Spam Classification Project

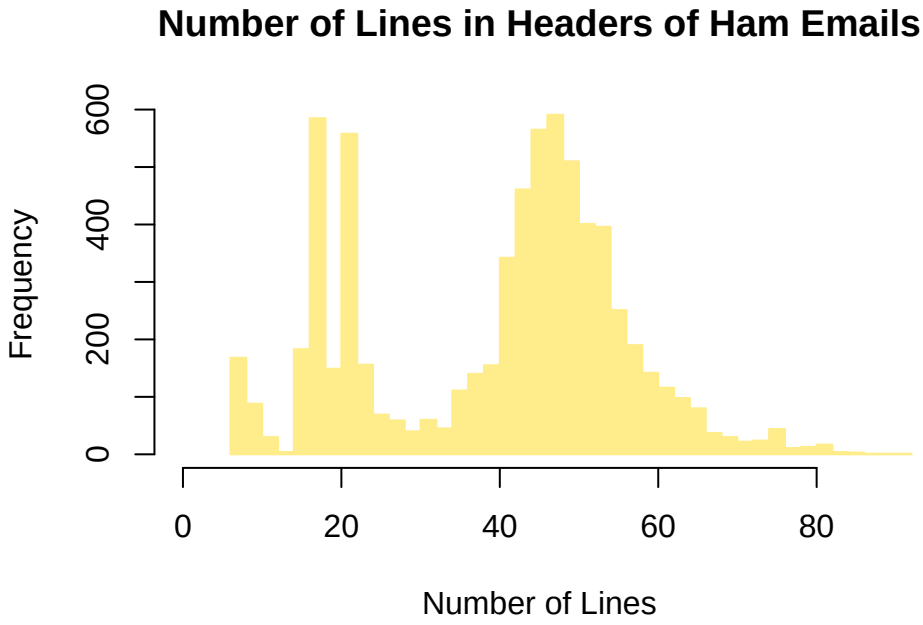
Isabel Arvelo

2022-05-17

Introduction

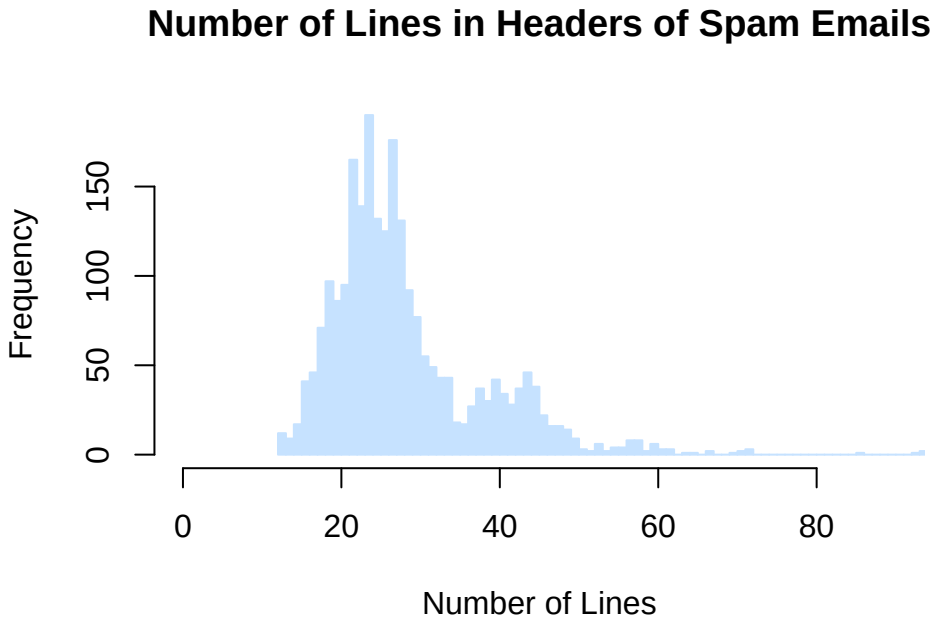
In this project, we developed and tested a detection algorithm for spam emails. We began by downloading and exploring a corpus of over 9,000 spam and non-spam emails from the website Spam Assassin, which is an open-source spam classification software. In order to study the format of email messages, we employed text processing and manipulation to extract emails into their different components. We examined and compared different aspects of spam vs non spam emails such as the number of lines in their headers. Next, we extracted all and the unique words that appear in the body of each email message. We then implemented a supervised learning algorithm using Naive Bayesian classification based on the words in each message to calculate the strength of evidence in support of an email being spam (as compared to the evidence against the email being spam). We trained and tested the algorithm on stratified samples of real spam and non-spam email messages. After implementation, we considered different threshold values for the log Bayes Factor, and analyzed how these choices affected Type-I (false alarm) and Type-II (missed detection) error rates. I also elected to further explore trends in the subject lines and domains of sending email addresses to assess whether any additional aspects of emails could be incorporated into the classifier to improve its performance.

Deliverable #1



The distribution of the number of lines in the headers of ham emails is bimodal and slightly skewed left. The mean is 40.28, with a standard deviation of 16.18. The median is 45 and the middle 75% of email headers fall

between 22 and 51 lines. The minimum number of lines in a header in this directory is 6 and the maximum is 91 .



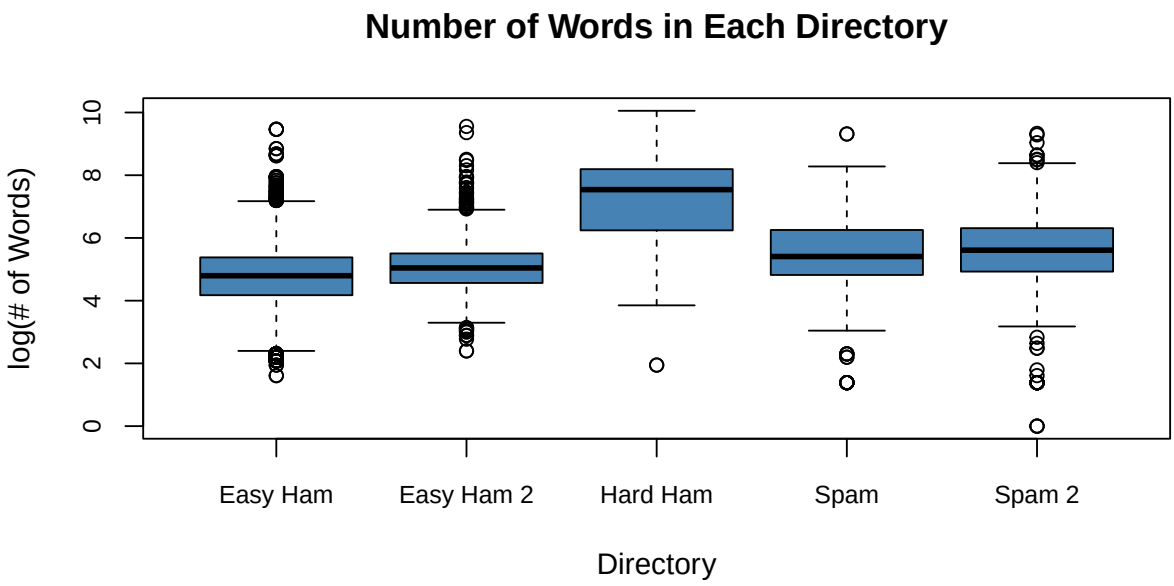
The distribution of the number of lines in the headers of spam emails is unimodal and skewed right. The mean number of lines is 29.09, with a standard deviation of 13.53. The median is 26 and the middle 75% of email headers fall between 22 and 32 lines. The minimum number of lines in a header in this directory is 12 and the maximum is 412. Out of the 2,397 emails in the spam directories there are 3 emails that have headers with more than 115 lines.

There appear to be distinct differences in the distributions of the number of lines in the headers of ham emails versus spam emails. Both distributions have similar standard deviations, but the the distribution of ham header lines are centered around higher values indicating that ham emails tend to have more lines than spam emails. However, it is important to not that there are several outliers in the spam email directory that have a high number of lines in their headers. It is definitely not deterministic, but the number of lines in an email header is a useful indicator of spam emails.

Deliverable #2

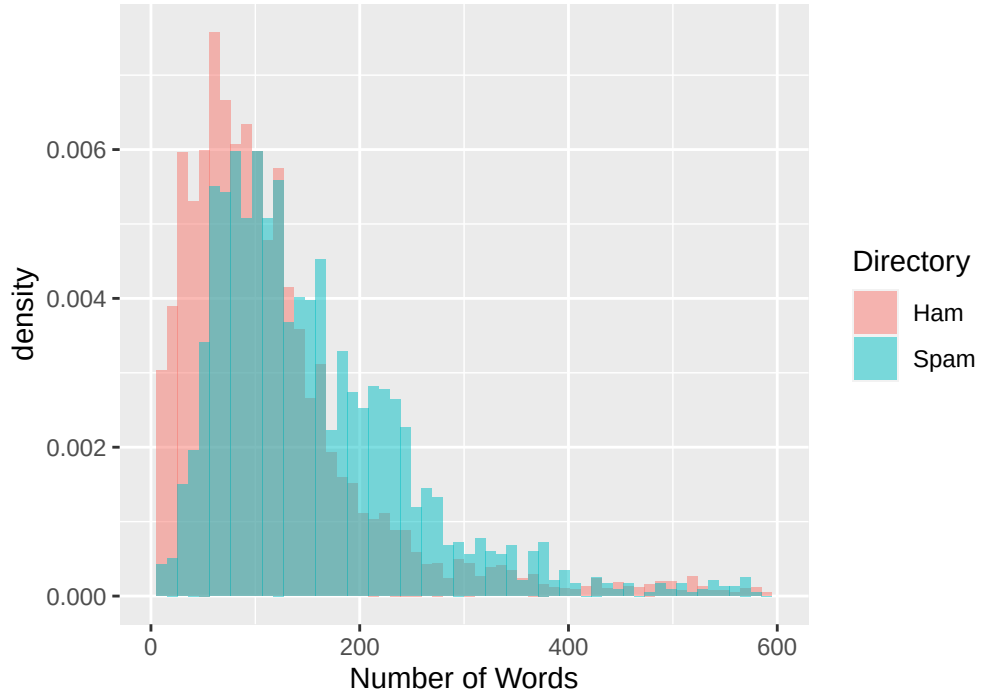
1. 92 emails in the hard_ham directory have attachments.
2. In order to be confident that my getBoundary() function is working correctly, I randomly selected 25 emails from each directory and of those 25, selected the emails with attachments. I then wrote a function that extracts the lines in the header that contain "boundary=" and compared the extracted boundary line to the output of the getBoundary() function. I found that of the 15 randomly selected boundaries I manually checked, the getBoundary() function successfully extracted the boundary from the boundary line. In addition to this brute force visual check, I also randomly selected 200 emails from the all of the emails. Of those 200 random emails, 20 had attachments, and the boundary extracted from those email headers appeared later in the bodies of each of those emails. The code I used for these checks can be found on page 11 in the appendix under "Boundary Check".

Deliverable 3



The hard ham directory generally contains emails with the most total words, as indicated by the center of its distribution that is notably greater than the medians of the other directories. The hard_ham directory also has the least outliers and is the only one that is noticeably skewed left. The two easy ham directories have a lot more outliers than the spam directories, but the distributions of log(#of words) appear to be fairly similar.

Deliverable 4



Looking at the plot, the number of unique words appearing in an email is not a decisively clear indicator of spam, but it could be used to help identify spam emails. A reasonable threshold for flagging an email as

"spam" would be about 250 unique words. Defining an outlier as any value that falls within 1.5*(IQR) above the 3rd quartile, this threshold is also consistent with the threshold for outliers for the number of unique words in the headers of ham emails, which is 242.

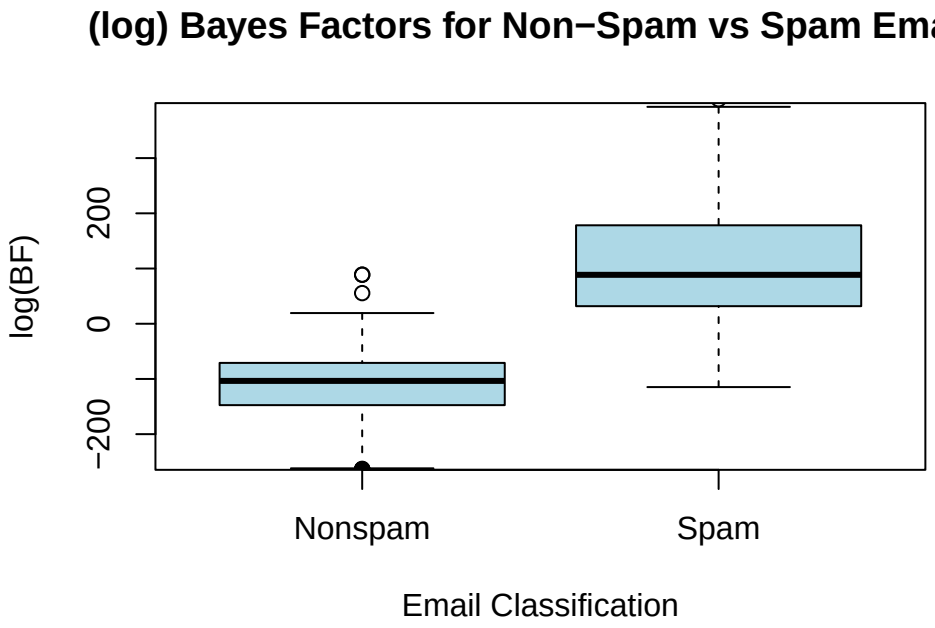
Deliverable 5

A derivation of the Bayes Factor, representing the evidence in favor of an email being spam (relative to the evidence against it being spam), can be found attached to the end of this report.

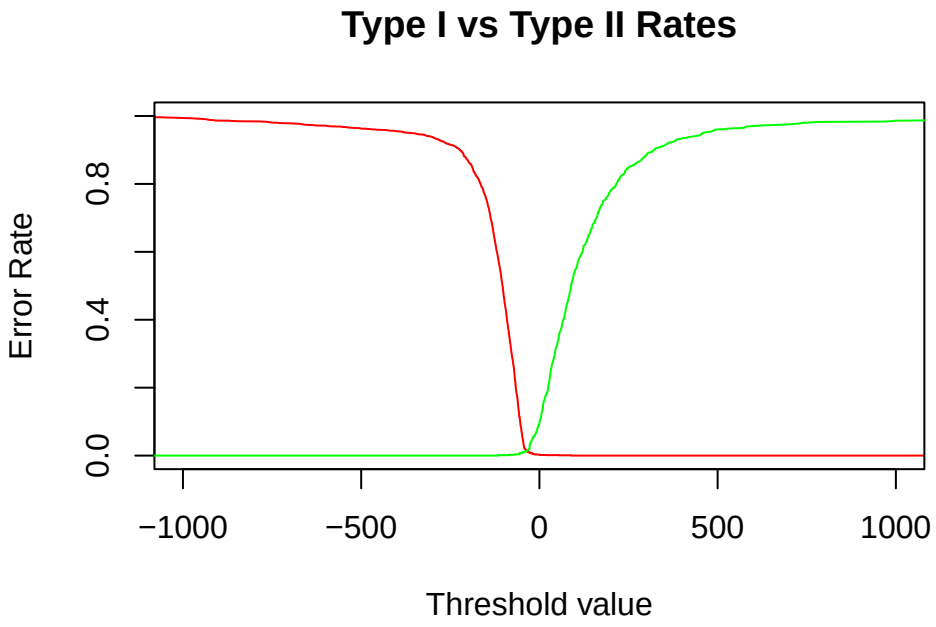
Deliverable 6

According to the training data, the word 'monday' is 5.560457 times more likely to appear in non-Spam emails than in spam emails. The word 'buy' is 2.205774 times more likely to appear in Spam emails than in non-Spam emails.

Deliverable 7



Using the plot above, a good threshold value c for classifying an emails as Spam would be 20.



Although the distinct values of the Bayes Factors for all emails in the test set range from -7588.88 to 7212.32, the error rates begin to asymptotically approach 1, so I limited the x range to give better resolution for the values with lower absolute value.

Deliverable 9

a. The c value that nearly achieves $p(Type - IError) = p(Type - IIError)$ is c = -29.702727.

b. The resulting type I rate and type II rates are 1.1653% and 1.251564%, respectively.

Deliverable 10

Enforcing a Type-I error rate of less than 0.1%, the smallest Type-II error rate that you can achieve is 0.3566959. This means that if we want to implement a classifier that is only expected to send 0.1% of non-Spam emails to spam, the probability that the classifier fails to detect a spam email is 35.67%.

Extensions

For my extensions, I was interested in observing other properties of emails that I thought could be useful in helping classify messages as spam or non-spam. Although users themselves aren't responsible for spam classification, I wanted to look further into the first thing I typically look at when I receive an email: the subject. I wrote a function to extract the subject from the header as well as the individual words from the subject to investigate whether or not this field could be a useful parameter for my classifier. In parts of this analysis, I decided to exclude stop words. Stop words are short commonly used words and I decided to exclude these words in some parts of text processing because they are so frequently used that they carry very little information and may not be helpful in distinguishing spam vs nonspam emails. I used the list of english stopwords in the Stopwords ISO library found in the R package "stopwords".

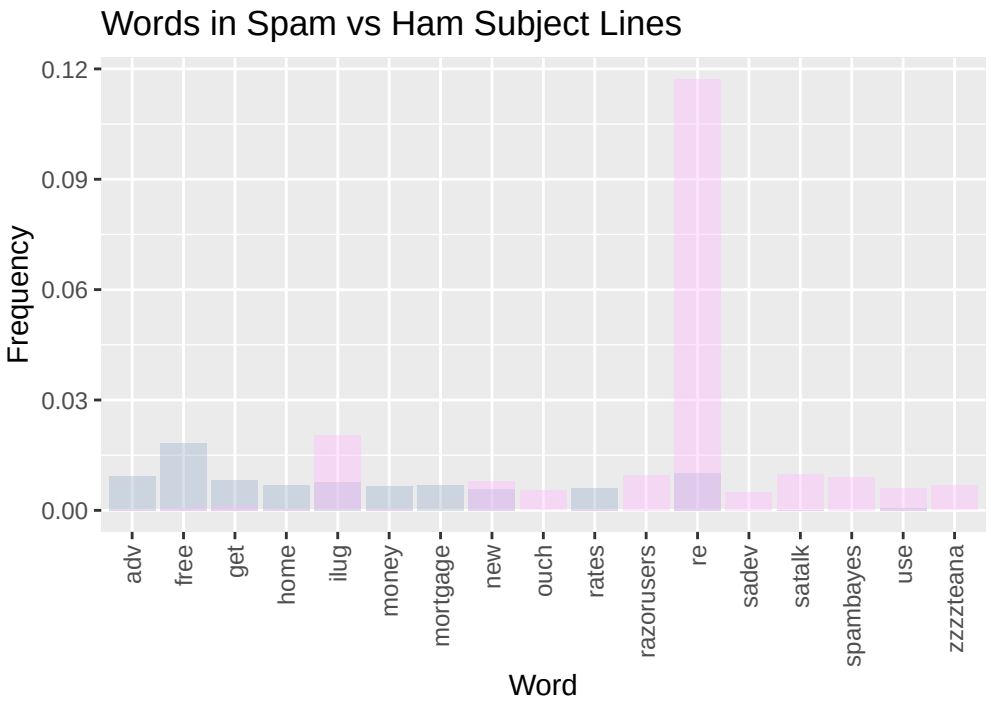
The top 10 most common (non-stopword) words in the subject lines of nonspam emails were

	Probability
re	11.7179742
ilug	2.0326465
satalk	0.9961519
razorusers	0.9558093
spambayes	0.9061569
new	0.7944389
zzzzteana	0.6734111
use	0.6113456
ouch	0.5430735
sadev	0.5027309

The top 10 most common (non-stopword) words in the subject lines of spam emails were

	Probability
free	1.8332237
re	1.0059227
adv	0.9401147
get	0.8273009
ilug	0.7614929
home	0.6862837
mortgage	0.6862837
money	0.6674814
rates	0.6016734
new	0.5828711

I then collected the 10 top most frequent (non-stopword) words in each directory and overlayed their frequency in the subjects of both directories. The gray bars represent the frequency in spam emails and the pink bars represent the frequency in ham emails.



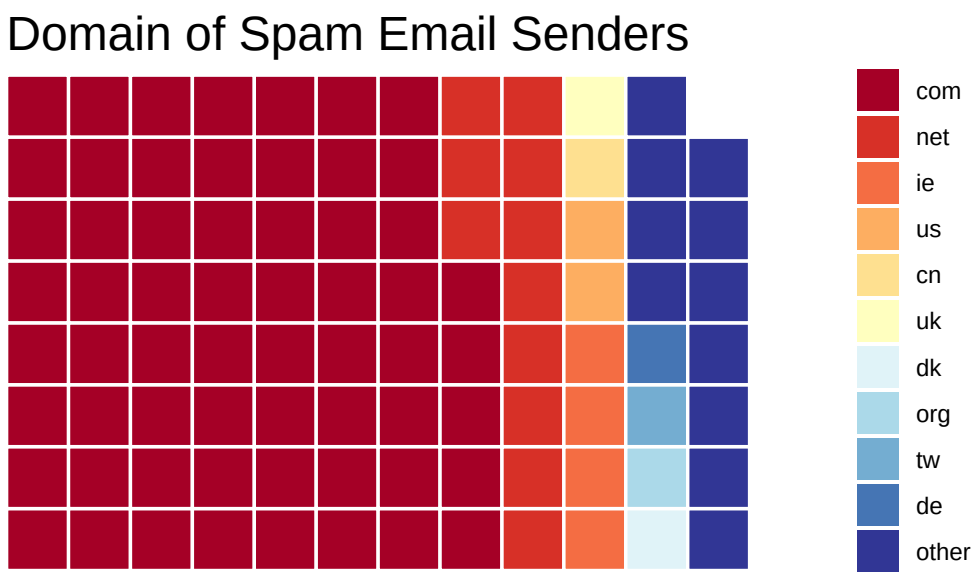
It is unsurprising that the words *free*, *rates*, *home*, *money*, and *get* are common in the subject of spam emails because spam emails often promise the opportunity to get free items or discounted rates on products like insurance or mortage(another common word in the subject of spam emails) to bait people to click on the message. The only words with fairly similar frequencies in both directories was *new*. I suspect that the frequency of "spammish-sounding" words like "zzzzteana" in the subject of non-spam emails originate from the hard ham directory which is closer in many respects to typical spam.

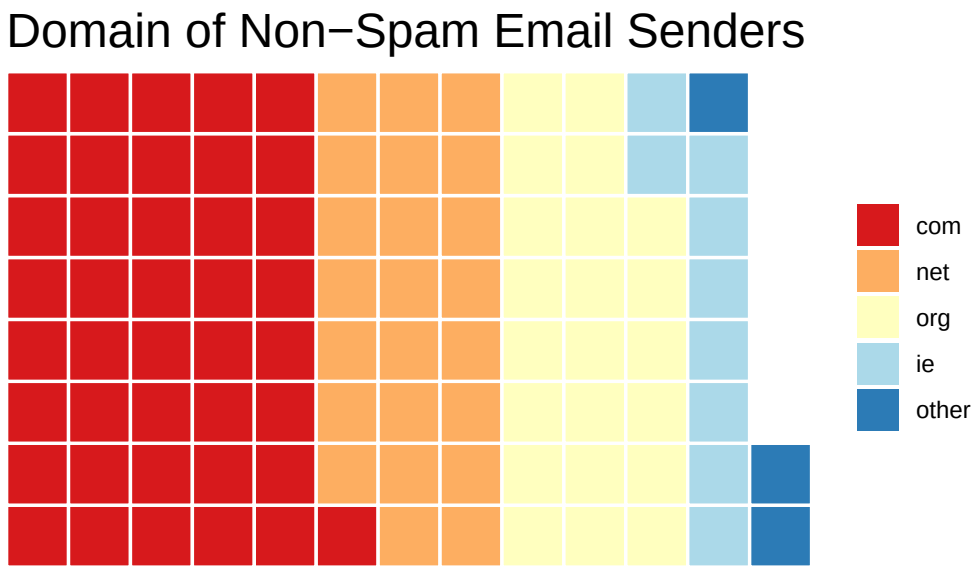
I then proceeded to include words in the subject of the email (including stop words) in my classifier to see whether or not it improved the performance of my algorithm. I chose to weigh the words in the subject the same as the words in the body of the email.

I found that including subject words in the classifier improved the performance of the classifier. By enforcing a Type-I error rate of less than 0.1%, the smallest Type-II error rate that you can achieve is 33.17%. This means that with a fixed type one error rate, including the domain in the classifier would decrease missed detections by about 2.5%.

I then analyzed the length of subject lines between spam and non spam emails, but did not find any meaningful disparities in the distribution of subject lengths to investigate further. The mean length of ham subject lines was 5.711 with a standard deviation of 2.627 and the mean length of spam subject lines was 5.798 with a standard deviation of 2.942.

The next element of the email header I was interested in analyzing further was the email address of the email, sending each email. I wrote functions to extract the username (eg. "robert.chambers"), the domain (eg. "baesystems.com"), and the top level domain (eg. ".com") for each email address in the "From:" field in the header. I decided to focus mostly on top level domains since the README file mentioned that 'Some address obfuscation has taken place, and hostnames in some cases have been replaced with "spamassassin.taint.org" '.





There are 60 unique top level domains observed in the sending email addresses in the spam directory, but only 14 unique domains in the sending emails of the non-spam directories. Sending email addresses that end with .com are the most common in both directories, but appear about 13% more frequently in the spam directory. The domain org is about 6.07 more likely amongst non-Spam emails than spam emails. 'ie', a domain in the top 4 most common domains in both spam and non spam directories is the country code top-level domain that corresponds with the ISO 3166-1 alpha-2 code for Ireland.

Including domain in the classifier does not improve performance. I tuned and chose the weight of this

parameter by using deliverable #10 as a metric to measure the success of my classifier. Enforcing a Type-I error rate of less than 0.1%, the smallest Type-II error rate that you can achieve is 40.30%. This means that with a fixed type one error rate, including the domain in the classifier would increase missed detections by about 4.6% as compared to the original classifier.

I then attempted to improve my extractWords function into a "deluxe version" that did not consider the previously collection of 174 stop words. I found Bayes Factor for each email only considering non-stopword words in the email and used the same metric to analyze whether or not this improved my classifier or not.

The extractWordsDeluxe function, only considering words not in the stopwords list, made a negligible difference in the performance of the classifier. This is not surprising since the words probably appeared at similar frequencies in both spam and non-spam emails so eliminating them did not make a big difference. Enforcing a Type-I error rate of less than 0.1%, the smallest Type-II error rate that you can achieve is 35.9199%. This means that with a fixed type one error rate, excluding stop words from the text processing would increase increase detections by about .2499%, as compared to the original algorithm.

Appendix:

```
splitMessage <- function(x) {
  firstl <- which(x == "")[1]
  header <- head(x, firstl - 1)
  body <- tail(x, length(x) - firstl)
  return (list(header = header, body = body))
}
```

```
hasAttachment <- function(x) {
  header <- x
  content_indices <- grep("content-type", header, ignore.case = TRUE)
 if (length(content_indices) == 0) {
    return (FALSE)
 }

  content_index = content_indices[1]

  if ( grepl("multipart", header[content_index],  fixed = TRUE) || grepl("MULTIPART", header[content_in
    return (TRUE)
  }
  else return (FALSE)
}
```

```
getBoundary <- function(x) {
  if (hasAttachment(x)) {
    boundary_indices <- grep("boundary=", x,  ignore.case = TRUE)
  if (length(boundary_indices) == 0) {
    print(x)
    stop('No boundary line')
  }

  boundary_index = boundary_indices[1]

  boundary_line <- x[boundary_index]
  boundary_line <- gsub('Content.+; ', "", boundary_line)
  boundary_line <- gsub('CONTENT.+;', "", boundary_line)
  boundary_line <- gsub('content.+;', "", boundary_line)
  boundary_line <- gsub('charset.+', "", boundary_line)
  boundary_line <- gsub(".*boundary=", "", boundary_line)
  boundary_line <- gsub(".*BOUNDARY=", "", boundary_line)
  boundary_line <- gsub('^[[:space:]]+', '', boundary_line)
  boundary_line <- gsub('[[:space:]]+$', '', boundary_line)
  boundary_line <- gsub('"', "", boundary_line)
  boundary_line <- gsub(';.?type=.+$', "", boundary_line)
  boundary_line <- gsub(';', "", boundary_line)

  return(boundary_line)
  }

  else {
    return ("Email does not have attachments")
  }
}
```

Boundary Check:

```
printBoundary <- function(directory, indices) {
  for(i in indices) {
    header <- splitMessage(directory[[i]])$header
    boundary <- getBoundary(header)
    boundaryLine <- grep("boundary=", header, value = TRUE)
    cat(boundaryLine, "\n", boundary, "\n" , "\n")
}
}
```

```
##     boundary="----=_NextPart_000_0300_01C23C66.DA1E1310"
##  ----=_NextPart_000_0300_01C23C66.DA1E1310
##
##     boundary="==_Exmh_1643180900P";
##  ==_Exmh_1643180900P
##

## Content-Type: multipart/alternative; boundary=6700390.1026560924954.JavaMail.root.umsan1
##  6700390.1026560924954.JavaMail.root.umsan1
##
##     boundary="----------=_1029244856-11739-4";
##  ----------=_1029244856-11739-4
##
## Content-Type: multipart/alternative; boundary="----=_NextPart_000_0000_01C24A39.4307A140"
##  ----=_NextPart_000_0000_01C24A39.4307A140
##
##  boundary="----=_LINDOWS_1890362a5620f32f969ae58b8fe2ef84"
##  ----=_LINDOWS_1890362a5620f32f969ae58b8fe2ef84
##
## Content-Type: multipart/alternative; boundary="_----------=_103005139443991"
##  _----------=_103005139443991
##
## Content-Type: multipart/alternative; boundary=15263843.1026556855445.JavaMail.root.umsan1
##  15263843.1026556855445.JavaMail.root.umsan1
##

##  boundary="______BoundaryOfDocument______"
##  ______BoundaryOfDocument______
##
## Content-Type: multipart/alternative; boundary="----=_NextPart_000_4098D_01C25F36.D7652D00"
##  ----=_NextPart_000_4098D_01C25F36.D7652D00
##
## Content-Type: multipart/mixed; boundary="----=_NextPart_000_00C5_22E25E1D.D0727B17"
##  ----=_NextPart_000_00C5_22E25E1D.D0727B17
##
## Content-Type: multipart/mixed; boundary=XX4E5F4E43-0C5F4E5FXX
##  XX4E5F4E43-0C5F4E5FXX
##
## Content-Type: multipart/related; boundary="----=_NextPart_QXVROEFEFX"
##  ----=_NextPart_QXVROEFEFX
##
##  boundary="----=_NextPart_000_00X9_70A11C1D.E1232J43"
```

```
##  ----=_NextPart_000_00X9_70A11C1D.E1232J43
##


## Content-Type: multipart/related; type="multipart/alternative"; boundary="----=_NextPart_dO8zJ9VZJNgv
##  ----=_NextPart_dO8zJ9VZJNgvpKVZoFA
##
## Content-Type: multipart/alternative; boundary="----=_NextPart_000_0108_01C206E1.3013D6D0"
##  ----=_NextPart_000_0108_01C206E1.3013D6D0
##
## Content-Type: multipart/related; type="multipart/alternative"; boundary="----=_NextPart_a4U4f4Lb5R29
##  ----=_NextPart_a4U4f4Lb5R29pMZQoWcLdBAFKn
##
## Content-Type: multipart/related; type="multipart/alternative"; boundary="----=_NextPart_HAubnVEDr91C
##  ----=_NextPart_HAubnVEDr91CVFKk0
##
```

```
sampleIndex <- sample(1:9348, 200)
sampleEmails <- allEmails[sampleIndex]

sampleHeaders <- sapply(sampleEmails, function(x) splitMessage(x)$header)
sampleBodies <- sapply(sampleEmails, function(x) splitMessage(x)$body)

sampleHasAttachments <- sapply(sampleHeaders, function(x) hasAttachment(x))

emailIndWithAttachments <- which(sampleHasAttachments)

for(i in emailIndWithAttachments) {
    header <- sampleHeaders[[i]]
    body <- sampleBodies[[i]]
    boundary <- getBoundary(header)
    if (length(grep(boundary, body, fixed = TRUE)) < 1 ) {
      print ("Failed Boundary Extraction")
      boundaryLine <- grep("boundary=", header, value = TRUE)
      print(boundary)
      cat(boundaryLine, "\n")
    }
}
```

```
extractBodyText <- function(x) {
  body <- splitMessage(x)$body
  header <- splitMessage(x)$header
  if (hasAttachment(x)) {
    boundary <- getBoundary(x)
    b_indices <- grep(boundary, body, useBytes = TRUE, fixed = TRUE)
    first_b <- b_indices[1]
    if (length(b_indices) == 0) {
      stop("No boundary")
    }
    if (length(b_indices) == 1) {
      return(tail(body, -first_b))
    }
    else {
      second_b <-b_indices[2]
```

```
      return(body[(first_b + 1):(second_b - 1)])
    }
  }
  else {
    return(body)
  }
}
```

```
extractWords <- function(x, unique) {
  body <- extractBodyText(x)
  dp_body <- unlist(lapply(body, function(x) deparse(x)))
  full_body <- paste(dp_body, collapse = ' ')
  full_body <- gsub('\\\\.{3}', ' ', full_body)
  full_body <- gsub('[[:digit:]]', ' ', full_body)
  full_body <- tolower(full_body)
  full_body <- gsub('[[:punct:]]', ' ', full_body)
  full_body <- gsub('[[:space:]][a-z][[:space:]]', ' ', full_body)
  full_body <- gsub('[[:space:]]+', ' ', full_body)
  full_body <- gsub('^[[:space:]]+', '', full_body)
  full_body <- gsub('[[:space:]]+$', '', full_body)

  words <- str_split(full_body, " ")

  if (unique) {
    return(unique(unlist(words)))
  } else {
    return(unlist(words))
  }
}
```

```
readEmailDirectory <- function(x) {
  email_files <- list.files(x, full.names = TRUE)
  emails <- lapply(email_files, readLines)
}
```

```
full_directory <- c(easy_ham, easy_ham_2, hard_ham, spam, spam_2)
emailsAll <- lapply(full_directory, function(x) extractWords(x, unique = TRUE))

isSpam <- c( rep(FALSE, (length(easy_ham) + length(easy_ham_2) + length(hard_ham))) , rep(TRUE, (length
```

```
`%!in%` <- Negate(`%in%`)

indices <- c(1: 9348)
ham_indices <- c(1: 6951)
spam_indices <- c(6952:9348)

num_Spam_training <- round(9348* 0.2564185 * trainingFrac)
num_ham_training <- round( (9348*trainingFrac) - num_Spam_training )

shuffled_spam <- sample(spam_indices, num_Spam_training)
shuffled_ham <- sample(ham_indices, num_ham_training)

training_indices <- c(shuffled_spam,shuffled_ham)
```

```
test_indices <- which(indices %!in% training_indices)


emailsTrain <- emailsAll[training_indices]
emailsTest <- emailsAll[test_indices]

isSpamTest <- unlist(lapply(test_indices, function(x) {
                      if (x %in% 1:6951) {
                        return (FALSE)
                      } else {
                        return (TRUE)
                      }

}))

isSpamTrain <- unlist(lapply(training_indices, function(x) {
                      if (x %in% 1:6951) {
                        return (FALSE)
                      } else {
                        return (TRUE)
                      }

}))

c(mean(isSpam), mean(isSpamTrain), mean(isSpamTest))
```

```
## [1] 0.2564185 0.2564185 0.2564185
```

```
length(emailsTrain) / length(emailsAll)
```

```
## [1] 0.6666667
```

```
bow <- unique(unlist(emailsTrain))
bow <- bow[bow != ""]
```

```
computeBF <- function(uniqueWords) {

  words <- uniqueWords[uniqueWords %in% bow]

  present_indices <- which(names(ProbPresentSpam) %in% words)
  absent_indices <- which(names(ProbPresentSpam) %!in% words)

  SpamPresentProbs <-  logProbPresentSpam[present_indices]
  HamPresentProbs <- logProbPresentHam[present_indices]

  SpamAbsentProbs <- logProbAbsentSpam[absent_indices]
  HamAbsentProbs <- logProbAbsentHam [absent_indices]

  spamHypothesis <- sum(SpamPresentProbs) + sum(SpamAbsentProbs)
  hamHypothesis  <- sum(HamPresentProbs) + sum(HamAbsentProbs)

  BF <- spamHypothesis - hamHypothesis
```

```
  return(BF)
}
```

```
plotErrorRates <- function(tOneVector,tTwoVector, cValues, zoom = FALSE) {

    plot(cValues,tOneVector,type="l",col="red", main = "Type I vs Type II Rates", xlab = "Threshold valu
lines(cValues,tTwoVector,col="green")
legend(500, 0.2, legend=c("Type I (false alarm)", "Type II (missed detection)"),
       col=c("red", "green"), lty = 1,  cex=0.8)

if (zoom) {
    plot(cValues,tOneVector,type="l",col="red", main = "Type I vs Type II Rates", xlab = "Threshold valu
lines(cValues,tTwoVector,col="green")
legend(500, 0.2, legend=c("Type I (false alarm)", "Type II (missed detection)"),
       col=c("red", "green"), lty = 1,  cex=0.8)

}

}
```

Extension code

```
extractSubject <- function(x) {
  body <- splitMessage(x)$body
  header <- splitMessage(x)$header
  subjectLine <- grep("subject:", header,  ignore.case = TRUE)
  subject <- gsub("Subject: ", "", header[subjectLine])
  return(subject)
}
```

```
extractSubjectWords <- function(x, punct = FALSE, unique = FALSE) {
  subject <- extractSubject(x)
  dp_subject <- unlist(lapply(subject, function(x) deparse(x)))
  subject <- paste(dp_subject, collapse = ' ')
  subject <- tolower(subject)

  swords <- gsub('[[:punct:]]', '', subject)
  spunct <- gsub('[[:alpha:]]', '', subject)

  swords <- gsub('[[:space:]]{2,}', ' ', swords)
  spunct <- gsub('[[:space:]]{2,}', ' ', spunct)

  swords <- gsub('^[[:space:]]+', '', swords)
  spunct <- gsub('^[[:space:]]+', '', spunct)

  swords <- gsub('[[:space:]]+$', '', swords)
  spunct <- gsub('[[:space:]]+$', '', spunct)

  punctuation <- str_split(spunct, " ")
  words <- str_split(swords, " ")

  if (punct) {
    both <- c(words, punctuation)
    return(both)
  }

  if (unique) {
    return(unique(unlist(words)))
  }
  return(unlist(words))

}
```

```
stopwords <- stopwords(kind = "en")
```

```
subjectLength <- function(x) {
  subject <- extractSubject(x)

  length <- nchar(subject, type = 'width', keepNA = FALSE)
  return(length)
}
```

```
subjectNumWords <- function(x) {
  words <- extractSubjectWords(x)
  numWords <- length(unlist(words))
  return(numWords)
}
```

```
hamSubjectLengths <- unlist(lapply(hamEmails, function(x) { subjectNumWords(x) }))
summary(hamSubjectLengths)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   4.000   5.000   5.711   7.000  20.000
```

```
spamSubjectLengths <- unlist(lapply(spamEmails, function(x) { subjectNumWords(x) }))
summary(spamSubjectLengths)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   4.000   6.000   5.798   7.000  23.000
```

```
extractSender <- function(x) {
  header <- splitMessage(x)$header
  senderIndex <- grep("From", header)

  if (length(senderIndex) < 1 ) {
    return(" ")
  }

  if (length(senderIndex) >= 1) {
    senderLine <- senderIndex[1]
  }

  sender <- gsub("^From", "", header[senderLine])
  sender <- gsub("^Received: from", "", sender)

  sender <- gsub("Mon.+$", "", sender)
  sender <- gsub("Tue.+$", "", sender)
  sender <- gsub("Wed.+$", "", sender)
  sender <- gsub("Thu.+$", "", sender)
  sender <- gsub("Fri.+$", "", sender)
  sender <- gsub("Sat.+$", "", sender)
  sender <- gsub("Sun.+$", "", sender)

  sender <- gsub("\\([^()]*\\)", "", sender)
#  sender <- gsub("//(.+//)", "", sender)

  sender <- gsub("^Return-Path: from", "", sender)

  sender <- gsub('[[:space:]]+$', '', sender)
  sender <- gsub('^[[:space:]]+', '', sender)

  sender <- gsub("\\([^()]*\\)", "", sender)
  sender <- gsub('>$', '', sender)
```

```
  return(sender)
}
```

```
extractUsername <- function(x) {
  sender <- extractSender(x)
  username <- gsub("@.+$", "", sender)
  return (username)
}
```

```
extractFullDomain <- function(x) {
  sender <- extractSender(x)
  domain <- gsub(".+@", "", sender)
  return (domain)
}
```

```
extractDomain <- function(x) {
  sender <- extractSender(x)
  domain <- unlist(str_split(sender, "[.]"))
  domain <- domain[length(domain)]

  domain <- gsub('[[:space:]]+$', '', domain)
  domain <- gsub('^[[:space:]]+', '', domain)

  return (domain)
}
```

```
spamSenders <- lapply(spamEmails, function(x) { extractSender(x)})
```

```
fullEmailsTrain <- allEmails[training_indices]
fullEmailsTest <- allEmails[test_indices]

fullSpamTrain <- fullEmailsTrain[which(isSpamTrain)]
fullHamTrain <-  fullEmailsTrain[which(!isSpamTrain)]

spamDomains <- tolower(unlist(lapply(fullSpamTrain,function(x){ extractDomain(x)})))
hamDomains <- tolower(unlist(lapply(fullHamTrain, function(x) { extractDomain(x)})))

#bag of domains
alldoms <- c(hamDomains, spamDomains)
bod <- unique(alldoms)
bod <- bod[bod != ""]

countDomainSpam <- c(rep(0, length(bod)))
countDomainHam <- c(rep(0, length(bod)))

names(countDomainSpam) <- bod
names(countDomainHam) <- bod
```

```
computeBFT_Domain <- function(email) {

  uniqueWords <- extractWords(email, unique = TRUE)
  domain <- extractDomain(email)
```

```
  words <- uniqueWords[uniqueWords %in% bow]

  present_indices <- which(names(ProbPresentSpam) %in% words)
  absent_indices <- which(names(ProbPresentSpam) %!in% words)

  SpamPresentProbs <-  logProbPresentSpam[present_indices]
  HamPresentProbs <- logProbPresentHam[present_indices]

  SpamAbsentProbs <- logProbAbsentSpam[absent_indices]
  HamAbsentProbs <- logProbAbsentHam[absent_indices]

  present_domain <- which(names(ProbPresentSpamDomain) %in% domain)
  absent_domains <- which(names(ProbPresentSpamDomain) %!in% domain)

  SpamPresentProbsDomain <-  logProbPresentSpamDomain[present_domain]
  HamPresentProbsDomain <- logProbPresentHamDomain[present_domain]

  SpamAbsentProbsDomain <- logProbAbsentSpam[absent_domains]
  HamAbsentProbsDomain <- logProbAbsentHam[absent_domains]

  spamHypothesis <- sum(SpamPresentProbs) + sum(SpamAbsentProbs) + 4*(sum(SpamPresentProbsDomain) +
    sum(SpamAbsentProbsDomain))
  hamHypothesis  <- sum(HamPresentProbs) + sum(HamAbsentProbs) + 4*(sum(HamPresentProbsDomain) +
    sum(HamAbsentProbsDomain))


  BF <- spamHypothesis - hamHypothesis

  return(BF)
}
```

```
emailsTrainSubjects <- lapply(allEmails, function(x) extractSubjectWords(x, unique = TRUE) )

spamSubjectTrain <- emailsTrainSubjects[which(isSpamTrain)]
hamSubjectTrain <-  emailsTrainSubjects[which(!isSpamTrain)]

#bag of domains
allSubjectWords <- c(unlist(spamSubjectTrain), unlist(hamSubjectTrain))
bosw <- unique(allSubjectWords)

countSubjectSpamPresent <- c(rep(0, length(bosw)))
countSubjectHamPresent <- c(rep(0, length(bosw)))

names(countSubjectSpamPresent) <- bosw
names(countSubjectHamPresent) <- bosw


allSpamWordsSubject <- unlist(spamSubjectTrain)
spamTableSubject <- table(allSpamWordsSubject)
spamValuesSubject <- as.vector(spamTableSubject)
names(spamValuesSubject) <- names(spamTableSubject)

spamOverlapSubject<- which( names(countSubjectSpamPresent) %in% names(spamValuesSubject) )
```

```
countSubjectSpamPresent[spamOverlapSubject] <- spamValuesSubject[names(countSubjectSpamPresent[spamOverl

allHamWordsSubject <- unlist(hamSubjectTrain)
hamTableSubject <- table(allHamWordsDeluxe)
hamValuesSubject <- as.vector(hamTableDeluxe)
names(hamValuesSubject) <- names(hamTableDeluxe)

hamOverlapSubject <- which( names(countSubjectHamPresent) %in% names(hamValuesSubject) )
countSubjectHamPresent[hamOverlapSubject] <- hamValuesSubject[names(countSubjectHamPresent[hamOverlapSu


CountAbsentHamSubject <- c(rep(0, length(bosw)))
CountAbsentSpamSubject <- c(rep(0, length(bosw)))

names(CountAbsentHamSubject) <- bosw
names(CountAbsentSpamSubject) <- bosw

CountAbsentHamSubject <-  CountAbsentHamSubject + 4634 - countSubjectHamPresent
CountAbsentSpamSubject  <- CountAbsentSpamSubject  + 1598 - countSubjectSpamPresent

ProbPresentSpamSubject<- (countSubjectSpamPresent  + 0.1) / (length(spamSubjectTrain) + 0.1)
ProbAbsentSpamSubject  <- (CountAbsentSpamSubject   + 0.1) / (length(spamSubjectTrain) + 0.1)

ProbPresentHamSubject <- (countSubjectHamPresent  + 0.1) / (length(hamSubjectTrain) + 0.1)
ProbAbsentHamSubject  <- (CountAbsentSpamSubject   + 0.1) / (length(hamSubjectTrain) + 0.1)

logProbPresentSpamSubject <- log(ProbPresentHamSubject)
logProbPresentHamSubject <- log(ProbAbsentHamSubject)

logProbAbsentSpamSubject <-  log(ProbAbsentSpamSubject)
logProbAbsentHamSubject<- log(ProbAbsentHamSubject)
```

```
computeBF_Subject <- function(email) {

  uniqueWords <- extractWords(email, unique = TRUE)
  uniqueSubjectWords <- unlist(extractSubjectWords(email, unique = TRUE))

  words <- uniqueWords[uniqueWords %in% bow]
  subjectWords <- uniqueSubjectWords[uniqueSubjectWords %in% bosw]

  present_indices <- which(names(ProbPresentSpam) %in% words)
  absent_indices <- which(names(ProbPresentSpam) %!in% words)

  present_Subjectindices <- which(names(ProbPresentSpamSubject) %in% subjectWords)
  absent_Subjectindices <- which(names(ProbPresentSpamSubject) %!in% subjectWords)

  SpamPresentProbs <-  logProbPresentSpam[present_indices]
  HamPresentProbs <- logProbPresentHam[present_indices]

  SpamAbsentProbs <- logProbAbsentSpam[absent_indices]
  HamAbsentProbs <- logProbAbsentHam[absent_indices]

  SpamPresentProbsSubject <-  logProbPresentSpamSubject[present_Subjectindices]
```

```
  HamPresentProbsSubject <- logProbPresentHamSubject[absent_Subjectindices]

  SpamAbsentProbsSubject<- logProbAbsentSpamSubject[absent_Subjectindices]
  HamAbsentProbsSubject <- logProbAbsentHamSubject[absent_Subjectindices]

  spamHypothesis <- sum(SpamPresentProbs) + sum(SpamAbsentProbs) + sum(SpamPresentProbsSubject) +
    sum(SpamAbsentProbsSubject)
  hamHypothesis  <- sum(HamPresentProbs) + sum(HamAbsentProbs) + sum(HamPresentProbsSubject) +
    sum(HamAbsentProbsSubject)


  BF <- spamHypothesis - hamHypothesis

  return(BF)
}
```

```
testBFSubject <- lapply(fullEmailsTest, computeBF_Subject)

numTestSpam <- length(which(isSpamTest))
numTestHam <- length(which(!isSpamTest))
tOneVector <- c()
tTwoVector <- c()

spamtestBF <- unlist(testBFSubject[which(isSpamTest)])
nonspamtestBF <- unlist(testBFSubject[which(!isSpamTest)])

cValues_Subject <- sort(unlist(testBFSubject))

tOneVector_Sub <- sapply(cValues_Subject , function(c) {

type1num <- length(which(nonspamtestBF >= c))
typeOneRate <- ( type1num / numTestHam)

return(typeOneRate)
})

tTwoVector_Sub <- sapply(cValues_Subject, function(c) {

type2num <- length(which(spamtestBF < c))
typeTwoRate <- (type2num / numTestSpam )

return(typeTwoRate)
}

)

tTwoVector_Sub[which(tOneVector_Sub < 0.001)[1]]
```

```
## [1] 0.09386733
```

```
spamSenders <- lapply(spamEmails, function(x) { extractSender(x)})

fullEmailsTrain <- allEmails[training_indices]
```

```
fullEmailsTest <- allEmails[test_indices]

fullSpamTrain <- fullEmailsTrain[which(isSpamTrain)]
fullHamTrain <-  fullEmailsTrain[which(!isSpamTrain)]

spamDomains <- tolower(unlist(lapply(fullSpamTrain,function(x){ extractDomain(x)})))
hamDomains <- tolower(unlist(lapply(fullHamTrain, function(x) { extractDomain(x)})))

#bag of domains
alldoms <- c(hamDomains, spamDomains)
bod <- unique(alldoms)
bod <- bod[bod != ""]

countDomainSpam <- c(rep(0, length(bod)))
countDomainHam <- c(rep(0, length(bod)))

countDomainSpamPresent <- c(rep(0, length(bod)))
countDomainHamPresent <- c(rep(0, length(bod)))

names(countDomainSpamPresent) <- bod
names(countDomainHamPresent) <- bod


allSpamDomain <- unlist(spamDomains)
spamTableDomain <- table(allSpamDomain)
spamValuesDomain <- as.vector(spamTableDomain)
names(spamValuesDomain) <- names(spamTableDomain)

spamOverlapDomain<- which( names(countDomainSpamPresent) %in% names(spamValuesDomain) )
countDomainSpamPresent[spamOverlapDomain] <- spamValuesDomain[names(countDomainSpamPresent[spamOverlapD

allHamDomain <- unlist(hamDomains)
hamTableDomain <- table(allHamDomain)
hamValuesDomain <- as.vector(hamTableDomain)
names(hamValuesDomain) <- names(hamTableDomain)

hamOverlapDomain<- which( names(countDomainHamPresent) %in% names(hamValuesDomain) )
countDomainHamPresent[hamOverlapDomain] <- hamValuesDomain[names(countDomainHamPresent[hamOverlapDomain
```

```
CountAbsentHamDomain <- c(rep(0, length(bod)))
CountAbsentSpamDomain <- c(rep(0, length(bod)))

names(CountAbsentSpamDomain) <- bod
names(CountAbsentHamDomain) <- bod

CountAbsentHamDomain <-  CountAbsentHamDomain + 4634 - countDomainHamPresent
CountAbsentSpamDomain <- CountAbsentSpamDomain + 1598 - countDomainSpamPresent

ProbPresentSpamDomain <- (countDomainSpamPresent  + 0.1) / (length(spamTrain) + 0.1)
ProbAbsentSpamDomain  <- (CountAbsentSpamDomain   + 0.1) / (length(spamTrain) + 0.1)

ProbPresentHamDomain <- (countDomainHamPresent  + 0.1) / (length(hamTrain) + 0.1)
ProbAbsentHamDomain  <- (CountAbsentHamDomain  + 0.1) / (length(hamTrain) + 0.1)
```

```
logProbPresentSpamDomain <- log(ProbPresentHamDomain)
logProbPresentHamDomain <- log(ProbAbsentHamDomain)

logProbAbsentSpamDomain <-  log(ProbAbsentSpamDomain)
logProbAbsentHamDomain <- log(ProbAbsentHamDomain)
```

```
## com net  ie  us  cn  dk org  uk  de  ru
## 987 186  84  46  24  20  20  20  16  16
```

```
##       com       net       org        ie        uk com (skip        au       edu
##      1930      1080      1039       458        37        27        25        12
##        ca        tf
##        11         7
```

```
extractWordsDeluxe <- function(x, unique) {
  body <- extractBodyText(x)
  dp_body <- unlist(lapply(body, function(x) deparse(x)))
  full_body <- paste(dp_body, collapse = ' ')
  full_body <- gsub('\\\\.{3}', ' ', full_body)
  full_body <- gsub('[[:digit:]]', ' ', full_body)
  full_body <- tolower(full_body)
  full_body <- gsub('[[:punct:]]', ' ', full_body)
  full_body <- gsub('[[:space:]][a-z][[:space:]]', ' ', full_body)
  full_body <- gsub('[[:space:]]+', ' ', full_body)
  full_body <- gsub('^[[:space:]]+', '', full_body)
  full_body <- gsub('[[:space:]]+$', '', full_body)

  words <- str_split(full_body, " ")

  words <- unlist(words)

  words <- words[!words %in% stopwords]


  if (unique) {
    return(unique(unlist(words)))
  } else {
    return(unlist(words))
  }

}
```

```
computeBFT_Three <- function(email) {

  uniqueWords <- extractWordsDeluxe(email, unique = FALSE)
  domain <- extractDomain(email)

  words <- uniqueWords[uniqueWords %in% bowDeluxe]

  present_indices <- which(names(ProbPresentSpamDeluxe) %in% words)
  absent_indices <- which(names(ProbPresentSpamDeluxe) %!in% words)
```

```
  SpamPresentProbs <-  logProbPresentSpamDeluxe[present_indices]
  HamPresentProbs <- logProbPresentHamDeluxe[present_indices]

  SpamAbsentProbs <- logProbAbsentSpamDeluxe[absent_indices]
  HamAbsentProbs <- logProbAbsentHamDeluxe[absent_indices]

  spamHypothesis <- sum(SpamPresentProbs) + sum(SpamAbsentProbs)
  hamHypothesis  <- sum(HamPresentProbs) + sum(HamAbsentProbs)

  BF <- spamHypothesis - hamHypothesis

  return(BF)
}
```