Spam Classification

Alex Bank

To get started with this exploration of Spam Classification, here's a one-line command that prints the number of files in each of the five email directories. We clearly have a large set of varied emails to work with.

```
sapply(list.files("data/messages"), function(x) length(list.files(paste0("data/messages/",
    x))))
```

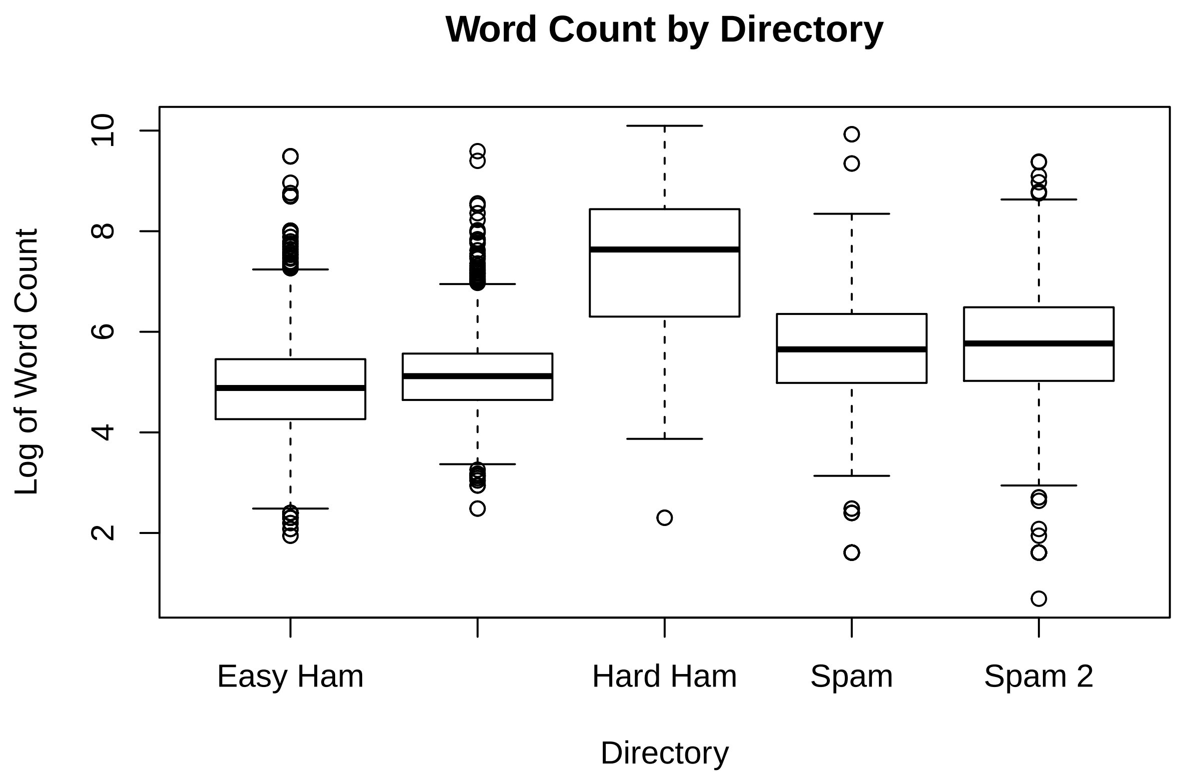
```
##   easy_ham easy_ham_2   hard_ham       spam     spam_2
##       5051       1400        500       1000       1397
```

The next step is to determine which emails have attachments so we can later discard the attachment to focus on the body of the email. This also requires pulling out the email's boundary if it has an attachment. For exapmle, in the "hard_ham" directory, there are 95 emails with attachments. The boundary of the 10th email in this directory with an attachment looks as follows:

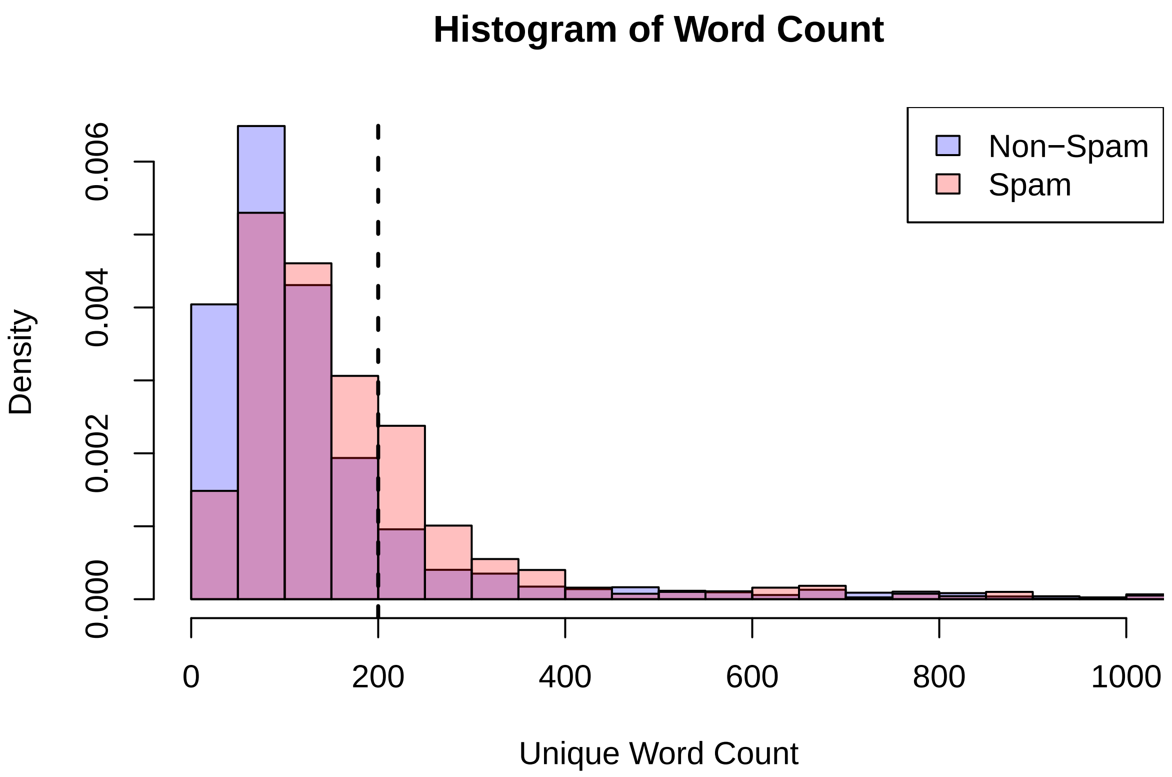
```
## Number of emails with attachments: 95
```

```
## Boundary of the 10th email with attachment: 6700390.1026560924954.JavaMail.root.umsan1
```

Now we are able to find and extract the body of the email. Below we can see the distibution of the log of word counts for each email directory. Clearly emails in the "hard_ham" directory contain the most words on average. The other four directories have a similar mean amount of words; a slight difference is that the spam directories appear to have a greater range of number of words than the easy ham directories.



The above boxplot shows the log of the toal number of words in an email, but for the sake of our classifier we will only consider unique words in the email body. Below is a histogram showing the distribution of the unique number of words for spam and non-spam emails (truncated at 1000 unique words because the outliers are not interesting). Overall, the spam emails tend to have a higher amount of unique words. Non-spam emails—more often than spam emails—have a unique word count between zero and 150, and are less likely to have higher unique word counts. If we were to try to classify emails based only on unique word count, we might set the threshold at 200. Of course this would cause plenty of errors since many non-spam emails have more than 200 unique words and there is no magic threshold where the word count perfectly tells us whether or not the email is spam; the false-positives would be too high to justify classifying emails solely on unique words.



Noting that unique word count will not give us a good enough classifier, we will now build a Naive Bayesian Classifier. On the following page, we will work through the derivation of the Bayes Factor we will use in determining if an email is spam.

Before we use the Bayes Factor, we need to develop the probabilities of each word in the bag of words for our training set. Our training set is composed of two-thirds of the emails in the total set, and the amount of spam vs. non-spam emails is proportional to the total data set. Below we can see that "monday" appears 4.689 times more likely to appear in non-spam vs spam emails, and "buy" is 1.887 times more likely to be in spam emails than non-spam.

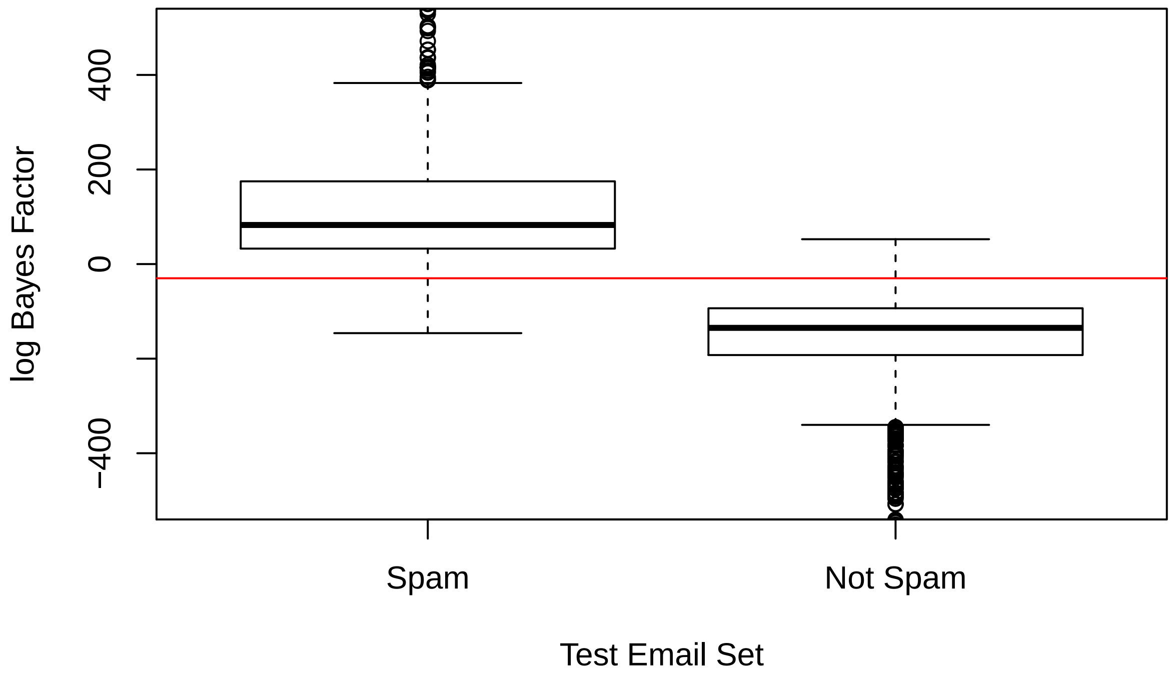
```
round(probPresentHam['monday']/probPresentSpam['monday'], 3)
```

```
## monday
##  4.913
```

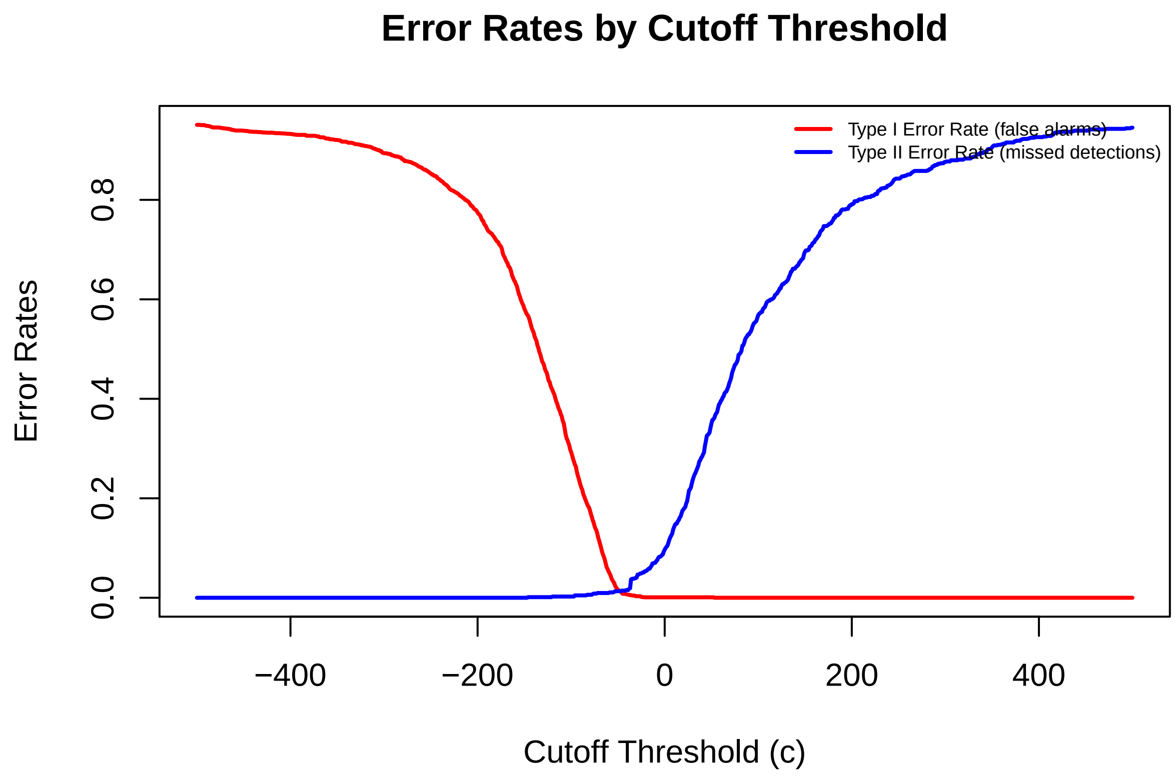
```
round(probPresentSpam['buy']/probPresentHam['buy'], 3)
```

```
##   buy
## 2.151
```

Now we can implement and find the Bayes Factor for each email in our test set. As a matter of note, we are calculating the Bayes Factor based only on words that appear in our bag of words. The following is a side-by-side boxplot comparing the Bayes Factors calculated for spam and non-spam emails in our test data set. Unlike with the unique word count, we can see a clear separation between the distributions of spam and non-spam emails. At first glance, an estimation of -30 (indicated by the red line) seems to be a good cutoff for distinguishing between the emails.



Now we will work to understand and tune the cutoff perameter. Below is a graph showing the Type I error rates (false alarms, or real emails sent to the spam folder) and Type II error rates (missed detections, or spam emails let into the inbox) for a range of cutoff thresholds.



We want to find the intersection of these two plots; that is the c where the error rates are closest to being equal. We find that $c = -47$ is the threshold which minmizes the difference between the error rates. Below we see the best c and the error rates at this value. We get values such that 13 out of 1000 non-spam emails are flaged as spam (type-I) and 13 out of 1000 spam emails get through to our inbox (type-II).

```r
bestCInd <- which.min(abs(t1error-t2error))
firstbestC <- c[bestCInd]
message('Best c: ',c[bestCInd],
        '\nType I Error Rate: ', round(t1error[bestCInd],3),
        '\nType II Error Rate: ', round(t2error[bestCInd],3))
```

```
## Best c: -47
## Type I Error Rate: 0.013
## Type II Error Rate: 0.013
```

The losing 14 potentially important emails to our spam box is not ideal. Instead, we can restrict the cutoff threshold to only those that yield a Type-I error rate of less than 0.1%. With this restriction, we get a Type-I error of 0.08%, but the Type-II error has climbed to 5.13%. This is still seems like an acceptable level given that we now have less than one out of 1000 real emails being marked as spam.

```
## New c: -22
```

```
## Type I Error Rate: 0.00088
## Type II Error Rate: 0.05125
```

Extensions

For the first part of the extension, we will explore cross-validation tuning for our cutoff threshold perameter. Specifically, we will look at two cases. One is where we simply split the training data into two sets, train on one half, and find c using the other half. The second is a k-fold cross-validation; we will split the training data into k subgroups, use *k-1* subgroups to train our data, find c based on the unused subgroup, then repeat this process for k combinations of the different subgroups. The end result will be k different values for c, and we will use the mean of these c values when we run the classification on the test data.

Our prediction is that both of these tuning methods will produce similar results (since they are essentially doing the same thing, just with a different number of subgroups), but the k-fold method will be more robust. The k-fold method takes into account all the words in the training data set, whereas the simple cross-validataion considers only half the available words we have to train on. For our purposes, we will use $k = 5$ and compare performance based on the error rates we get from the test data.

```
## Cutoff and Error Rates using simple Cross-Validation
```

```
## Type-I error with c=-41: 0.007
## Type-II error with c=-41: 0.015
```

```
## Cutoff and Error Rates using k-fold Cross-Validation
```

```
## Type-I error with c=-41.6: 0.007
## Type-II error with c=-41.6: 0.015
```

As expected, the two types of cross-validation yielded very simlar values of c and identical error rates. The interesting piece is that the k-fold method gave us a number that was the mean of five values of c, which is why we ended up with a decimal. While we did not expect to get radically different values for the cutoff threshold, it is nice to see how k-folds accounts for slightly different data than the simple cross-validation. It is also useful to note that the c found by k-folding the data is actually closer to the $c = -47$ that we found as the optimal c for the test data.

Now that we have these two classification systems, we can play with our extractWords function to see if we can improve the performance of the classifier. The newExtractWords function operates very similarly to the old implementation, but the new one looks and eliminates email addresses, gets rid of URLs, and uses the R package 'tm' to stem words. The 'tm' package also provides a list of stop words that we remove from consideration.

Here is an example of the same message split using the two different functions. We can see that the new extract words has eliminated stop words like "it" and "are." Also interesting is that "holidays" has become "holiday" and "official" has become "offici." While the latter does not look promising, as long as the 'tm' package is consistent, this should not be a problem.

```
##  [1] "eirikur"      "hallgrimsson" "wrote"        "it"
##  [5] "official"     "the"          "holidays"     "are"
##  [9] "here"         "for"          "which"        "year"
## [13] "or"           "joe"
```

```
##
```

```
## [1] "eirikur"      "hallgrimsson" "wrote"        "offici"
## [5] "holiday"      "year"         "joe"
```

Now we will apply the k-fold classification to this new list of emails generated by newExtractWords and compare the error rates to the previous ones. I would actually expect the performance to be worse under the newExtractWords function. Non-spam emails tend to have more stop words, which push their Bayes Factor in favor of non-spam classification. Likewise, eliminating email addresses and URLs give spam messages a chance to be classified as non-spam. Lots of spam emails contain strange or unique email addresses and URLs that contain strings not in the bag of words; eliminating these provides less "evidence" that an email is spam.

Despite this, we should still be interested in the results of the new classifications. The resulting c value and error will tell us how good spam emails are at imitating the word usage of non-spam emails when we eliminate email addresses, URLs, stop words, and stem the words.

```
## Cutoff Threshold and Error Rates for newExtractWords
```

```
## Type-I error with c=-37.2: 0.021
## Type-II error with c=-37.2: 0.035
```

As expected, these error rates are higher than what we had for our k-fold classification that used the old extractWords function. What is interesting, though, is that we still only send 2.1% of real emails to spam, and let 3.5% of spam through to to the inbox. This indicates that spam emails are really bad at using the same language as real mail. Even when we get rid of all the strange URLs and stop words, we are classifying emails correctly most of the time.

As a final, fun wrap up, we will look at just three spam and three non-spam emails from my own email and see if we can classify them correctly (using the old extractWords).

```
## Number correctly IDed: 6
## Number incorrectly IDed: 0
```

Thanks for an incredibly interesting semester and a great Winter Study. I will see you in Bayesian in the fall, but until then have a wonderful summer!

Appendix of Code

```r
splitMessage <- function(msg) {
  split <- which(msg == "")[1]
  #if(is.na(split)) print(msg)
  header <- msg[1:(split-1)]
  body <- msg[(split+1):length(msg)]
  return(list(header=header, body=body))
}
hasAttachment <- function(header) {
  ct <- grep("Content-Type:", header, value = TRUE)
  if (length(ct)==0) return(FALSE)
  return (grepl("multipart", tolower(ct)))
  return(FALSE)
}
getBoundary <- function(header) {
  bline <- grep("boundary=", header, value = TRUE)
  if(length(bline) < 1) {
    bline <- grep("BOUNDARY=", header, value = TRUE)
    boundary <- strsplit(bline, "OUNDARY=")
  }
  else {boundary <- strsplit(bline, "oundary=")}
  #if(length(boundary) < 1) boundary <- strsplit(bline, "OUNDARY=")
  #if(length(boundary) < 1) {print(boundary); print(header)}
  boundary <- boundary[[1]][2]
  boundary <- gsub('^[ ";]+', '', boundary)
  boundary <- gsub('[ ";]+$', '', boundary)
  if(grepl('";.*$', boundary)) boundary <- gsub('";.*$', '', boundary)
  return(boundary)
}
extractBodyText <- function(email) {
  msg <- splitMessage(email)
  if (hasAttachment(msg$header)) {
    boundary <- getBoundary(msg$header)
    inds <- grep(boundary, msg$body, fixed = TRUE)
    if (length(inds) == 1) {
      return(msg$body[(inds+1):length(msg$body)])
    }
    if(is.na(inds[1] +1) | is.na(inds[2]-1)) {print(inds); print(boundary)}
    #otherwise assume we get three hits
    return(msg$body[(inds[1]+1):(inds[2]-1)])
  }
  return(msg$body)
}
extractWords <- function(body) {
  #Uses deparse to take care of non-character encodings
  txt <- paste(body, collapse = ' ')
  txt <- deparse(txt)
  txt <- gsub('[[:punct:][:digit:][:blank:]]+', ' ', txt)
  txt <- tolower(txt)
  txt <- unlist(strsplit(txt, ' '))
  txt <- unique(txt)
  txt <- txt[which(nchar(txt)>1)]
  return(txt)
}
readEmailDirectory <- function(dir) {
  return(lapply(list.files(dir, full.names = TRUE), function(x) readLines(x)))
}
emailsAllFunc <- function() {
  toReturn <- list()
  for (dir in list.files("data/messages")) {
    mail <- readEmailDirectory(paste0("data/messages/",dir))
    message('check 1 ', dir)
    body <- lapply(mail, function(m) extractBodyText(m))
    message('check 2 ', dir)
    words <- lapply(body, function(b) extractWords(b))
    toReturn <- c(toReturn, words)
  }
  return(toReturn)
}
isSpamFunc <- function() {
  toReturn <- rep(FALSE, length(list.files('data/messages/easy_ham')))
  toReturn <- c(toReturn, rep(FALSE, length(list.files('data/messages/easy_ham_2'))))
  toReturn <- c(toReturn, rep(FALSE, length(list.files('data/messages/hard_ham'))))
  toReturn <- c(toReturn, rep(TRUE, length(list.files('data/messages/spam'))))
  return(c(toReturn, rep(TRUE, length(list.files('data/messages/spam_2')))))
}
getTrainInds <- function(emailsAll) {
  numTrain <- ceiling(length(emailsAll) * 2 / 3)
  numHam <- floor(.75 * numTrain)
  numSpam <- numTrain-numHam
  nums <- sapply(list.files("data/messages"), function(x) length(list.files(paste0("data/messages/", x)
  totalHam <- sum(nums[1:3])
  totalSpam <- sum(nums[4:5])
  s1 <- sample(1:totalHam, numHam)
  s2 <- sample(1:totalSpam, numSpam) + totalHam
  return(c(s1, s2))
}
bowFunc <- function(emailsTrain) {
  bow <- unique(unlist(emailsTrain))
```

```
}
computeBF <- function(words) {
  words <- words[which(words %in% bow)]

  bf <- sum(logProbPresentSpam[words]) +
    sum(logProbAbsentSpam[-which(names(logProbAbsentSpam) %in% words)]) -
    sum(logProbPresentHam[words]) -
    sum(logProbAbsentHam[-which(names(logProbPresentHam) %in% words)])

  return(bf)
}
plotErrorRates <- function() {
  c <- -500:500
  t1error <- numeric(length(c))
  t2error <- numeric(length(c))
  for (i in 1:length(c)) {
    t1error[i] <- length(which(bfTest[which(!isSpamTest)] > c[i])) / sum(!isSpamTest)
    t2error[i] <- length(which(bfTest[which(isSpamTest)] <= c[i])) / sum(isSpamTest)
  }
  plot(x=c, y=t1error, type='l', lwd=2, col='red', ylab='Error Rates', xlab='Cutoff Threshold (c)',
       main="Error Rates by Cutoff Threshold")
  lines(x=c, y=t2error, col='blue', lwd=2)
  legend('topright', col=c('red','blue'), lty=1, lwd = 2,
         legend=c('Type I Error Rate (false alarms)','Type II Error Rate (missed detections)'),
         bty = 'n', bg = 'gray100',
         cex=.6)
}

#Deliverable 2
hhAttachments <- sapply(list.files("data/messages/hard_ham", full.names = TRUE), function(e) hasAttachme
message('Number of emails with attachments: ', sum(hhAttachments))
index10 <- which(hhAttachments)[10]
message('Boundary of the 10th email with attachment: ',
getBoundary(splitMessage(readLines(list.files('data/messages/hard_ham', full.names = TRUE)[index10]))$he

#Deliverable 3
extractAllWords <- function(body) {
  #Uses deparse to take care of non-character encodings
  txt <- paste(body, collapse = ' ')
  txt <- deparse(txt)
  txt <- gsub('[[:punct:][:digit:][:blank:]]+', ' ', txt)
  txt <- tolower(txt)
  txt <- unlist(strsplit(txt, ' '))
  return(txt)
}

numWordsDir <- function(dir) {
  emails <- list.files(paste0('data/messages/',dir), full.names = TRUE)
  emails <- sapply(emails, readLines)
  emails <- sapply(emails, extractBodyText)
  emails <- sapply(emails, extractAllWords)
  emails <- sapply(emails, length)
  return(emails)
```

```
}


ehWC <- log(numWordsDir('easy_ham'))
eh2WC <- log(numWordsDir('easy_ham_2'))
hhWC <- log(numWordsDir('hard_ham'))
sWC <- log(numWordsDir('spam'))
s2WC <- log(numWordsDir('spam_2'))

boxplot(ehWC, eh2WC, hhWC, sWC, s2WC, names = c('Easy Ham', 'Easy Ham 2', 'Hard Ham', 'Spam', 'Spam 2')
        xlab='Directory', ylab='Log of Word Count', main='Word Count by Directory')

#Deliverable 4
emailsAll <- emailsAllFunc()
isSpam <- isSpamFunc()

numUnique <- lapply(emailsAll, length)
p1 <- hist(unlist(numUnique)[which(isSpam)], freq = FALSE, breaks = seq(0, 10^6, by=50)) #Spam emails
p2 <- hist(unlist(numUnique)[which(!isSpam)], freq = FALSE, breaks = seq(0,10^6,by=50)) #Non-Spam email

plot(p2, col=rgb(0,0,1,1/4), xlim = c(0, 1000), freq = FALSE, xlab = 'Unique Word Count', main = 'Histo
plot(p1, col = rgb(1,0,0,1/4), xlim=c(0,1000), freq=FALSE,add = TRUE)
abline(v=200, lty=2, lwd=2)
legend('topright',fill = c(rgb(0,0,1,1/4),rgb(1,0,0,1/4)), legend = c('Non-Spam','Spam'))

#Deliverable 7
specialInds <- getTrainInds(emailsAll)
emailsTrain <- emailsAll[specialInds]
isSpamTrain <- isSpam[specialInds]
emailsTest <- emailsAll[-specialInds]
isSpamTest <- isSpam[-specialInds]

bow <- bowFunc(emailsTrain)

tSpam <- table(unlist(emailsTrain[which(isSpamTrain)]))
tHam <- table(unlist(emailsTrain[which(!isSpamTrain)]))

bowSpam <- numeric(length(bow)); names(bowSpam) <- bow
bowHam <- numeric(length(bow)); names(bowHam) <- bow

bowSpam[names(tSpam)] <- tSpam
bowHam[names(tHam)] <- tHam

s1 <- sum(isSpamTrain)
probPresentSpam <- sapply(bowSpam, function(w) (w+.01)/(s1+.01))
probAbsentSpam <- 1-probPresentSpam

s2 <- sum(!isSpamTrain)
probPresentHam <- sapply(bowHam, function(w) (w+.01)/(s2+.01))
probAbsentHam <- 1-probPresentHam

logProbPresentSpam <- log(probPresentSpam)
logProbPresentHam  <- log(probPresentHam)
```

```r
logProbAbsentSpam  <- log(probAbsentSpam)
logProbAbsentHam   <- log(probAbsentHam)

bfTest <- sapply(emailsTest, computeBF)
boxplot(bfTest[which(isSpamTest)], bfTest[which(!isSpamTest)], names = c('Spam', 'Not Spam'),
        xlab = 'Test Email Set', ylab = 'log Bayes Factor',
        ylim= c(-500, 500))
abline(h=-30, col='red')

#Deliverable 9
c <- -500:500
t1error <- numeric(length(c))
t2error <- numeric(length(c))
for (i in 1:length(c)) {
  t1error[i] <- length(which(bfTest[which(!isSpamTest)] > c[i])) / sum(!isSpamTest)
  t2error[i] <- length(which(bfTest[which(isSpamTest)] <= c[i])) / sum(isSpamTest)
}

bestCInd <- which.min(abs(t1error-t2error))
firstbestC <- c[bestCInd]
message('Best c: ',c[bestCInd],
        '\nType I Error Rate: ', round(t1error[bestCInd],3),
        '\nType II Error Rate: ', round(t2error[bestCInd],3))

#Deliverable 10
smallIndexes <- which(t1error<.001)
bestIndex <- which(t2error == min(t2error[smallIndexes]))
bestIndex <- bestIndex[length(bestIndex)]
message('New c: ', c[bestIndex])
message('Type I Error Rate: ', round(t1error[bestIndex],5),
        '\nType II Error Rate: ', round(t2error[bestIndex],5))

#Extension Code

#single cross validation

#first we split the training set into two
cvSplitInds <- function(){
  spamInds <- which(isSpamTrain)
  nonSpamInds <- which(!isSpamTrain)

  inds1 <- sample(spamInds, floor(length(spamInds)/2))
  inds2 <- sample(nonSpamInds, floor(length(nonSpamInds)/2))

  return(c(inds1, inds2))
}

cvInds <- cvSplitInds()

emailsCV <- emailsTrain[cvInds]
isSpamCV <- isSpamTrain[cvInds]

emailsTrainCV <- emailsTrain[-cvInds]
```

```r
isSpamTrainCV <- isSpamTrain[-cvInds]

#now we train the data using emailsTrainCV
bow <- bowFunc(emailsTrainCV)

tSpam <- table(unlist(emailsTrainCV[which(isSpamTrainCV)]))
tHam <- table(unlist(emailsTrainCV[which(!isSpamTrainCV)]))

bowSpam <- numeric(length(bow)); names(bowSpam) <- bow
bowHam <- numeric(length(bow)); names(bowHam) <- bow

bowSpam[names(tSpam)] <- tSpam
bowHam[names(tHam)] <- tHam

s1 <- sum(isSpamTrainCV)
probPresentSpam <- sapply(bowSpam, function(w) (w+.01)/(s1+.01))
probAbsentSpam <- 1-probPresentSpam

s2 <- sum(!isSpamTrainCV)
probPresentHam <- sapply(bowHam, function(w) (w+.01)/(s2+.01))
probAbsentHam <- 1-probPresentHam

logProbPresentSpam <- log(probPresentSpam)
logProbPresentHam  <- log(probPresentHam)
logProbAbsentSpam  <- log(probAbsentSpam)
logProbAbsentHam   <- log(probAbsentHam)

#now we calculate c using emailsCV and isSpamCV
bfCV <- sapply(emailsCV, computeBF)

c <- -500:500
t1error <- numeric(length(c))
t2error <- numeric(length(c))
for (i in 1:length(c)) {
  t1error[i] <- length(which(bfCV[which(!isSpamCV)] > c[i])) / sum(!isSpamCV)
  t2error[i] <- length(which(bfCV[which(isSpamCV)] <= c[i])) / sum(isSpamCV)
}
bestCInd <- which.min(abs(t1error-t2error))
bestC <- c[bestCInd]

#now we evaluate how well this worked on our test set
t1errorTest <- length(which(bfTest[which(!isSpamTest)] > bestC)) / sum(!isSpamTest)
t2errorTest <- length(which(bfTest[which(isSpamTest)] <= bestC)) / sum(isSpamTest)

message('Cutoff and Error Rates using simple Cross-Validation')
message('Type-I error with c=',bestC,': ',round(t1errorTest, 3),
        '\nType-II error with c=',bestC,': ',round(t2errorTest, 3))

#k fold cross validation

#first we split the training set into k subsets by indicies
kFoldSplits <- function(k) {
  spamIndexes    <- which(isSpamTrain)
  nonSpamIndexes <- which(!isSpamTrain)

  numSpam <- length(spamIndexes)
  numHam  <- length(nonSpamIndexes)

  toReturn <- list()

  for (i in 1:k) {
    inds1 <- sample(1:length(spamIndexes), floor(numSpam/k))
    inds2 <- sample(1:length(nonSpamIndexes), floor(numHam/k))

    toReturn[[i]] <- c(spamIndexes[inds1], nonSpamIndexes[inds2])

    spamIndexes <- spamIndexes[-inds1]
    nonSpamIndexes <- nonSpamIndexes[-inds2]
  }

  return(toReturn)
}

findC <- function(k){
  eureka <- c()
  c <- -500:500
  t1error <- numeric(length(c))
  t2error <- numeric(length(c))

  #first we split
  kInds <- kFoldSplits(k)

  for (i in 1:k) {
    message('Check1: ',i)
    #we split into training and CV
    trainingInds <- unlist(kInds[-i])

    tempTrain <- emailsTrain[trainingInds]
    isSpamTempTrain <- isSpamTrain[trainingInds]

    tempCV <- emailsTrain[-trainingInds]
    isSpamTempCV <- isSpamTrain[-trainingInds]

    #we train using tempTrain and isSpamTempTrain
    bow <- bowFunc(tempTrain)

    tSpam <- table(unlist(tempTrain[which(isSpamTempTrain)]))
    tHam <- table(unlist(tempTrain[which(!isSpamTempTrain)]))

    bowSpam <- numeric(length(bow)); names(bowSpam) <- bow
    bowHam <- numeric(length(bow)); names(bowHam) <- bow

    bowSpam[names(tSpam)] <- tSpam
    bowHam[names(tHam)] <- tHam

    s1 <- sum(isSpamTempTrain)
```

```r
    probPresentSpam <- sapply(bowSpam, function(w) (w+.01)/(s1+.01))
    probAbsentSpam <- 1-probPresentSpam

    s2 <- sum(!isSpamTempTrain)
    probPresentHam <- sapply(bowHam, function(w) (w+.01)/(s2+.01))
    probAbsentHam <- 1-probPresentHam

    logProbPresentSpam <- log(probPresentSpam)
    logProbPresentHam  <- log(probPresentHam)
    logProbAbsentSpam  <- log(probAbsentSpam)
    logProbAbsentHam   <- log(probAbsentHam)

    message('Check2 done training: ',i)

    #now we calculate c using tempCV and isSpamTempCV
    bfCV <- sapply(tempCV, computeBF)

    for (i in 1:length(c)) {
      t1error[i] <- length(which(bfCV[which(!isSpamTempCV)] > c[i])) / sum(!isSpamTempCV)
      t2error[i] <- length(which(bfCV[which(isSpamTempCV)] <= c[i])) / sum(isSpamTempCV)
    }
    bestCInd <- which.min(abs(t1error-t2error))
    buenoC <- c[bestCInd]
    eureka <- c(eureka, buenoC)
    message('Check3 got a c: ',i)
    print(eureka)
  }

  return(mean(eureka))
}

#now we see how it did
mejorC <- findC(5)
t1errorTest <- length(which(bfTest[which(!isSpamTest)] > mejorC)) / sum(!isSpamTest)
t2errorTest <- length(which(bfTest[which(isSpamTest)] <= mejorC)) / sum(isSpamTest)

message('Cutoff and Error Rates using k-fold Cross-Validation')
message('Type-I error with c=',round(mejorC, 3),': ',round(t1errorTest, 3),
        '\nType-II error with c=',round(mejorC,3),': ',round(t2errorTest, 3))

'%!in%' <- function(x,y)!('%in%'(x,y))

newExtractWords <- function(body) {
  txt <- paste(body, collapse = ' ')
  txt <- deparse(txt)
  #get rid of blank spaces
  txt <- gsub('[[:blank:]]+', ' ', txt)

  #get rid of email addresses
  txt <- gsub(' [[:alnum:]._-]+@[[:alnum:].-]+ ', ' ', txt)

  #get rid of URLs
  txt <- gsub('(http(s)?:\\/\\/.)?(www\\.)?[-a-zA-Z0-9@:%._\\+~#=]{2,}\\.[a-z]{2,6}\b([-a-zA-Z0-9@:%_\\
               ' ', txt)

  #get rid of numbers, blank spaces, punctuation
  txt <- gsub('[[:digit:][:blank:][:punct:]]+', ' ', txt)

  txt <- tolower(txt)
  txt <- unlist(strsplit(txt, ' '))
  txt <- unique(txt)
  txt <- txt[which(nchar(txt)>1)]

  #use tm package to stem the words
  txt <- stemDocument(txt)

  #get rid of stop words
  txt <- txt[which(txt %!in% stopwords())]

  return(txt)
}

emailsAllFunc2 <- function() {
  toReturn <- list()
  for (dir in list.files("data/messages")) {
    mail <- readEmailDirectory(paste0("data/messages/",dir))
    message('check 1 ', dir)
    body <- lapply(mail, function(m) extractBodyText(m))
    message('check 2 ', dir)
    words <- lapply(body, function(b) newExtractWords(b))
    toReturn <- c(toReturn, words)
  }
  return(toReturn)
}

emailsAll2 <- emailsAllFunc2()

emailsTrain2 <- emailsAll2[specialInds]
emailsTest2 <- emailsAll2[-specialInds]

findC2 <- function(k){
  eureka <- c()
  c <- -500:500
  t1error <- numeric(length(c))
  t2error <- numeric(length(c))

  #first we split
  kInds <- kFoldSplits(k)

  for (i in 1:k) {
    message('Check1: ',i)
    #we split into training and CV
    trainingInds <- unlist(kInds[-i])

    tempTrain <- emailsTrain2[trainingInds]
    isSpamTempTrain <- isSpamTrain[trainingInds]
    tempCV <- emailsTrain2[-trainingInds]
    isSpamTempCV <- isSpamTrain[-trainingInds]

    #we train using tempTrain and isSpamTempTrain
    bow <- bowFunc(tempTrain)

    tSpam <- table(unlist(tempTrain[which(isSpamTempTrain)]))
    tHam <- table(unlist(tempTrain[which(!isSpamTempTrain)]))

    bowSpam <- numeric(length(bow)); names(bowSpam) <- bow
    bowHam <- numeric(length(bow)); names(bowHam) <- bow

    bowSpam[names(tSpam)] <- tSpam
    bowHam[names(tHam)] <- tHam

    s1 <- sum(isSpamTempTrain)
    probPresentSpam <- sapply(bowSpam, function(w) (w+.01)/(s1+.01))
    probAbsentSpam <- 1-probPresentSpam

    s2 <- sum(!isSpamTempTrain)
    probPresentHam <- sapply(bowHam, function(w) (w+.01)/(s2+.01))
    probAbsentHam <- 1-probPresentHam

    logProbPresentSpam <- log(probPresentSpam)
    logProbPresentHam  <- log(probPresentHam)
    logProbAbsentSpam  <- log(probAbsentSpam)
    logProbAbsentHam   <- log(probAbsentHam)

    message('Check2 done training: ',i)

    #now we calculate c using tempCV and isSpamTempCV
    bfCV <- sapply(tempCV, computeBF)

    for (i in 1:length(c)) {
      t1error[i] <- length(which(bfCV[which(!isSpamTempCV)] > c[i])) / sum(!isSpamTempCV)
      t2error[i] <- length(which(bfCV[which(isSpamTempCV)] <= c[i])) / sum(isSpamTempCV)
    }
    bestCInd <- which.min(abs(t1error-t2error))
    buenoC <- c[bestCInd]
    eureka <- c(eureka, buenoC)
    message('Check3 got a c: ',i)
    print(eureka)
  }

  return(mean(eureka))
}

cSupreme <- findC2(5)

bfTest <- sapply(emailsTest2, computeBF)
t1errorTest <- length(which(bfTest[which(!isSpamTest)] > cSupreme)) / sum(!isSpamTest)
t2errorTest <- length(which(bfTest[which(isSpamTest)] <= cSupreme)) / sum(isSpamTest)
message('Cutoff Threshold and Error Rates for newExtractWords')
```

```r
message('Type-I error with c=',round(cSupreme, 3),': ',round(t1errorTest, 3),
        '\nType-II error with c=',round(cSupreme,3),': ',round(t2errorTest, 3))

myEmails <- function() {
  toReturn <- list()
  for (dir in c('data/myHam', 'data/mySpam')) {
    mail <- readEmailDirectory(dir)
    body <- lapply(mail, extractBodyText)
    words <- lapply(body, extractWords)
    toReturn <- c(toReturn, words)
  }
  return(toReturn)
}

myMail <- myEmails()
isSpamMyMail <- c(rep(FALSE, 3), rep(TRUE, 3))

bfMyMail <- lapply(myMail, computeBF)

calculatedAsSpam <- which(bfMyMail <= firstbestC)
message('Number correctly IDed: ', sum(calculatedAsSpam == isSpamMyMail),
        '\nNumber incorrectly IDed: ', 6-sum(calculatedAsSpam == isSpamMyMail))
```