Solution 4

STAT6306

Introduction

For this assignment, let's attempt to make a spam filter. Usually, this would involve a lot of text processing on a huge number of emails. In this case, someone has created a feature matrix for us. The feature matrix has rows given by individual emails and columns given by the number of each word or character that appears in that email, as well as three different numerical measures regarding capital letters (average length of consecutive capitals, longest sequence of consecutive capitals, and total number of capital letters).
The supervisor, Y, is given by the user supplied label marking that email as either spam (Y = 1) or not (Y = 0). Here is a function that may be useful for this assignment:

```
misClass =function(pred.class,true.class,produceOutput=FALSE){
  confusion.mat = table(pred.class,true.class)
  if(produceOutput){
    return(1-sum(diag(confusion.mat))/sum(confusion.mat))
  }
  else{
    print('miss-class')
    print(1-sum(diag(confusion.mat))/sum(confusion.mat))
    print('confusion mat')
    print(confusion.mat)
  }
}
# this can be called using:
#     (assuming you make the appropriately named test predictions)
# misClass(Y.hat,Y_0)
```

Read in the R data set:

```
load("spam.Rdata")
```

Let's make a training and test set.

```
train = spam$train
test  = !train
X     = spam$XdataF[train,]
X_0   = spam$XdataF[test,]
Y     = factor(spam$Y[train])
Y_0   = factor(spam$Y[test])
```

Install necessary packages

```
repos = 'http://cran.us.r-project.org'
if(!require('randomForest')){install.packages('randomForest',repos = repos);require('randomForest')}
```

```
## Loading required package: randomForest
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

Question 1: Choosing the number of bagging iterations

To save computations, it is common to iteratively compute batches of random trees until the OOB error rate stabilizes. The following function implements does this as well as demonstrate some typical programming strategies:

```
checkNumberItersRF = function(ntrees = 10, tolParm = 1, maxIter = 10, verbose = 0){
  ###
  # tolParm: iterations will continue until the percent decrease
  #          is less than tolParm
  ###
  misClass_out   = list()
  totalTrees_out = list()

  n              = nrow(X)
  votes          = matrix(0,nrow=n,ncol=2)
  totalTrees     = 0
  iterations     = 0
  misClass_old   = 1
  while(iterations < maxIter){
    votes[is.nan(votes)] = 0
    iterations    = iterations + 1
    totalTrees    = totalTrees + ntrees
    if(verbose >= 2){cat('Total trees: ',totalTrees,'\n')}
    out.rf        = randomForest(X, Y,ntree = ntrees)

    oob.times        = out.rf$oob.times
    votes_iterations = out.rf$votes*oob.times
    votes[oob.times>0,] = matrix(votes + votes_iterations,nrow=n)[oob.times>0,]
    if(min(apply(votes,1,sum)) == 0){next}

    Yhat          = apply(votes,1,which.max) - 1
    misClass_new  = misClass(Yhat,Y,produceOutput = TRUE)
    misClass_out[[iterations]]   = misClass_new
    totalTrees_out[[iterations]] = totalTrees
    percentChange = 100*(misClass_new - misClass_old)/misClass_old
    if(verbose >= 1){cat('% change: ',percentChange,'\n')}
    if(percentChange > -tolParm){break}
    misClass_old = misClass_new
  }
  if(iterations == maxIter){
    stop("too many iterations, try a larger ntrees or maxIter value")
  }
  return(list('misClass' = unlist(misClass_out),
              'totalTree' = unlist(totalTrees_out)))
}
```

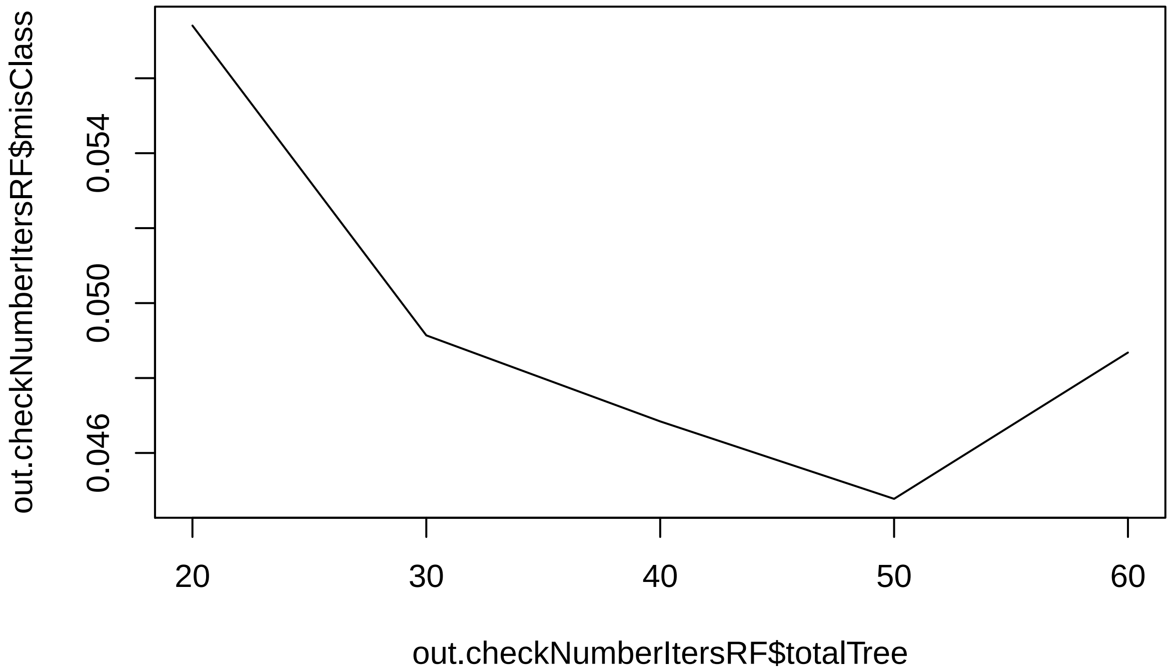
Comment on the roll of each of these pieces in the above function:

- next: If the condition is satisfied, then go to the next iteration
- maxIter: the maximum number of iterations that will be attempted by the function. This is important anytime a while loop is used to prevent an infinite loop.
- verbose: a conventional parameter used to specify how much printed output should be produced. Larger values tend to correspond to more output
- while: a loop that differs from a for loop by not having a prespecified number of iterations

- tolParm: a threshold defining the criteria for the while loop to terminate. If the solution doesn't change too much, then there is no point in continuing the computations
- misClass_old: This is a placeholder that allows us to compare the solution at the current iteration to the solution at the previous iteration
- stop: Stops the function and produces an error message

Call this function with a suitable value of maxIter and verbose so that the function produces the minimal amount of output. Report back the number of iterations you find

```
out.checkNumberItersRF = checkNumberItersRF(maxIter = 10, verbose = 0)
plot(out.checkNumberItersRF$totalTree,out.checkNumberItersRF$misClass,
     type='l')
```



SOLUTION

Since the reported B will be random (due to random forest being random), for this particular run, I get an ensemble size of:

```
print(max(out.checkNumberItersRF$totalTree))
```

```
## [1] 60
```

Question 2: Random Forest classifications

What is the test misclassification rate, sensitivity, specificity, precision, recall, and confusion matrix for the chosen random forest? How does the test misclassification rate compare with the OOB misclassification rate?

(Extra challenge: the above code doesn't save the random forests, only the OOB miss-classification rates. This approach requires that a new call to randomForest ismade with the chosen number of iterations. However, this is redundant as we could just save the intermediate trees. If you want, try and adapt the code to retain this information).

```r
#SOLUTION
ntrees = max(out.checkNumberItersRF$totalTree)
out.rf = randomForest(X, Y, ntree = ntrees)
class.rf = predict(out.rf,X_0,type='class')
misClass(class.rf,Y_0)
```

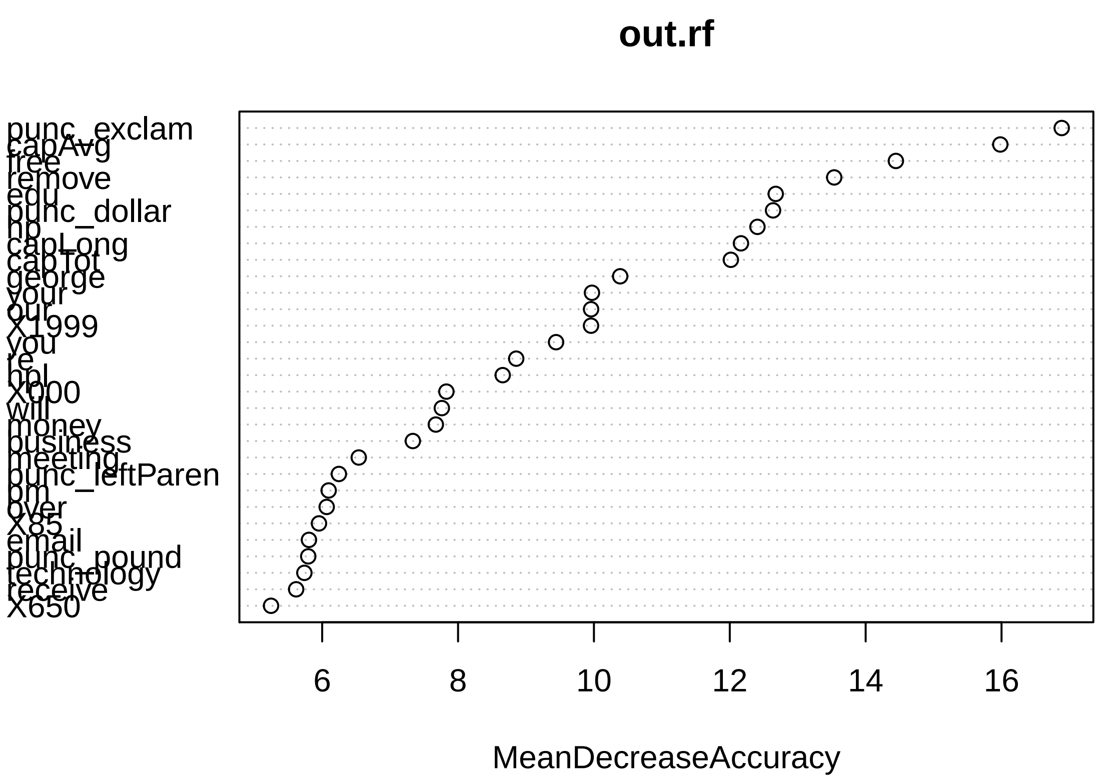
```
## [1] "miss-class"
## [1] 0.04471545
## [1] "confusion mat"
##            true.class
## pred.class   0   1
##          0 130   8
##          1   3 105
```

Question 3: Variable importance for random forests

Get a variable importance plot for the permuted OOB importance measure. Also, produce a proximity plot to visualize the emails. Describe both of these plots. Identify an extreme observation via the proximity plot. What is something notable about this email (I'm leaving this vague. There can be many answers to this question. Investigate!)

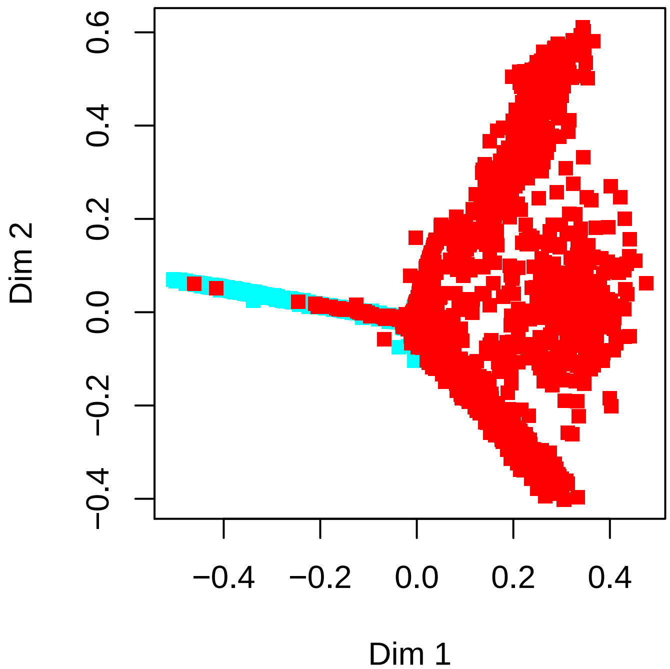
```r
#SOLUTION
out.rf = randomForest(X, Y, importance = TRUE,
                      proximity=TRUE, ntree = ntrees)

varImpPlot(out.rf,type=1)
```




```r
#I'm commenting this out to cut down on the size of the knitted document
#image(1 - out.rf$proximity)
```

```r
MDSplot(out.rf, fac = Y,
        palette = rainbow(2),pch = 15)
```




```r
###
# Here is my further investigating.  I'm looking for/at the
# 'not spam' email that has the smallest value on the first
# proximity plot dimension.  The "cmdscale" is implicitly
# called by the "MDSplot" function.  We are calling it here
# so that I can identify the point.
###
out.mds          = cmdscale(1 - out.rf$proximity, eig = TRUE, k = 2)
proximityCoords = out.mds$points

orderDim1           = order(proximityCoords[,1],decreasing = FALSE)
orderDim1_Yequals0 = orderDim1[Y[orderDim1] == 0]
extremeObs          = orderDim1_Yequals0[1]

#I've identified an extreme obs. from the MDS plot.  Now, let's look at which
#entries of that observation seem to be large relative to sample quantiles
(importantFeatures = names(X)[X[extremeObs,] > apply(X,2,quantile,probs=.95)])
```

```
## [1] "internet" "business" "credit"   "your"
```

```r
require(dplyr)
```

```
## Loading required package: dplyr

## Warning: package 'dplyr' was built under R version 3.4.1

##
## Attaching package: 'dplyr'

## The following object is masked from 'package:randomForest':
##
```

```
##     combine
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

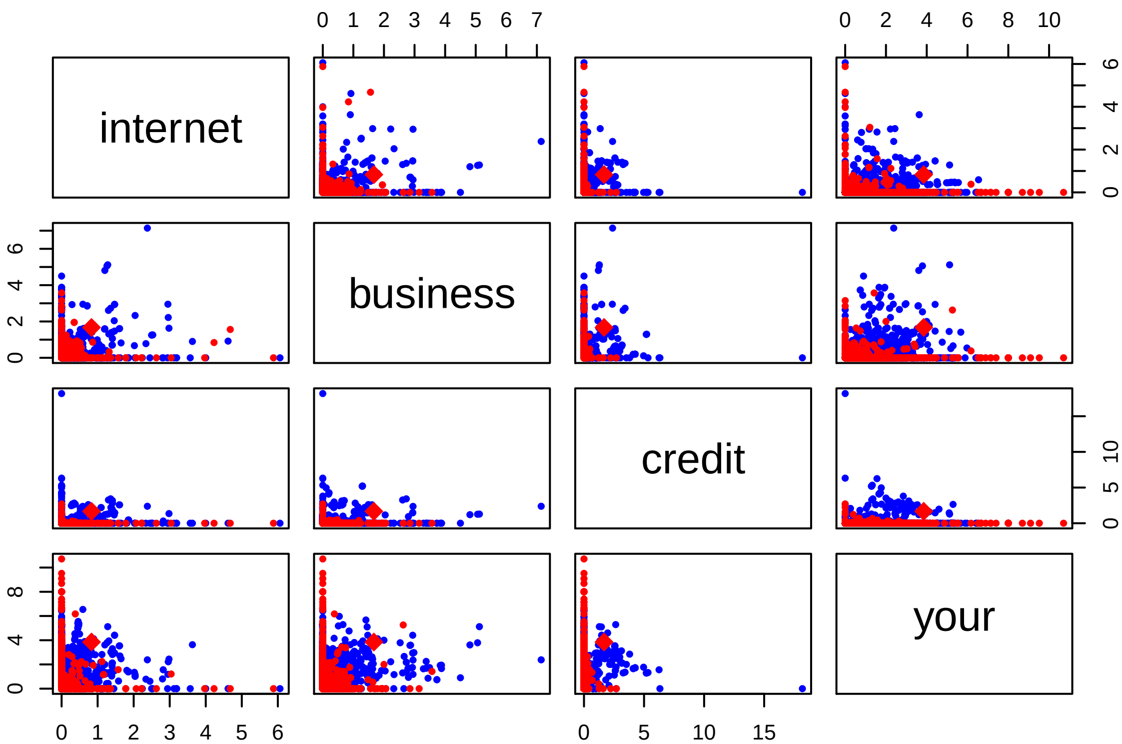
```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
# dplyr is very useful
XimportantFeatures = select(X, one_of(importantFeatures))

n   = nrow(X)
pch = rep(16,n)
pch[extremeObs] = 18
cex = rep(0.75,n)
cex[extremeObs] = 2

colVec          = rep('red',n)
colVec[Y == 1] = 'blue'

plot(XimportantFeatures,pch=pch,cex=cex,col=colVec)
```




```
#This is an abnormally large observations along the displayed features. Note that it is not labelled sp
```

Question 4: Pruned classification tree

Fit an unpruned classification tree to the training data. Prune the tree via weakest-link pruning (i.e. using the cv.tree and prune.misclass pair of functions as shown in lecture). Plot the pruned tree.

```r
#SOLUTION
require(tree)
```

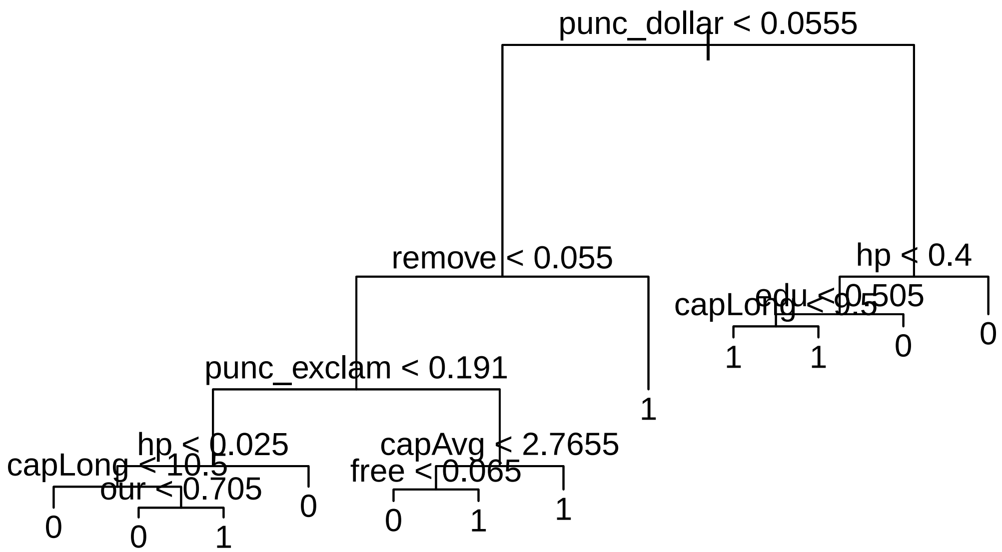
```
## Loading required package: tree
```

```r
out.tree.orig   = tree(Y~.,data=X)
out.tree.cv     = cv.tree(out.tree.orig,FUN=prune.misclass)
best.size       = out.tree.cv$size[which.min(out.tree.cv$dev)]
best.size       = out.tree.cv$size[max(which(out.tree.cv$dev == min(out.tree.cv$dev)))]
best.size
```

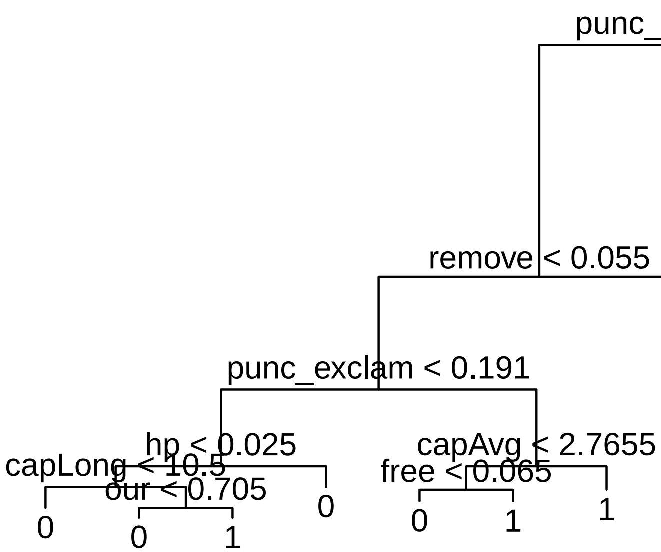
```
## [1] 11
```

```r
out.tree        = prune.misclass(out.tree.orig,best=best.size)
class.tree      = predict(out.tree,X_0,type='class')

plot(out.tree.orig)
text(out.tree.orig)
```



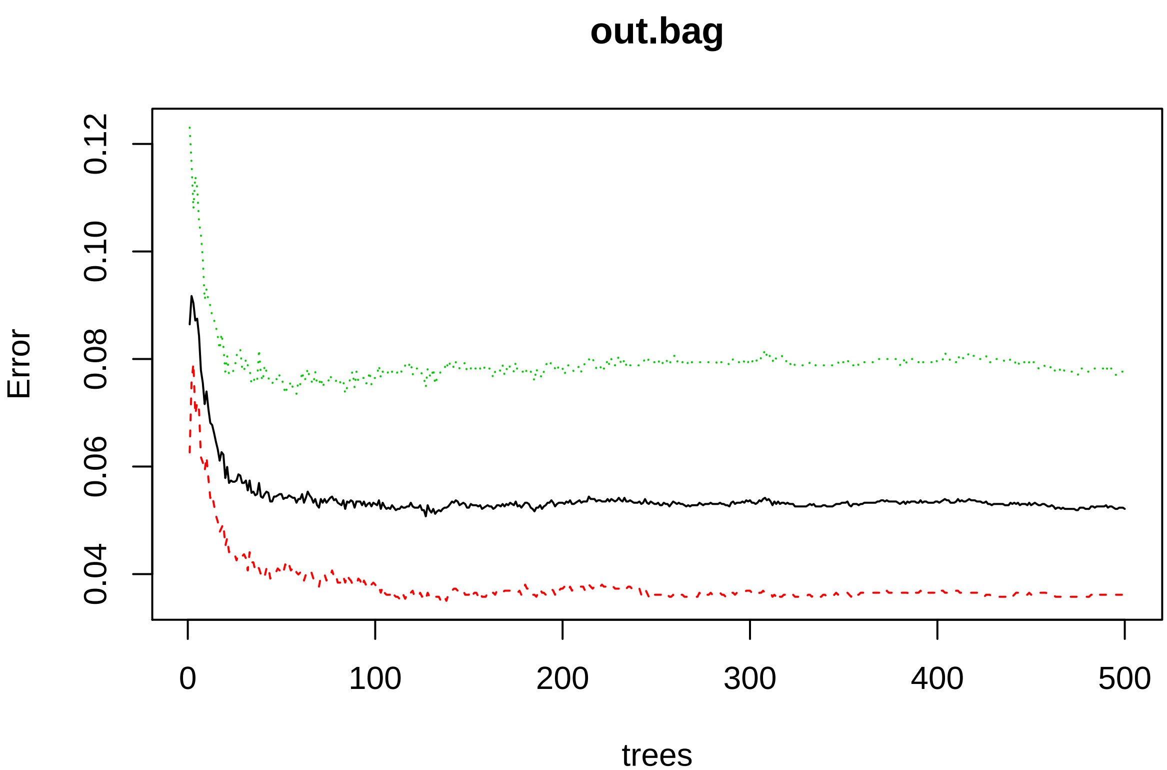

```r
plot(out.tree)
text(out.tree)
```



Question 5: Random Forest vs. Pruned Tree

Compare the test and training misclassification rates for the unpruned tree, the CV pruned tree, and random forest (do two different random forests, one with mtry set at the default and another set at mtry = p).

```r
#SOLUTION
out.bag = randomForest(X, Y, mtry = ncol(X))
plot(out.bag)
```




```r
class.bag = predict(out.bag,X_0,type='class')
```

```r
misClass(class.rf,Y_0)
```

```
## [1] "miss-class"
## [1] 0.04471545
## [1] "confusion mat"
##           true.class
## pred.class   0   1
##          0 130   8
##          1   3 105
```

```r
misClass(class.bag,Y_0)
```

```
## [1] "miss-class"
## [1] 0.05691057
## [1] "confusion mat"
##           true.class
## pred.class   0   1
##          0 129  10
##          1   4 103
```

```r
misClass(class.tree,Y_0)
```

```
## [1] "miss-class"
## [1] 0.07317073
## [1] "confusion mat"
##           true.class
## pred.class   0   1
##          0 127  12
##          1   6 101
```

```r
#How to get training misclass?
class.bag.train = predict(out.bag,X,type='class')
class.rf.train  = predict(out.rf,X,type='class')
misClass(class.rf.train,Y)
```

```
## [1] "miss-class"
## [1] 0.004592423
## [1] "confusion mat"
##           true.class
## pred.class    0    1
##          0 2654   19
##          1    1 1681
```

```r
misClass(class.bag.train,Y)
```

```
## [1] "miss-class"
## [1] 0.0006888634
## [1] "confusion mat"
##           true.class
## pred.class    0    1
##          0 2653    1
##          1    2 1699
```