# Project #4 stat319

Alan Lin

4/26/2022

## Introduction

Spam emails are quite common in this digital world. Many of these are sent via mass emailing and are often detected by some algorithm in most emailing services. In this project, we attempt to identify several properties of spam emails that can help us classify emails as spam or non-spam, through a Naive Bayesian approach. See the attached paper to understand what this approach implies for us. We will be using data from Spam Assassin, https://spamassassin.apache.org/doc.html. While they have their own classification algorithm that they sell, they have also compiled over 9000 emails for training/testing algorithms. In the downloads, emails are split between being spam or not spam(known as ham). Thus, we can use these emails to build our own algorithm. We will see the Type I and II error rates and use that to judge how well our classifier is doing.

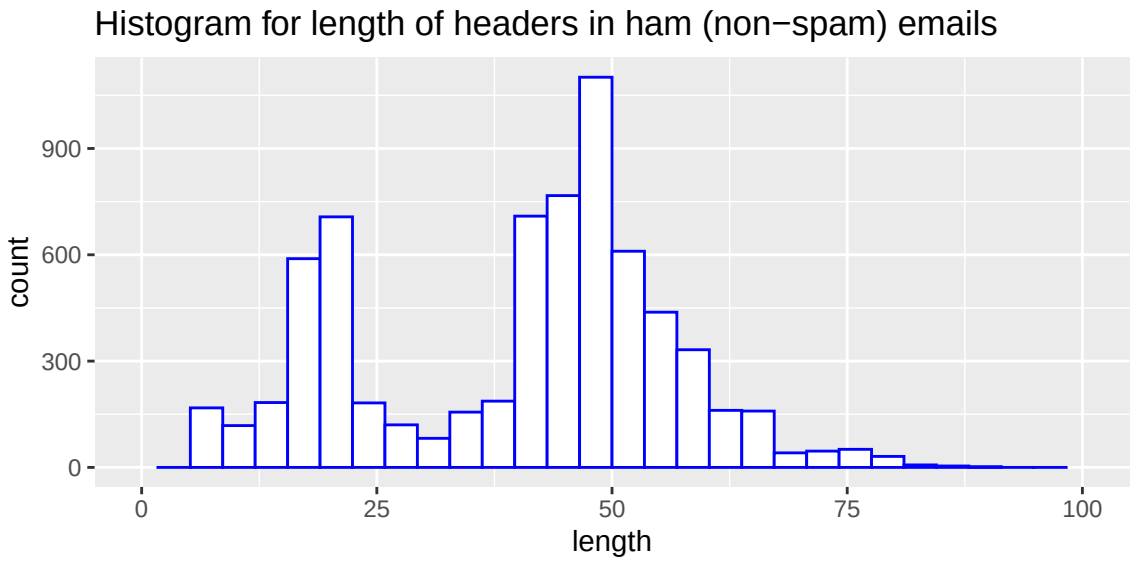
### Deliverable 1:

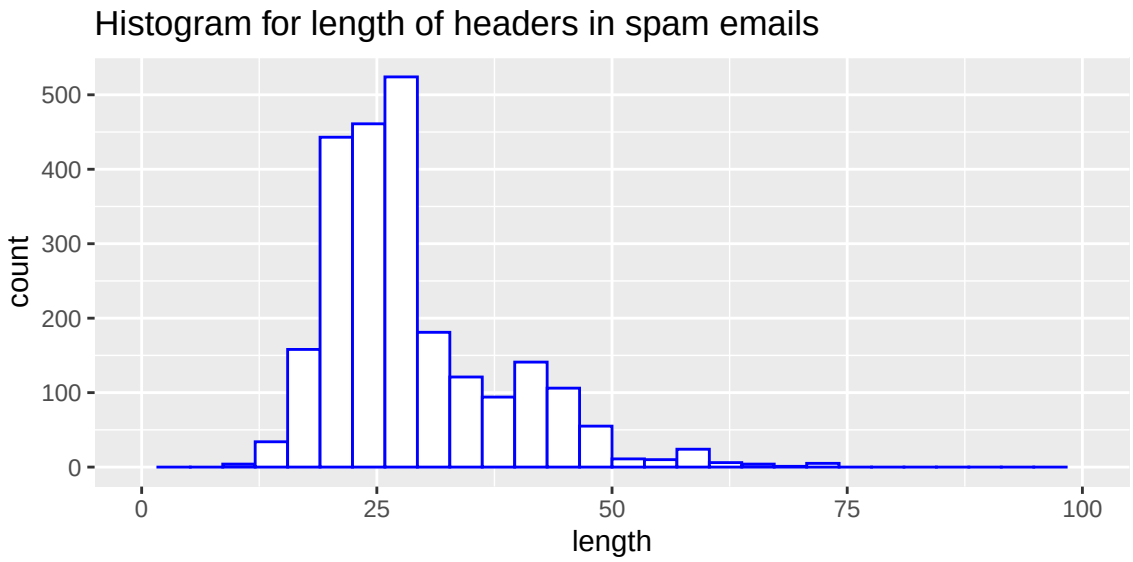
Here we use the splitMessage function that assumes that everything before the first blank line is the header and everything after is the body of the email. First, my function will search each line in an email for an empty string, and returns the vector TRUE/FALSE statements. Then, I can find the index of the first(or even the only) TRUE statement by using min(which(x == TRUE)). Afterwords, it is just a matter of saving all lines before as the header and all lines after this empty string as the body by indexing the email. Then, we can answer the question: do the number of lines in an email header act as a good indicator of spam emails?

First, let's look at the histogram for the number of lines in an email header of all ham emails. It seems bimodal, with two peaks at around 24 lines and 50 lines. On the other hand, the distribution for the number of lines in an email header of all spam emails appears to be unimodal, centered at around 25 lines. I limited the x-axis to be between 0 and 100 lines, in order to display an accurate comparison between spam/ham emails.

Additionally, after removing the extreme outliers for spam emails (more than 4 standard deviations away), the mean number of lines in the header for spam emails is 28.51 and the standard deviation is 9.42, while the ham emails have a mean number of lines in the heaver of 40.28 and a standard deviation of 16.18 (there doesn't seem to be any extreme outliers for ham emails so none were removed).

The number of lines in an email header may not be a very good indicator because the distribution of number lines in an email header for ham emails is bimodal with a center that is pretty much the same as it is for a spam email, around 25-28. This means that it's probably hard to determine if an email is spam or not by header length alone.




**Deliverable #2**

Here we use a new function, called hasAttachment, which takes in an email header, and checks if there is a Content-Type line. This line will sometimes contain the phrase "multipart", which indicates that there is an attachment in the email. Sometimes, there are multiple lines with "Content-Type", so we only use the first one to check if "multipart" is also in the line. The function will return True or False depending on the result.

So, if we apply this function to the headers from the harm_ham directory, we get that there are 95 emails with attachments

Next, we have another function called getBoundary, which extracts the boundary separator from the "boundary=" line in the header of an email. There are several special cases to keep in mind. What the function does is take the line that has the "boundary=" phrase in the header using grep and split the string to two parts, before and after the "boundary=" phrase. Then, we take the string that comes after the phrase and remove characters such as empty characters, back slashes, extra double quotations, and semi-colons. Here are some examples. In some cases, although grep has the ignore.case option, strsplit does not, so I had to add a gsub

line in there to change the phrase "boundary=" no matter what case it is to exactly "boundary=". This allows me to correctly find all characters to the right of the phrase and remove any unwanted characters from the string.

First, we have the 69th email in the easy_ham folder. Clearly, there's an attachment with the email. Following that is the boundary string - which we should extract all characters that aren't semi-colons, extra double quotations, or extra empty spaces. Thus, we'd expect the string "Apple-Mail-2-874629474" to return from the getBoundary function, which we see is the case. Next, we have the 404th email in the easy_ham folder; we see that there's extra double quotations that have been escaped with the backslash. There are also only numbers in this string. We'd expect the getBoundary function to return "————090602010909000705010009", with none of the hyphens lost (since they're not considered special characters to remove from the boundary string). Finally, notice how the third example has the phrase "BOUNDARY=". The getBoundary function receives this string and replaces it with "boundary=" in order to get "EG3OZ08L.0DOB2JAQ-euro.apple.com" as the output.

```
test1 <- sampleHeaders[[69]][17]
test1
```

```
## [1] "Content-Type: multipart/alternative; boundary=Apple-Mail-2-874629474"
```

```
getBoundary(test1)
```

```
## [1] "Apple-Mail-2-874629474"
```

```
test2 <- sampleHeaders[[404]][32]
test2
```

```
## [1] "    boundary=\"------------090602010909000705010009\""
```

```
getBoundary(test2)
```

```
## [1] "------------090602010909000705010009"
```

```
test3 <- hardHamHeaders[[270]][25]
test3
```

```
## [1] "    BOUNDARY=\"EG3OZ08L.0DOB2JAQ-euro.apple.com\";"
```

```
getBoundary(test3)
```

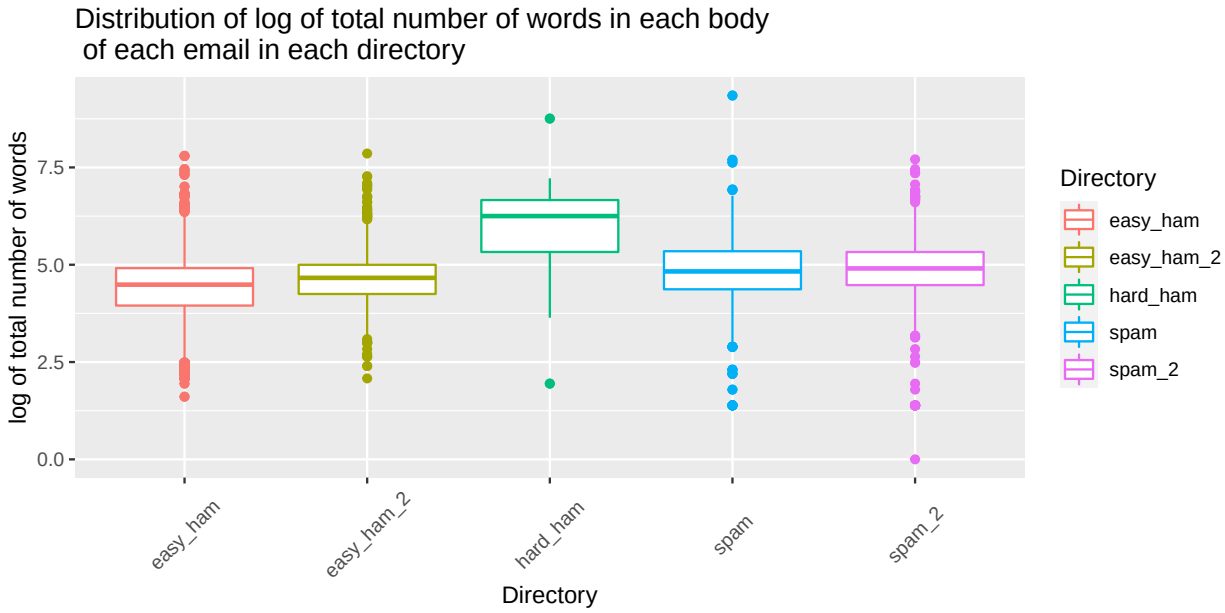
```
## [1] "EG3OZ08L.0DOB2JAQ-euro.apple.com"
```

### Deliverable 3.

It seems that the directory that contains the emails with the most total words is the hard ham directory.

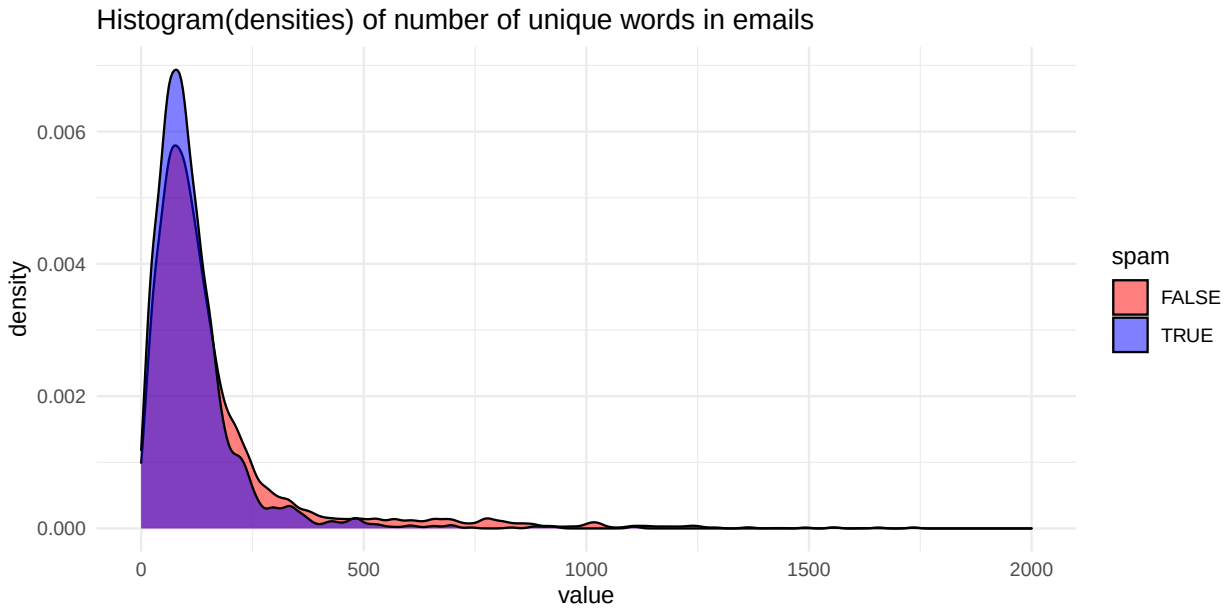
This result is achieved by reading each email and extracting the words using extractWords function by feeding in the body text of the email. We can parse out the body text by splitting an email using splitMessage function(explained at the beginning).

```
## Note: Using an external vector in selections is ambiguous.
## i Use ‘all_of(directories)‘ instead of ‘directories‘ to silence this message.
## i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.
```




**Deliverable #4**

Here we see a histogram showing the densities for the number of unique words appearing in spam/non-spam emails. It's easy to tell apart the two distributions, red for ham, blue for spam, purple is where they overlap. They seem to be heavily right skewed, with the bulk of values falling between 0-200 words for both distributions. This makes it seem that the number of unique words appearing in an email isn't very useful to help identify spam emails. The distribution for ham and spam isn't very distinct from one another.



### Deliverable 5. See attached paper for the derivation of the Bayes Factor.

**Deliverable #6**

**a) How many times more likely is the word "monday" to appear in a non-spam versus a spam email?**

What we can do is find the probability that "78" appears in a non-spam email and divide that by the probability that "monday" appears in a spam email. We already have a named vector of words for the probabilities of each word in spam/non-spam emails, so we can simply look for the word "monday" in each named vector and find the ratio of non-spam to spam probabilities.

It seems that "monday" is 4.952 times as likely to appear in a non-spam email than a spam email.

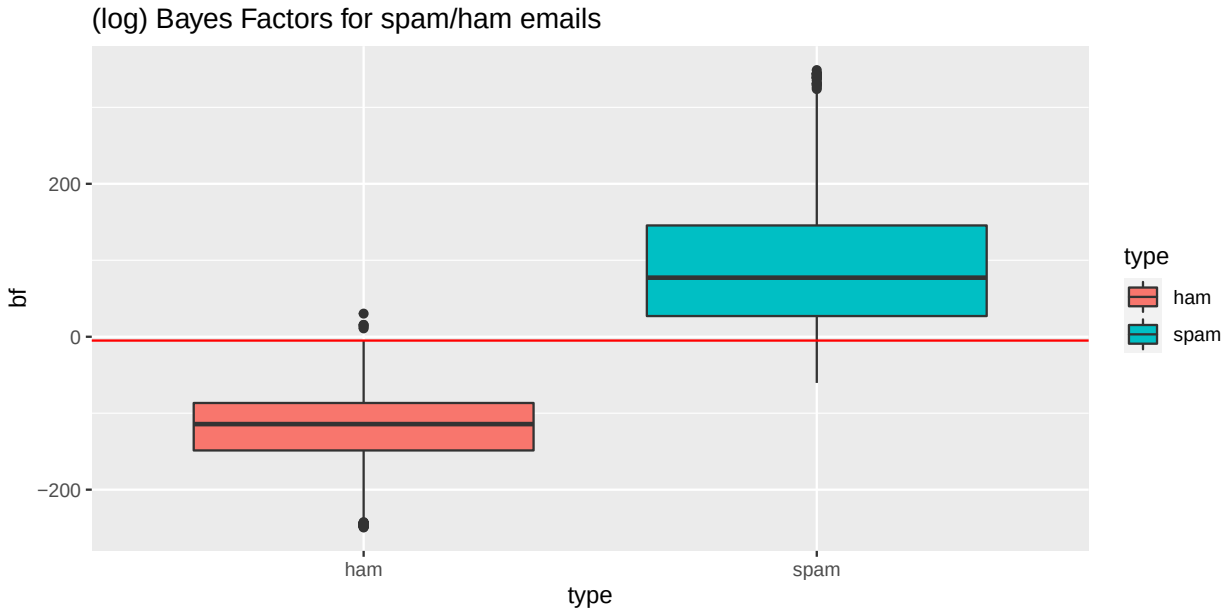
**b) How many times more likely is the word "buy" to appear in a spam vs. a non-spam email?**

Using the same principle as earlier, we search for the word "buy" in the named vector and find the ratio of probabilities, which comes out to be 2.279 times more likely to appear in a spam email vs. a non-spam email.

**Deliverable 7:**

Here we use the computeBF function on all emails in the training set, and separates them based on if they're spam or not. Thus, we have two groups and we can create a side-by-side boxplot for the distribution of the log Bayes Factors for each group, while setting the y-limits of the plot to a narrow scope of interest (around -250 to 350).

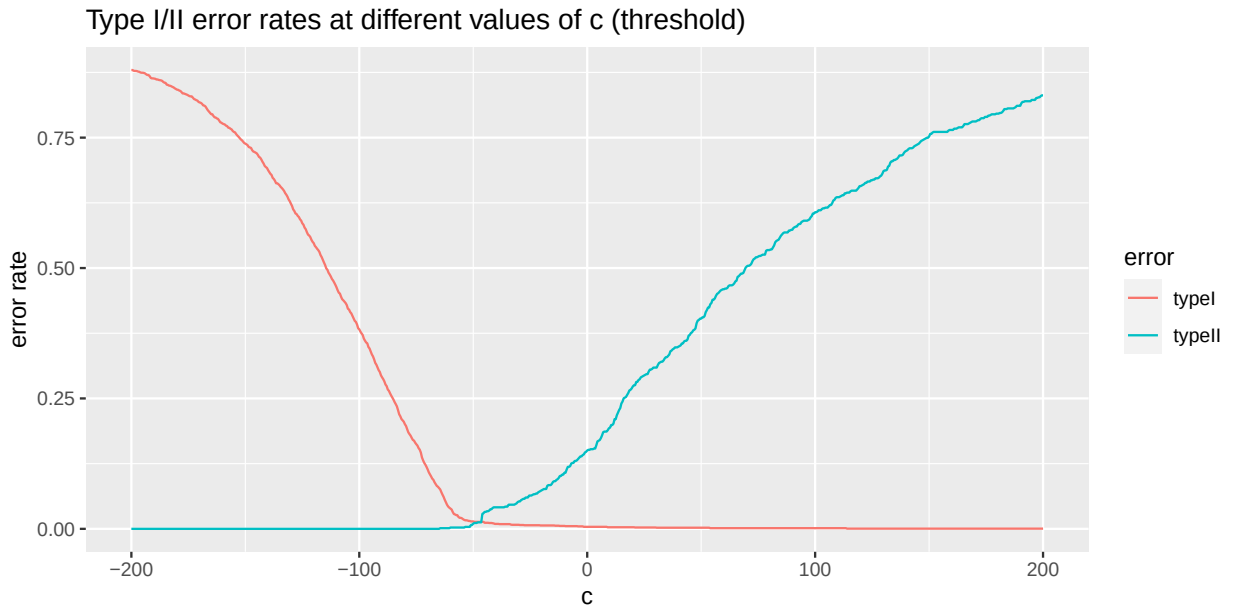
We see that the log BF of the spam emails are on average positive, with more than 75% of the emails above 0, while almost all the ham emails have log BF of 0 or below. Thus, I'd estimate that a good threshold value (c) for classifying when an email is spam is going to be in low negative numbers, possibly around -5 to -10. This allows us to capture more spam emails while also reducing the number of ham emails being classified as spam. This is visualized with the red horizontal line at -5. We see that there are very few ham emails that are caught and around 80% of spam emails that are above this threshold.



**Deliverable 8**

First we need to find the log BF for emails in the test set. We can do that easily by just feeding each email in the test set to the computeBF function by sapply. After that, we can use lapply on a sequence of possible threshold (c) values to generate a vector of boolean values that correspond to whether the log BF value is greater or equal to the threshold value. A value of TRUE will indicate an email being classified as spam. Thus, by finding the number of non-spam emails that have TRUE and dividing by the number of non-spam emails in the test set, we can find the Type 1 error, or the false detection error. Similarly, we find the number of spam emails that have FALSE and divide by the number of spam emails in the test set, we have the Type II error, or spam emails that were wrongly classified as ham.

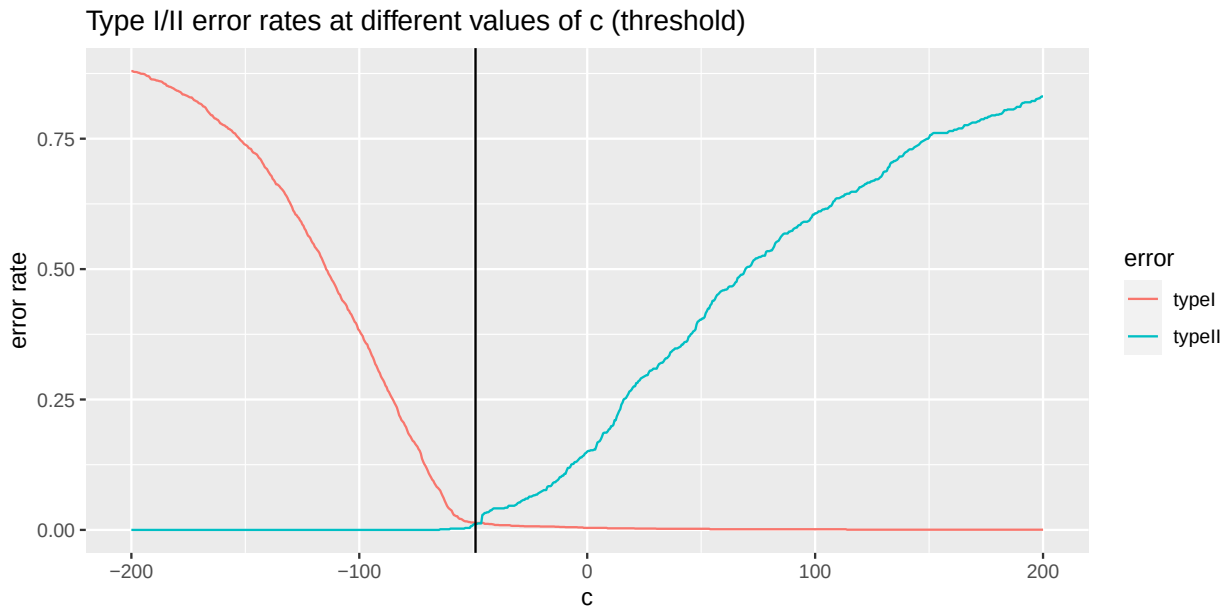
Then, we simply plot these values into a scatterplot with two lines, where we can see that at low values of c (really negative!), Type I error rate is high, or that many emails are wrongly classified as spam when they're not. At the same time, as the values of c increase, only a small number of emails are classified as spam, so the Type II error rates are pretty high because we mistakenly classified a lot of spam emails as ham.




**Deliverable 9**

Assuming we want to equal Type I and Type II error rates, the value of c that most nearly achieves this is around -49. The Type I error at this threshold value is 0.0134 and the Type II error is 0.0113

That means the classifier is wrongly classifying ham emails as spam 1.34% of the time and mistaking spam emails as ham 1.13% of the time.




```
## [1] 0.01337937
```

```
## [1] 0.01126408
```

**Deliverable 10**

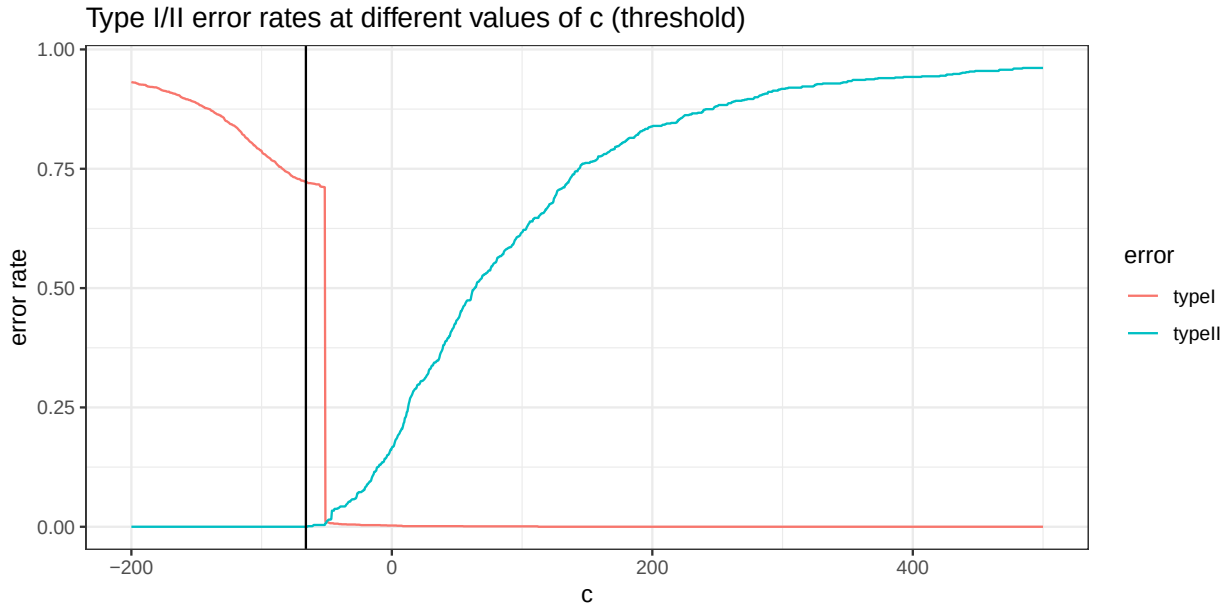
If we want Type I error to be less than 0.001, the smallest Type II error rate that we can achieve is 0.6446, or an error rate of 64.46%. That means that although we rarely classify ham emails as spam(0.1% chance), there is an extremely high chance (64.46%) that we let a spam email slip through as ham. Thus, our spam classifier is keeping most legitimate emails in the inbox but also many spam emails. This reciprocal behavior is the cost of choosing to minimize one type of error over the other.

```
## [1] 0.6445557
```

**Extension**

I will be grabbing the 500 most common words from this website: https://www.smart-words.org/500-most-commonly-used-english-words.html, with which I will then grab only the "short" words(3 or less characters). Though there are many words here, the shorter ones tend to be the pronouns, prepositions, and conjuctions, etc, which may not be very indicative of a spam email. Then I can filter these common words out from the vector of words in each body text. I will also be removing common features of a website, especially the top level domain extension such as "com", "org", and many more that I found from https://www.icdsoft.com/blog/what-are-the-most-popular-tlds-domain-extensions/. From there, we calculate log probabilities and use computeBF as we did before. We can check to see if this helps the classifier by looking at the Type I and Type II error rate when they are equivalent, as well as finding the smallest Type II error rate that goes with a Type 1 error rate of less than 0.1%. Looking at the graph, we see that the threshold for which the two error rates are similar to each other is at around -66, where the Type I error is 0.0518 and the Type II error is 0.0526. Both of these values are larger than they were before removing the common words from the body texts, which suggests that it may not be useful to remove them because the error rates for both types are larger than they were when the common words were still in the emails. In addition, in order to achieve a Type I error rate of 0.1%, the threshold value is set at 462.5, where the Type II error rate is 92.87%. This

is much higher than previously found, which continues to suggest that removing these common words and website characteristics might not conducive to classifying spam emails. Perhaps they rely on many of these words because they are simple and really help make a message more direct, such as stating, "YOU could really benefit from it" or linking a suspicious website in the case of scam emails or just a simple algorithm putting together simple short words in an automated spam email generator.




```
## [1] 0.7229176
```

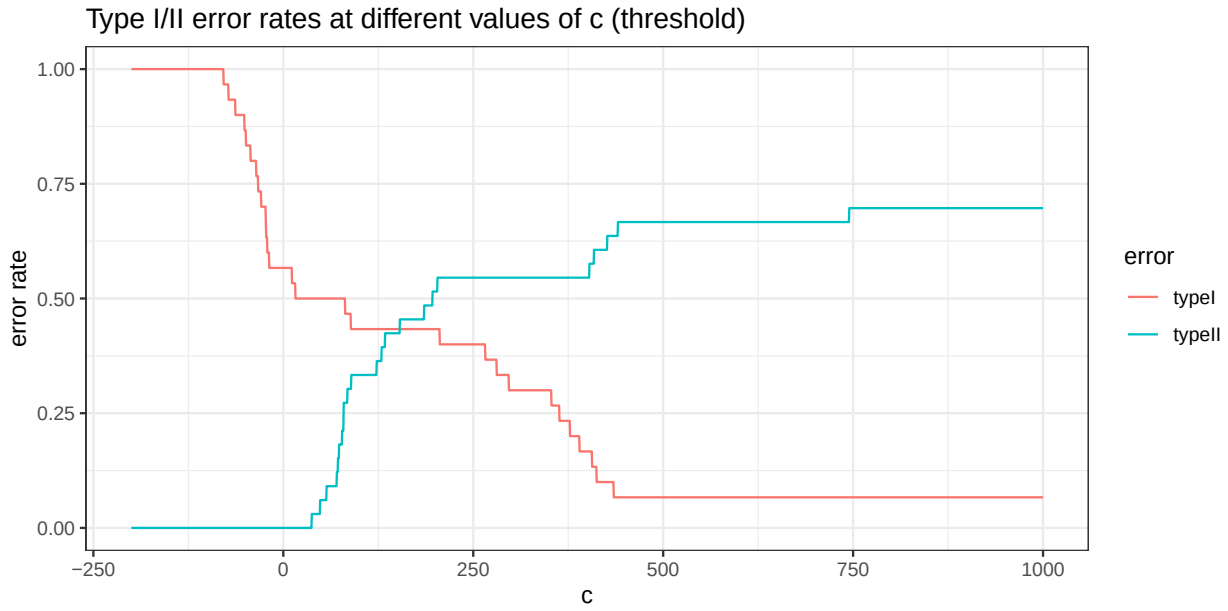
```
## [1] 0
```

```
## [1] 0.4618273
```

We can also test our classifier on my own emails.

Here I have 30 ham emails and 33 spam emails. Let's use the extractWords function that keeps the common words and the website features because I've shown that removing those words seems to worsen the effectiveness of the spam classifier.

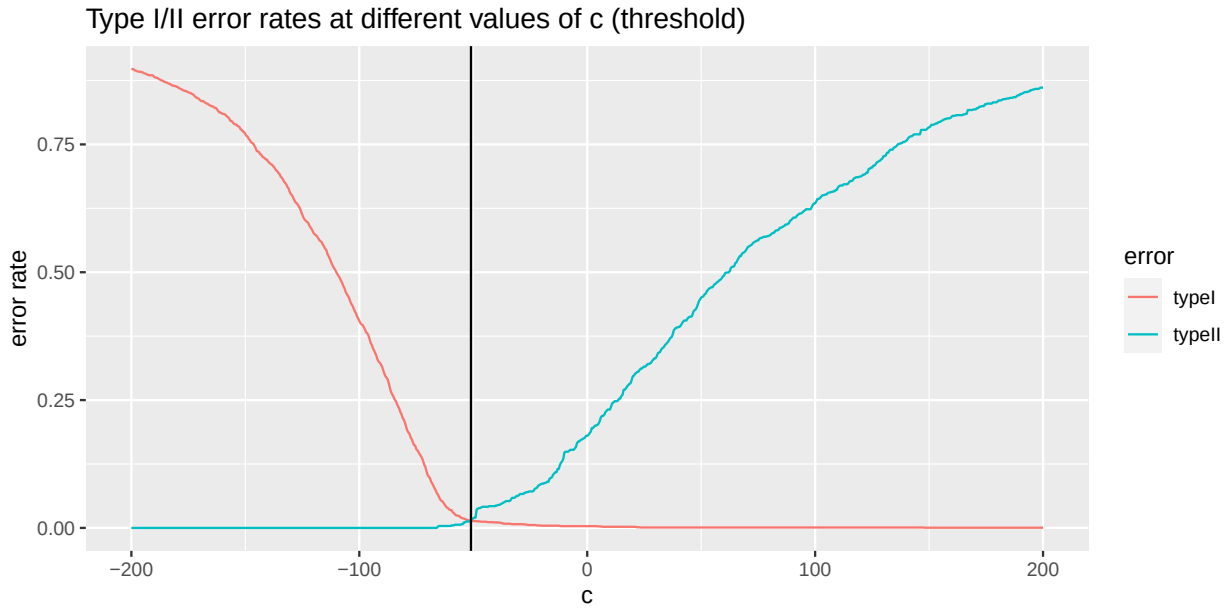
It seems that using the logprobabilities from the training set from before leads the classifier to require a much higher threshold value in the positives(!) to distinguish between spam and ham emails. When the error rates are similar, the classifier actually doesn't really know how to tell spam from ham, since the error rates are around 50%. In addition, at it's minimum, Type I error can go down to around 7%, while the Type II error plateaus around 70%. Neither of these percentages are really that good, so it seems that the sample emails from my spam folder and some emails from my inbox are giving our classifier a hard time because it is almost 50/50 at marking a spam email as ham or marking a ham email as spam. Perhaps my choice of emails may not have been the best, as some of it were from mailing listserves, so it has the format of a mass email, which could be very similar to qualities found in spam emails.




My last extension attempts to divide the corpus of emails into three groups, in order to finetune the threshhold value of c for the testing set. What I'm going to do is use the training set to create the bag of words, and logprobabilities that will then be tested on the cross validation set, which will help me narrow down a threshold value of c that is associated with equivalent error rates for Type I and Type II errors. Then, using that threshold value of c, I will evaluate the error rates for both Type I and II errors of the test set.

As we can see, the threshold value that results in similar error rates is around -51. So, let's quickly find the associated error rates at that threshold value. The type I error rate is 1.38% and the type II error rate is 1.76%. That's not too bad... marking a ham email as spam 1.38% of the time is pretty good and marking a spam email as ham at 1.76% of the time is pretty darn good.

So, perhaps we could look at the threshold value that reduces type I error to .01%. We find that the first occurrence of Type I error rate going below .01% is when the threshold value is set at 23.5. So, how does the testing set do at this threshold? Well, the Type I error rises to 0.04%, while the Type II error jumps up from 1.76% to 28.16%. That's not a horrible tradeoff. Either option is good depending on priorities.




```
## [1] 0.01337937 0.01001252
```

```
## [1] 23.5
```

```
## [1] 0.0004315926 0.2816020025
```

## Appendix

Code for splitMessage function. Returns a list of header and body of email.

```
splitMessage <- function(email){
  a <- sapply(email, function(x) which(x == ""))
  index <- min(which(a == TRUE))
  header <- email[1:index - 1 ]
  body <- email[index + 1:length(email)]

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

Code for hasAttachment function. Returns boolean whether or not the phrase "multipart" appears where there is a Content-type line. This indicates the email has an attachment.

```
hasAttachment <- function(header){

  ind <- grep("content-type", header, ignore.case = T)

  if(length(ind) == 0) return(FALSE)

  grepl("multipart", header[ind[1]], ignore.case = T)


}
```

Code for getBoundary function. Handles multiple special cases.

```
getBoundary <- function(header){
  b <- grep("boundary=", header, value = T, ignore.case = T)
  c <- gsub("boundary=", "boundary=", b, ignore.case = T)
  d <- strsplit(c, split = "boundary=")
  d1 <- sapply(d, "[[", 2)

  gsub("[ \";]", "", d1)
}
```

extractBodyText function, returns body text, lines of the email that are found between the first boundary line and the second boundary line.

```
extractBodyText <- function(email){
  a <- splitMessage(email)
  if(hasAttachment(a$header)) {
    s <- getBoundary(a$header)
    lines <- grep(s, a$body, fixed = T)
    if(length(lines) > 1){
      f <- lines[1]
      l <- lines[2]
      return(a$body[(f+1):(l-1)])
    } else {
      return(a$body[-lines])
    }
  } else {
    return(a$body)
  }
}
```

extractWords function, Returns a vector of unique words by default, total words if prompted by setting unique = f.

```
extractWords <- function(body, unique = T){
  d <- tolower(body)
  d1 <- gsub('[^abcdefghijklmnopqrstuvwxyz]', ' ', d)
  d2 <- gsub('\\b[[:lower:]]{1}\\b', " ", d1)
  d3 <- grep("[[:alpha:]]", d2)
  words <- d2[d3]
  collapse <- paste0(words, collapse = " ")
  un <- unlist(strsplit(collapse, " "))
  w <- un[which(un != "")]
  if(!unique) return (w)
  unique(w)
}
```

Have the option to read in 5 different directories, corresponding to its name: easy_ham, easy_ham2, hard_ham, spam, spam2. Give one of those into the function and it will return the emails within the directory.

```
readEmailDirectory <- function(directory) {
  str <- "C:/Users/super/Documents/stat319/messages/messages/"
  str1 <- paste(str, directory, sep = "")

  emails <- list.files(str1, full.names = T)
  lapply(emails, function(x) readLines(x))
}
```

Computes the associated log Bayes Factor for a given set of unique words in the body of an email.

Revised version of extractWords that filters out common short words and website features.

```
extractWords1 <- function(body, unique = T){
  d <- tolower(body)
  d1 <- gsub('[^abcdefghijklmnopqrstuvwxyz]', ' ', d)
  d2 <- gsub('\\b[[:lower:]]{1}\\b', " ", d1)
  d3 <- grep("[[:alpha:]]", d2)
  words <- d2[d3]
  collapse <- paste0(words, collapse = " ")
  un <- unlist(strsplit(collapse, " "))
  w <- un[which(un != "")]
  w1 <- w[which(!(w %in% common2))]
  w2 <- w1[which(!(w1 %in% website))]
  if(!unique) return (w2)
  unique(w2)
}
```