Homework4.R

student

Mon Feb 22 00:02:19 2016

##CS 480 Homework 4  
##Neeraj Asthana (nasthan2)  
  
#Setup environment  
spamPath = "/home/student/container-data/RDataScience/SpamAssassinMessages"  
dirNames = list.files(path = paste(spamPath, "messages",   
 sep = .Platform$file.sep))  
fullDirNames = paste(spamPath, "messages", dirNames,   
 sep = .Platform$file.sep)  
  
##Functions from chapter 3 that are necessary  
includeAttach = function(body, boundary){  
   
 bString = paste("--", boundary, sep = "")  
 bStringLocs = which(bString == body)  
   
 # if there are fewer than 2 beginning boundary strings,   
 # there is on attachment to drop  
 if (length(bStringLocs) <= 1) return(body)  
   
 # do ending string processing  
 eString = paste("--", boundary, "--", sep = "")  
 eStringLoc = which(eString == body)  
   
 # if no ending boundary string, grab contents between the first   
 # two beginning boundary strings as the message body  
 n = length(body)  
 if (length(eStringLoc) == 0)   
 return(body[c( (bStringLocs[1] + 1) : (bStringLocs[2] - 1), (bStringLocs[2] + 1) : n )])  
   
 # typical case of well-formed email with attachments  
 # grab contents between first two beginning boundary strings and   
 # add lines after ending boundary string  
 if (eStringLoc < n)   
 return( body[ c( (bStringLocs[1] + 1) : (bStringLocs[2] - 1), (bStringLocs[2] + 1) : (eStringLoc - 1),  
 ( (eStringLoc + 1) : n )) ] )  
   
 # fall through case  
 # note that the result is the same as the   
 # length(eStringLoc) == 0 case, so code could be simplified by   
 # dropping that case and modifying the eStringLoc < n check to   
 # be 0 < eStringLoc < n  
 return( body[ (bStringLocs[1] + 1) : (bStringLocs[2] - 1) ])  
}  
  
splitMessage = function(msg) {   
 splitPoint = match("", msg)  
 header = msg[1:(splitPoint-1)]  
 body = msg[ -(1:splitPoint) ]  
 return(list(header = header, body = body))  
}  
  
getBoundary = function(header) {  
 boundaryIdx = grep("boundary=", header)  
 boundary = gsub('"', "", header[boundaryIdx])  
 gsub(".\*boundary= \*([^;]\*);?.\*", "\\1", boundary)  
}  
  
cleanText =function(msg) {  
 tolower(gsub("[[:punct:]0-9[:space:][:blank:]]+", " ", msg))  
}  
  
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)  
 }  
  
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(includeAttach, bodyList[hasAttach],   
 boundaries, SIMPLIFY = FALSE)  
   
 # extract words from body  
 msgWordsList = lapply(bodyList, findMsgWords, stopWords)  
   
 invisible(msgWordsList)  
}  
  
library(tm)

## Loading required package: NLP

stopWords = stopwords()  
cleanSW = tolower(gsub("[[:punct:]0-9[:blank:]]+", " ", stopWords))  
SWords = unlist(strsplit(cleanSW, "[[:blank:]]+"))  
SWords = SWords[ nchar(SWords) > 1 ]  
stopWords = unique(SWords)  
  
msgWordsList = lapply(fullDirNames, processAllWords,   
 stopWords = stopWords)

## Warning in FUN(X[[i]], ...): incomplete final line found on '/home/  
## student/container-data/RDataScience/SpamAssassinMessages/messages/hard\_ham/  
## 00228.0eaef7857bbbf3ebf5edbbdae2b30493'

## Warning in FUN(X[[i]], ...): incomplete final line found on '/home/  
## student/container-data/RDataScience/SpamAssassinMessages/messages/hard\_ham/  
## 0231.7c6cc716ce3f3bfad7130dd3c8d7b072'

## Warning in FUN(X[[i]], ...): incomplete final line found on '/home/  
## student/container-data/RDataScience/SpamAssassinMessages/messages/hard\_ham/  
## 0250.7c6cc716ce3f3bfad7130dd3c8d7b072'

numMsgs = sapply(msgWordsList, length)  
isSpam = rep(c(FALSE, FALSE, FALSE, TRUE, TRUE), numMsgs)  
  
# Flatten the message words into a single list of lists of message words.  
msgWordsList = unlist(msgWordsList, recursive = FALSE)  
  
# Set a particular seed, so the results will be reproducible.  
set.seed(418910)  
  
# Take approximately 1/3 of the spam and ham messages as our test spam and ham messages.  
numEmail = length(isSpam)  
numSpam = sum(isSpam)  
numHam = numEmail - numSpam  
testSpamIdx = sample(numSpam, size = floor(numSpam/3))  
testHamIdx = sample(numHam, size = floor(numHam/3))  
  
# Use the test indices to select word lists for test messages.  
# Use training indices to select word lists for training messages.  
testMsgWords = c((msgWordsList[isSpam])[testSpamIdx],  
 (msgWordsList[!isSpam])[testHamIdx] )  
trainMsgWords = c((msgWordsList[isSpam])[ - testSpamIdx],   
 (msgWordsList[!isSpam])[ - testHamIdx])  
  
# Create variables indicating which testing and training messages are spam and not.  
testIsSpam = rep(c(TRUE, FALSE),   
 c(length(testSpamIdx), length(testHamIdx)))  
trainIsSpam = rep(c(TRUE, FALSE),   
 c(numSpam - length(testSpamIdx),   
 numHam - length(testHamIdx)))  
  
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 1/2 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)  
 }  
  
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])  
}  
  
readEmail = function(dirName) {  
 # retrieve the names of files in directory  
 fileNames = list.files(dirName, full.names = TRUE)  
 # drop files that are not email  
 notEmail = grep("cmds$", fileNames)  
 if ( length(notEmail) > 0) fileNames = fileNames[ - notEmail ]  
   
 # read all files in the directory  
 lapply(fileNames, readLines, encoding = "latin1")  
}  
  
processHeader = function(header)  
{  
 # modify the first line to create a key:value pair  
 header[1] = sub("^From", "Top-From:", header[1])  
   
 tch = textConnection(header)  
 headerMat = read.dcf(tch, all = TRUE)  
 # close the connection now that we are done reading from it  
 close(tch)  
 headerVec = unlist(headerMat)  
   
 dupKeys = sapply(headerMat, function(x) length(unlist(x)))  
 names(headerVec) = rep(colnames(headerMat), dupKeys)  
   
 return(headerVec)  
}  
  
processAttach = function(body, contentType){  
   
 n = length(body)  
 boundary = getBoundary(contentType)  
   
 bString = paste("--", boundary, sep = "")  
 bStringLocs = which(bString == body)  
 eString = paste("--", boundary, "--", sep = "")  
 eStringLoc = which(eString == body)  
   
 # if the ending boundary is missing, make the end of the file the end of the attachment  
 if (length(eStringLoc) == 0) eStringLoc = n  
   
 # get the locations of the beginning boundary strings for attachments, the ending boundary string,  
 # and the location of the last line of the main body  
 # make sure to handle case of no beginning boundary string  
 if (length(bStringLocs) <= 1) {  
 attachLocs = NULL  
 msgLastLine = n  
 if (length(bStringLocs) == 0) bStringLocs = 0  
 } else {  
 attachLocs = c(bStringLocs[ -1 ], eStringLoc)  
 msgLastLine = bStringLocs[2] - 1  
 }  
   
 # extract the actual body of the message  
 msg = body[ (bStringLocs[1] + 1) : msgLastLine]   
 # append any lines after ending boundary string to the body   
 if ( eStringLoc < n )  
 msg = c(msg, body[ (eStringLoc + 1) : n ])  
   
 # process the attachments if any exist  
 if ( !is.null(attachLocs) ) {  
 # lengths obtained from differences of boundary string locations for attachments  
 attachLens = diff(attachLocs, lag = 1)   
   
 # attachTypes will return the content type, non-ending attachment   
 # boundary locations and ending boundary location  
 # search for Content-Type in attachments portion of the message  
 # if not present, set MIMEType as NA  
 # otherwise extract the value from Content-Type field  
 attachTypes = mapply(function(begL, endL) {  
 CTloc = grep("^[Cc]ontent-[Tt]ype", body[ (begL + 1) : (endL - 1)])  
 if ( length(CTloc) == 0 ) {  
 MIMEType = NA  
 } else {  
 CTval = body[ begL + CTloc[1] ]  
 CTval = gsub('"', "", CTval )  
 MIMEType = sub(" \*[Cc]ontent-[Tt]ype: \*([^;]\*);?.\*", "\\1", CTval)   
 }  
 return(MIMEType)  
 }, attachLocs[-length(attachLocs)], attachLocs[-1])  
 }  
   
 # return a list containing the message body and a data frame containing   
 # the attachment lengths and the types  
 if (is.null(attachLocs)) return(list(body = msg, attachDF = NULL) )  
 return(list(body = msg,   
 attachDF = data.frame(aLen = attachLens,   
 aType = unlist(attachTypes),  
 stringsAsFactors = FALSE)))   
}   
  
processAllEmail = function(dirName, isSpam = FALSE)  
{  
 # read all files in the directory  
 messages = readEmail(dirName)  
 fileNames = names(messages)  
 n = length(messages)  
   
 # split header from body  
 eSplit = lapply(messages, splitMessage)  
 rm(messages)  
   
 # process header as named character vector  
 headerList = lapply(eSplit, function(msg)   
 processHeader(msg$header))  
   
 # extract content-type key (used to figure out if there are attachments there: referenced in a key-value pair)  
 contentTypes = sapply(headerList, function(header)   
 header["Content-Type"])  
   
 # extract the body (data clean up with eSplit)  
 bodyList = lapply(eSplit, function(msg) msg$body)  
 rm(eSplit)  
   
 # which email have attachments (look for cases that we have multi)  
 hasAttach = grep("^ \*multi", tolower(contentTypes))  
   
 # get summary stats for attachments and the shorter body  
 attList = mapply(processAttach, bodyList[hasAttach],   
 contentTypes[hasAttach], SIMPLIFY = FALSE)  
   
 bodyList[hasAttach] = lapply(attList, function(attEl)   
 attEl$body)  
   
 attachInfo = vector("list", length = n )  
 attachInfo[ hasAttach ] = lapply(attList,   
 function(attEl) attEl$attachDF)  
   
 # prepare return structure (listifying header, body, attach, and isspam)  
 emailList = mapply(function(header, body, attach, isSpam) {  
 list(isSpam = isSpam, header = header,   
 body = body, attach = attach)  
 },  
 headerList, bodyList, attachInfo,   
 rep(isSpam, n), SIMPLIFY = FALSE )  
 names(emailList) = fileNames  
   
 invisible(emailList)  
}  
  
  
##Q8  
computeFreqs2 =  
 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 1/2 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", ] =   
 wordTable["spam",]/wordTable["ham", ]  
 wordTable["absentLogOdds", ] =   
 (1 - wordTable["spam", ])/(1 -wordTable["ham", ])  
   
 invisible(wordTable)  
 }  
  
computeMsgLLR2 = 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  
   
 log(prod(freqTable["presentLogOdds", present])) +  
 log(prod(freqTable["absentLogOdds", !present]))  
}  
  
trainTable = computeFreqs2(trainMsgWords, trainIsSpam)  
testLLR = sapply(testMsgWords, computeMsgLLR2, trainTable)  
  
#time taken for testLLR2 (my new function)  
system.time(sapply(testMsgWords, computeMsgLLR2, trainTable))

## user system elapsed   
## 139.557 1.085 140.536

#time taken for testLLR (function written in class)  
system.time(sapply(testMsgWords, computeMsgLLR, trainTable))

## user system elapsed   
## 124.438 1.009 125.562

tapply(testLLR, testIsSpam, summary)

## $`FALSE`  
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## -Inf -181 -155 -136 Inf   
##   
## $`TRUE`  
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## -116 -45 4 Inf 98 Inf

#self defined accuracy function to evaluate strength of models  
accuracy = function(LLRVals, testIsSpam){  
 classify = LLRVals > 0 #True => is spam  
 correct = sum(classify== testIsSpam)  
 return(correct / length(testIsSpam))  
}  
  
accuracy(testLLR, testIsSpam)

## [1] 0.8735558

#Old model accuracy: 0.9396662  
#New model accuracy: 0.8735558  
  
##Q9  
computeFreqs3 =  
 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 1/2 to counts for words in message  
 counts.spam = table(unlist(lapply(wordsList[spam], unique)))  
 wordTable["spam", bow] = counts.spam[bow] + .5  
   
 # Similarly for ham messages  
 counts.ham = table(unlist(lapply(wordsList[!spam], unique)))   
 wordTable["ham", bow] = counts.ham[bow] + .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)  
 }  
  
#By changing the bow, line 'wordTable["spam", names(counts.spam)] = counts.spam + .5' gives an error as the there are an unequal number of rows in the matrix  
  
##Q13  
isYelling = function(msg) {  
 if ( "Subject" %in% names(msg$header) ) {  
 # if subject exists, remove non-alpha characters  
 el = gsub("[^[:alpha:]]", "", msg$header["Subject"])  
 if (nchar(el) > 0)  
 # delete all upper case characters and see if there are any characters left  
 nchar(gsub("[A-Z]", "", el)) < 1  
 else   
 FALSE  
 } else   
 NA  
}  
  
isYelling2 = function(msg) {  
 body = msg$body  
 if (length(body) > 0) {  
 # if body exists, remove non-alpha characters  
 lines = gsub("[^[:alpha:]]", "", body)  
   
 #removes all empty lines  
 lines = lines[nchar(lines) > 0]  
   
 #delete all upper case characters  
 lowercaselines = gsub("[A-Z]", "", lines)  
   
 #count number of lines that now have zero characters and return that value as the result of the function  
 count = length(lowercaselines[nchar(lowercaselines) < 1])  
   
 #adding percentages  
 percentage = count / length(body)  
   
 c(count, percentage)  
 #percentage  
   
 } else   
 NA  
}  
  
##Q14  
isRe = function(msg) {  
 "Subject" %in% names(msg$header) &&  
 length(grep("^[ \t]\*Re:", msg$header[["Subject"]])) > 0  
}  
  
isRe2 = function(msg) {  
 "Subject" %in% names(msg$header) && (  
 length(grep("^[ \t]\*Re:", msg$header[["Subject"]])) > 0   
 || length(grep("^[ \t]\*Fwd: Re:", msg$header[["Subject"]])) > 0 )  
}  
  
isRe3 = function(msg) {  
 "Subject" %in% names(msg$header) &&   
 length(grep("\*Re:", msg$header[["Subject"]])) > 0   
   
}  
  
emailStruct = mapply(processAllEmail, fullDirNames, isSpam = rep( c(FALSE, TRUE), 3:2))

## Warning in FUN(X[[i]], ...): incomplete final line found on '/home/  
## student/container-data/RDataScience/SpamAssassinMessages/messages/hard\_ham/  
## 00228.0eaef7857bbbf3ebf5edbbdae2b30493'

## Warning in FUN(X[[i]], ...): incomplete final line found on '/home/  
## student/container-data/RDataScience/SpamAssassinMessages/messages/hard\_ham/  
## 0231.7c6cc716ce3f3bfad7130dd3c8d7b072'

## Warning in FUN(X[[i]], ...): incomplete final line found on '/home/  
## student/container-data/RDataScience/SpamAssassinMessages/messages/hard\_ham/  
## 0250.7c6cc716ce3f3bfad7130dd3c8d7b072'

emailStruct = unlist(emailStruct, recursive = FALSE)  
  
length(lapply(emailStruct, isRe) == TRUE)

## [1] 9348

length(lapply(emailStruct, isRe2) == TRUE)

## [1] 9348

length(lapply(emailStruct, isRe3) == TRUE)

## [1] 9348