STAT480_Homework_4

Bin Feng

```
# include library
require("knitr")
```

```
## Loading required package: knitr
```

```
# set working directory
opts_knit$set(root.dir = "~/Stat480/Homework")
```

Setup

```
# run setup code from another .R file
source("HW4Setup.R")
```

```
## Loading required package: NLP
```

```
## Warning in FUN(X[[i]], ...): incomplete final line found on '/home/
## binfeng2/Stat480/RDataScience/SpamAssassinMessages/messages/hard_ham/
## 00228.0eaef7857bbbf3ebf5edbbdae2b30493'
```

```
## Warning in FUN(X[[i]], ...): incomplete final line found on '/home/
## binfeng2/Stat480/RDataScience/SpamAssassinMessages/messages/hard_ham/
## 0231.7c6cc716ce3f3bfad7130dd3c8d7b072'
```

```
## Warning in FUN(X[[i]], ...): incomplete final line found on '/home/
## binfeng2/Stat480/RDataScience/SpamAssassinMessages/messages/hard_ham/
## 0250.7c6cc716ce3f3bfad7130dd3c8d7b072'
```

Since the "HW4Setup.R" file involves larger number of lines, it is directly called from another file instead of pasting every line in the same file. Please put the setup file in the same directory while running this code.

Question 1

Create a function computeMsgLLR2 which implements the log of ratios of products of probabilities formula for the log likelihood ratio statistic. Compare the results from this definition with the results from the computeMsgLLR function used in the text which used the sum of differences of log probabilities. Specifically, compare accuracy for this formula compared to the one used in class (Hint: to estimate relative accuracy you should look at (observed-expected)/expected, and treat the results from computeMsgLLR2 as observed and the results from computeMsgLLR as expected) and note any issues that arise with non-representable numbers (e.g. very large or very small intermediate results that result in infinite, incorrect 0, or not a number results from your function).

```
# original code given in the lecture
computeMsgLLR = function(words, freqTable)
{
  # discards words not in training data.
  words = words[!is.na(match(words, colnames(freqTable)))]
  # find which words are present
  present = colnames(freqTable) %in% words
  # calculate the log likelihood
  sum(freqTable["presentLogOdds", present]) +
    sum(freqTable["absentLogOdds", !present])
}

# calculate the log likelihood based on the formular specified in the question
computeMsgLLR2 = function(words, freqTable)
{
  # discards words not in training data.
  words = words[!is.na(match(words, colnames(freqTable)))]
  # find which words are present
  present = colnames(freqTable) %in% words
  # calculate the log likelihood
  log(prod(freqTable["spam", present])/prod(freqTable["ham", present])) +
    log(prod(1 - freqTable["spam", !present])/prod(1 - freqTable["ham", !present]))
}

# results using the original code
testLLR1 = sapply(testMsgWords, computeMsgLLR, trainTable)
# results using the changed formular
testLLR2 = sapply(testMsgWords, computeMsgLLR2, trainTable)

relative.accuracy = (testLLR2 - testLLR1) / testLLR1
# count NaN frequency
sum(is.nan(relative.accuracy[1:1000]))
```

```
## [1] 136
```

```
# count Inf frequency
sum(is.infinite(relative.accuracy[1:1000]))
```

```
## [1] 75
```

Codes for log likelihood and relative accuracy calculation are shown above. Note that using the formular specified in the question can result in many infinite or not a number outputs. For the first 1000 cases, there are 136 NaN and 75 Inf. These non-representable numbers will largely influence the accuracy of spam-ham classification.

Question 2

Do exercise Q.13 from page 167 of Data Science in R: A Case Studies Approach to Computational Reasoning and Problem Solving, by Deborah Nolan and Duncan Temple Lang. Within the exercise, construct two functions: one that counts the number of yelling lines, and one that gives the percentage.

```
# count the number of yelling lines
isYelling.num = function(msg) {
  body = msg$body
  # consider no body cases
  if(length(body) == 0) return("no body")
  else{
    # remove non-alpha characters
    body = gsub("[^[:alpha:]]", "", body)
    # select none empty lines
    index.not.empty = which(nchar(body) > 0)
    # check if all characters are capital letters
    index.yell = which(nchar(gsub("[A-Z]", "", body[index.not.empty])) < 1)
    # return the count
    return(length(index.yell))
  }
}

# count the percentage of yelling lines
isYelling.per = function(msg) {
  body = msg$body
  if(length(body) == 0) return("no body")
  else{
    # count total number of lines for the message body
    total = length(body)
    # remove non-alpha characters
    body = gsub("[^[:alpha:]]", "", body)
    # select none empty lines
    index.not.empty = which(nchar(body) > 0)
    # check if all characters are capital letters
    index.yell = which(nchar(gsub("[A-Z]", "", body[index.not.empty])) < 1)
    # return the percentage
    return(length(index.yell)/total)
  }
}

# Don't show results because they are in large sizes
#unlist(unname(lapply(emailStruct, isYelling.num)))
#unlist(unname(lapply(emailStruct, isYelling.per)))
```

The two functions to count the number of yelling lines and the percentage of yelling lines are shown above. Since the results are in a large size, they are not presented here.

Question 3

Check that the hour feature in emailDF gives valid values for all of the email messages. Then perform descriptive analysis to compare this feature for spam and ham, and comment on the possibility of using this feature to classify email.

```
# check if all are numbers
sum(lapply(emailDF$hour, is.numeric)==FALSE)
```

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

```
# check if numbers are within range of [0, 23]
sum(unlist(lapply(emailDF$hour, function(x) x < 0 && x > 24)))
```

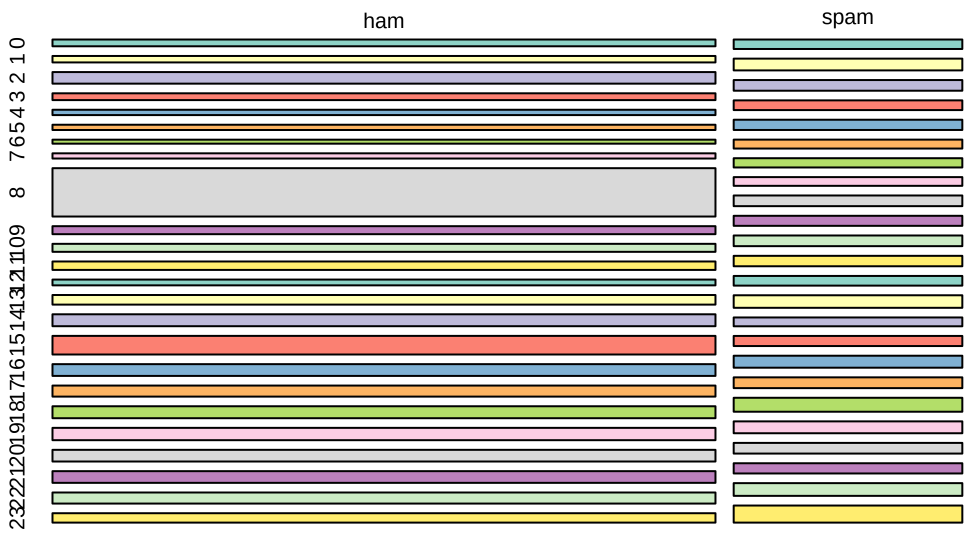
```
## [1] 0
```

```
# check if all are not null
sum(lapply(emailDF$hour, is.null)==TRUE)
```

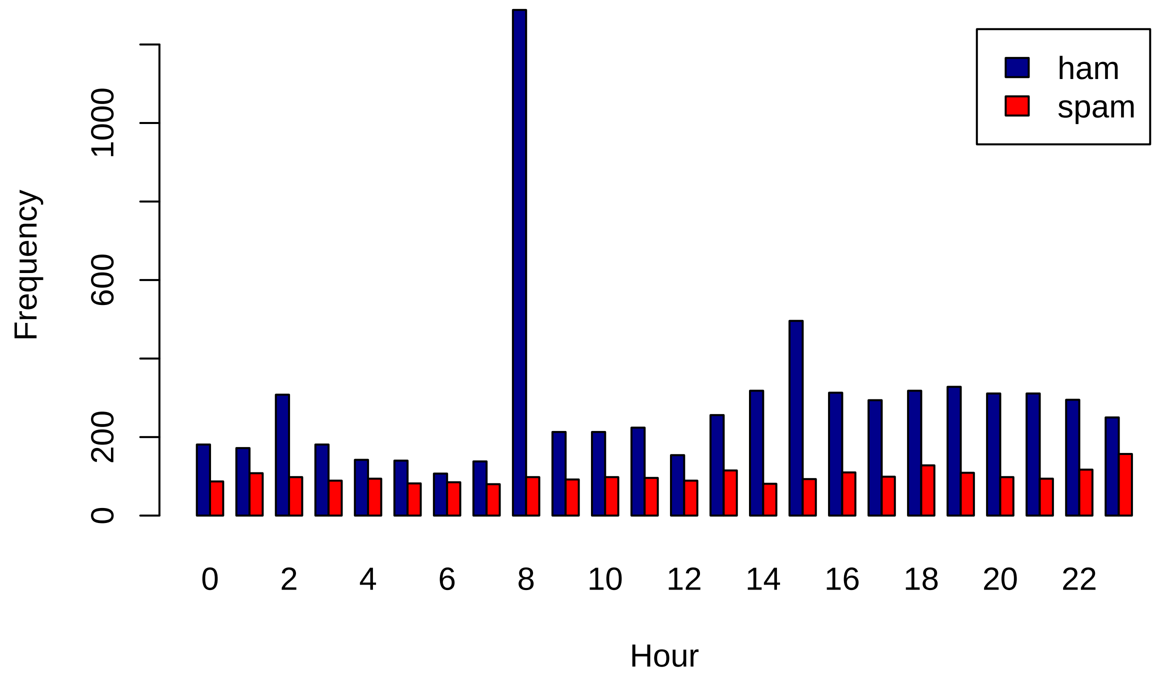
```
## [1] 0
```

```
library(RColorBrewer)
cols = brewer.pal(12, "Set3")
```

```r
# factorize the isSpam attribute
isSpamLabs = factor(emailDF$isSpam, labels = c("ham", "spam"))
# mosaicplot
mosaicplot(table(isSpamLabs, emailDF$hour), main = "",
           xlab = "", ylab = "", color = cols)
```



```r
# barplot
barplot(table(isSpamLabs, emailDF$hour),beside = TRUE, xlab="Hour",
        ylab="Frequency", col=c("darkblue","red"), legend = c("ham", "spam"))
```



For the hour attribute in emailDF, it is checked if all the values are numeric, if all values are within range of [0, 23], and also if there is any null value. Mosaic plot and bar plot are also shown above. Note that for the spam messages, they were sent/received quite evenly throughout the whole day while for ham messeges, there are two significant peaks in a given day: 8 o'clock, 15 o'clock. Such results are quite reasonable: at 8, people usually start working by dealing with emails first; at 15, people try to finish all the emails so they can off the work on time. From the mosaic plot, we can also notice that from midnight to early morning (before 8), ham emails are less likely to be sent/received. Therefore, such observations may be used the classify ham/spam emails: if the email was sent/received at 8 or 15, it is more likely to be a ham message; if it is sent/received from 0 to 7, it is more likely to be a spam message.

Question 4

Do exercise Q.14 from page 167 of Data Science in R: A Case Studies Approach to Computational Reasoning and Problem Solving, by Deborah Nolan and Duncan Temple Lang.

```
# Original code given in the lecture
isRe = function(msg){
  "Subject" %in% names(msg$header) &&
    length(grep("^[ \t]*Re:", msg$header[["Subject"]])) > 0
}

# Alternative 1
isRe1 = function(msg){
  "Subject" %in% names(msg$header) &&
    (length(grep("^[ \t]*Re:", msg$header[["Subject"]])) > 0 ||
       length(grep("^[ \t]*Fwd: Re:", msg$header[["Subject"]])) > 0 )
```

```r
}

# Alternative 2
isRe2 = function(msg){
  "Subject" %in% names(msg$header) &&
       length(grep("Re:", msg$header[["Subject"]])) > 0
}

idx = which(lapply(emailStruct, isRe)==TRUE)
idx1 =  which(lapply(emailStruct, isRe1)==TRUE)
idx2 = which(lapply(emailStruct, isRe2)==TRUE)
# total for original
sum(lapply(emailStruct, isRe) == TRUE)
```

```
## [1] 3005
```

```r
# isSpam fo original
sum(emailDF$isSpam[idx])
```

```
## [1] 71
```

```r
# total for alternative 1
sum(lapply(emailStruct, isRe1) == TRUE)
```

```
## [1] 3005
```

```r
# isSpam for alternative 1
sum(emailDF$isSpam[idx1])
```

```
## [1] 71
```

```r
# total for alternative 2
sum(lapply(emailStruct, isRe2) == TRUE)
```

```
## [1] 3218
```

```r
# isSpam for alternative 2
sum(emailDF$isSpam[idx2])
```

```
## [1] 80
```

Comparing the orignal results (3005 TRUE with 71 in spam) and the alterantive 1 results (3005 TRUE with 71 in spam), they are the same both in ham messages and spam messages. Comparing the orignal results and the alterantive 2 results (3218 TRUE with 80 in spam), the number of both ham messages and spam messages increase. The most useful in predicting spam would be determining if there is "Re:" in the subject line. If there is, then the message are highly unlikely to be a spam. All three ways to detect "Re:" won't make too much difference.