Authors: Justin Beer, Anthony Dalton, Harsha Kommanapalli

Date: March 19, 2018

Spam and Ham Analysis

**Abstract**

In this case study, we learn to how to automatically identify and classify unwanted emails as “spam” and needed emails as “ham”. This is accomplished by using a dataset of over 9000 already classified emails by SpamAssassin [4]. We take this dataset and perform an analysis in order to create a Naïve Bayes model for identification and classification of spam emails. We compare and note Type I and Type II errors of two different models where the difference is in the way we split our train and test datasets, ratio versus k-fold. We will vary one of the parameters: a threshold that achieves a 1% Type I error. We will conclude that the best model is a Naïve Bayes k-fold approach to achieve the lowest Type 1 error rate.

**Introduction**

Since the dawn of the computer age, junk email, also known as spam, has invaded the everyday lives of average citizens. These seemingly innocuous emails about winning the lottery, a cruise, or vacation package along with the ever-increasing demand for donations to African kings, despots, and criminals have led us into interesting times where we now question the very reason and existence of the emails in our inbox. Seemingly, there is no way to stop this torrent of spam from taking over more of our email organization time and creeping further into our lives and minds. However, with the advent of machine learning and classification algorithms, it seems that we are on the precipice, the very verge, of turning the tide of spam away.

**Literature review**

The text we are utilizing, “Data Science in R: A Case Studies Approach to Computational Reasoning and Problem Solving” [1], teaches us how to access, download, import, munge, and analyze the data from SpamAssassin [4]. In addition to the text [1], the code [2],and Professor Owen’s notes [3] in the files section on 2DS provide further direction on how to process, clean up, and get the data ready for analysis and implementation of the Naive Bayes model. They also provide the process for calculating the threshold, errors, and other parameters.

**Methods**

Initially, the emails need to be processed into data on which statistical analysis can be performed. We need to extract the header, subject, and body along with further splitting the words in the body out separately. Also, we need to remove attachments and stop words, which are common words, along with performing stemming, which is changing the words to related words for uniformity. Once we have our data split out we will then be able to add up all the words occurring in a message and find the frequencies of occurrence in ham or spam. We can then take the probability of occurrence of these words as inputs for Naive Bayes approach. Finally, we can implement additional measures of the email such as capitalization in the subject line to further analyze and classify whether the email is a spam or ham message.

In order to ensure the best possible model for spam/ham classification, we will utilize two different approaches to creating our models. The first approach is to use randomly selected emails for the two subsets of data. These will be random such that the proportion of spam in the test and training sets are close and not too far apart. If they are too far apart, such as if the training data has more than the testing data, we will only create a model that overfits our training data but will poorly represent our testing data. We also want at least half of our data in the training set and for our first model we choose to have 2/3 of our data in the training set, and 1/3 in our testing set.

The second model will be a Naïve Bayes model which will also use randomly selected emails for the training and testing sets. However, for this model, we will implement a 5-fold cross-validation and split the training and testing sets proportionally. For each fold, we will take the subset of the training data to use as our testing data thereby creating 5 different training sets and 5 different testing sets.

In order to assess our models, we will estimate the probabilities that a word occurs in the message given whether it is a spam or ham message as a log-likelihood ratio. We will extend these probabilities to our training and test datasets and plot out our Type I and Type II errors to assess the accuracy of our classification of a message as either ham or spam. We are looking to achieve a Type 1 error rate of 1% or less.

Below is how we create our first training and testing data, using randomly selected emails for our 2/3 train and 1/3 test datasets.

# Find number of emails of spam and ham to help in sampling decisions.

numEmail = length(isSpam) numSpam = sum(isSpam) numHam = numEmail - numSpam

set.seed(418910)

# Determine indices of test spam and test ham.

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])

# Create training and test sets.

# All spam messages are first, followed by ham.

testIsSpam = rep(c(TRUE, FALSE), c(length(testSpamIdx),length(testHamIdx)))

# All ham messages are first, followed by spam

testIsHam = rep(c(FALSE, TRUE), c(length(testSpamIdx), length(testHamIdx)))

trainIsSpam = rep(c(TRUE, FALSE), c(numSpam - length(testSpamIdx), numHam - length(testHamIdx)))

trainIsHam = rep(c(FALSE, TRUE), c(numSpam - length(testSpamIdx), numHam - length(testHamIdx)))

# Apply calculation of log likelihood ratio to each of the messages in our test set.

testLLR = sapply(testMsgWords, computeMsgLLR, trainTable)

# Compare the summary statistics of the LLR values for ham/spam in test data.

tapply(testLLR, testIsSpam, summary)

Now that we have created our first test and train datasets we want to plot out the log likelihood ratio for ham and spam messages, refer to the boxplots in Figure 1 below. We can see that there is a good amount of separation of the ham and spam means with an overlap of the 100th percentile of ham with the 25th percentile of spam.

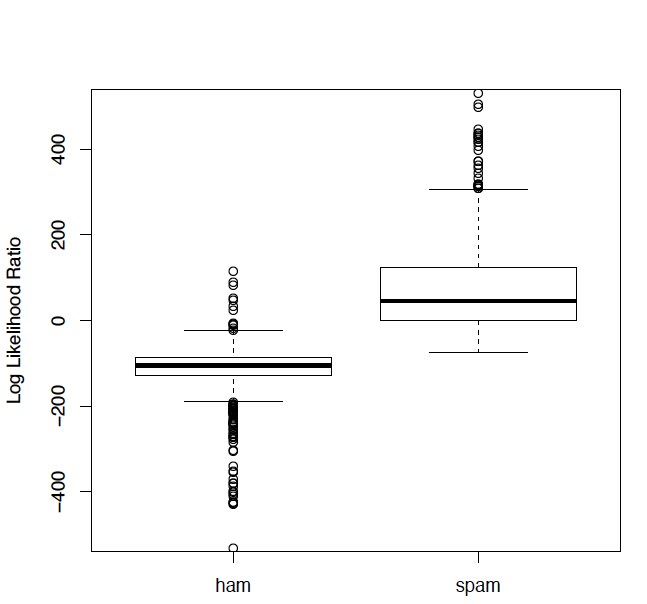


Figure 1. Boxplots of Ham and Spam Log Likelihood Ratio.

We now need to create a function for calculating the Type 1 error rate, which is the rate of misclassifications of ham as spam for a particular threshold value (tau).

# Compute the rate of misclassification of ham as spam for a particular value of tau. This function takes 3 inputs: the value of threshold tau, the vector of # LLR values for the test messages, and the hand-classified type of each message (spam or ham).

typeIErrorRate = function(tau, llrVals, spam) {

classify = llrVals > tau

sum(classify & !spam)/sum(!spam)

}

We also need to create a function for calculating the Type II error rate, which is the rate of misclassification of spam as ham also for a particular threshold value (tau).

typeIIErrorRate = function(tau, llrVals, ham) {

classify = llrVals < tau sum(classify & !ham)/sum(!ham)

}

We also created a vectorized way to calculate the Type I error:

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])

}

Using these functions, we then calculate the tau01 threshold for achieving a 1% Type I error for our first model, which is split 2/3 for training and 1/3 for testing:

xI = typeIErrorRates(testLLR, testIsSpam)

xII = typeIIErrorRates(testLLR, testIsSpam)

# tau01 for Type 1 error <= .01

tau01 = round(min(xI$values[xI$error <= 0.01])) t2 = max(xII$error[ xII$values < tau01 ])

**> #tau01**

**> tau01**

**[1] -23**

In order to visualize this error rate, reference Figure 2 below, we can plot out our Type I and Type II error rates and plot where our Type I Error rate equals 0.01 and the corresponding log likelihood ratio value, which we found is -23.

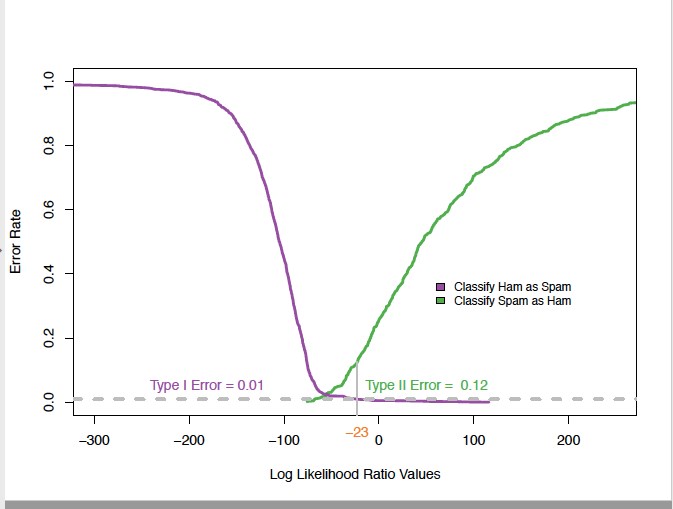


Figure 2. Type I and Type II Error Rates for Classifying Spam and Ham.

Now we will go about to create our second model consisting of a Naïve Bayes utilizing a 5-fold permutation. First, we will split the data into 5 equal-size sets, also known as folds. Here is the code we utilized for this:

k = 5

numTrain = length(trainMsgWords)

# Use the sample() function to permute the indices of the training set

partK = sample(numTrain) tot = k \* floor(numTrain/k)

partK = matrix(partK[1:tot], ncol = k)

For each of our five folds, we will take its corresponding subset from the training data to use as a test set. Then we will use the remaining data as our training set. Then, once again, we will calculate the log likelihood ratio along with our error rates so we can analyze and compare our models.

testFoldOdds = NULL

# organize these permuted indices into 5 equal-size sets, called folds

for (i in 1:k) { foldIdx = partK[ , i]

# For each fold, take the corresponding subset from the training data to use as a ‘testʼset. Use the remaining messages in the training data as the training set.

# Apply functions to estimate the probabilities that a word occurs in a message given it is spam or ham

trainTabFold = computeFreqs(trainMsgWords[-foldIdx], trainIsSpam[-foldIdx])

#use these probabilities to compute the log likelihood ratio for the messages in the training set.

testFoldOdds = c(testFoldOdds, sapply(trainMsgWords[ foldIdx ], computeMsgLLR, trainTabFold))

}

testFoldSpam = NULL

#Pool all of the LLR values from the messages in all of the folds, i.e., from all of the training data

for (i in 1:k) { foldIdx = partK[ , i]

testFoldSpam = c(testFoldSpam, trainIsSpam[foldIdx])

}

# use LLR values from the messages in all of the folds to calculate required error rates

xFoldI = typeIErrorRates(testFoldOdds, testFoldSpam)

xFoldII = typeIIErrorRates(testFoldOdds, testFoldSpam)

# Find threshold tauFoldI tau 01 for to keep type I error for K fold experiment at .01

tauFoldI = round(min(xFoldI$values[xFoldI$error <= 0.01]))

tFold2 = max(xFoldII$error[ xFoldII$values < tauFoldI ])

> #tauFoldI

> tauFoldI

[1] -16

Utilizing the above code we calculate a tau = -16, this is the point in which our Type I error rate = 0.01. We can see the exact intersection for our calculated tau in the below Figure 3.

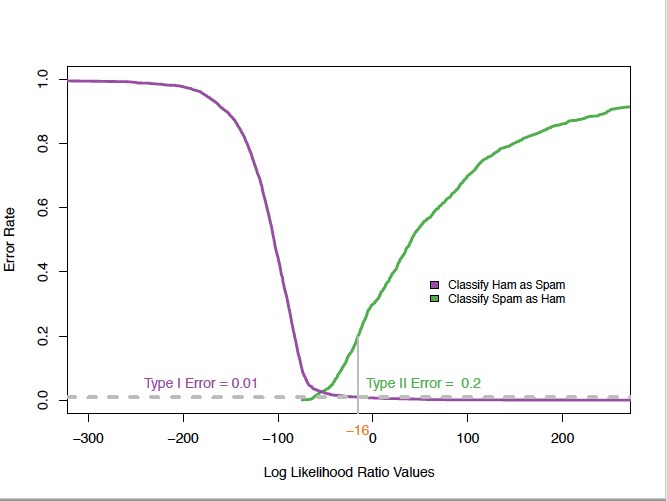


Figure 3. Type I and Type II Error Rates for Classifying Spam and Ham.

We can now apply our threshold of -16 to our original dataset to test out the actual Type I and Type II error rates on all the data.

**# Find type I error for original data set with tau = threshold to keep typeI at .01**

> typeIErrorRate(tau01, testLLR,testIsSpam)

[1] 0.009285714

**# Find type I error for original data set with tau = tauFoldI which is the threshold to keep typeI error of kfold at .01**

> typeIErrorRate(tauFoldI, testLLR,testIsSpam)

[1] 0.007857143

**# Find type II error of original data set with tau = threshold to keep typeI error at .01**

> typeIIErrorRate(tau01, testLLR,testIsHam)

[1] 0.125

**> #find type II error for original data set with tau = tauFoldI which is the threshold to keep typeI error of kfold at .01** > typeIIErrorRate(tauFoldI, testLLR,testIsHam) [1] 0.1598101

**Results**

For our first model where we split out our data into a 2/3 training set and a 1/3 testing set, we found that the log likelihood ratio for a Type 1 Error = 0.01 was -23. For our second model where we used k-fold Naïve Bayes, we found that our log likelihood ratio for a Type 1 error = 0.01 was -16.

Applying our two model’s tau thresholds back to the original data shows us that our first models Type 1 error rate = 0.929% and Type 2 error rate = 12.5%. We also find that our second models Type 1 error rate = 0.786% and Type 2 error rate = 15.981%.

**Conclusion**

Since we want to bring as low as possible the number of misclassifications of actual messages as spam, we find that our second model achieves a better result. The 5-fold cross validation Naïve Bayes model produces a Type 1 error rate of only 0.786% misclassifications of ham as spam and a Type 2 error rate of only 15.981% misclassifications of spam as ham. We recommend going with a Naïve Bayes k-fold implementation method for attempting to classify spam messages.

**Appendix**

**References**

1. Nolan, D., & Lang, D. T. (2015). Data Science in R: A Case Studies Approach to Computational Reasoning and Problem Solving. (Data science in R.) London: CRC Press.
2. Nolan, D., & Lang, D. T. Retrieved March 15, 2018, from http://rdatasciencecases.org/Spam/code.R
3. Professor Owen’s notes in the files section on 2DS.
4. SpamAssassin. Retrieved March 15, 2018, from http://spamassassin.apache.org/old/publiccorpus/

**Complete Code Base**

spamPath = "/Users/Harsha/desktop/SpamAssassin" setwd(spamPath)

list.dirs(spamPath, full.names = FALSE) dirNames = list.files(path = paste(spamPath, sep = .Platform$file.sep))

length(list.files(paste(spamPath, sep = .Platform$file.sep))) #Add path in front of relative directory names to parse through later sapply(paste(spamPath, dirNames, sep = .Platform$file.sep),

function(dir) length(list.files(dir)) ) # how many files are there

fullDirNames = paste(spamPath, dirNames, sep = .Platform$file.sep)

fullDirNames[1]

fileNames = list.files(fullDirNames[1], full.names = TRUE) fileNames[1] indx = c(1^5, 15, 27, 68, 69, 329, 404, 427, 516, 852, 971) fn = list.files(fullDirNames[1], full.names = TRUE)[indx] sampleEmail = sapply(fn, readLines)

sampleEmail msg = sampleEmail[[1]] which(msg == "")[1] #find usual split point position match("", msg) splitPoint = match("", msg) splitPoint

msg[ (splitPoint - 2):(splitPoint + 6) ] #split into header and body header = msg[1:(splitPoint-1)] body = msg[ -(1:splitPoint) ]

splitMessage = function(msg) {

#alternatively you could #splitPoint = match("", msg) splitPoint = which(msg == "")[1] if( is.na(splitPoint)) {splitPoint = 63}

header = msg[1:(splitPoint-1)] body = msg[ -(1:splitPoint) ]

return(list(header = header, body = body))

}

#split sample email to test

sampleSplit = lapply(sampleEmail, splitMessage) sampleSplit

header = sampleSplit[[1]]$header #find content type in email grep("Content-Type", header) grep("multi", tolower(header[46])) header[46]

headerList = lapply(sampleSplit, function(msg) msg$header)

CTloc = sapply(headerList, grep, pattern = "Content-Type")

CTloc

sapply(headerList, function(header) { CTloc = grep("Content-Type", header) if (length(CTloc) == 0) return(NA)

CTloc

})

#find if message has attachment using 'Content type' in header

hasAttach = sapply(headerList, function(header) { CTloc = grep("Content-Type", header) if (length(CTloc) == 0) return(FALSE)

grepl("multi", tolower(header[CTloc]))

})

hasAttach

#if you need to find the boundary string to extract and remove the attachments header = sampleSplit[[6]]$header boundaryIdx = grep("boundary=", header) header[boundaryIdx] sub(".\*boundary=\"(.\*)\";.\*", "\\1", header[boundaryIdx])

header2 = headerList[[9]] boundaryIdx2 = grep("boundary=", header2) header2[boundaryIdx2] sub('.\*boundary="(.\*)";.\*', "\\1", header2[boundaryIdx2])

boundary2 = gsub('"', "", header2[boundaryIdx2]) sub(".\*boundary= \*(.\*);?.\*", "\\1", boundary2) boundary = gsub('"', "", header[boundaryIdx]) sub(".\*boundary= \*(.\*);?.\*", "\\1", boundary)

sub(".\*boundary= \*([^;]\*);?.\*", "\\1", boundary)

#function to get boundary getBoundary = function(header) { boundaryIdx = grep("boundary=", header) boundary = gsub('"', "", header[boundaryIdx])

gsub(".\*boundary= \*([^;]\*);?.\*", "\\1", boundary)

}

sampleSplit[[6]]$body

boundary = getBoundary(headerList[[15]]) body = sampleSplit[[15]]$body

bString = paste("--", boundary, sep = "") bStringLocs = which(bString == body) bStringLocs

eString = paste("--", boundary, "--", sep = "") eStringLoc = which(eString == body) eStringLoc

#msg = body[ (bStringLocs[1] + 1) : (bStringLocs[2] - 1)] tail(msg)

#msg = c(msg, body[ (eStringLoc + 1) : length(body) ]) tail(msg)

head(sampleSplit[[1]]$body) msg = sampleSplit[[3]]$body head(msg)

tolower(gsub("[[:punct:]0-9[:blank:]]+", " ", msg)) msg[ c(1, 3, 26, 27) ]

cleanMsg = tolower(gsub("[[:punct:]0-9[:blank:]]+", " ", msg)) cleanMsg[ c(1, 3, 26, 27) ]

words = unlist(strsplit(cleanMsg, "[[:blank:]]+")) words = words[ nchar(words) > 1 ]

# Drop common small words, aka stop words library(tm) stopWords = stopwords()

cleanSW = tolower(gsub("[[:punct:]0-9[:blank:]]+", " ", stopWords))

SWords = unlist(strsplit(cleanSW, "[[:blank:]]+")) SWords = SWords[ nchar(SWords) > 1 ] stopWords = unique(SWords) head(stopWords)

words = words[ !( words %in% stopWords) ] head(words)

#process all words in the function after splitting message into header, body and finding stopwords and actual message words for calculating probabilities processAllWords = function(dirName, stopWords)

{

# read all files in the directory fileNames = list.files(dirName, full.names = TRUE) fileNames

# drop files that are not email, i.e., cmds notEmail = grep("cmds$", fileNames) notEmail if ( length(notEmail) > 0) fileNames = fileNames[ - notEmail ] messages = lapply(fileNames, readLines, encoding = "latin1")

# split header and body

emailSplit = lapply(messages, splitMessage) emailSplit

# put body and header in own lists

bodyList = lapply(emailSplit, function(msg) msg$body) emailSplit

headerList = lapply(emailSplit, function(msg) msg$header) headerList 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(dropAttach2, bodyList[hasAttach],

boundaries, SIMPLIFY = FALSE) # extract words from body

msgWordsList = lapply(bodyList, findMsgWords, stopWords) invisible(msgWordsList)

}

# create function `dropAttach`, which gets rid of the the attachment dropAttach2 = 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) ])

}

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)

}

# Extract into functions cleanText = function(msg) { tolower(gsub("[[:punct:]0-9[:space:][:blank:]]+", " ", msg))

}

fullDirNames

#find words list for all emails in all directories msgWordsList = lapply(fullDirNames, processAllWords, stopWords = stopWords)

#find length of message words list

numMsgs = sapply(msgWordsList, length) numMsgs

#create logical vectors based on directory names - first three dirs are spam, other two are ham

isSpam = rep(c(FALSE, FALSE, FALSE, TRUE, TRUE), numMsgs) isHam = rep(c(TRUE, TRUE, TRUE, FALSE, FALSE), numMsgs)

#flatten all lists into one list msgWordsList = unlist(msgWordsList, recursive = FALSE)

#find number of emails of spam and ham to help in sampling decisions numEmail = length(isSpam) numSpam = sum(isSpam) numHam = numEmail - numSpam

set.seed(418910)

#determine indices of test spam and test ham 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])

#create training and test sets

#all spam messages are first, followed by ham.

testIsSpam = rep(c(TRUE, FALSE), c(length(testSpamIdx), length(testHamIdx)))

#all ham messages are first, followed by spam testIsHam = rep(c(FALSE, TRUE), c(length(testSpamIdx), length(testHamIdx)))

trainIsSpam = rep(c(TRUE, FALSE),

c(numSpam - length(testSpamIdx), numHam - length(testHamIdx)))

trainIsHam = rep(c(FALSE, TRUE),

c(numSpam - length(testSpamIdx), numHam - length(testHamIdx)))

#find bag of words

bow = unique(unlist(trainMsgWords)) length(bow)

#find count of spam words spamWordCounts = rep(0, length(bow)) names(spamWordCounts) = bow

#find only unique spam words

tmp = lapply(trainMsgWords[trainIsSpam], unique)

tt = table( unlist(tmp) ) spamWordCounts[ names(tt) ] = tt

#find count of ham words hamWordCounts = rep(1, length(bow)) names(hamWordCounts) = bow

#find only unique ham words

tmp2 = lapply(trainMsgWords[trainIsHam], unique)

tt2 = table( unlist(tmp2) ) hamWordCounts[ names(tt2) ] = tt2

#find probability of spamwords spamWordProbs = (spamWordCounts + 0.5) / (sum(trainIsSpam) + 0.5) hamWordProbs = (hamWordCounts + 0.5) / (sum(trainIsHam) + 0.5) log(spamWordProbs) - log(hamWordProbs) log(1 - spamWordProbs) - log(1 - hamWordProbs)

#computefrequencieswith log odds 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 computation of frequencies to training set

trainTable = computeFreqs(trainMsgWords, trainIsSpam)

newMsg = testMsgWords[[1]] # test model against a message

newMsg = newMsg[!is.na(match(newMsg, colnames(trainTable)))] # drop words not in the bag of words, if any such

present = colnames(trainTable) %in% newMsg #locate the columns in the frequency table that contain the words

sum(trainTable["presentLogOdds", present]) + sum(trainTable["absentLogOdds", !present]) #compute the log of the ratio of the

probability a message is spam versus ham

newMsg = testMsgWords[[ which(!testIsSpam)[1] ]] newMsg = newMsg[!is.na(match(newMsg, colnames(trainTable)))] present = (colnames(trainTable) %in% newMsg) sum(trainTable["presentLogOdds", present]) + sum(trainTable["absentLogOdds", !present])

# when you compute log of the ratio of the probability, negative values indicate ham, positive values indicate spam, with magnitude indicating higher chance # calculate the log likelihood ratio (LLR) for all of the test messages 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])

}

#apply calculation of log likelihood ratio function to each of the messages in our test set with testLLR = sapply(testMsgWords, computeMsgLLR, trainTable)

#compare the summary statistics of the LLR values for the ham and spam in the test data with tapply(testLLR, testIsSpam, summary)

pdf("SP\_Boxplot.pdf", width = 6, height = 6) spamLab = c("ham", "spam")[1 + testIsSpam]

boxplot(testLLR ~ spamLab, ylab = "Log Likelihood Ratio",

# main = "Log Likelihood Ratio for Randomly Chosen Test Messages", ylim=c(-500, 500))

dev.off()

#We see from these statistics and the boxplots in the figure that there is a good deal of separation of the ham and spam.

#TypeII error rate needs to be calculated using IsHam

#computes the rate of misclassification of spam as ham for a particular value of tau . This function takes 3 inputs: the value of threshold tau , the vector of

# LLR values for the test messages, and the hand-classified type of each message (spam or ham).

typeIIErrorRate = function(tau, llrVals, ham)

{

classify = llrVals < tau sum(classify & !ham)/sum(!ham)

}

# compute the rate of misclassification of ham as spam for a particular value of tau . This function takes 3 inputs: the value of threshold tau , the vector of

# LLR values for the test messages, and the hand-classified type of each message (spam or ham).

typeIErrorRate = function(tau, llrVals, spam)

{ classify = llrVals > tau

sum(classify & !spam)/sum(!spam)

}

typeIErrorRate(0, testLLR,testIsSpam) typeIErrorRate(-20, testLLR,testIsSpam)

#vectorized way to compute the Type I errors 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]) }

#vectorized way to compute the Type II errors 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])

}

xI = typeIErrorRates(testLLR, testIsSpam) xII = typeIIErrorRates(testLLR, testIsSpam)

#tau01 for Type 1 error <= .01

tau01 = round(min(xI$values[xI$error <= 0.01]))

#tau01 tau01 t2 = max(xII$error[ xII$values < tau01 ])

#t2

t2

#Plot results of experiment

pdf("LinePlotTypeI+IIErrors.pdf", width = 8, height = 6)

library(RColorBrewer)

cols = brewer.pal(9, "Set1")[c(3, 4, 5)]

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")

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(tau01, side = 1, line = 0.5, at = tau01, col = cols[3]) segments(x0 = tau01, y0 = -.50, x1 = tau01, y1 = t2, lwd = 2, col = "grey")

text(tau01, 0.05, pos = 4, paste("Type II Error = ", round(t2, digits = 2)), col = cols[1])

dev.off() k = 5

numTrain = length(trainMsgWords)

#Use the sample() function to permute the indices of the training set partK = sample(numTrain) tot = k \* floor(numTrain/k) partK = matrix(partK[1:tot], ncol = k)

testFoldOdds = NULL

# organize these permuted indices into 5 equal-size sets, called folds for (i in 1:k) { foldIdx = partK[ , i]

# For each fold, take the corresponding subset from the training data to use as a ‘testʼset.

Use the remaining messages in the training data as the training set.

# Apply functions to estimate the probabilities that a word occurs in a message given it is spam or ham trainTabFold = computeFreqs(trainMsgWords[-foldIdx], trainIsSpam[-foldIdx])

#use these probabilities to compute the log likelihood ratio for the messages in the training set.

testFoldOdds = c(testFoldOdds,

sapply(trainMsgWords[ foldIdx ],

computeMsgLLR, trainTabFold))

}

testFoldSpam = NULL

#Pool all of the LLR values from the messages in all of the folds, i.e., from all of the training data for (i in 1:k) { foldIdx = partK[ , i]

testFoldSpam = c(testFoldSpam, trainIsSpam[foldIdx])

}

# use LLR values from the messages in all of the folds to calculate required error rates xFoldI = typeIErrorRates(testFoldOdds, testFoldSpam) xFoldII = typeIIErrorRates(testFoldOdds, testFoldSpam)

# threshold tauFoldI tau 01 for to keep type I error for K fold experiment at .01 tauFoldI = round(min(xFoldI$values[xFoldI$error <= 0.01])) tFold2 = max(xFoldII$error[ xFoldII$values < tauFoldI ])

#tauFoldI tauFoldI #tFold2 tFold2

pdf("LinePlotTypeI+IIErrors\_Kfold.pdf", width = 8, height = 6)

library(RColorBrewer)

cols = brewer.pal(9, "Set1")[c(3, 4, 5)]

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")

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=0.01, col ="grey", lwd = 3, lty = 2) text(-250, 0.05, pos = 4, "Type I Error = 0.01", col = cols[2])

mtext(tauFoldI, side = 1, line = 0.5, at = tauFoldI, col = cols[3]) segments(x0 = tauFoldI, y0 = -.50, x1 = tauFoldI, y1 = tFold2, lwd = 2, col = "grey")

text(tauFoldI, 0.05, pos = 4, paste("Type II Error = ", round(tFold2, digits = 2)), col = cols[1])

dev.off()

#tau01 tau01 #tauFoldI tauFoldI #find type I error for original data set with tau = threshold to keep typeI at .01 typeIErrorRate(tau01, testLLR,testIsSpam)

#find type I error for original data set with tau = tauFoldI which is the threshold to keep typeI error of kfold at .01

typeIErrorRate(tauFoldI, testLLR,testIsSpam)

#find type II error of original data set with tau = threshold to keep typeI error at .01 typeIIErrorRate(tau01, testLLR,testIsHam)

#find type II error for original data set with tau = tauFoldI which is the threshold to keep typeI error of kfold at .01

typeIIErrorRate(tauFoldI, testLLR,testIsHam)