Package ‘rbart’

Type  Package
Title  Bayesian Trees for Conditional Mean and Variance
Version 1.0
Date 2019-07-28

Description
A model of the form $Y = f(x) + s(x) \, Z$ is fit where functions $f$ and $s$ are modeled with ensembles of trees and $Z$ is standard normal.
This model is developed in the paper 'Heteroscedastic BART Via Multiplicative Regression Trees' (Pratola, Chipman, George, and McCulloch, 2019, <arXiv:1709.07542v2>).
BART refers to Bayesian Additive Regression Trees. See the R-package 'BART'.
The predictor vector $x$ may be high dimensional.
A Markov Chain Monte Carlo (MCMC) algorithm provides Bayesian posterior uncer-
tainty for both $f$ and $s$.
The MCMC uses the recent innovations in Efficient Metropolis–Hastings proposal mechanisms for Bayesian regression tree models (Pratola, 2015, Bayesian Analysis, <doi:10.1214/16-BA999>).

License GPL (>= 2)

Depends R (>= 2.10)
Imports Rcpp (>= 0.12.3)
Suggests knitr, rmarkdown, MASS, nnet

LinkingTo Rcpp

SystemRequirements C++11

NeedsCompilation yes

Author Robert McCulloch [aut, cre, cph],
   Matthew Pratola [aut, cph],
   Hugh Chipman [aut, cph]

Maintainer Robert McCulloch <robert.e.mcculloch@gmail.com>

Repository CRAN

Date/Publication 2019-08-01 09:20:02 UTC

1
R topics documented:

- hbartqqplot ........................................... 2
- plotFunctionDraws ................................. 3
- predict.rbart ........................................ 4
- rbart .................................................. 7
- rbartModelMatrix ................................. 11
- rbartonsimd ......................................... 11
- simdat ............................................... 12
- ucarprice .......................................... 14

Index .............................................. 15

---

hbartqqplot  

*Predictive qqplot for heterbart*

**Description**

Given results from `predict.rbart`, gets draws from the predictive distribution at each x and then computes the empirical inverse cdf to get draws which would be uniform if the predictive were the true distribution. Then draws a qqplot against the uniform. In large enough samples, if the model is correct, the qqplot should look like like a straight line with intercept 0 and slope 1. In small samples, we expect the predictive to be more spread out than the true distribution, even if the model is correct.

**Usage**

```
hbartqqplot(y, rbmod, nunif=10000, linecolor="red", linewd=3,...)
```

**Arguments**

- `y`  
  y values corresponding to x values given to `predict.rbart`.

- `rbmod`  
  Output list from `predict.rbart`.

- `nunif`  
  Number of uniform(0,1) draws used in constructing the qqplot.

- `linecolor, linewd`  
  Line color and width for (0,1) line.

- `...`  
  Arguments passed on to `stats::qqplot`.

**Value**

quantiles of y in draws from the predictive (conditional on each x value).
plotFunctionDraws

Examples

########################################################################
## please see vignette and/or www.rob-mcculloch.org for more realistic examples
########################################################################

## get simulated data
data(simdat)

## get rbart run on the simulated data
data(rbartonsimd)

## Predictive quantile-quantile plot
temp = hbartqqplot(simdat$yp,rbartonsimd,xlab="predictive quantile",ylab="uniform",
cex.axis=1.4,cex.lab=1.2)

########################################################################

plotFunctionDraws

Plot matrix of function draws evaluated on a set of x

Description

Given draws of a function $f_d$, $d = 1, 2, \ldots, D$ and a set of $x$ vectors $x_j$, $j = 1, 2, \ldots, J$, we have a $D \times J$ matrix of evaluations whose $(d, j)$ element is $f_d(x_j)$, the $d^{th}$ draw of the function evaluated at the $j^{th}$ $x$. This function plots the draws by plotting estimates of $f(x_j)$ versus intervals for $f(x_j)$. The estimates are the mean of the $j^{th}$ column and the intervals are two quantiles of the $j^{th}$ column (e.g. 5% and 95%).

Usage

plotFunctionDraws(fd,complevel=mean(fd),probs=c(.025,.975),
                  xlab="posterior mean of function",ylab="posterior intervals",
                  intervalcol="green",linecol="red",
                  pts=NA,ptscol="blue", ptspch=1, ptscex=1, ...)

Arguments

- **fd**: $D \times J$ matrix whose $(d, j)$ element is the $d^{th}$ function draw evaluated at the $j^{th}$ $x$.
- **complevel**: A horizontal line is drawn a complevel to compare the intervals to.
- **probs**: The two quantiles used to construct the intervals.
- **xlab**: Label for x axis.
- **ylab**: Label for y axis.
- **intervalcol**: Color to draw the intervals with.
- **linecol**: Color to draw the comparison horizontal line with.
- **pts**: Add $(x_j, pts_j)$ to the plot. For example pts could be fitted values from and alternative model such as the linear model.
predict.rbart

**Value**

NULL

**Examples**

```
############################################################
## please see vignette and/or www.rob-mcculloch.org for more realistic examples
############################################################

## get simulated data
data(simdat)

## get rbart run on the simulated data
data(rbartonsimd)

## plot function (f and s) draws
shat = sqrt(mean((simdat$yp-rbartonsimd$mmean)^2)) #overall estimate of sigma
lmfit = lm(y~x,data.frame(x=simdat$x,y=simdat$y))
yhatlm = predict(lmfit,data.frame(x=simdat$xp)) #fits from a linear model

#Now we use plotFunctionDraws to look at mdraws (left panel) and sdraws (right panel).

## in the mean inference, you can see that the linear model seem unlikely
## in the variance inference, you can see that the posteriors of s(x) are far from a constant value
par(mfrow=c(1,2))

## look at mean inference
plotFunctionDraws(rbartonsimd$mdraws,complevel=mean(simdat$y), probs=c(.05,.95),
                 xlab=expression(hat(f)(x)), pts=yhatlm, ptscol="black",
P20.1=1.2, P20.2=1.4, main="intervals for f(x)"

## look at the standard deviation inference
plotFunctionDraws(rbartonsimd$sdraws, complevel=shat, xlab=expression(hat(s)(x)),
                  intervalcol="magenta", linecol="blue",
P20.1=1.2, P20.2=1.4, main="intervals for s(x)"
```

**predict.rbart**

*Drawing Posterior Predictive Realizations for rbart models.*

**Description**

The function `predict.rbart()` is the main function for drawing posterior predictive realizations at new inputs using a fitted model stored in a `rbart` object returned from `rbart()`.
predict.rbart

Usage

## S3 method for class 'rbart'
predict(
  object,
  x.test=object$x.train,
  tc=1,
  fmean=mean(object$y.train),
  q.lower=0.025,
  q.upper=0.975,...)

Arguments

- **object**: Object of type rbart from a previous call to rbart().
- **x.test**: New input settings in the form of an npred x p matrix at which to construct predictions. Defaults to the training inputs.
- **tc**: Number of OpenMP threads to use for parallel computing.
- **fmean**: Mean-centering vector for the training data. Defaults to the value used by rbart() when fitting the model. Usually should be left to the default.
- **q.lower**: Lower quantile to return.
- **q.upper**: Upper quantile to return.
- **...**: not used.

Details

predict.rbart() is the main function for calculating posterior predictions and uncertainties once a model has been fit by rbart().

Returns an object of type rbart with the following entries.

Value

- **mdraws**: Posterior realizations of the mean function, \( f(x) \) stored in an ndpost x npred matrix, where ndpost is the number of kept MCMC draws in the rbart run.
- **sdraws**: Posterior realizations of the standard deviation function, \( s(x) \) stored in an ndpost x npred matrix, where ndpost is the number of kept MCMC draws in the rbart run.
- **mmean**: Posterior predictive mean of \( f(x) \).
- **smean**: Posterior predictive mean of the standard deviation, \( s(x) \).
- **msd**: Posterior standard deviation of the mean, \( f(x) \).
- **ssd**: Posterior standard deviation of the standard deviation, \( s(x) \).
- **m.5**: Posterior median of the mean function realizations, \( f(x) \).
- **m.lower**: Posterior q. lower quantile of the mean function realizations.
- **m.upper**: Posterior q. upper quantile of the mean function realizations.
- **s.5**: Posterior median of the standard deviation function realizations, \( s(x) \).
predict.rbart

s.lower  Posterior q.lower quantile of the standard deviation function realizations.

s.upper  Posterior q.upper quantile of the standard deviation function realizations.

q.lower  Lower quantile used in constructing the above.

q.upper  Upper quantile used in constructing the above.

Author(s)
Matthew T. Pratola <mpratola@stat.osu.edu> [aut, cre, cph], Robert E. McCulloch <robert.e.mcculloch@gmail.com> [aut, cre, cph], Hugh Chipman <hugh.chipman@gmail.com> [aut, cph] Maintainer: Matthew T. Pratola <mpratola@stat.osu.edu>, Robert E. McCulloch <robert.e.mcculloch@gmail.com>

References


See Also

rbart

Examples

########################################################################
## This is just a stub (runs fast) example for testing.
## For more realistic examples, please see:
## (i) the vignette at www.rob-mcculloch.org
## (ii) the example simulated data (see ?simdat)
## and the longer run in ?rbartonsimd,
## where a saved run of rbart is run on simdat is plotted.
########################################################################

## simulate data
set.seed(99)

# train data
n=500 # train data sample size
p=1 # just one x
x = matrix(sort(runif(n*p)),ncol=p) # iid uniform x values
fx = 4*(x[,1]^2) # quadratic function f
sx = .2*exp(2*x[,1]) # exponential function s
y = fx + sx*rnorm(n) # y = f(x) + s(x) Z
#test data (the p added to the variable names is for predict)
np=500 #test data sample size
xp = matrix(sort(runif(np*p)),ncol=p)
fxp = 4*(xp[,1]^2)
sxp = .2*exp(2*xp[,1])
yp = fxp + sxp*rnorm(np)

##run rbart MCMC
# The number of interations is kept small to make example run,
### REAL APPLICATIONS MAY NEED LONGER RUNS !!!!
# nskip: burn in draws,
# ndpost:kept draws,
# nadapt: initial draws to tune MCMC,
# numcut: number of cutpoints used for each x
# k: bigger k gives smoother f (default is 2)
set.seed(19)
res = rbart(x,y,nskip=10,ndpost=20,nadapt=0,numcut=1000,k=5) #again, this is way too short a run!!!
## now predict to get inference
resp = predict(res,x.test=xp)

##check out of sample fit
cat("out of sample cor(f,fhat) is ",cor(fxp,resp$mmean),"\n")
cat("out of sample cor(s,shat) is ",cor(sxp,resp$smean),"\n")

##plot estimated vs. true
##plot the data
plot(xp,yp,cex.axis=1.5,cex.lab=1.5)
lines(xp,fxp,col="blue")
lines(xp,fx+2*sxp,col="blue",lty=2)
lines(xp,fxp-2*sxp,col="blue",lty=2)

## add the fit
lines(xp,resp$mmean) #estimate of f
lines(xp,resp$mmean+2*resp$smean) #estimate of sd
lines(xp,resp$mmean-2*resp$smean) #estimate of sd

---

Fitting Bayesian Regression Tree models supported by rbart.

Description
The function rbart() is the main function for fitting Bayesian Regression Tree models, including single-tree models, Bayesian Additive Regression Tree (BART) models and Heteroscedastic BART models. rbart() maintains some degree of backwards compatibility with BayesTree::bart(), while offering many new options.

Usage
rbart(x.train, y.train, x.test=matrix(0.0,0,0), ntree=200, ntreeh=40, ndpost=1000, nskip=100, k=2, power=2.0, base=.95, tc=1, sigmav=rep(1,length(y.train)),

fmean=mean(y.train), overallsd = sd(y.train), overallnu=10, 
chv = cor(x.train,method="spearman"), pbd=.7, pb=.5, stepwpert=.1, probchv=.1, 
minnumber=5, printevery=100, numcut=100, xicuts=NULL, nadapt=1000, adaptevery=100, 
summarystats=FALSE)

Arguments

- **x.train**: nxp matrix of predictor variables for the training data.
- **y.train**: nx1 vector of the observed response for the training data.
- **x.test**: mxp matrix of predictor variables for the test set. Deprecated.
- **ntree**: Number of trees used for the mean model.
- **ntreeh**: Number of trees used for the variance model.
- **ndpost**: Number of iterations to run the MCMC algorithm after burn-in.
- **nskip**: Number of MCMC iterations treated as burn-in and discarded.
- **k**: Prior hyperparameter for the mean model.
- **power**: Power parameter in the tree depth penalizing prior.
- **base**: Base parameter in the tree depth penalizing prior.
- **tc**: Number of OpenMP threads to use.
- **sigmav**: Initialization of square-root of variance parameter.
- **fmean**: Overall mean of the data for pre-centering the data before running the model.
- **overallsd**: A rudimentary estimate of the process standard deviation. Used in calibrating the variance prior.
- **overallnu**: Shape parameter for the variance prior.
- **chv**: Predictor correlation matrix used as a pre-conditioner for MCMC change-of-variable proposals.
- **pbd**: Probability of performing a birth/death proposal, otherwise perform a rotate proposal.
- **pb**: Probability of performing a birth proposal given that we choose to perform a birth/death proposal.
- **stepwpert**: Initial width of proposal distribution for peturbing cutpoints.
- **probchv**: Probability of performing a change-of-variable proposal. Otherwise, only do a perturb proposal.
- **minnumber**: Minimum number of observations required in bottom (terminal) nodes.
- **printevery**: Outputs MCMC algorithm status every printevery iterations.
- **numcut**: Number of cutpoints to use for each predictor variable.
- **xicuts**: More detailed construction of cutpoints can be specified using makecuts() and passed as an argument here.
- **nadapt**: Number of MCMC iterations allowed for adaptive MCMC. These are also discarded.
- **adaptevery**: Adapt MCMC proposal distributions every adaptevery iterations until the algorithm has run for nadapt iterations.
- **summarystats**: Return detailed summary statistics about the fitting procedure.
Details

rbart() is the main model fitting function for continuous response data. The most general form of
the model allowed is \( Y(x) = f(x) + s(x)Z \) where \( Z \) is \( N(0, 1) \) and \( f(x) = \sum_{j=1}^{m} g(x; T_j, M_j) \)
and \( s(x) = \prod_{j=1}^{m'} h(x; T'_j, M'_j) \), where the \( g(\cdot; T_j, M_j) \) represent additive tree components used for
modeling the mean and \( h(\cdot; T'_j, M'_j) \) represent multiplicative tree components used for modeling the
variance.

The most common models to fit are a homoscedastic single-tree model, a homoscedastic BART
model and a heteroscedastic BART model.

For a BART model, set \( pbd=c(0.7, 0.0) \) and \( ntreeh=1 \). This forces a scalar (homoscedastic)
variance term.

For a single-tree model, set \( pbd=(0.7, 0.0) \), \( ntreeh=1 \) and \( ntree=1 \). This forces the mean com-
ponent to be modeled using only one tree.

The heteroscedastic BART model is the default.

Value

res Fitted model object of S3 class rbart.

Author(s)

Matthew T. Pratola <mpratola@stat.osu.edu> [aut, cre, cph], Robert E. McCulloch <robert.e.mcculloch@gmail.com>
[aut, cre, cph], Hugh Chipman <hugh.chipman@gmail.com> [aut, cph] Maintainer: Matthew T.
Pratola <mpratola@stat.osu.edu>, Robert E. McCulloch <robert.e.mcculloch@gmail.com>

References


Pratola, Matthew T. (2016) Efficient Metropolis Hastings proposal mechanisms for Bayesian re-

1–20.

See Also

predict.rbart

Examples

# This is just a stub (runs fast) example for testing.
# For more realistic examples, please see:
# (i) the vignette at www.rob-mcculloch.org
# (ii) the example simulated data (see ?simdat)
## and the longer run in ?rbartonsimd,
## where a saved run of rbart is run on simdat is plotted.

```
# simulate data
set.seed(99)

# train data
n=500  # train data sample size
p=1    # just one x
x = matrix(sort(runif(n*p)),ncol=p)  # iid uniform x values
fx = 4*(x[,1]^2)  # quadratic function f
sx = .2*exp(2*x[,1])  # exponential function s
y = fx + sx*runif(n)  # y = f(x) + s(x) Z

# test data (the p added to the variable names is for predict)
np=500  # test data sample size
xp = matrix(sort(runif(np*p)),ncol=p)
fxp = 4*(xp[,1]^2)
sxp = .2*exp(2*xp[,1])
yp = fxp + sxp*runif(np)

## run rbart MCMC
# The number of interations is kept small to make example run,
#!!!! REAL APPLICATIONS MAY NEED LONGER RUNS !!!!
# nskip: burn in draws,
# ndpost: kept draws,
# nadapt: initial draws to tune MCMC,
# numcut: number of cutpoints used for each x
# k: bigger k gives smoother f (default is 2)
set.seed(19)
res = rbart(x,y,nskip=10,ndpost=20,nadapt=0,numcut=1000,k=5)  # again, this is way too short a run!!!

## now predict to get inference
resp = predict(res,x.test=xp)

## check out of sample fit
cat("out of sample cor(f,fhat) is ",cor(fxp,resp$mmean),"\n")
cat("out of sample cor(s,shat) is ",cor(sxp,resp$smean),"\n")

## plot estimated vs. true
## plot the data
plot(xp,yp,cex.axis=1.5,cex.lab=1.5)
lines(xp,fxp,col="blue")
lines(xp,fxp+2*sxp,col="blue",lty=2)
lines(xp,fxp-2*sxp,col="blue",lty=2)

## add the fit
lines(xp,resp$mmean)  # estimate of f
lines(xp,resp$mmean+2*resp$smean)  # estimate of sd
lines(xp,resp$mmean-2*resp$smean)  # estimate of sd
```
rbartModelMatrix

Model Matrix for BART

Description

rbartModelMatrix takes a data frame of explanatory variables (x) and turns it into a numeric matrix suitable for BART. This is used when some of the x variables are factors. The returned matrix will first have columns for all the numeric variables in the data frame and then columns for all the factors expanded into binary dummy variables. Note that if a factor has k levels, then BART wants k dummies (not k-1 as in linear regression). So for example, a factor with two levels named xf, will result in two columns named xf1 and xf2.

Usage

rbartModelMatrix(xdf)

Arguments

xdf

Data frame to be turned into a matrix for BART.

Value

Matrix from of explanatory variables in the data frame xdf.

Examples

set.seed(99)
xdf = data.frame(x1=1:15,x2=as.factor(c(rep(1,5),rep(2,5),rep(3,5))),
x3=runif(15),x4=as.factor(c(rep(1,5),rep(2,10))))
print(head(xdf))

xm = rbartModelMatrix(xdf)
print(head(xm))

rbartonsimd

rbart run on simulated data

Description

predict.rbart results for simulated data.

Usage

data("rbartonsimd")
Format

rbartonsimd returned list from a call to predict.rbart on the simulated data.

Details

The data rbartonsimd is the results of an rbart run on the simulated data in simdat.

The code for the rbart run is:

```r
## load data
data(simdat)
attach(simdat) #some people think this is a bad idea

## run rbart
set.seed(99)
res = rbart(x,y,nskip=500,ndpost=100,nadapt=500,adaptevery=50)
rbartonsimd = predict(res,xp) #get prediction for test x in xp
```

Examples

```r
## load simulated data and rbart run.
data(rbartonsimd)
data(simdat)

## plot data and x vs f(x), x vs f(x) +/- 2s(x), x in test data, true and estimated
## data
plot(simdat$xp,simdat$yp)
## true
lines(simdat$xp,simdat$fxp,col="blue",lty=2,lwd=2)
lines(simdat$xp,simdat$fxp+2*simdat$sxp,col="blue",lty=2,lwd=2)
lines(simdat$xp,simdat$fxp-2*simdat$sxp,col="blue",lty=2,lwd=2)
## estimated
mhat = rbartonsimd$mmean; shat = rbartonsimd$smean
lines(simdat$xp,mhat,col="red",lty=1,lwd=2)
lines(simdat$xp,mhat + 2*shat,col="red",lty=1,lwd=2)
lines(simdat$xp,mhat - 2*shat,col="red",lty=1,lwd=2)

## note that you can get "nicer" looking fits by
## (i) running rbart longer (e.g. ndpost=500),
## (ii) using numcut=1000,k=5 in rbart.
```

simdat

**Simulated Example**

Description

Simulated data with nonlinear mean and heteroskedasticity.
simdat

Usage

data("simdat")

Format

x  simulated train x values
y  simulated train y values
xp simulated test xp values
yp simulated test yp values
fx  true f evaluated on train x
sx  true s evaluated on train x
fxp true f evaluated on test xp
sxp true s evaluated on test xp

Details

The simulated data in simdat was generated using the code:

```r
##simulate data
set.seed(99)
# train data
n=500 #train data sample size
p=1 #just one x
x = matrix(sort(runif(n*p)),ncol=p) #iid uniform x values
fx = 4*(x[,1]^2) #quadratic function f
sx = .2*exp(2*x[,1]) # exponential function s
y = fx + sx*rnorm(n) # y = f(x) + s(x) Z

#test data (the p added to the variable names is for predict)
np=1000 #test data sample size
xp = matrix(sort(runif(np*p)),ncol=p)
fxp = 4*(xp[,1]^2)
sxp = .2*exp(2*xp[,1])
yp = fkp + sxp*rnorm(np)
```

Examples

data(simdat)

```r
## plot x vs y with f(x) and f(x) +/- 2s(x) for train and test simulated data
plotted(simdat$x,simdat$y,xlab="x",ylab="y")
##test
points(simdat$xp,simdat$yp,col="red",pch=2)
lines(simdat$xp,simdat$fxp,col="blue",lwd=2)
lines(simdat$xp,simdat$fxp+2*simdat$sxp,col="blue",lwd=2,lty=2)
```
ucarprice

lines(simdat$xp,simdat$fxp-2*simdat$sxp,col="blue",lwd=2,lty=2)
legend("topleft",legend=c("train","test"),pch=c(1,2),col=c("black","red"))

ucarprice  

\textit{Used Car Prices}

\textbf{Description}

Prices of used cars and variables describing the cars.

\textbf{Usage}

\texttt{data("ucarprice")}

\textbf{Format}

A data frame with 1,000 observations on the following 7 variables.

\begin{itemize}
  \item \texttt{price} numeric: price of a car in dollars
  \item \texttt{trim} factor: trim of the car, for example, how the interior is styled
  \item \texttt{isOneOwner} factor: has the car had just one owner, t=yes, f=no
  \item \texttt{mileage} numeric: number of miles the car has been driven
  \item \texttt{year} numeric: model year of the car
  \item \texttt{color} factor: color of the car
  \item \texttt{displacement} factor: displacement of the car engine
\end{itemize}

\textbf{Examples}

\texttt{data(ucarprice)}
Index

*Topic **datasets**
  - rbartonsimd, 11
  - simdat, 12
  - ucarprice, 14

hbartqqplot, 2

plotFunctionDraws, 3
predict.rbart, 4, 9

rbart, 6, 7
rbartModelMatrix, 11
rbartonsimd, 11

simdat, 12

ucarprice, 14