

# Package ‘tree’

July 30, 2009

**Title** Classification and regression trees

**Version** 1.0-27

**Date** 2009-07-28

**Depends** R (>= 2.7.0), grDevices, graphics, stats

**Author** Brian Ripley <ripley@stats.ox.ac.uk>.

**Maintainer** Brian Ripley <ripley@stats.ox.ac.uk>

**Description** Classification and Regression Trees.

**LazyLoad** yes

**License** GPL-2 | GPL-3

**Repository** CRAN

**Date/Publication** 2009-07-30 12:27:39

## R topics documented:

cv.tree . . . . .	2
deviance.tree . . . . .	3
misclass.tree . . . . .	3
na.tree.replace . . . . .	4
partition.tree . . . . .	5
plot.tree . . . . .	6
plot.tree.sequence . . . . .	7
predict.tree . . . . .	8
prune.tree . . . . .	9
snip.tree . . . . .	11
text.tree . . . . .	12
tile.tree . . . . .	13
tree . . . . .	14
tree.control . . . . .	16
tree.screens . . . . .	17

<b>Index</b>	<b>19</b>
--------------	-----------

---

`cv.tree`*Cross-validation for Choosing Tree Complexity*

---

**Description**

Runs a K-fold cross-validation experiment to find the deviance or number of misclassifications as a function of the cost-complexity parameter  $k$ .

**Usage**

```
cv.tree(object, rand, FUN = prune.tree, K = 10, ...)
```

**Arguments**

<code>object</code>	An object of class "tree".
<code>rand</code>	Optionally an integer vector of the length the number of cases used to create <code>object</code> , assigning the cases to different groups for cross-validation.
<code>FUN</code>	The function to do the pruning.
<code>K</code>	The number of folds of the cross-validation.
<code>...</code>	Additional arguments to <code>FUN</code> .

**Value**

A copy of `FUN` applied to `object`, with component `dev` replaced by the cross-validated results from the sum of the `dev` components of each fit.

**Author(s)**

B. D. Ripley

**See Also**

[tree](#), [prune.tree](#)

**Examples**

```
data(cpus, package="MASS")
cpus.ltr <- tree(log10(perf) ~ syct + mmin + mmax + cach
  + chmin + chmax, data=cpus)
cv.tree(cpus.ltr, , prune.tree)
```

---

deviance.tree      *Extract Deviance from a Tree Object*

---

### Description

Extract deviance from a tree object.

### Usage

```
## S3 method for class 'tree':  
deviance(object, detail = FALSE, ...)
```

### Arguments

object	an object of class "tree"
detail	logical. If true, returns a vector of deviance contributions from each node.
...	arguments to be passed to or from other methods.

### Value

The overall deviance, or a vector of contributions from the cases at each node. The overall deviance is the sum over leaves in the latter case.

---

misclass.tree      *Misclassifications by a Classification Tree*

---

### Description

Report the number of mis-classifications made by a classification tree, either overall or at each node.

### Usage

```
misclass.tree(tree, detail = FALSE)
```

### Arguments

tree	Object of class "tree", representing a classification tree.
detail	If false, report overall number of mis-classifications. If true, report the number at each node.

### Details

The quantities returned are weighted by the observational weights if these are supplied in the construction of `tree`.

**Value**

Either the overall number of misclassifications or the number for each node.

**Author(s)**

B. D. Ripley

**See Also**

[tree](#)

**Examples**

```
ir.tr <- tree(Species ~., iris)
misclass.tree(ir.tr)
misclass.tree(ir.tr, detail=TRUE)
```

---

na.tree.replace      *Replace NAs in Predictor Variables*

---

**Description**

Adds a new level called "NA" to any discrete predictor in a data frame that contains NAs. Stops if any continuous predictor contains an NA.

**Usage**

```
na.tree.replace(frame)
```

**Arguments**

frame              data frame used to grow a tree.

**Details**

This function is used via the `na.action` argument to `tree`.

**Value**

data frame such that a new level named "NA" is added to any discrete predictor in `frame` with NAs.

**See Also**

[tree](#), [na.omit](#).

---

partition.tree      *Plot the Partitions of a simple Tree Model*

---

**Description**

Plot the partitions of a tree involving one or two variables.

**Usage**

```
partition.tree(tree, label = "yval", add = FALSE, ordvars, ...)
```

**Arguments**

tree	A object of class "tree".
label	A character string giving the column of the frame component of tree to be used to label the regions.
add	If true, add to existing plot, otherwise start a new plot.
ordvars	The ordering of the variables to be used in a 2D plot. Specify the names in a character string of length 2; the first will be used on the x axis.
...	Graphical parameters.

**Details**

This can be used with a regression or classification tree containing one or two continuous predictors (only).

If the tree contains one predictor, the predicted value (a regression tree) or the probability of the first class (a classification tree) is plotted against the predictor over its range in the training set.

If the tree contains two predictors, a plot is made of the space covered by those two predictors and the partition made by the tree is superimposed.

**Value**

None.

**Author(s)**

B. D. Ripley

**See Also**

[tree](#)

**Examples**

```

ir.tr <- tree(Species ~., iris)
ir.tr
ir.tr1 <- snip.tree(ir.tr, nodes = c(12, 7))
summary(ir.tr1)
par(pty = "s")
plot(iris[, 3], iris[, 4], type="n",
      xlab="petal length", ylab="petal width")
text(iris[, 3], iris[, 4], c("s", "c", "v")[iris[, 5]])
partition.tree(ir.tr1, add = TRUE, cex = 1.5)

# 1D example
ir.tr <- tree(Petal.Width ~ Petal.Length, iris)
plot(iris[,3], iris[,4], type="n", xlab="Length", ylab="Width")
partition.tree(ir.tr, add = TRUE, cex = 1.5)

```

plot.tree

*Plot a Tree Object***Description**

Plot a tree object on the current graphical device

**Usage**

```

## S3 method for class 'tree':
plot(x, y = NULL, type = c("proportional", "uniform"), ...)

```

**Arguments**

x	an object of class "tree".
y	ignored. Used for positional matching of type.
type	character string. If this partially matches "uniform", the branches are of uniform length. Otherwise they are proportional to the decrease in impurity.
...	graphical parameters.

**Value**

An (invisible) list with components x and y giving the coordinates of the tree nodes.

As a side effect, the value of `type == "uniform"` is stored in the variable `.Tree.unif.?` in the global environment, where ? is the device number.

**Author(s)**

B. D. Ripley

**See Also**

[tree](#)

---

`plot.tree.sequence` *Plot a Tree Sequence*

---

## Description

Allows the user to plot a tree sequence.

## Usage

```
## S3 method for class 'tree.sequence':  
plot(x, ..., type = "l", ylim = range(x$dev),  
      order = c("increasing", "decreasing"))
```

## Arguments

<code>x</code>	object of class <code>tree.sequence</code> . This is assumed to be the result of some function that produces an object with the same named components ( <code>size</code> , <code>deviance</code> , <code>k</code> ) as that returned by <code>prune.tree</code> .
<code>order</code>	of size on the plot. Use "decreasing" for the natural ordering of <code>k</code> and the amount of pruning. Only the first character is needed.
<code>type, ylim, ...</code>	graphical parameters.

## Details

This function is a method for the generic function `plot()` for class `tree.sequence`. It can be invoked by calling `plot(x)` for an object `x` of the appropriate class, or directly by calling `plot.tree.sequence(x)` regardless of the class of the object.

## Side Effects

Plots deviance or number of misclassifications (or total loss) versus size for a sequence of trees.

## Examples

```
data(cpus, package="MASS")  
cpus.ltr <- tree(log(perf) ~ syct + mmin + mmax + cach + chmin + chmax,  
                data = cpus)  
plot(prune.tree(cpus.ltr))
```

---

`predict.tree`*Predictions from a Fitted Tree Object*

---

## Description

Returns a vector of predicted responses from a fitted tree object.

## Usage

```
## S3 method for class 'tree':
predict(object, newdata = list(),
        type = c("vector", "tree", "class", "where"),
        split = FALSE, nwts, eps = 1e-3, ...)
```

## Arguments

<code>object</code>	fitted model object of class <code>tree</code> . This is assumed to be the result of some function that produces an object with the same named components as that returned by the <code>tree</code> function.
<code>newdata</code>	data frame containing the values at which predictions are required. The predictors referred to in the right side of <code>formula(object)</code> must be present by name in <code>newdata</code> . If missing, fitted values are returned.
<code>type</code>	character string denoting whether the predictions are returned as a vector (default) or as a tree object.
<code>split</code>	governs the handling of missing values. If false, cases with missing values are dropped down the tree until a leaf is reached or a node for which the attribute is missing, and that node is used for prediction. If <code>split = TRUE</code> cases with missing attributes are split into fractional cases and dropped down each side of the split. The predicted values are averaged over the fractions to give the prediction.
<code>nwts</code>	weights for the <code>newdata</code> cases, used when predicting a tree.
<code>eps</code>	a lower bound for the probabilities, used if events of predicted probability zero occur in <code>newdata</code> when predicting a tree.
<code>...</code>	further arguments passed to or from other methods.

## Details

This function is a method for the generic function `predict()` for class `tree`. It can be invoked by calling `predict(x)` for an object `x` of the appropriate class, or directly by calling `predict.tree(x)` regardless of the class of the object.

**Value**

If `type = "vector"`: vector of predicted responses or, if the response is a factor, matrix of predicted class probabilities. This new object is obtained by dropping `newdata` down `object`. For factor predictors, if an observation contains a level not used to grow the tree, it is left at the deepest possible node and `frame$yval` or `frame$yprob` at that node is the prediction.

If `type = "tree"`: an object of class "tree" is returned with new values for `frame$n` and `frame$dev`. If `newdata` does not contain a column for the response in the formula the value of `frame$dev` will be NA, and if some values in the response are missing, the some of the deviances will be NA.

If `type = "class"`: for a classification tree, a factor of the predicted classes (that with highest posterior probability, with ties split randomly).

If `type = "where"`: the nodes the cases reach.

**References**

Ripley, B. D. (1996). *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge. Chapter 7.

**See Also**

[predict, tree.](#)

**Examples**

```
data(shuttle, package="MASS")
shuttle.tr <- tree(use ~ ., shuttle, subset=1:253,
                  mindev=1e-6, minsize=2)
shuttle.tr
shuttle1 <- shuttle[254:256, ] # 3 missing cases
predict(shuttle.tr, shuttle1)
```

---

prune.tree

*Cost-complexity Pruning of Tree Object*


---

**Description**

Determines a nested sequence of subtrees of the supplied tree by recursively “snipping” off the least important splits.

**Usage**

```
prune.tree(tree, k = NULL, best = NULL, newdata, nwts,
           method = c("deviance", "misclass"), loss, eps = 1e-3)
```

```
prune.misclass(tree, k = NULL, best = NULL, newdata,
               nwts, loss, eps = 1e-3)
```

**Arguments**

<code>tree</code>	fitted model object of class <code>tree</code> . This is assumed to be the result of some function that produces an object with the same named components as that returned by the <code>tree()</code> function.
<code>k</code>	cost-complexity parameter defining either a specific subtree of <code>tree</code> ( <code>k</code> a scalar) or the (optional) sequence of subtrees minimizing the cost-complexity measure ( <code>k</code> a vector). If missing, <code>k</code> is determined algorithmically.
<code>best</code>	integer requesting the size (i.e. number of terminal nodes) of a specific subtree in the cost-complexity sequence to be returned. This is an alternative way to select a subtree than by supplying a scalar cost-complexity parameter <code>k</code> . If there is no tree in the sequence of the requested size, the next largest is returned.
<code>newdata</code>	data frame upon which the sequence of cost-complexity subtrees is evaluated. If missing, the data used to grow the tree are used.
<code>nwts</code>	weights for the <code>newdata</code> cases.
<code>method</code>	character string denoting the measure of node heterogeneity used to guide cost-complexity pruning. For regression trees, only the default, <code>deviance</code> , is accepted. For classification trees, the default is <code>deviance</code> and the alternative is <code>misclass</code> (number of misclassifications or total loss).
<code>loss</code>	a matrix giving for each true class (row) the numeric loss of predicting the class (column). The classes should be in the order of the levels of the response. It is conventional for a loss matrix to have a zero diagonal. The default is 0–1 loss.
<code>eps</code>	a lower bound for the probabilities, used to compute deviances if events of predicted probability zero occur in <code>newdata</code> .

**Details**

Determines a nested sequence of subtrees of the supplied tree by recursively "snipping" off the least important splits, based upon the cost-complexity measure. `prune.misclass` is an abbreviation for `prune.tree(method = "misclass")` for use with `cv.tree`.

If `k` is supplied, the optimal subtree for that value is returned.

The response as well as the predictors referred to in the right side of the formula in `tree` must be present by name in `newdata`. These data are dropped down each tree in the cost-complexity sequence and deviances or losses calculated by comparing the supplied response to the prediction. The function `cv.tree()` routinely uses the `newdata` argument in cross-validating the pruning procedure. A `plot` method exists for objects of this class. It displays the value of the deviance, the number of misclassifications or the total loss for each subtree in the cost-complexity sequence. An additional axis displays the values of the cost-complexity parameter at each subtree.

**Value**

If `k` is supplied and is a scalar, a `tree` object is returned that minimizes the cost-complexity measure for that `k`. If `best` is supplied, a `tree` object of size `best` is returned. Otherwise, an object of class `tree.sequence` is returned. The object contains the following components:

<code>size</code>	number of terminal nodes in each tree in the cost-complexity pruning sequence.
<code>deviance</code>	total deviance of each tree in the cost-complexity pruning sequence.
<code>k</code>	the value of the cost-complexity pruning parameter of each tree in the sequence.

**Examples**

```

data(fgl, package="MASS")
fgl.tr <- tree(type ~ ., fgl)
plot(print(fgl.tr))
fgl.cv <- cv.tree(fgl.tr,, prune.tree)
for(i in 2:5) fgl.cv$dev <- fgl.cv$dev +
  cv.tree(fgl.tr,, prune.tree)$dev
fgl.cv$dev <- fgl.cv$dev/5
plot(fgl.cv)

```

snip.tree

*Snip Parts of Tree Objects***Description**

snip.tree has two related functions. If nodes is supplied, it removes those nodes and all their descendants from the tree.

If nodes is not supplied, the user is invited to select nodes interactively; this makes sense only if the tree has already been plotted. A node is selected by clicking with the left mouse button; its number and the deviance of the current tree and that which would remain if that node were removed are printed. Selecting the same node again causes it to be removed (and the lines of its sub-tree erased). Clicking any other button terminates the selection process.

**Usage**

```

snip.tree(tree, nodes, xy.save = FALSE,
          digits = getOption("digits") - 3)

```

**Arguments**

tree	An object of class "tree".
nodes	An integer vector giving those nodes that are the roots of sub-trees to be snipped off. If missing, the user is invited to select a node at which to snip.
xy.save	If true, the x and y coordinates selected interactively are saved in the object .xy in the global environment.
digits	Precision used in printing statistics for selected nodes.

**Value**

A tree object containing the nodes that remain after specified or selected subtrees have been snipped off.

**Author(s)**

B. D. Ripley

**See Also**

[tree](#), [prune.tree](#).

---

text.tree

*Annotate a Tree Plot*

---

**Description**

Add text to a tree plot.

**Usage**

```
## S3 method for class 'tree':
text(x, splits = TRUE, label = "yval", all = FALSE,
     pretty = NULL, digits = getOption("digits") - 3,
     adj = par("adj"), xpd = TRUE, ...)
```

**Arguments**

x	an object of class "tree"
splits	logical. If TRUE the splits are labelled
label	The name of column in the frame component of x, to be used to label the nodes. Can be NULL to suppress node-labelling
all	logical. By default, only the leaves are labelled, but if true interior nodes are also labelled.
pretty	the manipulation used for split labels involving attributes. See Details.
digits	significant digits for numerical labels.
adj, xpd, ...	graphical parameters such as cex and font.

**Details**

If `pretty = 0` then the level names of a factor split attributes are used unchanged. If `pretty = NULL`, the levels are presented by a, b, ... If `pretty` is a positive integer, `abbreviate` is applied to the labels with that value for its argument `minlength`.

If the lettering is vertical (`par srt = 90`) and `adj` is not supplied it is adjusted appropriately.

**Value**

None.

**Author(s)**

B. D. Ripley

**See Also**[plot.tree](#)**Examples**

```
ir.tr <- tree(Species ~., iris)
plot(ir.tr)
text(ir.tr)
```

---

`tile.tree`*Add Class Barcharts to a Classification Tree Plot*

---

**Description**

This computes the frequencies of level of `var` for cases reaching each leaf of the tree, and plots barcharts of the set of frequencies underneath each leaf.

**Usage**

```
tile.tree(tree, var, screen.arg = ascr + 1, axes = TRUE)
```

**Arguments**

<code>tree</code>	fitted object of class "tree".
<code>var</code>	a factor variable to be displayed: by default it is the response factor of the tree.
<code>screen.arg</code>	The screen to be used: default the next after the currently active screen.
<code>axes</code>	logical flag for drawing of axes for the barcharts.

**Value**

A matrix of counts of categories (rows) for each leaf (columns). The principal effect is the plot.

**Author(s)**

B. D. Ripley

**See Also**[tree.screens](#)

## Examples

```
data(fgl, package="MASS")
fgl.tr <- tree(type ~ ., fgl)
summary(fgl.tr)
plot(fgl.tr); text(fgl.tr, all=TRUE, cex=0.5)
fgl.tr1 <- snip.tree(fgl.tr, node=c(108, 31, 26))
tree.screens()
plot(fgl.tr1)
text(fgl.tr1)
tile.tree(fgl.tr1, fgl$type)
close.screen(all = TRUE)
```

---

tree

*Fit a Classification or Regression Tree*

---

## Description

A tree is grown by binary recursive partitioning using the response in the specified formula and choosing splits from the terms of the right-hand-side.

## Usage

```
tree(formula, data, weights, subset,
      na.action = na.pass, control = tree.control(nobs, ...),
      method = "recursive.partition",
      split = c("deviance", "gini"),
      model = FALSE, x = FALSE, y = TRUE, wts = TRUE, ...)
```

## Arguments

formula	A formula expression. The left-hand-side (response) should be either a numerical vector when a regression tree will be fitted or a factor, when a classification tree is produced. The right-hand-side should be a series of numeric or factor variables separated by +; there should be no interaction terms. Both . and - are allowed: regression trees can have <code>offset</code> terms.
data	A data frame in which to preferentially interpret <code>formula</code> , <code>weights</code> and <code>subset</code> .
weights	Vector of non-negative observational weights; fractional weights are allowed.
subset	An expression specifying the subset of cases to be used.
na.action	A function to filter missing data from the model frame. The default is <code>na.pass</code> (to do nothing) as <code>tree</code> handles missing values (by dropping them down the tree as far as possible).
control	A list as returned by <code>tree.control</code> .
method	character string giving the method to use. The only other useful value is <code>"model.frame"</code> .
split	Splitting criterion to use.

<code>model</code>	If this argument is itself a model frame, then the <code>formula</code> and <code>data</code> arguments are ignored, and <code>model</code> is used to define the model. If the argument is logical and true, the model frame is stored as component <code>model</code> in the result.
<code>x</code>	logical. If true, the matrix of variables for each case is returned.
<code>y</code>	logical. If true, the response variable is returned.
<code>wts</code>	logical. If true, the weights are returned.
<code>...</code>	Additional arguments that are passed to <code>tree.control</code> . Normally used for <code>mincut</code> , <code>minsize</code> or <code>mindev</code> .

### Details

A tree is grown by binary recursive partitioning using the response in the specified formula and choosing splits from the terms of the right-hand-side. Numeric variables are divided into  $X < a$  and  $X > a$ ; the levels of an unordered factor are divided into two non-empty groups. The split which maximizes the reduction in impurity is chosen, the data set split and the process repeated. Splitting continues until the terminal nodes are too small or too few to be split.

Tree growth is limited to a depth of 31 by the use of integers to label nodes.

Factor predictor variables can have up to 32 levels. This limit is imposed for ease of labelling, but since their use in a classification tree with three or more levels in a response involves a search over  $2^{(k-1)} - 1$  groupings for  $k$  levels, the practical limit is much less.

### Value

The value is an object of class `"tree"` which has components

<code>frame</code>	A data frame with a row for each node, and <code>row.names</code> giving the node numbers. The columns include <code>var</code> , the variable used at the split (or <code>"&lt;leaf&gt;"</code> for a terminal node), <code>n</code> , the (weighted) number of cases reaching that node, <code>dev</code> the deviance of the node, <code>yval</code> , the fitted value at the node (the mean for regression trees, a majority class for classification trees) and <code>split</code> , a two-column matrix of the labels for the left and right splits at the node. Classification trees also have <code>yprob</code> , a matrix of fitted probabilities for each response level.
<code>where</code>	An integer vector giving the row number of the frame detailing the node to which each case is assigned.
<code>terms</code>	The terms of the formula.
<code>call</code>	The matched call to <code>Tree</code> .
<code>model</code>	If <code>model = TRUE</code> , the model frame.
<code>x</code>	If <code>x = TRUE</code> , the model matrix.
<code>y</code>	If <code>y = TRUE</code> , the response.
<code>wts</code>	If <code>wts = TRUE</code> , the weights.

and attributes `xlevels` and, for classification trees, `ylevels`.

A tree with no splits is of class `"singlenode"` which inherits from class `"tree"`.

**Author(s)**

B. D. Ripley

**References**

Breiman L., Friedman J. H., Olshen R. A., and Stone, C. J. (1984) *Classification and Regression Trees*. Wadsworth.

Ripley, B. D. (1996) *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge. Chapter 7.

**See Also**

[tree.control](#), [prune.tree](#), [predict.tree](#), [snip.tree](#)

**Examples**

```
data(cpus, package="MASS")
cpus.ltr <- tree(log10(perf) ~ syct+mmin+mmax+cach+chmin+chmax, cpus)
cpus.ltr
summary(cpus.ltr)
plot(cpus.ltr); text(cpus.ltr)

ir.tr <- tree(Species ~., iris)
ir.tr
summary(ir.tr)
```

---

tree.control

*Select Parameters for Tree*

---

**Description**

A utility function for use with the `control` argument of `tree`.

**Usage**

```
tree.control(nobs, mincut = 5, minsize = 10, mindev = 0.01)
```

**Arguments**

<code>nobs</code>	The number of observations in the training set.
<code>mincut</code>	The minimum number of observations to include in either child node. This is a weighted quantity; the observational weights are used to compute the ‘number’. The default is 5.
<code>minsize</code>	The smallest allowed node size: a weighted quantity. The default is 10.
<code>mindev</code>	The within-node deviance must be at least this times that of the root node for the node to be split.

**Details**

This function produces default values of `mincut` and `minsize`, and ensures that `mincut` is at most half `minsize`.

To produce a tree that fits the data perfectly, set `mindev = 0` and `minsize = 2`, if the limit on tree depth allows such a tree.

**Value**

A list:

<code>mincut</code>	The maximum of the input or default <code>mincut</code> and 1
<code>minsize</code>	The maximum of the input or default <code>minsize</code> and 2.
<code>nmax</code>	A estimate of the maximum number of nodes that might be grown.
<code>nobs</code>	The input <code>nobs</code> .

**Note**

The interpretation of `mindev` given here is that of Chambers and Hastie (1992, p. 415), and apparently not what is actually implemented in S. It seems S uses an absolute bound.

**Author(s)**

B. D. Ripley

**See Also**

[tree](#)

---

`tree.screens`      *Split Screen for Plotting Trees*

---

**Description**

Splits the screen in a way suitable for using `tile.tree`.

**Usage**

```
tree.screens(figs, screen.arg = 0, ...)
```

**Arguments**

<code>figs</code>	A specification of the split of the screen. See <a href="#">split.screen</a> for the allowed forms.
<code>screen.arg</code>	the screen to divide, by default the whole display area.
<code>...</code>	plot parameters to be passed to <code>par</code> .

**Value**

A vector of screen numbers for the newly-created screens.

**Author(s)**

B. D. Ripley

**See Also**

[tile.tree](#), [split.screen](#)

**Examples**

```
data(fgl, package="MASS")
fgl.tr <- tree(type ~ ., fgl)
summary(fgl.tr)
plot(fgl.tr); text(fgl.tr, all=TRUE, cex=0.5)
fgl.tr1 <- snip.tree(fgl.tr, node=c(108, 31, 26))
tree.screens()
plot(fgl.tr1)
tile.tree(fgl.tr1, fgl$type)
close.screen(all = TRUE)
```

# Index

## \*Topic **hplot**

- partition.tree, 4
- plot.tree, 6
- text.tree, 11
- tile.tree, 13
- tree.screens, 17

## \*Topic **tree**

- cv.tree, 1
- deviance.tree, 2
- misclass.tree, 3
- na.tree.replace, 4
- partition.tree, 4
- plot.tree, 6
- plot.tree.sequence, 6
- predict.tree, 7
- prune.tree, 9
- snip.tree, 10
- text.tree, 11
- tile.tree, 13
- tree, 14
- tree.control, 16
- tree.screens, 17

abbreviate, 12

cv.tree, 1, 10

deviance.singlenode  
(deviance.tree), 2

deviance.tree, 2

misclass.tree, 3

na.omit, 4

na.tree.replace, 4

partition.tree, 4

plot.tree, 6, 12

plot.tree.sequence, 6

predict, 8

predict.tree, 7, 16

print.summary.tree(tree), 14

print.tree(tree), 14

prune.misclass(prune.tree), 9

prune.tree, 2, 7, 9, 11, 16

residuals.tree(tree), 14

snip.tree, 10, 16

split.screen, 17, 18

summary.tree(tree), 14

text.tree, 11

tile.tree, 13, 18

tree, 2–6, 8, 11, 14, 17

tree.control, 16, 16

tree.screens, 13, 17