Movie Lens Report

By phil@pjb3.com

6/15/2019

#Part 1) EXECUTIVE SUMMARY #TODO

HarvardX Data Science Capstone Class

June 2019

This is my Move Lens project for the Capstone Course in the HarvardX Data Science Certificate Program.

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

My Movie Lens project files have also been stored in Movie_Lens_R_Script.Rmd, and can be viewed using these links

Movie_Lens_R_Script.Rmd

Movie_Lens_Report.Rmd

Movie_Lens_Report.pdf

also... you can view an html page generated from the Movie_Lens_R_Script.Rmd on git hub from here:

Movie_Lens_R_Script.nb.html

#Part 2) METHODS AND ANALYSIS #TODO

####Instructions from the assignment: 1) Develop your algorithm using the edx set.

2) For a final test of your algorithm, predict movie ratings in the validation set as if they were unknown.
3) RMSE will be used to evaluate how close your predictions are to the true values in the validation set.

R script = commented code Rmd = text + code + plots PDF = knit from Rmd

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.

Use multiple models, and see if results improve.

Use Regularization to limit effect of small data points

Use movie effect, user effect & genre effect to improve results.

Use cross-validation.

Show plots to visualize results.

_

1) Develop your algorithm using the edx set.
2) For a final test of your algorithm, predict movie ratings in the validation set as if they were unknown.

3) RMSE will be used to evaluate how close your predictions are to the true values in the validation set.

– General approach:

For many approaches, I will first try on a very small data set, just to get the code working, the 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, then I will.

SET UP libraries and enable turn multi-core processing for some of operations used by the caret package.

Set up.

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 <- FALSE
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
```

```
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
```

```
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
```

```
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
```

```
#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
#

# Note: this process could take a couple of minutes

if(runningInScript){
    #runningInScript
    print("runningInScript is TRUE")
    # 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"
    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)
}
```

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.

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

#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){
        edxFile <- "~/edxFile.rds"
        edx <- readRDS(edxFile)
    }
    edx
}

#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){
        validationFile <- "~/validationFile.rds"
        validation <- readRDS(validationFile)
    }
    validation
}


#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) {
    ifelse(!runningInScript,
    { #!runningInScript
        print ("runningInScipt is FALSE")

    },
    { #runningInScript
    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 ge
    glimpse(oneRow)

    # For each movie, Extract genres into T/F columns
    genreDf <- as_tibble(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 numer
    factors <- data  %>%
    transmute(userId=userId,
              movieId=movieId,
              titleId=as.factor(title),
              genres=as.factor(genres))
    dim(factors) #just to see the value

    # 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
    moviesWithGenre <- cbind(factors,genreDf,tibble(test=data$test)) %>%
                       mutate(rating=data$rating)
    moviesWithGenre #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)

    movieDataWithGenre <- myPreProcess(rbind(d1,d2)) # Preprocess and get subsets of the data
    glimpse(movieDataWithGenre)
    myTrain <- movieDataWithGenre %>% filter(test==FALSE) %>% select (-test,-genres)
    myTest  <- movieDataWithGenre %>% filter(test==TRUE) %>% select (-test)

    ## 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."
```

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
```

```
# 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(predicted_ratings, myTest$rating)

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(predicted_ratings, myTest$rating)
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(predicted_ratings, myTest$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Regularized Movie Effect Model",
                                 RMSE = model_3_rmse))
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 {#Best-Results} #TODO

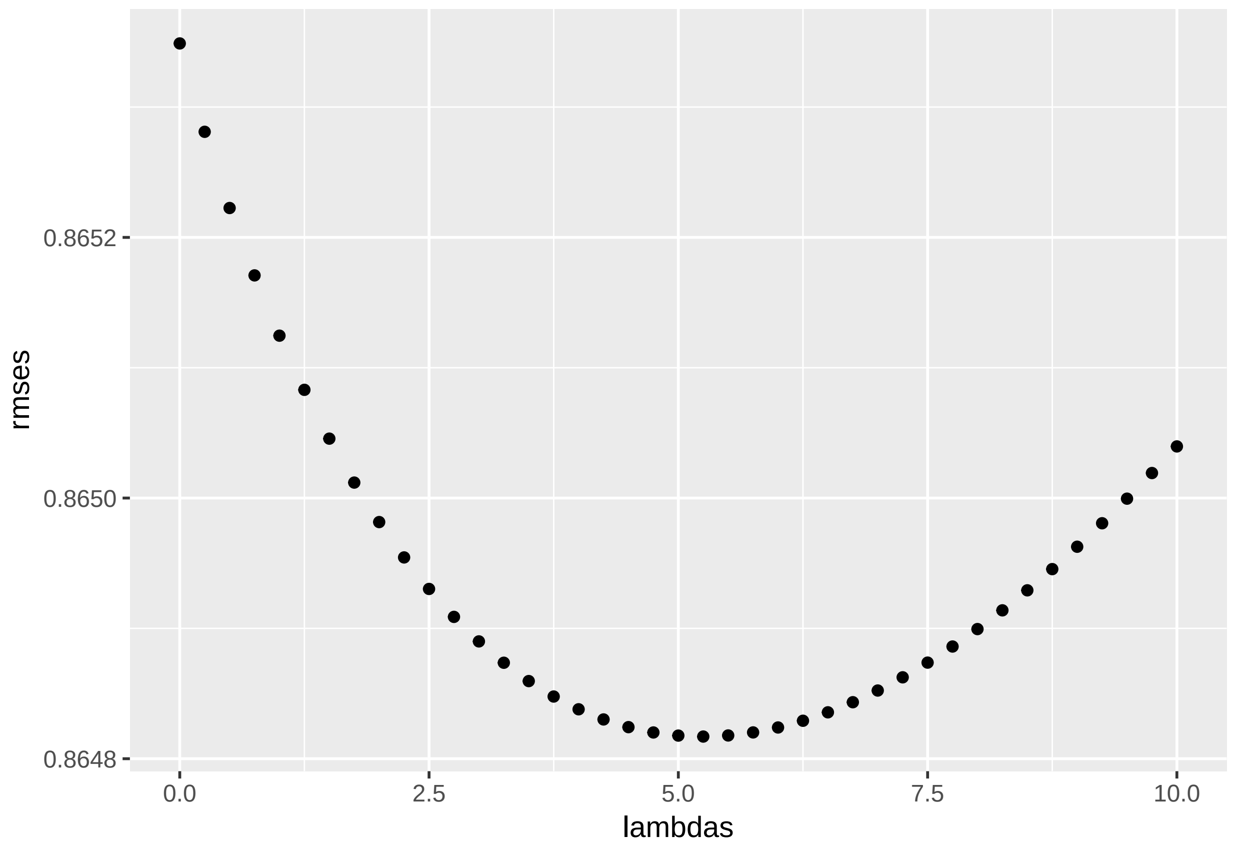
Now we will tune the parameter lambda.

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

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

```r
# The estimates that minimize this can be found similarly to what we did above.
# Here we use cross-validation to pick a l   ambda
```

```r
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)

        return(myRMSE(predicted_ratings, myTest$rating))
    })

    ## 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)

    #free memory
    rm(edx,validation,predicted_ratings,user_avgs,movie_avgs,movie_reg_avgs)
}

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

qplot(lambdas, rmses)
```



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

```
## [1] 5.25
```

```r
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	0.8648170

Additional examination, and trying different models.

```r
#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.

#Only do the additional analysis if runningInScript is TRUE
if(runningInScript){
```

```r
#runningInScript
print("runningInScript is TRUE")
x<-as.matrix(myTrain[5:23])
sds <- colSds(x, na.rm = TRUE)
o <- order(sds, decreasing = TRUE)[1:19]
dim(x)
#Draw a heatmap, using a subset of the data
x2 <- as.matrix(getSubset(x[,o],myTrain$rating,0.0005))
dim(x2)
sds2 <- colSds(x2, na.rm = TRUE)
o2 <- order(sds2, decreasing = TRUE)[1:19]
heatmap(x2[,o2], col = RColorBrewer::brewer.pal(11, "Spectral"))
o2
sds2
apply(x2,MARGIN=2,mean)
colnames(x2)

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

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])

# 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)

# 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)

# box and whisker plots to compare models
scales <- list(x=list(relation="free"), y=list(relation="free"))
bwplot(results, scales=scales)

# 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)


#
# # Create a Random Forest model with default parameters
# model1 <- randomForest(rating ~ ., data = myTrainSubset[colNames], importance = TRUE, mtry=6)
# model1
# print(model1)

dart.parm <- list(booster = "dart",
                  rate_drop = 0.1,
                  nthread = 4,
                  eta = 0.1,
                  max_depth = 3,
                  subsample = 1,
                  eval_metric = "rmse")
rowNames <- rownames(myTrainSubset)
dart <- xgboost(data = as.matrix(myTrainSubset[,-1]),
                label = rownames(myTrainSubset[,-1]),
                params = dart.parm,
                nrounds = 50,
                verbose = 0,
                seed = 2017)
colNames <- names(myTrainSubset[, -1])
colNames
```

```
    pred1 <- predict(dart, as.matrix(myTrainSubset[, -1]))
    pred2 <- predict(dart, as.matrix(myTest[colNames]))
    roc(as.factor(myTrainSubset$rating), pred1)
    # Area under the curve: 0.7734
    roc(as.factor(myTest$rating), pred2)
    # Area under the curve: 0.6517


    ##ctree2 with CV
      fitControl <- trainControl(method = 'cv', number=6,summaryFunction=defaultSummary)
      set.seed(123)
      Grid <- expand.grid(maxdepth = seq(10, 30,5),mincriterion=0.95)

      formula <- rating ~   userId + movieId + Drama + Comedy + Action
      fit.ctree2CV <- train(formula, data=myTrainSubset, method = 'ctree2',
                            trControl=fitControl,tuneGrid=Grid,metric='RMSE')
      print(fit.ctree2CV)
      predict <- predict(fit.ctree2CV, as.matrix(myTest[c("userId","movieId","Drama","Comedy","Action")]
      rmse <- myRMSE(myTest$rating,predict)
      rmse
      #results <- resamples(list(CTREE2=fit.ctree2CV))
}
```

#Part 4) CONCLUSION #TODO