R Code Output

Shon Mohsin

June 16, 2019

## 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:

spamPath = "data"  
  
#Import required packages:  
options(warn=-1)  
suppressPackageStartupMessages(require(tm))  
#install.packages("RColorBrewer")  
library(RColorBrewer)  
#library(RSpamData)  
library(rpart)  
library(NLP)  
library(tm)  
library(rpart.plot)  
suppressPackageStartupMessages(require(caret))  
suppressPackageStartupMessages(require(MLmetrics))  
library(naivebayes)

## naivebayes 0.9.6 loaded

library(e1071)  
suppressPackageStartupMessages(require(randomForest))  
library(xgboost)  
library(ggplot2)  
suppressPackageStartupMessages(require(kernlab))  
  
# Load Data: First Downloaded data and saved  
  
  
list.dirs(spamPath, full.names = FALSE)

## [1] "" "messages" "messages/easy\_ham"   
## [4] "messages/easy\_ham\_2" "messages/hard\_ham" "messages/spam"   
## [7] "messages/spam\_2"

list.files(path = paste(spamPath, "messages",   
 sep = .Platform$file.sep))

## [1] "easy\_ham" "easy\_ham\_2" "hard\_ham" "spam" "spam\_2"

head(list.files(path = paste(spamPath, "messages", "spam\_2",  
 sep = .Platform$file.sep)))

## [1] "00001.317e78fa8ee2f54cd4890fdc09ba8176"  
## [2] "00002.9438920e9a55591b18e60d1ed37d992b"  
## [3] "00003.590eff932f8704d8b0fcbe69d023b54d"  
## [4] "00004.bdcc075fa4beb5157b5dd6cd41d8887b"  
## [5] "00005.ed0aba4d386c5e62bc737cf3f0ed9589"  
## [6] "00006.3ca1f399ccda5d897fecb8c57669a283"

dirNames = list.files(path = paste(spamPath, "messages",   
 sep = .Platform$file.sep))  
length(list.files(paste(spamPath, "messages", dirNames,   
 sep = .Platform$file.sep)))

## [1] 9353

# Data Cleansing Process:  
dirNames = list.files(path = paste(spamPath, "messages",   
 sep = .Platform$file.sep))  
length(list.files(paste(spamPath, "messages", dirNames,   
 sep = .Platform$file.sep)))

## [1] 9353

sapply(paste(spamPath, "messages", dirNames,   
 sep = .Platform$file.sep),   
 function(dir) length(list.files(dir)) )

## data/messages/easy\_ham data/messages/easy\_ham\_2 data/messages/hard\_ham   
## 5052 1401 501   
## data/messages/spam data/messages/spam\_2   
## 1001 1398

fullDirNames = paste(spamPath, "messages", dirNames,   
 sep = .Platform$file.sep)  
  
fileNames = list.files(fullDirNames, full.names = TRUE)  
fileNames[1]

## [1] "data/messages/easy\_ham/00001.7c53336b37003a9286aba55d2945844c"

fileNames

## [1] "data/messages/easy\_ham/00001.7c53336b37003a9286aba55d2945844c"   
## [2] "data/messages/easy\_ham/00002.9c4069e25e1ef370c078db7ee85ff9ac"   
## [3] "data/messages/easy\_ham/00003.860e3c3cee1b42ead714c5c874fe25f7"   
## [4] "data/messages/easy\_ham/00004.864220c5b6930b209cc287c361c99af1"   
## [5] "data/messages/easy\_ham/00005.bf27cdeaf0b8c4647ecd61b1d09da613"   
msg = readLines(fileNames[1])  
head(msg)

## [1] "From exmh-workers-admin@redhat.com Thu Aug 22 12:36:23 2002"   
## [2] "Return-Path: <exmh-workers-admin@spamassassin.taint.org>"   
## [3] "Delivered-To: zzzz@localhost.netnoteinc.com"   
## [4] "Received: from localhost (localhost [127.0.0.1])"   
## [5] "\tby phobos.labs.netnoteinc.com (Postfix) with ESMTP id D03E543C36"  
## [6] "\tfor <zzzz@localhost>; Thu, 22 Aug 2002 07:36:16 -0400 (EDT)"

set.seed(531256)  
  
#a)Use the m function to permute the indices of the   
#training set, and organize these permuted indices into   
#5 equal-size sets, called folds.  
  
cleanMsg = tolower(gsub("[[:punct:]0-9[:blank:]]+", " ", msg))  
cleanMsg[ c(1, 15, 26, 27) ]

## [1] "from exmh workers admin redhat com thu aug "   
## [2] " edt "   
## [3] " exmh workers redhat com thu aug "   
## [4] "received from ratree psu ac th by mx spamassassin taint org"

stopWords = stopwords()  
cleanSW = tolower(gsub("[[:punct:]0-9[:blank:]]+", " ", stopWords))  
SWords = unlist(strsplit(cleanSW, "[[:blank:]]+"))  
SWords = SWords[ nchar(SWords) > 1 ]  
stopWords = unique(SWords)  
words = unlist(strsplit(cleanMsg, "[[:blank:]]+"))  
words = words[ nchar(words) > 1 ]  
words = words[ !( words %in% stopWords) ]  
head(words)

## [1] "exmh" "workers" "admin" "redhat" "com" "thu"

#To split the message into its header and body  
splitMessage = function(msg) {  
 #assign the location of first empty line to "splitpoint"  
 splitPoint = match("", msg)  
 header = msg[1:(splitPoint-1)]  
 body = msg[ -(1:splitPoint) ]  
 return(list(header = header, body = body))  
}  
# remove attachments from message body:  
dropAttach = function(body, boundary){  
   
 bString = paste("--", boundary, sep = "")  
 bStringLocs = which(bString == body)  
   
 if (length(bStringLocs) <= 1) return(body)  
   
 eString = paste("--", boundary, "--", sep = "")  
 eStringLoc = which(eString == body)  
 if (length(eStringLoc) == 0)   
 return(body[ (bStringLocs[1] + 1) : (bStringLocs[2] - 1)])  
   
 n = length(body)  
 if (eStringLoc < n)   
 return( body[ c( (bStringLocs[1] + 1) : (bStringLocs[2] - 1),   
 ( (eStringLoc + 1) : n )) ] )  
   
 return( body[ (bStringLocs[1] + 1) : (bStringLocs[2] - 1) ])  
}  
  
#This function returns teh boundry string:  
getBoundary = function(header) {  
 boundaryIdx = grep("boundary=", header)  
 boundary = gsub('"', "", header[boundaryIdx])  
 gsub(".\*boundary= \*([^;]\*);?.\*", "\\1", boundary)  
}  
  
# This function identifies and extracts words from the message body:  
findMsgWords = function(msg, stopWords) {  
 if(is.null(msg))  
 return(character())  
   
 words = unique(unlist(strsplit(cleanText(msg), "[[:blank:]\t]+")))  
   
 # drop empty and 1 letter words  
 words = words[ nchar(words) > 1]  
 words = words[ !( words %in% stopWords) ]  
 invisible(words)  
}  
  
cleanText = function(msg) {  
 tolower(gsub("[[:punct:]0-9[:space:][:blank:]]+", " ", msg))  
}  
  
#Completing the data preparation process:  
processAllWords = function(dirName, stopWords)  
{  
 # read all files in the directory  
 fileNames = list.files(dirName, full.names = TRUE)  
 # drop files that are not email, i.e., cmds  
 notEmail = grep("cmds$", fileNames)  
 if ( length(notEmail) > 0) fileNames = fileNames[ - notEmail ]  
   
 messages = lapply(fileNames, readLines, encoding = "latin1")  
   
 # split header and body  
 emailSplit = lapply(messages, splitMessage)  
 # put body and header in own lists  
 bodyList = lapply(emailSplit, function(msg) msg$body)  
 headerList = lapply(emailSplit, function(msg) msg$header)  
 rm(emailSplit)  
   
 # determine which messages have attachments  
 hasAttach = sapply(headerList, function(header) {  
 CTloc = grep("Content-Type", header)  
 if (length(CTloc) == 0) return(0)  
 multi = grep("multi", tolower(header[CTloc]))   
 if (length(multi) == 0) return(0)  
 multi  
 })  
   
 hasAttach = which(hasAttach > 0)  
   
 # find boundary strings for messages with attachments  
 boundaries = sapply(headerList[hasAttach], getBoundary)  
   
 # drop attachments from message body  
 bodyList[hasAttach] = mapply(dropAttach, bodyList[hasAttach],   
 boundaries, SIMPLIFY = FALSE)  
   
 # extract words from body  
 msgWordsList = lapply(bodyList, findMsgWords, stopWords)  
   
 invisible(msgWordsList)  
}  
  
# Apply "proceesAllwords" to each directory  
msgWordsList <- lapply(fullDirNames, processAllWords, stopWords = stopWords)   
# The number of elements (messages) in each list (1,2,3 is ham and 4,5 is spam)  
numMsgs <- sapply(msgWordsList, length)  
numMsgs

## [1] 5051 1400 500 1000 1397

isSpam = rep(c(FALSE, FALSE, FALSE, TRUE, TRUE), numMsgs)  
  
# Flatten teh 5 lists into one list:  
msgWordsList = unlist(msgWordsList, recursive = FALSE)  
#  
numEmail = length(isSpam)  
numSpam = sum(isSpam)  
numHam = numEmail - numSpam  
  
#set teh seed value=418910  
set.seed(418910)  
  
# sample() takes the sampel of teh specified size from teh elements of numSpam and numHam  
# using either with or without replacement. In here, it is without replacement.  
testSpamIdx = sample(numSpam, size = floor(numSpam/3))  
testHamIdx = sample(numHam, size = floor(numHam/3))  
testMsgWords = c((msgWordsList[isSpam])[testSpamIdx], (msgWordsList[!isSpam])[testHamIdx] )  
trainMsgWords = c((msgWordsList[isSpam])[ - testSpamIdx], (msgWordsList[!isSpam])[ - testHamIdx])  
  
testIsSpam = rep(c(TRUE, FALSE), c(length(testSpamIdx), length(testHamIdx)))  
trainIsSpam = rep(c(TRUE, FALSE), c(numSpam - length(testSpamIdx), numHam - length(testHamIdx)))  
  
#Bag of Words (bow) is the collection of unique words across all the messages  
bow = unique(unlist(trainMsgWords))  
length(bow)

## [1] 80059

spamWordCounts = rep(0, length(bow))  
names(spamWordCounts) = bow  
  
tmp = lapply(trainMsgWords[trainIsSpam], unique)  
tt = table( unlist(tmp) )  
spamWordCounts[ names(tt) ] = tt  
  
length(testHamIdx)

## [1] 2317

numHam

## [1] 6951

numMsgs

## [1] 5051 1400 500 1000 1397

#b) For each fold, take the corresponding subset from the training data to use as a 'test' set.   
#Use the remaining messages in the training data as the training set.   
#Apply the functions developed to estimate the probabilities that a word occurs in a message  
#given it is spam or ham, and use these probabilities to compute the log likelihood ratio for   
#the messages in the training set.  
  
# This function calculate frequencies.   
# All to find the sum of the log likelihood ratio for the messages in training set  
computeFreqs = function(wordsList, spam, bow = unique(unlist(wordsList)))  
{  
 # create a matrix for spam, ham, and log odds  
 wordTable = matrix(0.5, nrow = 4, ncol = length(bow),  
 dimnames = list(c("spam", "ham", "presentLogOdds", "absentLogOdds"), bow))  
   
 # For each spam message, add 1 to counts for words in message  
 counts.spam = table(unlist(lapply(wordsList[spam], unique)))  
 wordTable["spam", names(counts.spam)] = counts.spam + .5  
   
 # Similarly for ham messages  
 counts.ham = table(unlist(lapply(wordsList[!spam], unique)))   
 wordTable["ham", names(counts.ham)] = counts.ham + .5   
   
 # Find the total number of spam and ham  
 numSpam = sum(spam)  
 numHam = length(spam) - numSpam  
   
 # Prob(word|spam) and Prob(word | ham)  
 wordTable["spam", ] = wordTable["spam", ]/(numSpam + .5)  
 wordTable["ham", ] = wordTable["ham", ]/(numHam + .5)  
   
 # log odds  
 wordTable["presentLogOdds", ] = log(wordTable["spam",]) - log(wordTable["ham", ])  
 wordTable["absentLogOdds", ] = log((1 - wordTable["spam", ])) - log((1 -wordTable["ham", ]))  
 invisible(wordTable)  
}  
  
# Apply the computeFreqs() function to our training data:  
trainTable = computeFreqs(trainMsgWords, trainIsSpam)  
# Now trainTable has all the individual word probabilities needed to construct the log likelihood ratio for a message  
#trainTable  
  
# Sum of the log likelihood for spam test messages  
# Consider the set of words in the second message in "testMsgWords"  
newMsg = testMsgWords[[2]]  
#if there is any new words that dose not exist in the BOW, we just remove it from our calculation (no info about likelihood)  
newMsg = newMsg[!is.na(match(newMsg, colnames(trainTable)))]  
#put the remaining words in our calculations  
present = colnames(trainTable) %in% newMsg  
#compute the log of the ratio of probability (message is spam vs ham)  
sum(trainTable["presentLogOdds", present]) + sum(trainTable["absentLogOdds", !present])

## [1] 169.5238

#The result was positive and large, that means that the second message in "testmasgwords" is spam  
  
# Sum of the log likelihood for ham test messages  
#Now try the second message in test ham message, the result is negative, means it is ham.  
newMsg = testMsgWords[[ which(!testIsSpam)[2] ]]  
newMsg = newMsg[!is.na(match(newMsg, colnames(trainTable)))]  
present = (colnames(trainTable) %in% newMsg)  
sum(trainTable["presentLogOdds", present]) + sum(trainTable["absentLogOdds", !present])

## [1] -95.92779

#Below function exactly dose the same thing that we did in above lines but in a function form.  
#It is the log likelihood ratio function and can be used to find the distribution of words   
#in the ham and spam messages matching with that of the spam word bank  
computeMsgLLR = function(words, freqTable)   
{  
 # Discards words not in training data.  
 words = words[!is.na(match(words, colnames(freqTable)))]  
   
 # Find which words are present  
 present = colnames(freqTable) %in% words  
 sum(freqTable["presentLogOdds", present]) + sum(freqTable["absentLogOdds", !present])  
}  
  
  
# Creates the log likelihood ratio distribution for ham and spam messages  
#apply computeMsgLLR() function to each of the messages in our test set:   
testLLR = sapply(testMsgWords, computeMsgLLR, trainTable)  
#compare the summary statistic of the LLR values for ham and spam in test data:  
cat("The Log Likelihood Ratio Distribution For Ham and Spam Messages")

## The Log Likelihood Ratio Distribution For Ham and Spam Messages

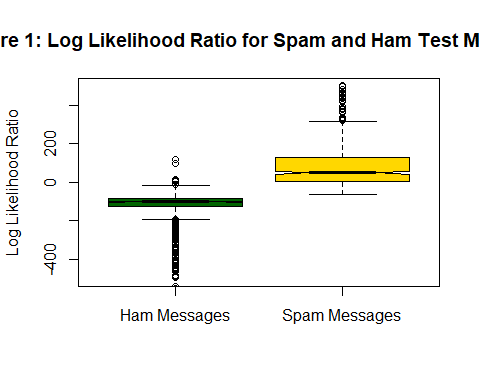
tapply(testLLR, testIsSpam, summary)

## $`FALSE`  
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## -1361.89 -127.06 -101.18 -116.25 -81.26 700.23   
##   
## $`TRUE`  
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## -60.574 6.369 49.837 137.546 131.719 23518.028

#There are 3116 LLR values corresponding to each test message  
length(testLLR)

## [1] 3116

#Create box plot of LLR for test messages  
spamLab = c("Ham Messages", "Spam Messages")[1 + testIsSpam]  
boxplot(testLLR ~ spamLab, ylab = "Log Likelihood Ratio",  
 main = "Figure 1: Log Likelihood Ratio for Spam and Ham Test Messages",  
 ylim=c(-500, 500),  
 notch=TRUE,  
 col=(c("darkgreen","gold")))



#c) Pool all of the likelihood-ratio values from the messages in all of the folds,  
#i.e., from all of the training data, and use these values and the typeIErrorRate() function   
#to select threshold that achieves a 1% Type I error.  
  
# Function to find the type I error rates  
typeIErrorRates = function(llrVals, isSpam)   
{  
 o = order(llrVals)  
 llrVals = llrVals[o]  
 isSpam = isSpam[o]  
   
 idx = which(!isSpam)  
 N = length(idx)  
 list(error = (N:1)/N, values = llrVals[idx])  
}  
  
# Function to find the type II error rates  
typeIIErrorRates = function(llrVals, isSpam) {  
 o = order(llrVals)  
 llrVals = llrVals[o]  
 isSpam = isSpam[o]  
   
 idx = which(isSpam)  
 N = length(idx)  
 list(error = (1:(N))/N, values = llrVals[idx])  
}   
  
#To find the estimate of threshold in order to have typeI error less than 1%.  
xI = typeIErrorRates(testLLR, testIsSpam)  
xII = typeIIErrorRates(testLLR, testIsSpam)  
threshold = round(min(xI$values[xI$error <= 0.01]))  
threshold

## [1] -43

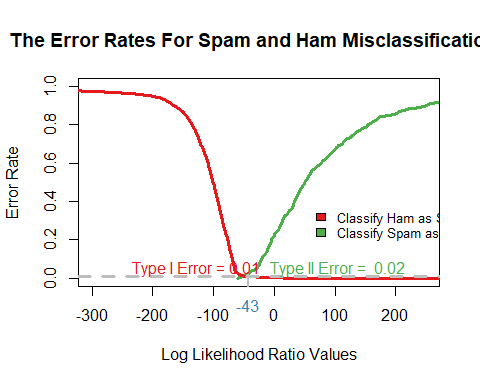
typeIerror = max(xI$error[ xI$values > threshold ])  
typeIerror

## [1] 0.01035822

typeIIerror = max(xII$error[ xII$values < threshold ])  
typeIIerror

## [1] 0.02252816

#A plot of type I and II error rate vs. LLR values  
cols = brewer.pal(9, "Set1")[c(3, 1, 2)]  
plot(xII$error ~ xII$values, type = "l", col = cols[1], lwd = 3,  
 xlim = c(-300, 250), ylim = c(0, 1),  
 xlab = "Log Likelihood Ratio Values", ylab="Error Rate",  
 main = "The Error Rates For Spam and Ham Misclassifications")  
points(xI$error ~ xI$values, type = "l", col = cols[2], lwd = 3)  
legend(x = 50, y = 0.4, fill = c(cols[2], cols[1]),   
 legend = c("Classify Ham as Spam", "Classify Spam as Ham"), cex = 0.8, bty = "n")  
abline(h=0.01, col ="grey", lwd = 3, lty = 2)  
text(-250, 0.05, pos = 4, "Type I Error = 0.01", col = cols[2])  
  
mtext(threshold, side = 1, line = 0.5, at = threshold, col = cols[3])  
segments(x0 = threshold, y0 = -.50, x1 = threshold, y1 = typeIIerror, lwd = 2, col = "grey")  
text(threshold + 20, 0.05, pos = 4, paste("Type II Error = ", round(typeIIerror, digits = 2)), col = cols[1])



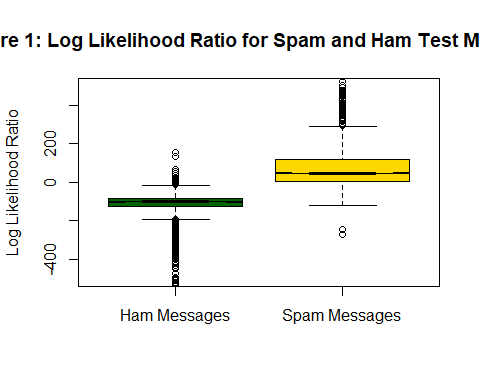
# pooling LLR values from all 5 fold cross validation sets to select the new threshold.  
# 5-fold cross validations  
k = 5  
trainwordsize <- length(trainMsgWords)  
partK <- sample(trainwordsize)  
total <- k \* floor(trainwordsize/k)  
partK <- matrix(partK[1:total], ncol = k)  
  
testFoldLLR = NULL  
for (i in 1:k) {  
 foldIdx <- partK[ , i]  
 traintableFold <- computeFreqs(trainMsgWords[-foldIdx], trainIsSpam[-foldIdx])  
 testFoldLLR <- c(testFoldLLR, sapply(trainMsgWords[foldIdx], computeMsgLLR, traintableFold))}  
  
  
testFoldSpam = NULL  
for (i in 1:k) {  
 foldIdx <- partK[ , i]  
 testFoldSpam <- c(testFoldSpam, trainIsSpam[foldIdx])}  
  
  
xFoldI <- typeIErrorRates(testFoldLLR, testFoldSpam)  
xFoldII <- typeIIErrorRates(testFoldLLR, testFoldSpam)  
tauFoldI <- round(min(xFoldI$values[xFoldI$error <= 0.01]))  
tFold2 <- xFoldII$error[ xFoldII$values < tauFoldI ]  
str(testFoldLLR)

## num [1:6230] -99.85 -161.7 71.23 -9.42 -98.08 ...

tauFoldI

## [1] -35

#Create box plot of LLR for test messages  
spamFoldLab = c("Ham Messages", "Spam Messages")[1 + testFoldSpam]  
boxplot(testFoldLLR ~ spamFoldLab, ylab = "Log Likelihood Ratio",  
 main = "Figure 1: Log Likelihood Ratio for Spam and Ham Test Messages",  
 ylim=c(-500, 500),  
 notch=TRUE,  
 col=(c("darkgreen","gold")))



# d)Apply this threshold to our original/real test set and find its Type I and Type II errors.  
typeIerror = max(xI$error[ xI$values > tauFoldI ])  
typeIerror

## [1] 0.008200259

typeIIerror = max(xII$error[ xII$values < tauFoldI ])  
typeIIerror

## [1] 0.0387985

# Graph for Types I and II Error rates by using 5 fold cross validations.  
#A plot of type I and II error rate vs. LLR values  
cols = brewer.pal(9, "Set1")[c(3, 1, 2)]  
plot(xFoldII$error ~ xFoldII$values, type = "l", col = cols[1], lwd = 3,  
 xlim = c(-300, 250), ylim = c(0, 1),  
 xlab = "Log Likelihood Ratio Values", ylab="Error Rate",  
 main = "The Error Rates For Spam and Ham Misclassifications")  
points(xFoldI$error ~ xFoldI$values, type = "l", col = cols[2], lwd = 3)  
legend(x = 50, y = 0.4, fill = c(cols[2], cols[1]),   
 legend = c("Classify Ham as Spam", "Classify Spam as Ham"), cex = 0.8, bty = "n")  
abline(h=0.008, col ="grey", lwd = 3, lty = 2)  
text(-250, 0.05, pos = 4, "Type I Error = 0.008", col = cols[2])  
mtext(tauFoldI, side = 1, line = 0.5, at = tauFoldI, col = cols[3])  
segments(x0 = tauFoldI, y0 = -.50, x1 = tauFoldI, y1 = typeIIerror, lwd = 2, col = "grey")  
text(tauFoldI + 20, 0.05, pos = 4, paste("Type II Error = ", round(typeIIerror, digits = 2)), col = cols[1])

