Package ‘pgbart’

March 13, 2019

Type Package

Title Bayesian Additive Regression Trees Using Particle Gibbs Sampler and Gibbs/Metropolis-Hastings Sampler

Version 0.6.16

Author Pingyu Wang [aut, cre],
Dai Feng [aut],
Yang Bai [aut],
Qiuyue Shi [aut],
Zhicheng Zhao [aut],
Fei Su [aut],
Hugh Chipman [aut],
Robert McCulloch [aut]

Maintainer Pingyu Wang <applewangpingyu@gmail.com>

Description The Particle Gibbs sampler and Gibbs/Metropolis-Hastings sampler were implemented to fit Bayesian additive regression tree model. Construction of the model (training) and prediction for a new data set (testing) can be separated. Our reference papers are:
Lakshminarayanan B, Roy D, Teh Y W. Particle Gibbs for Bayesian additive regression trees[C], Artificial Intelligence and Statistics. 2015: 553-561,
<http://proceedings.mlr.press/v38/lakshminarayanan15.pdf>
and Chipman, H., George, E., and McCulloch R. (2010) Bayesian Additive Regression Trees. The Annals of Applied Statistics, 4,1, 266-298, <doi:10.1214/09-aoas285>.

Depends R (>= 3.2.2)

Imports BayesTree (>= 0.3-1.4)

License GPL (>= 2)

Encoding UTF-8

NeedsCompilation yes

Repository CRAN

Date/Publication 2019-03-13 06:40:03 UTC

R topics documented:

`pdpgbart` *Partial Dependence Plots for PGBART*

Description

Display the effect of a single variable (`pdpgbart`) or pair of variables (`pd2pgbart`). Note that if response 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

```
pdpgbart(
   x.train, y.train,
   xind=1:ncol(x.train), levs=NULL, levquants=c(.05,(1:9)/10,0.95),
   pl=TRUE,  plquants=c(.05,.95),
    ...)

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

pd2pgbart(
   x.train, y.train,
   xind=1:2, levs=NULL, levquants=c(.05,(1:9)/10,.95),
   pl=TRUE, plquants=c(.05,.95),
   ...)

## S3 method for class 'pd2pgbart'
plot(x,
     plquants = c(0.05, 0.95), contour.color = 'white',
     justmedian = TRUE,
     ...)
```

Arguments

`x.train`	Explanatory variables for training (in sample) data. May be a matrix or a data frame, with (as usual) 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 is numeric a continous response model is fit (normal errors). If y is a factor (or just has values 0 and 1) then a binary response model with a probit link is fit.
`xind`	Integer vector indicating which variables are to be plotted. In `pdpgbart`, corresponds to the variables (columns of `x.train`) for which a plot is to be constructed. In `plotpdpgbart`, corresponds to the indices in list returned by `pdpgbart` for which plot is to be constructed. In `pd2pgbart`, the indicies of a pair of variables (columns of `x.train`) to plot.
`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 `pdpgbart`, it should have same length as `xind`. In `pd2pgbart`, it should have length 2. See also argument `levquants`.
`levquants`	If `levs` in `NULL`, the values of each variable used in the plot are set to the quantiles (in `x.train`) indicated by levquants. Must be a vector of numeric type.
`pl`	For `pdpgbart` and `pd2pgbart`, 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 `pgbart_train`. In `plot.pdbart`, they are passed on to `plot`. In `plot.pd2bart`, they are passed on to `image`.
`x`	For `plot.*`, object is returned from `pdpgbart` or `pd2pgbart`.
`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 pgbart 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 pgbart 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 pdpgbart x_s consists of a single variable in x and in pd2pgbart 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 pgbart draw. For this reason running pgbart with keepevery larger than 1 (eg. 10) makes the procedure much faster.

Value

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

pdpgbart and pd2pgbart return lists with components given below. The list returned by pdpgbart is assigned class pdpgbart and the list returned by pd2pgbart is assigned class pd2pgbart.

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 pdpgbart, 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 pd2pgbart, 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 pgbart_train. They are simply passed on from the pgbart run used to create the partial dependence plot.

References

Lakshminarayanan B, Roy D, Teh Y W. (2015) Particle Gibbs for Bayesian Additive Regression Trees *Artificial Intelligence and Statistics*, 553-561.

Chipman, H., George, E., and McCulloch R. (2010) Bayesian Additive Regression Trees. *The Annals of Applied Statistics*, **4,1**, 266-298.

Friedman, J. H. (2001) Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics*, **29**, 1189–1232.

Examples

```
## 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 <-
  pdpgbart(
    x, y, xind=c(1,2),
    levs=list(seq(-1,1,.2), seq(-1,1,.2)), pl=FALSE,
    keepevery=10, ntree=5, nskip=100, ndpost=200
  )
plot(pdb1,ylim=c(-.6,.6))
## pd2bart: two dimensional partial dependence plot
set.seed(99)
pdb2 <-
  pd2pgbart(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 = 5, keepevery = 10, verbose = FALSE
  )
plot(pdb2)
```

`pgbart_predict` *Make Predictions Using Bayesian Additive Regression Trees*

Description

Make predictions for a new test data set after building a model using trainng data by function `pgbart_train`.

Usage

```
pgbart_predict(x.test, model)
```

Arguments

`x.test`	Explanatory variables for test (out of sample) data. Should have same structure as `x.train` in `pgbart_train`.
`model`	The path to save the model file as specified in `pgbart_train`.

Details

PGBART is an Bayesian MCMC method. At each MCMC interation, 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 modelling methods in R, we do 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 a row from the test data (x.test).

Value

`pgbart_predict` returns a list assigned class 'pgbart'. In the numeric y case, the list has components:

`yhat.test`	A matrix with (ndpost/keepevery) rows and nrow(x.test) columns. Each row corresponds to a draw f^* from the posterior of f and each column corresponds to a row of x.test. The (i, j) value is $f^*(x)$ for the i^{th} kept draw of f and the j^{th} row of x.test. Burn-in is dropped.
`yhat.test.mean`	Test data fits = mean of yhat.test columns. Only exists when y is not binary.

In the binary y case, the returned list has the components yhat.test and binaryOffset.

Note that in the binary y case, yhat.test is $f(x)$ + binaryOffset. If you want draws of the probability $P(Y = 1|x)$ you need to apply the normal cdf (`pnorm`) to these values.

References

Chipman, H., George, E., and McCulloch R. (2010) Bayesian Additive Regression Trees. *The Annals of Applied Statistics*, **4,1**, 266-298.

Lakshminarayanan B, Roy D, Teh Y W. (2015) Particle Gibbs for Bayesian Additive Regression Trees *Artificial Intelligence and Statistics*, 553-561.

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.

Friedman, J.H. (1991) Multivariate Adaptive Regression Splines. *The Annals of Statistics*, **19**, 1–67.

Breiman, L. (1996) Bias, Variance, and Arcing Classifiers. *Tech. Rep.* **460**, Statistics Department, University of California, Berkeley, CA, USA.

See Also

`pgbart_train`, `pdpgbart`

Examples

```
##Example 1: simulated continuous outcome data (example from section 4.3 of Friedman's MARS paper)
f = function(x){
    10*sin(pi*x[,1]*x[,2]) + 20*(x[,3]-.5)^2+10*x[,4]+5*x[,5]
}
sigma = 1.0  #y = f(x) + sigma*z , z~N(0,1)
n = 100      #number of observations
set.seed(99)
x = matrix(runif(n*10), n, 10)
Ey = f(x)
y = Ey+sigma*rnorm(n)
model_path = file.path(tempdir(),'pgbart.model')
pgbartFit = pgbart_train(x[1:(n*.75),], y[1:(n*.75)],
                         model=model_path,
                         ndpost=200, ntree=5, usepg=TRUE)
```

```
pgbartPredict = pgbart_predict(x[(n*.75+1):n,], model=model_path)

cor(pgbartPredict$yhat.test.mean, y[(n*.75+1):n])

##Example 2: simulated binary outcome data (two normal example from Breiman)
f <- function (n, d = 20)
{
  x <- matrix(0, nrow = n, ncol = d)
  c1 <- sample.int(n, n/2)
  c2 <- (1:n)[-c1]
  a <- 2/sqrt(d)
  x[c1, ] <- matrix(rnorm(n = d * length(c1), mean = -a), ncol = d)
  x[c2, ] <- matrix(rnorm(n = d * length(c2), mean = a), ncol = d)

  x.train <- x
  y.train <- rep(0, n)
  y.train[c2] <- 1
  list(x.train=x.train, y.train=as.factor(y.train))
}


set.seed(99)
n <- 200
train <- f(n)
model_path = file.path(tempdir(),'pgbart.model')
pgbartFit = pgbart_train(train$x.train[1:(n*.75),], train$y.train[1:(n*.75)],
                         model=model_path, ndpost=200, ntree=5, usepg=TRUE)
pgbartPredict = pgbart_predict(train$x.train[(n*.75+1):n,], model=model_path)
class.pred = ifelse(colMeans(apply(pgbartPredict$yhat.test, 2, pnorm)) <= 0.5, 0, 1)
table(class.pred, train$y.train[(n*.75+1):n])
```

pgbart_train *Train Bayesian Additive Regression Trees Using PG Sampler or Gibbs/NH Sampler*

Description

Build a model based on training data or combine training and test procedures.

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

In both cases, f is the sum of many tree models. The goal is to have very flexible inference for the uknown function f.

In the spirit of “ensemble models”, each tree is constrained by a prior to be a weak learner so that it contributes a small amount to the overall fit.

Usage

```
pgbart_train(
   x.train, y.train, model,x.test=matrix(0.0,0,0),
   usepg=TRUE, numparticles=10,
   sigest=NA, sigdf=3, sigquant=.90,
   k=2.0,
   power=2.0, base=.95,
   binaryOffset=0,
   ntree=200,
   ndpost=1000, nskip=100,
   printevery=100, keepevery=1, keeptrainfits=TRUE,
   usequants=FALSE, numcut=100, printcutoffs=0,
   verbose=TRUE)
## S3 method for class 'pgbart'
plot(
   x,
   plquants=c(.05,.95), cols =c('blue','black'),
   ...)
```

Arguments

`x.train`	Explanatory variables for training (in sample) data. May be a matrix or a data frame, with (as usual) 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. `pgbart_train` will generate draws of $f(x)$ for each x which is a row of x.train.
`y.train`	Dependent variable for training (in sample) data. If y is numeric, a continous response model is fit (normal errors). If y is a factor (or just has values 0 and 1), then a binary response model with a probit link is fit.
`model`	The path to save a model file which contains details of the trees constructed.
`x.test`	Explanatory variables for test (out of sample) data. Should have same structure as x.train. `pgbart_train` will generate draws of $f(x)$ for each x which is a row of x.test.
`usepg`	Two sampling methods: "pg" and "cgm". The first method implements the particle Gibbs sampler in Lakshminarayanan et al. (2015). The second implements the Gibbs/Metropolis-Hastings sampler in Chipman et al. (2010). If true, sampling method is "pg". Otherwise, sampling method is "cgm".
`numparticles`	The number of particles used in "pg" sampler.
`sigest`	The prior for the error variance (σ^2) is inverted chi-squared (the standard conditionally conjugate prior). The prior is specified by choosing the degrees of freedom, a rough estimate of the corresponding standard deviation and a quantile to put this rough estimate at. If sigest=NA then the rough estimate will be the usual least squares estimator. Otherwise the supplied value will be used. Not used if y is binary.

`sigdf`	Degrees of freedom for error variance prior. Not used if y is binary.
`sigquant`	The quantile of the prior that the rough estimate (see 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 used if y is binary.
`k`	For numeric y, k is the number of prior standard deviations $E(Y\|x) = f(x)$ is away from +/-.5. The response (y.train) is internally scaled to range from -.5 to .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.
`power`	Power parameter for tree prior.
`base`	Base parameter for tree prior.
`binaryOffset`	Used for binary y. The model is $P(Y = 1\|x) = F(f(x) + binaryOffset)$. The idea is that f is shrunk towards 0, so the offset allows you to shrink towards a probability other than .5.
`ntree`	The number of trees in the sum.
`ndpost`	The number of posterior draws after burn in, ndpost/keepevery will actually be returned.
`nskip`	Number of MCMC iterations to be treated as burn in.
`printevery`	As the MCMC runs, a message is printed per printevery draws.
`keepevery`	Every keepevery draw is kept to be returned to the user. A “draw” will consist of values of the error standard deviation (σ) and $f^*(x)$ at x = rows from the train(optionally) and test data, where f^* denotes the current draw of f.
`keeptrainfits`	If true the draws of $f(x)$ for x = rows of x.train are returned.
`usequants`	Decision rules in the tree are of the form $x \leq c$ vs. $x > c$ for each variable corresponding to a column of x.train. usequants determines how the set of possible c is determined. If usequants is true, then the c is a subset of the values (xs[i]+xs[i+1])/2 where xs is 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.
`numcut`	The number of possible values of c (see usequants). If a single number if given, this is used for all variables. Otherwise a vector with length equal to ncol(x.train) is required, where the i^{th} element gives the number of c used for the i^{th} variable in x.train. 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 min(numcut, the number of unique values in the corresponding columns of x.train - 1) c values are used.
`printcutoffs`	The number of cutoff rules c to be printed to screen before the MCMC is run. Give a single integer, the same value will be used for all variables. If 0, nothing is printed.
`verbose`	Logical, if FALSE supress printing.
`x`	For `plot.*`, object returned from `pdpgbart` or `pd2pgbart`.

`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.
`...`	Additional arguments passed on to plot.

Details

PGBART is a Bayesian MCMC method. At each MCMC interation, 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 modelling methods in R, we do 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).

Value

The function returns a list assigned class 'pgbart'. In the numeric y case, the list has components:

`yhat.train`	A matrix with (ndpost/keepevery) rows and nrow(x.train) columns. Each row corresponds to a draw f^* from the posterior of f and each column corresponds to a row of x.train. The (i, j) value is $f^*(x)$ for the i^{th} kept draw of f and the j^{th} row of x.train. Burn-in is dropped.
`yhat.test`	same as yhat.train but now the x's are the rows of the test data if `x.test` is specified. Otherwise, it will be `NULL`.
`yhat.train.mean`	train data fits = mean of yhat.train columns.
`yhat.test.mean`	test data fits = mean of yhat.test columns if `x.test` is specified. Otherwise, it will be `NULL`.
`sigma`	post burn in draws of sigma, length = ndpost/keepevery.
`first.sigma`	burn-in draws of sigma.
`varcount`	a matrix with (ndpost/keepevery) rows and nrow(x.train) columns. Each row is for a draw. For each variable (corresponding to the columns), the total count of the number of times that variable is used in a tree decision rule (over all trees) is given.
`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 plotpgbart.

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 $f(x)$ + binaryOffset. If you want draws of the probability $P(Y = 1|x)$ you need to 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 pgbart 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.

References

Chipman, H., George, E., and McCulloch R. (2010) Bayesian Additive Regression Trees. *The Annals of Applied Statistics*, **4,1**, 266-298.

Lakshminarayanan B, Roy D, Teh Y W. (2015) Particle Gibbs for Bayesian Additive Regression Trees *Artificial Intelligence and Statistics*, 553-561.

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.

Friedman, J.H. (1991) Multivariate Adaptive Regression Splines. *The Annals of Statistics*, **19**, 1–67.

Breiman, L. (1996) Bias, Variance, and Arcing Classifiers. *Tech. Rep.* **460**, Statistics Department, University of California, Berkeley, CA, USA.

See Also

pdpgbart

Examples

```
##Example 1: simulated continuous outcome data (example from section 4.3 of Friedman's MARS paper)
f = function(x){
    10*sin(pi*x[,1]*x[,2]) + 20*(x[,3]-.5)^2+10*x[,4]+5*x[,5]
}
sigma = 1.0  #y = f(x) + sigma*z , z~N(0,1)
n = 100      #number of observations
set.seed(99)
x = matrix(runif(n*10),n,10) #10 variables, only first 5 matter
Ey = f(x)
y = Ey+sigma*rnorm(n)
lmFit = lm(y~.,data.frame(x,y)) #compare lm fit to pgbart later
##run pgBART
set.seed(99)
model_path = file.path(tempdir(),'pgbart.model')
pgbartFit = pgbart_train(x, y, model=model_path,ndpost=200, ntree=5, usepg=TRUE)
plot(pgbartFit) # plot pgbart fit
##compare pgbart fit to linear matter and truth = Ey
fitmat = cbind(y,Ey,lmFit$fitted,pgbartFit$yhat.train.mean)
colnames(fitmat) = c('y','Ey','lm','pgbart')
print(cor(fitmat))

##Example 2: simulated binary outcome data (two normal example from Breiman)
f <- function (n, d = 20)
{
  x <- matrix(0, nrow = n, ncol = d)
  c1 <- sample.int(n, n/2)
  c2 <- (1:n)[-c1]
  a <- 2/sqrt(d)

  x[c1, ] <- matrix(rnorm(n = d * length(c1), mean = -a), ncol = d)
  x[c2, ] <- matrix(rnorm(n = d * length(c2), mean = a), ncol = d)

  x.train <- x
  y.train <- rep(0, n)
  y.train[c2] <- 1
  list(x.train=x.train, y.train=as.factor(y.train))
}

#
set.seed(99)
train <- f(200)
model_path = file.path(tempdir(),'pgbart.model')
pgbartFit = pgbart_train(train$x.train, train$y.train,
                        model=model_path,
                        ndpost=200, ntree=5, usepg=TRUE)
class.pred = ifelse(colMeans(apply(pgbartFit$yhat.train, 2, pnorm)) <= 0.5, 0, 1)
table(class.pred, train$y.train)
```

Index