STATEMENT ON ASSIGNMENT ABOUT NOT CONSULTING RESOURCES:

"I completed this assignment myself without consulting books or tutorials that specifically addressed this data set."

Report for Project 2: Text Processing: Emails For SPAM/HAM Classification

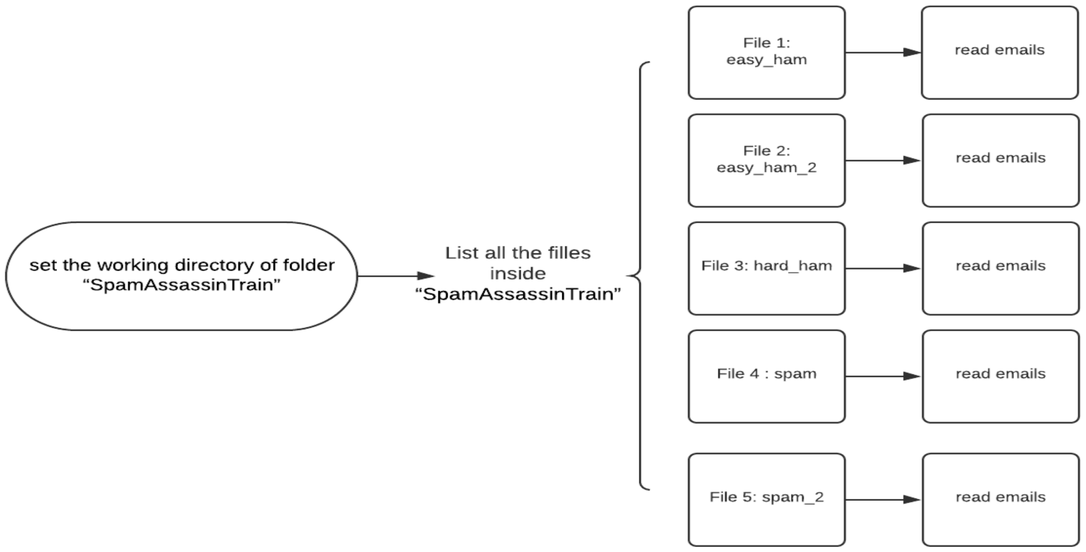
Read the data and get information

The data that we are going to deal with is email messages that can be divided into several parts, including header, body and attachment (optional). Each email message is inside folders with folder names mentioning either “spam” or “ham”. All the “spam” or “ham” folders are inside a folder called “SpamAssassinTrain”. In this report, I will introduce the way that I read emails from folders and how I derive information from each email to finally construct a data frame for email classification analysis.

Read Emails

My idea is to create a loop to read all the folders, and then I read emails inside each folder. I begin with the function “*setwd()* ” to set the working directory of folder “SpamAssassinTrain”. Then I use a loop with the function “*list.files()”* to list all the five files containing emails inside “SpamAssassinTrain”. Next, I set up another loop to read all email files inside the five folders with the function “*readLines().”* Up to now, I am able to read all email files inside the folder “SpamAssassinTrain”. In this case, I don’t need to write a function such as “*GetHeader*” or “*GetBody*” to get information from each file, since the loop will automatically read all the email messages in different files.

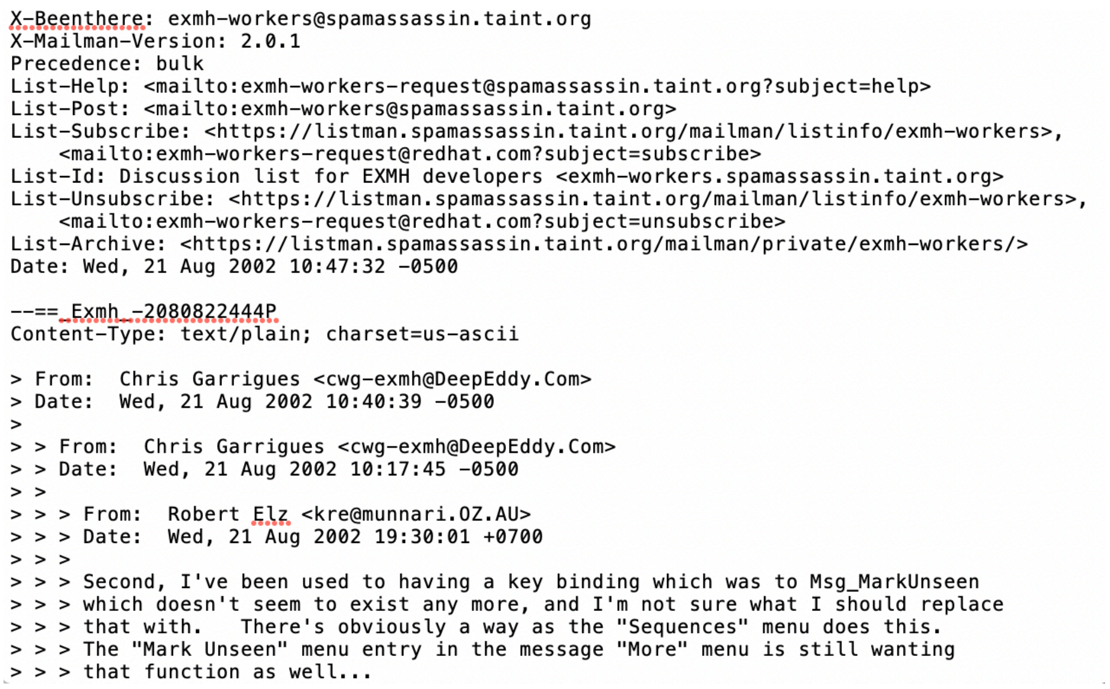
A flow chart representing the process of reading emails is shown below:



Split Emails

The next step is to split each data. The way that I subtract header from an email is to find the position of the empty line. If the entire email doesn't have an empty line, then it means that the email doesn't contain a header, which indicates that the email is not in a right format, and hence we skip it.

After we finish subtracting the header, we need to split the rest of email into body and attachments. The splitting sign is the beginning boundary of attachment. However, one challenge is that sometimes the body of emails are below the beginning boundary, typically when the email is a plain text with no attachment. Like the graph below, there is a string "Content-Type: text/plain" after the line of boundary string, and below which is the content of email body instead of attachment. Therefore, only using boundary string as a splitter is not enough, we also need to test if the content type is a plain text.

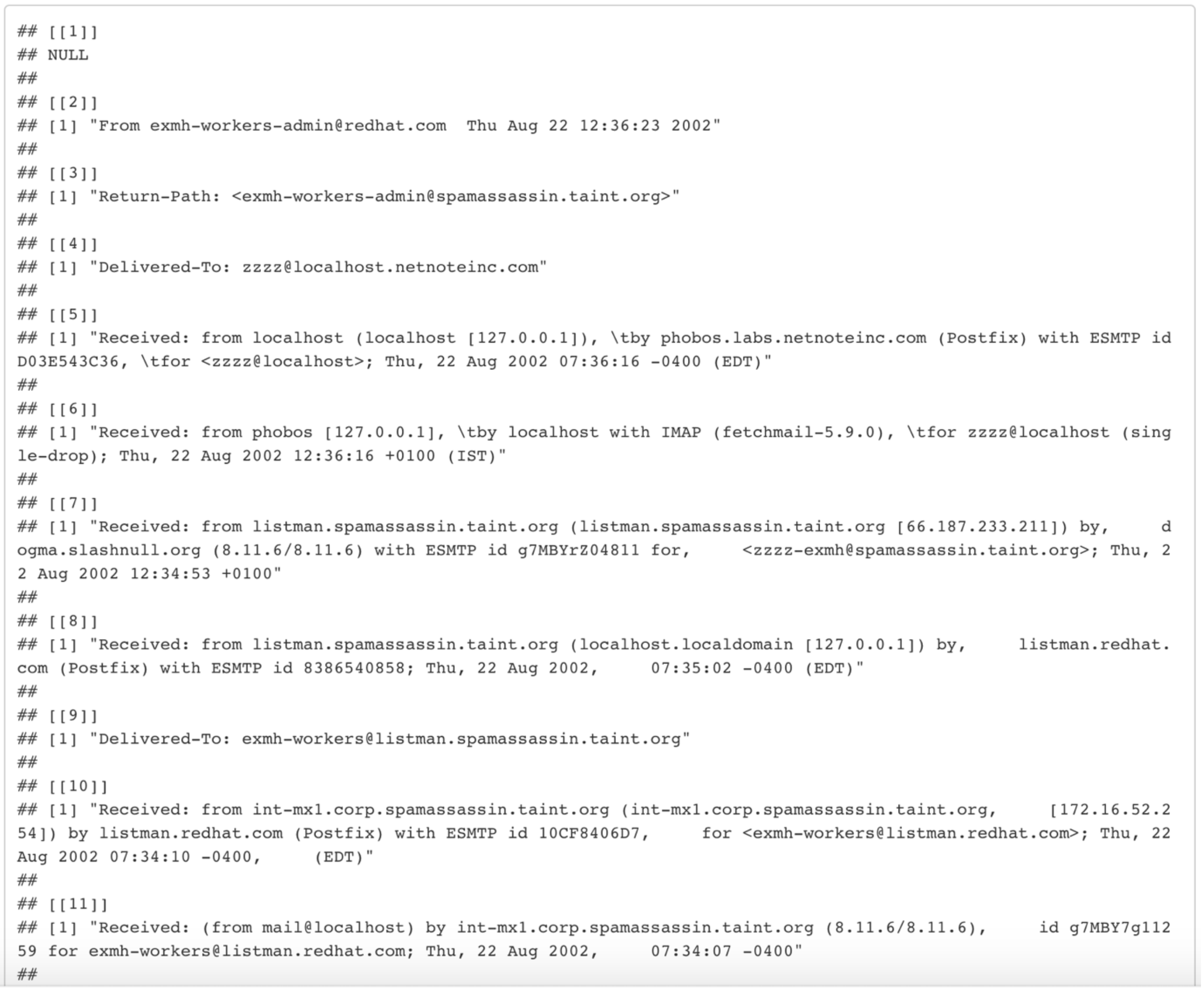

```
X-Beenthere: exmh-workers@spamassassin.taint.org
X-Mailman-Version: 2.0.1
Precedence: bulk
List-Help: <mailto:exmh-workers-request@spamassassin.taint.org?subject=help>
List-Post: <mailto:exmh-workers@spamassassin.taint.org>
List-Subscribe: <https://listman.spamassassin.taint.org/mailman/listinfo/exmh-workers>,
    <mailto:exmh-workers-request@redhat.com?subject=subscribe>
List-Id: Discussion list for EXMH developers <exmh-workers.spamassassin.taint.org>
List-Unsubscribe: <https://listman.spamassassin.taint.org/mailman/listinfo/exmh-workers>,
    <mailto:exmh-workers-request@redhat.com?subject=unsubscribe>
List-Archive: <https://listman.spamassassin.taint.org/mailman/private/exmh-workers/>
Date: Wed, 21 Aug 2002 10:47:32 -0500

--==_Exmh_-2080822444P
Content-Type: text/plain; charset=us-ascii

> From:  Chris Garrigues <cwg-exmh@DeepEddy.Com>
> Date:  Wed, 21 Aug 2002 10:40:39 -0500
>
> > From:  Chris Garrigues <cwg-exmh@DeepEddy.Com>
> > Date:  Wed, 21 Aug 2002 10:17:45 -0500
> >
> > > From:  Robert Elz <kre@munnari.OZ.AU>
> > > Date:  Wed, 21 Aug 2002 19:30:01 +0700
> > >
> > > Second, I've been used to having a key binding which was to Msg_MarkUnseen
> > > which doesn't seem to exist any more, and I'm not sure what I should replace
> > > that with.   There's obviously a way as the "Sequences" menu does this.
> > > The "Mark Unseen" menu entry in the message "More" menu is still wanting
> > > that function as well...
```

The way for me to look for attachments in emails is to find the string of “Content-Type:” that are not followed by “text/plain”. I firstly use “*grepl()”* function to look for lines that contains “--” and “=” to recognize the attachment boundary. Not surprisingly, I found many “text/plain”, meaning that we misclassified the email body as part of the attachment. Therefore, I use *grepl()* again and match the string “plain” (ignore the case) to the line that is exactly one line below my attachment boundary. If the *grepl()* function returns all FALSE, it means that the attachments may exist. Then we can include all the lines below the boundary as attachments. If the *grepl() function* returns any TRUE, it means that the email is a plain text (Content-Type: text/plain). Then I will conclude that the email has no attachment. However, surprisingly, sometimes the email is not a plain text even though it states “Content-Type: text/plain”, hence I have to continue reading lines to see if another attachment boundary exists. One of the examples of this can be found in file “*00014.cb20e10b2bfcb8210a1c310798532a57*” in “*easy_ham*”.

To be specific, the regular expressions that I used for matching specific strings are: `SL = grep('(^--.*=+)',Rest); grepl("plain", Rest[SL+1], ignore.case = TRUE)`, where the list "Rest" is all the lines except for the email header; "Rest[SL+1]" means the email line exactly below the attachment boundary (which is "*Rest[SL]")*, where strings of "Content-Type:" exists.

Derive Variables

We have already divided emails into header, body and attachment. Now we can dig some information from these structures. One challenge occurs when I derive information from headers. Since information for some keys may wrap across multiple lines, we have to group lines in the header with respect of keys and then merge the lines for each group into a string, like the format shown below:



```
## [[1]]
## NULL
##
## [[2]]
## [1] "From exmh-workers-admin@redhat.com  Thu Aug 22 12:36:23 2002"
##
## [[3]]
## [1] "Return-Path: <exmh-workers-admin@spamassassin.taint.org>"
##
## [[4]]
## [1] "Delivered-To: zzzz@localhost.netnoteinc.com"
##
## [[5]]
## [1] "Received: from localhost (localhost [127.0.0.1]), \tby phobos.labs.netnoteinc.com (Postfix) with ESMTP id
D03E543C36, \tfor <zzzz@localhost>; Thu, 22 Aug 2002 07:36:16 -0400 (EDT)"
##
## [[6]]
## [1] "Received: from phobos [127.0.0.1], \tby localhost with IMAP (fetchmail-5.9.0), \tfor zzzz@localhost (sing
le-drop); Thu, 22 Aug 2002 12:36:16 +0100 (IST)"
##
## [[7]]
## [1] "Received: from listman.spamassassin.taint.org (listman.spamassassin.taint.org [66.187.233.211]) by,     d
ogma.slashnull.org (8.11.6/8.11.6) with ESMTP id g7MBYrZ04811 for,     <zzzz-exmh@spamassassin.taint.org>; Thu, 2
2 Aug 2002 12:34:53 +0100"
##
## [[8]]
## [1] "Received: from listman.spamassassin.taint.org (localhost.localdomain [127.0.0.1]) by,     listman.redhat.
com (Postfix) with ESMTP id 8386540858; Thu, 22 Aug 2002,     07:35:02 -0400 (EDT)"
##
## [[9]]
## [1] "Delivered-To: exmh-workers@listman.spamassassin.taint.org"
##
## [[10]]
## [1] "Received: from int-mx1.corp.spamassassin.taint.org (int-mx1.corp.spamassassin.taint.org,     [172.16.52.2
54]) by listman.redhat.com (Postfix) with ESMTP id 10CF8406D7,     for <exmh-workers@listman.redhat.com>; Thu, 22
Aug 2002 07:34:10 -0400,     (EDT)"
##
## [[11]]
## [1] "Received: (from mail@localhost) by int-mx1.corp.spamassassin.taint.org (8.11.6/8.11.6),     id g7MBY7g112
59 for exmh-workers@listman.redhat.com; Thu, 22 Aug 2002,     07:34:07 -0400"
##
```

To achieve this format, I set up a loop to get positions of lines each with a key in headers (we call those lines as “key lines”). Then I merge the lines with a range which start with a key line and end up with the line before the next key line. Then, we can turn those lines into a big string, so that all values of the key are included in that string.

The next step is to derive variables. In this section, I would like to use a chart to introduce how I derive variables. After gaining all the lists of variables, we should put them into a data frame. Therefore, I unlisted them and combined them to a data frame.

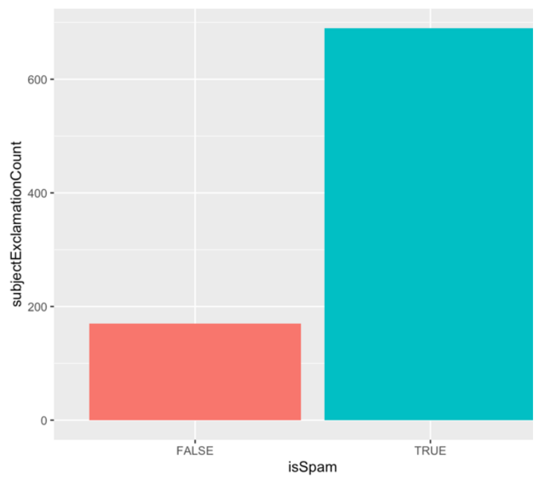
Variable Names	***Methods of derivation***
numLinesInBody	Use function "length()" to calculate the number of lines of email body
isRe	In the headers of email, use the function "grepl("^Subject: Re:+")" to match a string start with "Subject: Re:".
bodyCharacterCount	Use function “sum(nchar(...))" to calculate the sum of character numbers in each body message. Notice that we also uses "encoding = "latin1"" when reading each message because of the existence of irregular encodings
replyUnderline	Use grep() to get the position of Reply-To field. Then use regular expression: any(grep("^Reply-To:.*[a-zA-Z0-9]+.*_+|^Reply-To:.*_+.*[a-zA-Z0-9]+", Head)), where “Head” is all the content of Header.
subjectExclamationCount	Use function "grep("^Subject:+", Head, value = TRUE)" to find strings that contain subject information. Then use function str_count() to figure out the number of "!" inside that string.
subjectQuestCount	The same method as the deirvation of "subjectExclamationCount". Notice that the regular expression of the question mark is "\\?"

numAttachments	1) read lines of all attachments (function: readLines) 2) convert the lines into a string and store all of them in a list (function: toString()) 3) convert the list as a vector (as.vector()) 4) count of number of string "Content-Type:" for each attachment string inside the vector (function: str_count())
priority	use regular expression: *any(grepl("\\X-Priority.*High|\\X-Smell-Priority.*High", Head, ignore.case = TRUE))* to match high priority emails. Notice that we use any() so that it will return true if any line satisfies is matched.
numRecipient	use grep(…, value = TRUE) to match strings start with "Cc:" and "To:". Then use str_count() to count the number of "@" inside strings.
percentCapitals	use str_count() to calculate the number of upper case letters. Then divide the number by the total amount of character numbers in email body
isInReplyTo	isInReplyTo[i] = any(grepl("^In-Reply-To:", Head))
subjectPunctuationCheck	subjectPunctuationCheck[i] = any(grepl("^Subject:.[[:punct:]]+.|^Subject:.[0-9]+." ,Head))
hourSent	1) use grep(…, value = TRUE) to find the string begin with "Date:". 2) Split the string with ":" first, hence the hour number is the last element of the first fragment of the string. 3) Split the first fragment with "", hence the last piece of the string is the number of hours. 4) use as.numeric() to convert the string into number
multipartText	multipartText[i] = any(grepl("multipart", Head, ignore.case = TRUE))
numDollarSigns	numDollarSigns[i] = str_count(toString(Body), pattern = "\\$")
isSpam	isSpam[i] = any(grepl("spam+",Foldernames[j])); Note that the "Foldernames[j]" means the jth email folder(including easy_ham, spam, spam_2...)

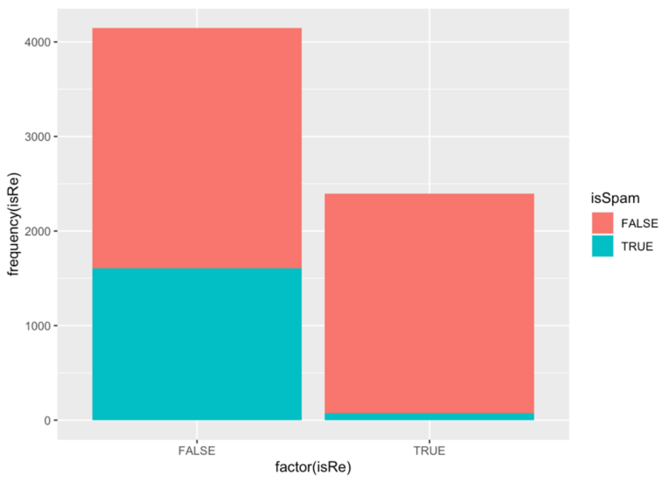
Explore the data

From the summary of variables, I found that “isInReplyTo”, “isRe” and “subjectExclamationCount” are possible variables that can help us to discriminate between Spam and Ham. Noticing that we have read 4864 of Ham emails (isSpam = FALSE) and 1676 of Spam emails (isSpam = TRUE), the result of Graph 1 indicates that the probability of noticing a “!” in Spam emails is much higher than that of noticing a “!” in Ham emails, because of the relatively higher occurrence of “!” with a relatively lower total amount of Spam emails.

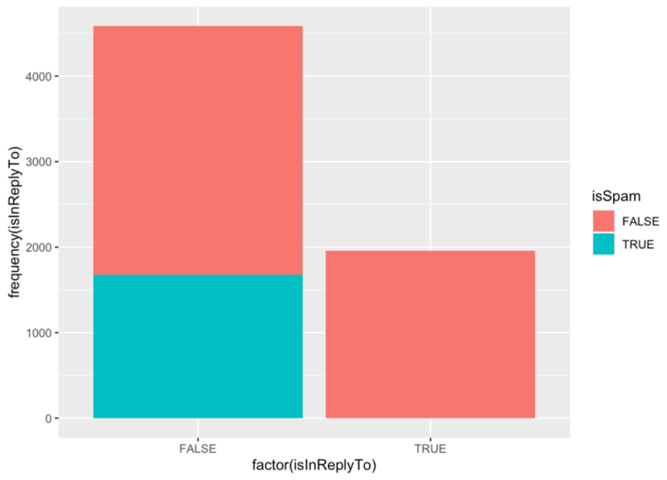
Graph 1



Graph 2



Graph 3



The rest of the two graphs compares the occurrence of results of “isRe”and “isInReplyTo” according to the email types. From these two graphs, we can tell that we can hardly see any Spam email with a subject beginning with “Re:” or has a “In-Reply-To” field, indicating that a Spam email would seldom be transmitted as a reply.

Besides of the three variables, some other variables may also be a candidate sign for email discrimination, such as “priority” and “numDollarSigns”. From the summary result of the data frame, there are 9 out of 1676 Spam emails with high priority, comparing to 1 out of 4864 in Ham emails; Also, the mean of “$” numbers is 3.445 for Spam emails and 0.661 for Ham emails. No wonder that many emails with commercial advertisements are considered as Spam emails.