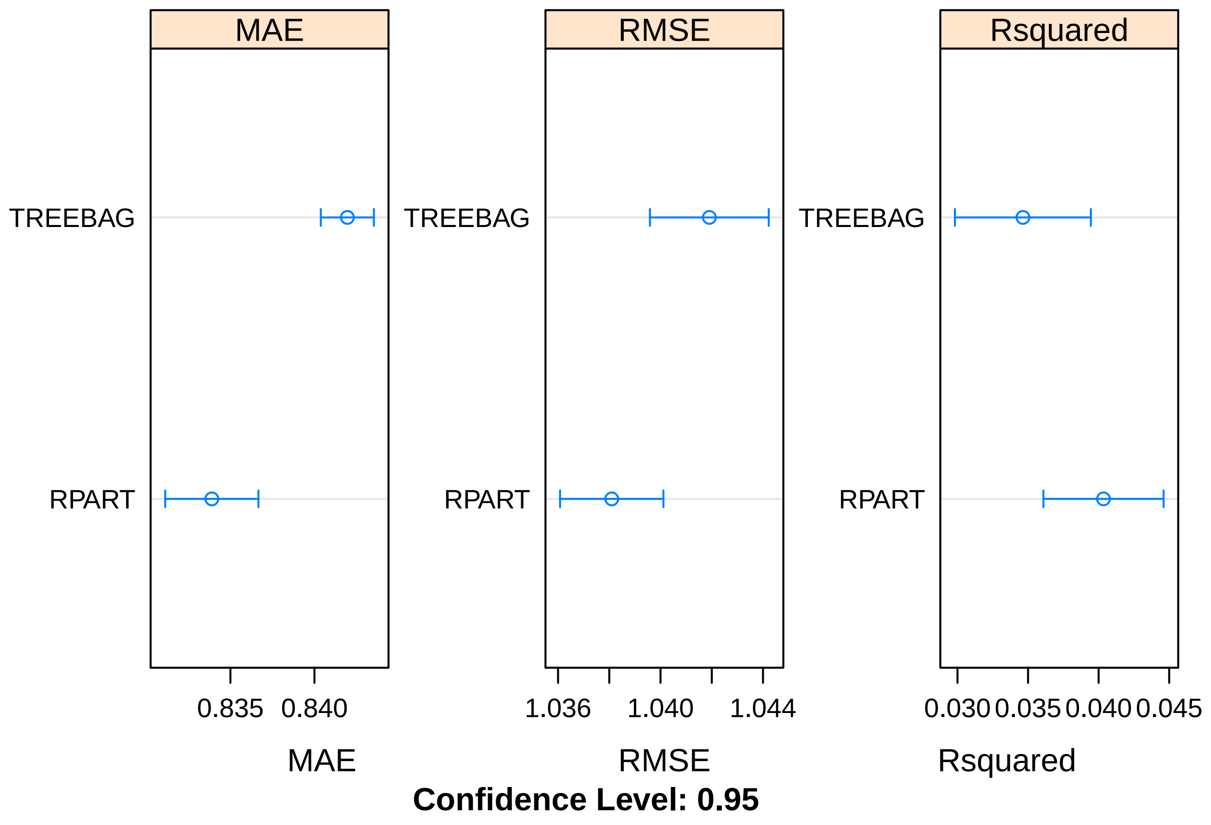
```r
    # print("second chart")
    # # dot plots of accuracy
    # #scales <- list(x=list(relation="free"), y=list(relation="free"))
    # dotplot(results, scales=scales)
    #
    # pairwise scatter plots of predictions to compare models
   # splom(results)
```

The Box and Whisker plot above shows that RPART algorithm tends to make more precise, yet less accurate predictions when compared to the TREEBAG algorith.

```r
print("second chart")
```

```
## [1] "second chart"
```

```r
    # dot plots of accuracy
    #scales <- list(x=list(relation="free"), y=list(relation="free"))
    dotplot(results, scales=scales)
```



While the dot plot above clearly shows that TREEBAG algorithm has greater variability in the residual values than the RPART algorithm.

Part 4) CONCLUSION

Processing 10 million rows of data on a personal laptop computer can be challenging. The realities of memory usage and processing speed were very real. For example, commonly used linearized regression models were too slow and required too many calcuations to be practical in this environment.

I discovered a library and options to set to enable multi-core parallel processing for some of the algorithms in the **caret** package, and this technique helped tremendously, as I was able to span 8 R-sessions that ran in parallel to process some of the algorithms.

Ultimately, the best results that I obtained were a **RMSE of 0.8648170** which was obtained ffrom the Tuned Regularized Movie + User Effect Model. This was deemed satisfactory based on criteria outlined in the problem statement and the provided grading rubric, and thus, after further exploration failed to improve the model, I concluded my work to present these findings which are considered a success.