Movie Lens Report

By phil@pjb3.com

6/15/2019

HarvardX Data Science Capstone Class

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

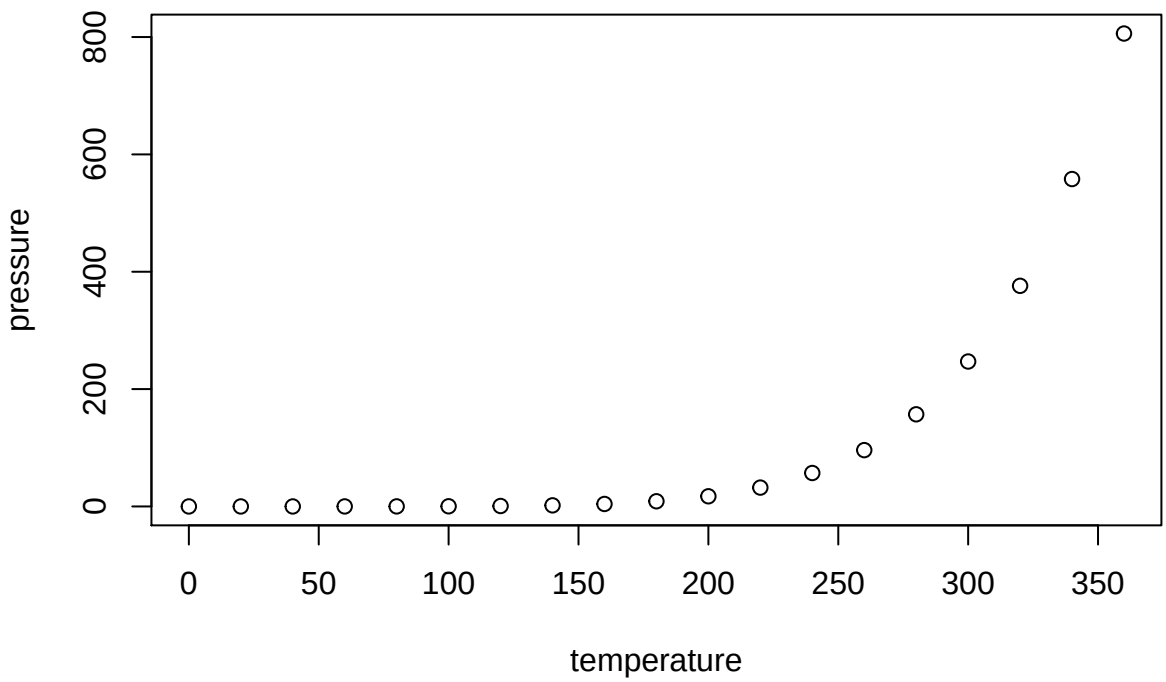
When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
summary(cars)
```

```
##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00  
```

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.

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

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.

---

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.
Set up.
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:
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

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" edxFile <- "~/edxFile.rds" validationFile <- "~/validationFile.rds" fileName <- "movieLensSetup.rds"
#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) } }
PJB - Changed repo 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,repo <- "https://cran.revolutionanalytics.com/" , repo <- "http://cran.us.r-project.org") "'

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

"'r # if(!require(tidyverse)) install.packages("tidyverse", repos = repo) "'

```
## Loading required package:
tidyverse
## -- Attaching packages
--------------------------------------------------------------------------------
tidyverse 1.2.1 --
```

```
## 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
--------------------------------------------------------------------------------
tidyverse_conflicts() -- ## x
dplyr::filter() masks
stats::filter() ## x dplyr::lag()
masks stats::lag()
r if(!require(caret))
install.packages("caret", repos =
repo)
## 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 # MovieLens 10M dataset: # https://grouplens.org/datasets/movielens/10m/ # http://files.grouplens.org/datasets/movielens/ml-10m.zip
dl <- tempfile() ## Don’t actually download for this report. Only do it in the “real” script
#download.file(“http://files.grouplens.org/datasets/movielens/ml-10m.zip”, dl)
dl “‘

```
## [1]
"/var/folders/j1/rf3x82cx5f9c38284dmgdrjh0000gn/T//RtmpUcntb8/file1445319f69
```

“‘r # Skip this step in the report. Only run it in the real script. #ratings <- read.table(text = gsub(“::”, “"̂, 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.
#For the report, don’t save intermediate files. #saveProgress <- TRUE
saveProgress <- FALSE
saveWork(t1,ratings,saveProgress)

```
## For the report, don't run the code below. Only run it in the real script.
#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.
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,fileName,isMac,repo)
```

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

“‘r 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
```

“‘r #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 }

```
# Function to get the repository to use
# Needed because on my Mac, the
standard CRAN repository gave me
errors. getRepos <- function() { isMac
<- TRUE #isMac <- FALSE ##Set to
false if you are NOT running on a Mac.
ifelse(isMac,repo <-
"https://cran.revolutionanalytics.com/" ,
repo <- "http://cran.us.r-project.org") }
repos <- getRepos() #Use a differnt
repository for R packages if on the Mac
#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$genres # A vector with one entry for each unique genere rm(allGenres) genresByMovie <- data %>% select(genres) # string containing multiple generes with | separator oneRow <- extractGenres-Data("Comedy",genresByMovie) #Get one row, just to test "extracting" one genre glimpse(oneRow)
For each movie, Extract genres into T/F columns genreDf <- as_tibble(sapply(allGenresStr,extractGenresData,genresByMovie)) rm(allGenresStr)
dim(genreDf) #Just to see the value
#Now convert certain fields to factors, since they are intended as labels, and not meaningful numeric values. factors <- data %>% transmute(userId=userId, movieId=movieId, titleId=as.factor(title), genres=as.factor(genres)) dim(factors)
#just to see the value
Get the movie data with factors and genres, # and add the rating, which is the random variable we want to predict, as the last column moviesWithGenre <- cbind(factors,genreDf,tibble(test=data*test*))rating)
moviesWithGenre #And return the now preProcessed data frame that is the result. }
Function getData is used to run the code on a subset of the data # during development, so that the code runs faster.
This function will either return the data pased, # or a subset of the data based on the value of returnSubSet second parameter # getData <- function (data,returnSubSet) { myData <- data if(returnSubSet) { #Only use a Subset if in development percent <- 0.001
#percent <- 0.01 dataSubset <- getSubset(data,edx$rating,percent)
myData <- dataSubset
} myData #Return the data } "'

The code below will load the edx and validation data from local files, after it has been downloaded, processed and saved per the instructions provided. You can re-run this chunk of code to reload the data, rather than download and process it again.
“‘r loadFromFile <- FALSE loadFromFile <- TRUE ## Comment out this line if we don’t want to reload the ojbects from files
load the data from the saved file edx <- restoreEdx(loadFromFile) validation <- restoreValidation(loadFromFile) “‘
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.
“‘r ##Preprocess and get either the training data, or a subset it, depending on the value of inDevlopment.
# To make computations faster, # only use a small subset of the training data for now. # Later, re-run the code using the full training set once everything works.
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.
data <- getData(edx,inDevelopment) # Get the training data, or a subset of it.
glimpse(data) “‘

```
## Observations: 9,002 ##
Variables: 6 ## $ userId    <int>
8, 8, 10, 13, 28, 78, 94, 96, 103,
119, 120, 122, 12... ## $ movieId
<dbl> 4701, 8371, 42, 2012, 2174,
7698, 442, 2991, 1298, 5... ## $
rating    <dbl> 3.0, 4.0, 4.0,
4.0, 4.0, 4.0, 3.0, 3.0, 3.0, 2.0,
3.... ## $ timestamp <int>
1111624117, 1111623255, 941544552,
1023205682, 91260... ## $ title
<chr> "Rush Hour 2 (2001)",
"Chronicles of Riddick, The (2...
## $ genres    <chr>
"Action|Comedy",
"Action|Adventure|Sci-Fi|Thriller",...
```

“‘r d1 <- data %>% mutate(test=FALSE) d2 <- validation %>% mutate(test=TRUE) movieDataWithGenre <- myPreProcess(rbind(d1,d2)) # Preprocess and get subsets of the data “‘

```
##  num [1:1009001] 1 0 0 1 1 0 0 0 0 1 ...
r glimpse(movieDataWithGenre)
```

```
## Observations: 1,009,001 ##
Variables: 25 ## $ userId
<int> 8, 8, 10, 13, 28, 78, 94,
96, 103, 119, 120, 122, ... ## $
movieId     <dbl> 4701, 8371, 42,
2012, 2174, 7698, 442, 2991,
1298,... ## $ titleId     <fct>
"Rush Hour 2 (2001)", "Chronicles
of Riddick, The ... ## $ genres
<fct> Action|Comedy,
Action|Adventure|Sci-Fi|Thriller,
A... ## $ Drama       <dbl> 0, 0,
1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1,
0, 1, 0, 1,... ## $ Comedy
<dbl> 1, 0, 0, 1, 1, 0, 0, 0, 0,
1, 0, 1, 0, 0, 0, 1, 0,... ## $
Action      <dbl> 1, 1, 1, 0, 0,
0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0,
1,... ## $ Thriller    <dbl> 0, 1,
0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 1,... ## $ Adventure
<dbl> 0, 1, 0, 0, 0, 0, 1, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0,... ## $
Romance     <dbl> 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0,... ## $ `Sci-Fi`    <dbl> 0, 1,
0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0,... ## $ Crime
<dbl> 0, 0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,... ## $
Fantasy     <dbl> 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,... ## $ Children    <dbl> 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0,... ## $ Horror
<dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,... ## $
Mystery     <dbl> 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,... ## $ War         <dbl> 0, 0,
0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
0, 0, 0, 0,... ## $ Animation
<dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,... ## $
Musical     <dbl> 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0,... ## $ Western     <dbl> 0, 0,
0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0,... ## $ `Film-Noir`
<dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,... ## $
Documentary <dbl> 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,... ## $ IMAX        <dbl> 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0,... ## $ test
<lgl> FALSE, FALSE, FALSE, FALSE,
FALSE, FALSE, FALSE, F... ## $
rating      <dbl> 3.0, 4.0, 4.0,
4.0, 4.0, 3.0, 3.0, 3.0, 2.0,
```

“‘r myTrain <- movieDataWithGenre %>% filter(test==FALSE) %>% select (-test,-genres) myTest <- movieDataWithGenre %>% filter(test==TRUE) %>% select (-test) rm(d1,d2,movieDataWithGenre) “‘
First look at the most naive approach. . . and notice that 3.51 is the mu_hat that has the lowest RMSE.

```
r # Using model Y_u_i = U + E_u_i
mu_hat <- mean(myTrain$rating)
mu_hat
## [1] 3.510442
```

“‘r #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)) } 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 “‘

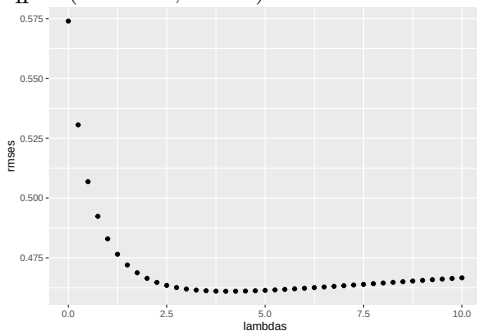
```
## # A tibble: 1 x 2 ##   method
RMSE ##   <chr>            <dbl>
## 1 Just the average  1.06
```

“‘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(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 1.08
"'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
1.08 ## 3 Movie + User Effects
Model 0.574 ## 4 Regularized
Movie Effect Model 0.912 Now we
will tune the parameter lambda.

```
r library(matrixStats)
## ## Attaching package:
'matrixStats'
## The following object is masked
from 'package:dplyr': ## ##
count
```

“‘r library(tidyverse) # The estimates that minimize this can be found similarly to what we did above. # Here we use cross-validation to pick a lambda 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)) }) qplot(lambdas, rmses) “‘




```
r lambda <-
lambdas[which.min(rmses)] lambda
## [1] 4
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 1.0612029 Movie Effect Model 1.0849646 Movie + User Effects Model 0.5739749 Regularized Movie Effect Model 0.9118584 Regularized Movie + User Effect Model 0.4610413

```r
#free memory
rm(data,edx,validation,predicted_ratings,user_avgs,movie_avgs,movie_reg_avgs)


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

x<-as.matrix(myTrain[5:23])
glimpse(x)
```

```
##  num [1:9002, 1:19] 1 0 0 1 1 0 0 0 0 1 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:19] "Comedy" "Action" "Thriller" "Adventure" ...
```

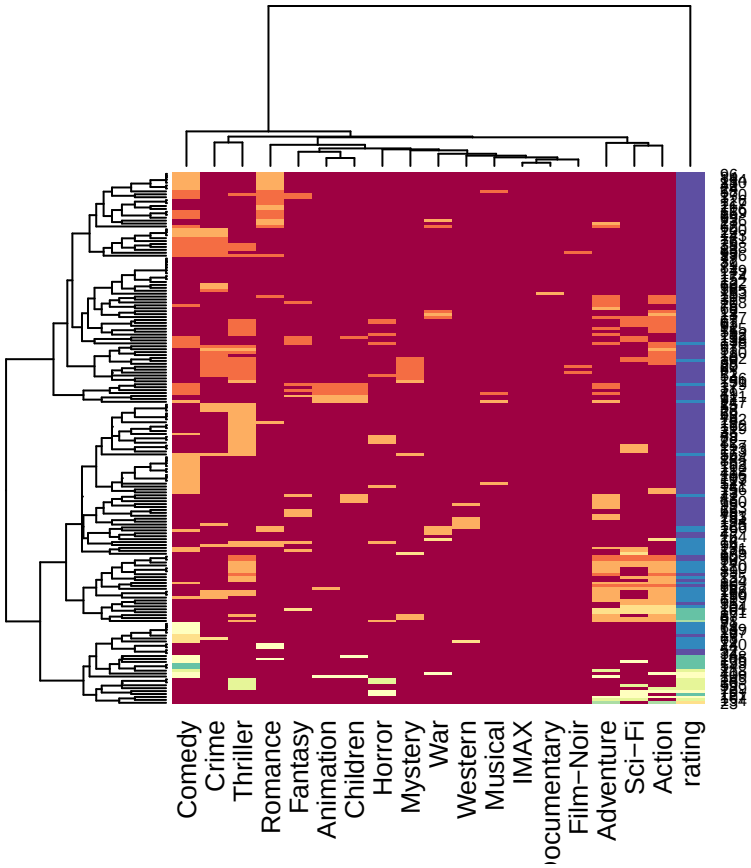
```r
sds <- colSds(x, na.rm = TRUE)
o <- order(sds, decreasing = TRUE)[1:19]
dim(x)
```

```
## [1] 9002   19
```

```r
x2 <- as.matrix(getSubset(x[,o],myTrain$rating,0.02))
dim(x2)
```

```
## [1] 182  19
```

```r
heatmap(x2[,o], col = RColorBrewer::brewer.pal(11, "Spectral"))
```



```r
apply(x2[,o],MARGIN=2,mean)
```

```
##        IMAX      rating      Comedy      Action    Thriller   Adventure
## 0.000000000 3.450549451 0.373626374 0.274725275 0.329670330 0.258241758
##       Crime     Romance      Sci-Fi     Fantasy    Children      Horror
## 0.186813187 0.159340659 0.192307692 0.087912088 0.071428571 0.082417582
##     Mystery         War   Animation     Musical     Western   Film-Noir
## 0.071428571 0.054945055 0.049450549 0.021978022 0.032967033 0.016483516
## Documentary
## 0.005494505
```

```r
top <- x[1,1:5]
cols <- names(top)
```

The code below is a useful utility that shows you the objects in memory, and how large they are. Courtesy of stack overflow
(https://stackoverflow.com/questions/1358003/tricks-to-manage-the-available-memory-in-an-r-session)

```r
# improved list of objects
.ls.objects <- function (pos = 1, pattern, order.by,
                         decreasing=FALSE, head=FALSE, n=5) {
    napply <- function(names, fn) sapply(names, function(x)
                                         fn(get(x, pos = pos)))
    names <- ls(pos = pos, pattern = pattern)
    obj.class <- napply(names, function(x) as.character(class(x))[1])
    obj.mode <- napply(names, mode)
    obj.type <- ifelse(is.na(obj.class), obj.mode, obj.class)
    obj.prettysize <- napply(names, function(x) {
                             format(utils::object.size(x), units = "auto") })
    obj.size <- napply(names, object.size)
    obj.dim <- t(napply(names, function(x)
                        as.numeric(dim(x))[1:2]))
    vec <- is.na(obj.dim)[, 1] & (obj.type != "function")
    obj.dim[vec, 1] <- napply(names, length)[vec]
    out <- data.frame(obj.type, obj.size, obj.prettysize, obj.dim)
    names(out) <- c("Type", "Size", "PrettySize", "Length/Rows", "Columns")
    if (!missing(order.by))
        out <- out[order(out[[order.by]], decreasing=decreasing), ]
    if (head)
        out <- head(out, n)
    out
}

# shorthand
lsos <- function(..., n=10) {
    .ls.objects(..., order.by="Size", decreasing=TRUE, head=TRUE, n=n)
}

lsos()
```

```
##                         Type      Size PrettySize Length/Rows Columns
## myTest            data.frame 180971912   172.6 Mb      999999      24
## myTrain           data.frame   2486008     2.4 Mb        9002      23
## x                     matrix   1370008     1.3 Mb        9002      19
## myPreProcess        function    111400   108.8 Kb          NA      NA
```

```
## x2                    matrix     29368    28.7 Kb         182      19
## getSubset           function     22736    22.2 Kb          NA      NA
## extractGenresData   function     20552    20.1 Kb          NA      NA
## myRMSE              function     19744    19.3 Kb          NA      NA
## saveWork            function     19584    19.1 Kb          NA      NA
## getData             function     13400    13.1 Kb          NA      NA
```

My notes:

Don't use lm on huge datasets. Look at the alternatives using movie_avgs and user_avgs in the lectures.

34.7.5 Modeling movie effects

Don't use lm_fit with huge amounts of data. Checkout 34.7.5 Modeling movie effects in EDXdatascience-book.pdf for an alternative.

Use Genres to improve RMSE scores.

https://courses.edx.org/courses/course-v1:HarvardX+PH125.9x+2T2018/discussion/forum/1ae77f8b04e1c45e4b425cab952e451 threads/5c7ac73b58adcb0994003491

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

Test out tidyr::separate_rows(edx, genres, sep = "|") on a very small subset of edx and see what it is doing..

========== So I just finished up the machine learning class and started this one. I started on my project and have calculated and used b_i and b_u to inform my prediction. I was expecting to have to do more, but by rmse is already below 8.7, which exceeds our target. Have I messed something up? Will I be punished for having code that doesn't do everything the lessons did (regularization, genres, etc)?

Hi, You can try different methods and select one with the best performances with a reasonable explanation.

I just finished my analysis using the same two parameters, and was wondering if something was wrong as those two produce a reasonably good model per the rubric. I was going over the code time and again to make sure everything is right, and it appears to be. I am glad I am not the only one. For the record, I implemented regularization as well, but the improvement is not significant.

Don't use lm on huge datasets. Look at the alternatives using movie_avgs and user_avgs in the lectures.

One tip from me is for the genre effect. It is important to note that each user has a different genre effect. Note also that each movie has different genres. Finally and most importantly, treat genres as independent. That is, if a movie is having Romance|Comedy, then treat it as having both romance and comedy rather than Romance|Comedy as one unit. This is due to sparseness of the data. If you treat each combination of the genre as unique then there will be many genres and very few ratings in each genre. This will not give you a robust estimate.

Using this it was possible for me to go below 0.85 RMSE.

I used most of the same methods. I personally could have spent more time on the regularization, but my first guess at lambda got me under the the required RMSE so I called it good. One of my reviewers called me on it saying It was dumb luck and i should have used cross validation (they were right) but they gave me full points.

Look at the rubric and see what you are being graded against. There isn't a section for originality. So dont sweat it too much. Again save the fear for the choose your own project. I'm waiting on my staff review and its nerve racking.

---

http://factominer.free.fr/factomethods/multiple-factor-analysis.html

res = MFA(wine, group=c(2,5,3,10,9,2), type=c("n",rep("s",5)), ncp=5, name.group=c("origin","odor","visual","odor.after.shaking", "taste","overall"), num.group.sup=c(1,6))
#wine: the data set used #group: a vector indicating the number of variables in each group #type: the type of the variables in each group. "s" for scaled continuous variables, "c" for centered (unscaled) continuous variables and "n" for categorical variables #ncp: number of dimensions kept in the result #name.group: names of the groups #num.group.sup: indexes of the supplementary groups
"'r #if you want to see the objects within a function, you have to use: lsos(pos = environment()), otherwise it'll only show #global variables. To write to standard error: write.table(lsos(pos=environment()), stderr(), quote=FALSE, sep="')
#I use the data.table package. With its := operator you can :
Add columns by reference # Modify subsets of existing columns by reference, and by group by reference # Delete columns by reference # None of these operations copy the (potentially large) data.table at all, not even once. # # Aggregation is also particularly fast because data.table uses much less working memory. # # https://www.gastonsanchez.com/visually-enforced/how-to/2012/06/17/PCA-in-R/
Use MCA to detemine which generes have the most influence # Multiple Correspondence Analysis (MCA)
#Similar to PCA, but useful for categorical data.
http://factominer.free.fr/factomethods/multiple-correspondence-analysis.html
#Here are some models to consider
#Sparse Partial Least Squares
method = 'spls' #Type: Regression, Classification
#Tuning parameters:
#K (#Components) #eta (Threshold) #kappa (Kappa) #Required packages: spls #
#Linear Regression with Stepwise Selection

```
# method = 'leapSeq' # Type: Regression #
# Tuning parameters: # # nvmax
(Maximum Number of Predictors) #
Required packages: leaps # # Linear
Regression with Stepwise Selection # #
method = 'lmStepAIC' # Type: Regression
# # No tuning parameters for this model #
# Required packages: MASS "'
```

Don't forget the github link: https://github.com/satyajitpatra1976/Capstone_PH125.9x

https://rafalab.github.io/dsbook/large-datasets.html#dimension-reduction

==========

Do a PCA to determie best predictors...

Check to see if they are correllated, and remove if they are.

Use multiple models, and see if results improve. Use an ensemble approach.

Tune parameters using cross-validation.

Use Regularization to limit effect of small data points

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

Show plots to visualize results.