Homework3.R

student

Fri Feb 12 18:14:44 2016

##CS 480 Homework 3  
##Neeraj Asthana (nasthan2)  
  
library(tm)

## Loading required package: NLP

#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)  
  
##Q2  
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))  
 }  
  
# This function extracts the words from a message and excludes the   
# specified stopwords. invisible avoids showing the result, which might be large.  
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)  
}  
  
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'

# See how many messages we have in each directory.  
numMsgs = sapply(msgWordsList, length)  
numMsgs

## [1] 5051 1400 500 1000 1397

# Define isSpam based on directory the message came from.  
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)  
}  
  
trainTable = computeFreqs(trainMsgWords, trainIsSpam)  
  
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])  
}  
  
testLLR = sapply(testMsgWords, computeMsgLLR, trainTable)  
  
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  
  
##Q3  
#The string manipulation functions in R can be used instead of regular expression functions for finding,   
#changing, extracting substrings from strings. These functions include: strsplit() to divide a string up   
#into pieces, substr() to extract a portion of a string, paste() to glue together multiple strings, and   
#nchar() which returns the number of characters in a string. Write your own version of getBoundary()   
#(see the section called “Removing Attachments from the Message Body”) using these functions to extract   
#the boundary string from the Content-Type. Debug your function with the messages in sampleEmail.  
myGetBoundary = function(header){  
 boundaryIdx = grep("boundary=", header)  
 line = header[boundaryIdx]  
   
 #remove all whitespace and quotes  
 line = gsub('"', "", line)  
 line = gsub(' ', "", line)  
   
 #split string to only include portion after "boundary="  
 line = strsplit(line, "boundary=")  
 line = unlist(line)[2]  
   
 #remove semicolon if it exists  
 line = unlist(strsplit(line, ";"))[1]  
}  
  
  
##Q6  
#Try to improve the text cleaning in findMsgWords() of the section called “Extracting Words from a Message   
#Body” by stemming the words in the messages. That is, make plural words singular and reduce present and   
#past tenses to their root words, e.g., run, ran, runs, running all have the same “stem”. To do this, use   
#the stemming functions available in the text mining package tm. Incorporate this stemming process into   
#the findMsgWords() function. Then recreate the vectors of words for all the email and see if the   
#classification improves.  
dropAttach = 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  
 if (length(eStringLoc) == 0)   
 return(body[ (bStringLocs[1] + 1) : (bStringLocs[2] - 1)])  
   
 # typical case of well-formed email with attachments  
 # grab contents between first two beginning boundary strings and   
 # add lines after ending boundary string  
 n = length(body)  
 if (eStringLoc < n)   
 return( body[ c( (bStringLocs[1] + 1) : (bStringLocs[2] - 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) ])  
}  
  
processAllWordsStemming = 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)  
 msgWordsListStemmed = stemDocument(unlist(msgWordsList))  
 invisible(msgWordsListStemmed)  
}  
  
msgWordsListStemmed = lapply(fullDirNames, processAllWordsStemming,   
 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'

set.seed(418910)  
  
testMsgWords = c((msgWordsListStemmed[isSpam])[testSpamIdx],  
 (msgWordsListStemmed[!isSpam])[testHamIdx] )  
trainMsgWords = c((msgWordsListStemmed[isSpam])[ - testSpamIdx],   
 (msgWordsListStemmed[!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)))  
  
trainTable = computeFreqs(trainMsgWords, trainIsSpam)  
testLLR = sapply(testMsgWords, computeMsgLLR, trainTable)  
accuracy(testLLR, testIsSpam)

## [1] 0.2570603

#stemming accuracy: .2570603