STAT480_Individual Project

Bin Feng

Setup

```
# include library
library(biganalytics)
```

```
## Loading required package: bigmemory

## Loading required package: foreach

## Loading required package: biglm

## Loading required package: DBI
```

```
library(ggplot2)
library(rpart)
library(RColorBrewer)
library(rpart.plot)
require("knitr")
```

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

```
# run setup code from another .R file
source("IndividualProjectSetup.R")
# set working directory
opts_knit$set(root.dir = "~/Stat480/RDataScience/AirlineDelays")
```

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

Airline Delay Exercises

1

Create a big.matrix object for the airline data from 1990 to 1995, and add an additional variable to the matrix that indicates if the arrival was delayed. The variable should be 1 for arrival delays greater than 0 and 0 for arrival delays of 0 or less. Be sure to show the contents of any script files you ran at command line and any commands you ran at command line to process the data. These command line steps can be included as text in the report.

```
# flight9095.csv was created in shell first by combining files from 1990 to 1995
# shell code as show below, which is modified based on the code given in lectures
  # cp 1990.csv flight9095.csv
  #         for year in {1991..1995}
  #                 do
  #                 tail -n+2 $year.csv >>flight9095.csv
  # done

# First time use, create a big.matrix and .desc file for future use
```

```r
# This code is modified based on the code given in lectures
# additional variable "ifdelay" is added to indicates if the arrival was delayed
  # x <- read.big.matrix("flight9095.csv", header = TRUE,
  #                       backingfile = "flight9095.bin",
  #                       descriptorfile = "flight9095.desc",
  #                       type = "integer", extraCols = "ifdelay")

# Attach the same big matrix to x using the descriptor file. This allows to use
# an existing big matrix without recreating it.
x <- attach.big.matrix("flight9095.desc")

# fill the "ifdelay" column, postive delay = 1, 0/negative delay = 0, NA is NA.
x[,"ifdelay"] <- as.numeric(x[,"ArrDelay"] > 0)
```

```
## Assignment will down cast from double to integer
## Hint: To remove this warning type:  options(bigmemory.typecast.warning=FALSE)
```

```
## Warning in SetCols.bm(x, j, value):
```

```r
# Count number of flights with known positive delayed arrival
pos.delay <- sum(x[,"ifdelay"] == 1, na.rm = TRUE)
pos.delay
```

```
## [1] 15362495
```

```r
# Count number of flights without known positive delayed arrival
nonpos.delay <- nrow(x) - pos.delay
nonpos.delay
```

```
## [1] 15655464
```

```r
# compute percentage of flights with known positive delayed arrivals
perpos.delay <- pos.delay / nrow(x)
perpos.delay
```

```
## [1] 0.4952774
```

```r
# average known arrival time deviation from expected
mean(x[,"ArrDelay"], na.rm = TRUE)
```

```
## [1] 5.757513
```

Based on the output, the number of flights with known positive delayed arrival is 15362495; the number of flights without known positive delayed arrival is 15655464; the percentage of flights with known positive delayed arrivals is 49.52774% (denominator also include rows with NA); and average known arrival time deviation from expected in the data 5.757513 minutes. Looking at the number of positive delayed arrivals and its percentage, we found that almost half of the total flights were in certain degree of delay. Such large number and percentage surprised me on how on time flights were in the 1990s. However, if we look at the average time deviation, we found that 5.757513 minutes is actually quite acceptable. Therefore, these numbers reveal us that although flights were very likely to be delayed in the 1990s but the overall delay time isn't significant.

2

Obtain quarterly results for arrival delay percentiles (include all numeric values for arrival delays), average known arrival deviations (again, include all numeric values for arrival delays), and percentages of flights with known positive delay. For the percentiles, obtain enough percentiles to make a plot from 5% to 95%. Be efficient and avoid going through the data more often than necessary.

```r
# split months into quarters
quarter <- split(1:nrow(x), floor(x[,"Month"]/3.1))

# computer quarterly results for average known arrivial deviations
quarter.delay.ave <- foreach(quarter.id = quarter, .combine = c) %do% {
  mean(x[quarter.id, "ArrDelay"], na.rm = TRUE)
}
quarter.delay.ave
```

```
## [1] 7.073707 4.953986 5.030579 6.031974
```

```r
# compute quarterly percentages of flightes with known positive delay
quarter.delay.per <- foreach(quarter.id = quarter, .combine = c) %do% {
  sum(x[quarter.id, "ArrDelay"] > 0, na.rm = TRUE) / length(quarter.id)
}
quarter.delay.per
```

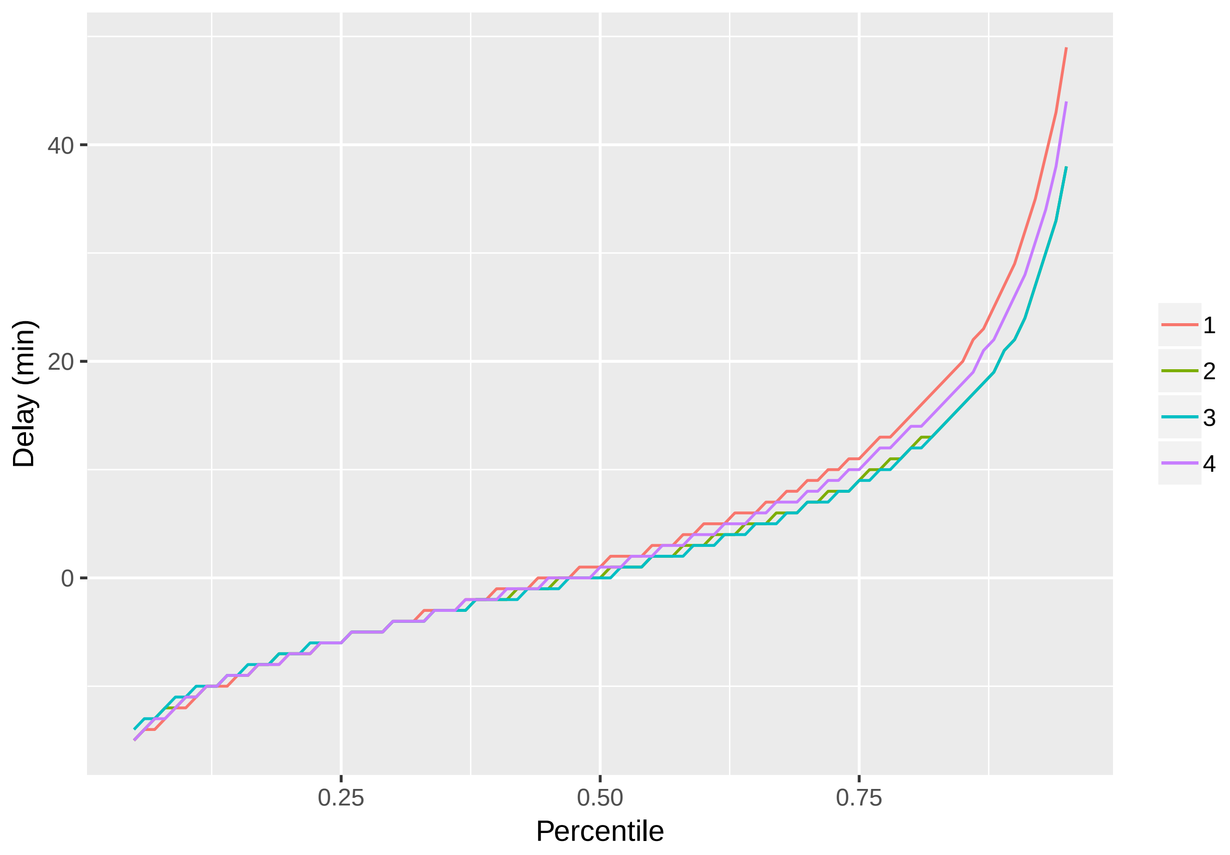
```
## [1] 0.5086707 0.4880573 0.4830486 0.5018343
```

The quarterly results for average known arrivial deviations are 7.073707, 4.953986, 5.030579, and 6.031974 minutes for 1st, 2nd, 3rd, 4th quarter, respectively. The quarterly results for percentages of flights with know positive delay are 50.86707%, 48.80573%, 48.30486%, and 50.18343% for 1st, 2nd, 3rd, 4th quarter, respectively. Note that when computing the percentages, the denominator include also rows with NAs since they should also be counted as flights. Looking at these numbers, we note that the 1st quarter has the largest average delay time and percentage, and followed by the 4th quarter. Basically, during winter seasons and Chrismas seasons, flights are more likely to delay which is reasonable considering the worse weather and higher flight demands. The 2nd and 3rd quaters are the ones with lower arrival delay times and percentages since the weather are normally better and demands are at a lower level.

```r
# compute arrival delay percentiles
myProbs <- seq(0.05, 0.95, 0.01)

# use foreach to find the quantiles for each hour.
delayQuantiles <- foreach(quarter.id = quarter, .combine=cbind) %do% {
  quantile(x[quarter.id, "ArrDelay"], myProbs, na.rm = TRUE)
}
delayQuantiles <- cbind(myProbs, delayQuantiles)
colnames(delayQuantiles) <- c("Percentile", "1","2","3","4")
delayQuantiles
```

```
##     Percentile   1   2   3   4
## 5%        0.05 -15 -15 -14 -15
## 6%        0.06 -14 -14 -13 -14
## 7%        0.07 -14 -13 -13 -13
## 8%        0.08 -13 -12 -12 -13
## 9%        0.09 -12 -12 -11 -12
## 10%       0.10 -12 -11 -11 -11
## 11%       0.11 -11 -11 -10 -11
## 12%       0.12 -10 -10 -10 -10
## 13%       0.13 -10 -10 -10 -10
## 14%       0.14 -10  -9  -9  -9
## 15%       0.15  -9  -9  -9  -9
## 16%       0.16  -9  -9  -8  -9
## 17%       0.17  -8  -8  -8  -8
## 18%       0.18  -8  -8  -8  -8
## 19%       0.19  -8  -7  -7  -8
## 20%          0.20  -7  -7  -7  -7
## 21%          0.21  -7  -7  -7  -7
## 22%          0.22  -7  -7  -6  -7
## 23%          0.23  -6  -6  -6  -6
## 24%          0.24  -6  -6  -6  -6
## 25%          0.25  -6  -6  -6  -6
## 26%          0.26  -5  -5  -5  -5
## 27%          0.27  -5  -5  -5  -5
## 28%          0.28  -5  -5  -5  -5
## 29%          0.29  -5  -5  -5  -5
## 30%          0.30  -4  -4  -4  -4
## 31%          0.31  -4  -4  -4  -4
## 32%          0.32  -4  -4  -4  -4
## 33%          0.33  -3  -4  -4  -4
## 34%          0.34  -3  -3  -3  -3
## 35%          0.35  -3  -3  -3  -3
## 36%          0.36  -3  -3  -3  -3
## 37%          0.37  -2  -3  -3  -2
## 38%          0.38  -2  -2  -2  -2
## 39%          0.39  -2  -2  -2  -2
## 40%          0.40  -1  -2  -2  -2
## 41%          0.41  -1  -2  -2  -1
## 42%          0.42  -1  -1  -2  -1
## 43%          0.43  -1  -1  -1  -1
## 44%          0.44   0  -1  -1  -1
## 45%          0.45   0  -1  -1   0
## 46%          0.46   0   0  -1   0
## 47%          0.47   0   0   0   0
## 48%          0.48   1   0   0   0
## 49%          0.49   1   0   0   0
## 50%          0.50   1   0   0   1
## 51%          0.51   2   1   0   1
## 52%          0.52   2   1   1   1
## 53%          0.53   2   1   1   2
## 54%          0.54   2   1   1   2
## 55%          0.55   3   2   2   2
## 56%          0.56   3   2   2   3
## 57%          0.57   3   2   2   3
## 58%          0.58   4   3   2   3
## 59%          0.59   4   3   3   4
## 60%          0.60   5   3   3   4
## 61%          0.61   5   4   3   4
## 62%          0.62   5   4   4   5
## 63%          0.63   6   4   4   5
## 64%          0.64   6   5   4   5
## 65%          0.65   6   5   5   6
## 66%          0.66   7   5   5   6
## 67%          0.67   7   6   5   7
## 68%          0.68   8   6   6   7
## 69%          0.69   8   6   6   7
## 70%          0.70   9   7   7   8
## 71%          0.71   9   7   7   8
## 72%          0.72  10   8   7   9
## 73%          0.73  10   8   8   9
## 74%        0.74  11   8   8  10
## 75%        0.75  11   9   9  10
## 76%        0.76  12  10   9  11
## 77%        0.77  13  10  10  12
## 78%        0.78  13  11  10  12
## 79%        0.79  14  11  11  13
## 80%        0.80  15  12  12  14
## 81%        0.81  16  13  12  14
## 82%        0.82  17  13  13  15
## 83%        0.83  18  14  14  16
## 84%        0.84  19  15  15  17
## 85%        0.85  20  16  16  18
## 86%        0.86  22  17  17  19
## 87%        0.87  23  18  18  21
## 88%        0.88  25  19  19  22
## 89%        0.89  27  21  21  24
## 90%        0.90  29  22  22  26
## 91%        0.91  32  24  24  28
## 92%        0.92  35  27  27  31
## 93%        0.93  39  30  30  34
## 94%        0.94  43  33  33  38
## 95%        0.95  49  38  38  44
```

```r
# plot the quarterly arrival delay percentiles from 5% to 95%
ggplot(as.data.frame(delayQuantiles), aes(x=Percentile)) +
  geom_line(aes(y = delayQuantiles[,"1"], colour = "1")) +
  geom_line(aes(y = delayQuantiles[,"2"], colour = "2")) +
  geom_line(aes(y = delayQuantiles[,"3"], colour = "3")) +
  geom_line(aes(y = delayQuantiles[,"4"], colour = "4")) +
  xlab("Percentile") + ylab("Delay (min)") + theme(legend.title=element_blank())
```



The computed quarterly results for arrivial delay percentiles are shown and plotted above. In small percentiles, all four quarters are pretty close. Note that the plotted lines for 1st quarter is always on top in high percentiles, followed by the 4th quarter. The 2nd and 3rd quarters are at the bottom. Such observations agrees with the quarterly averages and percentages discussed in the previous chunk of code: the winter and holiday season results in higher delays while mid-year times results in lower delays.
Considering both chunk of codes and results discussed above, we found that for the time from 1990 to 1995, flights are more likely to be delayed during winter and holiday seasons because of the worse weather and higher flight demands; flights are less likely to be delayed during 2nd and 3rd quarters due to a better weather and lower demand level.

Spam Detection Exercises

3

We used probabilities that a word did or did not appear in a message as the basis for Naive Bayes classificationin class. Now let's consider combining the words present and absent in the text with the hour of day the message was sent as the basis for a Naive Bayes classification.

```r
# create functions to create hourtable
computeHrs =
  function(msgDF)
  {
    # create a matrix for spam, ham, and log odds
    wordTable = matrix(0.5, nrow = 4, ncol = 24,
                       dimnames = list(c("spam", "ham",
                                           "presentLogOdds",
                                           "absentLogOdds"),  seq(0,23)))
    hours = seq(0, 23)
    count.spam = vector()
    count.ham = vector()

    for(i in 1:24){
      count.spam[i] = sum(msgDF$isSpam == TRUE & msgDF$hour == i-1)
      wordTable["spam", i] = count.spam[i]
    }

    for(i in 1:24){
      count.ham[i] = sum(msgDF$isSpam == FALSE & msgDF$hour == i-1)
      wordTable["ham", i] = count.ham[i]
    }

    # Prob(hour | spam) and Prob(hour | ham)
    wordTable["spam", ] = wordTable["spam", ]/(numSpam)
    wordTable["ham", ] = wordTable["ham", ]/(numHam)

    # log odds
    wordTable["presentLogOdds", ] =
      log(wordTable["spam",]) - log(wordTable["ham", ])
    # set hour absentLogOdds to be zero, because we don't need to add this prob.
    wordTable["absentLogOdds", ] = 0

    invisible(wordTable)
  }
# calculate the hourtable based on training data
hourTable = computeHrs(emailDF[c(-testHamIdx, -(testSpamIdx+numHam)),])

testIsSpam = rep(c(FALSE, TRUE), c(length(testHamIdx), length(testSpamIdx)))
# calculate hourLLR
testLLR.hrs = sapply(emailDF[c(testHamIdx, testSpamIdx+numHam),"hour"], computeMsgLLR, hourTable)
# tapply(testLLR.hrs, testIsSpam, summary)
# calculate msgLLR
testLLR.msg = sapply(msgWordsList[c(testHamIdx, testSpamIdx+numHam)], computeMsgLLR, trainTable)
# tapply(testLLR.msg, testIsSpam, summary)
# calculate totalLLR
testLLR.all = testLLR.hrs + testLLR.msg
# tapply(testLLR.all, testIsSpam, summary)

# calculate type I&IIerror and determine a good threshold
# choose the same type I error rate threshold (0.01) as discussed in class
xI = typeIErrorRates(testLLR.all, testIsSpam)
xII = typeIIErrorRates(testLLR.all, testIsSpam)
tau01 = round(min(xI$values[xI$error <= 0.01]))
tau01
```

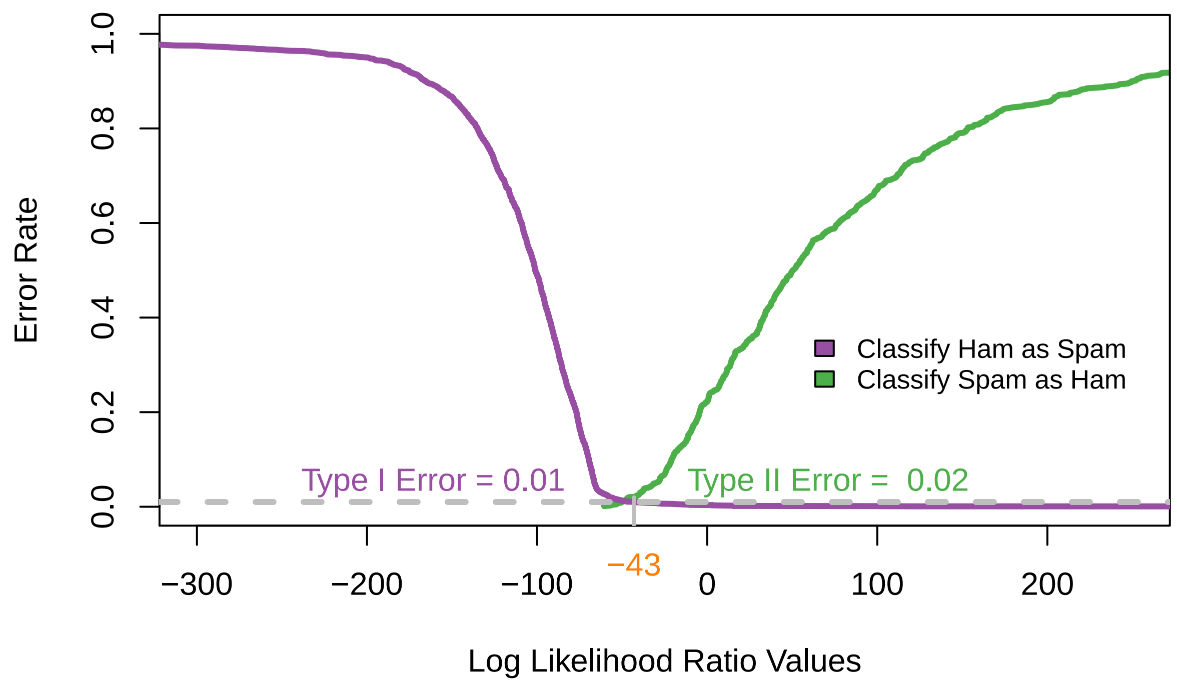
```
## [1] -43
```

```r
t2 = max(xII$error[ xII$values < tau01 ])
t2
```

```
## [1] 0.02002503
```

The above chunk of code done the following: 1) define a function “computeHrs” to compute the hourTable for preparation; 2) compute the log likelihood ratio (LLR) for word that did or did not appear in a message and hour of day the message was sent ,and add them together as the final LLR; 3) determine a threshold tau01 based on the type I error rate = 0.01; 4) calcute the corresponding type II error rate as well. The threshold determined is -43 at the type I error rate of 0.01, and the corresponding type II error rate is 0.02002503. Interestingly, all these numbers are the same as the ones calculated in class (-43, 0.02002503), which is based only on message text content. Such results are because the hour LLR is quite small and flattened out compared with the word LLR. Therefore, the hour infomation barely improve or hurt spam detection.

```r
# plot type I and type II error rate
cols = brewer.pal(9, "Set1")[c(3, 4, 5)]
plot(xII$error ~ xII$values,  type = "l", col = cols[1], lwd = 3,
     xlim = c(-300, 250), ylim = c(0, 1),
     xlab = "Log Likelihood Ratio Values", ylab="Error Rate")
points(xI$error ~ xI$values, type = "l", col = cols[2], lwd = 3)
legend(x = 50, y = 0.4, fill = c(cols[2], cols[1]),
       legend = c("Classify Ham as Spam",
                  "Classify Spam as Ham"), cex = 0.8,
       bty = "n")
abline(h=0.01, col ="grey", lwd = 3, lty = 2)
text(-250, 0.05, pos = 4, "Type I Error = 0.01", col = cols[2])

mtext(tau01, side = 1, line = 0.5, at = tau01, col = cols[3])
segments(x0 = tau01, y0 = -.50, x1 = tau01, y1 = t2,
         lwd = 2, col = "grey")
text(tau01 + 20, 0.05, pos = 4,
     paste("Type II Error = ", round(t2, digits = 2)),
     col = cols[1])
```



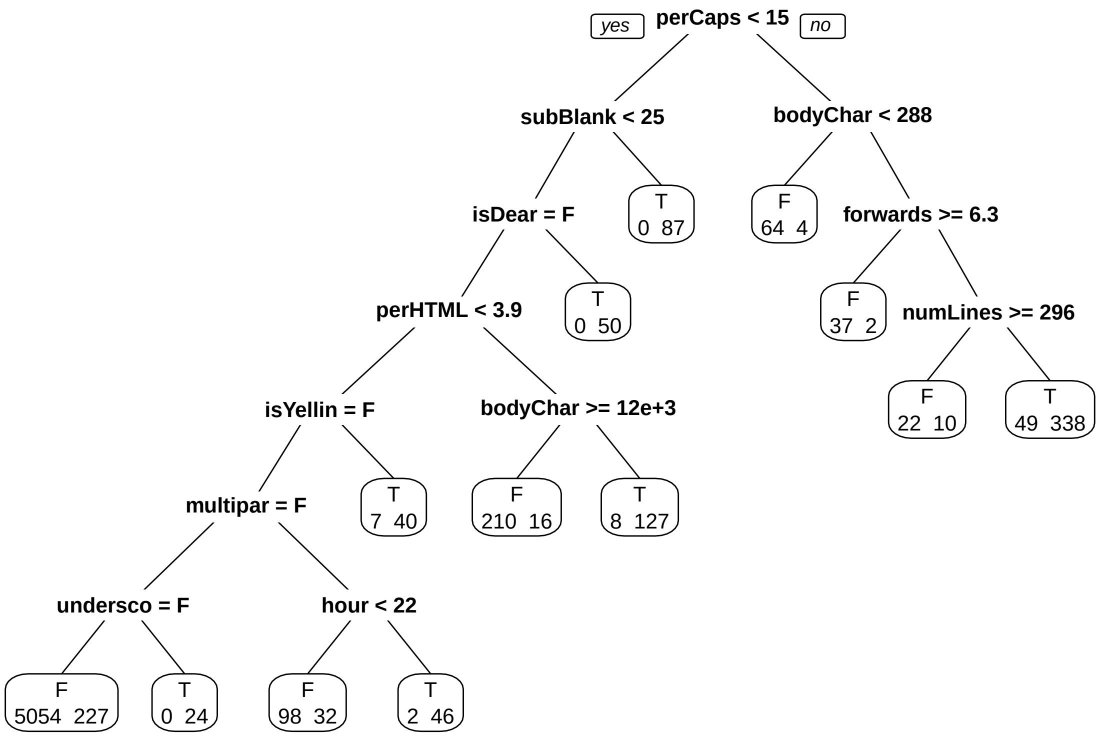
The above plot shows the LLR based on the new defined Naive Bayes classification. Note that the plot is the quite the same as the one given in class. Therefore, the hour infomation barely improve or hurt spam detection.

4

(compare major classification features)

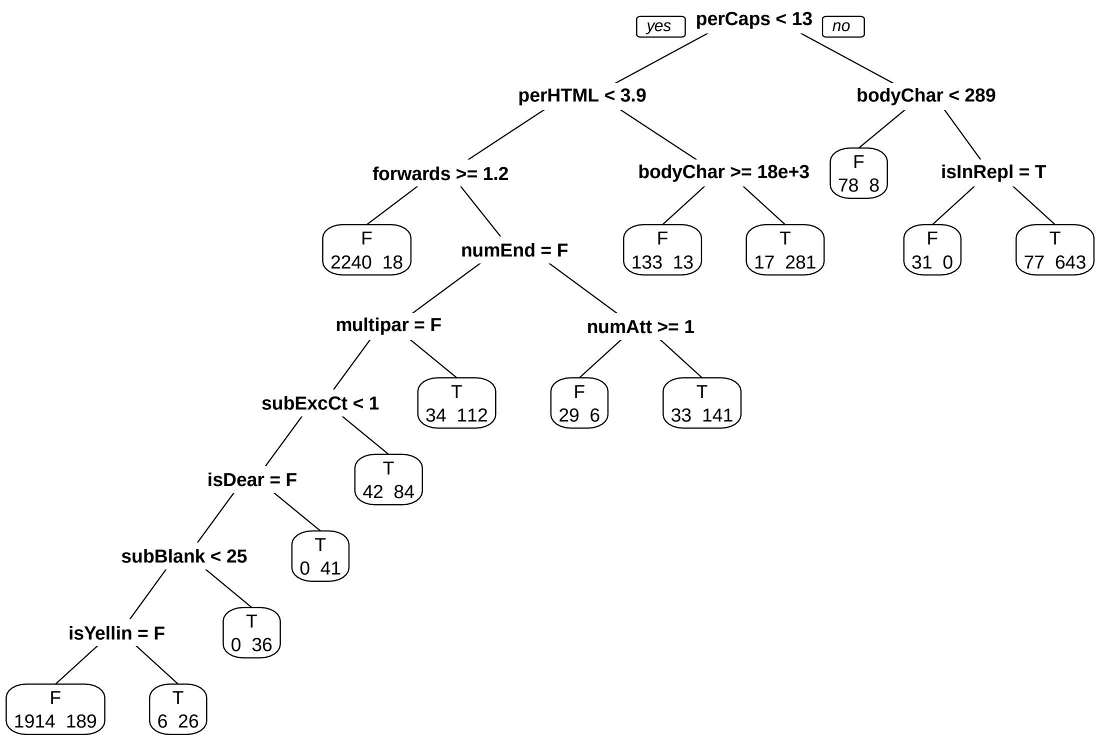
```
emailPath = "~/Stat480/RDataScience/SpamAssassinMessages"
dirNames = list.files(path = paste(emailPath, "messages", sep = .Platform$file.sep))
## new recursive partitioning fitting
# get the number of files in each directory.
count = sapply(paste(emailPath, "messages", dirNames, sep = .Platform$file.sep),
       function(dir) length(list.files(dir)) )
# -1 to remove the addition non-email file
count = unname(count) - 1

# define test ID based on the definition given in the question
testIdx.Ham = seq(count[1]+1, count[1] + count[2])
testIdx.Spam = seq(sum(count[1:4]+1), sum(count[1:5]))
testIdx = c(testIdx.Spam, testIdx.Ham)

# setup the emailDF
emailDFrp = setupRpart(emailDF)
```

```
# get test and train emailDF
testDF.new = emailDFrp[testIdx,]
trainDF.new = emailDFrp[-testIdx,]

# perform recursive partitioning fitting
rpartFit.new = rpart(isSpam ~ ., data = trainDF.new, method = "class")
# plot the results
prp(rpartFit.new, extra = 1)
```



Above is the partition results based on the new defined training data, which are from the easy_ham, hard_ham, and easy_spam directories.

```
## old recursive partitioning fitting
# get test and train emailDF
testDF.old =
  rbind( emailDFrp[emailDFrp$isSpam == "T", ][testSpamIdx, ],
         emailDFrp[emailDFrp$isSpam == "F", ][testHamIdx, ] )
trainDF.old =
  rbind( emailDFrp[emailDFrp$isSpam == "T", ][-testSpamIdx, ],
         emailDFrp[emailDFrp$isSpam == "F", ][-testHamIdx, ])

# perform recursive partitioning fitting
rpartFit.old = rpart(isSpam ~ ., data = trainDF.old, method = "class")
# plot the results
prp(rpartFit.old, extra = 1)
```



Above is the partition results based on the old defined training data, which uses random sampling. Comparing the most important classification features between the two models shown above, similarities include: 1) the first partition features are both "perCaps"; 2) "bodyChar" is used as second partition feature in both models. Differebces between the two model include: 1) the threshold for the major features (the first three) are slightly different for "perCaps" and "bodyChar"; 2) "subBlank" appears as the major feature in the new model while "perHTML" is used in the old model. Such differences occur because there is certain degree of biase when using easy_ham, hard_ham, and easy_spam directories as the training dataset. Some properties exist among these three directories may not be available in the other two test directiory.

(compare test error rates)

```
## new model
# Get predictions for all data based on the model.
predictions.new = predict(rpartFit.new, newdata = testDF.new[, names(testDF.new) != "isSpam"],
                         type = "class")

# See predictions for known ham.
predsForHam.new = predictions.new[ testDF.new$isSpam == "F" ]
# summary(predsForHam.new)

# Obtain the Type I error rate.
sum(predsForHam.new == "T") / length(predsForHam.new)
```

```
## [1] 0.007857143
```

```r
# Obtain the Type II error rate.
predsForSpam.new = predictions.new[ testDF.new$isSpam == "T" ]
sum(predsForSpam.new == "F") / length(predsForSpam.new)
```

```
## [1] 0.3421808
```

```r
## old model
# Get predictions for all data based on the model.
predictions.old = predict(rpartFit.old, newdata = testDF.old[, names(testDF.new) != "isSpam"],
                          type = "class")

# See predictions for known ham.
predsForHam.old = predictions.old[ testDF.old$isSpam == "F" ]
# summary(predsForHam.old)

# Obtain the Type I error rate.
sum(predsForHam.old == "T") / length(predsForHam.old)
```

```
## [1] 0.05394907
```

```r
# Obtain the Type II error rate.
predsForSpam.old = predictions.old[ testDF.old$isSpam == "T" ]
sum(predsForSpam.old == "F") / length(predsForSpam.old)
```

```
## [1] 0.1564456
```

Above chunk of code compare error rates for the model using the new training and test data to the rates found using randomly sampled data. Note that for type I error, the new model has a better performance than the old model (0.007857143 < 0.05394907). But for type II error, the new model has a worse performance than the old model (0.3421808 > 0.1564456). Such differences occur because there is certain degree of biase when using easy_ham, hard_ham, and easy_spam directories as the training dataset. Some properties exist among these three directories may not be available in the other two test directiory.

(compare error rates for easy_ham_2)

```r
testDF.ham = emailDFrp[testIdx.Ham,]
## new model
predictions.new = predict(rpartFit.new, newdata = testDF.ham[, names(testDF.ham) != "isSpam"],
                          type = "class")
predsForSpam.new = predictions.new[ testDF.ham$isSpam == "F" ]
# summary(predsForSpam.new)

# Obtain the Type I error rate.
sum(predsForSpam.new == "T") / length(predsForSpam.new)
```

```
## [1] 0.007857143
```

```r
# Obtain the Type II error rate.
predsForSpam.new = predictions.new[ testDF.ham$isSpam == "T" ]
sum(predsForSpam.new == "F") / length(predsForSpam.new)
```

```
## [1] NaN
```

```r
## old model
predictions.old = predict(rpartFit.old, newdata = testDF.ham[, names(testDF.ham) != "isSpam"],
                          type = "class")
predsForSpam.old = predictions.old[ testDF.ham$isSpam == "F" ]
# summary(predsForSpam.old)

# Obtain the Type I error rate.
sum(predsForSpam.old == "T") / length(predsForSpam.old)
```

```
## [1] 0.035
```

```r
# Obtain the Type II error rate.
predsForSpam.old = predictions.old[ testDF.ham$isSpam == "T" ]
sum(predsForSpam.old == "F") / length(predsForSpam.old)
```

```
## [1] NaN
```

Note that type II error is not available here since all the test data are ham. Also, note that the type I error for new model is the same while type I error for old model has improved.

(compare error rates for easy_spam_2)

```r
testDF.spam = emailDFrp[testIdx.Spam,]
## new model
predictions.new = predict(rpartFit.new, newdata = testDF.spam[, names(testDF.spam) != "isSpam"],
                          type = "class")
predsForSpam.new = predictions.new[ testDF.spam$isSpam == "F" ]
# summary(predsForSpam.new)

# Obtain the Type I error rate.
sum(predsForSpam.new == "T") / length(predsForSpam.new)
```

```
## [1] NaN
```

```r
# Obtain the Type II error rate.
predsForSpam.new = predictions.new[ testDF.spam$isSpam == "T" ]
sum(predsForSpam.new == "F") / length(predsForSpam.new)
```

```
## [1] 0.3421808
```

```r
## old model
predictions.old = predict(rpartFit.old, newdata = testDF.spam[, names(testDF.spam) != "isSpam"],
                          type = "class")
predsForSpam.old = predictions.old[ testDF.spam$isSpam == "F" ]
# summary(predsForSpam.old)

# Obtain the Type I error rate.
sum(predsForSpam.old == "T") / length(predsForSpam.old)
```

```
## [1] NaN
```

```r
# Obtain the Type II error rate.
predsForSpam.old = predictions.old[ testDF.spam$isSpam == "T" ]
sum(predsForSpam.old == "F") / length(predsForSpam.old)
```

```
## [1] 0.1362984
```

Note that type I error is not available here since all the test data are spam. Also, note that the type II error for new model is the same while type II error for old model has improved.
Based on the previous three chunks of code, we notice that for the new model, type I&II errors are the same regardless if the test data is seperated or not. But for the old model ,type I$II errors have improved when testing on easy_ham_2 and easy_spam_2. This is because the training dataset used by old model has already contain some data from easy_ham_2 and easy_spam_2 since the index is randomly assigned among all data. Therefore, when using the test dataset, which has some same training data in it, the old model's performance will definitely improved compared with a totally new testing dataset.