Package 'oblique.tree'

February 20, 2015

Type Package

Title Oblique Trees for Classification Data

Version 1.1.1

Date 2012-04-14

Author Alfred Truong

Maintainer Alfred Truong <alfred.truong@gmail.com>

Description Grows Oblique Trees to Classification Data.

License GPL-3

Depends tree, glmnet

Imports nnet

Suggests MASS

Repository CRAN

Date/Publication 2013-07-14 11:08:32

NeedsCompilation yes

R topics documented:

`generate.ith.superclass`
Internal Function to Identify of Ideal Outcomes

Description

Generates logical vectors that identify ideal outcomes.

Usage

```
generate.ith.superclass(
R,
superclass.index)
```

Arguments

`R`	The number of residual classes at a node.
`superclass.index`	Integer in $1, \ldots, 2^{R-1} - 1$ indicating required ideal outcome.

Details

With R residual classes at a node, $2^{R-1}-1$ different ideal outcomes arise. `generate.ith.superclass` identifies them.

Value

A vector of length R with 0's and 1's denoting association with the left superclass.

Author(s)

A. Truong

References

Truong. A (2009) *Fast Growing and Interpretable Oblique Trees via Probabilistic Models*

Examples

```
#list all ideal outcomes when there are 5 residual classes at a node
for (i in 1:(2^{5-1}-1)){
ideal.outcome.i <- oblique.tree:::generate.ith.superclass(
R = 5,
superclass.index = i)
print(ideal.outcome.i)
}
```

glmpath *Fits the entire L1 regularization path for generalized linear models*

Description

This algorithm uses predictor-corrector method to compute the entire regularization path for generalized linear models with L1 penalty.

Usage

```
glmpath(x, y, data, nopenalty.subset = NULL, family = binomial,
        weight = rep(1, n), offset = rep(0, n), lambda2 = 1e-5,
        max.steps = 10*min(n, m), max.norm = 100*m,
        min.lambda = (if (m >= n) 1e-6 else 0), max.vars = Inf,
        max.arclength = Inf, frac.arclength = 1, add.newvars = 1,
        bshoot.threshold = 0.1, relax.lambda = 1e-8,
        standardize = TRUE, function.precision = 3e-13,
        eps = .Machine$double.eps, trace = FALSE)
```

Arguments

x	matrix of features
y	response
data	a list consisting of x: a matrix of features and y: response. data is not needed if x and y are input separately.
nopenalty.subset	a set of indices for the predictors that are not subject to the L1 penalty
family	name of a family function that represents the distribution of y to be used in the model. It must be binomial, gaussian, or poisson. For each one, the canonical link function is used; logit for binomial, identity for gaussian, and log for poisson distribution. Default is binomial.
weight	an optional vector of weights for observations
offset	an optional vector of offset. If a column of x is used as offset, the corresponding column must be removed from x.
lambda2	regularization parameter for the L2 norm of the coefficients. Default is 1e-5.
max.steps	an optional bound for the number of steps to be taken. Default is 10 * min{nrow(x), ncol(x)}.
max.norm	an optional bound for the L1 norm of the coefficients. Default is 100 * ncol(x).
min.lambda	an optional (lower) bound for the size of λ. Default is 0 for ncol(x) < nrow(x) cases and 1e-6 otherwise.
max.vars	an optional bound for the number of active variables. Default is Inf.
max.arclength	an optional bound for arc length (L1 norm) of a step. If max.arclength is extremely small, an exact nonlinear path is produced. Default is Inf.

`frac.arclength` Under the default setting, the next step size is computed so that the active set changes right at the next value of lambda. When `frac.arclength` is assigned some fraction between 0 and 1, the step size is decreased by the factor of `frac.arclength` in arc length. If `frac.arclength=0.2`, the step length is adjusted so that the active set would change after five smaller steps. Either `max.arclength` or `frac.arclength` can be used to force the path to be more accurate. Default is `1`.

`add.newvars` `add.newvars` candidate variables (that are currently not in the active set) are used in the corrector step as potential active variables. Default is `1`.

`bshoot.threshold` If the absolute value of a coefficient is larger than `bshoot.threshold` at the first corrector step it becomes nonzero (therefore when λ is considered to have been decreased too far), λ is increased again. i.e. A backward distance in λ that makes the coefficient zero is computed. Default is `0.1`.

`relax.lambda` A variable joins the active set if $|l'(\beta)| > \lambda$*(1-`relax.lambda`). Default is `1e-8`. If no variable joins the active set even after many (>20) steps, the user should increase `relax.lambda` to `1e-7` or `1e-6`, but not more than that. This adjustment is sometimes needed because of the numerical precision/error propagation problems. In general, the paths are less accurate with relaxed lambda.

`standardize` If `TRUE`, predictors are standardized to have a unit variance.

`function.precision` `function.precision` parameter used in the internal solver. Default is `3e-13`. The algorithm is faster, but less accurate with relaxed, larger `function precision`.

`eps` an effective zero

`trace` If `TRUE`, the algorithm prints out its progress.

Details

This algorithm implements the predictor-corrector method to determine the entire path of the coefficient estimates as the amount of regularization varies; it computes a series of solution sets, each time estimating the coefficients with less regularization, based on the previous estimate. The coefficients are estimated with no error at the knots, and the values are connected, thereby making the paths piecewise linear.

We thank Michael Saunders of SOL, Stanford University for providing the solver used for the convex optimization in corrector steps of glmpath.

Value

A `glmpath` object is returned.

`lambda` vector of λ values for which the exact coefficients are computed

`lambda2` λ_2 used

`step.length` vector of step lengths in λ

`corr` matrix of $l'(\beta)$ values (derivatives of the log-likelihood)

`new.df` vector of degrees of freedom (to be used in the plot function)

`df` vector of degrees of freedom at each step

deviance	vector of deviance computed at each step
aic	vector of AIC values
bic	vector of BIC values
b.predictor	matrix of coefficient estimates from the predictor steps
b.corrector	matrix of coefficient estimates from the corrector steps
new.A	vector of boolean values indicating the steps at which the active set changed (to be used in the plot/predict functions)
actions	actions taken at each step
meanx	means of the columns of x
sdx	standard deviations of the columns of x
xnames	column names of x
family	family used
weight	weights used
offset	offset used
nopenalty.subset	nopenalty.subset used
standardize	TRUE if the predictors were standardized before fitting

Author(s)

Mee Young Park and Trevor Hastie

References

Mee Young Park and Trevor Hastie (2007) L1 regularization path algorithm for generalized linear models. *J. R. Statist. Soc.* B, 69, 659-677.

See Also

cv.glmpath, plot.glmpath, predict.glmpath, summary.glmpath

---

heart.data *Dataset for glmpath*

Description

South African Heart Disease dataset used to test glmpath algorithm

Usage

```
data(heart.data)
```

Format

A dataset with 462 observations on 9 variables and a binary response.

x x contains 9 columns of the following variables: sbp (systolic blood pressure); tobacco (cumulative tobacco); ldl (low density lipoprotein cholesterol); adiposity; famhist (family history of heart disease); typea (type-A behavior); obesity; alcohol (current alcohol consumption); age (age at onset)

y response, coronary heart disease

References

Hastie, T., Tibshirani, R., and Friedman, J. (2001) *Elements of Statistical Learning; Data Mining, Inference, and Prediction* Springer-Verlag, New York.

---

last.node.of.subtree *Internal Function that finds the Last Node of Subtrees*

Description

Returns the node number of the last node of a rooted subtree.

Usage

```
last.node.of.subtree(
tree,
subtree.root.name)
```

Arguments

`tree`	An object of class `tree`.
`subtree.root.name`	The name of the root of the subtree.

Details

Knowing the node numbers of both the root node (of a subtree) and the last node of such a subtree is sufficient to isolate all nodes from this rooted subtree (as node are numbered in a recursive manner, left first then right).

Value

The node number of the last node of a rooted subtree.

Author(s)

A. Truong

Examples

```
#grow a tree on the Pima Indian dataset
data(Pima.tr, package = "MASS")
ob.tree <- oblique.tree(formula = type~.,
data = Pima.tr,
oblique.splits = "on")
plot(ob.tree);text(ob.tree);title(main="Oblique Tree")

#what is the node number of the last node of the entire tree?
oblique.tree:::last.node.of.subtree(
tree = ob.tree,
subtree.root.name = 1)

#what is the node number of the last node of the subtree rooted at the node with name '4'?
oblique.tree:::last.node.of.subtree(
tree = ob.tree,
subtree.root.name = 4)
```

---

node.impurity *Internal Function to Calculate Impurity of Nodes*

Description

Calculates the impurity of a node.

Usage

```
node.impurity(
class.probabilities,
impurity.measure = c("deviance","gini"))
```

Arguments

`class.probabilities`
: A vector of observed class probabilities.

`impurity.measure`
: Impurity criterion to use.

Details

Where p_i are observed class probabilities, "Gini Impurity" is calculated as

$$i(p) = \sum_{i!=j} p_i p_j = 1 - \sum_i p_i^2.$$

The "deviance" measure is calculated as

$$i(p) = -2 \sum_i p_i \log(p_i)$$

to allow for internal compatibility. Note: $0 \log(0) = 0$.

Value

A value for node impurity is returned.

Author(s)

A. Truong

Examples

```
#A node only has observations of one type
oblique.tree:::node.impurity(
class.probabilities = c(1,0,0,0),
impurity.measure  = "deviance")

#Another node has equal numbers of observations of each type
oblique.tree:::node.impurity(
class.probabilities = c(0.25,0.25,0.25,0.25),
impurity.measure  = "deviance")
```

---

oblique.split.writer *Internal function to Summarize Information about Oblique Splits*

Description

Collates, rewrites and summarizes information about oblique splits.

Usage

```
oblique.split.writer(
coefficients,
impurity,
child.left.T.F,
superclass.composition)
```

Arguments

`coefficients`	A (named) vector of coefficients of an oblique split. The first component is the intercept followed by each attribute in turn.
`impurity`	Impurity of this split.
`child.left.T.F`	A logical vector denoting allocation of observations to left child node.
`superclass.composition`	A record of the targets of the classification problem that lead to this ideal split.

Details

Variable selection can reduce full oblique splits to using only one attribute which are better represented as axis-parallel splits. This function handles this representational issue.

Value

A list is returned with the following components,

`impurity`	Impurity of this split.
`child.left.T.F`	A logical vector denoting allocation of observations to left child node.
`details`	A list with `coefficients`, the coefficients of this split and `superclass.targets`, a record of the targets of the classification problem that lead to this ideal split.
`variable`	A string denoting the type of such a split, `""` for oblique splits and a variable name for axis-parallel splits.
`cutleft`	A string expressing the left branch.
`cutright`	A string expressing the right branch.

Author(s)

A. Truong

---

oblique.tree *Fit an Oblique Tree to Classification Data*

Description

An oblique tree is grown by binary recursive partitioning using the response in the specified formula with oblique splits composed of linear combinations of terms from the right-hand-side.

Usage

```
oblique.tree(
formula,
data,
subset,
control = tree.control(nobs, ...),
method = "recursive.partition",
split.impurity = c("deviance", "gini"),
model = FALSE,
oblique.splits = c("only", "on", "off"),
variable.selection = c( "none",
"model.selection.aic",
"model.selection.bic",
"lasso.aic",
"lasso.bic"),
...)
```

Arguments

formula	A formula expression. The left-hand-side (response) should be a factor. 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.
data	A data frame in which to interpret formula and subset.
subset	An expression specifying the subset of cases to be used.
control	A list as returned by tree.control.
method	A character string specifying the method to use. The only other useful value is "model.frame".
split.impurity	Splitting criterion to use.
model	A model frame containing a response and predictors that can be used in place of formula, data and subset to allow direct specification of the problem.
oblique.splits	If and how oblique splits should be used during tree-growth. only grows trees that only consider oblique splits, on grows those that consider oblique and axis-parallel splits simultaneously and off grows trees that only consider axis-parallel splits.
variable.selection	If and how concise oblique splits should be found during tree-growth. none grows oblique trees using full oblique splits, model.selection.aic performs variable selection with AIC from the full model upon the best ideal split, model.selection.bic similarly for BIC, lasso.aic applies L1 regularization from the full model and chooses the penalization parameter with AIC and lasso.bic similarly for BIC.
...	Additional arguments that are passed to tree.control. Normally used for mincut, minsize or mindev.

Details

An oblique tree is grown by binary recursive partitioning using the response in the specified formula and by choosing splits composed of terms from the right-hand-side. Where categorical attributes are considered levels of unordered factors are divided into two non-empty groups, where axis-parallel splits are considered numeric variables are divided into $X < a$ and $X \geq a$ and where oblique splits are considered numeric variables are divided into $\sum aX < c$ and $\sum aX \geq c$. The split that maximizes the reduction in impurity is chosen and the process repeated. Splitting continues until the terminal nodes are either pure, sufficiently pure or too small to be split.

When growing oblique trees, $2^{R-1} - 1$ logistic regression problems need to be (where R is the number of residual classes at a node). If observations come from more than 20 classes this approach will be slow.

Value

An object of class c("oblique.tree","tree") is returned with components

frame	A data frame with a row for each node and row.names giving the node numbers. The columns include var, the variable used to perform each split (where "" denotes an oblique split and "<leaf>" a terminal node), n, the number of cases reaching that node, dev the deviance of the node, yval, the class associated to

that node, split, a two-column matrix of the labels for left and right splits at the node and yprob, a matrix of fitted probabilities for each response level.

where	A vector indicating the row number of the frame detailing the node to which each case is assigned.
terms	The terms of the formula.
call	The matched call to oblique.tree.
y	Predicted classes of each observation by the tree (this differs from the implementation in the **tree** package where y is instead used to denote the actual classes).

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

Author(s)

A. Truong

References

Truong. A (2009) *Fast Growing and Interpretable Oblique Trees via Probabilistic Models*

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

See Also

tree.control in the **tree** package, predict.oblique.tree, prune.oblique.tree, trim.oblique.tree

Examples

```
#create the augmented crabs dataset
data(crabs, package = "MASS")
aug.crabs.data <- data.frame( g=factor(rep(1:4,each=50)),
predict(princomp(crabs[,4:8]))[,2:3])

plot( aug.crabs.data[,-1],type="n")
text( aug.crabs.data[,-1],
col=as.numeric(aug.crabs.data[,1]),
labels=as.numeric(aug.crabs.data[,1]))

#grow a full oblique tree
ob.tree <- oblique.tree(formula = g~.,
data = aug.crabs.data,
oblique.splits = "only")
plot(ob.tree);text(ob.tree)
```

`oblique.tree.complexity` *Internal Function that Quantifies the Complexity of Oblique Trees*

Description

Quantifies the complexity of subtrees of oblique trees

Usage

```
oblique.tree.complexity(
tree,
subtree.internal.node.names)
```

Arguments

`tree`	Fitted model object of class `oblique.tree`. This is assumed to be the result of some function that produces an object with the same named components as that returned by `oblique.tree`.
`subtree.internal.node.names`	A numeric vector containing the names of internal nodes of such a subtree.

Details

$$R_\alpha(T) = R(T) + \alpha size(T)$$

When pruning and trimming trees, $size(T)$ must be evaluated. Though counting the number of leaves (tree size) gives an good indication of the complexity of axis-parallel trees, it does not take into account the complexity of each split in oblique trees. This complexity measure generalizes tree size by inflating the cost of each leaf up from 1 for more complex branches. The cost of each leaf is defined to be the number of attributes used along its branch divided the by the number of tests (which coincides with tree size in the case of axis-parallel trees)

Value

A measure of complexity for oblique trees.

Author(s)

A. Truong

References

Truong. A (2009) *Fast Growing and Interpretable Oblique Trees via Probabilistic Models*

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

See Also

oblique.tree.

Examples

```
#consider various tree on the Pima Indian dataset
data(Pima.tr, package = "MASS")
ob.tree.only <- oblique.tree(formula = type~.,
data = Pima.tr,
oblique.splits = "only")
ob.tree.on <- oblique.tree(formula = type~.,
data = Pima.tr,
oblique.splits = "on")
ob.tree.off <- oblique.tree(formula = type~.,
data = Pima.tr,
oblique.splits = "off")
op <- par(mfrow=c(1,3))
plot(ob.tree.only);text(ob.tree.only);title(main="Oblique Splits Only")
plot(ob.tree.on);text(ob.tree.on);title(main="Oblique Splits On")
plot(ob.tree.off);text(ob.tree.off);title(main="Oblique Splits Off")
par(op)

#calculate the complexity of a subtree
oblique.tree:::oblique.tree.complexity(
tree = ob.tree.only,
subtree.internal.node.names = c(12,13,7))

#calculate the complexity of each tree
oblique.tree:::oblique.tree.complexity(
tree = ob.tree.only,
subtree.internal.node.names
        = as.numeric(row.names(ob.tree.only$frame)[ob.tree.only$frame$var == "<leaf>"]))
oblique.tree:::oblique.tree.complexity(
tree = ob.tree.on,
subtree.internal.node.names
= as.numeric(row.names(ob.tree.on$frame)[ob.tree.on$frame$var == "<leaf>"]))
oblique.tree:::oblique.tree.complexity(
tree = ob.tree.off,
subtree.internal.node.names
= as.numeric(row.names(ob.tree.off$frame)[ob.tree.off$frame$var == "<leaf>"]))
```

---

oblique.tree.prune.nodes

Internal Function that Prunes Oblique Tree Objects

Description

Prunes entire subtrees off objects of class c("oblique.tree","tree").

Usage

```
oblique.tree.prune.nodes(
tree,
list.of.node.names.to.prune)
```

Arguments

`tree`	Fitted model object of class `oblique.tree`. This is assumed to be the result of some function that produces an object with the same named components as that returned by `oblique.tree`.
`list.of.node.names.to.prune`	A list containing subtrees that are to be pruned. Each subtree is represented by a vector of its node names starting from its root to its the last node.

Details

The data structure representing objects of class `c("oblique.tree","tree")` need to be carefully manipulated to reflect the pruning of nodes. Rows of `tree$frame` are removed and `tree$frame$var`, `tree$frame$splits`, `tree$where`, `tree$y` `tree$details` need to be updated.

Value

An object of class `c("oblique.tree","tree")` is return.

Author(s)

A. Truong

See Also

`oblique.tree`.

Examples

```
#grow a tree on the Pima Indian dataset
data(Pima.tr, package = "MASS")
ob.tree <- oblique.tree(formula = type~.,
data = Pima.tr,
oblique.splits = "on")
plot(ob.tree);title(main="Oblique Tree")

#prune 1 subtree
plot( oblique.tree:::oblique.tree.prune.nodes(
tree = ob.tree,
list.of.node.names.to.prune = list(c(5,10,11,22,23,46,47,94,95))
)
);title(main="Prune one Subtree")
```

plot.trim.sequence *Plot a Trim Sequence*

Description

Allows the user to plot a trim sequence.

Usage

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

Arguments

x	object of class trim.sequence. This is assumed to be the result of some function that produces an object with the same named components (comp, deviance, h) as that returned by trim.oblique.tree.
order	of comp on the plot. Use "decreasing" for a natural ordering of the complexity of tree and the amount of trimming. Only the first character is needed.
type, ylim, ...	graphical parameters.

Details

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

Examples

```
#grow an oblique tree
data(Pima.tr, package="MASS")
ob.tree <- oblique.tree( type~.,
Pima.tr,
oblique.splits="only",
variable.selection="model.selection.aic")

#look at the tree
plot(ob.tree);text(ob.tree)

#and its trim sequence
plot(trim.oblique.tree(ob.tree))
```

`predict.glmpath` *Makes predictions at particular points along the fitted glmpath*

Description

This function makes predictions at particular points along the fitted `glmpath`. The linear predictor, estimated response, log-likelihood, or the coefficients can be computed.

Usage

```
predict.glmpath(object, newx, newy, s, type = c("link", "response",
                "loglik", "coefficients"), mode = c("step",
                "norm.fraction", "norm", "lambda.fraction", "lambda"),
                weight = NULL, offset = NULL, eps = .Machine$double.eps, ...)
```

Arguments

`object`	a `glmpath` object
`newx`	a matrix of features at which the predictions are made. If `type=link`, `type=response`, or `type=loglik`, `newx` is required.
`newy`	a vector of responses corresponding to `newx`. If `type=loglik`, `newy` is required.
`s`	the values of `mode` at which the predictions are made
`type`	If `type=link`, the linear predictors are returned; if `type=response`, the estimated responses are returned; if `type=loglik`, the log-likelihoods are returned, and if `type=coefficients`, the coefficients are returned. The coefficients for the initial input variables are returned (rather than the standardized coefficients). Default is `link`.
`mode`	what `mode=s` refers to. If `mode=step`, `s` is the number of steps taken; if `mode=norm.fraction`, `s` is the fraction of the L1 norm of the standardized coefficients (with respect to the largest norm); if `mode=norm`, `s` is the L1 norm of the standardized coefficients; if `mode=lambda.fraction`, `s` is the fraction of $\log(\lambda)$; and if `mode=lambda`, `s` is λ. Default is `step`.
`weight`	an optional vector of weights for observations. `weight` is effective only if `type=loglik`.
`offset`	
`eps`	an effective zero
`...`	other options for the prediction

Author(s)

Mee Young Park and Trevor Hastie

References

Mee Young Park and Trevor Hastie (2006) L1 Regularization Path Algorithm for Generalized Linear Models - available at the authors' websites, http://stat.stanford.edu/~mypark or http://stat.stanford.edu/~hastie/pub.htm.

See Also

cv.glmpath, glmpath, plot.glmpath

`predict.oblique.tree` *Predictions from Fitted Oblique Tree Object*

Description

Returns predictions from a fitted `oblique.tree` object.

Usage

```
## S3 method for class 'oblique.tree'
predict(
object,
newdata,
type = c("vector", "tree", "class", "where"),
eps = 1e-3,
update.tree.predictions = FALSE,
...)
```

Arguments

`object`	Fitted model object of class `oblique.tree`. This is assumed to be the result of some function that produces an object with the same named components as that returned by `oblique.tree`.
`newdata`	Data frame containing the values at which predictions are required. The predictors referred to in the right side of `formula(object)` must be present by name in `newdata`. If missing, fitted values are returned.
`type`	Character string denoting how predictions are to be returned, i.e. class probabilities (default), a tree object, class predictions or predictions to leaf nodes.
`eps`	A lower bound for the probabilities, used if events of predicted probability zero occur in `newdata` when predicting a tree.
`update.tree.predictions`	Logical vector denoting whether tree predictions (`frame$yval, frame$yprob, $where, $y` etc) are updated when `newdata` is provided.
`...`	Further arguments passed to or from other methods.

Details

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

Value

If `type = "vector"`: a matrix of predicted class probabilities is returned. This object is obtained by dropping observations down `object`.

If `type = "tree"`: an object of class `c("oblique.tree","tree")` is returned with new values for `frame$n` and `frame$dev`.

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

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

Author(s)

A. Truong

References

Truong. A (2009) *Fast Growing and Interpretable Oblique Trees via Probabilistic Models*

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

See Also

predict, oblique.tree.

Examples

```
#grow an oblique tree to the Pima Indian dataset
data(Pima.tr, package = "MASS")
ob.tree <- oblique.tree(formula = type~.,
data = Pima.tr,
oblique.splits = "on")
plot(ob.tree);text(ob.tree);title(main="Oblique Tree")

#predictions to in-sample data
#class probabilities for each observation
predict(ob.tree,type="vector")
#the tree itself
predict(ob.tree,type="tree")
#class predictions for each observation
predict(ob.tree,type="class")
#the leaf where each observation falls
predict(ob.tree,type="where")

#predictions to out-of-sample data
```

```
data(Pima.te, package = "MASS")
#class probabilities for each observation
predict(ob.tree,newdata=Pima.te,type="vector")
#the tree itself
predict(ob.tree,newdata=Pima.te,type="tree")
#class predictions for each observation
predict(ob.tree,newdata=Pima.te,type="class")
#the leaf where each observation falls
predict(ob.tree,newdata=Pima.te,type="where")
```

`prune.oblique.tree` *Cost-complexity Pruning of Oblique Tree Object*

Description

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

Usage

```
prune.oblique.tree(
tree,
k = NULL,
newdata,
prune.impurity = c("deviance", "misclass"),
penalty = c("complexity", "size"),
eps = 1e-3)
```

Arguments

`tree`	Fitted model object of class `oblique.tree`. This is assumed to be the result of some function that produces an object with the same named components as that returned by `oblique.tree`.
`k`	Cost-complexity parameter defining either a specific subtree of `tree` (`k` a scalar) or the (optional) sequence of subtrees minimizing the cost-complexity measure (`k` a vector). If missing, `k` is determined algorithmically.
`newdata`	Data frame upon which the sequence of cost-complexity subtrees is evaluated. If missing, the data used to grow the tree is used.
`prune.impurity`	Character string denoting the measure of node heterogeneity used to guide cost-complexity pruning. The default is `deviance` and the alternative is `misclass` (number of misclassifications or total loss).
`penalty`	Character string denoting the measure of tree complexity used to guide cost-complexity pruning. The default is `complexity` (that considers the complexity of splits used throughout the tree) and the alternative is `size` (that counts the number of leaves of the tree).
`eps`	A lower bound for the probabilities, used to compute deviances if events of predicted probability zero occur in `newdata`.

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.

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 tree sequence and deviances or losses calculated by comparing the supplied response to the prediction. 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 tree sequence. An additional axis displays the values of the cost-complexity parameter at each subtree.

Value

If k is supplied and is a scalar, an object of class `c("oblique.tree","tree")` is returned that minimizes the cost-complexity measure for that k. If k is a vector, an object of class `tree.sequence` is returned. The object contains the following components:

`size`	The complexity of each tree in the cost-complexity pruning sequence.
`deviance`	Total deviance of each tree in the cost-complexity pruning sequence.
`k`	The value of the cost-complexity pruning parameter of each tree in the sequence.

Author(s)

A. Truong

References

Truong. A (2009) *Fast Growing and Interpretable Oblique Trees via Probabilistic Models*

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

See Also

`oblique.tree`.

Examples

```
#grow an mixture tree on the Pima Indian dataset
data(Pima.tr, package = "MASS")
ob.tree <- oblique.tree(formula = type~.,
data = Pima.tr,
oblique.splits = "on")
plot(ob.tree);text(ob.tree);title(main="Mixture Tree")

#examine the tree sequence
tree.seq <- prune.oblique.tree( tree = ob.tree)
print(tree.seq);plot(tree.seq)

#examine test error over the tree sequence
```

```
data(Pima.te, package = "MASS")
tree.seq <- prune.oblique.tree( tree = ob.tree,
newdata = Pima.te)
print(tree.seq);plot(tree.seq)

#deviance is least when k = 8.148267
pruned <- prune.oblique.tree( tree = ob.tree,
k = 9)
plot(pruned);text(pruned);title(main="Pruned Tree")
```

`tree.impurity` *Internal Function that Determines the Impurity of a Subtree*

Description

Determines the impurity of a subtree.

Usage

```
  tree.impurity(
yprob,
number.of.observations.at.leaves,
leaf.classes,
impurity.measure = c("deviance","misclass"))
```

Arguments

`yprob`	A matrix of predicted class probabilities of leaves of the subtree considered in order of tree-growth (left to right).
`number.of.observations.at.leaves`	A vector denoting the number of observations at each of these leaves.
`leaf.classes`	A vector denoting the classes associated to these leaves.
`impurity.measure`	Character string denoting the measure of node heterogeneity to be used to guide cost-complexity pruning/trimming.

Details

$$R_\alpha(T) = R(T) + \alpha size(T)$$

When pruning and trimming trees, $R(T)$ must be evaluated. Extracted arguments from `oblique.tree` objects is easiest.

Value

The impurity of the subtree.

Author(s)

A. Truong

References

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

See Also

oblique.tree.

Examples

```
#grow a tree on the Pima Indian dataset
data(Pima.tr, package = "MASS")
ob.tree <- oblique.tree(formula = type~.,
data = Pima.tr,
oblique.splits = "on")

#deviance of entire tree
subtree.leaves <- ob.tree$frame$var == "<leaf>"
oblique.tree:::tree.impurity(
yprob  = ob.tree$frame$yprob[subtree.leaves,,drop=FALSE],
number.of.observations.at.leaves  = ob.tree$frame$n[subtree.leaves],
leaf.classes  = ob.tree$frame$yval[subtree.leaves],
impurity.measure  = "deviance")
```

---

trim.oblique.tree *Trims Oblique Splits of Fitted Oblique Tree Objects*

Description

Determines a sequence of concise subtrees of the supplied tree by recursively "trimming" off the least important attributes used in oblique splits.

Usage

```
trim.oblique.tree(
tree,
best = NULL,
newdata,
trim.impurity = c("deviance", "misclass"),
trim.depth = c("partial", "complete"),
eps = 1e-3)
```

Arguments

tree	Fitted model object of class oblique.tree. This is assumed to be the result of some function that produces an object with the same named components as that returned by oblique.tree.
best	Requests the complexity (i.e. 1 + number of attributes used throughout the tree) of the concise subtree of tree to return (best a scalar) or a (optional) sequence of concise subtrees (best a vector). If missing, best is determined algorithmically. If there is no tree in the sequence of the requested size, the next largest is returned.
newdata	Data frame upon which the sequence of cost-complexity subtrees is evaluated. If missing, the data used to grow the tree is used.
trim.impurity	Character string denoting the measure of node heterogeneity used to guide tree trimming. The default is deviance and the alternative is misclass (number of misclassifications or total loss).
trim.depth	A character string denoting if oblique splits should be trimmed towards axis-parallel splits partial or to the constant predictor complete.
eps	A lower bound for the probabilities, used to compute deviances if events of predicted probability zero occur in newdata.

Details

Determines a sequence of concise subtrees of the supplied tree by recursively "trimming" its splits, based upon the cost-complexity measure.

If best 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 trim sequence and deviances or losses calculated by comparing the supplied response to the prediction. 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 trim sequence. An additional axis displays the values of the cost-complexity parameter at each subtree.

Value

If best is a scalar, a c("oblique.tree","tree") object of size best is returned. Otherwise an object of class c("trim", "trim.sequence") is returned. The object contains the following components:

comp	The complexity of each tree in the cost-complexity pruning sequence.
dev	Total deviance of each tree in the cost-complexity pruning sequence.
h	The value of the cost-complexity pruning parameter of each tree in the sequence.

Author(s)

A. Truong

References

Truong. A (2009) *Fast Growing and Interpretable Oblique Trees via Probabilistic Models*

See Also

oblique.tree.

Examples

```
#grow a tree on the Pima Indian dataset
data(Pima.tr, package = "MASS")
ob.tree <- oblique.tree(formula = type~.,
data = Pima.tr,
oblique.splits = "only")
plot(ob.tree);text(ob.tree);title(main="Full Oblique Tree")

#partially trimming
#examine the tree sequence
trim.seq <- trim.oblique.tree( tree = ob.tree)
print(trim.seq);plot(trim.seq)

#examine test error over the trim sequence
data(Pima.te, package = "MASS")
trim.seq <- trim.oblique.tree( tree = ob.tree,
newdata = Pima.te)
print(trim.seq);plot(trim.seq)

#deviance is least when best = 7
p.trimmed <- trim.oblique.tree( tree = ob.tree,
best = 7)
plot(p.trimmed);text(p.trimmed);title(main="Partially Trimmed Tree")

#complete trimming
#examine the tree sequence
trim.seq <- trim.oblique.tree( tree = ob.tree,
trim.depth = "complete")
print(trim.seq);plot(trim.seq)

#examine test error over the trim sequence
data(Pima.te, package = "MASS")
trim.seq <- trim.oblique.tree( tree = ob.tree,
trim.depth = "complete",
newdata = Pima.te)
print(trim.seq);plot(trim.seq)

#deviance is least when best = 9
c.trimmed <- trim.oblique.tree( tree = ob.tree,
best = 9)
plot(c.trimmed);text(c.trimmed);title(main="Completely Trimmed Tree")
```

Index