STAT480 Individual Project

Chenz Zhang, NetID chenziz2

2/26/2019

Contents

Airline Delay Exercises

Exercise 1

1. Create `AirlineData1990_1995.csv` Using Unix

```
# Change to the data directory if we are not already in that director.
cd ~/Stat480/RDataScience/AirlineDelays

# Open a new file to enter shell code for combining csv file content.
vi combinescript_new

# Insert the following modified code from page 222 of Data Science in R in the file
# and then write and quit out of the file. We will just work with the 1980s data.
# To insert, type i . Then you can type or paste into the editor.
# To save the changes and exit the editor, first type the <Esc> key, then type :wq and hit <Enter>
cp 1990.csv AirlineData199095.csv
    for year in {1991..1995}
        do
        tail -n+2 $year.csv >>AirlineData199095.csv
done

# At command line, make the file executable and then run the script.
chmod u+x combinescript_new
./combinescript_new

# See entire storage currently used on .
df -h

rm 198*.csv
rm 199*.csv
rm 20**.csv
```

2. Do attach.big.matrix

```r
library(biganalytics)
```

```
## Loading required package: bigmemory

## Loading required package: foreach

## Loading required package: biglm

## Loading required package: DBI
```

```r
setwd("~/Stat480/RDataScience/AirlineDelays")
#create a big matrix
x <- read.big.matrix("AirlineData199095.csv", header = TRUE,
                     backingfile = "air9095.bin",
                     descriptorfile = "air9095.desc",
                     type = "integer", extraCols = "DelayIndex")
x <- attach.big.matrix("air9095.desc")
```

3. Wrtie 'DelayIndex', 1 for delay larger than 0 and 0 for delay not larger than 0

```r
x[which(x[,"DepDelay"] > 0), "DelayIndex"] <- 1
```

```
## Warning in SetElements.bm(x, i, ms, value): Assignment will down cast from double to integer
## Hint: To remove this warning type:  options(bigmemory.typecast.warning=FALSE)
```

```r
x[which(x[,"DepDelay"] <= 0), "DelayIndex"] <- 0
```

```
## Warning in SetElements.bm(x, i, ms, value): Assignment will down cast from double to integer
## Hint: To remove this warning type:  options(bigmemory.typecast.warning=FALSE)
```

4. Statistics for ArrDelay

```r
#1. Compute the number of flights with and without known positive delayed arrival
length(which(x[,"ArrDelay"] > 0))
```

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

```r
#2. Percentage of flights with known positive delayed arrivals
length(which(x[,"ArrDelay"] > 0))/(nrow(x) - sum(is.na(x[,"ArrDelay"])))
```

```
## [1] 0.5024065
```

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

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

From the result, we can conclude that 15362495 flights had delayed arrival and over 50% known flights had delayed arrivals. The average arrvial time deviation from expected is 5.757513 which means 5.757513 mins later than expected time.

Exercise 2

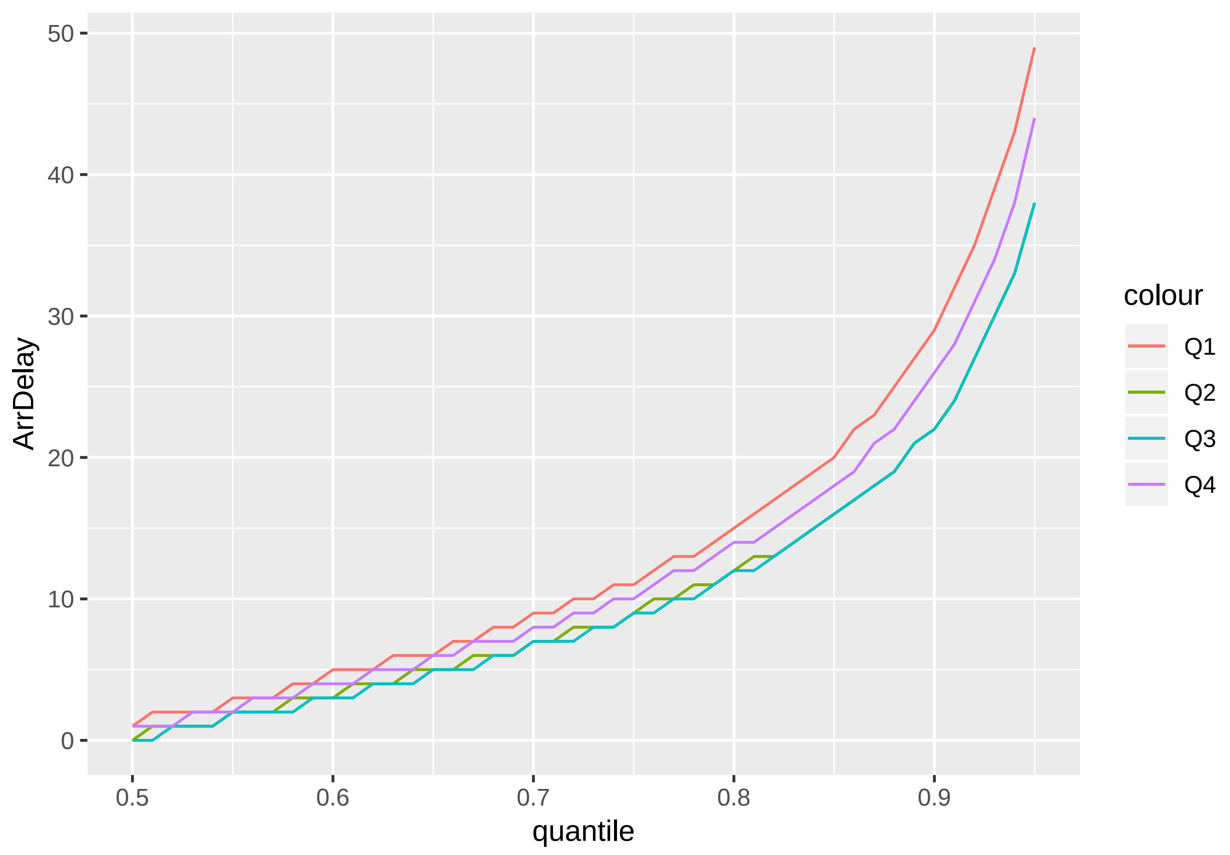
```r
library(foreach)
flight.quarter <- floor((x[,"Month"]-1)/3) + 1
quarterInds <- split(1:length(flight.quarter), flight.quarter)
quarterName <- c("Q1","Q2","Q3","Q4")

myProbs <- seq(0.5,0.95,0.01)
arrdelayBig <- foreach( quarter = quarterInds, .combine=cbind) %do% {
  c(quantile(x[quarter, "ArrDelay"], myProbs, na.rm = TRUE),
        mean(x[quarter, "ArrDelay"], na.rm = TRUE),
        (sum(x[quarter, "DelayIndex"], na.rm = TRUE)/length(which(is.na(x[quarter, "DelayIndex"]) == FA
}

colnames(arrdelayBig) <- quarterName
table1 <- tail(arrdelayBig, n = 2)
row.names(table1) <- c("AvgDeviation","DelayPercent")
print(table1)
```

```
##                    Q1        Q2        Q3        Q4
## AvgDeviation 7.0737071 4.9539858 5.0305786 6.0319742
## DelayPercent 0.4749769 0.4362343 0.4392693 0.4704521
```

```r
library(ggplot2)
arrpercent <- arrdelayBig[1:(nrow(arrdelayBig)-2),]
df <- data.frame(Q1 = arrpercent[,1], Q2 = arrpercent[,2],
                 Q3 = arrpercent[,3], Q4 = arrpercent[,4],
                 quantile = myProbs)
arrplot <- ggplot(df, aes(x = quantile)) +
           geom_line(aes(y = Q1, colour = "Q1")) +
           geom_line(aes(y = Q2, colour = "Q2")) +
           geom_line(aes(y = Q3, colour = "Q3")) +
           geom_line(aes(y = Q4, colour = "Q4")) +
           ylab("ArrDelay")
arrplot
```



- Average Know Deviation: The highest delay quarter was from January to March (Quarter1), the lowest delay quarter was from April to June (Quarter2). From Q2 to next year Q1, delay time was increasing. This might becuase comfortable weather in Quarter2 made flight condition better and cold or extreme weather in Winter made flight condition worse.
- Delay Percentage: The percentage was close to 50% but not that high. Q2 and Q3 had relative lower percentage; Q1 adn Q4 had relative higher percentage. This indicates the same condition as Average Know Deviation. Also, the increasing trend is the same.
- Percentiles: From the plot graph, we know that Q1 and Q4 has relatively higher delay time at the same quantile. Especially, when quantile bigger than 0.9, Q1 and Q4 turned to have extremely larger delay time. Besides, at 0.5 point, all quarters has delay time close to 0, but some of them has value bigger than 0.

Spam Detection Exercises

Exercise 3

```r
load("~/Stat480/RDataScience/Homework/msgWordsList.rda")
load("~/Stat480/RDataScience/Homework/trainTable.rda")
load("~/Stat480/RDataScience/Chapter3/spamAssassinDerivedDF.rda")
```

1. **Build trainHourTable**

```r
set.seed(418910)

# Take approximately 1/3 of the spam and ham messages as our test spam and ham messages.
isSpam = emailDF[,"isSpam"]
msgHourList = emailDF[,"hour"]

numEmail = length(isSpam)
numSpam = sum(isSpam)
numHam = numEmail - numSpam


testSpamIdx = sample(numSpam, size = floor(numSpam/3))
testHamIdx = sample(numHam, size = floor(numHam/3))

testMsgHours = c((msgHourList[isSpam])[testSpamIdx],
                 (msgHourList[!isSpam])[testHamIdx] )
trainMsgHours = c((msgHourList[isSpam])[ - testSpamIdx],
                  (msgHourList[!isSpam])[ - testHamIdx])

testIsSpam = rep(c(TRUE, FALSE),
                 c(length(testSpamIdx), length(testHamIdx)))
trainIsSpam = rep(c(TRUE, FALSE),
                  c(numSpam - length(testSpamIdx),
                    numHam - length(testHamIdx)))


computeFreqs = function(hourList, spam, bow = unique(hourList)){
    # create a matrix for spam, ham, and log odds
    hourTable = matrix(0.5, nrow = 3, ncol = length(bow),
                       dimnames = list(c("spam", "ham",
                                         "presentLogOdds"),  bow))

    # For each spam message, add 1/2 to counts for hours in message
    counts.spam = table(hourList[spam])
    hourTable["spam", names(counts.spam)] = counts.spam + .5

    # Similarly for ham messages
    counts.ham = table(hourList[!spam])
    hourTable["ham", names(counts.ham)] = counts.ham + .5


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

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

    # log odds
    hourTable["presentLogOdds", ] =
      log(hourTable["spam",]) - log(hourTable["ham", ])

    invisible(hourTable)
  }

# Obtain the probabilities and log odds for the training data.
trainHourTable = computeFreqs(trainMsgHours, trainIsSpam)
trainHourTable
```

```
##                         12         13          15         17         18
## spam           0.03534564 0.04910854  0.04222709 0.04535502 0.05411323
## ham            0.02297983 0.03722084  0.07152875 0.04045744 0.04175208
## presentLogOdds 0.43055828 0.27716402 -0.52703758 0.11427035 0.25932953
##                         21           8          11          4          7
## spam            0.04097591  0.03847357 0.03972474 0.03784798 0.03784798
## ham             0.04455713  0.18459381 0.03096343 0.02319560 0.02060632
## presentLogOdds -0.08378764 -1.56818629 0.24916750 0.48961515 0.60797975
##                         19         23          16         10          22
## spam            0.04222709 0.07037848  0.04097591 0.04910854  0.04285267
## ham             0.04563599 0.03614198  0.04649908 0.03117920  0.04347826
## presentLogOdds -0.07763477 0.66643250 -0.12644814 0.45428175 -0.01449301
##                         20          5          1          6          3
## spam            0.03909916 0.03909916 0.04347826 0.03472005 0.03784798
## ham             0.04412558 0.02190096 0.02449024 0.01629086 0.02686374
## presentLogOdds -0.12093879 0.57957039 0.57398654 0.75671299 0.34280028
##                           2          14          0          9
## spam           0.045355020  0.03096653 0.03534564 0.03472005
## ham            0.044988672  0.04671486 0.02556910 0.03074765
## presentLogOdds 0.008110149 -0.41115525 0.32379031 0.12150365
```

2. Calculate log likelihood ratio

```
computeWordsLLR = 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])
}

computeHoursLLR = function(hours, freqTableHours){
  # Discards hours not in training data.
  hours = hours[!is.na(match(hours, colnames(freqTableHours)))]

  # Find which hours are present
  presentHours = colnames(freqTableHours) %in% hours

  sum(freqTableHours["presentLogOdds", presentHours])
}
```

```r
testMsgWords = c((msgWordsList[isSpam])[testSpamIdx],
                 (msgWordsList[!isSpam])[testHamIdx] )
trainMsgWords = c((msgWordsList[isSpam])[ - testSpamIdx],
                  (msgWordsList[!isSpam])[ - testHamIdx])


testWordsLLR = sapply(testMsgWords, computeWordsLLR, trainTable)
testHoursLLR = sapply(testMsgHours, computeHoursLLR, trainHourTable)
testLLR = testWordsLLR + testHoursLLR
```

3. Find the optimal tau value

```r
typeIErrorRates =
  function(llrVals, isSpam)
  {
    # order the llr values and spam indicators
    o = order(llrVals)
    llrVals =  llrVals[o]
    isSpam = isSpam[o]

    # get indices for ham
    idx = which(!isSpam)
    N = length(idx)
    # get the error rates and llr values for the ham indices
    list(error = (N:1)/N, values = llrVals[idx])
  }

typeIIErrorRates = function(llrVals, isSpam) {

  o = order(llrVals)
  llrVals =  llrVals[o]
  isSpam = isSpam[o]


  idx = which(isSpam)
  N = length(idx)
  list(error = (1:(N))/N, values = llrVals[idx])
}

xI = typeIErrorRates(testLLR, testIsSpam)
xII = typeIIErrorRates(testLLR, testIsSpam)
tau01 = round(min(xI$values[xI$error <= 0.01]))
t2 = max(xII$error[ xII$values < tau01 ])
print(tau01)
```

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

```r
typeIErrorRate =
  function(tau, llrVals, spam)
  {
    classify = llrVals > tau
    sum(classify & !spam)/sum(!spam)
  }
```

```
typeIErrorRate(tau01, testLLR,testIsSpam)
```

```
## [1] 0.01035822
```

The old tau value was:

```
xIwords = typeIErrorRates(testWordsLLR, testIsSpam)
xIIwords = typeIIErrorRates(testWordsLLR, testIsSpam)
tau02 = round(min(xIwords$values[xIwords$error <= 0.01]))
t2words = max(xIIwords$error[ xIIwords$values < tau02 ])
print(tau02)
```

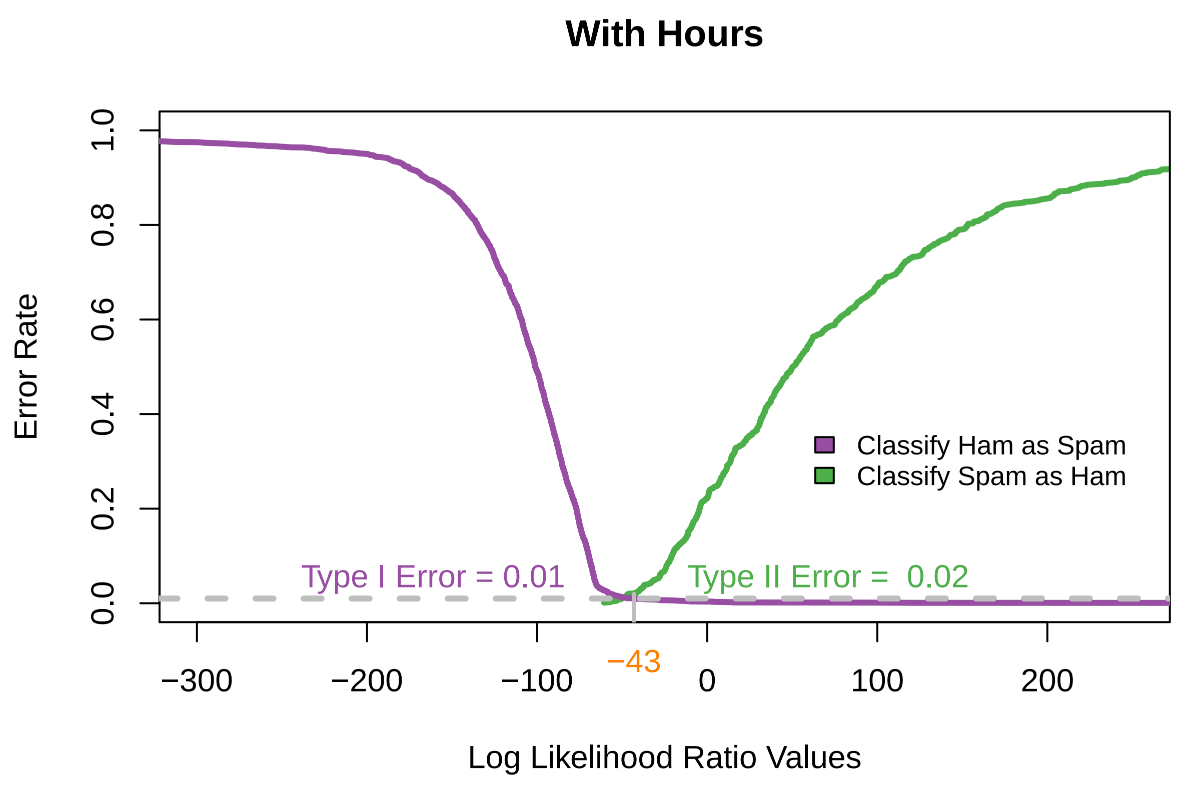
```
## [1] -43
```

Therefore, the former tau value was -43, and now the new tau value is also -43.

4. Compare TypeI error and TyeII error with and without hour condition

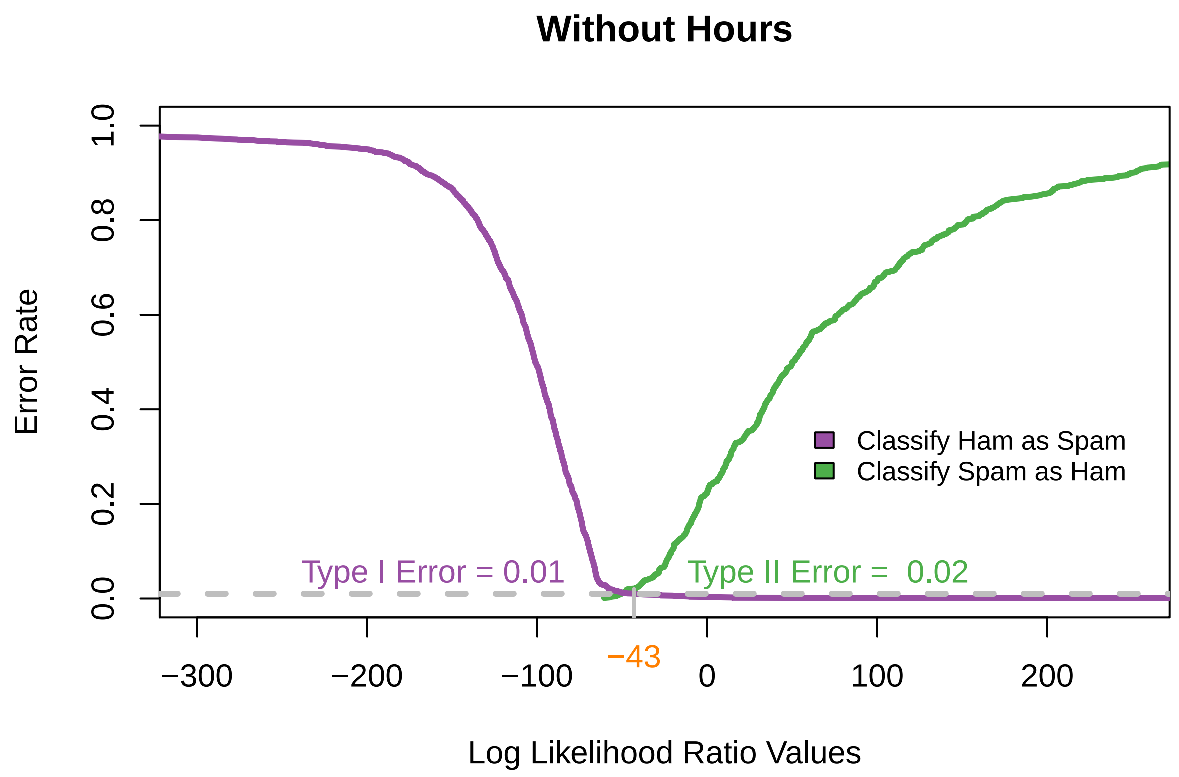
```
library(RColorBrewer)
cols = brewer.pal(9, "Set1")[c(3, 4, 5)]
plot(xII$error ~ xII$values,  type = "l", col = cols[1], lwd = 3,
     xlim = c(-300, 250), ylim = c(0, 1),
     xlab = "Log Likelihood Ratio Values", ylab="Error Rate",
     main = "With Hours")
points(xI$error ~ xI$values, type = "l", 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])
```



```r
plot(xIIwords$error ~ xIIwords$values,  type = "l", col = cols[1], lwd = 3,
     xlim = c(-300, 250), ylim = c(0, 1),
     xlab = "Log Likelihood Ratio Values", ylab="Error Rate",
     main = "Without Hours")
points(xIwords$error ~ xIwords$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(tau02, side = 1, line = 0.5, at = tau02, col = cols[3])
segments(x0 = tau02, y0 = -.50, x1 = tau02, y1 = t2,
         lwd = 2, col = "grey")
text(tau02 + 20, 0.05, pos = 4,
     paste("Type II Error = ", round(t2, digits = 2)),
     col = cols[1])
```



There is not obvious difference between former and present type I & II errors.

```
list(WithHours.xI.xII = cbind(sapply(xI, summary),sapply(xII,summary)),
     WithoutHours.xI.xII = cbind(sapply(xIwords, summary),sapply(xIIwords,summary)))
```

```
## $WithHours.xI.xII
##                error       values       error        values
## Min.    0.0004315926 -1360.81115 0.001251564    -60.581762
## 1st Qu. 0.2503236944  -127.19083 0.250938673      6.473308
## Median  0.5002157963  -101.14700 0.500625782     50.614767
## Mean    0.5002157963  -116.36909 0.500625782    137.443069
## 3rd Qu. 0.7501078981   -81.71613 0.750312891    130.172436
## Max.    1.0000000000   700.93311 1.000000000  23567.777647
##
## $WithoutHours.xI.xII
##                error       values       error        values
## Min.    0.0004315926 -1360.81926 0.001251564    -60.567269
## 1st Qu. 0.2503236944  -126.98394 0.250938673      6.076149
## Median  0.5002157963  -101.18199 0.500625782     50.173759
## Mean    0.5002157963  -116.16031 0.500625782    137.324695
## 3rd Qu. 0.7501078981   -81.22987 0.750312891    130.638022
## Max.    1.0000000000   700.26667 1.000000000  23567.528479
```

From this table, we can see the errors of this two conditions are the same but values are slightly different. Type I&II values under the condition with hours is slightly more than the condition withour hours.

So far, we can make a conclusion that the hour information neither improve the detection nor hurt the detection. The results are almost the same. So, hour may not be a good classification feature.

Exercise 4

One thing to clarify. The `easy_spam` and `easy_spam_2` in my directory are named `spam` and `spam_2`.

1. Do recursive partitioning fitting

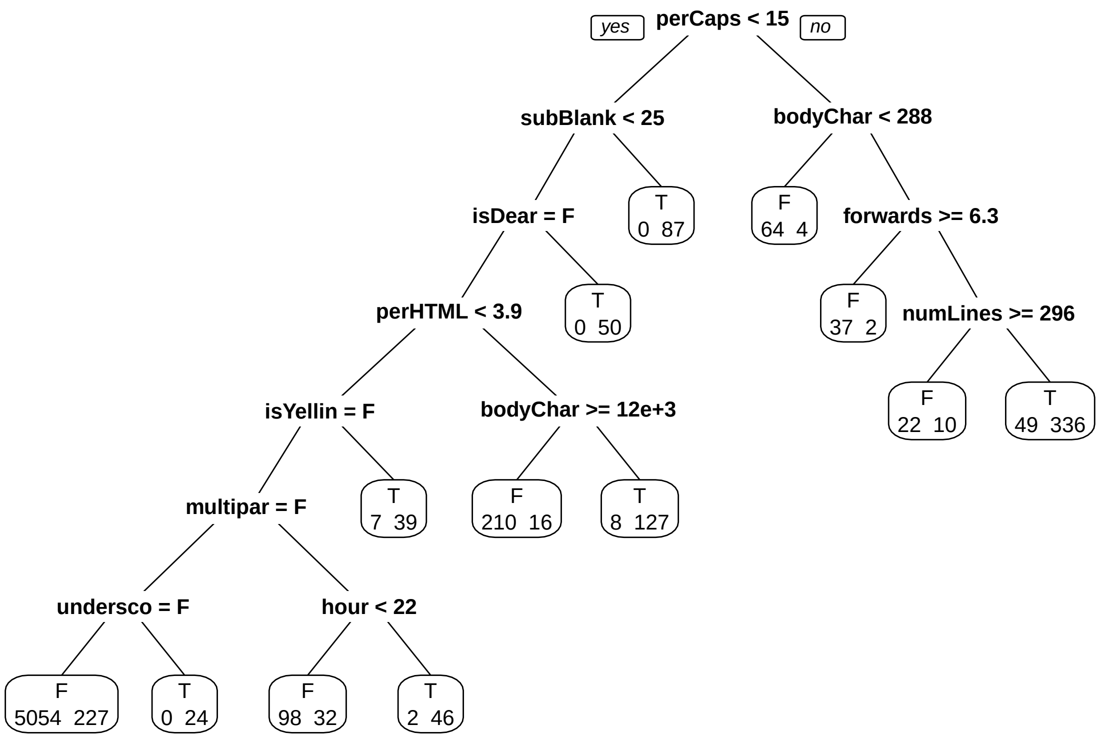
```r
library(rpart)
# Function to replace logical variables with factor variables
setupRpart = function(data) {
  logicalVars = which(sapply(data, is.logical))
  facVars = lapply(data[ , logicalVars],
                   function(x) {
                     x = as.factor(x)
                     levels(x) = c("F", "T")
                     x
                   })
  cbind(facVars, data[ , - logicalVars])
}

# Process the email data frame.
emailDFrp = setupRpart(emailDF)
trainIdx = c(grep("easy_ham[^_]",rownames(emailDFrp)),
                 grep("hard_ham",rownames(emailDFrp)),
                 grep("spam[^_]",rownames(emailDFrp)))

trainDF = emailDFrp[trainIdx, ]
testDF = emailDFrp[-trainIdx, ]
#trainSpam = emailDFrp[trainIdx, "isSpam"]

rpartFit = rpart(isSpam ~ ., data = trainDF, method = "class")

library(rpart.plot)
prp(rpartFit, extra = 1)
```

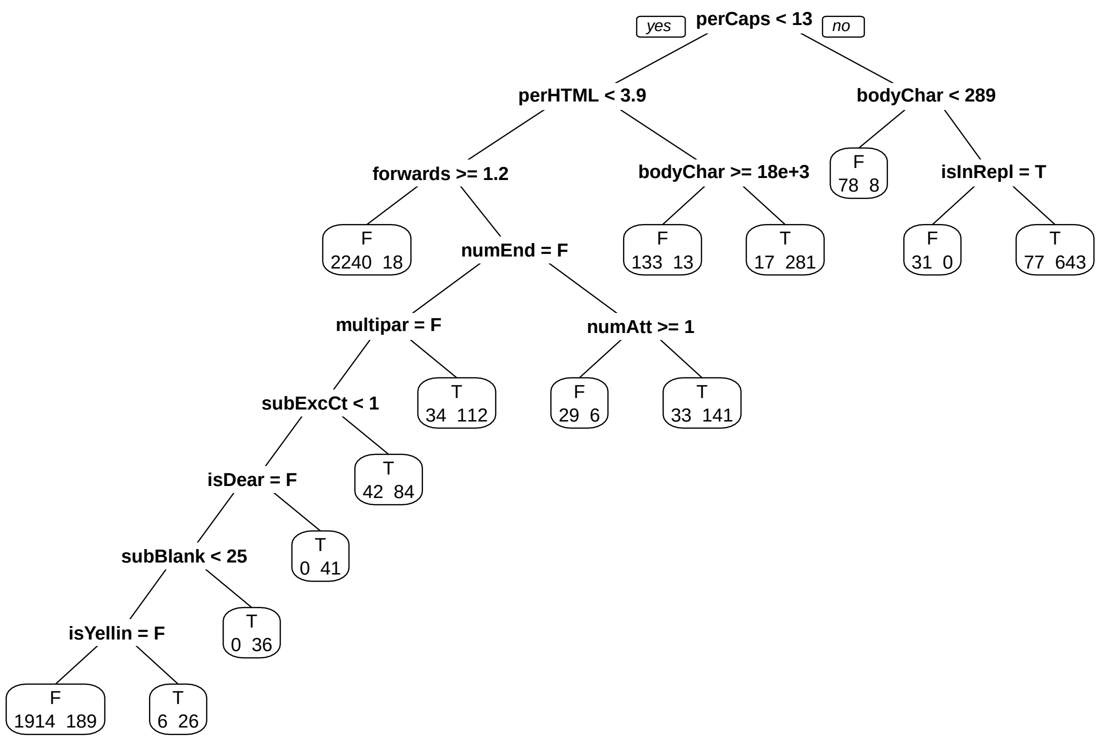


2. Method in the text

```r
set.seed(418910)
testSpamIdx2 = sample(numSpam, size = floor(numSpam/3))
testHamIdx2 = sample(numHam, size = floor(numHam/3))

testDF2 =
  rbind( emailDFrp[ emailDFrp$isSpam == "T", ][testSpamIdx2, ],
         emailDFrp[emailDFrp$isSpam == "F", ][testHamIdx2, ] )
trainDF2 =
  rbind( emailDFrp[emailDFrp$isSpam == "T", ][-testSpamIdx2, ],
         emailDFrp[emailDFrp$isSpam == "F", ][-testHamIdx2, ])

# Fit the recursive partitioning model for spam as a function of all variables in the data frame.
rpartFit2 = rpart(isSpam ~ ., data = trainDF2, method = "class")
prp(rpartFit2, extra = 1)
```



3. Compare classification features

On the one hand, the most important classification features `perCap` are the same, which is the percentage of captial characters. The classification standards are different. In my method, it is `perCap < 15`. In the text method, it is `perCap < 13`.

On the other hand, the three most important classification features are different. In my method, they are `perCap < 15`, `subBank < 25` and `bodyChar < 288`. In the text method, they are `perCap < 13`, `perHTML <3.9` and `bodyChar < 289`.

4. Compare error rates

1) New Method:

```
#----------test.data--easy_ham_2&spam_2----------------------------
predictions = predict(rpartFit,
                      newdata = testDF[, names(testDF) != "isSpam"],
                      type = "class")
# See predictions for known ham.
predsForHam = predictions[ testDF$isSpam == "F" ]
predsForSpam = predictions[ testDF$isSpam == "T" ]

errorRate1 = c(sum(predsForHam == "T") / length(predsForHam),
              sum(predsForSpam == "F") / length(predsForSpam))
#----------test.data--random-sample---------------------------------
```

```r
predictions3 = predict(rpartFit,
                       newdata = testDF2[, names(testDF2) != "isSpam"],
                       type = "class")
predsForHam3 = predictions3[ testDF2$isSpam == "F" ]
predsForSpam3 = predictions3[ testDF2$isSpam == "T" ]
errorRate3 = c(sum(predsForHam3 == "T") / length(predsForHam3),
              sum(predsForSpam3 == "F") / length(predsForSpam3))
```

2) Text Method:

```r
#----------test.data--random-sample----------------------------------
predictions2 = predict(rpartFit2,
                       newdata = testDF2[, names(testDF2) != "isSpam"],
                       type = "class")
# See predictions for known ham.
predsForHam2 = predictions2[ testDF2$isSpam == "F" ]
predsForSpam2 = predictions2[ testDF2$isSpam == "T" ]

errorRate2 = c(sum(predsForHam2 == "T") / length(predsForHam2),
               sum(predsForSpam2 == "F") / length(predsForSpam2))
#----------test.data--easy_ham_2&spam_2-----------------------------
predictions4 = predict(rpartFit2,
                       newdata = testDF[, names(testDF) != "isSpam"],
                       type = "class")
# See predictions for known ham.
predsForHam4 = predictions4[ testDF$isSpam == "F" ]
predsForSpam4 = predictions4[ testDF$isSpam == "T" ]

errorRate4 = c(sum(predsForHam4 == "T") / length(predsForHam4),
               sum(predsForSpam4 == "F") / length(predsForSpam4))
```

3) Compare:

```r
errorRate = rbind(errorRate1,errorRate3,errorRate2,errorRate4)
colnames(errorRate) = c("Type I","Type II")
row.names(errorRate) = c("NewMethod_Newtraindata","NewMethod_RandomSample"
                         , "TextMethod_Newtraindata", "TextMethod_RandomSample")
errorRate
```

```
##                              Type I   Type II
## NewMethod_Newtraindata  0.007857143 0.3414460
## NewMethod_RandomSample  0.011221407 0.3091364
## TextMethod_Newtraindata 0.053949072 0.1564456
## TextMethod_RandomSample 0.035000000 0.1360057
```

The new method has much samller Type I error rate but a little more Type II error rate. But analyzing aggregately, we can conclude that new method has totally less error rate than text method. In other words, this means new method is better.

4) Why difference exists:

New method training data ham percentage:

```r
trainHamIdx = c(grep("easy_ham[^_]",rownames(emailDFrp)),
                grep("hard_ham",rownames(emailDFrp)))
length(trainHamIdx)/length(trainIdx)
```

```
## [1] 0.8473515
```

testing data ham percentage:

```
textHamIdx = grep("easy_ham_2",rownames(emailDFrp))
length(textHamIdx)/(nrow(emailDFrp) - length(trainIdx))
```

```
## [1] 0.5005363
```

Text method training data ham percentage:

```
2/3
```

```
## [1] 0.6666667
```

Text method training data ham percentage:

```
1/3
```

```
## [1] 0.3333333
```

We can see that the prior distribution changed. From the text method to new method, ham percentage increased. So, it turns to more likely judge an email as ham.

To sum up, the new method traning data has larger percentage of ham data than that of text method. In the meanwhile, the percentage of trainham/textham for new method didn't change such much as the text method did. Therefore, based on the training data's difference, new method has much smaller Type I error rate.