

CARTWARE Documentation

CARTWARE is a collection of R functions written for Classification and Regression Tree (CART) Analysis of ecological data sets. All of these functions make use of existing R functions and the core `cart()` function is simply convenience wrapper for the `rpart()` function contained in the `rpart` library.

Author:

Brad Compton, Research Associate
Department of Natural Resources Conservation
University of Massachusetts, Amherst

Date Last Updated:

13 November, 2006

Functions:

Cartware.R includes 23 separate functions, but most of these functions are sub-functions that are called from other primary functions (i.e., the user would not likely call the sub-functions directly). The help files in this document include only the ‘primary’ cartware functions:

<code>cart</code>	2
<code>monte.cart</code>	4
<code>rhist</code>	6
<code>var.importance</code>	7

Function: cart

Description:

This is the primary cartware function. Cart() is simply a convenience wrapper for the rpart() function in the rpart library. Cart() computes both classification and regression trees, depending on the type of dependent variable. If the dependent variable is categorical (i.e., designated a 'factor' in R), a classification tree will be created, otherwise a regression tree will be created. In addition to all of the arguments available in rpart(), the cart() function includes a couple of additional arguments to generate smoothed relative error plot (aka the cp plot) and automate tree pruning based on a 10-fold cross-validation and the 1-SE rule.

Usage:

```
cart(formula, data, ..., control=usual, pick = TRUE, prune = 0, textsize = 0.75, smooth = 1, margin=.06)
```

Dependency:

library(rpart).

Arguments:

formula: *required* formula (symbolic description) of the model to be fit. A typical model has the form 'response ~ terms' where 'response' is the dependent variable or response vector (numeric or character) and 'terms' is a series of terms which specifies a linear predictor for 'response'. A typical model has the form $y \sim x_1 + x_2 + \dots$. Note, rpart does not allow for interaction terms (e.g., $x_1 * x_2$). Alternatively, all the variables in the data= object can be included as independent variables by specifying a period (.) on the right hand side of the equation. In this case, the data object should not include the dependent variable. Note, for a classification tree the term on the left hand side of the equation must be a 'factor'.

data: *required* data frame in which to interpret the variables named in the formula. Note, the data frame can include continuous, categorical and/or count variables.

... *optional* parameters for the splitting function, including parameters for specifying priors, costs and the splitting criterion (see help(rpart) for details).

control: *optional* controls on details of the 'rpart' algorithm. See help(rpart.controls) for details on the parameters that can be controlled. To change these controls, specify control=rpart.control("put list of controls here") as an argument in the cart() call. See example below. The default control settings are specified by the function usual(). Type 'usual' to see the current settings. [default = usual]

pick: *optional* logical (TRUE or FALSE) of whether to use a 10-fold cross-validation and 1-SE rule for picking the right-sized tree. [default = TRUE]

prune: *optional* level for pruning the tree. The level specified is the value of the complexity parameter (cp). [default = 0; no pruning]

textsize: *optional* textsize for the cart plot. [default = 0.75]

smooth: *optional* smoothing parameter (positive integer value) for the relative error plot (aka cp plot). The smoothing parameter specifies the number of times the V-fold cross-validation is conducted to estimate the relative error for trees of various sizes. If smoothing = m, the relative error is computed as the average across the m V-fold cross-validations. If $m \gg 1$, the cp plot will be smoothed over many runs and not be the reflection of a single random subsetting of the data. [default = 1; no smoothing]

margin: *optional* width of the margin for the tree plot. [default = .06]

Details:

cart() is simply a wrapper for rpart(). See the help files for rpart() for more details.

Value:

Returns an object of class 'rpart', a superset of class 'tree', which is a list containing several components. See help(rpart.object) for details on the list components.

Author:

B. Compton, August 30, 2006

References:

See references in rpart().

Examples:

```
turtle<-read.csv('byturtle.csv',header=TRUE)
grp<-as.factor(turtle[,3])
y<-turtle[,6:30]
z<-cart(grp~.,data=y,method='class',parms=list(split='gini'),control=rpart.control(minsplit=2,
minbucket=1),pick=FALSE)
```

Function: monte.cart

Description:

Computes a P-value for a tree, based on Monte Carlo resampling. Specifically, monte.cart() conducts a randomization test of the null hypothesis that the overall correct classification rate of the tree is greater than that expected if there were no real group structure to the data.

Usage:

```
monte.cart(model, data, size, n = 1000, ifplot=FALSE, ifcartplot=TRUE, control=usual,
           parms, ...)
```

Dependency:

library(rpart).

Arguments:

- model: *required* rpart model; same as formula in cart() and rpart().
- data: *required* data frame in which to interpret the variables named in the model formula. Note, the data frame can include continuous, categorical and/or count variables.
- size: *required* tree size to test; must be a positive integer.
- n: *optional* number of random permutation trees to create; must be a positive integer (the more the better). [default = 1000]
- ifplot: *optional* logical (TRUE or FALSE) indicating whether to produce a density plot of the random permutation distribution of correct classification rate (CCR) under the null hypothesis (curve) vs. the CCR of the original tree (triangle). [default = FALSE]
- ifcartplot: *optional* logical (TRUE or FALSE) indicating whether to produce a tree for the original model based on the full data set. Note, this is the same tree that is produced by cart() or rpart() for the same model. [default = TRUE]
- control: *optional* controls on details of the 'rpart' algorithm. See control argument in cart(). [default = usual]
- parms: *optional* parameters for the rpart splitting function, including parameters for specifying priors, costs and the splitting criterion (see help(rpart) for details). [default = use rpart defaults]
- ... *optional* arguments to rpart() and rpart.control(); see corresponding help files for

details. [default = use all function defaults]

Details:

The `monte.cart()` function works by permuting the grouping variable only. In this manner, the correlation structure of the observation vectors is maintained, but the group membership is made random with respect to the discriminating variables. `monte.cart()` then compares the original correct classification rate (CCR) with the distribution of CCRs from the random trees. If the original tree was produced by chance, we should expect to see a high P. On the other hand, if there is little chance that the original tree could have been produced from the data set without any real structure, then we should get a small P. A small P, say <0.05 , indicates that the observed CCR is significantly better than chance.

Value:

Prints a summary of the model, including its correct classification rate, Kappa or Tau statistic, tree size, and confusion matrix, as well as a summary of the permutation test result. Two p-values are reported. The first p-value is the proportion of permuted CCR's greater than the observed CCR. The second p-value is the proportion of the kernel density curve greater than the observed CCR. The kernel density curve is simply a smoothed version of the actual permutation distribution. These p-values should be similar in most cases.

In addition, `monte.cart()` returns a vector with the permuted CCR's.

Author:

B. Compton, August 30, 2006

References:

None.

Examples:

```
turtle<-read.csv('byturtle.csv',header=TRUE)
grp<-as.factor(turtle[,3])
y<-turtle[,6:30]
y.grp<-cbind(grp,y)
monte.cart(grp~strlen+ub,data=y.grp,method='class',parms=list(split='gini'),n=100,size=3,
  ifplot=TRUE)
```

Function: rhist

Description:

Creates a barplot (classification tree) or histogram (regression tree) for each leaf in the tree.

Usage:

```
rhist(a, digits=4)
```

Dependency:

```
library(rpart).
```

Arguments:

a: *required* rpart object, the result of running rpart() or cart().

digits: *optional* number of digits for rounding values in the histogram (regression tree).
[default = 4]

Details:

rhist() is simply a wrapper for barplot() and hist(). It produces a barplot of class membership for each node in a classification tree, giving the frequency of observations in each class, and a histogram of the dependent variable for each node in a regression tree, giving the frequency of observations in each bin. It reports the expected (or predicted) value for each node as well (y-hat). This function is useful in better understanding the composition of samples in each node.

Value:

None. Produces barplots or histograms corresponding to the nodes in the tree, from left to right, in a new graphics window.

Author:

B. Compton, August 30, 2006

References:

None.

Examples:

```
turtle<-read.csv('byturtle.csv',header=TRUE)
grp<-as.factor(turtle[,3])
y<-turtle[,6:30]
z<-cart(grp~.,data=y,method='class',parms=list(split='gini'),control=rpart.control(minsplit=2,
minbucket=1),pick=FALSE)
rhist(z)
```

Function: var.importance

Description:

Computes variable importance for classification and regression trees.

Usage:

```
var.importance(x, chatter = FALSE)
```

Dependency:

```
library(rpart).
```

Arguments:

x: *required* rpart object, typically derived by running rpart() or cart().

chatter: *optional* logical (TRUE or FALSE) indicating whether to print details of the variable importance calculations at each node in the tree. [default = FALSE]

Details:

While a parsimonious tree and accurate prediction is often the primary goal, we are sometimes also interested in how each of the measured variables, including those that did not make it into the final tree, perform in terms of their ability to predict. For this purpose, it is useful to compute a measure of variable importance. Variable importance is determined by calculating for each variable at each node the change in impurity attributable to the best surrogate split on that variable. Impurity is measured using the gini index in classification trees and deviance in regression trees. The sum of these deltas across all nodes for each variable, $M(x_m)$, is a measure of how much total decrease in impurity can be achieved by each variable. Since only the relative magnitude of the $M(x_m)$ are interesting, the actual measures of importance are generally given as the normalized quantiles.

Note that variable importance is calculated for the tree structure given in the rpart object specified (x). The structure of the tree (i.e., the depth of the tree and the splits at each node) is determined by the criteria specified for growing and pruning the tree. Changing the tree criteria will usually result in a different tree, and this will affect the variable importance measures. In particular, changing the number of surrogate splitters considered at each node can have a significant effect on variable importance.

BEWARE, the computed importance values for *classification* trees do not agree exactly with those produced by the commercial CART_(tm) software, for reasons that we have not been able to figure out, but they are very close. Moreover, the importance values for classification trees are only approximately correct when priors are proportional to group sample sizes (the default) and missclassification costs have not been specified (the default), although the results match CART_(tm) for regression trees. So, use this function with some caution.

Value:

Returns a data frame containing two columns: column 1 is the variable (including any variable that was evaluated as a surrogate splitter at any node in the tree); column 2 is the normalized relative importance value. The most important value is always given a score of 100 and all other values are relative.

Author:

B. Compton, August 30, 2006

References:

None.

Examples:

```
turtle<-read.csv('byturtle.csv',header=TRUE)
grp<-as.factor(turtle[,3])
y<-turtle[,6:30]
z<-cart(grp~.,data=y,method='class',parms=list(split='gini'),smooth=30,pick=TRUE)
var.importance(z)
```