Case Study 3

Gabriel Gonzales
MSDS 7333 – Quantifying the World
10/12/2020

Abstract

In this case study, we will create a Naïve Bayes model to classify words from emails in order to determine whether they are more likely to appear in solicited(ham) or unsolicited (spam) email messages. The dataset is a corpus of emails identified as either 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, a log likelihood ratio (LLR) will be given in order to determine whether the word is considered ham or spam. A critical value of tau will be calculated. Any word that has an LLR above or below tau will be considered spam or ham, respectively. 2/3 of the messages will be used as the training dataset to create and train the model, while the final 1/3 used as the test dataset to verify the accuracy of the final 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 aa vital part of people’s lives. Whether it’s communicating with clients or communicating with loved ones, email currently is the primary method of communication in the modern technological world.

Email has been around in one form or another for almost 40 years. 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 term spam is derived from the 1970 "Spam" sketch of the BBC television comedy series Monty Python's Flying Circus.

The goal of this paper is to develop a machine learning technique that can detect spam. Detecting spam has become an important aspect of security in a computer network environment. Most spam is advertisements and harmless to the user. However, some spam emails are used to send and proliferate viruses, worms, and trojans. Removing this threat is a critical and important function of spam email filters.

The technique employed to identify spam requires identifying keywords that are present in spam messages. The dataset used is a collection of known spam emails. The words are first collected in the spam emails and then categorized by the frequency they appear in spam email. The next step is to 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 the example, five directories of emails are available on the website https://spamassassin.apache.org/old/publiccorpus/. This website contains the data used. The directories within the data are: easy_ham, easy_ham2, hard_ham, spam and spam_2.

The 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. 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. When 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 needed to categorize. The header and attachments removed in previous steps are discarded. Individual words need to be isolated before they can be categorized. Isolating the important needed 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 single-letter words are deleted.

Next, a reduction in words is needed. This is done using a technique called 'Stemming". This is 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, assuming the results are satisfactory. Another step is needed 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 and removed from the body of the message. This leaves each message reduced to a list of words to be evaluated.

Background

The algorithm used to identify the spam is the Naïve Bayes classification. This algorithm uses probabilities to predict the class a message belongs in based on the attributes of the message. The algorithm is based on Bayes' theorem. Bayes' theorem is used to figure out a conditional probability of an event. In the case of emails, determination of ham or spam is only possible given specific knowledge about the spam emails.

Equation 1 - Bayes' Theorem

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

In the above equation, A and B are events such as $A = a\ message\ is\ spam\ and\ B = word\ in\ email$.

Building the model requires the use of a train/test split of the data. Typically 80%-90% of the dataset is used to train the model, while the remaining 10-20% is used to test the model. 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 is used to build the final model.

The Naïve Bayes classification is susceptible to high bias due to its use of a linear function to model data. This also means it has a low variance. Bias in a classification algorithm can be caused by a poor selection of data being used for modeling, leading to errors in the classification. These errors are kown as 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 where the error occurs. Failing to detect spam, or a type 2 error, is more serious than incorrectly classifying an email as spam. Allowing spam through the filter to a user's inbox may result in the downloading of a virus, infecting the user's system or the entire 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.

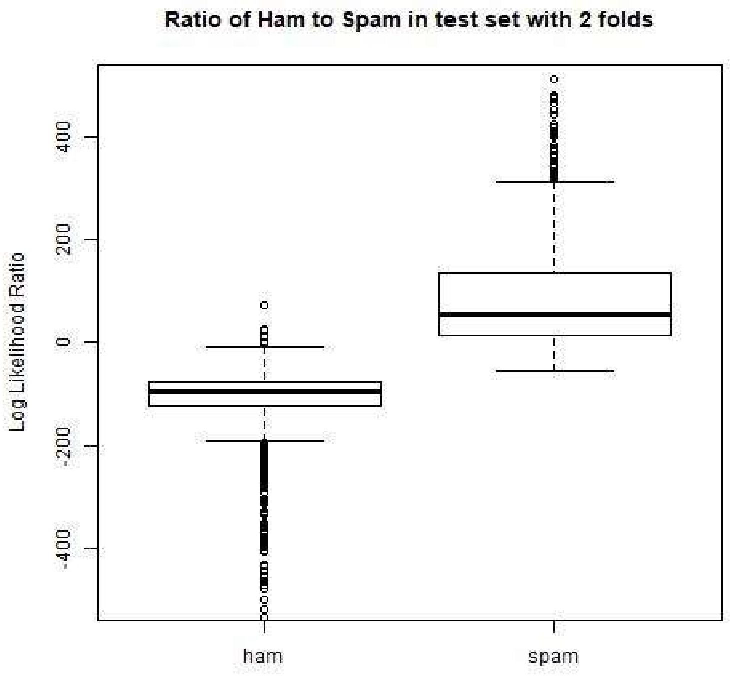
Method

The dataset used is comprised of 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 were then assigned a value as to whether they are more likely to be found in a spam or ham message. This method of grouping all the words together is known as a bag-of-words method, where all the words are placed together without considering sequence or sentence breaks. The words are then assigned a log likelihood ratio (LLR). The LLR helps determine whether they are more likely to appear in a ham or spam message. A negative value indicates the word is ham and a positive value is spam. To efficiently categorize an email, an optimum LLR is needed to further distinguish between the two types of messages; the critical LLR value is known as τ (tau).

As discussed earlier, the data must be split in order to train the model on a training dataset and verify the fit on the test dataset. To train the model, a k-fold cross-validation will be used. 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 needs to be minimized in order to keep the type I error rate at or below 1% and produce the lowest type II error rate possible. Three different sizes of k were used: 2, 5 & 10, to test the model and to determine which would give the best results.

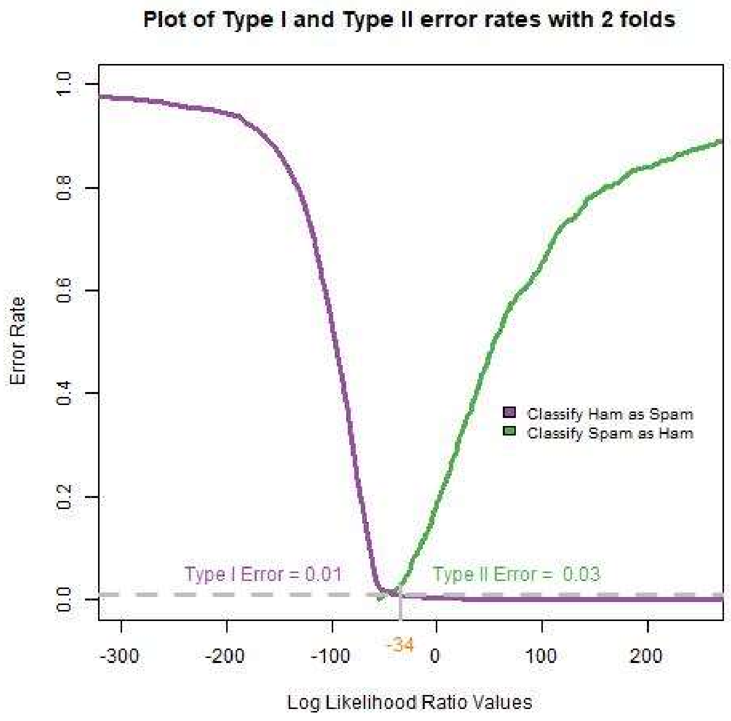
For each iteration of k-folds cross-validation, the model is rerun for each size of k. First, the model was run for k=2. In Figure 1 Boxplot of ham to spam split for LLR, the the boxplots show the distributions of the LLRs for words that are associated with either spam or ham messages. Overlap is evident between the ham and spam messages' LLR scores in the range of 0 to -50, indicating where the final value of τ may fall.

Figure 1 Boxplot of ham to spam split for LLR



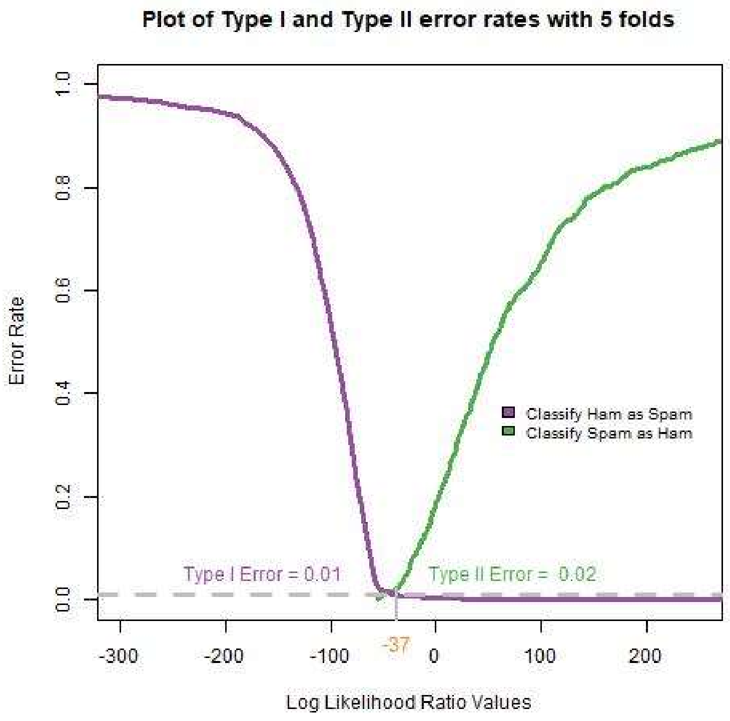
With the plot for k=2, a type I error rate of 1% with a type II error rate of 3%, is seen in Figure 2 Plot of Type I and Type II error rates for 2 folds. A τ of -34 falls in the estimated range of 0 to -50.

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



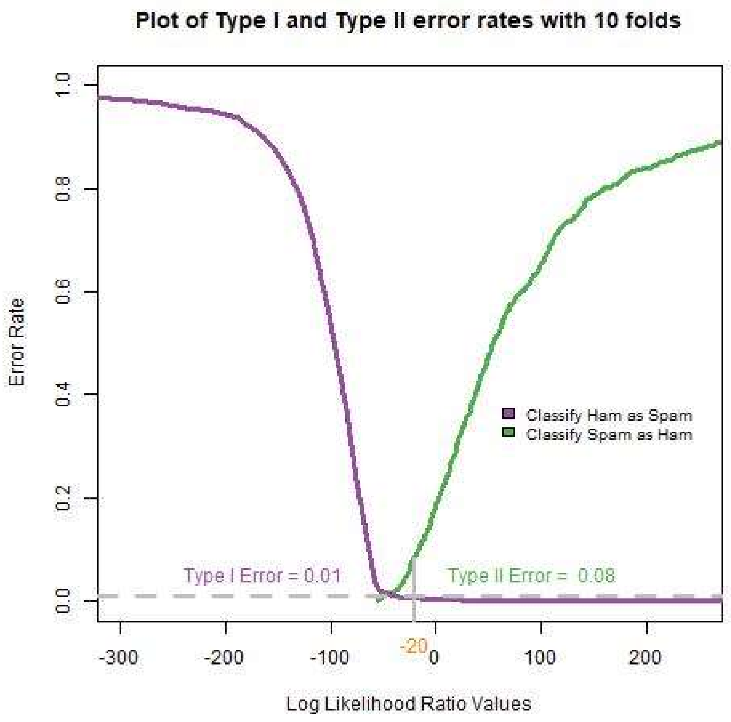
Next is the plot for k=5 in Figure 3 Plot of Type I and Type II error rates for 5 folds. This is a slight improvement in our type II error going from 8% to 2% in this model while maintaining a type error of 1%. A τ of -37 still falls within the predicted range.

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



Finally, the plot for k=10 folds to output can be further improved. In Figure 4 Plot of Type I and Type II error rates for 10 folds, the type I error rate of 1% was maintained, however the type II error rate increase to 8% indicating that k=5 is the optimal number. A τ value of -20 also stays withing the predicted range.

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, an optimal number of folds and an optimal value of τ were determined. Initial data analysis using boxplots for ham and spam LLR distribution values showed that the optimal value of τ was around 0 to -50. From running cross-validation for 2, 5, and 10 folds, 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%.

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.

Appendix

R Code

```
library(XML)
library(changepoint)
library(ggplot2)
library(tm)
library(RColorBrewer)

#LoadData

setwd('C:\\Users\\Gabri/OneDrive\\Documents\\SMU\\Fall 2020 Semester\\DS_7333\\')
spamPath = "Data"

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

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

head(list.files(path = paste(spamPath, "messages", "spam_2",
                         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) ])
}

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)

#pdf("SP_Boxplot.pdf", width = 6, height = 6)
spamLab = c("ham", "spam")[1 + testIsSpam]
boxplot(testLLR ~ spamLab, ylab = "Ratio of Ham to Spam in test set with 2 folds",
     #  main = "Log Likelihood Ratio for Randomly Chosen Test Messages",
       ylim=c(-500, 500))
#dev.off()

xI = typeIErrorRates(testLLR, testIsSpam)
xII = typeIIErrorRates(testLLR, testIsSpam)
tau01 = round(mean(metrics$tau01))
t2 = max(xII$error[ xII$values < tau01 ])


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()
```