STAT 542 / CS 598: Homework 4

Fall 2019, by Paul Nel (paulnel2)

Due: Monday, Oct 14 by 11:59 PM Pacific Time

Contents

Question 1 [70 Points] Tuning Random Forests in Virtual Twins

Set up of data sets

```
set.seed(11)
require(randomForest)
mydata = read.csv('Sepsis.csv')
n = nrow(mydata)
train.id = sample(1:n, round(n*0.75,0))
train.data = mydata[train.id,c(-1,-15)]
test.data = mydata[-train.id,c(-1,-15)]
```

I have removed the first predictor, `X`, which appears to be only an index as well as the last, `BEST` from the training and test sets.

Prediction and comparison functions

```
get_prediction <- function(therapy, mtry, nodesize){
  model.data = train.data[train.data[,"THERAPY"] == therapy,][,-2]
  rforest = randomForest(model.data$Health ~ ., data=model.data, mtry=mtry,
                         nodesize=nodesize)
  return (predict(rforest, test.data[,-2]))
}

get_treatment <- function(mtry, nodesize){
  pred.0 = get_prediction(0, mtry, nodesize)
  pred.1 = get_prediction(1, mtry, nodesize)
  return (pred.1>pred.0)
}

get_accuracy <- function(prediction, truth) {
  mean(prediction == truth)
}
```

Iteration loop

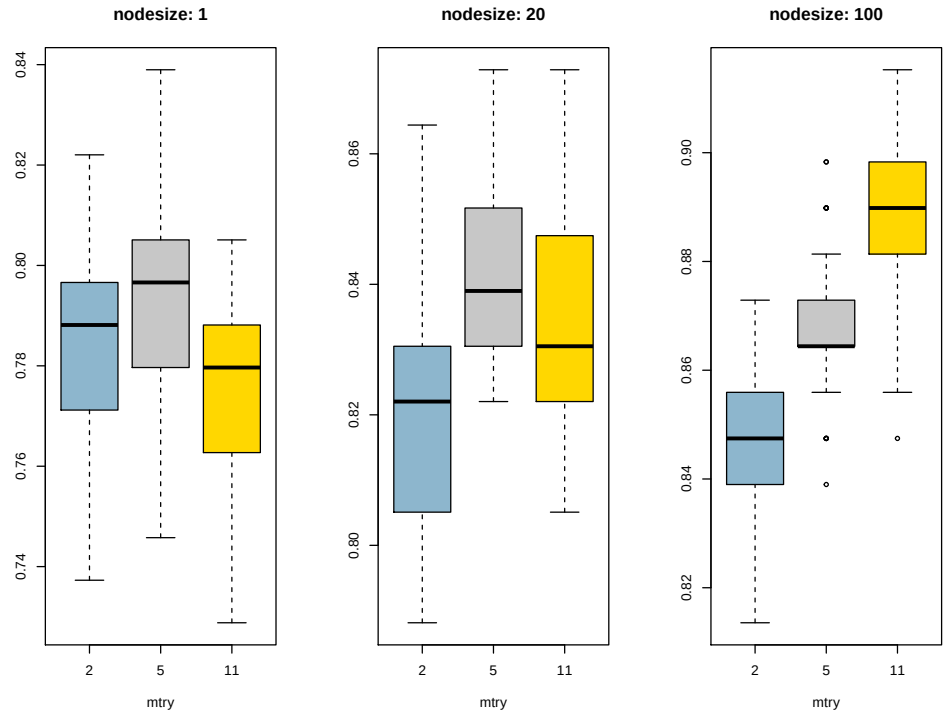
```
reps = 100
mtries = c(2,5,11)
nodesizes = c(1,20,100)
zeros = rep(0, length(mtries)*length(nodesizes)*reps)
accuracies =array(zeros, dim=c(reps,length(mtries),length(nodesizes)))
for (i in 1:reps){
```

```
  for (j in 1:length(mtries)){
    for (k in 1:length(nodesizes)){
      prediction = get_treatment(mtry=mtries[j], nodesize=nodesizes[k])
      accuracies[i,k,j] = get_accuracy(prediction, mydata[-train.id, "BEST"])
    }
  }
}
```

This leads to the following results with rows representing different `nodesize` and the coloumns different `mtry` values.

Table 1: Model performance for combinations of mtry and nodesize

	2	5	11
1	0.7843220	0.7935593	0.7775424
20	0.8183898	0.8426271	0.8350847
100	0.8485593	0.8684746	0.8894915



Intuition

Normally when one has highly correlated data, it helps to reduce the number of predictors (`mtry`) available for selection at each split. When all predictors are used (11 in this case if we ignore `THERAPY`) this process is similar to simple bagging. The result of the above analysis would suggest that the predictors are largely uncorrelated since the accuracy improves from 2 to 5 for all values for `nodesize` and also from 5 to 11 for `nodesize=100`.

The parameter `nodesize` determines the minimum size of nodes, in other words it determines the depth of each tree. For small values we can expect very deep trees. In this example I vary this from a minimum of `1` (largest possible tree) to `100`. This model also seems to be very sensitive to changes in , `mtry` with

a maximum accuracy achieved for `mtry = 11` and `nodesize = 100`, which will be considered the optimal values.

Question 2 [30 Points] Second Step in Virtual Twins

Generate predictions on all data based on the best parameters from Question 1.

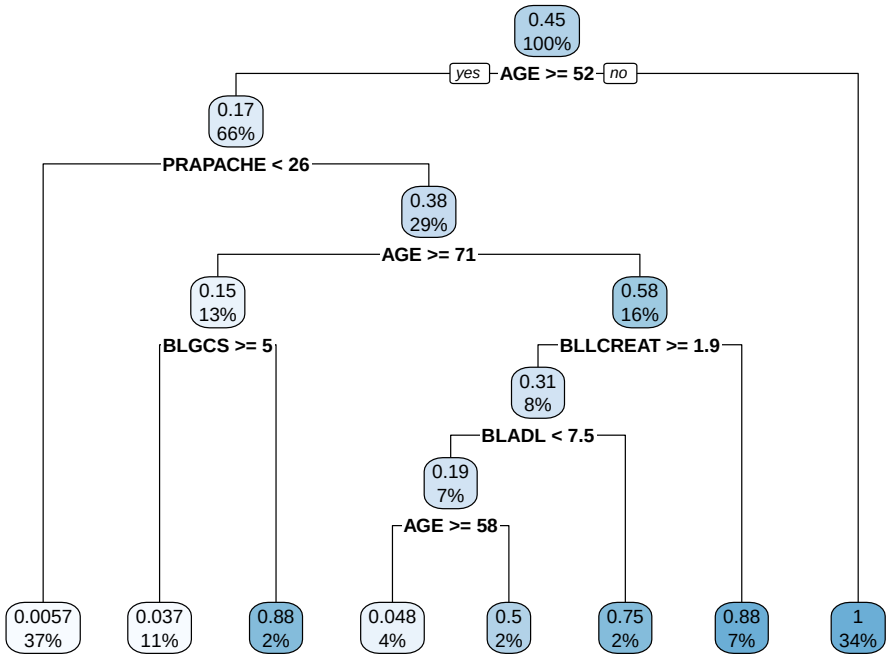
Helper function to choose treatment

```
choose_treatment <- function(pred.1, pred.0){
  return (ifelse(pred.1>pred.0, 1,0))
}
```

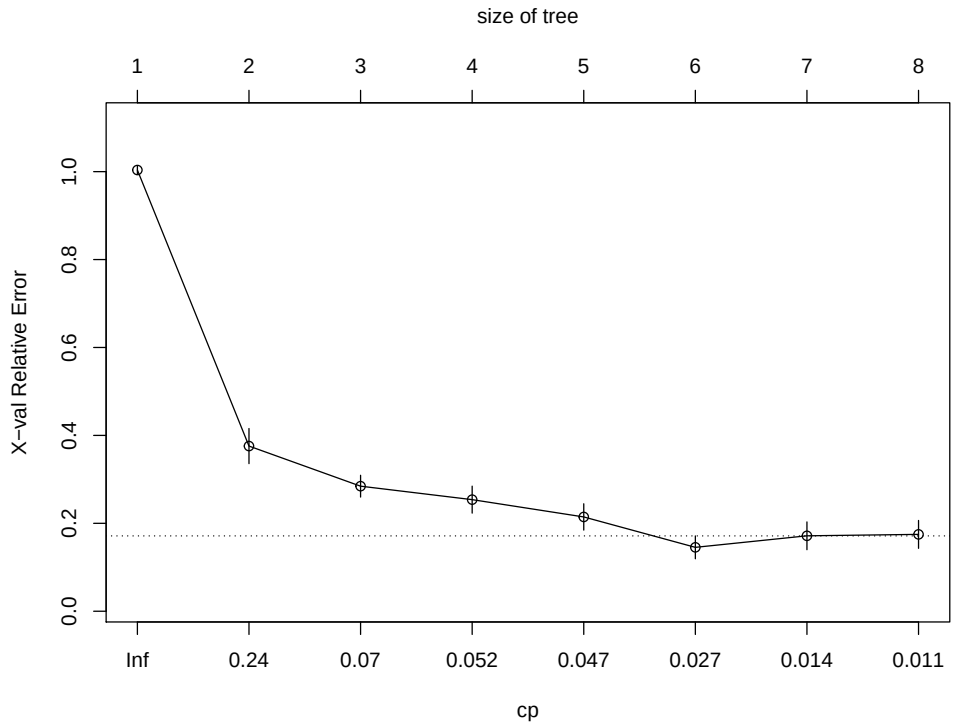
```
set.seed(124)
all_data = mydata[,c(-1,-15)]
model.data.0 = all_data[all_data["THERAPY"] == 0,][,c(-2,-14)]
model.data.1 = all_data[all_data["THERAPY"] == 1,][,c(-2,-14)]
rforest.0 = randomForest(model.data.0$Health ~ ., data=model.data.0, mtry=11, nodesize=100)
rforest.1 = randomForest(model.data.1$Health ~ ., data=model.data.1, mtry=11, nodesize=100)
pred.0 = predict(rforest.0, all_data)
pred.1 = predict(rforest.1, all_data)
all_data['PRED'] = choose_treatment(pred.1, pred.0)
#mean(all_data$PRED==mydata$BEST)
```

Build a single regression tree using the predicted values `all_data['PRED]` as the proposed treatment.

```
par(mfrow=c(1,2))
require(rpart)
require(rpart.plot)
train.all_data = all_data[,c(-1,-2)]
rtree = rpart(train.all_data$PRED~., data=train.all_data)
par(mfrow=c(1,1))
rpart.plot(rtree)
```




```
plotcp(rtree)
```



This gives us a tree with the following CP values:

```
##
## Regression tree:
```

```
## rpart(formula = train.all_data$PRED ~ ., data = train.all_data)
##
## Variables actually used in tree construction:
## [1] AGE      BLADL    BLGCS    BLLCREAT PRAPACHE
##
## Root node error: 116.37/470 = 0.24761
##
## n= 470
##
##         CP nsplit rel error  xerror      xstd
## 1 0.628119      0   1.00000 1.00367 0.0091916
## 2 0.090654      1   0.37188 0.37572 0.0399637
## 3 0.053312      2   0.28123 0.28451 0.0247947
## 4 0.051545      3   0.22791 0.25387 0.0306802
## 5 0.042042      4   0.17637 0.21451 0.0299692
## 6 0.016919      5   0.13433 0.14535 0.0260048
## 7 0.011913      6   0.11741 0.17156 0.0316150
## 8 0.010000      7   0.10550 0.17483 0.0317182
```

To prune the tree, we find the amount of splits that minimise the cross validation error `xerror` as follows:

```
opt = which.min(rtree$cptable[, "xerror"])
rtree$cptable[opt, 4]
```

```
## [1] 0.1453465
```

Find the upper limit based on 1 standard deviation and list all splits that have `xerror` less than this.

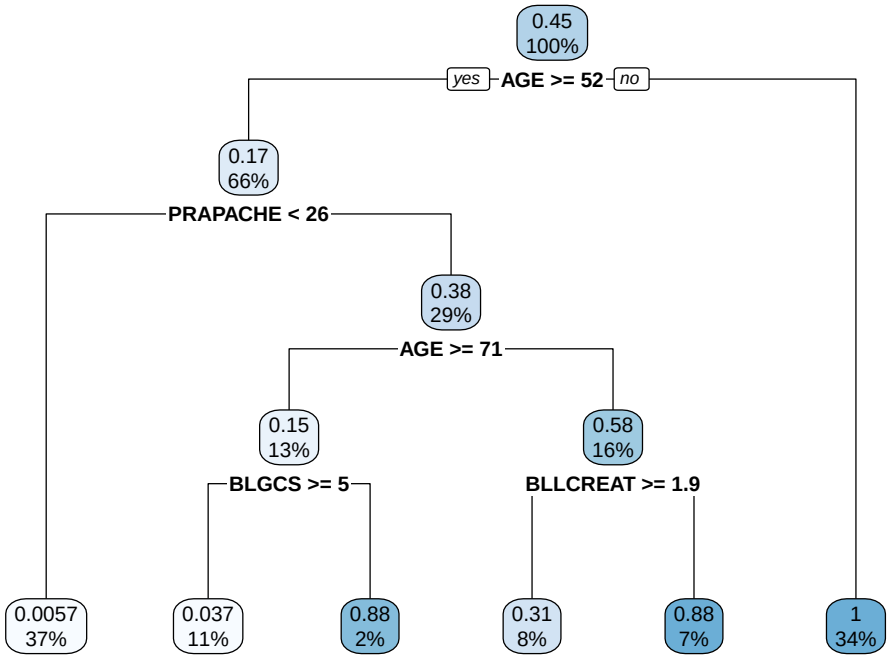
```
upper_1se = rtree$cptable[opt, 4] + rtree$cptable[opt, 5]
tmp.id = which(rtree$cptable[, 4] <= upper_1se)
```

Compute an average `CP` value between the two splits with the lowest resulting `xerror` of `0.171` which shoudl fall between rows `5` and `6`:

```
CP.1se = 0.5*(rtree$cptable[min(tmp.id[1]),1] + rtree$cptable[min(tmp.id[1])-1,1])
final.rtree = prune(rtree, cp=CP.1se)
```

which gives the following tree based on a CP_{1se} of `0.029`.

```
par(mfrow=c(1,1))
rpart.plot(final.rtree)
```


0.45
100%
yes
AGE >= 52
no
0.17
66%
PRAPACHE < 26
0.38
29%
AGE >= 71
0.15
13%
0.58
16%
BLGCS >= 5
BLLCREAT >= 1.9
0.0057
37%
0.037
11%
0.88
2%
0.31
8%
0.88
7%
1
34%