Homework 2

Liting Hu

November 15, 2016

Problem 1

1. Logistic regression

```
# Logistic Regression
glm.fit <- glm(default~balance+student+income, data=Default, family=binomial)
summary(glm.fit)
summary(glm.fit$coefficients)

glm.probs <- predict(glm.fit, data=Default, type ="response")

glm.pred=rep("No", 10000)
glm.pred[glm.probs >.5]="Yes"

table(glm.pred, default)
mean(glm.pred == default)
# 0.9732

glm.probs <- predict(glm.fit, DefaultPredict, type="response")
glm.pred <- rep("No", dimDP[1])
glm.pred[glm.probs >.5] <- "Yes"
DefaultPredict$glm.pred <- glm.pred
```

2. Linear discriminant analysis

```
lda.fit <- lda(default~balance+student+income, data=Default)
lda.fit
plot(lda.fit)

lda.pred <- predict(lda.fit, Default)
lda.class <- lda.pred$class
table(lda.class, default)
mean(lda.class == default)
# 0.9724

sum(lda.pred$posterior[, 1] >= .5)
sum(lda.pred$posterior[, 1] < .5)
sum(lda.pred$posterior[, 1] > .9)

lda.probs <- predict(lda.fit, DefaultPredict, type="response")
lda.pred <- rep("No", dimDP[1])
lda.pred[lda.probs$posterior[, 2] > .5] <- "Yes"
DefaultPredict$lda.pred <- lda.pred
```

3. quadratic discriminant analysis

```
qda.fit <- qda(default~balance+student+income, data=Default)
qda.fit

qda.probs <- predict(qda.fit, Default)
qda.class <- qda.pred$class
table(qda.class, default)
mean(qda.class == default)
# 0.973

qda.probs <- predict(qda.fit, DefaultPredict, type="response")
DefaultPredict$qda.pred <- qda.probs$class
```

4. K-nearest neighbor classification

```
# K-Nearest Neighbors
Default.X <- cbind(Default$student, Default$balance, Default$income)
DefaultPredict.X <- cbind(DefaultPredict$student, DefaultPredict$balance, DefaultPredict$income)
knn.pred1 <- knn(Default.X, DefaultPredict.X, default, k = 1)
knn.pred3 <- knn(Default.X, DefaultPredict.X, default, k = 3)
knn.pred6 <- knn(Default.X, DefaultPredict.X, default, k = 6)
knn.pred10 <- knn(Default.X, DefaultPredict.X, default, k = 10)
DefaultPredict$knn.pred1 <- knn.pred1
DefaultPredict$knn.pred3 <- knn.pred3
DefaultPredict$knn.pred6 <- knn.pred6
DefaultPredict$knn.pred10 <- knn.pred10
detach(Default)
```

All results are stored in dataframe DefaultPredict:

```
> DefaultPredict
   index student      balance   income glm.pred lda.pred qda.pred knn.pred1 knn.pred3 knn.pred6 knn.pred10
1      1     Yes  911.8313379 16603.36       No       No       No        No        No        No         No
2      2      No  171.9919262 30733.33       No       No       No        No        No        No         No
3      3     Yes  771.7039646 13711.96       No       No       No        No        No        No         No
4      4      No  754.6104381 40701.96       No       No       No        No        No        No         No
5      5      No  618.7979243 59501.38       No       No       No        No        No        No         No
6      6      No  510.0996253 49837.10       No       No       No        No        No        No         No
7      7      No  886.2620474 52852.43       No       No       No        No        No        No         No
8      8      No  831.1602780 39944.24       No       No       No        No        No        No         No
9      9     Yes  400.9717764 15317.38       No       No       No        No        No        No         No
10    10      No 1092.2819530 45468.86       No       No       No        No        No        No         No
11    11      No    0.1797604 41724.94       No       No       No        No        No        No         No
12    12     Yes  996.9292528 20057.01       No       No       No        No        No        No         No
13    13      No  376.3586612 25420.12       No       No       No        No        No        No         No
14    14      No  654.0181025 54844.04       No       No       No        No        No        No         No
15    15      No 1109.1008990 45525.52       No       No       No        No        No        No         No
16    16      No  937.2591203 56696.03       No       No       No        No        No        No         No
17    17     Yes  184.3792088 14995.35       No       No       No        No        No        No         No
18    18      No  710.9217520 53003.81       No       No       No        No        No        No         No
19    19      No  747.1608518 19649.92       No       No       No        No        No        No         No
20    20      No  852.5276677 58628.33       No       No       No        No        No        No         No
21    21      No 1567.9187140 36668.33       No       No       No        No        No        No         No
22    22     Yes  187.4985488 16885.93       No       No       No        No        No        No         No
23    23     Yes 1490.8859400 17899.42       No       No       No        No        No        No         No
24    24     Yes 2204.5416940 14309.70      Yes      Yes      Yes       Yes       Yes        No         No
25    25     Yes 1777.7084840 20363.12       No       No       No       Yes        No        No         No
26    26      No 1877.1215410 48977.54      Yes       No       No       Yes       Yes       Yes         No
27    27     Yes 1902.5514700 20575.27       No       No       No       Yes        No        No         No
28    28     Yes 1571.8059100 14911.16       No       No       No       Yes        No        No         No
29    29      No 1968.9988070 39084.55      Yes      Yes      Yes       Yes       Yes        No        Yes
30    30      No 1526.1299570 30008.75       No       No       No       Yes        No        No         No
31    31      No 1641.5593160 46818.25       No       No       No       Yes        No        No         No
32    32      No 2004.5460050 42163.04      Yes      Yes      Yes       Yes       Yes       Yes        Yes
33    33      No 1550.4492640 56258.10       No       No       No       Yes        No        No         No
34    34      No 1327.8387310 34719.57       No       No       No       Yes        No        No         No
35    35      No 1700.2979030 30458.13       No       No       No        No        No        No         No
36    36     Yes 1118.1097410 21884.13       No       No       No       Yes        No        No         No
37    37      No 1132.5249430 37242.38       No       No       No       Yes        No        No         No
38    38      No 1980.7302230 28134.70      Yes      Yes       No       Yes       Yes       Yes         No
39    39      No 1708.7024140 38350.59       No       No       No       Yes       Yes       Yes         No
40    40      No 1457.0124430 58698.10       No       No       No       Yes        No        No         No
41    41      No 1767.1601410 46209.69       No       No       No       Yes        No        No         No
42    42     Yes 1759.1077950 15980.12       No       No       No       Yes        No        No         No
```

Figure 1: Default predict under different measures

In K-nearest neighbor classification, the results under $k = 1$ and $k = 10$ seem unreasonable compared to other classifications. Choosing a k value between 3 and 6 may produce a better result.

Problem 2

1. Read the Email Messages

```
spamPath = "/Users/apple/Desktop/582/RSpamData"
dirNames = list.files(path = paste(spamPath, "messages",
                                   sep = .Platform$file.sep))
length(list.files(paste(spamPath, "messages", dirNames,
                        sep = .Platform$file.sep)))
```

```
# [1] 9353
sapply(paste(spamPath, "messages", dirNames,
             sep = .Platform$file.sep),
       function(dir) length(list.files(dir)) )

fullDirNames = paste(spamPath, "messages", dirNames,
                     sep = .Platform$file.sep)
```

There are 9353 messages in the 5 directories combined.

2. Find the Words in a Message

Firstly, define a function to split the message into header and body. The body of a message is separated from the header by a single empty line.

```
splitMessage = function(msg) {
    splitPoint = match("", msg)
    header = msg[1:(splitPoint-1)]
    body = msg[ -(1:splitPoint) ]
    return(list(header = header, body = body))
}
```

Then we should remove attachments from the message. Based on the anatomy of email messages, If an attachment is added to a message, the MIME type is multipart and the Content-Type field provides a boundary string that can be used to locate the attachments. To get the boundary, define function:

```
getBoundary = function(header) {
    boundaryIdx = grep("boundary=", header)
    boundary = gsub('"', "", header[boundaryIdx])
    gsub(".*boundary= *([^;]*);?.*", "\\1", boundary)
}
```

Then we can remove attachments from messages by

```
dropAttach = function(body, boundary){

    bString = paste("--", boundary, sep = "")
    bStringLocs = which(bString == body)
    # Search for the boundary string
    if (length(bStringLocs) <= 1) return(body)

    eString = paste("--", boundary, "--", sep = "")
    eStringLoc = which(eString == body)
    # The closing boundary
    if (length(eStringLoc) == 0)
        return(body[ (bStringLocs[1] + 1) : (bStringLocs[2] - 1)])

    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) ])
}
```

After all these steps, words can be extracted from the messages.

```
cleanText =
    function(msg)   {
        tolower(gsub("[[:punct:]0-9[:space:][:blank:]]+", " ", msg))
    }

findMsgWords =
    function(msg) {
        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]
        invisible(words)
    }

processAllWords = function(dirName)
{
    # read all files in the directory
    fileNames = list.files(dirName, full.names = TRUE)
    # drop files that are not email, i.e., cmds
    notEmail = grep("cmds$", fileNames)
    if ( length(notEmail) > 0) fileNames = fileNames[ - notEmail ]

    messages = lapply(fileNames, readLines, encoding = "latin1")

    # split header and body
    emailSplit = lapply(messages, splitMessage)
    # put body and header in own lists
    bodyList = lapply(emailSplit, function(msg) msg$body)
    headerList = lapply(emailSplit, function(msg) msg$header)
    rm(emailSplit)

    # determine which messages have attachments
    hasAttach = sapply(headerList, function(header) {
        CTloc = grep("Content-Type", header)
        if (length(CTloc) == 0) return(0)
        multi = grep("multi", tolower(header[CTloc]))
        if (length(multi) == 0) return(0)
        multi
    })

    hasAttach = which(hasAttach > 0)

    # find boundary strings for messages with attachments
    boundaries = sapply(headerList[hasAttach], getBoundary)

    # drop attachments from message body
    bodyList[hasAttach] = mapply(dropAttach, bodyList[hasAttach],
                                 boundaries, SIMPLIFY = FALSE)

    # extract words from body
    msgWordsList = lapply(bodyList, findMsgWords)
```

```
    invisible(msgWordsList)
}
msgWordsList = lapply(fullDirNames, processAllWords)
```

For now we have collected all the necessary words from all the emails. Create a logical vector to define whether the messages are spam or not based on the number of elements in each list and flatten all five lists into one list:

```
numMsgs = sapply(msgWordsList, length)
numMsgs
# [1] 5051 1400  500 1000 1397

isSpam = rep(c(FALSE, FALSE, FALSE, TRUE, TRUE), numMsgs)
msgWordsList = unlist(msgWordsList, recursive = FALSE)
```

3. The Naive Bayes Classifier

For spam messages (similar to ham messages), we should estimate the following probabilities (1/2 used to avoid zero probabilities):

$$P(\text{awordispresent}|\text{spam}) \approx \frac{\#\text{ofspammessageswiththisword} + 1/2}{\#\text{ofspammessages} + 1/2}$$

$$P(\text{awordisabsent}|\text{spam}) \approx \frac{\#\text{ofspammessageswithoutthisword} + 1/2}{\#\text{ofspammessages} + 1/2}$$

We use log of ratios to avoid products and it tends to have better statistical properties. So for a new message we compute the log likelihood ratio as

$$\sum_{\text{wordsinmessage}} (\log P(\text{wordpresent}|\text{spam}) - \log P(\text{wordpresent}|\text{ham})) +$$
$$\sum_{\text{wordsnotinmessage}} (\log P(\text{wordabsent}|\text{spam}) - \log P(\text{wordabsent}|\text{ham})) + \log P(\text{spam}) - \log P(\text{ham})$$

where $\log P(\text{spam}) - \log P(\text{ham})$ should be constant.
To collect these probabilities, construct the function that returns all log likelihood ratios as a matrix:

```
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 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)
    }

trainTable = computeFreqs(trainMsgWords, trainIsSpam)
```

The trainTable can be used to compute the log likelihood ratio for a new message. This value is used to classify the message as spam or ham. Besides, we apply function computeMsgLLR to each of the messages in our test set

```
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])
}
```

To compare the summary statistics of the LLR values for the ham and spam in the test data with

```
> tapply(testLLR, testIsSpam, summary)
$`FALSE`
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-1367.00  -132.50  -104.60  -120.50   -84.96   697.80

$`TRUE`
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
  -63.650     9.347    56.330   142.500   138.300 23600.000
```

and the boxplots for these data:

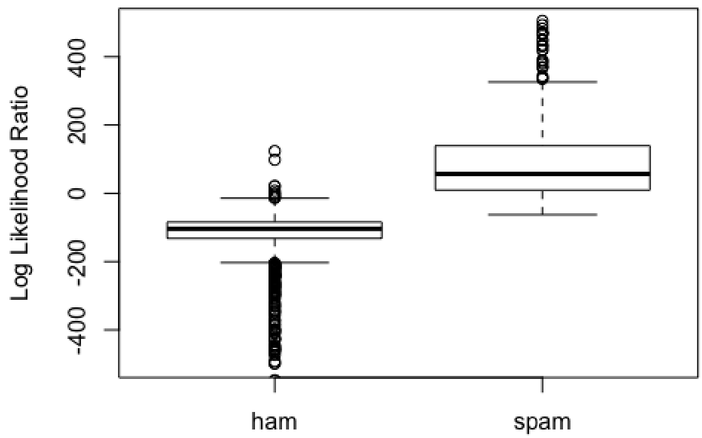


Figure 2: Boxplot of log likelyhood ratio for spam and ham

which shows that most ham data have log likelihood ratio below zero and spam data above zero. So naive Bayes approximation is a good way to group spam or ham messages. And we need to decide a cut-off τ which used to judge a message is spam or not.

Before that, define functions to calculate type I and type II errors. Then draw the plot to display these two errors in different τ.

```
typeIErrorRates =
    function(llrVals, isSpam)
    {
        o = order(llrVals)
        llrVals =  llrVals[o]
        isSpam = isSpam[o]

        idx = which(!isSpam)
        N = length(idx)
        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])
}
```

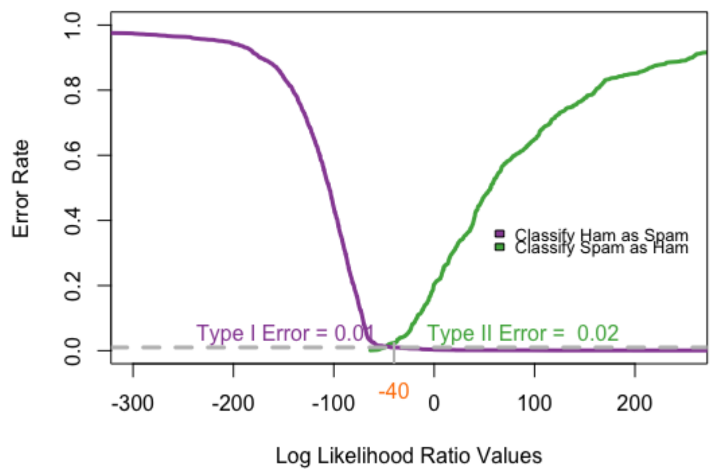


Figure 3: Type I and II Error Rates

with a threshold of $\tau = -40$, messages with an LLR value above -40 will be classified as spam messages while others as ham. Since in this case, type I error is 0.01 and type II error is 0.023, 1% of ham is misclassified as spam and 2.3% of spam is misclassified as ham.

4. Result by removing the stop words

Stop words also can be found in "tm" package and they are easily extracted as a vector.

```
library(tm)
stopwords <- stopwords()
```

Then add a function after processAllWords to remove stop words as:

```
removeStopWords =
    function(x, stopWords)
    {
        stopWords <- stopwords()
        if(is.character(x))
            setdiff(x, stopWords)
        else if(is.list(x))
            lapply(x, removeStopWords, stopWords)
        else x
    }

msgWordsList = lapply(msgWordsList, removeStopWords, stopWords)
```

Other steps are similar. Then we have the result under the condition that the stop words are removed. The summary statistics of the LLR values for the ham and spam in the test data are

```
> tapply(testLLR, testIsSpam, summary)
$`FALSE`
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-1363.00  -127.90  -101.60  -116.90   -81.32   700.00

$`TRUE`
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
  -60.33     7.15    50.01   138.50   131.70 23600.00
```

And the boxplot:

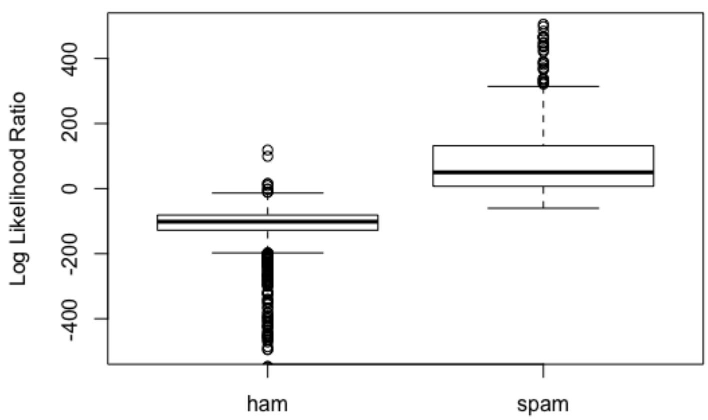


Figure 4: Boxplot of log likelyhood ratio for spam and ham when stop words are removed

Type I and type II Error Rates:

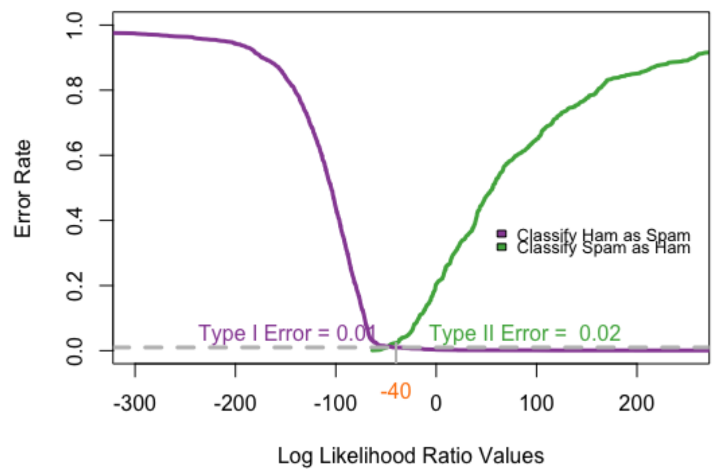


Figure 5: Type I and II Error Rates when stop words are removed

That is just a little different with the situation that remains stop words. Setting $\tau = -43$, messages with an LLR value above -43 will be classified as spam messages while others as ham. Since in this case, type I error is 0.01 and type II error is 0.02, 1% of ham is misclassified as spam and 2% of spam is misclassified as ham.

Problem 3

1. Get and Read the Data

Firstly, download the historical stock data from Yahoo Finance and save these contents to local files.

```
library(XML)
u = paste("http://real-chart.finance.yahoo.com/table.csv",
                "s=PEP&a=5&b=1&c=1972&d=10&e=9&f=2016&g=d",sep = "?")
download.file(u,"PEP.csv")
u = paste("http://real-chart.finance.yahoo.com/table.csv",
                "s=KO&a=0&b=2&c=1962&d=10&e=9&f=2016&g=d",sep = "?")
download.file(u,"KO.csv")
u = paste("http://real-chart.finance.yahoo.com/table.csv",
                "s=DPS&a=4&b=7&c=2008&d=10&e=9&f=2016&g=d",sep = "?")
download.file(u,"DPS.csv")
# Download CSV files
```

Then read the data and make sure that the date values remain as strings and not interpreted as a factor but converted into date format.

```
readData =
    #read the data and convert the Date column to an object of class Date.
    function(fileName, dateFormat = c("%Y-%m-%d", "%Y/%m/%d"), ...)
    {
        data = read.csv(fileName, header = TRUE,
                        stringsAsFactors = FALSE, ...)
```

```
        for(fmt in dateFormat) {
            tmp = as.Date(data$Date, fmt)
            if(all(!is.na(tmp))) {
                data$Date = tmp
                break
            }
        }

        data[ order(data$Date), ]
    }

pep <- readData("PEP.csv")
ko <- readData("KO.csv")
dps <- readData("DPS.csv")
```

2. Proceed the Data

For now we use stock "PEP" and "KO" to do the pairs trading.

Define a function to compute the subsets with common dates and return a data frame with records for each day with the adjusted closing prices for the two stocks. After combining stock "PEP" and "KO", we can create the ratio of the adjusted closing price.

```
combine2Stocks =
    function(a, b, stockNames = c(deparse(substitute(a)),
                                  deparse(substitute(b))))
    {
        rr = intersect(a$Date, b$Date)
        a.sub=a[which(a$Date %in% rr),]
        b.sub=b[which(b$Date %in% rr),]
        structure(data.frame(as.Date(a.sub$Date),
                             a.sub$Adj.Close,
                             b.sub$Adj.Close),
                  names = c("Date", stockNames))
    }
overlap = combine2Stocks(pep, ko)
r = overlap$pep/overlap$ko
```

Visualize the ratio ($k = 1$):

```
plotRatio =
    function(r, k = 1, date = seq(along = r), ...)
    {
        plot(date, r, type = "l", ...)
        abline(h = c(mean(r),
                     mean(r) + k * sd(r),
                     mean(r) - k * sd(r)),
               col = c("darkgreen", rep("red", 2*length(k))),
               lty = "dashed")
    }

plotRatio(r, k = 1, overlap$Date, col = "lightgray", xlab = "Date", ylab = "Ratio")
length(r)
```

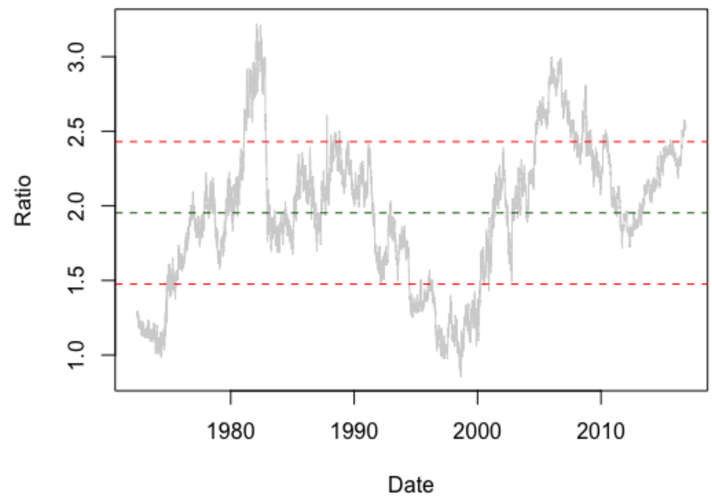


Figure 6: The ratio of stock prices for PEP and KO

3. Finding positions

In our case, $k = 1$, the upper bound is $m + ks$ and the lower bound is $m - ks$. Compute two opening positions (green dots) and two closing positions (red dots). The plot was drawn below.

```
findNextPosition =
    # Check they are increasing and correctly offset
    function(ratio, startDay = 1, k = 1,
             m = mean(ratio), s = sd(ratio))
    {
        up = m + k *s
        down = m - k *s

        if(startDay > 1)
            ratio = ratio[ - (1:(startDay-1)) ]

        isExtreme = ratio >= up | ratio <= down

        if(!any(isExtreme))
            return(integer())

        start = which(isExtreme)[1]
        backToNormal = if(ratio[start] > up)
            ratio[ - (1:start) ] <= m
        else
            ratio[ - (1:start) ] >= m

        # return either the end of the position or the index
        # of the end of the vector.
        # Could return NA for not ended, i.e. which(backToNormal)[1]
        # for both cases. But then the caller has to interpret that.
```

```
        end = if(any(backToNormal))
            which(backToNormal)[1] + start
        else
            length(ratio)

        c(start, end) + startDay - 1
    }

k = 1
a = findNextPosition(r, k = k)          #1 1095
b = findNextPosition(r, a[2], k = k)    #2186 2661

symbols(overlap$Date[a[1]],r[a[1]],circles = 60,fg = "darkgreen", add=TRUE, inches = FALSE)
symbols(overlap$Date[a[2]],r[a[2]],circles = 60,fg = "red", add=TRUE, inches = FALSE)
symbols(overlap$Date[b[1]],r[b[1]],circles = 60,fg = "darkgreen", add=TRUE, inches = FALSE)
symbols(overlap$Date[b[2]],r[b[2]],circles = 60,fg = "red", add=TRUE, inches = FALSE)
```

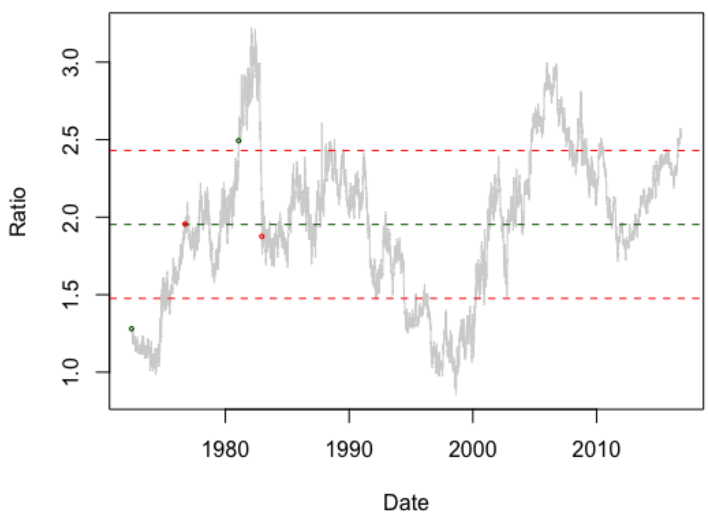


Figure 7: Position finding function test

In order to find all positions, define the function “getPositions”:

```
getPositions =
    function(ratio, k = 1, m = mean(ratio), s = sd(ratio))
    {
        when = list()
        cur = 1

        while(cur < length(ratio)) {
            tmp = findNextPosition(ratio, cur, k, m, s)
            if(length(tmp) == 0)  # done
                break
```

```
            when[[length(when) + 1]] = tmp
            if(is.na(tmp[2]) || tmp[2] == length(ratio))
                break
            cur = tmp[2]
        }

        when
    }

pos <- getPositions(r,k)
plotRatio(r, k, overlap$Date, col = "lightgray", xlab = "Date", ylab = "Ratio")
invisible(lapply(pos, function(p) showPosition(overlap$Date[p],r[p])))
```

Then draw the plot.

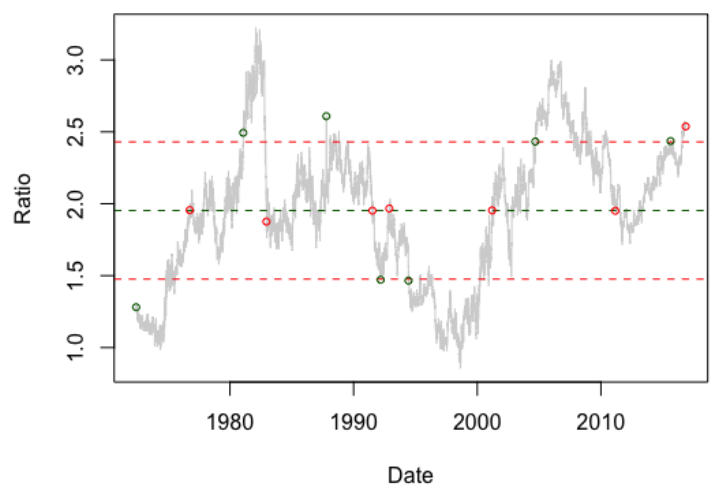


Figure 8: All positions when k=1

Set $k = 0.5$, the plot is showed below. There are much more positions than that in $k = 1$.

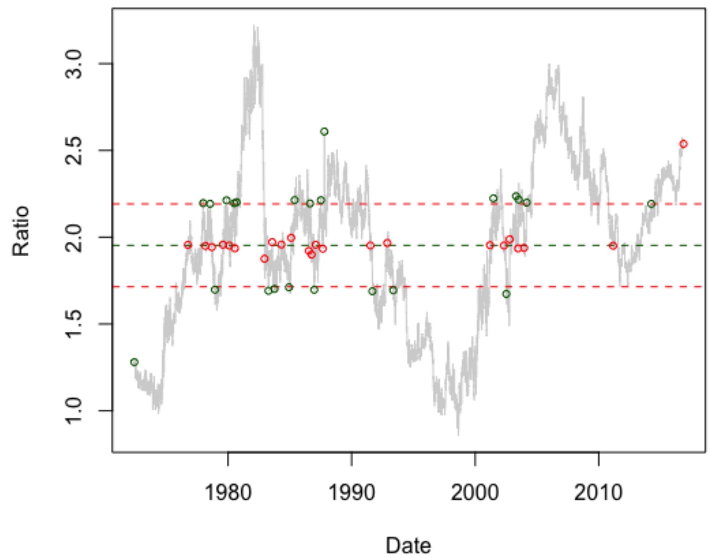


Figure 9: All positions when k=0.5

More positions existing means that we may earn more but cost more in transaction fees. To pursue the maximum earnings, we should find appropriate value k based on the historical data for our pair of stocks.

4. Computing the Profit and Find the Optimal Value for k

Calculate the position profits:

```
positionProfit =
    #  r = overlap$pep/overlap$ko
    #  pos = getPositions(r, k)
    #  positionProfit(pos[[1]], overlap$pep, overlap$ko)
    function(pos, stockPriceA, stockPriceB,
             ratioMean = mean(stockPriceA/stockPriceB),
             p = .0002, byStock = FALSE)
    {
        if(is.list(pos)) {
            ans = sapply(pos, positionProfit,
                         stockPriceA, stockPriceB, ratioMean, p, byStock)
            if(byStock)
                rownames(ans) = c("A", "B", "commission")
            return(ans)
        }
        # prices at the start and end of the positions
        priceA = stockPriceA[pos]
        priceB = stockPriceB[pos]

        # how many units can we by of A and B with $1
        unitsOfA = 1/priceA[1]
```

```
        unitsOfB = 1/priceB[1]

        # The dollar amount of how many units we would buy of A and B
        # at the cost at the end of the position of each.
        amt = c(unitsOfA * priceA[2], unitsOfB * priceB[2])

        # Which stock are we selling
        sellWhat = if(priceA[1]/priceB[1] > ratioMean) "A" else "B"

        profit = if(sellWhat == "A")
            c((1 - amt[1]),  (amt[2] - 1), - p * sum(amt))
        else
            c( (1 - amt[2]),  (amt[1] - 1),  - p * sum(amt))

        if(byStock)
            profit
        else
            sum(profit)
    }
```

When calculating for PEP/KO ratios, the result is:

```
[1]  0.37333954  0.42307159  0.99590876  0.34030945  0.84320973  0.35930336 -0.04535196
```

which seems reasonable.
Then to find the optimal value for k, divide the data into a training period and a test period. we just use the training period to determine optimal value.

```
i = 1:floor(nrow(overlap)/2)
train = overlap[i, ]
test = overlap[ - i, ]
# Divide the data half to a training period and and half to a test period

r.train = train$pep/train$ko
r.test = test$pep/test$ko

k.max = max((r.train - mean(r.train))/sd(r.train))
k.min = min((abs(r.train - mean(r.train))/sd(r.train)))
ks = seq(k.min, k.max, length = 1000)
m  = mean(r.train)

profits =
    sapply(ks,
           function(k) {
               pos = getPositions(r.train, k)
               sum(positionProfit(pos, train$pep, train$ko,
                                  mean(r.train)))
           })

plot(ks, profits, type = "l", xlab = "k", ylab = "Profit")
tmp.k = ks[profits == max(profits)]
# [1] 0.03661572
pos = getPositions(r.test, tmp.k, mean(r.train), sd(r.train))
testProfit = sum(positionProfit(pos, test$pep, test$ko))
testProfit
# [1] 1.548414
```

Loop over 1000 values of k and compute the 1000 corresponding profits. We can see the relationship between k and profits as below

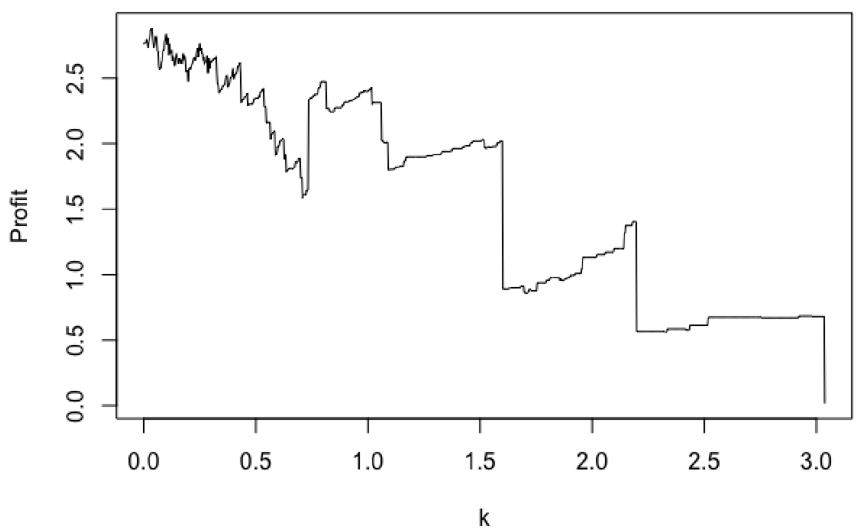


Figure 10: Relationship between profits and k

The profit reach its maximum 1.548414 (about 155%) at $k = 0.03661572$.

5. Pairs trading on PEP and DPS

All steps are similar to former trading.

```
# PEP vs DPS
overlap_pd <- combine2Stocks(pep, dps)
r_pd <- overlap_pd$ko/overlap_pd$dps

i = 1:floor(nrow(overlap_pd)/2)
train = overlap_pd[i, ]
test = overlap_pd[ - i, ]
# Divide the data half to a training period and and half to a test period

r.train = train$pep/train$dps
r.test = test$pep/test$dps
sd(r.train)
k.max = min(max((r.train - mean(r.train))/sd(r.train)),
            (mean(r.train)-min(r.train))/sd(r.train))
#k.max = max((r.train - mean(r.train))/sd(r.train))
k.min = max(0.1, min((abs(r.train - mean(r.train))/sd(r.train))))
ks = seq(k.min, k.max, length = 1000)
m  = mean(r.train)

profits =
    sapply(ks,
```

```
            function(k) {
                pos = getPositions(r.train, k)
                sum(positionProfit(pos, train$pep, train$dps,
                                   mean(r.train)))
            })

plot(ks, profits, type = "l", xlab = "k", ylab = "Profit")
tmp.k = ks[profits == max(profits)]
#[1] 1.061245 1.062219 1.063193 1.064167 1.065141 1.066115 1.067089 1.068063 1.069037 1.070011
#[11] 1.070984 1.071958

k.star = mean(ks[profits == max(profits)])
# [1] 1.066602

pos = getPositions(r.test, k.star)
testProfit = sum(positionProfit(pos, test$pep, test$dps))
testProfit
```

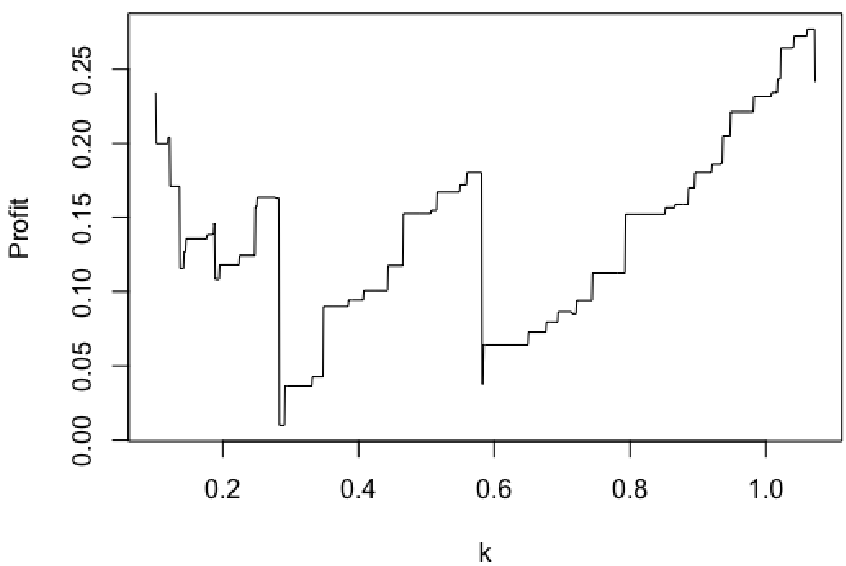
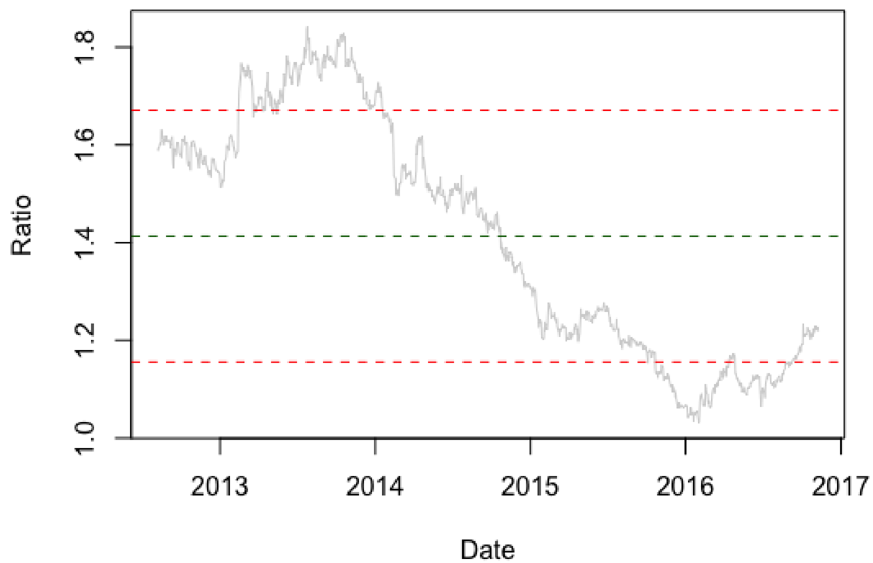


Figure 11: Relationship between profits and k in pairs trading PEP vs DPS

The profit reach its maximum 0.3586618 (about 35.9%) at $k = 1.066602$.



6. Pairs trading on KO and DPS

```
# KO vs DPS
overlap_kd <- combine2Stocks(ko, dps)
r_kd <- overlap_kd$ko/overlap_kd$dps

i = 1:floor(nrow(overlap_kd)/2)
train = overlap_kd[i, ]
test = overlap_kd[ - i, ]
# Divide the data half to a training period and and half to a test period

r.train = train$ko/train$dps
r.test = test$ko/test$dps

k.max = max((r.train - mean(r.train))/sd(r.train))
k.min = max(0.1, min((abs(r.train - mean(r.train))/sd(r.train))))
ks = seq(k.min, k.max, length = 1000)
m  = mean(r.train)

profits =
    sapply(ks,
           function(k) {
               pos = getPositions(r.train, k)
               sum(positionProfit(pos, train$ko, train$dps,
                                  mean(r.train)))
           })

plot(ks, profits, type = "l", xlab = "k", ylab = "Profit")
tmp.k = ks[profits == max(profits)]
# [1] 0.4365644 0.4402227
```

```
k.star = mean(ks[profits == max(profits)])
# [1] 0.4383935

pos = getPositions(r.test, k.star)
testProfit = sum(positionProfit(pos, test$ko, test$dps))
testProfit
```

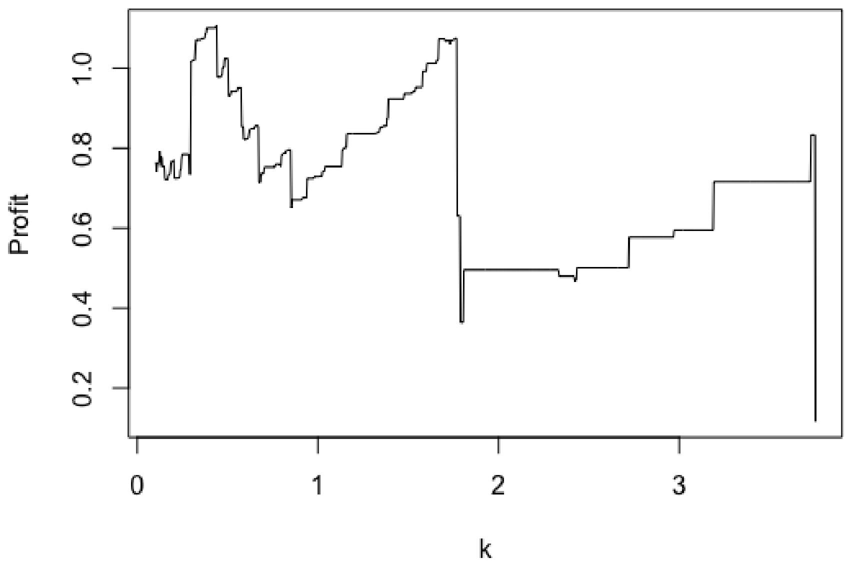


Figure 12: Relationship between profits and k in pairs trading KO vs DPS

The profit reach its maximum 0.1493933 (about 14.9%) at $k = 0.4383935$.

0.9
0.8
0.7
0.6
0.5
Ratio
2013
2014
2015
2016
2017
Date