# Abstract

In this analysis, we are going to create a Naïve Bayes model to classify words from emails as to whether they are more likely to appear in solicited(ham) or unsolicited (spam) messages. The dataset is a sample of emails that are already broken up into ham or spam. The body of the emails will be extracted into "bags of words" and if a word has appeared in a spam or ham in the past we will give it a log likelihood ratio (LLR) as to whether the word is considered ham or spam. A critical value of tau will be calculated such that depending on whether a word has an LLR above or below tau it will be considered spam or ham, respectively. 1/3 of the messages will be separated into a test dataset to verify the accuracy of our final model and the other 2/3 will be used to create our model. K-fold cross-validation will be used to calculate an optimal value of tau, where the training set is separated into k equal parts or folds, for k iterations. After testing multiple values of k, we find that 5-fold cross validation gives us the most consistent results with a value of -37 for τ (tau).

# Introduction

Email has become every day, indispensable part of our lives. We use it in our personal lives to communicate with friends and family. We use in our business lives to communicate with customers, fellow employees, and business partners. Email has become the de-facto method of communication in the modern technological world.

Email has been around in one form or another for a long time. Its emergence in networked environments was most visibly marked by Ray Tomlinson in 1971 when he formulated the now iconic split between mailbox identifier and hostname as in "user@hostname" on the ARPANET [1]. Once email became common, people started receiving unsolicited email. We’ve come to call unsolicited email spam. The use of the word spam is a reference to a Monty Python sketch in which a waitress is talking to a table of Vikings. When the waitress says “spam”, the Vikings to start repeating “spam, spam, spam” so the person talking can’t be heard. Hence spam refers to junk email that drowns out all important messages, so they aren’t noticed.

Our work in this paper is the develop a machine learning technique that can detect spam. Detecting spam has become an important aspect of security in a computer network environment. Some spam is simply annoying or meaningless email to the user. However, spam emails are also used to send and proliferate viruses, worms, and trojans. Removing this threat is one of the important functions of spam email filters.

The technique we employ to identify spam requires identifying keywords that are present in spam messages. We have a collection of known spam emails. We collect the words in the spam emails and categorize them as to the frequency they appear in spam email. We train a model using spam emails and then use classification techniques to identify the log-likelihood the word is from spam or ham.

Before any classification can take place, the emails need to be cleaned and put into a format appropriate for classification. This involves properly setting up our environment, splitting emails into header/body, removing attachments and finally creating a bag of words that can used in classification. This process is detailed in the online videos and is available in the text book *Data Science in R – A Case Studies Approach to Computational Reasoning and Problem Solving* [2].

In our example, we have five directories of emails available on the website <https://spamassassin.apache.org/old/publiccorpus/>. This website contains the data we use. Once we download this, we setup our environment to be able to access the directories easy\_ham, easy\_ham2, hard\_ham, spam and spam\_2.

Our next step is to split each email into two parts, the header and the body. The header contains information relevant to managing the message. Information such as sender, date, subject, content type, routing information and message ID are all contained in the header. The body of the email contains the actual message being sent and any attachments included with the email. The header and body are separated by the first blank line contained in the message. This delineator is used to remove the header from the body of the message.

All attachments must be removed from the body of the message. An attachment can be identified by looking at the MIME (Multipurpose Internet Mail Extensions) type. If the field “mime-type = multipart” is present in the header, we know an attachment is present. Once we have identified an attachment is present, the header field “Content-Type” contains the string delineating the boundary between the message text and the attachment, or between multiple attachments. An issue identified in this set was the field names having different letter case. All letters in the message are converted to lower case to resolve this issue. All data located below this string can be removed from the body.

The body of the message contains the words we want to categorize. The header and attachments removed in previous steps are discarded. There may be potential to use this information to identify spam, but that is beyond the scope of this paper. The individual words need to be isolated before we can categorize them. Isolating words deemed important in identifying spam involves several steps. All letters were converted to lower case when the attachments were removed. Punctuation, numbers, extra white space and tabs are removed from the message and a single white space is inserted between the remaining words. The remaining strings are then converted into a list of words. Any remaining one letter words are deleted. Next the word count is reduced using two steps. The first step is one called ‘Stemming”, where words are modified from the original text. Words in past tense are converted to present tense, plurals are converted to singular. This is not an exact process in simplification, but in this case, we are assuming the results are satisfactory. The second step to further reduce word count is the removal of “Stop Words”. Stop words are words deemed not important in analyzing text and are typically pronouns, prepositions and conjunctions but does include others. R has a library of stop words used in this exercise. The list of stop words provided is cleaned in a similar fashion to the words in the body of the message, where the letters a converted to lower case and punctuation is removed. The list of stop words is reduced to a unique set with no single letter words. Once the list of stop words is ready, these words are removed from the body of the message. This leaves each message reduced to a list of words to be put into the bag of words for evaluation.

# Background

The algorithm used to identify spam is messages is the Naïve Bayes classification. This algorithm uses probabilities to predict the class a message belongs in based on the attributes of the message, with those attributes being the words. This algorithm is based on Bayes’ theorem. Bayes’ theorem can be used to figure out conditional probability of an event. That is, figuring out the probability of an event when we know something about that event. In the case of email, figuring out if a message is spam given we know something about spam email (i.e. the probability of certain words being in spam messages). In mathematical terms Bayes’ theorem is the following:

Equation 1 - Bayes' Theorem

In the above equation, A and B are events such as .

Naïve Bayes for classification says that the conditional probability of one event is independent of the other events. In the spam example the naivety of the algorithm means the probability of one word being in spam is independent of other words being present. This may not be true as some words are likely associated with other words, but naivety assumption helps simplify the math involved, and in reality, provides predictions that are accurate enough to justify the simplification.

Building the model requires a training set of data, and then a set of data to test the model against. When no specific test data set is provided, one method to build and then test a model is to break the data set into two pieces. Typically, 80%-90% of the data is used to train the model with the remainder left as the test set. During the training of model, a k-fold cross validation is used to train the model. K-fold cross validation involves breaking the training set into *K* equal folds. Training is run on K-1 of the portions and the resulting model is tested on the remaining fold. This is repeated *K* times and each resulting model used to build the final model describing the training data set. This model is then run on the results are deleted from the test set and this is used to test the model.

The Naïve Bayes classification can suffer from high bias because it uses a linear function to model data. This means it also has a low variance since a measure of bias and a measure of variance move in opposite directions. Bias in a classification algorithm can be introduced by poor selection of data used for modeling. The errors caused by this bias may have very negative, unintended consequences. Incorrectly classifying items such as was seen in the recent Google image recognition software labeling African Americans as gorillas was said to be caused by a lack of diversity in Silicon Valley tech companies [3]. These loss functions are made up of type 1 and type 2 errors. A type 1 error occurs when the null hypotheses is rejected incorrectly. For example, if a test shows that an email is spam when in fact the email is not spam. A type 2 error occurs when we do not accurately reject an incorrect null hypothesis. An example of a type 2 error would be when the test for spam email does not accurately detect spam. Both of these errors can have serious consequences depending on the test where the error occurs. In the example of spam, failing to detect spam, or a type 2 error, would be more serious than incorrectly classifying an email as spam. Allowing spam through the filter to a user’s inbox may result in that user taking action based on the spam and downloading a virus, or some other action that causes harm to the user’s system or information on the network. In the case of a type 1 error, we can quarantine all email identified as spam and provide the user an opportunity to look at these emails in a safe environment and decide for themselves in the email is spam or ham.

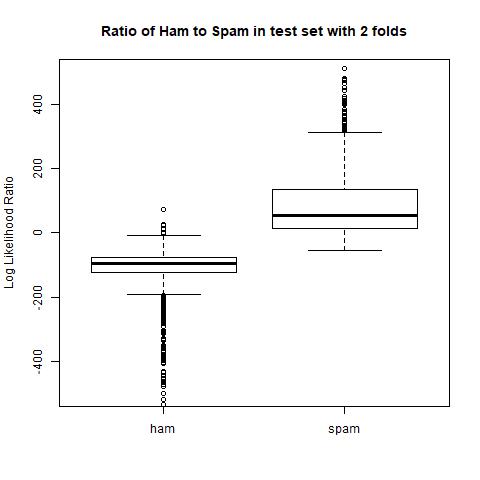
# Method

To identify messages as legitimate (ham) or unwanted (spam) we will use Naïve Bayes to classify the log-likelihood that a given word is from a ham or spam message. As discussed earlier Naïve Bayes takes a naïve approach to computing the probability of by assuming that a given word is a ham or spam word, independent of any other words in the message. [2] The dataset that we had is comprised of the email messages which were broken out into spam or ham messages. Cleansing of the dataset was performed as described in the introduction. The remaining words are then assigned a value as to whether they are more likely found in a spam or ham message. This method of grouping all the words together is known as a bag-of-words method, where we are taking all the words together without considering sequence or sentence breaks. The words are assigned a log likelihood ratio (LLR) whether they are more likely to appear in a ham or spam message, where a negative value indicates the word is more likely ham where a positive value is spam. To efficiently categorize email as ham or spam we need to find the optimum LLR where we can distinguish between the two types of messages; the critical LLR value we are looking for is known as τ.

Before we run analysis on the words we will split the data into a test and training set. As discussed earlier, we split the dataset into training and test datasets, so we can train our model on one dataset and verify the fit on the test dataset. This is done to avoid bias in our final model and prevent overfitting. To train the model we will use what is known as k-fold cross-validation. K-fold cross-validation splits the training dataset into k number of folds of equal size. K-1 folds are used to train the model and 1 of the folds is used to test the accuracy of the model. The loss function, which we are trying to minimize, is keeping the type I error rate at or below 1% and get the lowest type II error rate possible. We used three different sizes of k, 2; 5; 10, to test our model to see which would give the best results with our loss function.

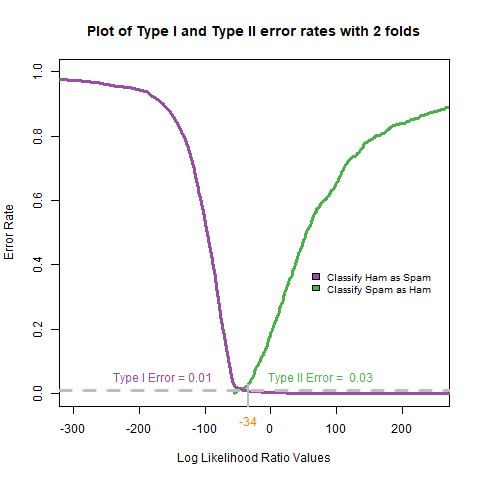
For each iteration of k-folds cross-validation, the model needs to be rerun each time for the size of k. We will analyze the model and fit for each iteration of the model for a given size of k. First, the model was run on k of size two. This is the smallest k possible since we need a test and training set for cross-validation. In Figure 1 we have the box plot of distributions of the LLRs for certain words that are associated with spam or ham messages. We see a lot of overlap between ham and spam messages' LLR scores in the range of 0 to -50, so this will probably be where our final value of τ will fall. Regardless of where we set τ, it appears we will still have type I and type II errors, but our end goal is to minimize both error types. Keeping this in mind we will look at the error rates for our different fitted models.

Figure 1 Boxplot of ham to spam split for LLR



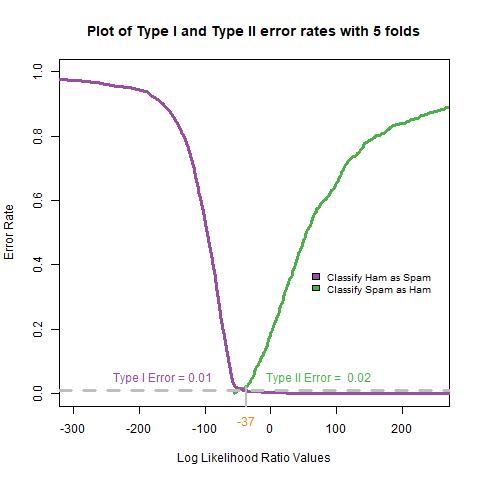
With our plot of the fit for k of size two, we find that we can maintain a type I error rate of 1% with a type II error rate of 8%, as seen in Figure 2. This will be our baseline numbers with a τ of -34, which falls in our estimated range of 0 to -50.

Figure 2 Plot of Type I and Type II error rates for 2 folds



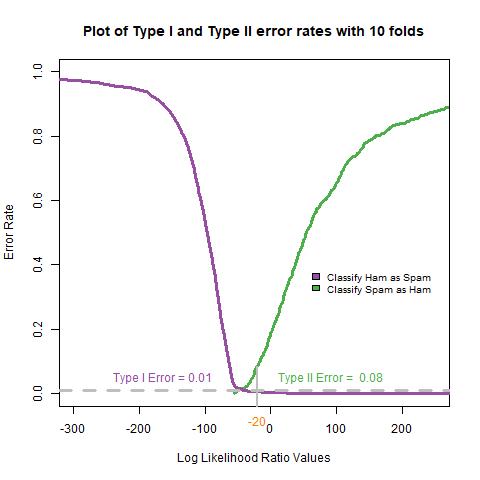
Next, we will run the model on k equal 5 number of folds in Figure 3. This is a significant improvement in our type II error going from 8% to 2% in this model while maintaining a type error of 1%. Now we have a τ of -37 which still falls within our predicted range.

Figure 3 Plot of Type I and Type II error rates for 5 folds



Finally, we run the model using 10 for k folds to see if we can improve the output further. In Figure 4 we see that we can maintain the type I error rate of 1%, but there is an uptick to 8% type II error rate indicating that around 5 folds is the optimal number. This again falls in the range of 0 to -50 for the τ with a value of -20.

Figure 4 Plot of Type I and Type II error rates for 10 folds



# Results

After running the model through multiple iterations of k-fold cross validation we found an optimal number of folds and an optimal value of τ. Initial visual inspection using boxplots for ham and spam of the distribution values for LLR showed that our optimal value of τ was around 0 to -50. From running cross-validation for 2, 5, and 10 folds we found that 5 was the optimal value of folds due to the lowest type II error rate of 2% while maintaining a type I error rate of 1%. Our final value for τ was -37 which we are confident in due to running cross validation and testing that value on the hold our test data.

# Future Work

From our analysis, we found several areas that we could focus on for future work. Naïve Bayes is a competent algorithm for use with spam detection, but advanced uses that focus techniques that deal with natural language processing (NLP) can better classify emails as ham or spam. Advanced algorithms such as long short-term memory (LSTM) can provide context for words in sentences. Any long-term speaker of a language understands that the ordering of words can change the meaning of a sentence and LSTM can better categorize sentences rather than using the bag of words technique. LSTM as a technique relies on the concept of neural networks or deep learning which using weighted values in several matrices to create rules of how to identify certain words from the supplied labels. Deep learning is a very involved topic and training the model requires special equipment and is often considered a black box as it is not as easy to explain the working of the final model. Something like LSTM is beneficial if this model was being rolled out to a production environment and incremental improvements can net large returns.

Another area of research is looking at other parts of the message. For this analysis, only the body of the email was used in the construction of our model. Additional work can be done to consider sources or attachments for email messages when categorizing messages as ham or spam. Frameworks like Lockheed Martin's Cyber Kill Chain which is focused on identifying methods of attack from Advanced Persistent Threats (APT) can be tied in with any classification method used. [4] For example, if most spam is sent without attachments but messages from certain sources start being received with attachments we may let the message in but in a quarantined environment. This strategy would be used if an organization is worried about being attacked by APTs rather than opportunist attackers. Analysis of the source information and attachments requires significantly more work due to how this information is structured. For example, mapping out DNS relationships can involve fields of network and graph theory to understand the interconnected nature of DNS hosts. Opening and analyzing attachments should be done in a secure and isolated environment due to the possibility that the file contains malicious code and preventing the execution of any payload in an uncontrolled environment could release malware into the environment.

# References

|  |  |
| --- | --- |
| [1] | V. G. Cerf, "Spam, spim and spit," *Communications of the ACM,* pp. 39-43, April 2005. |
| [2] | D. N. Lang and D. T. Lang, "Chapter 3 - Using Statistics to Identify Spam," in *Data Science in R - A Case Studies Approach to Computational Reasoning and Problem Solving*, Boca Rotan, CRC Press, 2015, p. 50. |
| [3] | J. Guynn, "Google Photos labled black peole 'gorillas'," USA Today, July 1 2015. [Online]. Available: https://www.usatoday.com/story/tech/2015/07/01/google-apologizes-after-photos-identify-black-people-as-gorillas/29567465/. |
| [4] | E. M. Hutchins, M. J. Cloppert and R. M. Amin, *Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigs and Intrustion Kill Chains,* Lockheed Martin Corporation, 2011. |

# Appendix

R Code

library(XML)

library(changepoint)

library(ggplot2)

setwd('C:\\Users\\Steven\\Dropbox\\School\\MSDS 7333 Quantifying the World\\Session 10\\')

spamPath = "Data"

list.dirs(spamPath, full.names = FALSE)

list.files(path = paste(spamPath, "messages",

sep = .Platform$file.sep))

dirNames = list.files(path = paste(spamPath, "messages",

sep = .Platform$file.sep))

length(list.files(paste(spamPath, "messages", dirNames,

sep = .Platform$file.sep)))

fullDirNames = paste(spamPath, "messages", dirNames,

sep = .Platform$file.sep)

fileNames = list.files(fullDirNames[1], full.names = TRUE)

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)

}

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) ])

}

library(tm)

stopWords = stopwords()

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(dropAttach, bodyList[hasAttach],

boundaries, SIMPLIFY = FALSE)

# extract words from body

msgWordsList = lapply(bodyList, findMsgWords, stopWords)

invisible(msgWordsList)

}

msgWordsList = lapply(fullDirNames, processAllWords,

stopWords = stopWords)

numMsgs = sapply(msgWordsList, length)

numMsgs

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

msgWordsList = unlist(msgWordsList, recursive = FALSE)

#Start of Bayes computation

numEmail = length(isSpam)

numSpam = sum(isSpam)

numHam = numEmail - numSpam

set.seed(123456)

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)))

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)

}

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])

}

typeIErrorRate =

function(tau, llrVals, spam)

{

classify = llrVals > tau

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

}

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])

}

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])

}

list\_k\_folds=list(2,5,10)

for(k\_folds in list\_k\_folds){

metrics <- data.frame(tau=double(),t2=double())

for(i in 1:k\_folds){

fold\_numEmail <- length(trainIsSpam)

fold\_numSpam <- sum(trainIsSpam)

fold\_numHam <- fold\_numEmail - fold\_numSpam

test\_fold\_SpamIdx = sample(fold\_numSpam, size = floor(fold\_numSpam/k\_folds))

test\_fold\_HamIdx = sample(fold\_numHam, size = floor(fold\_numHam/k\_folds))

test\_fold\_MsgWords = c((trainMsgWords[trainIsSpam])[test\_fold\_SpamIdx],

(trainMsgWords[!trainIsSpam])[test\_fold\_HamIdx] )

train\_fold\_MsgWords = c((trainMsgWords[trainIsSpam])[ - test\_fold\_SpamIdx],

(trainMsgWords[!trainIsSpam])[ - test\_fold\_HamIdx])

test\_fold\_IsSpam = rep(c(TRUE, FALSE),

c(length(test\_fold\_SpamIdx), length(test\_fold\_HamIdx)))

train\_fold\_IsSpam = rep(c(TRUE, FALSE),

c(fold\_numSpam - length(test\_fold\_SpamIdx),

fold\_numHam - length(test\_fold\_HamIdx)))

train\_fold\_Table <- computeFreqs(train\_fold\_MsgWords, train\_fold\_IsSpam)

test\_fold\_LLR <- sapply(test\_fold\_MsgWords, computeMsgLLR, train\_fold\_Table)

xI = typeIErrorRates(test\_fold\_LLR, test\_fold\_IsSpam)

xII = typeIIErrorRates(test\_fold\_LLR, test\_fold\_IsSpam)

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

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

metrics<-rbind(metrics,data.frame(tau01,t2))

}

metrics

colMeans(metrics)

testLLR = sapply(testMsgWords, computeMsgLLR, trainTable)

tapply(testLLR, testIsSpam, summary)

jpeg("output/SP\_test\_Boxplot.jpeg")

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))

title('Ratio of Ham to Spam in test set')

dev.off()

xI = typeIErrorRates(testLLR, testIsSpam)

xII = typeIIErrorRates(testLLR, testIsSpam)

tau01 = round(mean(metrics$tau01))

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

jpeg(paste("output/LinePlot\_Test\_fold\_",k\_folds,"\_TypeI+IIErrors.jpeg",sep=""))

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 + 20, 0.05, pos = 4,

paste("Type II Error = ", round(t2, digits = 2)),

col = cols[1])

title(paste('Plot of Type I and Type II error rates with',k\_folds,'folds'))

dev.off()

}