Package ‘dbarts’

December 3, 2024

Version 0.9-30

Date 2024-12-03

Title Discrete Bayesian Additive Regression Trees Sampler

Depends R (>= 3.1-0)

Imports stats, methods, graphics, parallel

Suggests testthat (>= 0.9-0), knitr, rmarkdown

VignetteBuilder knitr, rmarkdown

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/Metropolis-Hastings 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] (<https://orcid.org/0000-0002-9576-3064>),
Hugh Chipman [aut],
Robert McCulloch [aut],
Armon Dadgar [ctb] (adaptive radix tree),
R Core Team [ctb] (basis of RNG),
Guido U Draheim [ctb] (ax_check_compile_flag.m4),
Maarten Bosmans [ctb] (ax_check_compile_flag.m4),
Christophe Tournayre [ctb] (ax_compiler_ext.m4, ax_ext.m4),
Michael Petch [ctb] (ax_compiler_ext.m4, ax_ext.m4,
ax_gcc_x86_avx_xgetbv.m4, ax_gcc_x86_cpuid.m4),
Rafael de Lucena Valle [ctb] (ax_compiler_ext.m4, ax_ext.m4),
Steven G. Johnson [ctb] (ax_compiler_vendor.m4, ax_gcc_x86_cpuid.m4,
ax_pthread.m4, <https://orcid.org/0000-0001-7327-4967>),
Matteo Frigo [ctb] (ax_compiler_vendor.m4, ax_gcc_x86_cpuid.m4),
John Zaitseff [ctb] (ax_compiler_vendor.m4),

Todd Veldhuizen [ctb] (ax_cxx_namespace_std.m4),
Luc Maisonobe [ctb] (ax_cxx_namespace_std.m4),
Scott Pakin [ctb] (ax_func_posix_memalign.m4,
<https://orcid.org/0000-0002-5220-1985>),
Daniel Richard G. [ctb] (ax_pthread.m4)

Maintainer Vincent Dorie <vdorie@gmail.com>

Repository CRAN

Date/Publication 2024-12-03 15:50:02 UTC

Contents

`bart` *Bayesian Additive Regression Trees*

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, $P(Y = 1 \mid x) = \Phi(f(x))$, 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, splitprobs = 1 / numvars,
    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, sampleronly = FALSE,
    seed = NA_integer_,
    proposalprobs = NULL,
    keepsampler = keeptrees)

bart2(
    formula, data, test, subset, weights, offset, offset.test = offset,
    sigest = NA_real_, sigdf = 3.0, sigquant = 0.90,
    k = NULL,
    power = 2.0, base = 0.95, split.probs = 1 / num.vars,
    n.trees = 75L,
    n.samples = 500L, n.burn = 500L,
    n.chains = 4L, n.threads = min(dbarts::guessNumCores(), n.chains),
    combineChains = FALSE,
    n.cuts = 100L, useQuantiles = FALSE,
    n.thin = 1L, keepTrainingFits = TRUE,
    printEvery = 100L, printCutoffs = 0L,
    verbose = TRUE, keepTrees = FALSE,
    keepCall = TRUE, samplerOnly = FALSE,
    seed = NA_integer_,
    proposal.probs = NULL,
    keepSampler = keepTrees,
    ...)

## S3 method for class 'bart'
plot(
    x,
    plquants = c(0.05, 0.95), cols = c('blue', 'black'),
    ...)

## S3 method for class 'bart'
predict(
    object, newdata, offset, weights,
    type = c("ev", "ppd", "bart"),
    combineChains = TRUE, ...)

extract(object, ...)
## S3 method for class 'bart'
extract(
    object,
    type = c("ev", "ppd", "bart", "trees"),
    sample = c("train", "test"),
    combineChains = TRUE, ...)

## S3 method for class 'bart'
```

```
fitted(
    object,
    type = c("ev", "ppd", "bart"),
    sample = c("train", "test"),
    ...)

## S3 method for class 'bart'
residuals(object, ...)
```

Arguments

`x.train` 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.

`y.train` Dependent variable for training (in sample) data. If `y.train` is numeric a continous response model is fit (normal errors). If `y.train` is a binary factor or has only values 0 and 1, then a binary response model with a probit link is fit.

`x.test` Explanatory variables for test (out of sample) data. Should have same column structure as `x.train`. `bart` will generate draws of $f(x)$ for each x which is a row of `x.test`.

`sigest` 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 `sigquant` for more information. Not applicable when y is binary.

`sigdf` Degrees of freedom for error variance prior. Not applicable when y is binary.

`sigquant` The quantile of the error variance prior that the rough estimate (`sigest`) is placed at. The closer the quantile is to 1, the more aggresive 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.

`k` For numeric y, `k` is the number of prior standard deviations $E(Y|x) = f(x)$ is away from ± 0.5. The response (`y.train`) 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. The value can be either a fixed number, or the a *hyperprior* of the form `chi(degreesOfFreedom = 1.25, scale = Inf)`. For `bart2`, the default of `NULL` uses the value 2 for continuous reponses and a `chi` hyperprior for binary ones. The default `chi` hyperprior is improper, and slightly penalizes small values of `k`.

`power` Power parameter for tree prior.

`base` Base parameter for tree prior.

`splitprobs, split.probs`
Prior and transition probabilities of variables used to generate splits. Can be missing/empty/`NULL` for equiprobability, a numeric vector of length equal to the number variables, or a named numeric vector with only a subset of the variables specified and a `.default` named value. Values given for factor variables are replicated for each resulting column in the generated model matrix.

numvars and num.vars symbols will be rebound before execution to the number of columns in the model matrix.

binaryOffset
: Used for binary y. When present, the model is $P(Y = 1 \mid x) = \Phi(f(x) + \text{binaryOffset})$, allowing fits with probabilities shrunk towards values other than 0.5.

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.

ntree, n.trees
: The number of trees in the sum-of-trees formulation.

ndpost, n.samples
: The number of posterior draws after burn in, ndpost / keepevery will actually be returned.

nskip, n.burn
: Number of MCMC iterations to be treated as burn in.

printevery, printEvery
: As the MCMC runs, a message is printed every printevery draws.

keepevery, n.thin
: Every keepevery draw is kept to be returned to the user. Useful for “thinning” samples.

keeptrainfits, keepTrainingFits
: If TRUE the draws of $f(x)$ for x corresponding to the rows of x.train are returned.

usequants, useQuantiles
: When TRUE, determine tree decision rules using estimated quantiles derived from the x.train variables. When FALSE, splits are determined using values equally spaced across the range of a variable. See details for more information.

numcut, n.cuts
: The maximum number of possible values used in decision rules (see usequants, details). If a single number, it is recycled for all variables; otherwise must be a vector of length equal to ncol(x.train). Fewer rules may be used if a covariate lacks enough unique values.

printcutoffs, printCutoffs
: 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.

verbose
: Logical; if FALSE supress printing.

nchain, n.chains
: Integer specifying how many independent tree sets and fits should be calculated.

nthread, n.threads
: 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.

combinechains, combineChains
: Logical; if TRUE, samples will be returned in arrays of dimensions equal to nchain $\times$ ndpost $\times$ number of observations.

keeptrees, keepTrees
: Logical; must be TRUE in order to use predict with the result of a bart fit. Note that for models with a large number of observations or a large number of trees, keeping the trees can be very memory intensive.

keepcall, keepCall
: Logical; if FALSE, returned object will have call set to call("NULL"), otherwise the call used to instantiate BART.

seed
: Optional integer specifying the desired pRNG seed. It should not be needed when running single-threaded - set.seed will suffice, and can be used to obtain reproducible results when multi-threaded. See Reproducibility section below.

proposalprobs, proposal.probs
: Named numeric vector or NULL, optionally specifying the proposal rules and their probabilities. Elements should be "birth_death", "change", and "swap" to control tree change proposals, and "birth" to give the relative frequency of birth/death in the "birth_death" step. Defaults are 0.5, 0.1, 0.4, and 0.5 respectively.

keepsampler, keepSampler
: Logical that can be used to save the underlying dbartsSampler-class object even if keepTrees is false.

formula
: The same as x.train, the name reflecting that a formula object can be used instead.

data
: The same as y.train, the name reflecting that a data frame can be specified when a formula is given instead.

test
: The same as x.train. Can be missing.

subset
: A vector of logicals or indicies used to subset of the data. Can be missing.

offset
: The same as binaryOffset. Can be missing.

offset.test
: A vector of offsets to be used with test data, in case it is different than the training offset. If offest is missing, defaults to NULL.

object
: An object of class bart, returned from either the function bart or bart2.

newdata
: Test data for prediction. Obeys all the same rules as x.train but cannot be missing.

sampleronly, samplerOnly
: Builds the sampler from its arguments and returns it without running it. Useful to use the bart2 interface in more complicated models.

x
: Object of class bart, returned by function bart, which contains the information to be plotted.

plquants
: In the plots, beliefs about $f(x)$ are indicated by plotting the posterior median and a lower and upper quantile. plquants 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.

type
: The quantity to be returned by generic functions. Options are "ev" - samples from the posterior of the individual level expected value, "bart" - the sum of trees component; same as "ev" for linear models but on the probit scale for

binary ones, "ppd" - samples from the posterior predictive distribution, and "trees" - a data frame with tree information for when model was fit with keepTrees equal to TRUE. To synergize with predict.glm, "response" can be used as a synonym for "ev" and "link" can be used as a synonym for "bart". For information on extracting trees, see the subsection below.

sample Either "train" or "test".

... Additional arguments passed on to plot, dbartsControl, or extract when type is "trees". Not used in predict.

Details

BART is an Bayesian MCMC method. At each MCMC interation, we produce a draw from the joint posterior $(f, \sigma) \mid (x, y)$ 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.

End-node prior parameter k: The amount of shrinkage of the node parameters is controlled by k. k can be given as either a fixed, positive number, or as any value that can be used to build a supported hyperprior. At present, only $\chi_\nu s$ priors are supported, where ν is a degrees of freedom and s is a scale. Both values must be positive, however the scale can be infinite which yields an improper prior, which is interpretted as just the polynomial part of the distribution. If nu is 1 and s is ∞, the prior is "flat".

For BART on binary outcomes, the degree of overfitting can be highly sensitive to k so it is encouraged to consider a number of values. The default hyperprior for binary BART, chi(1.25, Inf), has been shown to work well in a large number of datasets, however crossvalidation may be helpful. Running for a short time with a flat prior may be helpful to see the range of values of k that are consistent with the data.

Generics: bart and rbart_vi support fitted to return the posterior mean of a predicted quantity, as well as predict to return a set of posterior samples for a different sample. In addition, the extract generic can be used to obtain the posterior samples for the training data or test data supplied during the initial fit.

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 and is off by default.

All generics return values on the scale of expected value of the response by default. This means that `predict`, `extract`, and `fitted` for binary outcomes return probabilities unless specifically the sum-of-trees component is requested (`type = "bart"`). This is in contrast to `yhat.train`/`yhat.test` that are returned with the fitted model.

Saving: `save`ing and `load`ing fitted BART objects for use with `predict` requires that R's serialization mechanism be able to access the underlying trees, in addition to being fit with `keeptrees`/`keepTrees` as `TRUE`. For memory purposes, the trees are not stored as R objects unless specifically requested. To do this, one must "touch" the sampler's state object before saving, e.g. for a fitted object `bartFit`, execute `invisible(bartFit$fit$state)`.

Reproducibility: Behavior differs when running multi- and single-threaded, as the pseudo random number generators (pRNG) used by R are not thread safe. When single-threaded, R's built-in generator is used; if set at the start, the global `.Random.seed` will be used and its value updated as samples are drawn. When multi-threaded, the default behavior is to draw new random seeds for each thread using the clock and use thread-specific pRNGs.

This behavior can be modified by setting `seed`, or by using `...` to pass arguments to `dbartsControl`. For the single-threaded case, a new pRNG is built using that seed that is separate from R's native generator. As such, the global state will not be modified by subsequent calls to the generator. For multi-threaded, the seeds for threads are drawn sequentially using the supplied seed, and will again be separate from R's native generator.

Consequently, the `seed` argument is not needed when running single-threaded - `set.seed` will suffice. However, when multi-threaded the `seed` argument can be used to obtain reproducible results.

Extracting Trees: When a model is fit with `keeptrees` (`bart`) or `keepTrees` (`bart2`) equal to `TRUE`, the generic `extract` can be used to retrieve a data frame containing the tree fit information. In this case, `extract` will accept the additional, optional arguments: `chainNums`, `sampleNums`, and `treeNums`. Each should be an integer vector detailing the desired trees to be returned.

The result of `extract` will be a data frame with columns:

- `sample`, `chain`, `tree` - index variables
- `n` - number of observations in node
- `var` - either the index of the variable used for splitting or -1 if the node is a leaf
- `value` - either the value such that observations less than or equal to it are sent down the left path of the tree or the predicted value for a leaf node

The order of nodes in the result corresponds to a depth-first traversal, going down the left-side first. The names of variables used in splitting can be recovered by examining the column names of the `fit$data@x` element of a fitted `bart` or `bart2` model. See the package vignette "Working with dbarts Saved Trees".

Value

`bart` and `bart2` return lists assigned the class `bart`. For applicable quantities, `ndpost / keepevery` samples are returned. In the numeric y case, the list has components:

`yhat.train` A array/matrix of posterior samples. The (i, j, k) value is the jth draw of the posterior of f evaluated at the kth row of `x.train` (i.e. $f^*(x_k)$) corresponding to chain i. When `nchain` is one or `combinechains` is `TRUE`, the result is a collapsed down to a matrix.

yhat.test	Same as yhat.train but now the xs are the rows of the test data.
yhat.train.mean	Vector of means of yhat.train across columns and chains, with length equal to the number of training observations.
yhat.test.mean	Vector of means of yhat.test across columns and chains.
sigma	Matrix of posterior samples of sigma, the residual/error standard deviation. Dimensions are equal to the number of chains times the numbers of samples unless nchain is one or combinechains is TRUE.
first.sigma	Burn-in draws of sigma.
varcount	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).
sigest	The rough error standard deviation (σ) used in the prior.
y	The input dependent vector of values for the dependent variable. This is used in plot.bart.
fit	Optional sampler object which stores the values of the tree splits. Required for using predict and only stored if keeptrees or keepsampler is TRUE.
n.chains	Information that can be lost if combinechains is TRUE is tracked here.
k	Optional matrix of posterior samples of k. Only present when k is modeled, i.e. there is a hyperprior.
first.k	Burn-in draws of k, if modeled.

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 $f(x) + \mathrm{binaryOffset}$. 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 a good 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: https://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 Regreesion 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,
```

```
    proposal.probs = c(
        birth_death = 0.5, swap = 0.1, change = 0.4, birth = 0.5),
    control = dbarts::dbartsControl(), sigma = NA_real_)
```

Arguments

`formula`	An object of class `formula` following an analogous model description syntax as `lm`. For backwards compatibility, can also be the `bart` matrix `x.train`.
`data`	An optional data frame, list, or environment containing predictors to be used with the model. For backwards compatibility, can also be the `bart` vector `y.train`.
`test`	An optional matrix or data frame with the same number of predictors as `data`, or `formula` 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 underyling function, $f(x)$, and the response y. Only is useful for binary responses, in which case the model fit is to assume $P(Y = 1 \mid X = x) = \Phi(f(x) + \mathrm{offset})$, where Φ is the standard normal cumulative distribution function.
`offset.test`	The equivalent of `offset` for test observations. Will attempt to use `offset` 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 overriden at run-time. See `dbartsControl`.
`tree.prior`	An expression of the form `cgm` or `cgm(power, base, split.probs)` setting the tree prior used in fitting.
`node.prior`	An expression of the form `normal` or `normal(k)` that sets the prior used on the averages within nodes.
`resid.prior`	An expression of the form `chisq` or `chisq(df, quant)` that sets the prior used on the residual/error variance.
`proposal.probs`	Named numeric vector or `NULL`, optionally specifying the proposal rules and their probabilities. Elements should be `"birth_death"`, `"change"`, and `"swap"` to control tree change proposals, and `"birth"` to give the relative frequency of birth/death in the `"birth_death"` step.
`control`	An object inheriting from `dbartsControl`, 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 = dbarts::guessNumCores(), n.thin = 1L, printEvery = 100L,
    printCutoffs = 0L,
    rngKind = "default", rngNormalKind = "default", rngSeed = NA_integer_,
    updateState = TRUE)
```

Arguments

- `verbose` Logical controlling sampler output to console.
- `keepTrainingFits` 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.
- `useQuantiles` 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.
- `keepTrees` 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 n.trees. When keepTrees is TRUE, a set of n.trees * n.samples 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.
- `n.samples` A non-negative integer giving the default number of samples to return each time the sampler is run. Generally specified by dbarts instead, and can be overridden on a per-use basis whenever the sampler is run.

n.cuts	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.
n.burn	A non-negative integer determining how many samples, if any, are thrown away at the beginning of a run of the sampler.
n.trees	A positive integer giving the number of trees used in the sum-of-trees formulation.
n.chains	A positive integer detailing the number of independent chains for the sampler to use.
n.threads	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.
n.thin	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. n.samples are returned regardless of the value of n.thin.
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.
rngSeed	Random number generator seed, as used in set.seed. If the sampler is running single-threaded or has one chain, the behavior will be as any other sequential algorithm. If the sampler is multithreaded, the seed will be used to create an additional pRNG object, which in turn will be used sequentially seed the thread-specific pRNGs. If equal to NA, the clock will be used to seed pRNGs when applicable.
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'
sampleNodeParametersFromPrior(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, updateScale = FALSE, updateState = NA)
## S4 method for signature 'dbartsSampler'
setSigma(sigma, 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 = c(
        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 of a distinct 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.
updateScale	Logical indicating whether BART's internal scale should update with the new offset. Should only be `TRUE` during burn-in.
offset.test	A numeric vector of length equal to that of the test matrix, or `NULL`. Can be missing for `setTestPredictors`.
sigma	Numeric vector of residual standard deviations, one for each chain.
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 substituted.
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 named numeric vector containing the 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.

Saving: `save`-ing and `load`ing a `dbarts` sampler for future use requires that R's serialization mechanism be able to access the state of the sampler which, for memory purposes, is only made available to R on request. To do this, one must "touch" the sampler's state object before saving, e.g. for the object `sampler`, execute `invisible(sampler$state)`. This is in addition to guaranteeing that the `state` object is not `NULL`, which can be done by setting the sampler's control to an object with `updateState` as `TRUE` or passing `TRUE` as the `updateState` argument to any of the sampler's applicable methods.

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. This function has two use cases. The first is when `keepTrees` of `dbartsControl` is `TRUE`, in which case the sampler should be run to completion and the function can be used to interrogate the existing fit.

When `keepTrees` is `FALSE`, the function can be used to obtain the likelihood as part of a proposed new set of covariates in a Metropolis-Hastings step in a full-Bayes sampler. This would typically be followed by a call to `setPredictor` if the step is accepted.

`guessNumCores` *Guess Number of Cores*

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` *Make Model Matrix from Data Frame*

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)
makeTestModelMatrix(data, newdata)
```

Arguments

`x`	Data frame of explanatory variables.
`drop`	Logical or list controling 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: • 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.
`data`	An existing `dbartsData` object.
`newdata`	Test data frame.

Details

Character vectors are included as factors. If you have numeric data coded as characters, convert it using `as.numeric` first.

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 training data and use it when creating a matrix from the test data, as in the example given below.

Use of these functions is not required when using `bart`, `bart2`, or `dbartsSampler`; they exist to allow the user finer control and to assist with writing packages that separate the creation of training from test data.

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 = NULL,
    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 = NULL,
    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

- `x.train` — Explanatory variables for training (in sample) data. Can be any valid input to `bart`, such as a matrix or a formula. Also accepted are fitted `bart` models or `dbartsSampler` with `keepTrees` equal to `TRUE`.
- `y.train` — Dependent variable for training (in sample) data. Can be a numeric vector or, when passing `x.train` as a formula, a `data.frame` or other object used to find variables. Not required if `x.train` is a fitted model or sampler.
- `xind` — Integer, character vector, or the right-hand side of a formula indicating which variables are to be plotted. In `pdbart`, corresponds to the variables (columns of `x.train`) for which a plot is to be constructed. In `plot.pdbart`, corresponds to the indices in list returned by `pdbart` for which plot is to be constructed. In `pd2bart`, the indicies of a pair of variables (columns of `x.train`) to plot. If `NULL` a default of all columns is used for `pdbart` and the first two columns is used for `pd2bart`.
- `levs` — Gives the values of a variable at which the plot is to be constructed. Must be a list, where the ith component gives the values for the ith variable. In `pdbart`, it should have same length as `xind`. In `pd2bart`, it should have length 2. See also argument `levquants`.
- `levquants` — If `levs` in `NULL`, the values of each variable used in the plot is set to the quantiles (in `x.train`) indicated by levquants. Must be a vector of numeric type.
- `pl` — For `pdbart` and `pd2bart`, if `TRUE`, plot is subsequently made (by calling `plot.*`).
- `plquants` — In the plots, beliefs about $f(x)$ are indicated by plotting the posterior median and a lower and upper quantile. `plquants` is a double vector of length two giving the lower and upper quantiles.
- `...` — Additional arguments. In `pdbart` and `pd2bart`, arguments are passed on to `bart`. In `plot.pdbart`, they are passed on to `plot`. In `plot.pd2bart`, they are passed on to `image`.
- `x` — For `plot.*`, object returned from `pdbart` or `pd2bart`.
- `cols` — Vector of two colors. The first color is for the median of f, while the second color is for the upper and lower quantiles.
- `contour.color` — Color for contours plotted on top of the image.
- `justmedian` — A logical where if `TRUE` just one plot is created for the median of $f(x)$ draws. If `FALSE`, three plots are created one for the median and two additional ones for the lower and upper quantiles. In this case, `mfrow` is set to `c(1,3)`.

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 ith 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 ith draw of $f_s(x_s)$ for the jth value of x_s. "fd" is for "function draws".

For `pdbart` `fd` is actually a list whose kth component is the matrix described above corresponding to the kth 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@acadiau.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: https://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, 0.6))

## 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))

## example showing the use of a pre-fitted model
df <- data.frame(y, x)
set.seed(99)
bartFit <- bart(
    y ~ rob + hugh + ed, df,
    keepevery = 10, ntree = 100, keeptrees = TRUE)
pdb1 <- pdbart(bartFit, xind = rob + ed, pl = FALSE)
```

```
## 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, weights, offset, offset.test = offset,
    group.by, group.by.test, 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(dbarts::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,
    seed = NA_integer_,
    keepSampler = keepTrees,
    keepTestFits = TRUE,
    callback = NULL,
    ...)

## S3 method for class 'rbart'
plot(
    x, plquants = c(0.05, 0.95), cols = c('blue', 'black'), ...)

## S3 method for class 'rbart'
fitted(
    object,
    type = c("ev", "ppd", "bart", "ranef"),
    sample = c("train", "test"),
    ...)

## S3 method for class 'rbart'
extract(
    object,
```

```
    type = c("ev", "ppd", "bart", "ranef", "trees"),
    sample = c("train", "test"),
    combineChains = TRUE,
    ...)

## S3 method for class 'rbart'
predict(
    object, newdata, group.by, offset,
    type = c("ev", "ppd", "bart", "ranef"),
    combineChains = TRUE,
    ...)

## S3 method for class 'rbart'
residuals(object, ...)
```

Arguments

- `group.by` — Grouping factor. Can be an integer vector/factor, or a reference to such in `data`.
- `group.by.test` — Grouping factor for test data, of the same type as `group.by`. Can be missing.
- `prior` — 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, `x` - the standard deviation, and `rel.scale` - the standard deviation of the response variable before random effects are fit. Built in priors are `cauchy` with a scale of 2.5 times the relative scale and `gamma` with a shape of 2.5 and scale of 2.5 times the relative scale.
- `n.thin` — 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.
- `keepTestFits` — Logical where, if false, test fits are obtained while running but not returned. Useful with `callback`.
- `callback` — Optional function of `trainFits`, `testFits`, `ranef`, `sigma`, and `tau`. Called after every post-burn-in iteration and the results of which are collected and stored in the final object.
- `formula`, `data`, `test`, `subset`, `weights`, `offset`, `offset.test`, `sigest`, `sigdf`, `sigquant`, `k`, `power`, `base`, `n.trees`, `n.samples`, `n.burn`, `n.chains`, `n.threads`, `combineChains`, `n.cuts`, `useQuantiles`, `keepTrainingFits`, `printEvery`, `printCutoffs`, `verbose`, `keepTrees`, `keepCall`, `seed`, `keepSampler`, `...` — Same as in `bart2`.
- `object` — A fitted `rbart` model.
- `newdata` — Same as `test`, but named to match `predict` generic.
- `type` — One of `"ev"`, `"ppd"`, `"bart"`, `"ranef"`, or `"trees"` for the posterior of the expected value, posterior predictive distribution, non-parametric/BART component, random effect, or saved trees respectively. The expected value is the sum of the BART component and the random effects, while the posterior predictive distribution is a response sampled with that mean. To synergize with `predict.glm`, `"response"` can be used as a synonym for `"value"` and `"link"`

can be used as a synonym for "bart". For additional details on tree extraction, see the corresponding subsection in bart.

sample — One of "train" or "test", referring to the training or tests samples respectively.

x, plquants, cols — Same as in plot.bart.

Details

Fits a BART model with additive random intercepts, one for each factor level of group.by. For continuous responses:

- $y_i \sim N(f(x_i) + \alpha_{g[i]}, \sigma^2)$
- $\alpha_j \sim N(0, \tau^2)$.

For binary outcomes the response model is changed to $P(Y_i = 1) = \Phi(f(x_i) + \alpha_{g[i]})$. i indexes observations, $g[i]$ is the group index of observation i, $f(x)$ and σ_y come from a BART model, and α_j are the independent and identically distributed random intercepts. Draws from the posterior of tau are made using a slice sampler, with a width dynamically determined by assessing the curvature of the posterior distribution at its mode.

Out Of Sample Groups: Predicting random effects for groups not in the training sample is supported by sampling from their posterior predictive distribution, that is a draw is taken from $p(\alpha \mid y) = \int p(\alpha \mid \tau)p(\tau \mid y)d\alpha$. For out-of-sample groups in the test data, these random effect draws can be kept with the saved object. For those supplied to predict, they cannot and may change for subsequent calls.

Generics: See the generics section of bart.

Value

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

ranef — Samples from the posterior of the random effects. A array/matrix of posterior samples. The (k, l, j) value is the lth draw of the posterior of the random effect for group j (i.e. α_j^*) corresponding to chain k. When n.chains is one or combineChains is TRUE, the result is a collapsed down to a matrix.

ranef.mean — Posterior mean of random effects, derived by taking mean across group index of samples.

tau — 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 n.chains is one or combineChains is TRUE.

first.tau — Burn-in draws of tau.

callback — Optional results of callback function.

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 ntree parameters using K-fold or repeated random subsampling crossvalidation, sharing burn-in between parameter settings. Results are given an array of evalulations 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", "log", "mcr"), n.threads = dbarts::guessNumCores(), n.trees = 75L,
    k = NULL, power = 2, base = 0.95, drop = TRUE,
    resid.prior = chisq, control = dbarts::dbartsControl(), sigma = NA_real_,
    seed = NA_integer_)
```

Arguments

`formula`	An object of class `formula` following an analogous model description syntax as `lm`. For backwards compatibility, can also be the `bart` matrix `x.train`. See `dbarts`.
`data`	An optional data frame, list, or environment containing predictors to be used with the model. For backwards compatibility, can also be the `bart` vector `y.train`.
`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 underyling 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 pre-set loss functions as character-strings (`mcr` - missclassification rate for binary responses, `rmse` - root-mean-squared-error for continuous response), `log` - negative log-loss for binary response (`rmse` serves this purpose for continuous responses), a function, or a function-evaluation environment

list-pair. Functions should have prototypes of the form `function(y.test, y.test.hat, weights)`, where `y.test` is the held out test subsample, `y.test.hat` is a matrix of dimension `length(y.test) * n.samples`, and `weights` are an optional vector of user-supplied weights. See examples.

`n.threads` Across different sets of parameters (`k` × `power` × `base` × `n.trees`) and `n.reps`, results are independent. For `n.threads > 1`, evaluations of the above are divided into approximately equal size evaluations chunks and executed in parallel. The default uses `link{guessNumCores}`, which should work across the most common operating system/hardware pairs. A value of `NA` is interpretted as 1.

`n.trees` A vector of positive integers setting the BART hyperparameter for the number of trees in the sum-of-trees formulation. See `bart`.

`k` A vector of positive real numbers, setting the BART hyperparameter for the node-mean prior standard deviation. If `NULL`, the default of `bart2` will be used - 2 for continuous response and a Chi hyperprior for binary. Hyperprior crossvalidation not possible at this time.

`power` A vector of real numbers greater than one, setting the BART hyperparameter for the tree prior's growth probability, given by $base/(1 + depth)^{power}$.

`base` A vector of real numbers in $(0, 1)$, setting the BART hyperparameter for the tree prior's growth probability.

`drop` Logical, determining if dimensions with a single value are dropped from the result.

`resid.prior` An expression of the form `chisq` or `chisq(df, quant)` that sets the prior used on the residual/error variance.

`control` An object inheriting from `dbartsControl`, 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.

`seed` Optional integer specifying the desired pRNG seed. It should not be needed when running single-threaded - `set.seed` will suffice, and can be used to obtain reproducible results when multi-threaded. See Reproducibility section of `bart`.

Details

Crossvalidates `n.reps` replications against the crossproduct of given hyperparameter vectors `n.trees` × `k` × `power` × `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 average negative log-loss for binary outcomes and root-mean-squared error for continuous outcomes, missclassification rates for binary 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. RMSE is a monotonic transformation of the average log-loss for continuous outcomes, so specifying log-loss in that case calculates RMSE instead.

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, weights) {
    # note, weights are ignored
    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