Example: An Implementation of the Classification Trees with R

```
> library(rpart)
>data(kyphosis, package="rpart")
>n<- length(kyphosis[,1])
>kyphosis.class<-rep(0, n)
>for(i in 1:n) {if(kyphosis[i,1]==”present”) kyphosis.class[i]<-1}
# Class: “present” -> 1, and “absent” -> 0

> kyphosis.all<-cbind(kyphosis[,2:4], kyphosis.class)
>kyphosis.train<-kyphosis.all[1:50,]
>kyphosis.test<-kyphosis.all[51:n,]
# Create the objects for the train and test data.

> result.Tree1<- rpart(kyphosis.class ~ ., method=”class", data=kyphosis.train)
> summary(result.Tree1)
# Grow the Classification Tree.

>plot(result.Tree1, uniform=TRUE, margin=0.1 )
>text(result.Tree1, use.n=TRUE, all=TRUE )
# Plot the Classification Tree.
```

~~#install.packages(“party”)~~
~~>library(party)~~
~~>result.Tree1.pa<-as.party(result.Tree1)~~
~~>plot (result.Tree1.pa)~~
~~# Plot the Classification Tree using the package “party”)~~

```
>plotcp(result.Tree1)
>result.Tree2<- prune(result.Tree1, cp=
result.Tree1$cptable[which.min(result.Tree1$cptable[,"xerror"]),"CP"])
# Prune the tree
# Note that the pruned tree is not provided in this example.

> result.Tree3<-predict(result.Tree1, kyphosis.test[-4], type="class")
> table(kyphosis.test[,4], result.Tree3)
 # Create the confusion matrix for the test data (the threshed =0.5)
```

```
> library(ROCR)
>result.Tree4<-predict(result.Tree1, kyphosis.test[-4], type="prob")
>fit.Tree.pred<- prediction(result.Tree4[,2], kyphosis.test[,4])
>fit.Tree.perf <- performance(fit.Tree.pred,"tpr","fpr")
> plot(fit.Tree.perf,lwd=2,col="blue", main="ROC: Classification Tree on Kyphosis")
>abline(a=0,b=1)
# Plot the ROC Curve by using the package ROCR

>auc.Tree.tmp <- performance(fit.Tree.pred, "auc")
> auc.Tree <- as.numeric(auc.Tree.tmp@y.values)
>auc.Tree
# Obtain the AUC by using the package ROCR
```

See the web site: http://www.statmethods.net/advstats/cart.html and below:

Tree-Based Models

Recursive partitioning is a fundamental tool in data mining. It helps us explore the structure of a set of data, while developing easy to visualize decision rules for predicting a categorical (classification tree) or continuous (regression tree) outcome.

CART Modeling via rpart

Classification and regression trees (as described by Brieman, Freidman, Olshen, and Stone) can be generated through the rpart package. Detailed information on rpart is available in An Introduction to Recursive Partitioning Using the RPART Routines. The general steps are provided below followed by two examples.

1. Grow the Tree

To grow a tree, use

rpart(*formula*, **data=**, **method=,control=)** where

formula	is in the format *outcome ~ predictor1+predictor2+predictor3*+ect.
data=	specifies the data frame
method=	**"class"** for a classification tree **"anova"** for a regression tree
control=	optional parameters for controlling tree growth. For example, control=rpart.control(minsplit=30, cp=0.001) requires that the minimum number of observations in a node be 30 before attempting a split and that a split must decrease the overall lack of fit by a factor of 0.001 (cost complexity factor) before being attempted.

2. Examine the results

The following functions help us to examine the results.

printcp(*fit***)**	display cp table
plotcp(*fit***)**	plot cross-validation results
rsq.rpart(*fit***)**	plot approximate R-squared and relative error for different splits (2 plots). labels are only appropriate for the "anova" method.
print(*fit***)**	print results
summary(*fit***)**	detailed results including surrogate splits
plot(*fit***)**	plot decision tree
text(*fit***)**	label the decision tree plot
post(*fit*, **file=)**	create postscript plot of decision tree

In trees created by **rpart( )**, move to the **LEFT** branch when the stated condition is true.

3. prune tree

Prune back the tree to avoid overfitting the data. Typically, you will want to select a tree size that minimizes the cross-validated error, the **xerror** column printed by **printcp( )**.

Prune the tree to the desired size using

prune(*fit*, cp=)

Specifically, use **printcp( )** to examine the cross-validated error results, select the complexity parameter associated with minimum error, and place it into the **prune( )** function. Alternatively, you can use the code fragment

```
fit$cptable[which.min(fit$cptable[,"xerror"]),"CP"]
```

to automatically select the complexity parameter associated with the smallest cross-validated error. Thanks to HSAUR for this idea.