CART

CART 1: Oct. 18 Introduce to CART

- library for CART: rpart

```
library(rpart)
library(rpart.plot)
```

```
## Warning: package 'rpart.plot' was built under R version 3.6.1
```

Conduct CART tree: classification

Introduce how to construct the tree, read the result and plot the tree.

Use the standard iris data set, we are going to find the speices (classification problem)

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
y=iris[,5]
x=iris[,-5]
```

we use the definied x and y to build a tree

```
tree=rpart(y~.,data=x)
tree
```

```
## n= 150
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 150 100 setosa (0.33333333 0.33333333 0.33333333)
##   2) Petal.Length< 2.45 50   0 setosa (1.00000000 0.00000000 0.00000000) *
##   3) Petal.Length>=2.45 100  50 versicolor (0.00000000 0.50000000 0.50000000)
##     6) Petal.Width< 1.75 54   5 versicolor (0.00000000 0.90740741 0.09259259) *
##     7) Petal.Width>=1.75 46   1 virginica (0.00000000 0.02173913 0.97826087) *
```

Interpret tree result

Classification:

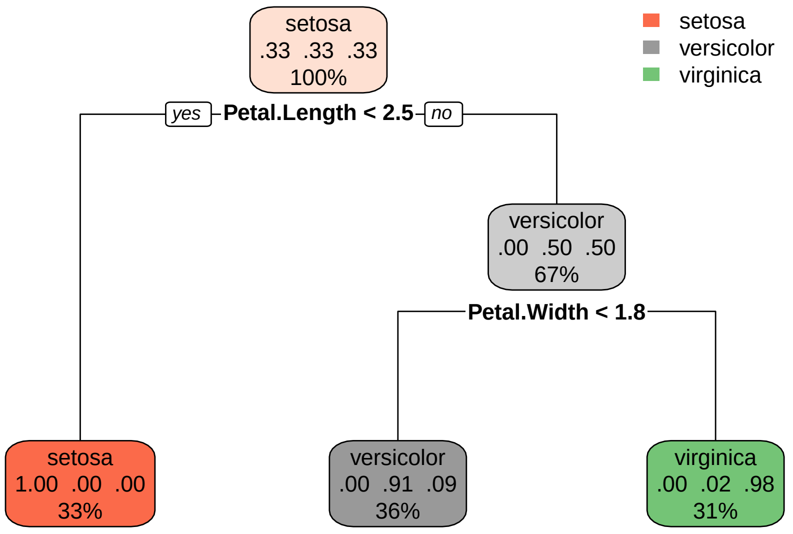
how can we read the result? (in classification case)

- Total 150 observations
- the second part of the result composed of

    1. t_i: t_i split into t_{2i} & t_{2i+1} (here we can assume it as a full tree, so that we can really see the location of the node or leaf)
    2. Quetions (or Root)
    3. nb of observations in this node
    4. nb of wrong predicted observation in each node
    5. Estimated result for this node (for the root we see from which percentage is higher, be careful if there's a pre-set known percentage for each class)
    6. percentage matrix (not always the num/total)

Now we try to plot the tree: We see that it's clearly not a full tree, due to the control minsplit=20 (stoping criteria when node has less then 20 observation it stop).

```r
#help(rpart)
#plot(tree)
#text(tree)
rpart.plot(tree)
```



Regression example:

in regression there's no nb0f wrong value, because it's often all are not exactly right

```r
tree=rpart(dist~speed,data=cars)
tree
```

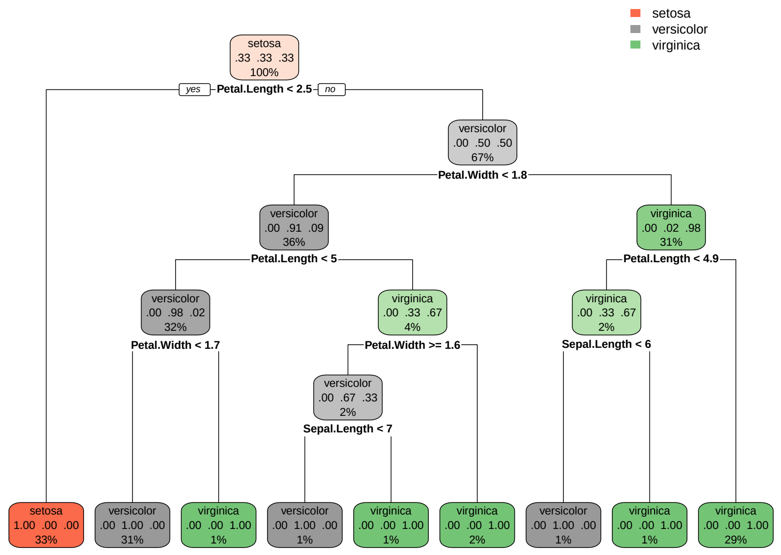
```
## n= 50
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
## 1) root 50 32538.980 42.98000
##   2) speed< 17.5 31  8306.774 29.32258
##     4) speed< 12.5 15  1176.400 18.20000 *
##     5) speed>=12.5 16  3535.000 39.75000 *
##   3) speed>=17.5 19  9015.684 65.26316 *
```

CART step 1: Build maximal tree

```r
#help(rpart)
tree=rpart(y~.,data=x,control = rpart.control(minsplit = 1,cp=10^-9))
tree
```

```
## n= 150
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 150 100 setosa (0.33333333 0.33333333 0.33333333)
##    2) Petal.Length< 2.45 50   0 setosa (1.00000000 0.00000000 0.00000000) *
##    3) Petal.Length>=2.45 100  50 versicolor (0.00000000 0.50000000 0.50000000)
##      6) Petal.Width< 1.75 54   5 versicolor (0.00000000 0.90740741 0.09259259)
##       12) Petal.Length< 4.95 48   1 versicolor (0.00000000 0.97916667 0.02083333)
##         24) Petal.Width< 1.65 47   0 versicolor (0.00000000 1.00000000 0.00000000) *
##         25) Petal.Width>=1.65 1   0 virginica (0.00000000 0.00000000 1.00000000) *
##       13) Petal.Length>=4.95 6   2 virginica (0.00000000 0.33333333 0.66666667)
##         26) Petal.Width>=1.55 3   1 versicolor (0.00000000 0.66666667 0.33333333)
##           52) Sepal.Length< 6.95 2   0 versicolor (0.00000000 1.00000000 0.00000000) *
##           53) Sepal.Length>=6.95 1   0 virginica (0.00000000 0.00000000 1.00000000) *
##         27) Petal.Width< 1.55 3   0 virginica (0.00000000 0.00000000 1.00000000) *
##      7) Petal.Width>=1.75 46   1 virginica (0.00000000 0.02173913 0.97826087)
##       14) Petal.Length< 4.85 3   1 virginica (0.00000000 0.33333333 0.66666667)
##         28) Sepal.Length< 5.95 1   0 versicolor (0.00000000 1.00000000 0.00000000) *
##         29) Sepal.Length>=5.95 2   0 virginica (0.00000000 0.00000000 1.00000000) *
##       15) Petal.Length>=4.85 43   0 virginica (0.00000000 0.00000000 1.00000000) *
```

```r
rpart.plot(tree)#structure of the tree
```


setosa
versicolor
virginica
setosa .33 .33 .33 100%
yes
Petal.Length < 2.5
no
versicolor .00 .50 .50 67%
Petal.Width < 1.8
versicolor .00 .91 .09 36%
Petal.Length < 5
virginica .00 .02 .98 31%
Petal.Length < 4.9
versicolor .00 .98 .02 32%
Petal.Width < 1.7
virginica .00 .33 .67 4%
Petal.Width >= 1.6
virginica .00 .33 .67 2%
Sepal.Length < 6
versicolor .00 .67 .33 2%
Sepal.Length < 7
setosa 1.00 .00 .00 33%
versicolor .00 1.00 .00 31%
virginica .00 .00 1.00 1%
versicolor .00 1.00 .00 1%
virginica .00 .00 1.00 1%
virginica .00 .00 1.00 2%
versicolor .00 1.00 .00 1%
virginica .00 .00 1.00 1%
virginica .00 .00 1.00 29%

- note that the class of the node is not max(count) when there's a prior percentage, it is actually max(matrix percentage), see below example

```
# dataset: kyphosis with prior percentage c(.65,.35)
k=kyphosis[,1]
table(k)
```

```
## k
##  absent present
##      64      17
```

```
table(k)/81 #although here the percentage is (.8,.2)
```

```
## k
##    absent   present
## 0.7901235 0.2098765
```

```
fit <- rpart(Kyphosis ~ Age + Number + Start, data = kyphosis)
fit # this is without prior tree
```

```
## n= 81
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 81 17 absent (0.79012346 0.20987654)
##    2) Start>=8.5 62  6 absent (0.90322581 0.09677419)
##      4) Start>=14.5 29  0 absent (1.00000000 0.00000000) *
##      5) Start< 14.5 33  6 absent (0.81818182 0.18181818)
##       10) Age< 55 12  0 absent (1.00000000 0.00000000) *
##       11) Age>=55 21  6 absent (0.71428571 0.28571429)
##         22) Age>=111 14  2 absent (0.85714286 0.14285714) *
##         23) Age< 111 7  3 present (0.42857143 0.57142857) *
##    3) Start< 8.5 19  8 present (0.42105263 0.57894737) *
```

```
fit2 <- rpart(Kyphosis ~ Age + Number + Start, data = kyphosis,
              parms = list(prior = c(.65,.35), split = "information"))
fit2 # with prior percentage
```

```
## n= 81
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 81 28.350000 absent (0.65000000 0.35000000)
##   2) Start>=12.5 46  3.335294 absent (0.91563089 0.08436911) *
##   3) Start< 12.5 35 16.453120 present (0.39676840 0.60323160)
##     6) Age< 34.5 10  1.667647 absent (0.81616742 0.18383258) *
##     7) Age>=34.5 25  9.049219 present (0.27932897 0.72067103) *
```

CART 2: Oct. 19 continue in CART

Prediction and accuracy from training and testing data

we first split the learning and testing set (not necessary in CART, there's already cross validatino in next step, just for representation), tree2 is the maximal tree while tree1 is not

```
u=sample(1:150,120)
learning=iris[u,]
test=iris[-u,]
tree2=rpart(Species~.,data=learning,control=rpart.control(minsplit=2,cp=0))
tree1=rpart(Species~.,data=learning)
```

now we make the prediction, in classification case if we don't specify `type='class'` the result would be the probability of each class. While in regression case by default it's the prediction of y already.

```
head(predict(tree1))
```

```
##     setosa versicolor  virginica
## 59       0 0.97560976 0.02439024
## 113      0 0.09756098 0.90243902
## 47       1 0.00000000 0.00000000
## 131      0 0.09756098 0.90243902
## 67       0 0.97560976 0.02439024
## 128      0 0.09756098 0.90243902
```

```
head(predict(tree1,type='class'))
```

```
##         59        113         47        131         67        128
## versicolor  virginica     setosa  virginica versicolor  virginica
## Levels: setosa versicolor virginica
```

how to check the accuracy? Here we see that even the maximal tree is still not perfectly for prediction (might over fit the training data), so how do we choose the best one? we need to have CART step 2 prunning and step 3 model selection.

- for tree 1

```r
head(predict(tree1, newdata=test[,-5],type='class')) #with new data set to testx
```

```
##      2      3      6      8     13     19 
## setosa setosa setosa setosa setosa setosa 
## Levels: setosa versicolor virginica
```

```r
true_y=test[,5]
yp=predict(tree1, newdata=test[,-5],type='class')
#true_y==yp # we can check which prediction is not the same from true value
#error
sum(true_y!=yp)/length(true_y)
```

```
## [1] 0.06666667
```

```r
#miss classigication error: for classification
```

- for tree 2

```r
head(predict(tree2, newdata=test[,-5],type='class')) #with new data set to testx
```

```
##      2      3      6      8     13     19 
## setosa setosa setosa setosa setosa setosa 
## Levels: setosa versicolor virginica
```

```r
true_y=test[,5]
yp=predict(tree2, newdata=test[,-5],type='class')
#true_y==yp # we can check which prediction is not the same from true value
#error
sum(true_y!=yp)/length(true_y)
```

```
## [1] 0.06666667
```

```r
#miss classigication error: for classification
```

Summary(tree): cp table

we can access computed `splits` when we construct a tree

```r
names(tree2)# we can see what is computed in the tree2, let's access splits
```

```
##  [1] "frame"               "where"               "call"
##  [4] "terms"               "cptable"             "method"
##  [7] "parms"               "control"             "functions"
## [10] "numresp"             "splits"              "variable.importance"
## [13] "y"                   "ordered"
```

```
#head(tree[[11]])
```

the cp table from the summary, access using `printcp` we have * construct tree code * variable used * Root node error * number of obervation

in the cp table, this is not all the possible subtrees, this is for the T_{final} (T_{max} if cp=0) a sequence of subtrees that is interesting for us (**a sequence of nested subtree**). That is why the result is not all the possible subtrees and that if we change final cp value, we might get very different result.

cp here is the penalize criterion $Crit_\alpha(T) = f(T) + \alpha\frac{|\widetilde{T}|}{n}$ with $\alpha \geq 0$ and $\widetilde{T}$ number of leaf

- $f(T)$: goodness of fit (avg square error for regression, avg missclassification error for classification)
- $\alpha\frac{|\widetilde{T}|}{n}$: complexity
- rel error:real error on the training sample, but the root value is not always one, so its computed by error/root node error to make root always 1.
- xerror, xstd: cross validation error and std.

note that here the cross validation is the reason why we don't need to split train and test set for CART, also this is the only randomess happen in CART. Due to this randomess, the final tree we would choose might not be the same everytime.

```
#summary(tree2) # we just look at the cp table first
printcp(tree2)
```

```
##
## Classification tree:
## rpart(formula = Species ~ ., data = learning, control = rpart.control(minsplit = 2,
##     cp = 0))
##
## Variables actually used in tree construction:
## [1] Petal.Length Petal.Width  Sepal.Length
##
## Root node error: 76/120 = 0.63333
##
## n= 120
##
##          CP nsplit rel error  xerror     xstd
## 1 0.5000000      0  1.000000 1.00000 0.069459
## 2 0.4342105      1  0.500000 0.48684 0.066563
## 3 0.0131579      2  0.065789 0.13158 0.039838
## 4 0.0065789      6  0.013158 0.10526 0.035954
## 5 0.0000000      8  0.000000 0.11842 0.037965
```

```r
#summary(tree1)
printcp(tree1)
```

```
##
## Classification tree:
## rpart(formula = Species ~ ., data = learning)
##
## Variables actually used in tree construction:
## [1] Petal.Length
##
## Root node error: 76/120 = 0.63333
##
## n= 120
##
##        CP nsplit rel error  xerror     xstd
## 1 0.50000      0  1.000000 1.07895 0.067049
## 2 0.43421      1  0.500000 0.57895 0.069459
## 3 0.01000      2  0.065789 0.13158 0.039838
```

step 2: Prunning

With CP table we are able to construct step 2 prunning.

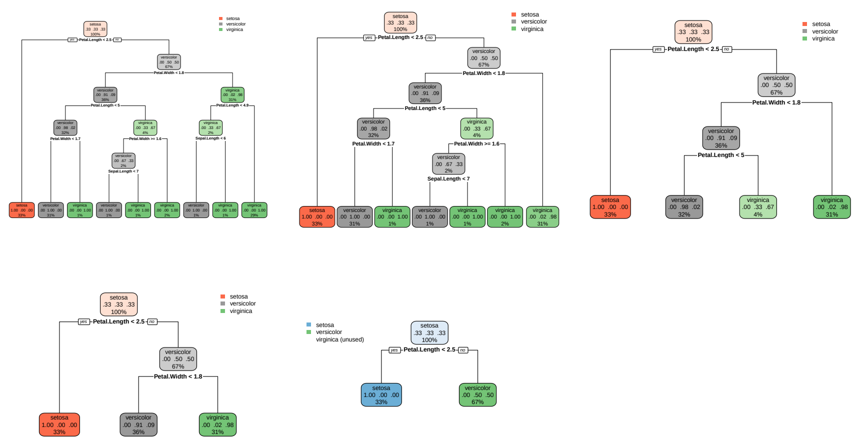
```r
Tree=rpart(Species~.,data=iris,control=rpart.control(minsplit=2,cp=0))
Treep=prune(Tree,cp=0.02)
A=printcp(Tree)
```

```
##
## Classification tree:
## rpart(formula = Species ~ ., data = iris, control = rpart.control(minsplit = 2,
##     cp = 0))
##
## Variables actually used in tree construction:
## [1] Petal.Length Petal.Width  Sepal.Length
##
## Root node error: 100/150 = 0.66667
##
## n= 150
##
##      CP nsplit rel error xerror     xstd
## 1 0.500      0      1.00   1.15 0.051801
## 2 0.440      1      0.50   0.65 0.060690
## 3 0.020      2      0.06   0.11 0.031927
## 4 0.010      3      0.04   0.09 0.029086
## 5 0.005      6      0.01   0.08 0.027520
## 6 0.000      8      0.00   0.09 0.029086
```

```r
cp=A[,1] #retrieve the cp value
```

with this we can plot all the subtree, note that R cannot plot the root

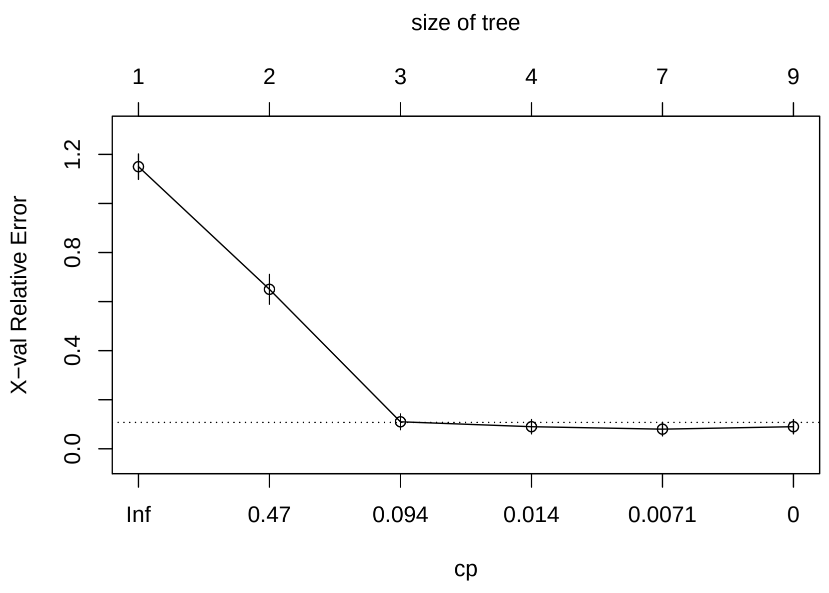
```r
par(mfrow=c(2,3))
for (k in 1:length(cp)){
  a=cp[length(cp)-k+1]
  if (length(cp)-k+1>1){
    T=prune(Tree,cp=a)
    rpart.plot(T)
  }

}
```



select best cp for punning: 1-SE rule.

choose smallest xerror and add its xstd as the threshold choose the biggest cp below this threshold

```r
plotcp(Tree)
```



```r
printcp(Tree)
```

```
##
## Classification tree:
## rpart(formula = Species ~ ., data = iris, control = rpart.control(minsplit = 2,
##     cp = 0))
##
## Variables actually used in tree construction:
## [1] Petal.Length Petal.Width  Sepal.Length
##
## Root node error: 100/150 = 0.66667
##
## n= 150
##
##      CP nsplit rel error xerror     xstd
## 1 0.500      0      1.00   1.15 0.051801
## 2 0.440      1      0.50   0.65 0.060690
## 3 0.020      2      0.06   0.11 0.031927
## 4 0.010      3      0.04   0.09 0.029086
## 5 0.005      6      0.01   0.08 0.027520
## 6 0.000      8      0.00   0.09 0.029086
```

CART step 3: model selection (2 possibilities)

1. With 1-SE rule

Construct the final tree with the 1-SE rule chosen above and build the model with the whole dataset, then we don't need to split the data to training and testing, useful when we don't have much observations.

2. Test error

- random split dataset to training and validation
- construct maximal tree with training set
- every subtree we use the validation set to compute the error
- choose the smallest error to be the final result

CART unstablility solution: bagging

Idea: we have a training dataset of m observations, we would bootstrap sample set (k) that each sample have a size of n (normally n=m), but we do like `sample(1:m,m,replace = TRUE)` that is we might have duplicated observations. for each sample set we do a CART to get a final tree $T_1, T_2, ...T_k$ and then we aggregate those k models.

Aggregate:

- Regression: average predicted $\hat{Y}$ with equation $\hat{Y_i} = \frac{1}{k}\sum_{i=1}^{k} \hat{Y_{i,T_k}}$
- Classification: most often seen predicted $\hat{Y}$

in theory k is as big as possible, and that if we take k almost 500 there we can see the stabilization

```
bag_procedure<- function(dataXl,dataYl,dataXt,dataYt,K){
  #X1 Y1 learning sample
  P=matrix(0,ncol=length(dataYt),nrow=K) #k number of sample
  nl=nrow(dataXl) #nb of observations in the learning
  u=1:nl
  for (i in 1:K){
    a=sample(u,replace=TRUE)
    Xl=dataXl[a,] #bootstrap datasets
    Yl=dataYl[a]
    tree=rpart(Yl~.,data=Xl)
    P[i,]=predict(tree, newdata=dataXt, type='class')
  }
  bag=P
}

u=sample(1:150,10)
#create learning and test
dataXt=iris[u,1:4]
dataYt=iris[u,5]
dataXl=iris[-u,1:4]
dataYl=iris[-u,5]
A=bag_procedure(dataXl,dataYl,dataXt,dataYt,5)
A
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    2    1    2    1    1    3    1    2    3     3
## [2,]    2    1    2    1    1    3    1    2    3     3
## [3,]    2    1    2    1    1    3    1    2    3     3
## [4,]    2    1    2    1    1    3    1    2    3     3
## [5,]    2    1    2    1    1    3    1    2    3     3
```

Bagging exercise:

data mtcars in R, y =mpg 32 observations

1. bagging (from myself)

take random 10 observations for the test sample evaluate the error on the test sample with bagging and CART and plot the error according to different k (to 500)

define bag function for predict y

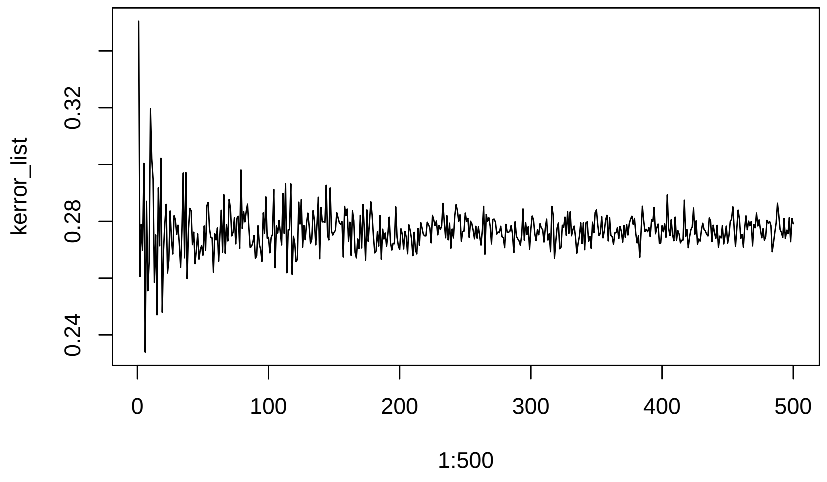
```
bag_avg_error<- function(dataXl,dataYl,dataXt,dataYt,K){
  #X1 Y1 learning sample
  P=matrix(0,ncol=length(dataYt),nrow=K) #k number of sample
  nl=nrow(dataXl) #nb of observations in the learning

  u=1:nl
  for (i in 1:K){
    a=sample(u,replace=TRUE)
    Xl=dataXl[a,] #bootstrap datasets
    Yl=dataYl[a]
    tree=rpart(Yl~.,data=Xl)
    P[i,]=predict(tree, newdata=dataXt)
  }
  #get average Y for each observation
  c=c()
  for (i in 1:length(dataYt)){
    c[i]=mean(P[,i])
  }
  #now we can compute the error
  bag_avg_error=mean((c-dataYt)**2)
}
```

let's now try the function to put error in the list for k 1 to 500

```
u=sample(1:32,10)
#create learning and test
dataXt=mtcars[u,1:4]
dataYt=mtcars[u,5]
dataXl=mtcars[-u,1:4]
dataYl=mtcars[-u,5]
kerror_list=c()
for (k in 1:500){
  kerror_list[k]=bag_avg_error(dataXl,dataYl,dataXt,dataYt,k)
}
```

plot the error list

```
plot(1:500,kerror_list,type ="l")
```



2. determine CART tree to this dataset and compare the result

```
head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

step 1: max tree and get the ideal cp

```
x=mtcars[,-1]
y=mtcars[,1]
#step 1: create maximal tree
maxtree=rpart(y~., data=x,control=rpart.control(minsplit=2,cp=10^(-9)))
#plot(maxtree)
#text(maxtree)
#printcp(maxtree)
A=maxtree$cptable
cverr=A[,4]
mincverr=which(cverr==min(cverr)) #it counld be one value of more as a list
s=A[mincverr,4]+A[mincverr,5] #set a threshould
s=min(s)
B=1*(cverr<=s)
a=min(which(B==1)) # we get the cp value index
cp=A[a,1]
```

step 2: prunnning, and model selection

```
final_tree=prune(maxtree,cp=cp)
final_tree
```

```
## n= 32
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 32 1126.047000 20.09062
##    2) wt>=2.26 26  346.566500 17.78846
##      4) cyl>=7 14   85.200000 15.10000
##        8) disp>=450 2    0.000000 10.40000 *
##        9) disp< 450 12   33.656670 15.88333 *
##      5) cyl< 7 12   42.122500 20.92500
##       10) wt>=3.3275 3    1.086667 18.36667 *
##       11) wt< 3.3275 9   14.855560 21.77778 *
##    3) wt< 2.26 6   44.553330 30.06667
##      6) qsec< 19.185 4   14.907500 28.52500 *
##      7) qsec>=19.185 2    1.125000 33.15000 *
```

#CART 3: oct.21

summary(tree): below cp table

continue from last exercise question 2 look at the summary of the table.

After the printcp table,we have the variable importance, each nodes primary splits and surrogate splits. In practice we only see the first primary split and use the information of the surrogate splits, since other primary splits only consider the error, while surrogate split make sure that the data splitting is similar to the best primary split. (see more in the course note)

In primary splits we only take the first one, what is the other one? see below example:

```
Primary splits:
  wt   < 3.3275 to the right, improve=0.6215272, (0 missing)
  cyl  < 5      to the right, improve=0.5573591, (0 missing)
Surrogate splits:
  disp < 163.8  to the right, agree=0.917, adj=0.667, (0 split)
```

- cyl < 5 we find the best split for the node without taking into account the variable wt, but it's totally wrong, we cannot really replace it. It was just the first idea
- t hat's why we need surrogate splits:disp<163.8 : we take the best split for t4 without taking into account wt, which do quite the same than the best split with wt (agree = 93%, similar split as wt)
- interpret the surrogate split question: `cyl < 5 to the right` the question is actually cyl>=5, cuz always true to the left, wrong to the right

```r
summary(final_tree)
```

```
## Call:
## rpart(formula = y ~ ., data = x, control = rpart.control(minsplit = 2,
##     cp = 10^(-9)))
##   n= 32
##
##           CP nsplit  rel error    xerror       xstd
## 1 0.65266121      0 1.00000000 1.0538395 0.24658893
## 2 0.19470235      1 0.34733879 0.6291235 0.15850344
## 3 0.04577369      2 0.15263644 0.4004718 0.09849726
## 4 0.02532828      3 0.10686275 0.3347179 0.09216838
## 5 0.02324972      4 0.08153448 0.2918381 0.09158653
## 6 0.01248838      5 0.05828476 0.2610732 0.07780450
##
## Variable importance
##   wt disp   hp drat  cyl qsec   vs
##   26   25   19   11    9    6    4
##
## Node number 1: 32 observations,    complexity param=0.6526612
##   mean=20.09062, MSE=35.18897
##   left son=2 (26 obs) right son=3 (6 obs)
##   Primary splits:
##       wt   < 2.26   to the right, improve=0.6526612, (0 missing)
##       cyl  < 5      to the right, improve=0.6431252, (0 missing)
##       disp < 163.8  to the right, improve=0.6130502, (0 missing)
##       hp   < 118    to the right, improve=0.6010712, (0 missing)
##       vs   < 0.5    to the left,  improve=0.4409477, (0 missing)
##   Surrogate splits:
##       disp < 101.55 to the right, agree=0.969, adj=0.833, (0 split)
##       hp   < 92     to the right, agree=0.938, adj=0.667, (0 split)
##       drat < 4      to the left,  agree=0.906, adj=0.500, (0 split)
##       cyl  < 5      to the right, agree=0.844, adj=0.167, (0 split)
##
## Node number 2: 26 observations,    complexity param=0.1947024
##   mean=17.78846, MSE=13.32948
##   left son=4 (14 obs) right son=5 (12 obs)
##   Primary splits:
##       cyl  < 7      to the right, improve=0.6326174, (0 missing)
##       disp < 266.9  to the right, improve=0.6326174, (0 missing)
##       hp   < 136.5  to the right, improve=0.5803554, (0 missing)
##       wt   < 3.325  to the right, improve=0.5393370, (0 missing)
##       qsec < 18.15  to the left,  improve=0.4210605, (0 missing)
##   Surrogate splits:
##       disp < 266.9  to the right, agree=1.000, adj=1.000, (0 split)
##       hp   < 136.5  to the right, agree=0.962, adj=0.917, (0 split)
##       wt   < 3.49   to the right, agree=0.885, adj=0.750, (0 split)
##       qsec < 18.15  to the left,  agree=0.885, adj=0.750, (0 split)
##       vs   < 0.5    to the left,  agree=0.885, adj=0.750, (0 split)
##
## Node number 3: 6 observations,    complexity param=0.02532828
##   mean=30.06667, MSE=7.425556
##   left son=6 (4 obs) right son=7 (2 obs)
##   Primary splits:
##       qsec < 19.185 to the left,  improve=0.6401504, (0 missing)
##       disp < 78.85  to the right, improve=0.6322011, (0 missing)
##       vs   < 0.5    to the left,  improve=0.4454287, (0 missing)
##       wt   < 1.885  to the right, improve=0.3030076, (0 missing)
##       hp   < 65.5   to the right, improve=0.2922527, (0 missing)
##   Surrogate splits:
##       disp < 78.85  to the right, agree=0.833, adj=0.5, (0 split)
##       carb < 1.5    to the right, agree=0.833, adj=0.5, (0 split)
##
## Node number 4: 14 observations,    complexity param=0.04577369
##   mean=15.1, MSE=6.085714
##   left son=8 (2 obs) right son=9 (12 obs)
##   Primary splits:
##       disp < 450    to the right, improve=0.6049687, (0 missing)
##       wt   < 4.66   to the right, improve=0.4782188, (0 missing)
##       hp   < 192.5  to the right, improve=0.4669349, (0 missing)
##       carb < 3.5    to the right, improve=0.4669349, (0 missing)
##       qsec < 17.71  to the right, improve=0.4306658, (0 missing)
##   Surrogate splits:
##       drat < 3.035  to the left,  agree=0.929, adj=0.5, (0 split)
##       wt   < 4.66   to the right, agree=0.929, adj=0.5, (0 split)
##       qsec < 17.71  to the right, agree=0.929, adj=0.5, (0 split)
##
## Node number 5: 12 observations,    complexity param=0.02324972
##   mean=20.925, MSE=3.510208
##   left son=10 (3 obs) right son=11 (9 obs)
##   Primary splits:
##       wt   < 3.3275 to the right, improve=0.6215272, (0 missing)
##       cyl  < 5      to the right, improve=0.5573591, (0 missing)
##       hp   < 96     to the right, improve=0.5507811, (0 missing)
##       disp < 163.8  to the right, improve=0.4615111, (0 missing)
##       carb < 3      to the right, improve=0.2857431, (0 missing)
##   Surrogate splits:
##       disp < 163.8  to the right, agree=0.917, adj=0.667, (0 split)
##       hp   < 116.5  to the right, agree=0.833, adj=0.333, (0 split)
##
## Node number 6: 4 observations
##   mean=28.525, MSE=3.726875
##
## Node number 7: 2 observations
##   mean=33.15, MSE=0.5625
##
## Node number 8: 2 observations
##   mean=10.4, MSE=0
##
## Node number 9: 12 observations
##   mean=15.88333, MSE=2.804722
##
## Node number 10: 3 observations
##   mean=18.36667, MSE=0.3622222
##
## Node number 11: 9 observations
##   mean=21.77778, MSE=1.650617
```

Another usage of surrogate split

previously we see that surrogate split can perform variable selection in the course note (13,14). There's another usage which is dealing with missing explanatory.

In this dataset there's missing value for x and y (Ozone). We should first suppress the missing y.

```r
library(datasets)
data=airquality
#help(airquality)
head(data) #'there's missing data
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    NA      NA 14.3   56     5   5
## 6    28      NA 14.9   66     5   6
```

```r
ozon=data[,1]
u=which(is.na(ozon)==TRUE)
datab=data[-u,]#suppress the missing value on y
head(datab)
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 6    28      NA 14.9   66     5   6
## 7    23     299  8.6   65     5   7
```

Now create test and training dataset. CART if in one node the spliting x for that observation is missing then for this observation it would be split by the surrogate split.

```
test=datab[1:7,]
train=datab[-(1:7),]
#create tree, don't check max, only until enough to explain
tree=rpart(train[,1]~.,data=train[,-1],control=rpart.control(minsplit=2,cp=0))

#take test 6
test
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 6    28      NA 14.9   66     5   6
## 7    23     299  8.6   65     5   7
## 8    19      99 13.8   59     5   8
```

```
#summary(tree)
#see node 272 there's one missing value in the best primary split (so we cannot use this)
#assume that observation 96 in datab is in node 272
predict(tree, newdata=test) # on test
```

```
##  1  2  3  4  6  7  8
##  7  7 16 32 11 34 32
```

```
predict(tree)# on train
```

```
##   9  11  12  13  14  15  16  17  18  19  20  21  22  23  24  28  29  30
##   8   7  16  11  14  18  14  34   6  30  11   1  11   4  32  23  45 115
##  31  38  40  41  44  47  48  49  50  51  62  63  64  66  67  68  69  70
##  37  29  71  39  23  21  37  20  12  13 135  49  32  64  40  77  97  97
##  71  73  74  76  77  78  79  80  81  82  85  86  87  88  89  90  91  92
##  85  10  27   7  48  35  61  79  63  16  80 108  20  52  82  50  64  59
##  93  94  95  96  97  98  99 100 101 104 105 106 108 109 110 111 112 113
##  39   9  16  78  35  66 122  89 110  44  28  65  22  59  23  31  44  21
## 114 116 117 118 120 121 122 123 124 125 126 127 128 129 130 131 132 133
##   9  45 168  73  76 118  84  85  96  78  73  91  47  32  20  23  21  24
## 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 151 152
##  44  21  28   9  13  46  18  13  24  16  13  23  36   7  14  30  14  18
## 153
##  20
```

```
#that's why even there's missing value it still can split them
```

what we should not do in practice:

(okay) 1. we create datab by supression the observations i such that yi=NA

(okay) 2. we constructe a tree with datab, denoted T

should not go to step 3, we can but we should not use it as true value

(still can do it) 3. we predict the prediction associated to observation such that yi=NA

but should not do more that we cannot replace predicted of missing yi to replace by the true yi. by doing this you are forcing your model to be the true model! So that in practice, we should just remove those observation and not use it.

Conclude with compare RF and R

Random Forest can deal with high dimension dataset while CART cannot, RF dosen't have surrogate split that is it cannot deal with missing value. The biggest problem of RF is the visualization of the final estimator. In practice it's often to combine them:

1. Perform RF
2. Variable selection thanks to RF (VSURF)
3. Applu CART algorithm only on the subset selected

Both CART and RF has their way to help variable selection. CART is due to the surrogate split the software is able to compute the variable importance. RF is by using `VSURF`