Homework 03

1. Write a function to drop URLs from a message body. The function should take a message body as its input and return the content with URLs removed. Apply the function to the sample text testmsgBody in HW3Setup.R to show that the function works.

Homework code***********************************

```
#A function to drop URLs from the message body
removeurls = function(messagebody){
    output = gsub("\\s?(((f|ht)(tp)s?(://))|www\\.)(.*)[.][a-z]+", "", messagebody)
              return(output)
}


#Testing the function on testmsgBody
removeurls(testmsgBody)
```

```
## [1] "This is a url."
## [2] "Sometimes URLs just start with a world wide web address like."
## [3] "They can also be secure, like."
## [4] "There are also file transfer protocol addresses like."
## [5] "But we wouldn't want to remove www, http, https, ftp, or ftps on their own."
```

2. Write a function to extract the words from the subject value in a message header. The function should take a message header and stop words as its inputs. To implement this function, you can use the findMsgWordsfunction from the text inside your new function after doing some initial processing to get the subject content. Be sure to handle the case where no subject key is present in the message. Apply your function to the 15 sample email headers from sampleEmail to show that it works.

```
#Get the findMsgWords from the Homework3 code
findMsgWords =
  function(msg, stopWords) {
    if(is.null(msg))
      return(character())

    words = unique(unlist(strsplit(cleanText(msg), "[[:blank:]\t]+")))

    # drop empty and 1 letter words
    words = words[ nchar(words) > 1]
    words = words[ !( words %in% stopWords) ]
    invisible(words)
  }

#Obtain the headerlist
headerList = lapply(sampleSplit, function(msg) msg$header)

#########################Function to extract subject words#############################################
# Get and see the subject locations in the headerlist and replace missing 'Subject' with NA.
# Use findMsgWords function to extract the words from the subject content

custom_func = function(headerList){
Subjectlines = sapply(headerList, function(header) {

  Subjectloc = grep("Subject", header)
```

```r
  if (length(Subjectloc) == 0) return(NA)
  header[Subjectloc]

})

Subject_words = gsub("Subject:", "", Subjectlines)
Subject_words = findMsgWords(Subject_words, stopWords)
Subject_words
}

#Test the function
custom_func(headerList)
```

```
##  [1] "new"         "sequences"   "window"      "zzzzteana"   "alexander"
##  [6] "moscow"      "bomber"      "irr"         "klez"        "virus"
## [11] "die"         "nothing"     "like"        "mama"        "used"
## [16] "make"        "ilug"        "sun"         "solaris"     "tiny"
## [21] "dns"         "swap"        "cvs"         "report"      "rambus"
## [26] "man"         "liberalism"  "america"     "activebuddy"
```

3.

```r
#Using the original function to find the log odds
computeFreqs =
  function(wordsList, spam, bow = unique(unlist(wordsList)))
  {
    # create a matrix for spam, ham, and log odds
    wordTable = matrix(0.5, nrow = 4, ncol = length(bow),
                       dimnames = list(c("spam", "ham",
                                         "presentLogOdds",
                                         "absentLogOdds"),  bow))

    # For each spam message, add 1/2 to counts for words in message
    counts.spam = table(unlist(lapply(wordsList[spam], unique)))
    wordTable["spam", names(counts.spam)] = counts.spam + .5

    # Similarly for ham messages
    counts.ham = table(unlist(lapply(wordsList[!spam], unique)))
    wordTable["ham", names(counts.ham)] = counts.ham + .5


    # Find the total number of spam and ham
    numSpam = sum(spam)
    numHam = length(spam) - numSpam

    # Prob(word|spam) and Prob(word | ham)
    wordTable["spam", ] = wordTable["spam", ]/(numSpam + .5)
    wordTable["ham", ] = wordTable["ham", ]/(numHam + .5)

    # log odds
    wordTable["presentLogOdds", ] =
      log(wordTable["spam",]) - log(wordTable["ham", ])
    wordTable["absentLogOdds", ] =
      log((1 - wordTable["spam", ])) - log((1 -wordTable["ham", ]))
    invisible(wordTable)
  }

# Obtain the probabilities and log odds for the training data.
trainTable = computeFreqs(trainMsgWords, trainIsSpam)

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

  sum(freqTable["presentLogOdds", present]) +
    sum(freqTable["absentLogOdds", !present])
}

#Creating the train table
trainTable = computeFreqs(trainMsgWords, trainIsSpam)

#Obtaining the final log liklihood ratio of the test words
testLLR = sapply(testMsgWords, computeMsgLLR, trainTable)

#Obtaining the time taken to run the original function
system.time(sapply(testMsgWords, computeMsgLLR, trainTable))
```

```
##    user  system elapsed
##  64.579   0.251  64.834
```

```
#Computing the log odds using the alternate formula

computeFreqs_alt =
  function(wordsList, spam, bow = unique(unlist(wordsList)))
  {
    # create a matrix for spam, ham, and log odds
    wordTable = matrix(0.5, nrow = 4, ncol = length(bow),
                       dimnames = list(c("spam", "ham",
                                         "presentLogOdds",
                                         "absentLogOdds"),  bow))

    # For each spam message, add 1/2 to counts for words in message
    counts.spam = table(unlist(lapply(wordsList[spam], unique)))
    wordTable["spam", names(counts.spam)] = counts.spam + .5

    # Similarly for ham messages
    counts.ham = table(unlist(lapply(wordsList[!spam], unique)))
    wordTable["ham", names(counts.ham)] = counts.ham + .5


    # Find the total number of spam and ham
    numSpam = sum(spam)
    numHam = length(spam) - numSpam
    # Prob(word|spam) and Prob(word | ham)
    wordTable["spam", ] = wordTable["spam", ]/(numSpam + .5)
    wordTable["ham", ] = wordTable["ham", ]/(numHam + .5)

    # Alternate formula
    wordTable["presentLogOdds", ] =
    wordTable["spam",]/wordTable["ham", ]
    wordTable["absentLogOdds", ] =
    (1 - wordTable["spam", ])/(1 -wordTable["ham", ])

    invisible(wordTable)
  }


computeMsgLLR_alt = 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

  log(prod(freqTable["presentLogOdds", present])) +
    log(prod(freqTable["absentLogOdds", !present]))
}

#Computing the log odds ratio of the test cases using the above functions
trainTable_alt = computeFreqs_alt(trainMsgWords, trainIsSpam)
testLLR_alt = sapply(testMsgWords, computeMsgLLR_alt, trainTable_alt)

#Calculating the time taken to run this function

system.time(sapply(testMsgWords, computeMsgLLR_alt, trainTable_alt))
```

```
##    user  system elapsed
##  66.016   0.000  66.020
```

The time taken for the user defined function is higher.

```
###############################################Comparing the results from both the functions###########
testLLR = round(testLLR,3)
testLLR_alt = round(testLLR_alt,3)

#Let us get the indeces which are a mismatch
compare_output = is.na(match(testLLR, testLLR_alt))

Index_false = which(compare_output == "TRUE", arr.ind = TRUE)
Index_false
```

```
##  [1]   37   60   82  120  139  143  169  207  254  290  304  307  335  363
## [15]  417  429  438  453  457  479  483  502  517  522  606  682 1583 2000
## [29] 2906
```

The above are the indeces which donot match because of the Inf values. Although mathematically on paper the same, the alternate function using the alternate formula gets Inf in a couple of instances.

We can see that both the results are not the same. There are a total of 29 mismatches and those are mentioned above.

4.

```r
#The function given in the homework code

dropAttach = function(body, boundary){

  bString = paste("--", boundary, sep = "")
  bStringLocs = which(bString == body)

  # if there are fewer than 2 beginning boundary strings,
  # there is on attachment to drop
  if (length(bStringLocs) <= 1) return(body)

  # do ending string processing
  eString = paste("--", boundary, "--", sep = "")
  eStringLoc = which(eString == body)

  # if no ending boundary string, grab contents between the first
  # two beginning boundary strings as the message body
  if (length(eStringLoc) == 0)
    return(body[ (bStringLocs[1] + 1) : (bStringLocs[2] - 1)])

  # typical case of well-formed email with attachments
  # grab contents between first two beginning boundary strings and
  # add lines after ending boundary string
  n = length(body)
  if (eStringLoc < n)
    return( body[ c( (bStringLocs[1] + 1) : (bStringLocs[2] - 1),
                     ( (eStringLoc + 1) : n )) ] )

  return( body[ (bStringLocs[1] + 1) : (bStringLocs[2] - 1) ])
}
```

```r
#The alternate function for dropping the attachments when a third parameter is mentioned
#Code with the third parameter
dropAttach_alt = function(body, boundary, drop){

  bString = paste("--", boundary, sep = "")
  bStringLocs = which(bString == body)

  # do ending string processing
  eString = paste("--", boundary, "--", sep = "")
  eStringLoc = which(eString == body)

  # if there are fewer than 2 beginning boundary strings,
  # there is no attachment to drop

  if (length(bStringLocs) <= 1 ){return(body)}


  if(drop == FALSE){
```

```r
  body1 = body[ (bStringLocs[2] + 1)  : (eStringLoc - 1) ]


  Contentlines = sapply(body1, function(body) {
  Contentloc = grep("Content-Type", body)
  if (length(Contentloc) == 0) return(NA)
  body[Contentloc]
  })
  names(Contentlines) = NULL
  a = grep("(plain|html)", Contentlines )
  if (length(a) > 0)
  {
    n = length(body)
    # if no ending boundary string, grab contents between the first
    # two beginning boundary strings as the message body
    if (length(eStringLoc) == 0)
      return(body[c( (bStringLocs[1] + 1) : (bStringLocs[2] - 1), (bStringLocs[2] + 1) : n )])
    #typical case of well-formed email with attachments grab contents between first two beginning bound
    #and add lines after ending boundary string

    if (eStringLoc < n)
    return( body[ c( (bStringLocs[1] + 1) : (bStringLocs[2] - 1), (bStringLocs[2] + 1)  : (eStringLoc -
                     ( (eStringLoc + 1) : n )) ] )

  }
  }



  # if no ending boundary string, grab contents between the first
  # two beginning boundary strings as the message body
  if (length(eStringLoc) == 0)
    return(body[ (bStringLocs[1] + 1) : (bStringLocs[2] - 1)])

  # typical case of well-formed email with attachments
  # grab contents between first two beginning boundary strings and
  # add lines after ending boundary string
  n = length(body)
  if (eStringLoc < n)
    return( body[ c( (bStringLocs[1] + 1) : (bStringLocs[2] - 1),
                     ( (eStringLoc + 1) : n )) ] )

  # fall through case
  # note that the result is the same as the
  # length(eStringLoc) == 0 case, so code could be simplified by
  # dropping that case and modifying the eStringLoc < n check to
  # be 0 < eStringLoc < n
  return( body[ (bStringLocs[1] + 1) : (bStringLocs[2] - 1) ])

  }

#Calculating the different parameters that need to be passed into the function
bodyList = lapply(sampleSplit, function(msg) msg$body)
```

```r
headerList = lapply(sampleSplit, function(msg) msg$header)
getBoundary = function(header) {
  boundaryIdx = grep("boundary=", header)
  boundary = gsub('"', "", header[boundaryIdx])
  gsub(".*boundary= *([^;]*);?.*", "\\1", boundary)
}
boundaryList = lapply(headerList, getBoundary)

#Result of the original function from homework code

result = mapply(dropAttach, bodyList, boundaryList)
lengths_1 = sapply(result, length)
names(lengths_1) = NULL
print(lengths_1)
```

```
##  [1] 50 26 38 32 31 54 35 36 65 58 70 31 38 28 34
```

```r
#Result when the third parameter is TRUE i.e 'drop attachments'
result_alt_true = mapply(dropAttach_alt, bodyList, boundaryList, TRUE)
lengths_2 = sapply(result_alt_true, length)
names(lengths_2) = NULL
print(lengths_2)
```

```
##  [1] 50 26 38 32 31 54 35 36 65 58 70 31 38 28 34
```

```r
#Result when the third parameter is FALSE i.e 'donot drop attachments'
result_alt_false = mapply(dropAttach_alt, bodyList, boundaryList, FALSE)
lengths_3 = sapply(result_alt_false, length)
names(lengths_3) = NULL
print(lengths_3)
```

```
##  [1] 50 26 38 32 31 54 35 79 65 58 70 31 38 28 75
```

As we can see the lengths are the same when the drop= TRUE but the lengths do differ when the drop= FALSE where the content type is plain text or HTML.