Exercises: Tree-based classification methods

In this exercise, you will know

- How to use tree-based classification methods

Don't forget to change your working directory!

1 Heart data to predict whether a patient has heart disease

Load the `Heart.csv` data. These data contain a binary outcome HD for 303 patients who presented with chest pain. An outcome value of `Yes` indicates the presence of heart disease based on an angiographic test, while No means no heart disease. There are 13 predictors including Age, Sex, Chol (a cholesterol measurement), and other heart and lung function measurements.

```
#install.packages("caret")
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```
heart=read.csv("Heart.csv",header = TRUE)
#check missing values
sum(is.na(heart))
```

```
## [1] 6
```

```
#remove missing values
heart=heart[complete.cases(heart),]
#create training/test split
set.seed(345)
train.index=createDataPartition(heart[,ncol(heart)],p=0.7,list=FALSE)
train=heart[train.index,]
test=heart[-train.index,]
```

2 Decision trees

2.1 Grow a tree

The `tree()` function is in the package `tree`. The package provides nice functions to display the graphs, use cross-validation to prune the tree.

```
#install.packages("tree")
library(tree)
####################################################################
#########Train a decision tree#####################################
heart.tree=tree(AHD ~ . , train)
#have a look at the summary of the tree
summary(heart.tree)
```

```
##
## Classification tree:
```

```
## tree(formula = AHD ~ ., data = train)
## Variables actually used in tree construction:
##  [1] "Thal"      "Ca"        "RestBP"    "MaxHR"     "Age"
##  [6] "ChestPain" "Sex"       "Slope"     "RestECG"   "Oldpeak"
## Number of terminal nodes:  16
## Residual mean deviance:  0.4737 = 90.96 / 192
## Misclassification error rate: 0.101 = 21 / 208
```

In the summary, the residual mean deviance is calculated as $-\frac{2}{n-|T_0|}\sum_m\sum_k n_{mk}\log\hat{p}_{mk}$, where n_{mk} is the number of observations in the mth terminal node that belong to the kth class. A small deviance indicates a tree that provides a good fit to the (training) data.

To have a look at the details of how the tree is splitted, just type the name of the tree.

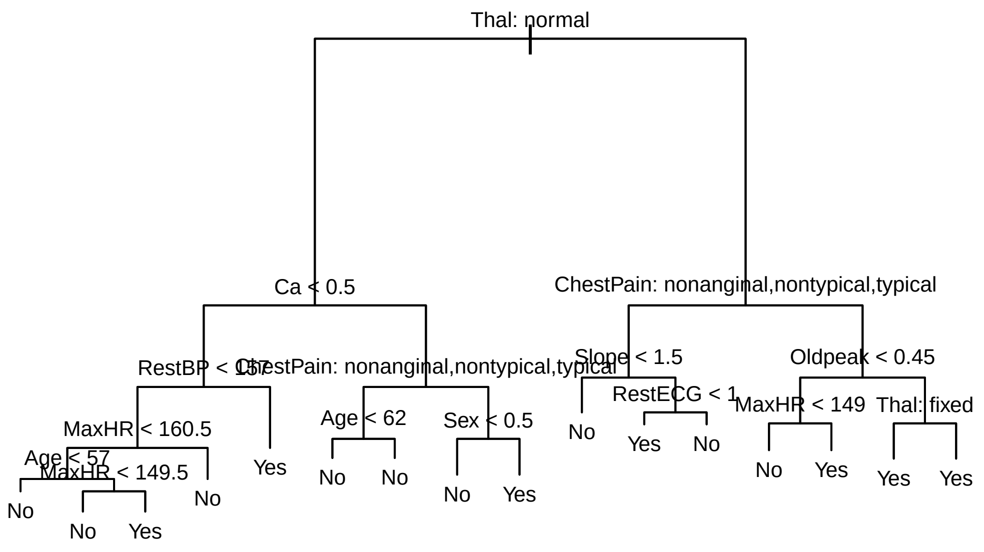
```
#have a look at how the tree is splitted
heart.tree
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 208 287.100 No ( 0.53846 0.46154 )
##    2) Thal: normal 117 124.000 No ( 0.77778 0.22222 )
##      4) Ca < 0.5 83  56.980 No ( 0.89157 0.10843 )
##        8) RestBP < 157 78  37.150 No ( 0.93590 0.06410 )
##         16) MaxHR < 160.5 38  29.590 No ( 0.86842 0.13158 )
##           32) Age < 57 21   8.041 No ( 0.95238 0.04762 ) *
##           33) Age > 57 17  18.550 No ( 0.76471 0.23529 )
##             66) MaxHR < 149.5 12   6.884 No ( 0.91667 0.08333 ) *
##             67) MaxHR > 149.5 5   6.730 Yes ( 0.40000 0.60000 ) *
##         17) MaxHR > 160.5 40   0.000 No ( 1.00000 0.00000 ) *
##        9) RestBP > 157 5   5.004 Yes ( 0.20000 0.80000 ) *
##      5) Ca > 0.5 34  47.130 Yes ( 0.50000 0.50000 )
##       10) ChestPain: nonanginal,nontypical,typical 18  19.070 No ( 0.77778 0.22222 )
##         20) Age < 62 11  14.420 No ( 0.63636 0.36364 ) *
##         21) Age > 62 7   0.000 No ( 1.00000 0.00000 ) *
##       11) ChestPain: asymptomatic 16  15.440 Yes ( 0.18750 0.81250 )
##         22) Sex < 0.5 5   6.730 No ( 0.60000 0.40000 ) *
##         23) Sex > 0.5 11   0.000 Yes ( 0.00000 1.00000 ) *
##    3) Thal: fixed,reversable 91  98.320 Yes ( 0.23077 0.76923 )
##      6) ChestPain: nonanginal,nontypical,typical 23  31.490 No ( 0.56522 0.43478 )
##       12) Slope < 1.5 6   0.000 No ( 1.00000 0.00000 ) *
##       13) Slope > 1.5 17  23.030 Yes ( 0.41176 0.58824 )
##         26) RestECG < 1 9   9.535 Yes ( 0.22222 0.77778 ) *
##         27) RestECG > 1 8  10.590 No ( 0.62500 0.37500 ) *
##      7) ChestPain: asymptomatic 68  49.260 Yes ( 0.11765 0.88235 )
##       14) Oldpeak < 0.45 16  21.170 Yes ( 0.37500 0.62500 )
##         28) MaxHR < 149 7   8.376 No ( 0.71429 0.28571 ) *
##         29) MaxHR > 149 9   6.279 Yes ( 0.11111 0.88889 ) *
##       15) Oldpeak > 0.45 52  16.950 Yes ( 0.03846 0.96154 )
##         30) Thal: fixed 7   8.376 Yes ( 0.28571 0.71429 ) *
##         31) Thal: reversable 45   0.000 Yes ( 0.00000 1.00000 ) *
```

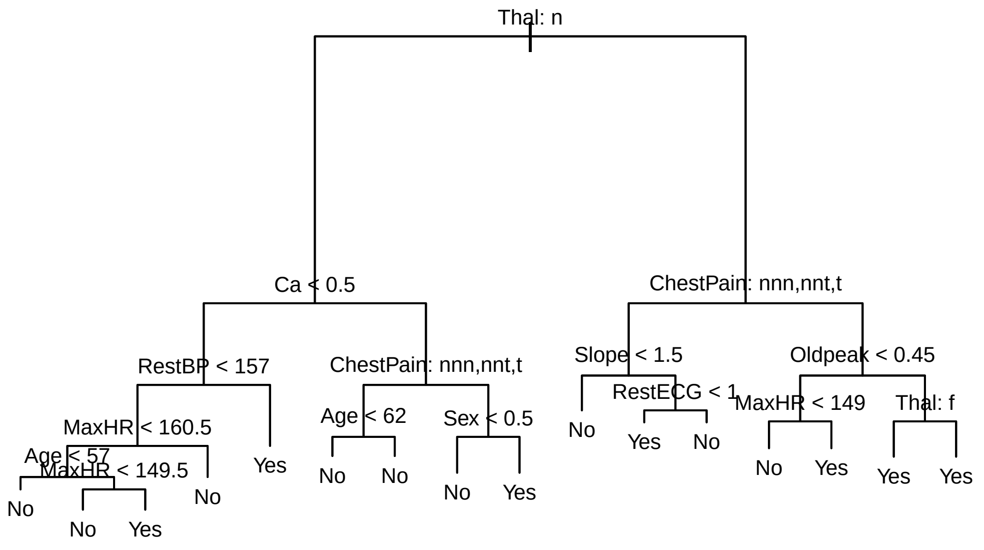
One important advantage of classification trees is that we can visualise the tree and see how the final decisions are made.

```
#plot the tree
plot(heart.tree)
```

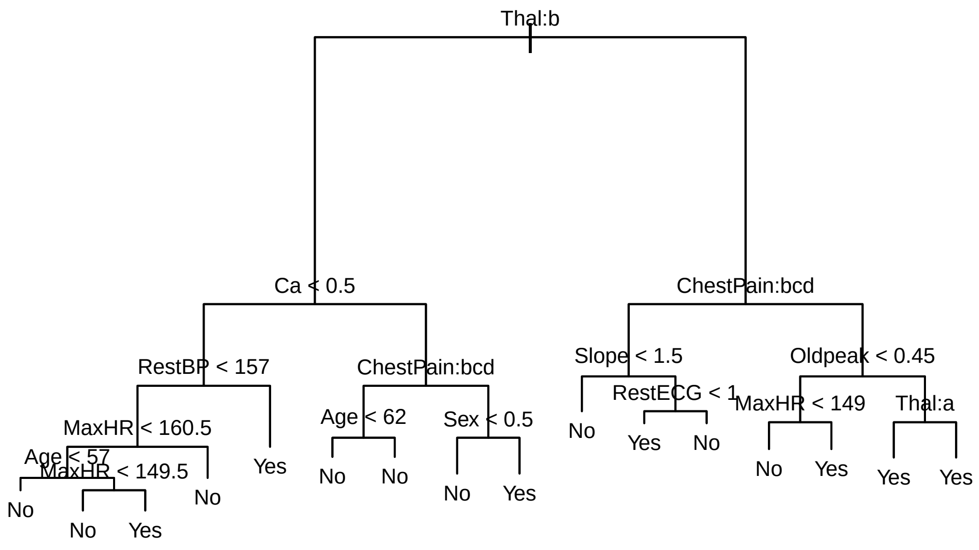
```
text(heart.tree,pretty=0,cex=0.7)
```



```
##try different parameters for pretty
##pretty=1
plot(heart.tree)
text(heart.tree,pretty=1,cex=0.7)
```



```
##try different parameters for pretty
##pretty=NULL
plot(heart.tree)
text(heart.tree,pretty=NULL,cex=0.7)
```



To predict the classes of the test set, we need to specify `type` as `class`.

```
##################################################################
#########Test error##########################################################
pred=predict(heart.tree,test[,-ncol(test)],type="class")
table(pred,test[,ncol(test)])
```

```
##
## pred  No Yes
##    No  42  15
##    Yes  6  26
```

```
mean(pred==test[,ncol(test)])
```

```
## [1] 0.7640449
```

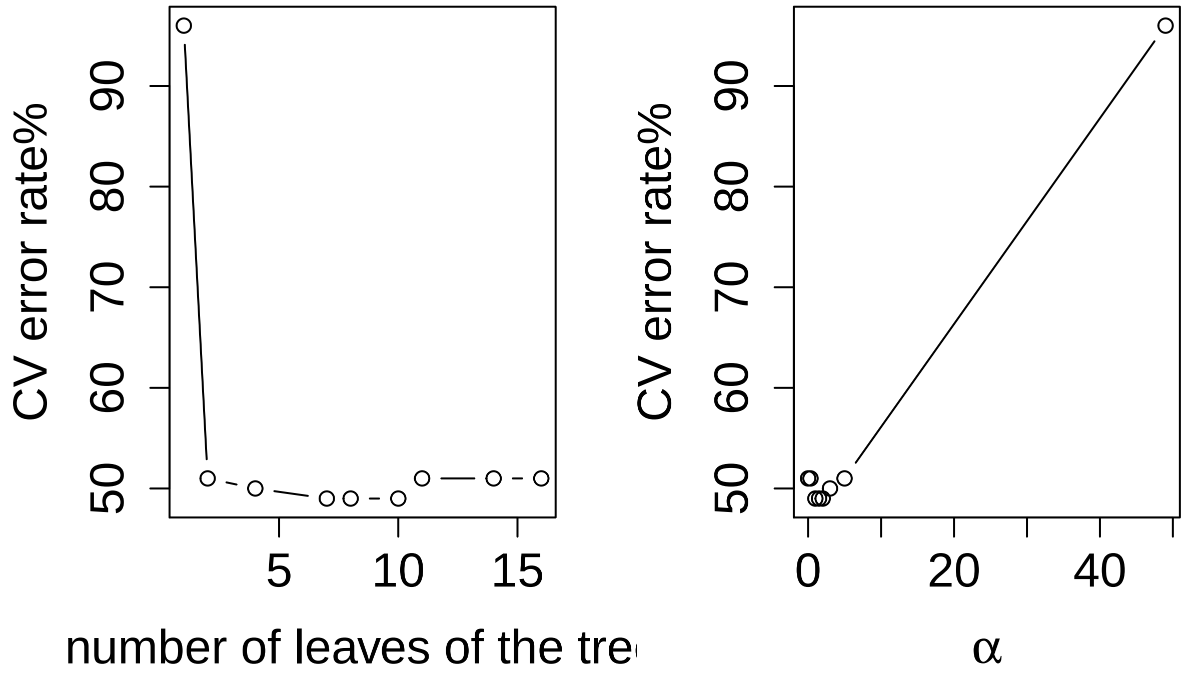
2.2 Prune the tree

To prune the tree, we first use `cv.tree` to get the best size of the tree. Here we set `FUN=prune.misclass` to indicate that we want the classification error rate to guide the cross-validation and pruning process, rather than the default for the `cv.tree()` function, which is deviance. Note that in this case, `dev` denotes the CV classification error rate and `k` is α.

```
set.seed(203)
heart.cv=cv.tree(heart.tree,FUN=prune.misclass)
heart.cv
```

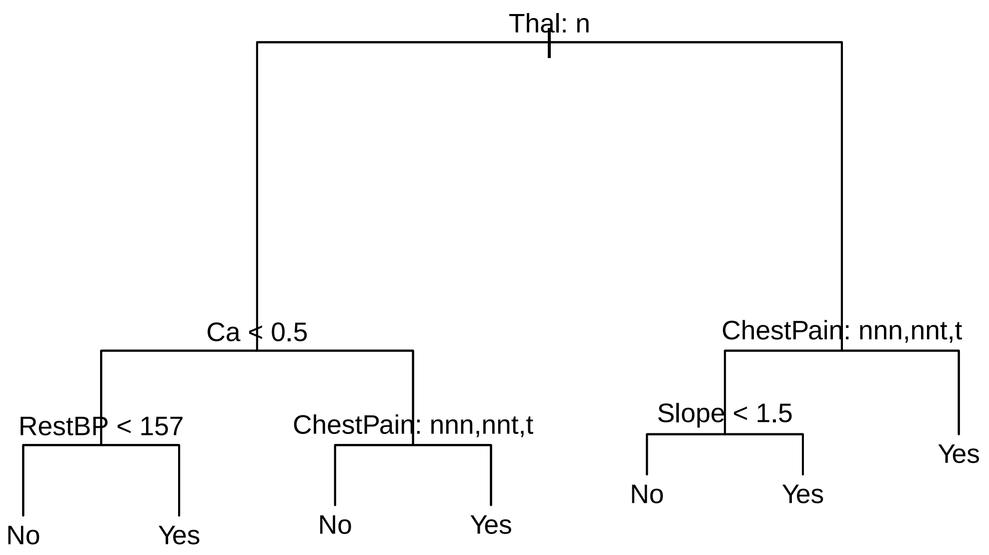
```
## $size
## [1] 16 14 11 10  8  7  4  2  1
##
## $dev
## [1] 51 51 51 49 49 49 50 51 96
##
## $k
## [1]       -Inf  0.0000000  0.3333333  1.0000000  1.5000000  2.0000000
## [7]  3.0000000  5.0000000 49.0000000
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```
#plot the cross-validation results
par(mfrow=c(1,2))
plot(heart.cv$size,heart.cv$dev,type="b",
     xlab="number of leaves of the tree",ylab="CV error rate%",
     cex.lab=1.5,cex.axis=1.5)
plot(heart.cv$k,heart.cv$dev,type="b",
     xlab=expression(alpha),ylab="CV error rate%",
     cex.lab=1.5,cex.axis=1.5)
```



From the outputs and the plots, we can prune the tree to the subtree with size 7.

```
#######prune the tree
heart.prune=prune.misclass(heart.tree,best=7)
plot(heart.prune)
text(heart.prune,pretty=1,cex=0.8)
```



Finally, we can obtain the prediction from the pruned tree.

```
#######predict the test instances
pred.prune=predict(heart.prune,test[,-ncol(test)],type="class")
table(pred.prune,test[,ncol(test)])
```

```
##
## pred.prune No Yes
##        No  40  10
##        Yes  8  31
```

```
mean(pred.prune==test[,ncol(test)])
```

```
## [1] 0.7977528
```

2.3 Use caret library to build decision trees

In `caret` library, we can use `rpart` library to build a decision tree. Here we repeate 10-fold CV 3 times to tune the parameter `cp`, i.e. complexity parameter, specified in the `rpart` library.

```
fitcontrol=trainControl(method = "repeatedcv",
                     number = 10,
                     repeats = 3)
set.seed(1)
heart.rpart=train(train[,-ncol(heart)],
               train[,ncol(heart)],
               method = "rpart",
               tuneLength=5,
               trControl = fitcontrol)
heart.rpart
```

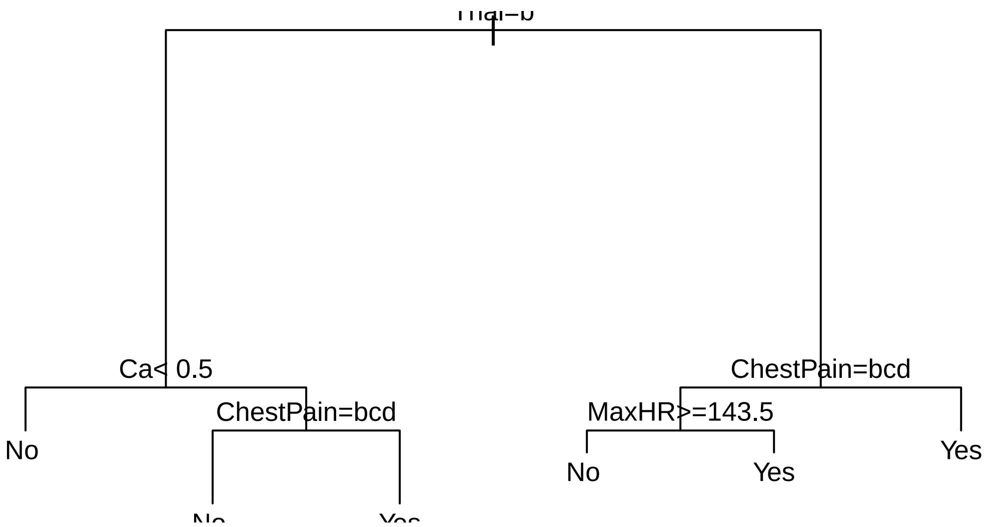
```
## CART
##
## 208 samples
##  13 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 187, 188, 188, 187, 187, 187, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.00000000  0.7650433  0.5296389
##   0.01041667  0.7715512  0.5405180
##   0.03125000  0.7633045  0.5227185
##   0.05208333  0.7566378  0.5096050
##   0.51041667  0.6268470  0.2178092
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.01041667.
```

We can have a look at the details of this tree by the following command.

```
#####To look at the details of this tree
print(heart.rpart$finalModel)
```

We can obtain the tree plot as before.

```
plot(heart.rpart$finalModel)
text(heart.rpart$finalModel, cex=.8)
```

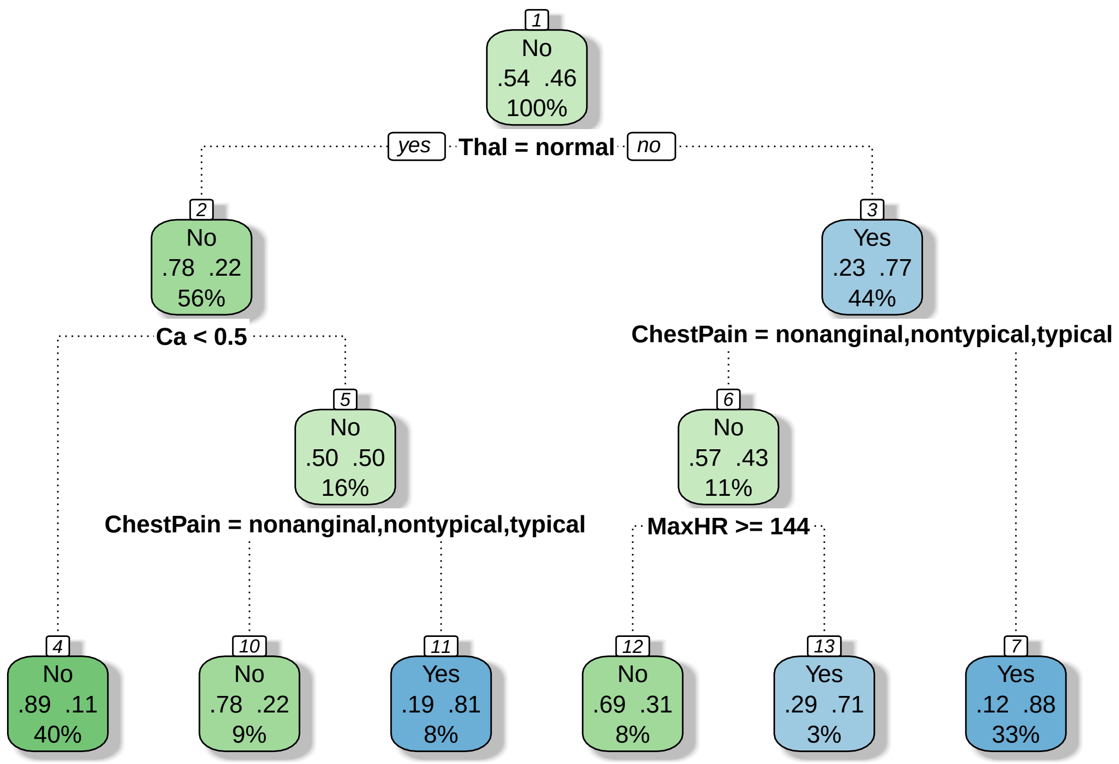


We can use `rattle` library to produce fancy trees. If you don't have this package, install it first and then library it.

```
library(rattle)
```

```
## Rattle: A free graphical interface for data science with R.
## Version 5.3.0 Copyright (c) 2006-2018 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
fancyRpartPlot(heart.rpart$finalModel)
```



Rattle 2022−Jan−17 20:52:36 MuMu

Instead of using the default parameters of `cp`, we can specify our own values by the following.

```
fitcontrol=trainControl(method = "repeatedcv",
                      number = 10,
                      repeats = 3)

set.seed(1)
cpGrid=expand.grid(cp=c(0.01,0.02,0.03))
heart.rparts=train(train[,-ncol(heart)],
               train[,ncol(heart)],
               method = "rpart",
               tuneGrid=cpGrid,
               trControl = fitcontrol)
heart.rparts
```

```
## CART
##
## 208 samples
##  13 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 187, 188, 188, 187, 187, 187, ...
## Resampling results across tuning parameters:
##
##   cp    Accuracy   Kappa
##   0.01  0.7715512  0.5405180
##   0.02  0.7762410  0.5468052
##   0.03  0.7633045  0.5227185
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.02.
```

3 Bagging and random forest

The `randomForest()` function is in the package `randomForest` can perform random forest. Remember that bagging is equivalent to random forest when we use all features in each split. Thus, in order to use bagging, we set `mtry=13`.

```
#install.packages("randomForest")
library(randomForest)
```

```
## randomForest 4.6-14

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

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:rattle':
##
##     importance

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
set.seed(103)
heart.bag=randomForest(AHD~.,data=train,mtry=13,importance=TRUE,ntree=500)
heart.bag
```

```
##
## Call:
##  randomForest(formula = AHD ~ ., data = train, mtry = 13, importance = TRUE,      ntree = 500)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 13
##
##         OOB estimate of  error rate: 19.71%
## Confusion matrix:
##     No Yes class.error
## No  92  20   0.1785714
## Yes 21  75   0.2187500
```

```
pred.bag=predict(heart.bag,newdata=test[,-ncol(test)])
mean(pred.bag==test[,ncol(test)])
```

```
## [1] 0.7865169
```

To use random forest, we need to set `mtry` to a smaller number.

```
set.seed(921)
heart.rf=randomForest(AHD~.,data=train,mtry=6,importance=TRUE,ntree=500)
heart.rf
```

```
##
```

```
## Call:
##  randomForest(formula = AHD ~ ., data = train, mtry = 6, importance = TRUE,      ntree = 500)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 20.67%
## Confusion matrix:
##     No Yes class.error
## No  94  18   0.1607143
## Yes 25  71   0.2604167
```

```
pred.rf=predict(heart.rf,newdata=test[,-ncol(test)])
mean(pred.rf==test[,ncol(test)])
```

```
## [1] 0.8202247
```

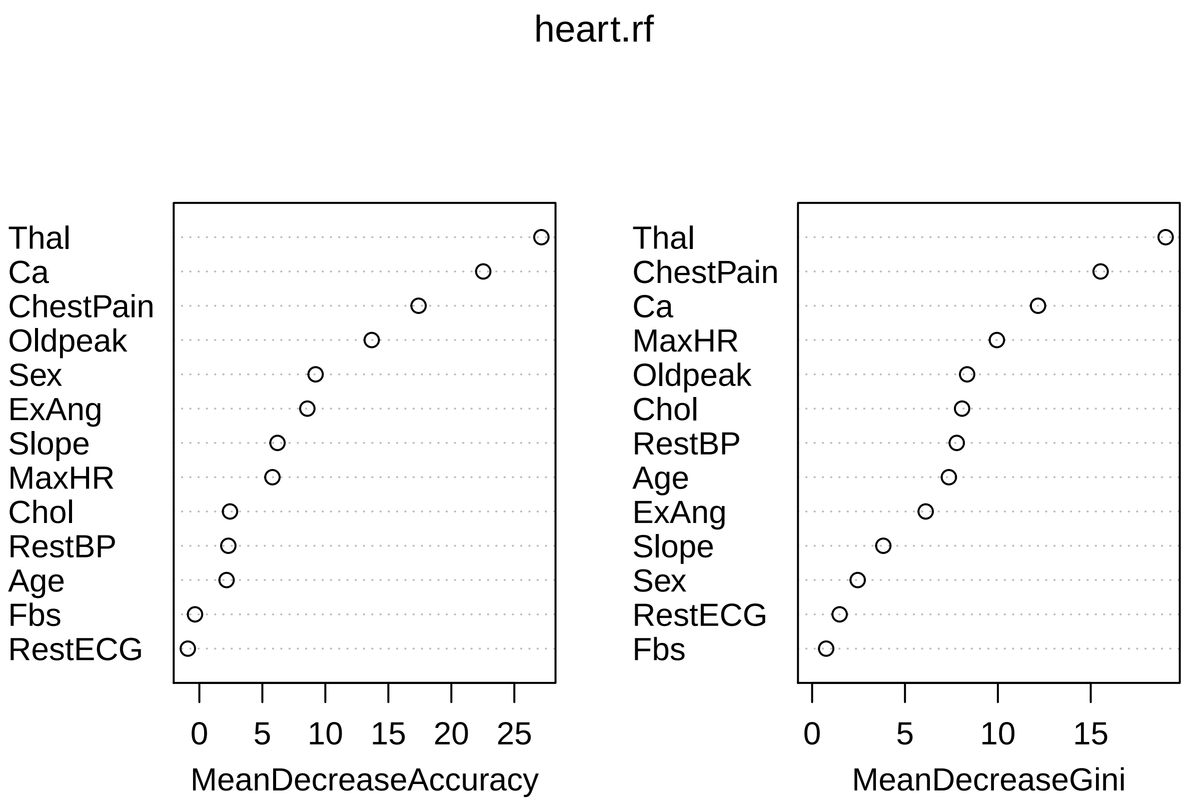
Change `mtry` to see its effect on the classification accuracy.

To view the importance of variables

```
importance(heart.rf)
```

```
##                  No        Yes MeanDecreaseAccuracy MeanDecreaseGini
## Age        2.441914  0.3476965            2.1632695         7.362045
## Sex        7.072532  6.5971326            9.2266184         2.455001
## ChestPain  7.701321 17.5506959           17.3996740        15.530698
## RestBP     2.733387  0.5130701            2.3008015         7.786261
## Chol       4.172631 -1.1331553            2.4325230         8.073093
## Fbs        1.036562 -1.4449975           -0.3452513         0.757226
## RestECG   -2.603338  1.5040638           -0.9149606         1.482132
## MaxHR      6.340872  1.7714783            5.8003275         9.947163
## ExAng      4.962371  6.7487807            8.5727718         6.113873
## Oldpeak   11.333329  8.7500849           13.6815565         8.344661
## Slope      4.308077  4.4151352            6.2018139         3.833562
## Ca        20.659808 13.9920101           22.5334214        12.161654
## Thal      21.732936 19.2986340           27.1464430        19.032381
```

```
varImpPlot(heart.rf)
```

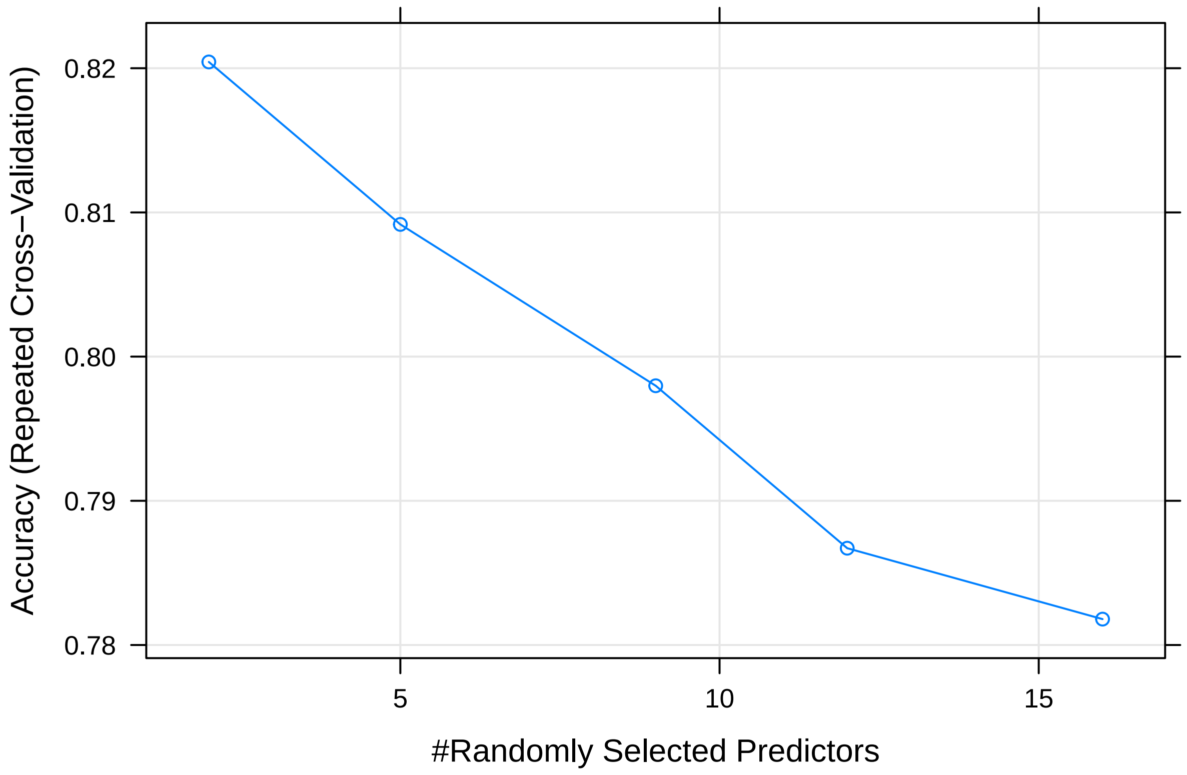


3.1 Use caret library for random forest

```
fitControl=trainControl(
  method = "repeatedcv",
  number = 5,
  repeats = 3)
set.seed(2)
rfFit=train(AHD~.,data=train,method="rf",metric="Accuracy",
            trControl=fitControl,tuneLength=5)
# Have a look at the model
rfFit
```

```
## Random Forest
##
## 208 samples
##  13 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 166, 167, 166, 167, 166, 166, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.8204341  0.6386703
##    5    0.8091682  0.6167541
##    9    0.7979778  0.5943423
##   12    0.7867119  0.5719138
##   16    0.7817951  0.5623955
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
plot(rfFit)
```



```
rfFit$finalModel
```

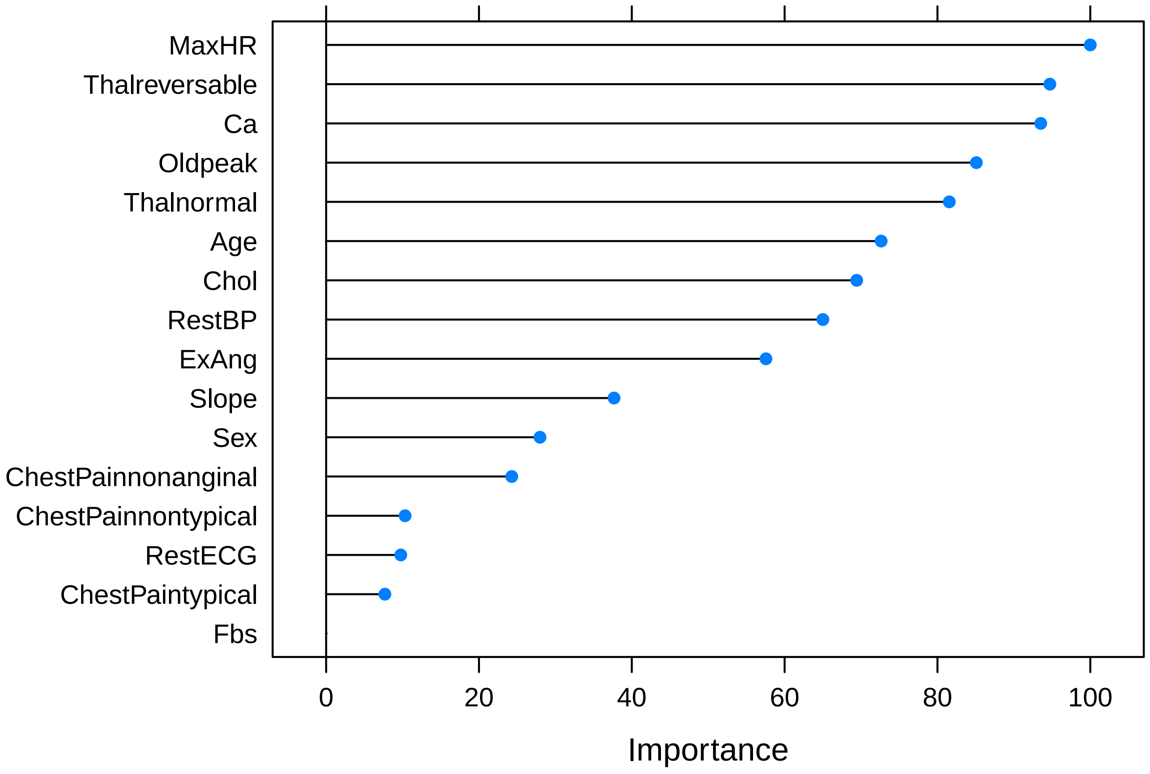
```
##
## Call:
##  randomForest(x = x, y = y, mtry = min(param$mtry, ncol(x)))
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 17.31%
## Confusion matrix:
##     No Yes class.error
## No  96  16   0.1428571
## Yes 20  76   0.2083333
```

Have a look at variable importance

```
varImp(rfFit)
```

```
## rf variable importance
##
##                      Overall
## MaxHR                100.000
## Thalreversable        94.712
## Ca                       93.515
## Oldpeak                  85.092
## Thalnormal               81.549
## Age                      72.632
## Chol                     69.433
## RestBP                   65.004
## ExAng                    57.557
## Slope                    37.677
## Sex                      27.989
## ChestPainnonanginal      24.292
## ChestPainnontypical      10.333
## RestECG                   9.767
## ChestPaintypical          7.694
## Fbs                       0.000
```

```
plot(varImp(rfFit))
```



4 Boosting

The `gbm()` function is in the package **gbm** can perform boosting. Before using this function, we have to transform the label variable to 0-1.

```
#install.packages("gbm")
library(gbm)
```

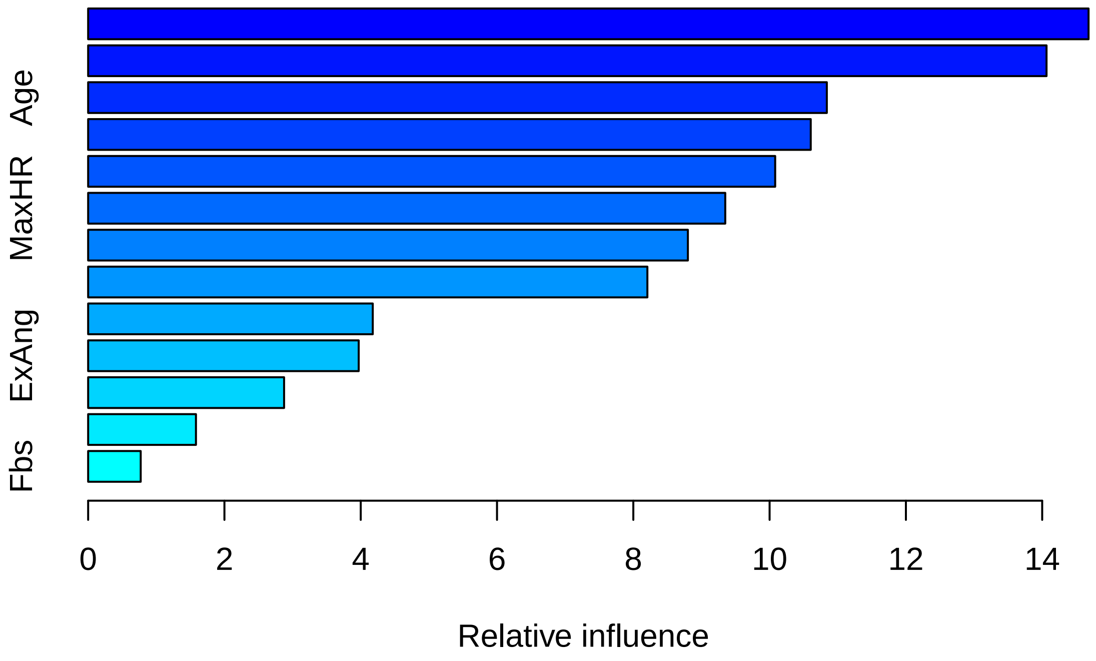
```
## Loaded gbm 2.1.5
```

```
set.seed(18)
train[,ncol(train)]=ifelse(train$AHD=="No",0,1)
```

```
test[,ncol(test)]=ifelse(test$AHD=="No",0,1)
```

Use `distribution="bernoulli"` for classification.

```
heart.boost=gbm(AHD~.,data=train,distribution="bernoulli",n.trees=5000,
                interaction.depth=2,shrinkage=0.01)
summary(heart.boost)
```



```
##                 var    rel.inf
## Thal           Thal 14.6803299
## ChestPain ChestPain 14.0635729
## Age             Age 10.8394947
## Ca               Ca 10.6033795
## Chol           Chol 10.0815991
## MaxHR         MaxHR  9.3491266
## Oldpeak     Oldpeak  8.8005816
## RestBP       RestBP  8.2063350
## Slope         Slope  4.1765056
## ExAng         ExAng  3.9704146
## Sex             Sex  2.8753579
## RestECG     RestECG  1.5820328
## Fbs             Fbs  0.7712698
```

```
pred.boost=predict(heart.boost,newdata=test[,-ncol(test)],n.trees=5000,type="response")
pred.boost=ifelse(pred.boost<0.5,0,1)
mean(pred.boost==test[,ncol(test)])
```

```
## [1] 0.8202247
```

Change `interaction.depth` and `shrinkage` to see their effect.