

Package ‘dbarts’

April 18, 2018

Version 0.9-3

Date 2018-04-17

Title Discrete Bayesian Additive Regression Trees Sampler

Depends R (>= 3.1-0)

Imports stats, methods, graphics, parallel

Suggests testthat (>= 0.9-0)

Description Fits Bayesian additive regression trees (BART; Chipman, George, and McCulloch (2010) <doi:10.1214/09-AOAS285>) while allowing the updating of predictors or response so that BART can be incorporated as a conditional model in a Gibbs/MH sampler. Also serves as a drop-in replacement for package 'BayesTree'.

License GPL (>= 2)

NeedsCompilation yes

Biarch yes

URL <https://github.com/vdorie/dbarts>

BugReports <https://github.com/vdorie/dbarts/issues>

Author Vincent Dorie [aut, cre],
Hugh Chipman [aut],
Robert McCulloch [aut]

Maintainer Vincent Dorie <vdorie@gmail.com>

Repository CRAN

Date/Publication 2018-04-18 07:59:22 UTC

R topics documented:

bart	2
dbarts	7
dbartsControl	8
dbartsData	10
dbartsSampler-class	11
guessNumCores	13

makeModelMatrixFromDataFrame	13
pdbart	15
rbart	18
xbart	20

Index	24
--------------	-----------

bart	<i>Bayesian Additive Regression Trees</i>
------	---

Description

BART is a Bayesian “sum-of-trees” model in which each tree is constrained by a prior to be a weak learner.

- For numeric response $y = f(x) + \epsilon$, where $\epsilon \sim N(0, \sigma^2)$.
- For binary response y , $P(Y = 1 | x) = \Phi(f(x))$, where Φ denotes the standard normal cdf (probit link).

Usage

```
bart(x.train, y.train, x.test = matrix(0.0, 0, 0),
     sigest = NA, sigdf = 3, sigquant = 0.90,
     k = 2.0,
     power = 2.0, base = 0.95,
     binaryOffset = 0.0, weights = NULL,
     ntree = 200,
     ndpost = 1000, nskip = 100,
     printevery = 100, keepevery = 1, keeptrainfits = TRUE,
     usequants = FALSE, numcut = 100, printcutoffs = 0,
     verbose = TRUE, nchain = 1, nthread = 1, combinechains = TRUE,
     keeptrees = FALSE, keepcall = TRUE)

bart2(formula, data, test, subset, weights, offset, offset.test = offset,
     sigest = NA_real_, sigdf = 3.0, sigquant = 0.90,
     k = 2.0,
     power = 2.0, base = 0.95,
     n.trees = 75L,
     n.samples = 500L, n.burn = 500L,
     n.chains = 4L, n.threads = min(guessNumCores(), n.chains), combineChains = FALSE,
     n.cuts = 100L, useQuantiles = FALSE,
     n.thin = 1L, keepTrainingFits = TRUE,
     printEvery = 100L, printCutoffs = 0L,
     verbose = TRUE,
     keepTrees = TRUE, keepCall = TRUE, ...)

## S3 method for class 'bart'
plot(x,
```

```

plquants = c(0.05, 0.95), cols = c('blue', 'black'),
...)

## S3 method for class 'bart'
predict(object, test, offset.test, combineChains, ...)

```

Arguments

<code>x.train</code>	Explanatory variables for training (in sample) data. May be a matrix or a data frame, with rows corresponding to observations and columns to variables. If a variable is a factor in a data frame, it is replaced with dummies. Note that q dummies are created if $q > 2$ and one dummy is created if $q = 2$, where q is the number of levels of the factor.
<code>y.train</code>	Dependent variable for training (in sample) data. If <code>y.train</code> is numeric a continuous response model is fit (normal errors). If <code>y.train</code> is a binary factor or has only values 0 and 1, then a binary response model with a probit link is fit.
<code>x.test</code>	Explanatory variables for test (out of sample) data. Should have same column structure as <code>x.train</code> . <code>bart</code> will generate draws of $f(x)$ for each x which is a row of <code>x.test</code> .
<code>sigest</code>	For continuous response models, an estimate of the error variance, σ^2 , used to calibrate an inverse-chi-squared prior used on that parameter. If not supplied, the least-squares estimate is derived instead. See <code>sigquant</code> for more information. Not applicable when y is binary.
<code>sigdf</code>	Degrees of freedom for error variance prior. Not applicable when y is binary.
<code>sigquant</code>	The quantile of the error variance prior that the rough estimate (<code>sigest</code>) is placed at. The closer the quantile is to 1, the more aggressive the fit will be as you are putting more prior weight on error standard deviations (σ) less than the rough estimate. Not applicable when y is binary.
<code>k</code>	For numeric y , k is the number of prior standard deviations $E(Y x) = f(x)$ is away from ± 0.5 . The response (<code>y.train</code>) is internally scaled to range from -0.5 to 0.5 . For binary y , k is the number of prior standard deviations $f(x)$ is away from ± 3 . In both cases, the bigger k is, the more conservative the fitting will be.
<code>power</code>	Power parameter for tree prior.
<code>base</code>	Base parameter for tree prior.
<code>binaryOffset</code>	Used for binary y . When present, the model is $P(Y = 1 x) = \Phi(f(x) + \text{binaryOffset})$, allowing fits with probabilities shrunk towards values other than 0.5.
<code>weights</code>	An optional vector of weights to be used in the fitting process. When present, BART fits a model with observations $y x \sim N(f(x), \sigma^2/w)$, where $f(x)$ is the unknown function.
<code>ntree</code>	The number of trees in the sum-of-trees formulation.
<code>ndpost</code>	The number of posterior draws after burn in, <code>ndpost / keepevery</code> will actually be returned.
<code>nskip</code>	Number of MCMC iterations to be treated as burn in.

<code>printevery</code>	As the MCMC runs, a message is printed every <code>printevery</code> draws.
<code>keepevery</code>	Every <code>keepevery</code> draw is kept to be returned to the user. Useful for “thinning” samples.
<code>keeptrainfits</code>	If TRUE the draws of $f(x)$ for x corresponding to the rows of <code>x.train</code> are returned.
<code>usequants</code>	When TRUE, determine tree decision rules using estimated quantiles derived from the <code>x.train</code> variables. When FALSE, splits are determined using values equally spaced across the range of a variable. See details for more information.
<code>numcut</code>	The maximum number of possible values used in decision rules (see <code>usequants</code> , details). If a single number, it is recycled for all variables; otherwise must be a vector of length equal to <code>ncol(x.train)</code> . Fewer rules may be used if a covariate lacks enough unique values.
<code>printcutoffs</code>	The number of cutoff rules to printed to screen before the MCMC is run. Given a single integer, the same value will be used for all variables. If 0, nothing is printed.
<code>verbose</code>	Logical; if FALSE suppress printing.
<code>nchain</code>	Integer specifying how many independent tree sets and fits should be calculated.
<code>nthread</code>	Integer specifying how many threads to use. Depending on the CPU architecture, using more than the number of chains can degrade performance for small/medium data sets. As such some calculations may be executed single threaded regardless.
<code>combinechains</code>	Logical; if TRUE, samples will be returned in arrays of dimensions equal to <code>nchain × ndpost × number of observations</code> .
<code>keeptrees</code>	Logical; must be TRUE in order to use <code>predict</code> with the result of a <code>bart</code> fit.
<code>keepcall</code>	Logical; if FALSE, returned object will have <code>call</code> set to <code>call("NULL")</code> , otherwise the call used to instantiate BART.
<code>formula</code>	The same as <code>x.train</code> , the name reflecting that a formula object can be used instead.
<code>data</code>	The same as <code>y.train</code> , the name reflecting that a data frame can be specified when a formula is given instead.
<code>test</code>	The same as <code>x.train</code> . Can be missing.
<code>subset</code>	A vector of logicals or indices used to subset of the data. Can be missing.
<code>offset</code>	The same as <code>binaryOffset</code> . Can be missing.
<code>offset.test</code>	A vector of offsets to be used with test data, in case it is different than the training offset. If <code>offset</code> is missing, defaults to NULL.
<code>object</code>	An object of class <code>bart</code> , returned from either the function <code>bart</code> or <code>bart2</code> .
<code>n.trees, n.samples, n.burn, n.chains, n.threads, combineChains, n.cuts, n.thin, keepTrainingFits, useQuantiles</code>	Same as their counterparts for <code>bart</code> .
<code>x</code>	Object of class <code>bart</code> , returned by function <code>bart</code> , which contains the information to be plotted.
<code>plquants</code>	In the plots, beliefs about $f(x)$ are indicated by plotting the posterior median and a lower and upper quantile. <code>plquants</code> is a double vector of length two giving the lower and upper quantiles.

cols	Vector of two colors. First color is used to plot the median of $f(x)$ and the second color is used to plot the lower and upper quantiles.
...	Additional arguments passed on to plot or dbartsControl, respectively. Not used in predict.

Details

BART is an Bayesian MCMC method. At each MCMC iteration, we produce a draw from the joint posterior $(f, \sigma) | (x, y)$ in the numeric y case and just f in the binary y case.

Thus, unlike a lot of other modeling methods in R, bart does not produce a single model object from which fits and summaries may be extracted. The output consists of values $f^*(x)$ (and σ^* in the numeric case) where $*$ denotes a particular draw. The x is either a row from the training data (`x.train`) or the test data (`x.test`).

Decision Rules: Decision rules for any tree are of the form $x \leq c$ vs. $x > c$ for each ' x ' corresponding to a column of `x.train`. `usequants` determines the means by which the set of possible c is determined. If `usequants` is TRUE, then the c are a subset of the values interpolated half-way between the unique, sorted values obtained from the corresponding column of `x.train`. If `usequants` is FALSE, the cutoffs are equally spaced across the range of values taken on by the corresponding column of `x.train`.

The number of possible values of c is determined by `numcut`. If `usequants` is FALSE, `numcut` equally spaced cutoffs are used covering the range of values in the corresponding column of `x.train`. If `usequants` is TRUE, then for a variable the minimum of `numcut` and one less than the number of unique elements for that variable are used.

Predict: Using `predict` with a bart object requires that it be fitted with the option `keeptrees/keepTrees` as TRUE. Keeping the trees for a fit can require a sizeable amount of memory.

Value

`bart` returns a list assigned class `bart`. For applicable quantities, `ndpost / keepevery` samples are returned. In the numeric y case, the list has components:

<code>yhat.train</code>	A array/matrix of posterior samples. The (i, j, k) value is the j th draw of the posterior of f evaluated at the k th row of <code>x.train</code> (i.e. $f^*(x_k)$) corresponding to chain i . When <code>nchain</code> is one or <code>combinechains</code> is TRUE, the result is a collapsed down to a matrix.
<code>yhat.test</code>	Same as <code>yhat.train</code> but now the x s are the rows of the test data.
<code>yhat.train.mean</code>	Vector of means of <code>yhat.train</code> across columns and chains, with length equal to the number of training observations.
<code>yhat.test.mean</code>	Vector of means of <code>yhat.test</code> across columns and chains.
<code>sigma</code>	Matrix of posterior samples of <code>sigma</code> , the residual/error standard deviation. Dimensions are equal to the number of chains times the numbers of samples unless <code>nchain</code> is one or <code>combinechains</code> is TRUE.
<code>first.sigma</code>	Burn-in draws of <code>sigma</code> .

<code>varcount</code>	A matrix with number of rows equal to the number of kept draws and each column corresponding to a training variable. Contains the total count of the number of times that variable is used in a tree decision rule (over all trees).
<code>sigest</code>	The rough error standard deviation (σ) used in the prior.
<code>y</code>	The input dependent vector of values for the dependent variable. This is used in <code>plot.bart</code> .
<code>fit</code>	Optional sampler object which stores the values of the tree splits. Required for using <code>predict</code> .

In the binary y case, the returned list has the components `yhat.train`, `yhat.test`, and `varcount` as above. In addition the list has a `binaryOffset` component giving the value used.

Note that in the binary y , case `yhat.train` and `yhat.test` are $f(x) + \text{binaryOffset}$. For draws of the probability $P(Y = 1|x)$, apply the normal cdf (`pnorm`) to these values.

The `plot` method sets `mfrow` to `c(1,2)` and makes two plots. The first plot is the sequence of kept draws of σ including the burn-in draws. Initially these draws will decline as BART finds fit and then level off when the MCMC has burnt in. The second plot has y on the horizontal axis and posterior intervals for the corresponding $f(x)$ on the vertical axis.

Author(s)

Hugh Chipman: <hugh.chipman@gmail.com>
 Robert McCulloch: <robert.mcculloch1@gmail.com>
 Vincent Dorie: <vdorie@gmail.com>

References

Chipman, H., George, E., and McCulloch, R. (2009) BART: Bayesian Additive Regression Trees.
 Chipman, H., George, E., and McCulloch R. (2006) Bayesian Ensemble Learning. Advances in Neural Information Processing Systems 19, Scholkopf, Platt and Hoffman, Eds., MIT Press, Cambridge, MA, 265-272.
 both of the above at: <http://www.rob-mcculloch.org>
 Friedman, J.H. (1991) Multivariate adaptive regression splines. *The Annals of Statistics*, **19**, 1–67.

See Also

[pdbart](#)

Examples

```
## simulate data (example from Friedman MARS paper)
## y = f(x) + epsilon , epsilon ~ N(0, sigma)
## x consists of 10 variables, only first 5 matter

f <- function(x) {
  10 * sin(pi * x[,1] * x[,2]) + 20 * (x[,3] - 0.5)^2 +
  10 * x[,4] + 5 * x[,5]
```

```

}

set.seed(99)
sigma <- 1.0
n     <- 100

x <- matrix(runif(n * 10), n, 10)
Ey <- f(x)
y <- rnorm(n, Ey, sigma)

## run BART
set.seed(99)
bartFit <- bart(x, y)

plot(bartFit)

## compare BART fit to linear matter and truth = Ey
lmFit <- lm(y ~ ., data.frame(x, y))

fitmat <- cbind(y, Ey, lmFit$fitted, bartFit$yhat.train.mean)
colnames(fitmat) <- c('y', 'Ey', 'lm', 'bart')
print(cor(fitmat))

```

dbarts

Discrete Bayesian Additive Regression Trees Sampler

Description

Creates a sampler object for a given problem which fits a Bayesian Additive Regression Trees model. Internally stores state in such a way as to be mutable.

Usage

```

dbarts(formula, data, test, subset, weights, offset, offset.test = offset,
       verbose = FALSE, n.samples = 800L,
       tree.prior = cgm, node.prior = normal, resid.prior = chisq,
       control = dbartsControl(), sigma = NA_real_)

```

Arguments

formula	An object of class <code>formula</code> following an analogous model description syntax as <code>lm</code> . For backwards compatibility, can also be the <code>bart</code> matrix <code>x.train</code> .
data	An optional data frame, list, or environment containing predictors to be used with the model. For backwards compatibility, can also be the <code>bart</code> vector <code>y.train</code> .
test	An optional matrix or data frame with the same number of predictors as <code>data</code> , or <code>formula</code> in backwards compatibility mode. If column names are present, a matching algorithm is used.

subset	An optional vector specifying a subset of observations to be used in the fitting process.
weights	An optional vector of weights to be used in the fitting process. When present, BART fits a model with observations $y \mid x \sim N(f(x), \sigma^2/w)$, where $f(x)$ is the unknown function.
offset	An optional vector specifying an offset from 0 for the relationship between the underlying function, $f(x)$, and the response y . Only is useful for binary responses, in which case the model fit is to assume $P(Y = 1 \mid X = x) = \Phi(f(x) + \text{offset})$, where Φ is the standard normal cumulative distribution function.
offset.test	The equivalent of <code>offset</code> for test observations. Will attempt to use <code>offset</code> when applicable.
verbose	A logical determining if additional output is printed to the console. See dbartsControl .
n.samples	A positive integer setting the default number of posterior samples to be returned for each run of the sampler. Can be overridden at run-time. See dbartsControl .
tree.prior	An expression of the form <code>cgm</code> or <code>cgm(power, base)</code> setting the tree prior used in fitting.
node.prior	An expression of the form <code>normal</code> or <code>normal(k)</code> that sets the prior used on the averages within nodes.
resid.prior	An expression of the form <code>chisq</code> or <code>chisq(df, quant)</code> that sets the prior used on the residual/error variance.
control	An object inheriting from <code>dbartsControl</code> , created by the dbartsControl function.
sigma	A positive numeric estimate of the residual standard deviation. If NA, a linear model is used with all of the predictors to obtain one.

Details

“Discrete sampler” refers to that `dbarts` is implemented using [ReferenceClasses](#), so that there exists a mutable object constructed in C++ that is largely obscured from R. The `dbarts` function is the primary way of creating a [dbartsSampler](#), for which a variety of methods exist.

Value

A reference object of [dbartsSampler](#).

`dbartsControl`

Discrete Bayesian Additive Regression Trees Sampler Control

Description

Convenience function to create a control object for use with a `dbarts` sampler.

Usage

```
dbartsControl(verbose = FALSE, keepTrainingFits = TRUE, useQuantiles = FALSE,
              keepTrees = FALSE, n.samples = NA_integer_,
              n.cuts = 100L, n.burn = 200L, n.trees = 75L, n.chains = 4L,
              n.threads = guessNumCores(), n.thin = 1L, printEvery = 100L,
              printCutoffs = 0L, rngKind = "default", rngNormalKind = "default",
              updateState = TRUE)
```

Arguments

<code>verbose</code>	Logical controlling sampler output to console.
<code>keepTrainingFits</code>	Logical controlling whether or not training fits are returned when the sampler runs. These are always computed as part of the fitting procedure, so disabling will not substantially impact running time.
<code>useQuantiles</code>	Logical to determine if the empirical quantiles of a columns of predictors should be used to determine the tree decision rules. If FALSE, the rules are spaced uniformly throughout the range of covariate values.
<code>keepTrees</code>	A logical that determines whether or not trees are cached as they are sampled. In all cases, the current state of the sampler is stored as a single set of <code>n.trees</code> . When <code>keepTrees</code> is TRUE, a set of <code>n.trees * n.samples</code> trees are set aside and populated as the sampler runs. If the sampler is stopped and restarted, samples proceed from the previously stored tree, looping over if necessary.
<code>n.samples</code>	A non-negative integer giving the default number of samples to return each time the sampler is run. Generally specified by <code>dbarts</code> instead, and can be overridden on a per-use basis whenever the sampler is <code>run</code> .
<code>n.cuts</code>	A positive integer or integer vector giving the number of decision rules to be used for each given predictor. If of length less than the number of predictors, earlier values are recycled. If for any predictor more values are specified than are coherent, fewer may be used. See details for more information.
<code>n.burn</code>	A non-negative integer determining how many samples, if any, are thrown away at the beginning of a run of the sampler.
<code>n.trees</code>	A positive integer giving the number of trees used in the sum-of-trees formulation.
<code>n.chains</code>	A positive integer detailing the number of independent chains for the sampler to use.
<code>n.threads</code>	A positive integer controlling how many threads will be used for various internal calculations, as well as the number of chains. Internal calculations are highly optimized so that single-threaded performance tends to be superior unless the number of observations is very large (>10k), so that it is often not necessary to have the number of threads exceed the number of chains.
<code>n.thin</code>	A positive integer determining how many iterations the MCMC chain should jump on the decision trees alone before recording a sample. Serves to “thin” the samples against serial correlation. <code>n.samples</code> are returned regardless of the value of <code>n.thin</code> .

printEvery	If verbose is TRUE, every printEvery potential samples (after thinning) will issue a verbal statement. Must be a positive integer.
printCutoffs	A non-negative integer specifying how many of the decision rules for a variable are printed in verbose mode.
rngKind	Random number generator kind, as used in set.seed . For type "default", the built-in generator will be used if possible. Otherwise, will attempt to match the built-in generator's type. Success depends on the number of threads.
rngNormalKind	Random number generator normal kind, as used in set.seed . For type "default", the built-in generator will be used if possible. Otherwise, will attempt to match the built-in generator's type. Success depends on the number of threads and the rngKind.
updateState	Logical setting the default behavior for many sampler methods with regards to the immediate updating of the cached state of the object. A current, cached state is only useful when saving/loading the sampler.

Value

An object of class `dbartControl`.

See Also

[dbarts](#)

dbartsData

Discrete Bayesian Additive Regression Trees Sampler Data

Description

Convenience function to create a data object for use with a [dbarts](#) sampler.

Usage

```
dbartsData(formula, data, test, subset, weights, offset, offset.test = offset)
```

Arguments

formula, data, test, subset, weights, offset, offset.test

As in [dbarts](#). Retains backwards compatibility with [bart](#), so that formula/data can be a [formula/data.frame](#) pair, or a pair of `x.train/y.train` matrices/vector.

Value

An object of class `dbartData`.

See Also

[dbarts](#)

dbartsSampler-class *Class "dbartsSampler" of Discrete Bayesian Additive Regression Trees Sampler*

Description

A reference class object that contains a Bayesian Additive Regression Trees sampler in such a way that it can be modified, stopped, and started all while maintaining its own state.

Usage

```
## S4 method for signature 'dbartsSampler'
run(numBurnIn, numSamples, updateState = NA)
## S4 method for signature 'dbartsSampler'
sampleTreesFromPrior(updateState = NA)
## S4 method for signature 'dbartsSampler'
copy(shallow = FALSE)
## S4 method for signature 'dbartsSampler'
show()
## S4 method for signature 'dbartsSampler'
predict(x.test, offset.test)
## S4 method for signature 'dbartsSampler'
setControl(control)
## S4 method for signature 'dbartsSampler'
setModel(model)
## S4 method for signature 'dbartsSampler'
setData(data)
## S4 method for signature 'dbartsSampler'
setResponse(y, updateState = NA)
## S4 method for signature 'dbartsSampler'
setOffset(offset, updateState = NA)
## S4 method for signature 'dbartsSampler'
setPredictor(x, column, updateState = NA)
## S4 method for signature 'dbartsSampler'
setTestPredictor(x.test, column, updateState = NA)
## S4 method for signature 'dbartsSampler'
setTestPredictorAndOffset(x.test, offset.test, updateState = NA)
## S4 method for signature 'dbartsSampler'
setTestOffset(offset.test, updateState = NA)
## S4 method for signature 'dbartsSampler'
printTrees(treeNums)
## S4 method for signature 'dbartsSampler'
plotTree(treeNum, treePlotPars = list(nodeHeight = 12, nodeWidth = 40, nodeGap = 8), ...)
```

Arguments

numBurnIn A non-negative integer determining how many iterations the sampler should skip before storing results. If missing or NA, the default is filled in from the sampler's

	control object.
numSamples	A positive integer determining how many posterior samples should be returned. If missing or NA, the default is also filled in from the control object.
updateState	A logical determining if the local cache of the sampler's state should be updated after the completion of the run. If NA, the default is also filled in from the control object.
shallow	A logical determining if the copy should retain the underlying data of the sampler (TRUE) or have its own copies (FALSE).
control	An object inheriting from dbartsControl .
model	An object inheriting from dbartsModel .
data	An object inheriting from dbartsData .
y	A numeric response vector of length equal to that with which the sampler was created.
x	A numeric predictor vector of length equal to that with which the sampler was created. Can be an entire matrix of new number of rows for setTestPredictor .
x.test	A new matrix of test predictors, of the number of columns equal to that in the current model.
offset	A numeric vector of length equal to that with which the sampler was created, or NULL. If offset.test was set from offset , will attempt to update that as well.
offset.test	A numeric vector of length equal to that of the test matrix, or NULL. Can be missing for setTestPredictors .
column	An integer or character string vector specifying which column/columns of the predictor matrix is to be replaced. If missing, the entire matrix is substitute.
treeNums	An integer vector listing the indices of the trees to print.
treeNum	An integer listing the indices of the tree to plot.
treePlotPars	A list containing the number quantities nodeHeight , nodeWidth , and nodeGap , all of which control aspects of the resulting plot.
...	Extra arguments to plot .

Details

A `dbartsSampler` is a “mutable” object which contains information pertaining to fitting a Bayesian additive regression tree model. The sampler is first created and then, in a separate instruction, run or modified. In this way, MCMC samplers can be constructed with BART components filling arbitrary roles.

Value

For `run`, a named-list with contents `sigma`, `train`, `test`, and `varcount`.

For `setPredictor`, TRUE/FALSE depending on whether or not the operation was successful. The operation can fail if the new predictor results in a tree with an empty leaf-node. If only single columns were replaced, on the update is rolled-back so that the sampler remains in a valid state.

`predict` keeps the current test matrix in place and uses the current set of tree splits. It is intended that this function only be used when the `runMode` of [dbartsControl](#) is “fixedSamples”, since otherwise only a single set of trees are stored.

guessNumCores	<i>Guess Number of Cores</i>
---------------	------------------------------

Description

Attempts to guess the number of CPU ‘cores’, both physical and logical.

Usage

```
guessNumCores(logical = FALSE)
```

Arguments

`logical` A logical value. When FALSE, an estimate of the number of physical cores is returned. When TRUE, so-called "logical" cores as also included.

Details

Because of different definitions of cores used by different manufacturers, the distinction between logical and physical cores is not universally recognized. This function will attempt to use operating system definitions when available, which should usually match the CPU itself.

Value

An integer, or NA if no clear answer was obtained.

Author(s)

Vincent Dorie: <vdorie@gmail.com>.

makeModelMatrixFromDataFrame	<i>Make Model Matrix from Data Frame</i>
------------------------------	--

Description

Converts a data frame with numeric and factor contents into a matrix, suitable for use with [bart](#). Unlike in linear regression, factors containing more than two levels result in dummy variables being created for each level.

Usage

```
makeModelMatrixFromDataFrame(x, drop = TRUE)  
makeind(x, all = TRUE)
```

Arguments

x	Data frame of explanatory variables.
drop	Logical or list controlling whether or not columns that are constants or factor levels with no instances are omitted from the result. When a list, must be of length equal to x. Elements correspond to x according to: <ul style="list-style-type: none">• vector - single logical• matrix - vector of logicals, one per column• factor - table of factor levels to be referenced; levels with counts of 0 are to be dropped
all	Not currently implemented.

Details

Note that if you have train and test data frames, it may be best to `rbind` the two together, apply `makeModelMatrixFromDataFrame` to the result, and then pull them back apart. Alternatively, save the `drop` attribute used in creating the train data and use it when creating a matrix from the test data. Example given below.

Value

A matrix with columns corresponding to the elements of the data frame. If `drop = TRUE` or is a list, the attribute `drop` on the result is set to the list used when creating the matrix.

Author(s)

Vincent Dorie: <vdorie@gmail.com>.

Examples

```
iv <- 1:10
rv <- runif(10)
f <- factor(rep(seq.int(3), c(4L, 4L, 2L)),
            labels = c("alice", "bob", "charlie"))
df <- data.frame(iv, rv, f)

mm <- makeModelMatrixFromDataFrame(df)

## create test and train matrices with disjoint factor levels
train.df <- df[1:8,]
test.df <- df[9:10,]
train.mm <- makeModelMatrixFromDataFrame(train.df)
test.mm <- makeModelMatrixFromDataFrame(test.df, attr(train.mm, "drop"))
```

pdbart

*Partial Dependence Plots for BART***Description**

Run `bart` at test observations constructed so that a plot can be created displaying the effect of a single variable (`pdbart`) or pair of variables (`pd2bart`). Note that if y is a binary with $P(Y = 1|x) = F(f(x))$, F the standard normal cdf, then the plots are all on the f scale.

Usage

```
pdbart(x.train, y.train,
       xind = seq_len(ncol(x.train)),
       levs = NULL, levquants = c(0.05, seq(0.1, 0.9, 0.1), 0.95),
       pl = TRUE, plquants = c(0.05, 0.95),
       ...)

## S3 method for class 'pdbart'
plot(x,
     xind = seq_len(length(x$fd)),
     plquants = c(0.05, 0.95), cols = c('black', 'blue'),
     ...)

pd2bart(x.train, y.train,
        xind = c(1, 2),
        levs = NULL, levquants = c(0.05, seq(0.1, 0.9, 0.1), 0.95),
        pl = TRUE, plquants = c(0.05, 0.95),
        ...)

## S3 method for class 'pd2bart'
plot(x,
     plquants = c(0.05, 0.95), contour.color = 'white',
     justmedian = TRUE,
     ...)
```

Arguments

<code>x.train</code>	Explanatory variables for training (in sample) data. Must be a matrix of numeric type with rows corresponding to observations and columns to variables. Categorical variables/factors need to be converted to dummies, with a full set of columns present if there are more than two levels.
<code>y.train</code>	Dependent variable for training (in sample) data. Must be a numeric vector with length equal to the number of rows in <code>x.train</code> .
<code>xind</code>	Integer vector indicating which variables are to be plotted. In <code>pdbart</code> , corresponds to the variables (columns of <code>x.train</code>) for which a plot is to be constructed. In <code>plot.pdbart</code> , corresponds to the indices in list returned by <code>pdbart</code>

	for which plot is to be constructed. In <code>pd2bart</code> , the indices of a pair of variables (columns of <code>x.train</code>) to plot.
<code>levs</code>	Gives the values of a variable at which the plot is to be constructed. Must be a list, where the i th component gives the values for the i th variable. In <code>pdbart</code> , it should have same length as <code>xind</code> . In <code>pd2bart</code> , it should have length 2. See also argument <code>levquants</code> .
<code>levquants</code>	If <code>levs</code> in <code>NULL</code> , the values of each variable used in the plot is set to the quantiles (in <code>x.train</code>) indicated by <code>levquants</code> . Must be a vector of numeric type.
<code>pl</code>	For <code>pdbart</code> and <code>pd2bart</code> , if <code>TRUE</code> , plot is subsequently made (by calling <code>plot.*</code>).
<code>plquants</code>	In the plots, beliefs about $f(x)$ are indicated by plotting the posterior median and a lower and upper quantile. <code>plquants</code> is a double vector of length two giving the lower and upper quantiles.
<code>...</code>	Additional arguments. In <code>pdbart</code> and <code>pd2bart</code> , arguments are passed on to <code>bart</code> . In <code>plot.pdbart</code> , they are passed on to <code>plot</code> . In <code>plot.pd2bart</code> , they are passed on to <code>image</code> .
<code>x</code>	For <code>plot.*</code> , object returned from <code>pdbart</code> or <code>pd2bart</code> .
<code>cols</code>	Vector of two colors. The first color is for the median of f , while the second color is for the upper and lower quantiles.
<code>contour.color</code>	Color for contours plotted on top of the image.
<code>justmedian</code>	A logical where if <code>TRUE</code> just one plot is created for the median of $f(x)$ draws. If <code>FALSE</code> , three plots are created one for the median and two additional ones for the lower and upper quantiles. In this case, <code>mfrow</code> is set to <code>c(1, 3)</code> .

Details

We divide the predictor vector x into a subgroup of interest, x_s and the complement $x_c = x \setminus x_s$. A prediction $f(x)$ can then be written as $f(x_s, x_c)$. To estimate the effect of x_s on the prediction, Friedman suggests the partial dependence function

$$f_s(x_s) = \frac{1}{n} \sum_{i=1}^n f(x_s, x_{ic})$$

where x_{ic} is the i th observation of x_c in the data. Note that (x_s, x_{ic}) will generally not be one of the observed data points. Using BART it is straightforward to then estimate and even obtain uncertainty bounds for $f_s(x_s)$. A draw of $f_s^*(x_s)$ from the induced BART posterior on $f_s(x_s)$ is obtained by simply computing $f_s^*(x_s)$ as a byproduct of each MCMC draw f^* . The median (or average) of these MCMC draws $f_s^*(x_s)$ then yields an estimate of $f_s(x_s)$, and lower and upper quantiles can be used to obtain intervals for $f_s(x_s)$.

In `pdbart` x_s consists of a single variable in x and in `pd2bart` it is a pair of variables.

This is a computationally intensive procedure. For example, in `pdbart`, to compute the partial dependence plot for 5 x_s values, we need to compute $f(x_s, x_c)$ for all possible (x_s, x_{ic}) and there would be $5n$ of these where n is the sample size. All of that computation would be done for each kept BART draw. For this reason running BART with `keepevery` larger than 1 (eg. 10) makes the procedure much faster.

Value

The plot methods produce the plots and don't return anything.

pdbart and pd2bart return lists with components given below. The list returned by pdbart is assigned class pdbart and the list returned by pd2bart is assigned class pd2bart.

fd	A matrix whose (i, j) value is the i th draw of $f_s(x_s)$ for the j th value of x_s . "fd" is for "function draws". For pdbart fd is actually a list whose k th component is the matrix described above corresponding to the k th variable chosen by argument xind. The number of columns in each matrix will equal the number of values given in the corresponding component of argument levs (or number of values in levquants). For pd2bart, fd is a single matrix. The columns correspond to all possible pairs of values for the pair of variables indicated by xind. That is, all possible (x_i, x_j) where x_i is a value in the levs component corresponding to the first x and x_j is a value in the levs components corresponding to the second one. The first x changes first.
levs	The list of levels used, each component corresponding to a variable. If argument levs was supplied it is unchanged. Otherwise, the levels in levs are as constructed using argument levquants.
xlbs	A vector of character strings which are the plotting labels used for the variables.

The remaining components returned in the list are the same as in the value of `bart`. They are simply passed on from the BART run used to create the partial dependence plot. The function `plot.bart` can be applied to the object returned by pdbart or pd2bart to examine the BART run.

Author(s)

Hugh Chipman: <hugh.chipman@acadiiau.ca>
Robert McCulloch: <robert.mcculloch@chicagogsb.edu>

References

Chipman, H., George, E., and McCulloch, R. (2006) BART: Bayesian Additive Regression Trees.
Chipman, H., George, E., and McCulloch R. (2006) Bayesian Ensemble Learning.
both of the above at: <http://www.rob-mcculloch.org/>
Friedman, J.H. (2001) Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, **29**, 1189–1232.

Examples

```
## Not run:
## simulate data
f <- function(x) { return(0.5 * x[,1] + 2 * x[,2] * x[,3]) }

sigma <- 0.2
n     <- 100
```

```

set.seed(27)
x <- matrix(2 * runif(n * 3) - 1, ncol = 3);
colnames(x) <- c('rob', 'hugh', 'ed')

Ey <- f(x)
y <- rnorm(n, Ey, sigma)

## first two plot regions are for pdbart, third for pd2bart
par(mfrow = c(1, 3))

## pdbart: one dimensional partial dependence plot
set.seed(99)
pdb1 <-
  pdbart(x, y, xind = c(1, 2),
         levs = list(seq(-1, 1, 0.2), seq(-1, 1, 0.2)),
         pl = FALSE, keepevery = 10, ntree = 100)
plot(pdb1, ylim = c(-0.6,.6))

## pd2bart: two dimensional partial dependence plot
set.seed(99)
pdb2 <-
  pd2bart(x, y, xind = c(2, 3),
         levquants = c(0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95),
         pl = FALSE, ntree = 100, keepevery = 10, verbose = FALSE)
plot(pdb2)

## compare BART fit to linear model and truth = Ey
lmFit <- lm(y ~., data.frame(x, y))
fitmat <- cbind(y, Ey, lmFit$fitted, pdb1$yhat.train.mean)
colnames(fitmat) <- c('y', 'Ey', 'lm', 'bart')
print(cor(fitmat))

## End(Not run)

```

rbart

Bayesian Additive Regression Trees with Random Effects

Description

Fits a varying intercept/random effect BART model.

Usage

```

rbart_vi(
  formula, data, test, subset, weight, offset, offset.test = offset,
  group.by, prior = cauchy,
  sigest = NA_real_, sigdf = 3.0, sigquant = 0.90,
  k = 2.0,
  power = 2.0, base = 0.95,
  n.trees = 75L,

```

```
n.samples = 1500L, n.burn = 1500L,
n.chains = 4L, n.threads = min(guessNumCores(), n.chains), combineChains = FALSE,
n.cuts = 100L, useQuantiles = FALSE,
n.thin = 5L, keepTrainingFits = TRUE,
printEvery = 100L, printCutoffs = 0L,
verbose = TRUE,
keepTrees = TRUE, keepCall = TRUE, ...)
```

Arguments

<code>group.by</code>	Grouping factor. Can be an integer vector/factor, or a reference to such in data.
<code>prior</code>	A function or symbolic reference to built-in priors. Determines the prior over the standard deviation of the random effects. Supplied functions take two arguments, <code>x</code> - the standard deviation, and <code>rel.scale</code> - the standard deviation of the response variable before random effects are fit. Built in priors are <code>cauchy</code> with a scale of 2.5 times the relative scale and <code>gamma</code> with a shape of 2.5 and scale of 2.5 times the relative scale.
<code>n.thin</code>	The number of tree jumps taken for every stored sample, but also the number of samples from the posterior of the standard deviation of the random effects before one is kept.
<code>formula, data, test, subset, weight, offset, offset.test, sigest, sigdf, sigquant, k, power, base,</code>	Same as in bart2 .

Details

Fits a BART model with additive random intercepts, one for each factor level of `group.by`. That is

- $y_i = b_{g[i]} + f(x_i) + \epsilon$,
- $b_j \sim N(0, \tau^2)$.

where i indices observations, $g[i]$ is the group index of observation i , $f(x)$ and ϵ come from a BART model, and b_j are the independent and identically distributed random intercepts.

Value

An object of class `rbart`. Contains all of the same elements of an object of class `bart`, as well as the elements

<code>ranef</code>	Samples from the posterior of the random effects. A array/matrix of posterior samples. The (i, j, k) value is the j th draw of the posterior of the random effect for group k (i.e. b_k^*) corresponding to chain i . When <code>nchain</code> is one or <code>combineChains</code> is TRUE, the result is a collapsed down to a matrix.
<code>ranef.mean</code>	Posterior mean of random effects, derived by taking mean across group index of samples.
<code>tau</code>	Matrix of posterior samples of tau, the standard deviation of the random effects. Dimensions are equal to the number of chains times the numbers of samples unless <code>nchain</code> is one or <code>combineChains</code> is TRUE.
<code>first.tau</code>	Burn-in draws of tau.

Author(s)

Vincent Dorie: <vdorie@gmail.com>

See Also

[bart](#), [dbarts](#)

Examples

```
f <- function(x) {
  10 * sin(pi * x[,1] * x[,2]) + 20 * (x[,3] - 0.5)^2 +
  10 * x[,4] + 5 * x[,5]
}

set.seed(99)
sigma <- 1.0
n <- 100

x <- matrix(runif(n * 10), n, 10)
Ey <- f(x)
y <- rnorm(n, Ey, sigma)

n.g <- 10
g <- sample(n.g, length(y), replace = TRUE)
sigma.b <- 1.5
b <- rnorm(n.g, 0, sigma.b)

y <- y + b[g]

df <- as.data.frame(x)
colnames(df) <- paste0("x_", seq_len(ncol(x)))
df$y <- y
df$g <- g

## low numbers to reduce run time
rbartFit <- rbart_vi(y ~ . - g, df, group.by = g,
  n.samples = 40L, n.burn = 10L, n.thin = 2L, n.chains = 1L,
  n.trees = 25L, n.threads = 1L)
```

xbart

Crossvalidation For Bayesian Additive Regression Trees

Description

Fits the BART model against varying k , power, base, and n_{tree} parameters using K -fold or repeated random subsampling crossvalidation, sharing burn-in between parameter settings. Results are given an array of evaluations of a loss functions on the held-out sets.

Usage

```
xbart(formula, data, subset, weights, offset, verbose = FALSE, n.samples = 200L,
      method = c("k-fold", "random subsample"), n.test = c(5, 0.2),
      n.reps = 40L, n.burn = c(200L, 150L, 50L),
      loss = c("rmse", "mcr"), n.threads = guessNumCores(), n.trees = 75L,
      k = 2, power = 2, base = 0.95, drop = TRUE,
      resid.prior = chisq, control = dbartsControl(), sigma = NA_real_)
```

Arguments

formula	An object of class <code>formula</code> following an analogous model description syntax as <code>lm</code> . For backwards compatibility, can also be the <code>bart</code> matrix <code>x.train</code> . See <code>dbarts</code> .
data	An optional data frame, list, or environment containing predictors to be used with the model. For backwards compatibility, can also be the <code>bart</code> vector <code>y.train</code> .
subset	An optional vector specifying a subset of observations to be used in the fitting process.
weights	An optional vector of weights to be used in the fitting process. When present, BART fits a model with observations $y \mid x \sim N(f(x), \sigma^2/w)$, where $f(x)$ is the unknown function.
offset	An optional vector specifying an offset from 0 for the relationship between the underlying function, $f(x)$, and the response y . Only is useful for binary responses, in which case the model fit is to assume $P(Y = 1 \mid X = x) = \Phi(f(x) + \text{offset})$, where Φ is the standard normal cumulative distribution function.
verbose	A logical determining if additional output is printed to the console.
n.samples	A positive integer, setting the number of posterior samples drawn for each fit of training data and used by the loss function.
method	Character string, either "k-fold" or "random subsample".
n.test	For each fit, the test sample size or proportion. For method "k-fold", is expected to be the number of folds, and in $[2, n]$. For method "random subsample", can be a real number in $(0, 1)$ or a positive integer in $(1, n)$. When a given as proportion, the number of test observations used is the proportion times the sample size rounded to the nearest integer.
n.reps	A positive integer setting the number of cross validation steps that will be taken. For "k-fold", each replication corresponds to fitting each of the K folds in turn, while for "random subsample" a replication is a single fit.
n.burn	Between one and three positive integers, specifying the 1) initial burn-in, 2) burn-in when moving from one parameter setting to another, and 3) the burn-in between each random subsample replication. The third parameter is also the burn in when moving between folds in "k-fold" crossvalidation.
loss	Either a one of the present loss functions as character-strings (mcr - missclassification rate for binary responses, rmse - root-mean-squared-error for continuous response), a function, or a function- evaluation environment list-pair. Functions

should have prototypes of the form `function(y.test, y.test.hat)`, where `y.test` is the held out test subsample and `y.test.hat` is a matrix of dimension `length(y.test) * n.samples`. See examples.

<code>n.threads</code>	Across different sets of parameters ($k \times \text{power} \times \text{base} \times \text{n.trees}$) and <code>n.reps</code> , results are independent. For <code>n.threads > 1</code> , evaluations of the above are divided into approximately equal size evaluations chunks and executed in parallel. The default uses <code>link{guessNumCores}</code> , which should work across the most common operating system/hardware pairs. A value of <code>NA</code> is interpreted as 1.
<code>n.trees</code>	A vector of positive integers setting the BART hyperparameter for the number of trees in the sum-of-trees formulation. See bart .
<code>k</code>	A vector of positive real numbers, setting the BART hyperparameter for the node-mean prior standard deviation.
<code>power</code>	A vector of real numbers greater than one, setting the BART hyperparameter for the tree prior's growth probability, given by $\text{base}/(1 + \text{depth})^{\text{power}}$.
<code>base</code>	A vector of real numbers in $(0, 1)$, setting the BART hyperparameter for the tree prior's growth probability.
<code>drop</code>	Logical, determining if dimensions with a single value are dropped from the result.
<code>resid.prior</code>	An expression of the form <code>chisq</code> or <code>chisq(df, quant)</code> that sets the prior used on the residual/error variance.
<code>control</code>	An object inheriting from <code>dbartsControl</code> , created by the dbartsControl function.
<code>sigma</code>	A positive numeric estimate of the residual standard deviation. If <code>NA</code> , a linear model is used with all of the predictors to obtain one.

Details

Crossvalidates `n.reps` replications against the crossproduct of given hyperparameter vectors `n.trees` \times `k` \times `power` \times `base`. For each fit, either one fold is withheld as test data and `n.test - 1` folds are used as training data or `n * n.test` observations are withheld as test data and `n * (1 - n.test)` used as training. A replication corresponds to fitting all K folds in "k-fold" crossvalidation or a single fit with "random subsample". The training data is used to fit a model and make predictions on the test data which are used together with the test data itself to evaluate the loss function.

loss functions are either the default of missclassification rates for binary outcomes and mean-squared error for continuous outcomes, or a function with arguments `y.test` and `y.test.hat`. `y.test.hat` is of dimensions equal to `length(y.test) * n.samples`. A third option is to pass a list of `list(function, evaluationEnvironment)`, so as to provide default bindings.

Value

An array of dimensions `n.reps` \times `length(n.trees)` \times `length(k)` \times `length(power)` \times `length(base)`. If `drop` is `TRUE`, dimensions of length 1 are omitted. If all hyperparameters are of length 1, then the result will be a vector of length `n.reps`. When the result is an array, the `dimnames` of the result shall be set to the corresponding hyperparameters.

For method "k-fold", each element is an average across the K fits. For "random subsample", each element represents a single fit.

Author(s)

Vincent Dorie: <vdorie@gmail.com>

See Also

[bart](#), [dbarts](#)

Examples

```
f <- function(x) {
  10 * sin(pi * x[,1] * x[,2]) + 20 * (x[,3] - 0.5)^2 +
  10 * x[,4] + 5 * x[,5]
}

set.seed(99)
sigma <- 1.0
n <- 100

x <- matrix(runif(n * 10), n, 10)
Ey <- f(x)
y <- rnorm(n, Ey, sigma)

mad <- function(y.train, y.train.hat)
  mean(abs(y.train - apply(y.train.hat, 1L, mean)))

## low iteration numbers to to run quickly
xval <- xbart(x, y, n.samples = 15L, n.reps = 4L, n.burn = c(10L, 3L, 1L),
  n.trees = c(5L, 7L),
  k = c(1, 2, 4),
  power = c(1.5, 2),
  base = c(0.75, 0.8, 0.95), n.threads = 1L,
  loss = mad)
```

Index

- *Topic **crossvalidation**
 - xbart, 20
- *Topic **dplot**
 - pdbart, 15
- *Topic **factor**
 - makeModelMatrixFromDataFrame, 13
- *Topic **nonlinear**
 - bart, 2
 - pdbart, 15
- *Topic **nonparametric**
 - bart, 2
 - pdbart, 15
 - rbart, 18
 - xbart, 20
- *Topic **parallel**
 - guessNumCores, 13
- *Topic **randomeffects**
 - rbart, 18
- *Topic **regression**
 - bart, 2
 - pdbart, 15
 - rbart, 18
 - xbart, 20
- *Topic **tree**
 - bart, 2
 - pdbart, 15
 - rbart, 18
 - xbart, 20

- bart, 2, 7, 10, 13, 15–17, 19–23
- bart2, 19
- bart2 (bart), 2

- control, 12

- data.frame, 10
- dbarts, 7, 8–10, 20, 21, 23
- dbartsControl, 8, 8, 12, 22
- dbartsData, 10, 12
- dbartsSampler, 8

- dbartsSampler (dbartsSampler-class), 11
- dbartsSampler-class, 11

- formula, 7, 10, 21

- guessNumCores, 13

- image, 16

- lm, 7, 21
- loading, 10

- makeind (makeModelMatrixFromDataFrame), 13
- makeModelMatrixFromDataFrame, 13
- mfrow, 16

- pd2bart (pdbart), 15
- pdbart, 6, 15
- plot, 12, 16
- plot.bart, 17
- plot.bart (bart), 2
- plot.pd2bart (pdbart), 15
- plot.pdbart (pdbart), 15
- predict.bart (bart), 2

- rbart, 18
- rbart_vi (rbart), 18
- rbind, 14
- ReferenceClasses, 8
- run, 9

- sampler, 10
- saving, 10
- set.seed, 10

- xbart, 20