Package ‘BART’

Type Package
Title Bayesian Additive Regression Trees
Version 2.7
Date 2019-12-04
License GPL (>= 2)
Depends R (>= 2.10), nlme, nnet, survival
Imports Rcpp (>= 0.12.3), parallel, tools
LinkingTo Rcpp
Suggests MASS, knitr, rmarkdown
VignetteBuilder knitr
NeedsCompilation yes
Author Robert McCulloch [aut], Rodney Sparapani [aut, cre], Robert Gramacy [aut], Charles Spanbauer [aut], Matthew Pratola [aut], Martyn Plummer [ctb], Nicky Best [ctb], Kate Cowles [ctb], Karen Vines [ctb]
Maintainer Rodney Sparapani <rsparapa@mcw.edu>
Repository CRAN
Date/Publication 2019-12-05 09:10:05 UTC

R topics documented:

BART-package ................................................................. 3
## R topics documented:

- abart
- ACTG175
- alligator
- arq
- bartModelMatrix
- bladder
- crisk bart
- crisk.pre.bart
- crisk2.bart
- draw.lambda.i
- gbart
- gbm
- gewekediag
- lbart
- leukemia
- lung
- mbart
- mbart2
- mc.cores.openmp
- mc.crisk.pwbart
- mc.crisk2.pwbart
- mc.lbart
- mc.pbart
- mc.surv.pwbart
- mc.wbart
- mc.wbart.gse
- pbart
- predict.crisk2bart
- predict.criskbart
- predict.lbart
- predict.mbart
- predict.pbart
- predict.recurbart
- predict.survbart
- predict.wbart
- recur.bart
- recur.pre.bart
- rs.pbart
- rtgamma
- rtnorm
- spectrum0ar
- srstepwise
- strats
- surv.bart
- surv.pre.bart
- transplant
- wbart

Maintainer

Rodney Sparapani: <rsparapa@mcw.edu>

Author(s)

Robert McCulloch: <robert.e.mcculloch@gmail.com>, Rodney Sparapani: <rsparapa@mcw.edu>, Robert Gramacy: <rgb@vt.edu>, Charles Spanbauer: <cspanbauer@mcw.edu> and Matthew Pratola: <mpratola@stat.osu.edu>.

Description

BART is a Bayesian “sum-of-trees” model. For a numeric response \( y \), we have \( y = f(x) + \epsilon \), where \( \epsilon \sim N(0, \sigma^2) \).

\( f \) is the sum of many tree models. The goal is to have very flexible inference for the unknown 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

abart(
  x.train, times, delta,
  x.test=matrix(0,0,0), K=100,
  type='abart', ntype=1,
  sparse=FALSE, theta=0, omega=1,
  a=0.5, b=1, augment=FALSE, rho=NULL,
  xinfo=matrix(0,0,0), usequants=FALSE,
  rm.const=TRUE,
  sigest=NA, sigdf=3, sigquant=0.90,
  k=2, power=2, base=0.95,
  lambda=NA, tau.num=c(NA, 3, 6)[ntype],
  offset=NULL, w=rep(1, length(times)),
  ntree=c(200L, 50L, 50L)[ntype], numcut=100L,
  ndpost=1000L, nskip=100L,
  keepevery=c(1L, 10L, 10L)[ntype],
  printevery=100L, transposed=FALSE,
  mc.cores = 1L, ## mc.abart only
  nice = 19L, ## mc.abart only
  seed = 99L ## mc.abart only
)

mc.abart(
  x.train, times, delta,
  x.test=matrix(0,0,0), K=100,
  type='abart', ntype=1,
  sparse=FALSE, theta=0, omega=1,
  a=0.5, b=1, augment=FALSE, rho=NULL,
  xinfo=matrix(0,0,0), usequants=FALSE,
  rm.const=TRUE,
  sigest=NA, sigdf=3, sigquant=0.90,
  k=2, power=2, base=0.95,
  lambda=NA, tau.num=c(NA, 3, 6)[ntype],
  offset=NULL, w=rep(1, length(times)),
  ntree=c(200L, 50L, 50L)[ntype], numcut=100L,
  ndpost=1000L, nskip=100L,
  keepevery=c(1L, 10L, 10L)[ntype],
  printevery=100L, transposed=FALSE,
  mc.cores = 2L, nice = 19L, seed = 99L
)
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 created if q = 2 where q is the number of levels of the factor. abart will generate draws of f(x) for each x which is a row of x.train.

times
The time of event or right-censoring.
If y.train is NULL, then times (and delta) must be provided.
delta
The event indicator: 1 is an event while 0 is censored.
If y.train is NULL, then delta (and times) must be provided.
x.test
Explanatory variables for test (out of sample) data. Should have same structure as x.train. abart will generate draws of f(x) for each x which is a row of x.test.
K
If provided, then coarsen times per the quantiles 1/K, 2/K, ..., K/K.
type
You can use this argument to specify the type of fit. 'abart' for AFT BART.
ntype
The integer equivalent of type where 'abart' is 1.
sparse
Whether to perform variable selection based on a sparse Dirichlet prior rather than simply uniform; see Linero 2016.
theta
Set theta parameter; zero means random.
omega
Set omega parameter; zero means random.
a
Sparse parameter for Beta(a, b) prior: 0.5 <= a <= 1 where lower values inducing more sparsity.
b
Sparse parameter for Beta(a, b) prior; typically, b = 1.
rho
Sparse parameter: typically rho = p where p is the number of covariates under consideration.
augment
Whether data augmentation is to be performed in sparse variable selection.
xinfo
You can provide the cutpoints to BART or let BART choose them for you. To provide them, use the xinfo argument to specify a list (matrix) where the items (rows) are the covariates and the contents of the items (columns) are the cutpoints.
usequants
If usequants=FALSE, then the cutpoints in xinfo are generated uniformly; otherwise, if TRUE, uniform quantiles are used for the cutpoints.
rm.const
Whether or not to remove constant variables.
sigest
The prior for the error variance (sigma^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.
smquant

The quantile of the prior that the rough estimate (see sigest) is placed at. The
closer the quantile is to 1, the more aggressive the fit will be as you are putting
more prior weight on error standard deviations (sigma) less than the rough es-

k

For numeric y, k is the number of prior standard deviations E(Y|x) = f(x) is
away from +/-0.5. The response, codey.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. The bigger k is, the more conservative the fitting will be.

tau.num

The numerator in the tau definition, i.e., tau=tau.num/(k*sqrt(ntree)).

offset

Continuous BART operates on y.train centered by offset which defaults to
mean(y.train). With binary BART, the centering is P(Y = 1|x) = F(f(x) +
offset) where offset defaults to f^(-1)(mean(y.train)). You can use the
offset parameter to over-ride these defaults.

w

Vector of weights which multiply the standard deviation. Not used if y is binary.

ntree

The number of trees in the sum.

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, thenumberofuniquevaluesinthecorrespondingco-
1) values are used.

ndpost

The number of posterior draws returned.

nskip

Number of MCMC iterations to be treated as burn in.

printevery

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

keepevery

Every keepevery draw is kept to be returned to the user.

transposed

When running abart in parallel, it is more memory-efficient to transpose x.train
and x.test, if any, prior to calling mc.abart.

seed

Setting the seed required for reproducible MCMC.

mc.cores

Number of cores to employ in parallel.

nice

Set the job niceness. The default niceness is 19: niceness goes from 0 (highest)
to 19 (lowest).

Details

BART is a Bayesian MCMC method. At each MCMC iteration, we produce a draw from the joint
posterior (f,σ)|(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

`abart` returns an object of type `abart` which is essentially a list. In the numeric `y` case, the list has components:

- `yhat.train`: A matrix with `ndpost` 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.
- `yhat.train.mean`: Train data fits = mean of `yhat.train` columns.
- `yhat.test.mean`: Test data fits = mean of `yhat.test` columns.
- `sigma`: Post burn-in draws of sigma, length = `ndpost`.
- `first.sigma`: Burn-in draws of sigma.
- `varcount`: A matrix with `ndpost` 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 ($\sigma$) used in the prior.

Author(s)

Robert McCulloch: `<robert.e.mcculloch@gmail.com>`,
Rodney Sparapani: `<rsparapa@mcw.edu>`.

References


See Also

- `wbart`

Examples

```r
N = 1000
P = 5    #number of covariates
M = 8
```
```r
set.seed(12)
x.train=matrix(runif(N*P, -2, 2), N, P)
mu = x.train[, 1]^3
y=rnorm(N, mu)
offset=mean(y)
T=exp(y)
C=rexp(N, 0.05)
delta=(T<C)*1
table(delta)/N
times=(T*delta+C*(1-delta))

##test BART with token run to ensure installation works
set.seed(99)
post1 = abart(x.train, times, delta, nskip=5, ndpost=10)

## Not run:
post1 = mc.abart(x.train, times, delta, mc.cores=M, seed=99)
post2 = mc.abart(x.train, times, delta, offset=offset, mc.cores=M, seed=99)

Z=8
plot(mu, post1$yhat.train.mean, asp=1,
     xlim=c(-Z, Z), ylim=c(-Z, Z))
abline(a=0, b=1)
plot(mu, post2$yhat.train.mean, asp=1,
     xlim=c(-Z, Z), ylim=c(-Z, Z))
abline(a=0, b=1)
plot(post1$yhat.train.mean, post2$yhat.train.mean, asp=1,
     xlim=c(-Z, Z), ylim=c(-Z, Z))
abline(a=0, b=1)

## End(Not run)
```

---

**ACTG175**

**AIDS Clinical Trials Group Study 175**

---

**Description**

ACTG 175 was a randomized clinical trial to compare monotherapy with zidovudine or didanosine with combination therapy with zidovudine and didanosine or zidovudine and zalcitabine in adults infected with the human immunodeficiency virus type I whose CD4 T cell counts were between 200 and 500 per cubic millimeter.
Usage
data(ACTG175)

Format
A data frame with 2139 observations on the following 27 variables:

- **pidnum**: patient's ID number
- **age**: age in years at baseline
- **wtkg**: weight in kg at baseline
- **hemo**: hemophilia (0=no, 1=yes)
- **homo**: homosexual activity (0=no, 1=yes)
- **drugs**: history of intravenous drug use (0=no, 1=yes)
- **karnof**: Karnofsky score (on a scale of 0-100)
- **oprior**: non-zidovudine antiretroviral therapy prior to initiation of study treatment (0=no, 1=yes)
- **z30**: zidovudine use in the 30 days prior to treatment initiation (0=no, 1=yes)
- **zprior**: zidovudine use prior to treatment initiation (0=no, 1=yes)
- **preanti**: number of days of previously received antiretroviral therapy
- **race**: race (0=white, 1=non-white)
- **gender**: gender (0=female, 1=male)
- **str2**: antiretroviral history (0=naive, 1=experienced)
- **strat**: antiretroviral history stratification (1='antiretroviral naive', 2='> 1 but ≤ 52 weeks of prior antiretroviral therapy', 3='> 52 weeks')
- **symptom**: symptomatic indicator (0=asymptomatic, 1=symptomatic)
- **treat**: treatment indicator (0=zidovudine only, 1=other therapies)
- **offtrt**: indicator of off-treatment before 96±5 weeks (0=no, 1=yes)
- **cd40**: CD4 T cell count at baseline
- **cd420**: CD4 T cell count at 20±5 weeks
- **cd496**: CD4 T cell count at 96±5 weeks (=NA if missing)
- **r**: missing CD4 T cell count at 96±5 weeks (0=missing, 1=observed)
- **cd80**: CD8 T cell count at baseline
- **cd820**: CD8 T cell count at 20±5 weeks
- **cens**: indicator of observing the event in days
- **days**: number of days until the first occurrence of: (i) a decline in CD4 T cell count of at least 50 (ii) an event indicating progression to AIDS, or (iii) death.
- **arms**: treatment arm (0=zidovudine, 1=zidovudine and didanosine, 2=zidovudine and zalcitabine, 3=didanosine).

Details
The variable days contains right-censored time-to-event observations. The data set includes the following post-randomization covariates: CD4 and CD8 T cell count at 20±5 weeks and the indicator of whether or not the patient was taken off-treatment before 96±5 weeks.
References


American alligator Food Choice

Description

In 1985, American alligators were harvested by hunters from August 26 to September 30 in peninsular Florida from lakes Oklawaha (Putnam County), George (Putnam and Volusia counties), Hancock (Polk County) and Trafford (Collier County). Lake, length and sex were recorded for each alligator. Stomachs from a sample of alligators 1.09-3.89m long were frozen prior to analysis. After thawing, stomach contents were removed and separated and food items were identified and tallied. Volumes were determined by water displacement. The stomach contents of 219 alligators were classified into five categories of primary food choice: Fish (the most common primary food choice), Invertebrate (snails, insects, crayfish, etc.), Reptile (turtles, alligators), Bird, and Other (amphibians, plants, household pets, stones, and other debris).

Usage
data(alligator)

Format

A data frame with 80 observations on the following 5 variables.

- **lake** a factor with levels George Hancock Oklawaha Trafford
- **sex** a factor with levels female male
- **size** alligator size, a factor with levels large (>2.3m) small (<=2.3m)
- **food** primary food choice, a factor with levels bird fish invert other reptile
- **count** cell frequency, a numeric vector

Details

The table contains a fair number of 0 counts. food is the response variable. fish is the most frequent choice, and often taken as a baseline category in multinomial response models.

Source


References

data(alligator)

## nnet::multinom Multinomial logit model fit with neural nets
fit <- multinom(food ~ lake+size+sex, data=alligator, weights=count)
summary(fit$fitted.values)
## 1=bird, 2=fish, 3=invert, 4=other, 5=reptile

(L=length(alligator$count))
(N=sum(alligator$count))
y.train=integer(N)
x.train=matrix(nrow=N, ncol=3)
x.test=matrix(nrow=L, ncol=3)
k=1
for(i in 1:L) {
  x.test[i, ]=as.integer(c(alligator$lake[i], alligator$size[i], alligator$sex[i]))
  if(alligator$count[i]>0)
    for(j in 1:alligator$count[i]) {
      y.train[k]=as.integer(alligator$food[i])
      x.train[k, ]=as.integer(c(alligator$lake[i], alligator$size[i], alligator$sex[i]))
      k=k+1
    }
}
table(y.train)
## test mbart with token run to ensure installation works
set.seed(99)
check = mbart(x.train, y.train, nskip=1, ndpost=1)
## Not run:
set.seed(99)
check = mbart(x.train, y.train, nskip=1, ndpost=1)
post=mbart(x.train, y.train, x.test)
##post=mc.mbart(x.train, y.train, x.test, mc.cores=8, seed=99)
##check=predict(post, x.test, mc.cores=8)
##print(cor(post$prob.test.mean, check$prob.test.mean)^2)

par(mfrow=c(3, 2))
K=5
for(j in 1:5) {
  h=seq(j, L*K, K)
  print(cor(fit$fitted.values[ , j], post$prob.test.mean[h])^2)
  plot(fit$fitted.values[ , j], post$prob.test.mean[h],
       xlim=0:1, ylim=0:1,
       xlab=paste0('NN: Est. Prob. j=', j),
       ylab=paste0('BART: Est. Prob. j=', j))
  abline(a=0, b=1)
}
par(mfrow=c(1, 1))

L=16
x.test=matrix(nrow=L, ncol=3)
k=1
for(size in 1:2)
  for(sex in 1:2)
    for(lake in 1:4) {
      x.test[k, ]=c(lake, size, sex)
      k=k+1
    }
x.test

## two sizes: 1=large: >2.3m, 2=small: <=2.3m
pred=predict(post, x.test)
#pred=predict(post, x.test, mc.cores=8)
ndpost=nrow(pred$prob.test)

size.test=matrix(nrow=ndpost, ncol=K*2)
for(i in 1:K) {
  j=seq(i, L*K/2, K) ## large
  size.test[, i]=apply(pred$prob.test[, j], 1, mean)
  j=j+L*K/2 ## small
  size.test[, i+K]=apply(pred$prob.test[, j], 1, mean)
}
size.test.mean=apply(size.test, 2, mean)
size.test.025=apply(size.test, 2, quantile, probs=0.025)
size.test.975=apply(size.test, 2, quantile, probs=0.975)

plot(factor(1:K, labels=c('bird', 'fish', 'invert', 'other', 'reptile')),
     rep(1, K), col=1:K, type='n', lwd=1, lty=0,
     xlim=c(1, K), ylim=c(0, 0.5), ylab='Prob.',
     sub="Multinomial BART\nFriedman's partial dependence function")
points(1:K, size.test.mean[1:K*K], col=1)
lines(1:K, size.test.025[1:K*K], col=1, lty=2)
lines(1:K, size.test.975[1:K*K], col=1, lty=2)
points(1:K, size.test.mean[1:K], col=2)
lines(1:K, size.test.025[1:K], col=2, lty=2)
lines(1:K, size.test.975[1:K], col=2, lty=2)
## legend('topright', legend=c('Small', 'Large'),
##       pch=1, col=1:2)

## End(Not run)
Description

This data set was created from the National Health and Nutrition Examination Survey (NHANES) 2009-2010 Arthritis Questionnaire.

Usage

data(arq)

Details

We have two outcomes of interest. Chronic neck pain: Yes arq010a=1 vs. No arq010a=0. Chronic lower-back/buttock pain: Yes arq010de=1 vs. No arq010de=0. seqn is a unique survey respondent identifier. wtint2yr is the survey sampling weight. riagendr is gender: 1 for males, 2 for females. ridageyr is age in years. There are several anthropometric measurements: bmxwt, weight in kg; bmxht, height in cm; bmxbmi, body mass index in kg/m^2; and bmxwaist, waist circumference in cm. The data was subsetted to ensure non-missing values of these variables.

References


bartModelMatrix

Create a matrix out of a vector or data.frame

Description

The external BART functions operate on matrices in memory. Therefore, if the user submits a vector or data.frame, then this function converts it to a matrix. Also, it determines the number of cutpoints necessary for each column when asked to do so.

Usage

bartModelMatrix(X, numcut=0L, usequants=FALSE, type=7, rm.const=FALSE, cont=FALSE, xinfo=NULL)

Arguments

X A vector or data.frame to create the matrix from.
numcut The maximum number of cutpoints to consider. If numcut=0, then just return a matrix; otherwise, return a list containing a matrix X, a vector numcut and a list xinfo.
usequants If usequants is FALSE, then the cutpoints in xinfo are generated uniformly; otherwise, if TRUE, then quantiles are used for the cutpoints.
type Determines which quantile algorithm is employed.
rm.const Whether or not to remove constant variables.
Whether or not to assume all variables are continuous.

You can provide the cutpoints to BART or let BART choose them for you. To provide them, use the xinfo argument to specify a list (matrix) where the items (rows) are the covariates and the contents of the items (columns) are the cutpoints.

See Also
class.ind

Examples

```r
set.seed(99)
a <- rbinom(10, 4, 0.4)
table(a)
x <- runif(10)
df <- data.frame(a=factor(a), x=x)
b <- bartModelMatrix(df)
b
b <- bartModelMatrix(df, numcut=9)
b
b <- bartModelMatrix(df, numcut=9, usequants=TRUE)
b
## Not run:
f <- bartModelMatrix(as.character(a))
## End(Not run)
```

Description

This interesting example is from a clinical trial conducted by the Veterans Administration Cooperative Urological Research Group. This data on recurrence of bladder cancer has been used by many to demonstrate methodology for recurrent events modelling. In this study, all patients had superficial bladder tumors when they entered the trial. These tumors were removed transurethrally.
and patients were randomly assigned to one of three treatments: placebo, thiotepa or pyridoxine (vitamin B6). Many patients had multiple recurrences of tumors during the study and new tumors were removed at each visit. For each patient, their recurrence time, if any, was measured from the beginning of treatment.

bladder is the data set that appears most commonly in the literature. It uses only the 85 subjects with nonzero follow-up who were assigned to either thiotepa or placebo and only the first four recurrences for any patient. The status variable is 1 for recurrence and 0 for everything else (including death for any reason). The data set is laid out in the competing risks format of the paper by Wei, Lin, and Weissfeld (WLW).

bladder1 is the full data set from the study. It contains all three treatment arms and all recurrences for 118 subjects; the maximum observed number of recurrences is 9.

bladder2 uses the same subset of subjects as bladder, but formatted in the (start, stop] or Anderson-Gill (AG) style. Note that in transforming from the WLW to the AG style data set there is a quite common programming mistake that leads to extra follow-up time for 12 subjects: all those with follow-up beyond their fourth recurrence. Over this extended time these subjects are by definition not at risk for another event in the WLW data set.

**Format**

bladder

| id: | Patient id |
| rx: | Treatment 1=placebo 2=thiotepa |
| number: | Initial number of tumours (8=8 or more) |
| size: | Size (cm) of largest initial tumour |
| stop: | recurrence or censoring time |
| enum: | which recurrence (up to 4) |

bladder1

| id: | Patient id |
| treatment: | Placebo, pyridoxine (vitamin B6), or thiotepa |
| number: | Initial number of tumours (8=8 or more) |
| size: | Size (cm) of largest initial tumour |
| recur: | Number of recurrences |
| start,stop: | The start and end time of each time interval |
| status: | End of interval code, 0=censored, 1=recurrence, 2=death from bladder disease, 3=death other/unknown cause |
| rtumor: | Number of tumors found at the time of a recurrence |
| rsize: | Size of largest tumor at a recurrence |
| enum: | Event number (observation number within patient) |

bladder2

| id: | Patient id |
| rx: | Treatment 1=placebo 2=thiotepa |
number: Initial number of tumours (8=8 or more)
size: size (cm) of largest initial tumour
start: start of interval (0 or previous recurrence time)
stop: recurrence or censoring time
enum: which recurrence (up to 4)

References

Examples
data(bladder)

crisk.bart       BART for competing risks

Description
Here we have implemented a simple and direct approach to utilize BART for competing risks that is very flexible, and is akin to discrete-time survival analysis. Following the capabilities of BART, we allow for maximum flexibility in modeling the dependence of competing failure times on covariates. In particular, we do not impose proportional hazards.

To elaborate, consider data in the form: \((s_i, \delta_i, x_i)\) where \(s_i\) is the event time; \(\delta_i\) is an indicator distinguishing events, \(\delta_i = h\) due to cause \(h\in\{1,2\}\), from right-censoring, \(\delta_i = 0\); \(x_i\) is a vector of covariates; and \(i = 1,\ldots,N\) indexes subjects.

We denote the \(K\) distinct event/censoring times by \(0 < t_{(1)} < \cdots < t_{(K)} < \infty\) thus taking \(t_{(j)}\) to be the \(j^{th}\) order statistic among distinct observation times and, for convenience, \(t_{(0)} = 0\). Now consider event indicators for cause \(h\): \(y_{hij}\) for each subject \(i\) at each distinct time \(t_{(j)}\) up to and including the subject’s last observation time \(s_i = t_{(n_i)}\) with \(n_i = \arg\max_j [t_{(j)} \leq s_i]\) for cause 1, but only up to \(n_i - y_{1ij}\) for cause 2.

We then denote by \(p_{hij}\) the probability of an event at time \(t_{(j)}\) conditional on no previous event. We now write the model for \(y_{hij}\) as a nonparametric probit (or logistic) regression of \(y_{hij}\) on the time \(t_{(j)}\) and the covariates \(x_{hi}\), and then utilize BART for binary responses. Specifically, \(y_{hij} = I[\delta_i = h]I[s_i = t_{(j)}], j = 1,\ldots,n_i - I[h = 2]y_{1ij}.\) Therefore, we have \(p_{hij} = F(m_{hij}), m_{hij} = m_{wh} + f_h(t_{(j)}, x_{hi})\) where \(F\) denotes the Normal (or Logistic) cdf. As in the binary response case, \(f_h\) is the sum of many tree models. Finally, based on these probabilities, \(p_{hij}\), we can construct targets of inference such as the cumulative incidence functions.
crisk.bart

Usage

crisk.bart(x.train=matrix(0,0,0), y.train=NULL,
    x.train2=x.train, y.train2=NULL,
    times=NULL, delta=NULL, K=NULL,
    x.test=matrix(0,0,0), x.test2=x.test, cond=NULL,
    sparse=FALSE, theta=0, omega=1,
    a=0.5, b=1, augment=FALSE,
    rho=NULL, rho2=NULL,
    xinfo=matrix(0,0,0), xinfo2=matrix(0,0),
    usequants=FALSE,
    rm.const=TRUE, type='pbart',
    ntype=as.integer(
        factor(type, levels=c('wbart', 'pbart', 'lbart'))),
    k=2, power=2, base=0.95,
    offset=NULL, offset2=NULL,
    tau.num=c(NA, 3, 6)[ntype],
    ntree=50, numcut=100, ndpost=1000, nskip=250,
    keep every = 10L,

    printevery=100L,

    id=NULL, ## crisk.bart only
    seed=99, ## mc.crisk.bart only
    mc.cores=2, ## mc.crisk.bart only
    nice=19L ## mc.crisk.bart only
    )

mc.crisk.bart(x.train=matrix(0,0,0), y.train=NULL,
    x.train2=x.train, y.train2=NULL,
    times=NULL, delta=NULL, K=NULL,
    x.test=matrix(0,0,0), x.test2=x.test, cond=NULL,
    sparse=FALSE, theta=0, omega=1,
    a=0.5, b=1, augment=FALSE,
    rho=NULL, rho2=NULL,
    xinfo=matrix(0,0,0), xinfo2=matrix(0,0,0),
    usequants=FALSE,
    rm.const=TRUE, type='pbart',
    ntype=as.integer(
        factor(type, levels=c('wbart', 'pbart', 'lbart'))),
    k=2, power=2, base=0.95,
    offset=NULL, offset2=NULL,
    tau.num=c(NA, 3, 6)[ntype],
    ntree=50, numcut=100, ndpost=1000, nskip=250,
keepevery = 10L,

printevery = 100L,

id=NULL,    ## crisk.bart only
seed=99,    ## mc.crisk.bart only
mc.cores=2, ## mc.crisk.bart only
nice=19L    ## mc.crisk.bart only
)

Arguments

**x.train**
Covariates for training (in sample) data of cause 1.
Must be a data.frame or a matrix with rows corresponding to observations and
columns to variables.
crisk.bart will generate draws of $f_1(t, x)$ for each $x$ which is a row of $x.train$
(note that the definition of $x.train$ is dependent on whether $y.train$ has been
specified; see below).

**y.train**
Cause 1 binary response for training (in sample) data.
If $y.train$ is NULL, then $y.train$ ($x.train$ and $x.test$, if specified) are gen-
erated by a call to crisk.pre.bart (which require that times and delta be
provided; see below); otherwise, $y.train$ ($x.train$ and $x.test$, if specified)
are utilized as given assuming that the data construction has already been per-
formed.

**x.train2**
Covariates for training (in sample) data of cause 2. Similar to $x.train$ above.

**y.train2**
Cause 2 binary response for training (in sample) data, i.e., failure from any cause
besides the cause of interest which is cause 1. Similar to $y.train$ above.

**times**
The time of event or right-censoring, $s_i$.
If $y.train$ is NULL, then times (and delta) must be provided.

**delta**
The event indicator: 1 for cause 1, 2 for cause 2 and 0 is censored.
If $y.train$ is NULL, then delta (and times) must be provided.

**K**
If provided, then coarsen times per the quantiles $1/K, 2/K, ..., K/K$.

**x.test**
Covariates for test (out of sample) data of cause 1.
Must be a data.frame or a matrix and have the same structure as $x.train$.
crisk.bart will generate draws of $f_1(t, x)$ for each $x$ which is a row of $x.test$.

**x.test2**
Covariates for test (out of sample) data of cause 2. Similar to $x.test$ above.

**cond**
A vector of indices for $y.train2$ indicating subjects who did not suffer a cause
1 event and, therefore, are eligible for cause 2.

**sparse**
Whether to perform variable selection based on a sparse Dirichlet prior; see
Linero 2016.

**theta**
Set theta parameter; zero means random.

**omega**
Set omega parameter; zero means random.
Sparse parameter for Beta\((a, \ b)\) prior: \(0.5 <= a <= 1\) where lower values inducing more sparsity.

Sparse parameter for Beta\((a, \ b)\) prior; typically, \(b=1\).

Sparse parameter: typically \(\rho = p\) where \(p\) is the number of covariates in \(x.train\).

Sparse parameter: typically \(\rho_2 = p\) where \(p\) is the number of covariates in \(x.train2\).

Whether data augmentation is to be performed in sparse variable selection.

You can provide the cutpoints to BART or let BART choose them for you. To provide them, use the \(xinfo\) argument to specify a list (matrix) where the items (rows) are the covariates and the contents of the items (columns) are the cutpoints.

Cause 2 cutpoints.

If \(\)usequants\(=\)FALSE, then the cutpoints in \(xinfo\) are generated uniformly; otherwise, if TRUE, uniform quantiles are used for the cutpoints.

Whether or not to remove constant variables.

Whether to employ probit BART via Albert-Chib, \(\)pbart\(\), or logistic BART by Holmes-Held, \(\)lbart\(\).

The integer equivalent of \(\)type\(\) where \(\)wbart\(\) is 1, \(\)pbart\(\) is 2 and \(\)lbart\(\) is 3.

\(k\) is the number of prior standard deviations \(f_h(t, x)\) is away from +/-3. The bigger \(k\) is, the more conservative the fitting will be.

Power parameter for tree prior.

Base parameter for tree prior.

Cause 1 binary offset.

Cause 2 binary offset.

The numerator in the \(\tau\) definition.

The number of trees in the sum.

The number of possible values of cutpoints (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 cutpoints used for the \(i^{th}\) variable in \(x.train\). If \(\)usequants\(=\)FALSE, numcut equally spaced cutoffs are used covering the range of values in the corresponding column of \(x.train\). If \(\)usequants\(=\)TRUE, then \(\)min\((\)numcut, the number of unique values in the corresponding columns of \(x.train\) - 1) cutpoint values are used.

The number of posterior draws returned.

Number of MCMC iterations to be treated as burn in.

Every keepeverything draw is kept to be returned to the user.

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

\(\)crisk.bart\ only: unique identifier added to returned list.

\(\)mc.crisk.bart\ only: seed required for reproducible MCMC.

\(\)mc.cores\ \(\)mc.crisk.bart\ only: number of cores to employ in parallel.

\(\)mc.crisk.bart\ only: set the job niceness. The default niceness is 19: niceness goes from 0 (highest priority) to 19 (lowest priority).
crisk.bart returns an object of type criskbart which is essentially a list. Besides the items listed below, the list has offset, offset2, times which are the unique times, $K$ which is the number of unique times, tx.train and tx.test, if any.

- **yhat.train** A matrix with ndpost rows and nrow(x.train) columns. Each row corresponds to a draw $f_{i}^{*}$ from the posterior of $f_1$ and each column corresponds to a row of x.train. The $(i,j)$ value is $f_{i}^{*}(t,x)$ for the $i^{th}$ kept draw of $f_1$ 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.

- **surv.test** test data fits for the survival function, $S(t,x)$.

- **surv.test.mean** mean of surv.test over the posterior samples.

- **prob.test** The probability of suffering cause 1.

- **prob.test2** The probability of suffering cause 2.

- **cif.test** The cumulative incidence function of cause 1, $F_1(t,x)$.

- **cif.test2** The cumulative incidence function of cause 2, $F_2(t,x)$.

- **cif.test.mean** mean of cif.test columns for cause 1.

- **cif.test2.mean** mean of cif.test2 columns for cause 2.

- **varcount** a matrix with ndpost 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 this variable is used for cause 1 in a tree decision rule (over all trees) is given.

- **varcount2** For each variable the total count of the number of times this variable is used for cause 2 in a tree decision rule is given.

**Author(s)**

Rodney Sparapani: <rsparapa@mcw.edu>

**References**


Examples

data(transplant)

pfit <- survfit(Surv(futime, event) ~ abo, transplant)

# competing risks for type O
plot(pfit[4,], xscale=7, xmax=735, col=1:3, lwd=2, ylim=c(0, 1),
     xlab='t (weeks)', ylab='Aalen-Johansen (AJ) CI(t)'
legend(450, .4, c("Death", "Transplant", "Withdrawal"), col=1:3, lwd=2)
## plot(pfit[4,], xscale=30.5, xmax=735, col=1:3, lwd=2, ylim=c(0, 1),
##     xlab='t (months)', ylab='Aalen-Johansen (AJ) CI(t)'
## legend(450, .4, c("Death", "Transplant", "Withdrawal"), col=1:3, lwd=2)

delta <- (as.numeric(transplant$event)-1)
## recode so that delta=1 is cause of interest; delta=2 otherwise
delta[delta==1] <- 4
delta[delta==2] <- 1
delta[delta>1] <- 2
table(delta, transplant$event)

times <- pmax(1, ceiling(transplant$futime/7)) ## weeks
##times <- pmax(1, ceiling(transplant$futime/30.5)) ## months
table(times)

typeO <- 1*(transplant$abo=='O')
typeA <- 1*(transplant$abo=='A')
typeB <- 1*(transplant$abo=='B')
typeAB <- 1*(transplant$abo=='AB')
table(typeA, typeO)

x.train <- cbind(typeO, typeA, typeB, typeAB)

x.test <- cbind(1, 0, 0, 0)
dimnames(x.test)[[2]] <- dimnames(x.train)[[2]]

##test BART with token run to ensure installation works
set.seed(99)
post <- crisk.bart(x.train=x.train, times=times, delta=delta,
    x.test=x.test, nskip=1, ndpost=1, keepevery=1)

## Not run:

## run one long MCMC chain in one process
## set.seed(99)
## post <- crisk.bart(x.train=x.train, times=times, delta=delta, x.test=x.test)

## in the interest of time, consider speeding it up by parallel processing
crisk.pre.bart

Data construction for competing risks with BART

Description

Competing risks contained in \((t, \delta, x)\) must be translated to data suitable for the BART competing risks model; see crisk.bart for more details.

Usage

```r
crisk.pre.bart( times, delta, x.train=NULL, x.test=NULL, x.train2=x.train, x.test2=x.test, K=NULL )
```

Arguments

- **times**
  - The time of event or right-censoring.

- **delta**
  - The event indicator: 1 is a cause 1 event, 2 a cause 2 while 0 is censored.
Explanatory variables for training (in sample) data of cause 1. If provided, must be a matrix with (as usual) rows corresponding to observations and columns to variables.

Explanatory variables for test (out of sample) data of cause 1. If provided, must be a matrix and have the same structure as x.train.

Explanatory variables for training (in sample) data of cause 2. If provided, must be a matrix with (as usual) rows corresponding to observations and columns to variables.

Explanatory variables for test (out of sample) data of cause 2. If provided, must be a matrix and have the same structure as x.train.

If provided, then coarsen times per the quantiles $1/K, 2/K, ..., K/K$.

surv.pre.bart returns a list. Besides the items listed below, the list has a times component giving the unique times and $K$ which is the number of unique times.

A vector of binary responses for cause 1.

A vector of binary responses for cause 2.

A vector of indices of y.train indicating censored subjects.

The binary offset for y.train.

The binary offset for y.train2.

A matrix with rows consisting of time and the covariates of the training data for cause 1.

A matrix with rows consisting of time and the covariates of the training data for cause 2.

A matrix with rows consisting of time and the covariates of the test data, if any, for cause 1.

A matrix with rows consisting of time and the covariates of the test data, if any, for cause 2.

Rodney Sparapani: <rsparapa@mcw.edu>


See Also

crisk.bart
Examples

```r
data(transplant)

delta <- (as.numeric(transplant$event)-1)
delta[delta==1] <- 4
delta[delta==2] <- 1
delta[delta>1] <- 2

table(delta, transplant$event)

table(1+floor(transplant$futime/30.5)) ## months
times <- 1+floor(transplant$futime/30.5)

typeO <- 1*(transplant$abo=='O')
typeA <- 1*(transplant$abo=='A')
typeB <- 1*(transplant$abo=='B')
typeAB <- 1*(transplant$abo=='AB')

table(typeA, typeO)

x.train <- cbind(typeO, typeA, typeB, typeAB)

N <- nrow(x.train)
x.test <- x.train

x.test[1:N, 1:4] <- matrix(c(1, 0, 0, 0), nrow=N, ncol=4, byrow=TRUE)

pre <- crisk.pre.bart(x.train=x.train, times=times, delta=delta, x.test=x.test)
```

---

**crisk2.bart**

*BART for competing risks*

---

**Description**

Here we have implemented another approach to utilize BART for competing risks that is very flexible, and is akin to discrete-time survival analysis. Following the capabilities of BART, we allow for maximum flexibility in modeling the dependence of competing failure times on covariates. In particular, we do not impose proportional hazards.

Similar to `crisk.bart`, we utilize two BART models, yet they are two different BART models than previously considered. First, given an event of either cause occurred, we employ a typical binary BART model to discriminate between cause 1 and 2. Next, we proceed as if it were a typical survival analysis with BART for an absorbing event from either cause.

To elaborate, consider data in the form: \((s_i, \delta_i, x_i)\) where \(s_i\) is the event time; \(\delta_i\) is an indicator distinguishing events, \(\delta_i = h\) due to cause \(h\in\{1, 2\}\), from right-censoring, \(\delta_i = 0\); \(x_i\) is a vector of covariates; and \(i = 1, ..., N\) indexes subjects. We denote the \(K\) distinct event/censoring times by
$0 < t(1) < \ldots < t(K) < \infty$ thus taking $t(j)$ to be the $j^{th}$ order statistic among distinct observation times and, for convenience, $t(0) = 0$.

First, consider event indicators for an event from either cause: $y_{1ij}$ for each subject $i$ at each distinct time $t(j)$ up to and including the subject’s last observation time $s_i = t(n_i)$ with $n_i = \arg \max_j [t(j) \leq s_i]$. We denote by $p_{1ij}$ the probability of an event at time $t(j)$ conditional on no previous event. We now write the model for $y_{1ij}$ as a nonparametric probit (or logistic) regression of $y_{1ij}$ on the time $t(j)$ and the covariates $x_{1i}$, and then utilize BART for binary responses. Specifically, $y_{1ij} = I[\delta_i > 0]I[s_i = t(j)], \ j = 1, \ldots, n_i$. Therefore, we have $p_{1ij} = F(\mu_{1ij}), \ mu_{1ij} = \mu_{1} + f_1(t(j), x_{1i})$ where $F$ denotes the Normal (or Logistic) cdf.

Next, we denote by $p_{2i}$ the probability of a cause 1 event at time $s_i$ conditional on an event having occurred. We now write the model for $y_{2i}$ as a nonparametric probit (or logistic) regression of $y_{2i}$ on the time $s_i$ and the covariates $x_{2i}$, via BART for binary responses. Specifically, $y_{2i} = I[\delta_i = 1]$. Therefore, we have $p_{2i} = F(\mu_{2i}), \ mu_{2i} = \mu_{2} + f_2(s_i, x_{2i})$ where $F$ denotes the Normal (or Logistic) cdf. Although, we modeled $p_{2i}$ at the time of an event, $s_i$, we can estimate this probability at any other time points on the grid via $p(t(j), x_2) = F(\mu_{2} + f_2(t(j), x_2))$. Finally, based on these probabilities, $p_{hij}$, we can construct targets of inference such as the cumulative incidence functions.

Usage

```r
> crisk2.bart(x.train=matrix(0,0,0), y.train=NULL, 
  x.train2=x.train, y.train2=NULL, 
  times=NULL, delta=NULL, K=NULL, 
  x.test=matrix(0,0,0), x.test2=x.test, 
  sparse=FALSE, theta=0, omega=1, 
  a=0.5, b=1, augment=FALSE, 
  rho=NULL, rho2=NULL, 
  xinfo=matrix(0,0,0), xinfo2=matrix(0,0,0), 
  usequants=FALSE, 
  rm.const=TRUE, type='pbart', 
  ntype=as.integer(
    factor(type, levels=c('wbart', 'pbart', 'lbart'))), 
  k=2, power=2, base=0.95, 
  offset=NULL, offset2=NULL, 
  tau.num=c(NA, 3, 6)[ntype], 
  ntree=50, numcut=100, ndpost=1000, nskip=250, 
  keepevery = 10L, 

  printevery=100L, 

  id=NULL,    ## crisk2.bart only 
  seed=99,    ## mc.crisk2.bart only 
  mc.cores=2, ## mc.crisk2.bart only 
  nice=19L    ## mc.crisk2.bart only
```

mc.crisk2.bart(x.train=matrix(0,0,0), y.train=NULL, x.train2=x.train, y.train2=NULL, times=NULL, delta=NULL, K=NULL, x.test=matrix(0,0,0), x.test2=x.test, sparse=FALSE, theta=0, omega=1, a=0.5, b=1, augment=FALSE, rho=NULL, rho2=NULL, xinfo=matrix(0,0,0), xinfo2=matrix(0,0,0), usequants=FALSE, rm.const=TRUE, type="Var pbart", ntype=as.integer("factor(type, levels=c('wbart', 'pbart', 'lbart'))"), k=2, power=2, base=0.95, offset=NULL, offset2=NULL, tau.num=c(NA, 3, 6)[ntype], ntree=50, numcut=100, ndpost=1000, nskip=250, keepevery = 10L, printevery=100L, id=NULL, ## crisk2.bart only seed=99, ## mc.crisk2.bart only mc.cores=2, ## mc.crisk2.bart only nice=19L ## mc.crisk2.bart only )

Arguments

x.train Covariates for training (in sample) data for an event. Must be a data.frame or a matrix with rows corresponding to observations and columns to variables. crisk2.bart will generate draws of \( f_1(t, x) \) for each \( x \) which is a row of x.train (note that the definition of x.train is dependent on whether y.train has been specified; see below).

y.train Event binary response for training (in sample) data. If y.train is NULL, then y.train (x.train and x.test, if specified) are generated by a call to surv.pre.bart (which require that times and delta be provided: see below); otherwise, y.train (x.train and x.test, if specified) are utilized as given assuming that the data construction has already been performed.

x.train2 Covariates for training (in sample) data of for a cause 1 event. Similar to x.train above.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>y.train2</strong></td>
<td>Cause 1 event binary response for training (in sample) data. Similar to <code>y.train</code> above.</td>
</tr>
<tr>
<td><strong>times</strong></td>
<td>The time of event or right-censoring, ( s_i ). If <code>y.train</code> is NULL, then <code>times</code> (and <code>delta</code>) must be provided.</td>
</tr>
<tr>
<td><strong>delta</strong></td>
<td>The event indicator: 1 for cause 1, 2 for cause 2 and 0 is censored. If <code>y.train</code> is NULL, then <code>delta</code> (and <code>times</code>) must be provided.</td>
</tr>
<tr>
<td><strong>K</strong></td>
<td>If provided, then coarsen <code>times</code> per the quantiles ( 1/K, 2/K, ..., K/K ).</td>
</tr>
<tr>
<td><strong>x.test</strong></td>
<td>Covariates for test (out of sample) data of an event. Must be a data.frame or a matrix and have the same structure as <code>x.train</code>. <code>crisk2.bart</code> will generate draws of ( f_1(t, x) ) for each ( x ) which is a row of <code>x.test</code>.</td>
</tr>
<tr>
<td><strong>x.test2</strong></td>
<td>Covariates for test (out of sample) data of a cause 1 event. Similar to <code>x.test</code> above.</td>
</tr>
<tr>
<td><strong>sparse</strong></td>
<td>Whether to perform variable selection based on a sparse Dirichlet prior; see Linero 2016.</td>
</tr>
<tr>
<td><strong>theta</strong></td>
<td>Set <code>theta</code> parameter; zero means random.</td>
</tr>
<tr>
<td><strong>omega</strong></td>
<td>Set <code>omega</code> parameter; zero means random.</td>
</tr>
<tr>
<td><strong>a</strong></td>
<td>Sparse parameter for ( Beta(a, b) ) prior: ( 0.5 \leq a \leq 1 ) where lower values inducing more sparsity.</td>
</tr>
<tr>
<td><strong>b</strong></td>
<td>Sparse parameter for ( Beta(a, b) ) prior; typically, ( b=1 ).</td>
</tr>
<tr>
<td><strong>rho</strong></td>
<td>Sparse parameter: typically ( \rho=p ) where ( p ) is the number of covariates in <code>x.train</code>.</td>
</tr>
<tr>
<td><strong>rho2</strong></td>
<td>Sparse parameter: typically ( \rho2=p ) where ( p ) is the number of covariates in <code>x.train2</code>.</td>
</tr>
<tr>
<td><strong>augment</strong></td>
<td>Whether data augmentation is to be performed in sparse variable selection.</td>
</tr>
<tr>
<td><strong>xinfo</strong></td>
<td>You can provide the cutpoints to BART or let BART choose them for you. To provide them, use the <code>xinfo</code> argument to specify a list (matrix) where the items (rows) are the covariates and the contents of the items (columns) are the cutpoints.</td>
</tr>
<tr>
<td><strong>xinfo2</strong></td>
<td>Cause 2 cutpoints.</td>
</tr>
<tr>
<td><strong>usequants</strong></td>
<td>If <code>usequants=FALSE</code>, then the cutpoints in <code>xinfo</code> are generated uniformly; otherwise, if <code>TRUE</code>, uniform quantiles are used for the cutpoints.</td>
</tr>
<tr>
<td><strong>rm.const</strong></td>
<td>Whether or not to remove constant variables.</td>
</tr>
<tr>
<td><strong>type</strong></td>
<td>Whether to employ probit BART via Albert-Chib, 'pbart', or logistic BART by Holmes-Held, 'lbart'.</td>
</tr>
<tr>
<td><strong>ntype</strong></td>
<td>The integer equivalent of <code>type</code> where 'wbart' is 1, 'pbart' is 2 and 'lbart' is 3.</td>
</tr>
<tr>
<td><strong>k</strong></td>
<td>( k ) is the number of prior standard deviations ( f_h(t, x) ) is away from +/-3. The bigger ( k ) is, the more conservative the fitting will be.</td>
</tr>
<tr>
<td><strong>power</strong></td>
<td>Power parameter for tree prior.</td>
</tr>
<tr>
<td><strong>base</strong></td>
<td>Base parameter for tree prior.</td>
</tr>
<tr>
<td><strong>offset</strong></td>
<td>Cause 1 binary offset.</td>
</tr>
</tbody>
</table>
offset2  Cause 2 binary offset.
tau.num  The numerator in the tau definition.
ntree  The number of trees in the sum.
umcut  The number of possible values of cutpoints (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 cutpoints 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 $\min(numcut, the number of unique values in the corresponding columns of x.train - 1)$ cutpoint values are used.
ndpost  The number of posterior draws returned.
nskip  Number of MCMC iterations to be treated as burn in.
keepevery  Every keepevery draw is kept to be returned to the user.
printevery  As the MCMC runs, a message is printed every printevery draws.
id  crisk2.bart only: unique identifier added to returned list.
seed  mc.crisk2.bart only: seed required for reproducible MCMC.
mc.cores  mc.crisk2.bart only: number of cores to employ in parallel.
nice  mc.crisk2.bart only: set the job niceness. The default niceness is 19: niceness goes from 0 (highest priority) to 19 (lowest priority).

Value

crisk2.bart returns an object of type crisk2bart which is essentially a list. Besides the items listed below, the list has offset, offset2, times which are the unique times, K which is the number of unique times, tx.train and tx.test, if any.

yhat.train  A matrix with ndpost rows and nrow(x.train) columns. Each row corresponds to a draw $f_1^*$ from the posterior of $f_1$ and each column corresponds to a row of x.train. The $(i,j)$ value is $f_1^*(t,x)$ for the $i^{th}$ kept draw of $f_1$ 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.
surv.test  test data fits for the survival function, $S(t,x)$.
surv.test.mean  mean of surv.test over the posterior samples.
prob.test  The probability of suffering an event.
prob.test2  The probability of suffering a cause 1 event.
cif.test  The cumulative incidence function of cause 1, $F_1(t,x)$.
cif.test2  The cumulative incidence function of cause 2, $F_2(t,x)$.
cif.test.mean  mean of cif.test columns for cause 1.
cif.test2.mean  mean of cif.test2 columns for cause 2.
varcount  a matrix with ndpost 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 this variable is used for an event in a tree decision rule (over all trees) is given.

varcount2  For each variable the total count of the number of times this variable is used for a cause 1 event in a tree decision rule is given.

Author(s)
Rodney Sparapani: <rsparapa@mcw.edu>

References


See Also
surv.pre.bart, predict.crisk2.bart, mc.crisk2.pwbart, crisk.bart

Examples

data(transplant)

pfit <- survfit(Surv(futime, event) ~ abo, transplant)

# competing risks for type O
plot(pfit[,4], xscale=7, xmax=735, col=1:3, lwd=2, ylim=c(0, 1),
     xlab='t (weeks)', ylab='Aalen-Johansen (AJ) CI(t)')
legend(450, .4, c("Death", "Transplant", "Withdrawal"), col=1:3, lwd=2)
## plot(pfit[,4], xscale=30.5, xmax=735, col=1:3, lwd=2, ylim=c(0, 1),
##      xlab='t (months)', ylab='Aalen-Johansen (AJ CI(t))'
##      legend(450, .4, c("Death", "Transplant", "Withdrawal"), col=1:3, lwd=2)

delta <- (as.numeric(transplant$event)-1)
## recode so that delta=1 is cause of interest; delta=2 otherwise
delta[delta==1] <- 4
delta[delta==2] <- 1
delta[delta>1] <- 2
table(delta, transplant$event)

times <- pmax(1, ceiling(transplant$futime/7)) # weeks
# times <- pmax(1, ceiling(transplant$futime/30.5)) # months
table(times)

typeO <- 1*(transplant$abo=='O')
typeA <- 1*(transplant$abo=='A')
typeB <- 1*(transplant$abo=='B')
typeAB <- 1*(transplant$abo=='AB')
table(typeA, typeO)

x.train <- cbind(typeO, typeA, typeB, typeAB)

x.test <- cbind(1, 0, 0, 0)
dimnames(x.test)[[2]] <- dimnames(x.train)[[2]]

## Test BART with token run to ensure installation works
set.seed(99)
post <- crisk2.bart(x.train=x.train, times=times, delta=delta,
                     x.test=x.test, nskip=1, ndpost=1, keepevery=1)

## Not run:
## run one long MCMC chain in one process
## set.seed(99)
## post <- crisk2.bart(x.train=x.train, times=times, delta=delta, x.test=x.test)

## in the interest of time, consider speeding it up by parallel processing
## run "mc.cores" number of shorter MCMC chains in parallel processes
post <- mc.crisk2.bart(x.train=x.train, times=times, delta=delta,
                        x.test=x.test, seed=99, mc.cores=8)

K <- post$K

typeO.cif.mean <- apply(post$cif.test, 2, mean)
typeO.cif.025 <- apply(post$cif.test, 2, quantile, probs=0.025)
typeO.cif.975 <- apply(post$cif.test, 2, quantile, probs=0.975)

plot(pfit[4,], xscale=7, xmax=735, col=1:3, lwd=2, ylim=c(0, 0.8),
     xlab="t (weeks)", ylab="CI(t)"
points(c(0, post$times)*7, c(0, typeO.cif.mean), col=4, type='s', lwd=2)
points(c(0, post$times)*7, c(0, typeO.cif.025), col=4, type='s', lwd=2, lty=2)
points(c(0, post$times)*7, c(0, typeO.cif.975), col=4, type='s', lwd=2, lty=2)
legend(450, .4, c("Transplant(BART)", "Transplant(AJ)",
                   "Death(AJ)", "Withdrawal(AJ)"),
col=c(4, 2, 1, 3), lwd=2)

# dev.copy2pdf(file='..//vignettes/figures/liver-BART.pdf')
# plot(pfit[4,], xscale=30.5, xmax=735, col=1:3, lwd=2, ylim=c(0, 0.8),
#      xlab="t (months)", ylab="CI(t)"
# points(c(0, post$times)*30.5, c(0, typeO.cif.mean), col=4, type='s', lwd=2)
# points(c(0, post$times)*30.5, c(0, typeO.cif.025), col=4, type='s', lwd=2, lty=2)
# points(c(0, post$times)*30.5, c(0, typeO.cif.975), col=4, type='s', lwd=2, lty=2)
Description

Truncated Normal latents with non-unit variance are necessary for logistic BART.

Usage

draw_lambda_i(lambda, mean, kmax=1000, thin=1)

Arguments

lambda: Previous value of lambda.
mean: Mean of truncated Normal.
kmax: The number of terms in the mixture.
thin: The thinning parameter.

Value

Returns the variance for a truncated Normal, i.e., $N(\text{mean}, \lambda)I(\tau, \infty)$.

Author(s)

Robert McCulloch: <robert.e.mcculloch@gmail.com>,
Rodney Sparapani: <rsparapa@mcw.edu>,
Robert Gramacy: <rbg@vt.edu>.

See Also

rtnorm, lbart

Examples

set.seed(12)
draw_lambda_i(1, 2)
rtnorm(1, 2, sqrt(6.773462), 6)
draw_lambda_i(6.773462, 2)
**Description**

BART is a Bayesian “sum-of-trees” model.

For a numeric response $y$, we have $y = f(x) + \epsilon$, where $\epsilon \sim N(0, \sigma^2)$.

$f$ is the sum of many tree models. The goal is to have very flexible inference for the unknown 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**

```r
gbart(x.train, y.train, 
x.test=matrix(0,0,0), type='wbart',
ntype=as.integer(
    factor(type, levels=c('wbart', 'pbart', 'lbart'))),
sparse=FALSE, theta=0, omega=1,
a=0.5, b=1, augment=FALSE, rho=NULL,
xinfo=matrix(0,0,0), usequants=FALSE,
rm.const=TRUE,
sigest=NA, sigdf=3, sigquant=0.90,
k=2, power=2, base=0.95,
lambda=NA, tau.num=c(NA, 3, 6)[ntype],
offset=NULL, w=rep(1, length(y.train)),
ntree=c(200L, 50L, 50L)[ntype], numcut=100L,
ndpost=1000L, nskip=100L,
keepevery=c(1L, 10L, 10L)[ntype],
printevery=100L, transposed=FALSE,
hostname=FALSE,
mc.cores = 1L, ## mc.gbart only
nice = 19L,    ## mc.gbart only
seed = 99L      ## mc.gbart only
)
```

```r
mc.gbart(x.train, y.train, 
x.test=matrix(0,0,0), type='wbart',
ntype=as.integer(
    factor(type, levels=c('wbart', 'pbart', 'lbart'))),
sparse=FALSE, theta=0, omega=1,
lambda=NA, tau.num=c(NA, 3, 6)[ntype],
offset=NULL, w=rep(1, length(y.train)),
ntree=c(200L, 50L, 50L)[ntype], numcut=100L,
ndpost=1000L, nskip=100L,
keepevery=c(1L, 10L, 10L)[ntype],
printevery=100L, transposed=FALSE,
hostname=FALSE,
mc.cores = 1L, ## mc.gbart only
nice = 19L,    ## mc.gbart only
seed = 99L      ## mc.gbart only
```
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 created if \(q = 2\) where \(q\) is the number of levels of the factor. \texttt{gbart} will generate draws of \(f(x)\) for each \(x\) which is a row of \texttt{x.train}.

**y.train**  
Continuous or binary dependent variable for training (in sample) data.  
If \(y\) is numeric, then a continuous BART model is fit (Normal errors).  
If \(y\) is binary (has only 0’s and 1’s), then a binary BART model with a probit link is fit by default: you can over-ride the default via the argument \texttt{type} to specify a logit BART model.

**x.test**  
Explanatory variables for test (out of sample) data. Should have same structure as \texttt{x.train}. \texttt{gbart} will generate draws of \(f(x)\) for each \(x\) which is a row of \texttt{x.test}.

**type**  
You can use this argument to specify the type of fit. ‘\texttt{wbart}’ for continuous BART, ‘\texttt{pbart}’ for probit BART or ‘\texttt{lbart}’ for logit BART.

**ntype**  
The integer equivalent of \texttt{type} where ‘\texttt{wbart}’ is 1, ‘\texttt{pbart}’ is 2 and ‘\texttt{lbart}’ is 3.

**sparse**  
Whether to perform variable selection based on a sparse Dirichlet prior rather than simply uniform; see Linero 2016.

**theta**  
Set \(\theta\) parameter; zero means random.

**omega**  
Set \(\omega\) parameter; zero means random.

**a**  
Sparse parameter for \(Beta(a, b)\) prior: 0.5 \(\leq a \leq 1\) where lower values inducing more sparsity.

**b**  
Sparse parameter for \(Beta(a, b)\) prior; typically, \(b = 1\).
rho Sparse parameter: typically $\rho = p$ where $p$ is the number of covariates under consideration.

augment Whether data augmentation is to be performed in sparse variable selection.

xinfo You can provide the cutpoints to BART or let BART choose them for you. To provide them, use the xinfo argument to specify a list (matrix) where the items (rows) are the covariates and the contents of the items (columns) are the cutpoints.

usequants If usequants=FALSE, then the cutpoints in xinfo are generated uniformly; otherwise, if TRUE, uniform quantiles are used for the cutpoints.

rm.const Whether or not to remove constant variables.

sigest The prior for the error variance ($\sigma^2$) 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 aggressive the fit will be as you are putting more prior weight on error standard deviations ($\sigma$) 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 +/-0.5. The response, codey.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. The bigger $k$ is, the more conservative the fitting will be.

power Power parameter for tree prior.

base Base parameter for tree prior.

lambda The scale of the prior for the variance. If lambda is zero, then the variance is to be considered fixed and known at the given value of sigest. Not used if $y$ is binary.

tau.num The numerator in the tau definition, i.e., $\tau = \tau.num/(k*\sqrt{ntree})$.

offset Continuous BART operates on $y.train$ centered by offset which defaults to mean($y.train$). With binary BART, the centering is $P(Y = 1|x) = F(f(x) + offset)$ where offset defaults to $F^{-1}(\text{mean}(y.train))$. You can use the offset parameter to over-ride these defaults.

w Vector of weights which multiply the standard deviation. Not used if $y$ is binary.

ntree The number of trees in the sum.

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 $\min(\text{numcut}, \text{the number of unique values in the corresponding column of x.train})$ values are used.
The number of posterior draws returned.

**nskip**

Number of MCMC iterations to be treated as burn in.

**printevery**

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

**keepevery**

Every keepevery draw is kept to be returned to the user.

**transposed**

When running gbart in parallel, it is more memory-efficient to transpose x.train and x.test, if any, prior to calling mc.gbart.

**hostname**

When running on a cluster occasionally it is useful to track on which node each chain is running; to do so set this argument to TRUE.

**seed**

Setting the seed required for reproducible MCMC.

**mc.cores**

Number of cores to employ in parallel.

**nice**

Set the job niceness. The default niceness is 19: niceness goes from 0 (highest) to 19 (lowest).

**Details**

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

Thus, unlike a lot of other 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 \(\sigma^*\) 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.

For \(x\).train/x.test with missing data elements, gbart will singly impute them with hot decking. For one or more missing covariates, record-level hot-decking imputation deWaPann11 is employed that is biased towards the null, i.e., nonmissing values from another record are randomly selected regardless of the outcome. Since mc.gbart runs multiple gbart threads in parallel, mc.gbart performs multiple imputation with hot decking, i.e., a separate imputation for each thread. This record-level hot-decking imputation is biased towards the null, i.e., nonmissing values from another record are randomly selected regardless of \(y\).train.

**Value**

gbart returns an object of type gbart which is essentially a list. In the numeric \(y\) case, the list has components:

- **yhat.train**
  A matrix with ndpost 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.

- **yhat.train.mean**
  Train data fits = mean of yhat.train columns.

- **yhat.test.mean**
  Test data fits = mean of yhat.test columns.

- **sigma**
  Post burn in draws of sigma, length = ndpost.
first.sigma  burn-in draws of sigma.
varcount  a matrix with ndpost 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 ($\sigma$) used in the prior.

Author(s)

Robert McCulloch: <robert.e.mcculloch@gmail.com>,
Rodney Sparapani: <rsparapa@mcw.edu>.

References


See Also

pbart

Examples

```r
# simulate data (example from Friedman 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)
ImFit = lm(y~.,data.frame(x,y))  #compare lm fit to BART later

# test BART with token run to ensure installation works
set.seed(99)
```
gbmm

BART is a Bayesian “sum-of-trees” model. For a numeric response $y$, we have $y = f(x) + \epsilon$, where $\epsilon \sim N(0, \sigma^2)$. $f$ is the sum of many tree models. The goal is to have very flexible inference for the unknown 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.
hostname=FALSE,
mc.cores = 1L,  ## mc.gbmm only
nice = 19L,      ## mc.gbmm only
seed = 99L,     ## mc.gbmm only
)
mc.gbmm(
x.train, y.train,
x.test=matrix(0,0,0), type='wbart',
u.train=NULL, B=NULL,
ntype=as.integer(
    factor(type, levels=c('wbart', 'pbart'))),
sparse=FALSE, theta=0, omega=1,
a=0.5, b=1, augment=FALSE, rho=NULL,
xinfo=matrix(0,0,0), usequants=FALSE,
rm.const=TRUE,
sigest=NA, sigdf=3, sigquant=0.90,
k=2, power=2, base=0.95,
lambda=NA, tau.num=c(NA, 3, 6)[ntype],
offset=NULL,
ntree=c(200L, 50L, 50L)[ntype], numcut=100L,
ndpost=1000L, nskip=100L,
keepevery=c(1L, 10L, 10L)[ntype],
printevery=100L, transposed=FALSE,
hostname=FALSE,
mc.cores = 2L, nice = 19L, seed = 99L
)

Arguments

x.train  Explanatory variables for training (in sample) data.
May be a matrix or a data frame, with (as usual) rows corresponding to obser-
vations 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 created if q = 2 where q is
the number of levels of the factor. gbmm will generate draws of f(x) for each x
which is a row of x.train.

y.train  Continuous or binary dependent variable for training (in sample) data.
If y is numeric, then a continuous BART model is fit (Normal errors).
If y is binary (has only 0’s and 1’s), then a binary BART model with a probit link
is fit by default: you can over-ride the default via the argument type to specify
a logit BART model.

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

u.train  Integer indices specifying the random effects.
B         The prior for the standard deviation of the random effects is U(0, B).
**type**
You can use this argument to specify the type of fit. 'wbart' for continuous BART or 'pbart' for probit BART.

**ntype**
The integer equivalent of type where 'wbart' is 1 and 'pbart' is 2.

**sparse**
Whether to perform variable selection based on a sparse Dirichlet prior rather than simply uniform; see Linero 2016.

**theta**
Set theta parameter; zero means random.

**omega**
Set omega parameter; zero means random.

**a**
Sparse parameter for Beta(a, b) prior: 0.5 <= a <= 1 where lower values inducing more sparsity.

**b**
Sparse parameter for Beta(a, b) prior; typically, b = 1.

**rho**
Sparse parameter: typically rho = p where p is the number of covariates under consideration.

**augment**
Whether data augmentation is to be performed in sparse variable selection.

**xinfo**
You can provide the cutpoints to BART or let BART choose them for you. To provide them, use the xinfo argument to specify a list (matrix) where the items (rows) are the covariates and the contents of the items (columns) are the cutpoints.

**usequants**
If usequants=FALSE, then the cutpoints in xinfo are generated uniformly; otherwise, if TRUE, uniform quantiles are used for the cutpoints.

**rm.const**
Whether or not to remove constant variables.

**sigest**
The prior for the error variance (sigma^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 aggressive the fit will be as you are putting more prior weight on error standard deviations (sigma) 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 +/-0.5. The response, codey.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. The bigger k is, the more conservative the fitting will be.

**power**
Power parameter for tree prior.

**base**
Base parameter for tree prior.

**lambda**
The scale of the prior for the variance. If lambda is zero, then the variance is to be considered fixed and known at the given value of sigest. Not used if y is binary.

**tau.num**
The numerator in the tau definition, i.e., tau=tau.num/(k*sqrt(ntree)).
offset

Continuous BART operates on \( y_{\text{train}} \) centered by \( \text{offset} \) which defaults to \( \text{mean}(y_{\text{train}}) \). With binary BART, the centering is \( P(Y = 1|x) = F(f(x) + \text{offset}) \) where \( \text{offset} \) defaults to \( F^{-1}(\text{mean}(y_{\text{train}})) \). You can use the \( \text{offset} \) parameter to over-ride these defaults.

ntree

The number of trees in the sum.

numcut

The number of possible values of \( c \) (see usequants). If a single number is given, this is used for all variables. Otherwise a vector with length equal to \( ncol(x_{\text{train}}) \) is required, where the \( i^{\text{th}} \) element gives the number of \( c \) used for the \( i^{\text{th}} \) variable in \( x_{\text{train}} \). If usequants is false, numcut equally spaced cutoffs are used covering the range of values in the corresponding column of \( x_{\text{train}} \). If usequants is true, then \( \min(\text{numcut}, \text{the number of unique values in the corresponding column of } x_{\text{train}} - 1) \) values are used.

ndpost

The number of posterior draws returned.

nskip

Number of MCMC iterations to be treated as burn in.

printevery

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

keepevery

Every keepevery draw is kept to be returned to the user.

transposed

When running gbmm in parallel, it is more memory-efficient to transpose \( x_{\text{train}} \) and \( x_{\text{test}} \), if any, prior to calling \( \text{mc.gbmm} \).

hostname

When running on a cluster occasionally it is useful to track on which node each chain is running; to do so set this argument to \( \text{TRUE} \).

seed

Setting the seed required for reproducible MCMC.

mc.cores

Number of cores to employ in parallel.

nice

Set the job niceness. The default niceness is 19: niceness goes from 0 (highest) to 19 (lowest).

Details

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

Thus, unlike a lot of other 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 \( \sigma^* \) in the numeric case) where * denotes a particular draw. The \( x \) is either a row from the training data, \( x_{\text{train}} \) or the test data, \( x_{\text{test}} \).

For \( x_{\text{train}}/x_{\text{test}} \) with missing data elements, gbmm will singly impute them with hot decking. For one or more missing covariates, record-level hot-decking imputation deWaPann11 is employed that is biased towards the null, i.e., nonmissing values from another record are randomly selected regardless of the outcome. Since \( \text{mc.gbmm} \) runs multiple gbmm threads in parallel, \( \text{mc.gbmm} \) performs multiple imputation with hot docking, i.e., a separate imputation for each thread. This record-level hot-decking imputation is biased towards the null, i.e., nonmissing values from another record are randomly selected regardless of \( y_{\text{train}} \).
Value

gbm returns an object of type gbmm which is essentially a list. In the numeric y case, the list has components:

- **yhat.train**: A matrix with ndpost 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.

- **yhat.train.mean**: Train data fits = mean of yhat.train columns.

- **yhat.test.mean**: Test data fits = mean of yhat.test columns.

- **sigma**: Post burn in draws of sigma, length = ndpost.

- **first.sigma**: Burn-in draws of sigma.

- **varcount**: A matrix with ndpost 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 ($\sigma$) used in the prior.

Author(s)

Robert McCulloch: <robert.e.mcculloch@gmail.com>,
Rodney Sparapani: <rsparapa@mcw.edu>.

References


See Also

- wbart, pbart
Examples

```r
# simulate data (example from Friedman 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 BART later

# test BART with token run to ensure installation works
set.seed(99)
bartFit = wbart(x,y,nskip=5,ndpost=5)

## Not run:
## run BART
set.seed(99)
bartFit = wbart(x,y)

## compare BART fit to linear matter and truth = Ey
fitmat = cbind(y,Ey,lmFit$fitted,bartFit$yhat.train.mean)
colnames(fitmat) = c('y','Ey','lm','bart')
print(cor(fitmat))
```

---

geweke.diag

Geweke’s convergence diagnostic

Description

Geweke (1992) proposed a convergence diagnostic for Markov chains based on a test for equality of the means of the first and last part of a Markov chain (by default the first 10% and the last 50%). If the samples are drawn from the stationary distribution of the chain, the two means are equal and Geweke’s statistic has an asymptotically standard normal distribution.

The test statistic is a standard Z-score: the difference between the two sample means divided by its estimated standard error. The standard error is estimated from the spectral density at zero and so takes into account any autocorrelation.

The Z-score is calculated under the assumption that the two parts of the chain are asymptotically independent, which requires that the sum of \( \frac{1}{1} \) and \( \frac{2}{2} \) be strictly less than 1.

Adapted from the `geweke.diag` function of the coda package which passes `mcmc` objects as arguments rather than matrices.

Usage

```r
geweke.diag(x, frac1=0.1, frac2=0.5)
```
gewekediag

Arguments

x Matrix of MCMC chains: the rows are the samples and the columns are different "parameters". For BART, generally, the columns are estimates of \( f \). For pbart, they are different subjects. For surv.bart, they are different subjects at a grid of times.

frac1 fraction to use from beginning of chain

frac2 fraction to use from end of chain

Value

Z-scores for a test of equality of means between the first and last parts of the chain. A separate statistic is calculated for each variable in each chain.

References


See Also

spectrum0ar.

Examples

## load survival package for the advanced lung cancer example
data(lung)

group <- -which(is.na(lung[, 7])) ## remove missing row for ph.karno
times <- lung[group, 2] ##lung$time
delta <- lung[group, 3]-1 ##lung$status: 1=censored, 2=dead
    ##delta: 0=censored, 1=dead

## this study reports time in days rather than months like other studies
## coarsening from days to months will reduce the computational burden
times <- ceiling(times/30)

summary(times)
table(delta)

x.train <- as.matrix(lung[group, c(4, 5, 7)]) ## matrix of observed covariates

## lung$age: Age in years
## lung$sex: Male=1 Female=2
## lung$ph.karno: Karnofsky performance score (dead=0:normal=100:by=10)
## rated by physician
dimnames(x.train[[2]]) <- c('age(yr)', 'M(1):F(2)', 'ph.karno(0:100:10)')

summary(x.train[, 1])
table(x.train[, 2])
table(x.train[, 3])

x.test <- matrix(nrow=84, ncol=3) ## matrix of covariate scenarios

dimnames(x.test)[[2]] <- dimnames(x.train)[[2]]
i <- 1

for(age in 5*(9:15)) for(sex in 1:2) for(ph.karno in 10*(5:10)) {
  x.test[i, ] <- c(age, sex, ph.karno)
i <- i+1
}

### Not run:
set.seed(99)
post <- surv.bart(x.train=x.train, times=times, delta=delta, x.test=x.test)
### in the interest of time, consider speeding it up by parallel processing
### run "mc.cores" number of shorter MCMC chains in parallel processes
### post <- mc.surv.bart(x.train=x.train, times=times, delta=delta,
### x.test=x.test, mc.cores=8, seed=99)

N <- nrow(x.test)
K <- post$K
## select 10 lung cancer patients uniformly spread out over the data set
h <- seq(1, N*K, floor(N/10)*K)

for(i in h) {
  post.mcmc <- post$yhat.test[, (i-1)+1:K]
  z <- gewekediag(post.mcmc)$z
  y <- max(c(4, abs(z)))

  ## plot the z scores vs. time for each patient
  if(i==1) plot(post$times, z, ylim=c(-y, y), type='l',
               xlab='t', ylab='z')
  else lines(post$times, z, type='l')
}

## add two-sided alpha=0.05 critical value lines
lines(post$times, rep(-1.96, K), type='l', lty=2)
lines(post$times, rep( 1.96, K), type='l', lty=2)

### End(Not run)
**Description**

BART is a Bayesian “sum-of-trees” model.
For numeric response \( y \), we have \( y = f(x) + \epsilon \), where \( \epsilon \sim \text{Log}(0,1) \).
For a binary response \( y \), \( P(Y = 1|x) = F(f(x)) \), where \( F \) denotes the standard Logistic CDF (logit link).
In both cases, \( f \) is the sum of many tree models. The goal is to have very flexible inference for the unknown 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**

```r
lbart(x.train, y.train, x.test=matrix(0.0,0,0), sparse=FALSE, a=0.5, b=1, augment=FALSE, rho=NULL, xinfo=matrix(0.0,0,0), usequants=FALSE, cont=FALSE, rm.const=TRUE, tau.interval=0.95, k=2.0, power=2.0, base=.95, binaryoffset=NULL, ntree=200L, numcut=100L, ndpost=1000L, nskip=100L, keepevery=1L, nkeeptrain=ndpost, nkeeptest=ndpost, nkeeptreedraws=ndpost, printevery=100L, transposed=FALSE)
```

**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. \( lbart \) will generate draws of \( f(x) \) for each \( x \) which is a row of \( x.train \).

- **y.train** Binary dependent variable for training (in sample) data.

- **x.test** Explanatory variables for test (out of sample) data. Should have same structure as \( x.train \). \( lbart \) will generate draws of \( f(x) \) for each \( x \) which is a row of \( x.test \).

- **sparse** Whether to perform variable selection based on a sparse Dirichlet prior rather than simply uniform; see Linero 2016.

- **a** Sparse parameter for \( Beta(a,b) \) prior: \( 0.5 \leq a \leq 1 \) where lower values inducing more sparsity.
b
  Sparse parameter for Beta(a,b) prior; typically, b = 1.

rho
  Sparse parameter: typically rho = p where p is the number of covariates under
  consideration.

augment
  Whether data augmentation is to be performed in sparse variable selection.

xinfo
  You can provide the cutpoints to BART or let BART choose them for you. To
  provide them, use the xinfo argument to specify a list (matrix) where the items
  (rows) are the covariates and the contents of the items (columns) are the cut-
  points.

usequants
  If usequants=FALSE, then the cutpoints in xinfo are generated uniformly; oth-
  erwise, if TRUE, uniform quantiles are used for the cutpoints.

cont
  Whether or not to assume all variables are continuous.

rm.const
  Whether or not to remove constant variables.

tau.interval
  The width of the interval to scale the variance for the terminal leaf values.

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).

ntree
  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.

ndpost
  The number of posterior draws returned.

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

nkeeptrain
  Number of MCMC iterations to be returned for train data.

nkeeptest
  Number of MCMC iterations to be returned for test data.

nkeeptreedraws
  Number of MCMC iterations to be returned for tree draws.

keepevery
  Every keepevery draw is kept to be returned to the user.

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

transposed
  When running lbart in parallel, it is more memory-efficient to transpose x.train
  and x.test, if any, prior to calling mc.lbart.
Details

BART is a Bayesian MCMC method. At each MCMC iteration, we produce a draw from the joint posterior $f|(x, y)$ 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 $f^*(x)$

where * denotes a particular draw. The $x$ is either a row from the training data (x.train) or the test data (x.test).

Value

`lbart` returns an object of type `lbart` which is essentially a list.

- **yhat.train** A matrix with ndpost 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.

- **yhat.train.mean** train data fits = mean of yhat.train columns.

- **yhat.test.mean** test data fits = mean of yhat.test columns.

- **varcount** a matrix with ndpost 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.

In addition, the list has a `binaryOffset` 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 Logistic CDF ($plogis$) to these values.

Author(s)

Robert McCulloch: <robert.e.mcculloch@gmail.com>,
Robert Gramacy: <rbg@vt.edu>,
Rodney Sparapani: <rsparapa@mcw.edu>.

References


**See Also**

 wbart

**Examples**

```r
data(ACTG175)

## exclude those who do not have CD4 count at 96 weeks
ex <- is.na(ACTG175$cd496)
table(ex)

## inclusion criteria are CD4 counts between 200 and 500
ACTG175$cd40 <- min(500, max(250, ACTG175$cd40))

## calculate relative CD4 decline
y <- ((ACTG175$cd496-ACTG175$cd40)/ACTG175$cd40)[!ex]
summary(y)

## 0=failure, 1=success
y <- 1*(y > -0.5)

## summarize CD4 outcomes
table(y, ACTG175$arms[!ex])

## drop unneeded and unwanted variables
# 1: 'pidnum' patient ID number
# 14: 'str2' which will be handled by strat1 below
# 15: 'strat' which will be handled by strat1-strat3 below
# 17: 'treat' handled by arm0-arm3 below
# 18: 'offtrt' indicator of off-treatment before 96 weeks
# 20: 'cd420' CD4 T cell count at 20 weeks
# 21: 'cd496' CD4 T cell count at 96 weeks
# 22: 'r' missing CD4 T cell count at 96 weeks
# 24: 'cd820' CD8 T cell count at 20 weeks
# 25: 'cens' indicator of observing the event in days
# 26: 'days' number of days until the primary endpoint
# 27: 'arms' handled by arm0-arm3 below

train <- as.matrix(ACTG175)[!ex, -c(1, 14:15, 17, 18, 20:22, 24:27)]
train <- cbind(1*(ACTG175$strat[!ex]==1), 1*(ACTG175$strat[!ex]==2),
               1*(ACTG175$strat[!ex]==3), train)
dimnames(train)[[2]][1:3] <- paste0('strat', 1:3)
train <- cbind(1*(ACTG175$arms[!ex]==0), 1*(ACTG175$arms[!ex]==1),
               train)
```
1*(ACTG175$arms[!ex]==2), 1*(ACTG175$arms[!ex]==3), train)

dimnames(train)[[2]][1:4] <- paste0('arm', 0:3)

N <- nrow(train)

test0 <- train; test0[, 1:4] <- 0; test0[, 1] <- 1
test1 <- train; test1[, 1:4] <- 0; test1[, 2] <- 1
test2 <- train; test2[, 1:4] <- 0; test2[, 3] <- 1
test3 <- train; test3[, 1:4] <- 0; test3[, 4] <- 1

test <- rbind(test0, test1, test2, test3)

##test BART with token run to ensure installation works
## set.seed(21)
## post <- lbart(train, y, test, nskip=5, ndpost=5)

## Not run:
set.seed(21)
post <- lbart(train, y, test)

## turn z-scores into probabilities
post$prob.test <- plogis(post$yhat.test)

## average over the posterior samples
post$prob.test.mean <- apply(post$prob.test, 2, mean)

## place estimates for arms 0-3 next to each other for convenience
itr <- cbind(post$prob.test.mean[(1:N)], post$prob.test.mean[N+(1:N)],
post$prob.test.mean[2*N+(1:N)], post$prob.test.mean[3*N+(1:N)])

## find the BART ITR for each patient
itr.pick <- integer(N)
for(i in 1:N) itr.pick[i] <- which(itr[i, ]==max(itr[i, ]))-1

## arms 0 and 3 (monotherapy) are never chosen
table(itr.pick)

## do arms 1 and 2 show treatment heterogeneity?
diff. <- apply(post$prob.test[ , 2*N+(1:N)]-post$prob.test[ , N+(1:N)], 2, mean)
plot(sort(diff.), type='h', main='ACTG175 trial: 50% CD4 decline from baseline at 96 weeks',
xlab='Arm 2 (1) Preferable to the Right (Left)', ylab='Prob.Diff.: Arms 2 - 1')

library(rpart)
library(rpart.plot)

## make data frame for nicer names in the plot
var <- as.data.frame(train[, -(1:4)])

dss <- rpart(diff. ~ var$age+var$gender+var$race+var$wtkg+var$cd40+var$cd80+
var$karnof+var$symptom+var$hemo+var$homo+var$drugs+var$z30+
var$zprior+var$oprior+var$strat1+var$strat2+var$strat3,
method='anova', control=rpart.control(cp=0.1))
rpart.plot(dss, type=3, extra=101)
### Description

137 patients with acute myelocytic leukemia (AML) and acute lymphoblastic leukemia (ALL) were given oral busulfan (Bu) 4 mg/kg on each of 4 days and intravenous cyclophosphamide (Cy) 60 mg/kg on each of 2 days (BuCy2) followed by allogeneic bone marrow transplantation from an HLA-identical or one antigen disparate sibling.

### Usage

data(leukemia)
leukemia

Format

A data frame with 137 subjects on the following 22 variables.

G Disease Group (1=ALL, 2=AML Low Risk in first remission, 3=AML High Risk not in first remission)
T0 Time To Death Or On Study Time
TB Disease Free Survival Time (Time To Relapse, Death Or End Of Study)
D Death Indicator (0=Alive, 1=Dead)
R Relapse Indicator (0=Disease Free, 1=Relapsed)
B Disease Free Survival Indicator (0=Alive and Disease Free, 1=Dead or Relapsed)
TA Time To Acute Graft-Versus-Host Disease (GVHD)
A Acute GVHD Indicator (0=Never Developed Acute GVHD, 1=Developed Acute GVHD)
TC Time To Chronic Graft-Versus-Host Disease (GVHD)
C Chronic GVHD Indicator (0=Never Developed Chronic GVHD, 1=Developed Chronic GVHD)
TP Time of Platelets Returning to Normal Levels
P Platelet Recovery Indicator (0=Platelets Never Returned to Normal, 1=Platelets Returned To Normal)
X1 Patient Age In Years
X2 Donor Age In Years
X3 Patient Gender (0=female, 1=male)
X4 Donor Gender (0=female, 1=male)
X5 Patient Cytomegalovirus (CMV) Immune Status (0=CMV Negative, 1=CMV Positive)
X6 Donor Cytomegalovirus (CMV) Immune Status (0=CMV Negative, 1=CMV Positive)
X7 Waiting Time to Transplant In Days
X8 AML Patients with Elevated Risk By French-American-British (FAB) Classification (0=Not AML/Elevated, 1=FAB M4 Or M5 with AML)
X9 Hospital (1=The Ohio State University in Columbus, 2=Alfred in Melbourne, 3=St. Vincent in Sydney, 4=Hahnemann University in Philadelphia)
X10 Methotrexate Used as a Graft-Versus-Host Disease Prophylactic (0=No, 1=Yes)

Source


References

**Description**

Survival in patients with advanced lung cancer from the North Central Cancer Treatment Group. Performance scores rate how well the patient can perform usual daily activities.

**Format**

- **inst:** Institution code
- **time:** Survival time in days
- **status:** censoring status 1=censored, 2=dead
- **age:** Age in years
- **sex:** Male=1 Female=2
- **ph.ecog:** ECOG performance score (0=good 5=dead)
- **ph.karno:** Karnofsky performance score (bad=0-good=100) rated by physician
- **pat.karno:** Karnofsky performance score as rated by patient
- **meal.cal:** Calories consumed at meals
- **wt.loss:** Weight loss in last six months

**Source**

Terry Therneau

**References**


**Examples**

```r
data(lung)
```

**mbart**

*Multinomial BART for categorical outcomes with fewer categories*
Description

BART is a Bayesian “sum-of-trees” model.
For numeric response \( y \), we have \( y = f(x) + \epsilon \), where \( \epsilon \sim N(0, 1) \).
For a multinomial response \( Y \), \( P(Y = y|x) = F(f(x)) \), where \( F \) denotes the standard Normal CDF (probit link) or the standard Logistic CDF (logit link).

In both cases, \( f \) is the sum of many tree models. The goal is to have very flexible inference for the unknown 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

```r
mbart(
  x.train, y.train,
  x.test=matrix(0,0,0), type='pbart',
  ntype=as.integer(
    factor(type,
      levels=c('wbart', 'pbart', 'lbart'))),
  sparse=FALSE, theta=0, omega=1,
  a=0.5, b=1, augment=FALSE, rho=NULL,
  xinfo=matrix(0,0,0), usequants=FALSE,
  rm.const=TRUE,
  k=2, power=2, base=0.95,
  tau.num=c(NA, 3, 6)[ntype],
  offset=NULL,
  ntree=c(200L, 50L, 50L)[ntype], numcut=100L,
  ndpost=1000L, nskip=100L,
  keepevery=c(1L, 10L, 10L)[ntype],
  printevery=100L, transposed=FALSE,
  hostname=FALSE,
  mc.cores = 2L, ## mc.bart only
  nice = 19L,     ## mc.bart only
  seed = 99L      ## mc.bart only
)
```

```r
mc.mbart(
  x.train, y.train,
  x.test=matrix(0,0,0), type='pbart',
  ntype=as.integer(
    factor(type,
      levels=c('wbart', 'pbart', 'lbart'))),
  sparse=FALSE, theta=0, omega=1,
  a=0.5, b=1, augment=FALSE, rho=NULL,
  xinfo=matrix(0,0,0), usequants=FALSE,
  rm.const=TRUE,
  k=2, power=2, base=0.95,
  tau.num=c(NA, 3, 6)[ntype],
  offset=NULL,
)```
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. mbart will generate draws of \( f(x) \) for each \( x \) which is a row of x.train.

y.train
Categorical dependent variable for training (in sample) data.

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

type
You can use this argument to specify the type of fit. 'pbart' for probit BART or 'lbart' for logit BART.

ntype
The integer equivalent of type where 'pbart' is 2 and 'lbart' is 3.

sparse
Whether to perform variable selection based on a sparse Dirichlet prior rather than simply uniform; see Linero 2016.

theta
Set theta parameter; zero means random.

omega
Set omega parameter; zero means random.

a
Sparse parameter for Beta(\(a,b\)) prior: 0.5 \(<= a <1\) where lower values inducing more sparsity.

b
Sparse parameter for Beta(\(a,b\)) prior; typically, \(b=1\).

rho
Sparse parameter: typically \(\rho = p\) where \(p\) is the number of covariates under consideration.

augment
Whether data augmentation is to be performed in sparse variable selection.

xinfo
You can provide the cutpoints to BART or let BART choose them for you. To provide them, use the xinfo argument to specify a list (matrix) where the items (rows) are the covariates and the contents of the items (columns) are the cutpoints.

usequants
If usequants=FALSE, then the cutpoints in xinfo are generated uniformly; otherwise, if TRUE, uniform quantiles are used for the cutpoints.

rm.const
Whether or not to remove constant variables.
k
For categorical y.train, k is the number of prior standard deviations \( f(x) \) is away from +/-3.

power
Power parameter for tree prior.

base
Base parameter for tree prior.

tau.num
The numerator in the tau definition, i.e., \( \tau = \tau.num/(k*\sqrt{n.tree}) \).

offset
With Multinomial BART, the centering is \( P(y_j = 1|x) = F(f_j(x)+offset[j]) \) where offset defaults to \( F^{-1}(\text{mean}(y.train)) \). You can use the offset parameter to over-ride these defaults.

ntree
The number of trees in the sum.

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.

ndpost
The number of posterior draws returned.

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

keepevery
Every keepevery draw is kept to be returned to the user.

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

transposed
When running mbart in parallel, it is more memory-efficient to transpose x.train and x.test, if any, prior to calling mc.mbart.

hostname
When running on a cluster occasionally it is useful to track on which node each chain is running; to do so set this argument to TRUE.

seed
Setting the seed required for reproducible MCMC.

mc.cores
Number of cores to employ in parallel.

nice
Set the job niceness. The default niceness is 19: niceness goes from 0 (highest) to 19 (lowest).

Details
BART is an Bayesian MCMC method. At each MCMC interation, we produce a draw from \( f \) in the categorical 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 \( f^*(x) \) where * denotes a particular draw. The x is either a row from the training data (x.train).

Value
mbart returns an object of type mbart which is essentially a list.
yhat.train  A matrix with ndpost rows and nrow(x.train)*K columns. Each row corresponds to a draw \( f^* \) from the posterior of \( f \) and each column corresponds to an estimate for a row of x.train. For the \( i \)th row of x.train, we provide the corresponding \((i-1)*K+j\)th column of yhat.train where \( j=1,...,K \) indexes the categories. Burn-in is dropped.

yhat.train.mean
train data fits = mean of yhat.train columns.

varcount a matrix with ndpost 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.

In addition, the list has a offset vector giving the value used.

Note that in the multinomial \( y \) case yhat.train is \( f(x) + offset[j] \).

Author(s)

Robert McCulloch: <robert.e.mcculloch@gmail.com>,
Rodney Sparapani: <rsparapa@mcw.edu>,
Robert Gramacy: <rbg@vt.edu>.

References


See Also

gbart, alligator

Examples

N=500
set.seed(12)
x1=runif(N)
x2=runif(N, max=1-x1)
x3=1-x1-x2
x.train=cbind(x1, x2, x3)
y.train=0
for(i in 1:N)
  y.train[i]=sum((1:3)*rmultinom(1, 1, x.train[i, ]))
table(y.train)/N

##test mbart with token run to ensure installation works
set.seed(99)
post = mbart(x.train, y.train, nskip=1, ndpost=1)

## Not run:
set.seed(99)
post=mbart(x.train, y.train, x.train)
##mc.post=mbart(x.train, y.train, x.test, mc.cores=8, seed=99)
K=3
i=seq(1, N*K, K)-1
for(j in 1:K)
  print(cor(x.train[, j], post$prob.test.mean[i+j])^2)

## End(Not run)

---

**mbart2**

*Multinomial BART for categorical outcomes with more categories*

**Description**

BART is a Bayesian “sum-of-trees” model.
For numeric response \( y \), we have \( y = f(x) + \epsilon \) \( y = f(x) + e \), where \( \epsilon \sim N(0, 1) \).
For a multinomial response \( y \), \( P(Y = y|x) = F(f(x)) \), where \( F \) denotes the standard Normal CDF (probit link) or the standard Logistic CDF (logit link).

In both cases, \( f \) is the sum of many tree models. The goal is to have very flexible inference for the unknown 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**

```r
mbart2(
  x.train, y.train,
  x.test=matrix(0,0,0), type='lbart',
  ntype=as.integer(
    factor(type,
      levels=c('wbart', 'pbart', 'lbart'))),
  sparse=FALSE, theta=0, omega=1,
  a=0.5, b=1, augment=FALSE, rho=NULL,
  xinfo=matrix(0,0,0), usequants=FALSE,
  rm.const=TRUE,
  k=2, power=2, base=0.95,
```

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. `mbart2` will generate draws of \( f(x) \) for each \( x \) which is a row of `x.train`.

**y.train**  
Categorical dependent variable for training (in sample) data.

**x.test**  
Explanatory variables for test (out of sample) data.
mbart2 should have the same structure as x.train.

mbart2 will generate draws of \( f(x) \) for each \( x \) which is a row of x.test.

type
You can use this argument to specify the type of fit. 'pbart' for probit BART or 'lbart' for logit BART.

ntype
The integer equivalent of type where 'pbart' is 2 and 'lbart' is 3.

sparse
Whether to perform variable selection based on a sparse Dirichlet prior rather than simply uniform; see Linero 2016.

theta
Set \( \theta \) parameter; zero means random.

omega
Set \( \omega \) parameter; zero means random.

a
Sparse parameter for \( \text{Beta}(a, b) \) prior: \( 0.5 \leq a \leq 1 \) where lower values inducing more sparsity.

b
Sparse parameter for \( \text{Beta}(a, b) \) prior; typically, \( b = 1 \).

rho
Sparse parameter: typically \( \rho = p \) where \( p \) is the number of covariates under consideration.

augment
Whether data augmentation is to be performed in sparse variable selection.

xinfo
You can provide the cutpoints to BART or let BART choose them for you. To provide them, use the xinfo argument to specify a list (matrix) where the items (rows) are the covariates and the contents of the items (columns) are the cutpoints.

usequants
If usequants=FALSE, then the cutpoints in xinfo are generated uniformly; otherwise, if TRUE, uniform quantiles are used for the cutpoints.

rm.const
Whether or not to remove constant variables.

k
For categorical y.train, \( k \) is the number of prior standard deviations \( f(x) \) is away from +/-3.

power
Power parameter for tree prior.

base
Base parameter for tree prior.

tau.num
The numerator in the \( \tau \) definition, i.e., \( \tau = \text{tau.num}/(k*\sqrt{\text{ntree}}) \).

offset
With Multinomial BART, the centering is \( P(y_j = 1|x) = F(f_j(x) + \text{offset}[j]) \) where offset defaults to \( F^{-1}(\text{mean}(y.train)) \). You can use the offset parameter to over-ride these defaults.

ntree
The number of trees in the sum.

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.

ndpost
The number of posterior draws returned.

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

keepevery
Every keepevery draw is kept to be returned to the user.
printevery  As the MCMC runs, a message is printed every printevery draws.

transposed  When running mbart2 in parallel, it is more memory-efficient to transpose x.train and x.test, if any, prior to calling mc.mbart2.

hostname  When running on a cluster occasionally it is useful to track on which node each chain is running; to do so set this argument to TRUE.

seed  Setting the seed required for reproducible MCMC.

mc.cores  Number of cores to employ in parallel.

nice  Set the job niceness. The default niceness is 19: niceness goes from 0 (highest) to 19 (lowest).

Details

BART is an Bayesian MCMC method. At each MCMC iteration, we produce a draw from \( f \) in the categorical \( y \) 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 \( f^*(x) \)

where * denotes a particular draw. The \( x \) is either a row from the training data (x.train).

Value

mbart2 returns an object of type mbart2 which is essentially a list.

\[
yhat.train \quad \text{A matrix with ndpost rows and nrow(x.train)*K columns. Each row corresponds to a draw } f^* \text{ from the posterior of } f \text{ and each column corresponds to an estimate for a row of } x.train. \text{ For the } ith \text{ row of } x.train, \text{ we provide the corresponding } (i-1)*K+jth \text{ column of yhat.train where } j=1,\ldots,K \text{ indexes the categories. } \text{Burn-in is dropped.}
\]

\[
yhat.train.mean \quad \text{train data fits = mean of yhat.train columns.}
\]

\[
varcount \quad \text{a matrix with ndpost rows and nrow(x.train) columns. Each row is for a draw. For each variable (corresponding to the columns), the total count of the number of times that variable is used in a tree decision rule (over all trees) is given.}
\]

In addition, the list has a offset vector giving the value used.

Note that in the multinomial \( y \) case yhat.train is \( f(x) + offset[j] \).

Author(s)

Robert McCulloch: <robert.e.mcculloch@gmail.com>,
Rodney Sparapani: <rsparapa@mcw.edu>,
Robert Gramacy: <rbg@vt.edu>. 
References


See Also

`gbart`, `alligator`

Examples

```r
N=500
set.seed(12)
x1=runif(N)
x2=runif(N, max=1-x1)
x3=1-x1-x2
x.train=cbind(x1, x2, x3)
y.train=0
for(i in 1:N)
  y.train[i]=sum((1:3)*rmultinom(1, 1, x.train[i, ]))
table(y.train)/N

##test mbart2 with token run to ensure installation works
set.seed(99)
post = mbart2(x.train, y.train, nskip=1, ndpost=1)

## Not run:
set.seed(99)
post=mbart2(x.train, y.train, x.train)
##mc.post=mbart2(x.train, y.train, x.test, mc.cores=8, seed=99)
K=3
i=seq(1, N*K, K)-1
for(j in 1:K)
  print(cor(x.train[, j], post$prob.test.mean[i+j])^2)

## End(Not run)
```
mc.cores.openmp  Detecting OpenMP

Description

This package was designed for OpenMP. For example, the `pwbart` function can use OpenMP or the parallel R package for multi-threading. On UNIX/Unix-like systems, OpenMP, if available, is discovered at install time; for the details, see the `configure.ac` file which can be found in the source version of this package. However, we know of no GPL licensed code available to detect OpenMP on Windows (for Artistic licensed OpenMP detection code on Windows, see the Bioconductor R package rGADEM). To determine whether OpenMP is available at run time, we provide the function documented here.

Usage

```r
mc.cores.openmp()
```

Value

Returns a zero when OpenMP is not available, otherwise, an integer greater than zero when OpenMP is available (returns one unless you are running in a multi-threaded process).

Author(s)

Robert McCulloch: `<robert.e.mcculloch@gmail.com>`.
Rodney Sparapani: `<rsparapa@mcw.edu>`.

See Also

`pwbart`

Examples

```r
mc.cores.openmp()
```
**Description**

BART is a Bayesian “sum-of-trees” model. For a numeric response $y$, we have $y = f(x) + \epsilon$, where $\epsilon \sim N(0, \sigma^2)$.

$f$ is the sum of many tree models. The goal is to have very flexible inference for the unknown 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**

```r
mc.crisk.pwbart( x.test, x.test2,
    treedraws, treedraws2,
    binaryOffset=0, binaryOffset2=0,
    mc.cores=2L, type='pbart',
    transposed=FALSE, nice=19L
)
```

**Arguments**

- `x.test` Matrix of covariates to predict $y$ for cause 1.
- `x.test2` Matrix of covariates to predict $y$ for cause 2.
- `treedraws` $t$reedraws for cause 1.
- `treedraws2` $t$reedraws for cause 2.
- `binaryOffset` Mean to add on to $y$ prediction for cause 1.
- `binaryOffset2` Mean to add on to $y$ prediction for cause 2.
- `mc.cores` Number of threads to utilize.
- `type` Whether to employ Albert-Chib, 'pbart', or Holmes-Held, 'lbart'.
- `transposed` When running `pwbart` or `mc.pwbart` in parallel, it is more memory-efficient to transpose `x.test` prior to calling the internal versions of these functions.
- `nice` Set the job niceness. The default niceness is 19: niceness goes from 0 (highest) to 19 (lowest).

**Details**

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

Thus, unlike a lot of other 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 $\sigma^*$ 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

Returns an object of type `crisk.bart` which is essentially a list with components:

- **yhat.test**
  - A matrix with `ndpost` 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.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.

- **surv.test**
  - Test data fits for survival probability.

- **surv.test.mean**
  - Mean of `surv.test` over the posterior samples.

- **prob.test**
  - The probability of suffering cause 1 which is occasionally useful, e.g., in calculating the concordance.

- **prob.test2**
  - The probability of suffering cause 2 which is occasionally useful, e.g., in calculating the concordance.

- **cif.test**
  - The cumulative incidence function of cause 1, $F_1(t,x)$, where x’s are the rows of the test data.

- **cif.test2**
  - The cumulative incidence function of cause 2, $F_2(t,x)$, where x’s are the rows of the test data.

- **yhat.test.mean**
  - Test data fits = mean of `yhat.test` columns.

- **cif.test.mean**
  - Mean of `cif.test` columns for cause 1.

- **cif.test2.mean**
  - Mean of `cif.test2` columns for cause 2.

Author(s)

Robert McCulloch: <robert.e.mcculloch@gmail.com>,
Rodney Sparapani: <rsparapa@mcw.edu>.

References


See Also

`pwbart`, `crisk.bart`, `mc.crisk.bart`

Examples

data(transplant)

```r
delta <- (as.numeric(transplant$event)-1)
# recode so that delta=1 is cause of interest; delta=2 otherwise
delta[delta==1] <- 4
delta[delta==2] <- 1
delta[delta>1] <- 2```

table(delta, transplant$event)

times <- pmax(1, ceiling(transplant$futime/7)) ## weeks
##times <- pmax(1, ceiling(transplant$futime/30.5)) ## months
table(times)

typeO <- 1*(transplant$abo=='O')
typeA <- 1*(transplant$abo=='A')
typeB <- 1*(transplant$abo=='B')
typeAB <- 1*(transplant$abo=='AB')
table(typeA, typeO)

x.train <- cbind(typeO, typeA, typeB, typeAB)
x.test <- cbind(1, 0, 0, 0)
dimnames(x.bind)[[2]] <- dimnames(x.train)[[2]]

## parallel::mcparallel/mccollect do not exist on windows
if(.Platform$OS.type=='unix') {
  ##test BART with token run to ensure installation works
  post <- mc.crisk.bart(x.train=x.train, times=times, delta=delta,
                        seed=99, mc.cores=2, nskip=5, ndpost=5,
                        keepevery=1)

  pre <- surv.pre.bart(x.train=x.train, x.test=x.test,
                       times=times, delta=delta)

  K <- post$K

  pred <- mc.crisk.pwbart(pre$tx.test, pre$tx.test,
                          post$treedraws, post$treedraws2,
                          post$binaryOffset, post$binaryOffset2)
}

## Not run:
## run one long MCMC chain in one process
## set.seed(99)
## post <- crisk.bart(x.train=x.train, times=times, delta=delta, x.test=x.test)

## in the interest of time, consider speeding it up by parallel processing
## run "mc.cores" number of shorter MCMC chains in parallel processes
post <- mc.crisk.bart(x.train=x.train,
                      times=times, delta=delta,
                      x.test=x.test, seed=99, mc.cores=8)

check <- mc.crisk.pwbart(post$tx.test, post$tx.test,
                          post$treedraws, post$treedraws2,
                          post$binaryOffset, post$binaryOffset2, mc.cores=8)

## check <- predict(post, newdata=post$tx.test, newdata2=post$tx.test2,
##                   mc.cores=8)
mc.crisk2.pwbart

Predicting new observations with a previously fitted BART model

Description

BART is a Bayesian “sum-of-trees” model.
For a numeric response $y$, we have $y = f(x) + \epsilon$, where $\epsilon \sim N(0, \sigma^2)$.

$f$ is the sum of many tree models. The goal is to have very flexible inference for the unknown 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

```r
mc.crisk2.pwbart( x.test, x.test2,
    treedraws, treedraws2,
    binaryOffset=0, binaryOffset2=0,
    mc.cores=2L, type='pbart',
    transposed=FALSE, nice=19L
)
```

Arguments

- **x.test**: Matrix of covariates to predict $y$ for cause 1.
- **x.test2**: Matrix of covariates to predict $y$ for cause 2.
- **treedraws**: $t$reedraws for cause 1.
treedraws2 $treedraws for cause 2.

binaryOffset Mean to add on to $y$ prediction for cause 1.

binaryOffset2 Mean to add on to $y$ prediction for cause 2.

mc.cores Number of threads to utilize.

type Whether to employ Albert-Chib, 'pbart', or Holmes-Held, 'lbart'.

transposed When running pbart or mc.pbart in parallel, it is more memory-efficiency to transpose $x.test$ prior to calling the internal versions of these functions.

nice Set the job niceness. The default niceness is 19: niceness goes from 0 (highest) to 19 (lowest).

**Details**

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

Thus, unlike a lot of other 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 $\sigma^*$ 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**

Returns an object of type `crisk2bart` which is essentially a list with components:

- `yhat.test` A matrix with `ndpost` 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 $x.train$. 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.

- `surv.test` test data fits for survival probability.

- `surv.test.mean` mean of `surv.test` over the posterior samples.

- `prob.test` The probability of suffering cause 1 which is occasionally useful, e.g., in calculating the concordance.

- `prob.test2` The probability of suffering cause 2 which is occasionally useful, e.g., in calculating the concordance.

- `cif.test` The cumulative incidence function of cause 1, $F_1(t, x)$, where $x$’s are the rows of the test data.

- `cif.test2` The cumulative incidence function of cause 2, $F_2(t, x)$, where $x$’s are the rows of the test data.

- `yhat.test.mean` test data fits = mean of `yhat.test` columns.

- `cif.test.mean` mean of `cif.test` columns for cause 1.

- `cif.test2.mean` mean of `cif.test2` columns for cause 2.
Author(s)

Robert McCulloch: <robert.e.mcculloch@gmail.com>,
Rodney Sparapani: <rsparapa@mcw.edu>.

References


See Also

pwbart, crisk2.bart, mc.crisk2.bart

Examples

data(transplant)

delta <- (as.numeric(transplant$event)-1)
## recode so that delta=1 is cause of interest; delta=2 otherwise
delta[delta==1] <- 4
delta[delta==2] <- 1
delta[delta>1] <- 2

plot(delta, transplant$event)

times <- pmax(1, ceiling(transplant$futime/7)) ## weeks
# times <- pmax(1, ceiling(transplant$futime/30.5)) ## months

plot(times)

typeO <- 1*(transplant$abo=='O')
typeA <- 1*(transplant$abo=='A')
typeB <- 1*(transplant$abo=='B')
typeAB <- 1*(transplant$abo=='AB')

plot(typeA, typeO)

x.train <- cbind(typeO, typeA, typeB, typeAB)
x.test <- cbind(1, 0, 0, 0)
dimnames(x.test)[[2]] <- dimnames(x.train)[[2]]

## parallel::mcparallel/mccollect do not exist on windows
if(.Platform$OS.type=='unix') {
  ##test BART with token run to ensure installation works
  post <- mc.crisk2.bart(x.train=x.train, times=times, delta=delta,
                          seed=99, mc.cores=2, nskip=5, ndpost=5,
                          keepevery=1)
  pre <- surv.pre.bart(x.train=x.train, x.test=x.test,
                       times=times, delta=delta)

  K <- post$K
pred <- mc.crisk2.pwbart(pre$tx.test, pre$tx.test, 
    post$treedraws, post$treedraws2, 
    post$binaryOffset, post$binaryOffset2)
}

## Not run:
## run one long MCMC chain in one process
## set.seed(99)
## post <- crisk2.bart(x.train=x.train, times=times, delta=delta, x.test=x.test)

## in the interest of time, consider speeding it up by parallel processing
## run "mc.cores" number of shorter MCMC chains in parallel processes
post <- mc.crisk2.bart(x.train=x.train, 
    times=times, delta=delta, 
    x.test=x.test, seed=99, mc.cores=8)

check <- mc.crisk2.pwbart(post$tx.test, post$tx.test, 
    post$treedraws, post$treedraws2, 
    post$binaryOffset, 
    post$binaryOffset2, mc.cores=8)

## check <- predict(post, newdata=post$tx.test, newdata2=post$tx.test2, 
## mc.cores=8)

print(c(post$surv.test.mean[1], check$surv.test.mean[1], 
    post$surv.test.mean[1]-check$surv.test.mean[1]), digits=22)
print(all(round(post$surv.test.mean, digits=9)==
    round(check$surv.test.mean, digits=9)))

print(c(post$cif.test.mean[1], check$cif.test.mean[1], 
    post$cif.test.mean[1]-check$cif.test.mean[1]), digits=22)
print(all(round(post$cif.test.mean, digits=9)==
    round(check$cif.test.mean, digits=9)))

print(c(post$cif.test2.mean[1], check$cif.test2.mean[1], 
    post$cif.test2.mean[1]-check$cif.test2.mean[1]), digits=22)
print(all(round(post$cif.test2.mean, digits=9)==
    round(check$cif.test2.mean, digits=9)))

## End(Not run)
Description

BART is a Bayesian “sum-of-trees” model. For numeric response \( y \), we have \( y = f(x) + \epsilon \), where \( \epsilon \sim \text{Log}(0, 1) \). For a binary response \( y \), \( P(Y = 1|x) = F(f(x)) \), where \( F \) denotes the standard Logistic CDF (logit link).

In both cases, \( f \) is the sum of many tree models. The goal is to have very flexible inference for the unknown 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

\[
\text{mc.lbart}(
  x.train, \ y.train, \ x.test=\text{matrix}(0.0,0,0),
  \text{sparse}=\text{FALSE}, \ a=0.5, \ b=1, \ \text{augment}=\text{FALSE}, \ \text{rho}=\text{NULL},
  \text{xinfo}=\text{matrix}(0.0,0,0), \ \text{usequants}=\text{FALSE},
  \text{cont}=\text{FALSE}, \ \text{rm.const}=\text{TRUE}, \ \text{tau.interval}=0.95,
  k=2.0, \ \text{power}=2.0, \ \text{base}=.95,
  \text{binaryoffset}=\text{NULL}, \ \text{ntree}=50L, \ \text{numcut}=100L,
  \text{ndpost}=1000L, \ \text{nskip}=100L,
  \text{keepevery}=1L, \ \text{printevery}=100,
  \text{keeptrainfits}=\text{TRUE}, \ \text{transposed}=\text{FALSE},
  \text{mc.cores} = 2L, \ \text{nice} = 19L,
  \text{seed} = 99L
\)
\]

Arguments

\text{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. \text{lbar}t will generate draws of \( f(x) \) for each \( x \) which is a row of \text{x.train}.

\text{y.train} Dependent variable for training (in sample) data. If \( y \) is numeric a continuous 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 logit link is fit.

\text{x.test} Explanatory variables for test (out of sample) data. Should have same structure as \text{x.train}. \text{lbar}t will generate draws of \( f(x) \) for each \( x \) which is a row of \text{x.test}.

\text{sparse} Whether to perform variable selection based on a sparse Dirichlet prior rather than simply uniform; see Linero 2016.
Sparse parameter for \( Beta(a, b) \) prior: \( 0.5 <= a <= 1 \) where lower values inducing more sparsity.

Sparse parameter for \( Beta(a, b) \) prior; typically, \( b = 1 \).

Sparse parameter: typically \( \rho = p \) where \( p \) is the number of covariates under consideration.

Whether data augmentation is to be performed in sparse variable selection.

You can provide the cutpoints to BART or let BART choose them for you. To provide them, use the \( xinfo \) argument to specify a list (matrix) where the items (rows) are the covariates and the contents of the items (columns) are the cutpoints.

If \( usequants=FALSE \), then the cutpoints in \( xinfo \) are generated uniformly; otherwise, if \( TRUE \), uniform quantiles are used for the cutpoints.

Whether or not to assume all variables are continuous.

Whether or not to remove constant variables.

The width of the interval to scale the variance for the terminal leaf values.

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.

Power parameter for tree prior.

Base parameter for tree prior.

Used for binary \( y \).

The model is \( P(Y = 1|x) = F(f(x) + binaryOffset) \).

The number of trees in the sum.

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.

The number of posterior draws returned.

Number of MCMC iterations to be treated as burn in.

Every keepevery draw is kept to be returned to the user.

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

Whether to keep yhat.train or not.

When running lbart in parallel, it is more memory-efficient to transpose \( x.train \) and \( x.test \), if any, prior to calling \( mc.lbart \).

Setting the seed required for reproducible MCMC.

Number of cores to employ in parallel.

Set the job niceness. The default niceness is 19: niceness goes from 0 (highest) to 19 (lowest).
Details

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

Thus, unlike a lot of other 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 \(\sigma^*\) 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

\texttt{mc.lbart} returns an object of type \texttt{lbart} which is essentially a list.

- \texttt{yhat.train} A matrix with \(ndpost\) 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.
- \texttt{yhat.test} Same as \texttt{yhat.train} but now the \(x\)'s are the rows of the test data.
- \texttt{yhat.train.mean} train data fits = mean of \texttt{yhat.train} columns.
- \texttt{yhat.test.mean} test data fits = mean of \texttt{yhat.test} columns.
- \texttt{varcount} a matrix with \(ndpost\) 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.

In addition, the list has a \texttt{binaryOffset} giving the value used.

Note that in the binary \(y\), case \texttt{yhat.train} and \texttt{yhat.test} are \(f(x) + \text{binaryOffset}\). If you want draws of the probability \(P(Y = 1|x)\) you need to apply the Logistic cdf (\texttt{plogis}) to these values.

Author(s)

Robert McCulloch: \texttt{<robert.e.mcculloch@gmail.com>},
Rodney Sparapani: \texttt{<rsparapa@mcw.edu>}.

References


mc.lbart

See Also

lbart

Examples

```r
set.seed(99)
n=5000
x = sort(-2+4*runif(n))
X=matrix(x,ncol=1)
f = function(x) {return((1/2)*x^3)}
FL = function(x) {return(exp(x)/(1+exp(x)))}
pv = FL(f(x))
y = rbinom(n,1,pv)
np=100
xp=-2+4*(1:n)/np
Xp=matrix(xp,ncol=1)

## parallel::mcparallel/mccollect do not exist on windows
## if(.Platform$OS.type=='unix') {
## ##test BART with token run to ensure installation works
## mf = mc.lbart(X, y, nskip=5, ndpost=5, mc.cores=1, seed=99)
## }

## Not run:
set.seed(99)

pf = lbart(X,y,Xp)
plot(f(Xp), pf$yhat.test.mean, xlim=c(-4, 4), ylim=c(-4, 4), xlab='True f(x)', ylab='BART f(x)')
lines(c(-4, 4), c(-4, 4))

mf = mc.lbart(X,y,Xp, mc.cores=4, seed=99)
plot(f(Xp), mf$yhat.test.mean, xlim=c(-4, 4), ylim=c(-4, 4), xlab='True f(x)', ylab='BART f(x)')
lines(c(-4, 4), c(-4, 4))

par(mfrow=c(2,2))

plot(range(xp),range(pf$yhat.test),xlab='x',ylab='f(x)',type='n')
lines(x,f(x),col='blue',lwd=2)
lines(xp,apply(pf$yhat.test,2,mean),col='red')
qpl = apply(pf$yhat.test,2,quantile,probs=c(.025,.975))
lines(xp,qpl[1,],col='green',lty=1)
lines(xp,qpl[2,],col='green',lty=1)
title(main='BART::lbart f(x) with 0.95 intervals')

plot(range(xp),range(mf$yhat.test),xlab='x',ylab='f(x)',type='n')
lines(x,f(x),col='blue',lwd=2)
lines(xp,apply(mf$yhat.test,2,mean),col='red')
qpl = apply(mf$yhat.test,2,quantile,probs=c(.025,.975))
lines(xp,qpl[1,],col='green',lty=1)
```

mc.pbart

Probit BART for dichotomous outcomes with Normal latents and parallel computation

Description

BART is a Bayesian “sum-of-trees” model.

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 unknown 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

```r
mc.pbart(
x.train, y.train, x.test=matrix(0.0,0,0),
sparse=FALSE, theta=0, omega=1,
a=0.5, b=1, augment=FALSE, rho=NULL,
xinfo=matrix(0.0,0,0), usequants=FALSE,
cont=FALSE, rm.const=TRUE,
k=2.0, power=2.0, base=.95,
binaryoffset=NULL,
ntree=50L, numcut=100L,
ndpost=1000L, nskip=100L,
keepevery=1L, printevery=100,
keeptrainfits=TRUE, transposed=FALSE,
mc.cores = 2L, nice = 19L,
seed = 99L
)
```
mc.pbart

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. \texttt{pbart} will generate draws of \( f(x) \) for each \( x \) which is a row of \( x.train \).

y.train Binary dependent variable for training (in sample) data.

x.test Explanatory variables for test (out of sample) data. Should have same structure as \( x.train \). \texttt{pbart} will generate draws of \( f(x) \) for each \( x \) which is a row of \( x.test \).

sparse Whether to perform variable selection based on a sparse Dirichlet prior rather than simply uniform; see Linero 2016.

theta Set \( \theta \) parameter; zero means random.

omega Set \( \omega \) parameter; zero means random.

a Sparse parameter for \( \text{Beta}(a,b) \) prior: \( 0.5 <= a <= 1 \) where lower values inducing more sparsity.

b Sparse parameter for \( \text{Beta}(a,b) \) prior; typically, \( b = 1 \).

rho Sparse parameter: typically \( \rho = p \) where \( p \) is the number of covariates under consideration.

augment Whether data augmentation is to be performed in sparse variable selection.

xinfo You can provide the cutpoints to BART or let BART choose them for you. To provide them, use the \texttt{xinfo} argument to specify a list (matrix) where the items (rows) are the covariates and the contents of the items (columns) are the cutpoints.

usequants If \texttt{usequants}=FALSE, then the cutpoints in \texttt{xinfo} are generated uniformly; otherwise, if \texttt{TRUE}, uniform quantiles are used for the cutpoints.

cont Whether or not to assume all variables are continuous.

rm.const Whether or not to remove constant variables.

k For binary \( y \), \( k \) is the number of prior standard deviations \( f(x) \) is away from +/-3. 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 \( P(Y = 1|x) = F(f(x) + \text{binaryOffset}) \).

ntree The number of trees in the sum.
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.

ndpost  The number of posterior draws returned.

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

keepevery  Every keepevery draw is kept to be returned to the user.

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

keeptainfits  Whether to keep yhat.train or not.

transposed  When running pbart in parallel, it is more memory-efficient to transpose x.train and x.test, if any, prior to calling mc.pbart.

seed  Setting the seed required for reproducible MCMC.

mc.cores  Number of cores to employ in parallel.

nice  Set the job niceness. The default niceness is 19: niceness goes from 0 (highest) to 19 (lowest).

Details  

BART is an Bayesian MCMC method. At each MCMC interation, we produce a draw from $f$ 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 $f^*(x)$ where * denotes a particular draw. The $x$ is either a row from the training data (x.train) or the test data (x.test).

Value  

mc.pbart returns an object of type pbart which is essentially a list.

yhat.train  A matrix with ndpost 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.

varcount  A matrix with ndpost 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.

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.
mc.pbart

Author(s)

Robert McCulloch: <robert.e.mcculloch@gmail.com>,
Rodney Sparapani: <rsparapa@mcw.edu>.

References


See Also

pbart

Examples

```r
set.seed(99)
n=5000
x = sort(-2+4*runif(n))
X=matrix(x,ncol=1)
f = function(x) {return((1/2)*x^3)}
FL = function(x) {return(exp(x)/(1+exp(x)))}
pv = FL(f(x))
y = rbinom(n,1,pv)
np=100
xp=-2+4*(1:np)/np
Xp=matrix(xp,ncol=1)

## parallel::mcpparallel/mccollect do not exist on windows
if(.Platform$OS.type=='unix') {
  ##test BART with token run to ensure installation works
  mf = mc.pbart(X, y, nskip=5, ndpost=5, mc.cores=1, seed=99)
}

## Not run:
set.seed(99)
pf = pbart(X,y,Xp)

## plot(f(Xp), pf$yhat.test.mean, xlim=c(-4, 4), ylim=c(-4, 4),
## xlab='True f(x)', ylab='BART f(x)'
## lines(c(-4, 4), c(-4, 4))

mf = mc.pbart(X,y,Xp, mc.cores=4, seed=99)
```
mc.surv.pwbart

Predicting new observations with a previously fitted BART model

Description

BART is a Bayesian “sum-of-trees” model.
For a numeric response \( y \), we have \( y = f(x) + \epsilon \), where \( \epsilon \sim N(0, \sigma^2) \).

\( f \) is the sum of many tree models. The goal is to have very flexible inference for the unknown 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

```
surv.pwbart(
  x.test,
  treedraws,
  binaryOffset=0,
  mc.cores=1L,
  type='pbart',
```
mc.surv.pwbart

transposed=FALSE, nice=19L
)

mc.surv.pwbart(
  x.test,
  treedraws,
  binaryOffset=0,
  mc.cores=2L,
  type='pbart',
  transposed=FALSE, nice=19L
)

mc.recur.pwbart(
  x.test,
  treedraws,
  binaryOffset=0,
  mc.cores=2L,
  type='pbart',
  transposed=FALSE, nice=19L
)

Arguments

x.test Matrix of covariates to predict y for.
binaryOffset Mean to add on to y prediction.
treedraws $treedraws returned from surv.bart, mc.surv.bart, recur.bart or mc.recur.bart.
mc.cores Number of threads to utilize.
type Whether to employ Albert-Chib, 'pbart', or Holmes-Held, 'lbart'.
transposed When running pwbart or mc.pwbart in parallel, it is more memory-efficient to
  transpose x.test prior to calling the internal versions of these functions.
nice Set the job niceness. The default niceness is 19: niceness goes from 0 (highest)
  to 19 (lowest).

Details

BART 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 \(\sigma^*\) 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

Returns an object of type survbart which is essentially a list with components:
A matrix with ndpost 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.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.

surv.test test data fits for survival probability: not available for mc.recur.pwbart.
surv.test.mean mean of surv.test over the posterior samples: not available for mc.recur.pwbart.
haz.test test data fits for hazard: available for mc.recur.pwbart only.
haz.test.mean mean of haz.test over the posterior samples: available for mc.recur.pwbart only.
cum.test test data fits for cumulative hazard: available for mc.recur.pwbart only.
cum.test.mean mean of cum.test over the posterior samples: available for mc.recur.pwbart only.

Author(s)
Robert McCulloch: <robert.e.mcculloch@gmail.com>,
Rodney Sparapani: <rsparapa@mcw.edu>.

References

See Also
pwbart

Examples

```r
## load the advanced lung cancer example
data(lung)

group <- -which(is.na(lung[, 7]))  ## remove missing row for ph.karno
times <- lung[group, 2]  ##lung$time
delta <- lung[group, 3]-1  ##lung$status: 1=censored, 2=dead
  ##delta: 0=censored, 1=dead

## this study reports time in days rather than months like other studies
## coarsening from days to months will reduce the computational burden
times <- ceiling(times/30)

summary(times)
table(delta)

x.train <- as.matrix(lung[group, c(4, 5, 7)])  ## matrix of observed covariates
  ##lung$age: Age in years
```
## lung$sex: Male=1 Female=2
## lung$ph.karno: Karnofsky performance score (dead=0:normal=100:by=10)
## rated by physician

dimnames(x.train)[[2]] <- c('age(yr)', 'M(1):F(2)', 'ph.karno(0:100:10)')

summary(x.train[, 1])
table(x.train[, 2])
table(x.train[, 3])

x.test <- matrix(nrow=84, ncol=3) ## matrix of covariate scenarios
dimnames(x.test)[[2]] <- dimnames(x.train)[[2]]

i <- 1
for(age in 5*(9:15)) for(sex in 1:2) for(ph.karno in 10*(5:10)) {
  x.test[i, ] <- c(age, sex, ph.karno)
  i <- i+1
}

## this x.test is relatively small, but often you will want to
## predict for a large x.test matrix which may cause problems
## due to consumption of RAM so we can predict separately

## mcparallel/mccollect do not exist on windows
if(.Platform$OS.type=='unix') {
  ##test BART with token run to ensure installation works
  set.seed(99)
  post <- surv.bart(x.train=x.train, times=times, delta=delta, nskip=5, ndpost=5, keepevery=1)

  pre <- surv.pre.bart(x.train=x.train, times=times, delta=delta, x.test=x.test)

  pred <- mc.surv.pwbart(pre$tx.test, post$treedraws, post$binaryOffset)
}

## Not run:
## run one long MCMC chain in one process
set.seed(99)
post <- surv.bart(x.train=x.train, times=times, delta=delta)

## run "mc.cores" number of shorter MCMC chains in parallel processes
## post <- mc.surv.bart(x.train=x.train, times=times, delta=delta,
## mc.cores=8, seed=99)
pre <- surv.pre.bart(x.train=x.train, times=times, delta=delta, x.test=x.test)

pred <- surv.pwbart(pre$tx.test, post$treedraws, post$binaryOffset)

## let's look at some survival curves
## first, a younger group with a healthier KPS
## age 50 with KPS=90: males and females
## males: row 17, females: row 23
x.test[c(17, 23), ]

low.risk.males <- 16*post$K+1:post$K  ## K=unique times including censoring
low.risk.females <- 22*post$K+1:post$K

plot(post$times, pred$surv.test.mean[low.risk.males], type='s', col='blue',
     main='Age 50 with KPS=90', xlab='t', ylab='S(t)', ylim=c(0, 1))
points(post$times, pred$surv.test.mean[low.risk.females], type='s', col='red')

## End(Not run)

---

**mc.wbart**

**BART for continuous outcomes with parallel computation**

---

**Description**

BART is a Bayesian “sum-of-trees” model. For numeric response \( y \), we have \( y = f(x) + \epsilon \), where \( \epsilon \sim N(0, \sigma^2) \).

In both cases, \( f \) is the sum of many tree models. The goal is to have very flexible inference for the unknown 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**

```R
mc.wbart(
  x.train, y.train, x.test=matrix(0.0,0,0),
  sparse=FALSE, theta=0, omega=1,
  a=0.5, b=1, augment=FALSE, rho=NULL,
  xinfo=matrix(0.0,0,0), usequants=FALSE,
  cont=FALSE, rm.const=TRUE,
  sigest=NA, sigdf=3, sigquant=0.90,
  k=2.0, power=2.0, base=.95,
  sigmaf=NA, lambdaf=NA, fmean=mean(y.train),
  w=rep(1,length(y.train)),
  ntree=200L, numcut=100L,
  ndpost=1000L, nskip=100L,
  keepevery=1L, printevery=100,
  keeptrainfits=TRUE, transposed=FALSE,
  mc.cores = 2L, nice = 19L,
  seed = 99L
)
```
Arguments

\texttt{x.train} \hspace{1cm} 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. \texttt{wbart} will generate draws of f(x) for each x which is a row of \texttt{x.train}.

\texttt{y.train} \hspace{1cm} Dependent variable for training (in sample) data.
If y is numeric a continous response model is fit (normal errors).

\texttt{x.test} \hspace{1cm} Explanatory variables for test (out of sample) data.
Should have same structure as \texttt{x.train}.
\texttt{wbart} will generate draws of f(x) for each x which is a row of \texttt{x.test}.

\texttt{sparse} \hspace{1cm} Whether to perform variable selection based on a sparse Dirichlet prior rather than simply uniform; see Linero 2016.

\texttt{theta} \hspace{1cm} Set \texttt{theta} parameter; zero means random.

\texttt{omega} \hspace{1cm} Set \texttt{omega} parameter; zero means random.

\texttt{a} \hspace{1cm} Sparse parameter for Beta(a, b) prior: 0.5 \leq a \leq 1 where lower values inducing more sparsity.

\texttt{b} \hspace{1cm} Sparse parameter for Beta(a, b) prior; typically, b = 1.

\texttt{rho} \hspace{1cm} Sparse parameter: typically rho = p where p is the number of covariates under consideration.

\texttt{augment} \hspace{1cm} Whether data augmentation is to be performed in sparse variable selection.

\texttt{xinfo} \hspace{1cm} You can provide the cutpoints to BART or let BART choose them for you. To provide them, use the \texttt{xinfo} argument to specify a list (matrix) where the items (rows) are the covariates and the contents of the items (columns) are the cutpoints.

\texttt{usequants} \hspace{1cm} If \texttt{usequants=FALSE}, then the cutpoints in \texttt{xinfo} are generated uniformly; otherwise, if \texttt{TRUE}, uniform quantiles are used for the cutpoints.

\texttt{cont} \hspace{1cm} Whether or not to assume all variables are continuous.

\texttt{rm.const} \hspace{1cm} Whether or not to remove constant variables.

\texttt{sigest} \hspace{1cm} The prior for the error variance (\sigma^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 \texttt{sigest=NA} then the rough estimate will be the usual least squares estimator. Otherwise the supplied value will be used.

\texttt{sigdf} \hspace{1cm} Degrees of freedom for error variance prior.

\texttt{sigquant} \hspace{1cm} The quantile of the prior that the rough estimate (see \texttt{sigest}) is placed at. The closer the quantile is to 1, the more aggressive the fit will be as you are putting more prior weight on error standard deviations (\sigma) less than the rough estimate.
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.

The bigger $k$ is, the more conservative the fitting will be.

- **power**: Power parameter for tree prior.
- **base**: Base parameter for tree prior.
- **sigmaf**: The SD of $f$.
- **lambda**: The scale of the prior for the variance.
- **fmean**: BART operates on $y.train$ centered by fmean.
- **w**: Vector of weights which multiply the variance.
- **ntree**: The number of trees in the sum.
- **numcut**: The number of possible values of $c$ (see usequants). If a single number is 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.
- **ndpost**: The number of posterior draws returned.
- **nskip**: Number of MCMC iterations to be treated as burn in.
- **keepevery**: Every keepevery draw is kept to be returned to the user.
- **printevery**: As the MCMC runs, a message is printed every printevery draws.
- **keeptrainfits**: Whether to keep $yhat.train$ or not.
- **transposed**: When running wbart in parallel, it is more memory-efficient to transpose x.train and x.test, if any, prior to calling mc.wbart.
- **seed**: Setting the seed required for reproducible MCMC.
- **mc.cores**: Number of cores to employ in parallel.
- **nice**: Set the job niceness. The default niceness is 19: niceness goes from 0 (highest) to 19 (lowest).

**Details**

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

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 $\sigma^*$ 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

mc.wbart returns an object of type wbart which is essentially a list.

- `yhat.train`: A matrix with `ndpost` 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.
- `yhat.train.mean`: Train data fits = mean of `yhat.train` columns.
- `yhat.test.mean`: Test data fits = mean of `yhat.test` columns.
- `varcount`: A matrix with `ndpost` 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.

Author(s)

Robert McCulloch: <robert.e.mcculloch@gmail.com>,
Rodney Sparapani: <rsparapa@mcw.edu>.

References


See Also

wbart

Examples

```r
##simulate data (example from Friedman 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 BART later

## parallel::mcparallel/mccollect do not exist on windows
if(.Platform$OS.type=='unix') {
  ## test BART with token run to ensure installation works
  bartFit = mc.wbart(x, y, mc.cores=2, seed=99, nskip=5, ndpost=5)
}

## Not run:
## run BART
bartFit = mc.wbart(x, y, mc.cores=5, seed=99)
# compare BART fit to linear matter and truth = Ey
fitmat = cbind(y, Ey, lmFit$fitted, bartFit$yhat.train.mean)
colnames(fitmat) = c('y', 'Ey', 'lm', 'bart')
print(cor(fitmat))
## End(Not run)

---

mc.wbart.gse  

Global SE variable selection for BART with parallel computation

Description

Here we implement the global SE method for variable selection in nonparametric survival analysis with BART. Unfortunately, the method is very computationally intensive so we present some trade-offs below.

Usage

```r
mc.wbart.gse( x.train, y.train,
P=50L, R=5L, ntree=20L, numcut=100L, C=1, alpha=0.05,
k=2.0, power=2.0, base=0.95,
ndpost=2000L, nskip=100L,
printevery=100L, keepevery=1L, keeptrainfits=FALSE,
seed=99L, mc.cores=2L, nice=19L
)
```

Arguments

- **x.train**  
  Explanatory variables for training (in sample) data.
  Must be a matrix with (as usual) rows corresponding to observations and columns to variables.
  surv.bart will generate draws of $f(t, x)$ for each $x$ which is a row of x.train.

- **y.train**  
  The continuous outcome.

- **P**  
  The number of permutations: typically 50 or 100.

- **R**  
  The number of replicates: typically 5 or 10.

- **ntree**  
  The number of trees. In variable selection, the number of trees is smaller than what might be used for the best fit.
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.

C  The starting value for the multiple of SE. You should not need to change this except in rare circumstances.

alpha  The global SE method relies on simultaneous 1-alpha coverage across the permutations for all predictor variables.

k  k is the number of prior standard deviations $f(t, x)$ is away from +/-3. The bigger k is, the more conservative the fitting will be.

power  Power parameter for tree prior.

base  Base parameter for tree prior.

ndpost  The number of posterior draws after burn in. In the global SE method, generally, the method is repeated several times to establish the variable count probabilities. However, we take the alternative approach of simply running the MCMC chain longer which should result in the same stabilization of the estimates. Therefore, the number of posterior draws in variable selection should be set to a larger value than would be typically anticipated for fitting.

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

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

keepevery  Every keepevery draw is kept.

keeptrainfits  If TRUE the draws of $f(t, x)$ for $x =$ rows of x.train are generated.

seed  seed required for reproducible MCMC.

mc.cores  Number of cores to employ in parallel.

nice  Set the job priority. The default priority is 19: priorities go from 0 (highest) to 19 (lowest).

Value

mc.wbart.gse returns a list.

Author(s)

Rodney Sparapani: <rsparapa@mcw.edu>

References


See Also

mc.wbart
### Examples

```r
## Not run:
library(ElemStatLearn)
data(phoneme)
x.train <- matrix(NA, nrow=4509, ncol=257)
dimnames(x.train)[[2]] <- c(paste0('x.', 1:256), 'speaker')
x.train[, 257] <- as.numeric(phoneme$speaker)
for(j in 1:256) x.train[, j] <- as.numeric(phoneme[, paste0('x.', j)])
gse <- mc.wbart.gse(x.train, as.numeric(phoneme$g), mc.cores=5, seed=99)
## important variables
dimnames(x.train)[[2]][gse$which]
## End(Not run)
```

---

**pbart**  
*Probit BART for dichotomous outcomes with Normal latents*

### Description

BART is a Bayesian “sum-of-trees” model.

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 unknown 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

```r
pbart(  
x.train, y.train, x.test=matrix(0.0,0,0),  
sparse=FALSE, theta=0, omega=1,  
a=0.5, b=1, augment=FALSE, rho=NULL,  
xinfo=matrix(0.0,0,0), usequants=FALSE,  
cont=FALSE, rm.const=TRUE,  
k=2.0, power=2.0, base=.95,  
binaryOffset=NULL, 
```

\texttt{pbart(nmtree=50L, numcut=100L, ndpost=1000L, nskip=100L, keepevery=1L,}
\texttt{nkeeptrain=ndpost, nkeeptest=ndpost,}
\texttt{nkeeptreedraws=ndpost,}
\texttt{printevery=100L, transposed=FALSE)}

**Arguments**

- **\texttt{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. \texttt{pbart} will generate draws of \( f(x) \) for each \( x \) which is a row of \( x.\text{train} \).

- **\texttt{y.train}**: Binary dependent variable for training (in sample) data.

- **\texttt{x.test}**: Explanatory variables for test (out of sample) data. Should have same structure as \( x.\text{train} \). \texttt{pbart} will generate draws of \( f(x) \) for each \( x \) which is a row of \( x.\text{test} \).

- **\texttt{sparse}**: Whether to perform variable selection based on a sparse Dirichlet prior rather than simply uniform; see Linero 2016.

- **\texttt{theta}**: Set \( \theta \) parameter; zero means random.

- **\texttt{omega}**: Set \( \omega \) parameter; zero means random.

- **\texttt{a}**: Sparse parameter for \( \text{Beta}(a, b) \) prior: \( 0.5 \leq a \leq 1 \) where lower values inducing more sparsity.

- **\texttt{b}**: Sparse parameter for \( \text{Beta}(a, b) \) prior; typically, \( b = 1 \).

- **\texttt{rho}**: Sparse parameter: typically \( \rho = p \) where \( p \) is the number of covariates under consideration.

- **\texttt{augment}**: Whether data augmentation is to be performed in sparse variable selection.

- **\texttt{xinfo}**: You can provide the cutpoints to BART or let BART choose them for you. To provide them, use the \texttt{xinfo} argument to specify a list (matrix) where the items (rows) are the covariates and the contents of the items (columns) are the cutpoints.

- **\texttt{usequants}**: If \texttt{usequants=FALSE}, then the cutpoints in \texttt{xinfo} are generated uniformly; otherwise, if \texttt{TRUE}, uniform quantiles are used for the cutpoints.

- **\texttt{cont}**: Whether or not to assume all variables are continuous.

- **\texttt{rm.const}**: Whether or not to remove constant variables.

- **\texttt{k}**: For binary \( y \), \( k \) is the number of prior standard deviations \( f(x) \) is away from +/-3. The bigger \( k \) is, the more conservative the fitting will be.

- **\texttt{power}**: Power parameter for tree prior.
base  Base parameter for tree prior.
binaryOffset  Used for binary $y$.
The model is $P(Y = 1|x) = F(f(x) + \text{binaryOffset})$.

ntree  The number of trees in the sum.
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.

ndpost  The number of posterior draws returned.
nskip  Number of MCMC iterations to be treated as burn in.
nkeeptrain  Number of MCMC iterations to be returned for train data.
nkeeptest  Number of MCMC iterations to be returned for test data.
nkeeptree  Number of MCMC iterations to be returned for tree draws.
keepevery  Every keeperevery draw is kept to be returned to the user.
printevery  As the MCMC runs, a message is printed every printevery draws.
transposed  When running pbart in parallel, it is more memory-efficient to transpose x.train and x.test, if any, prior to calling mc.pbart.

Details

BART is an Bayesian MCMC method. At each MCMC iteration, we produce a draw from $f$ 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 $f^*(x)$ where * denotes a particular draw. The $x$ is either a row from the training data (x.train) or the test data (x.test).

Value

pbart returns an object of type pbart which is essentially a list.

yhat.train  A matrix with ndpost 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.
varcount  a matrix with ndpost 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.
In addition the list has a binaryOffset component giving the value used. Note that in the binary \( y \), case yhat.train and yhat.test are \( f(x) + \text{binaryOffset}. \) If you want draws of the probability \( P(Y = 1| x) \) you need to apply the Normal CDF (\texttt{pnorm}) to these values.

**Author(s)**

Robert McCulloch: <robert.e.mcculloch@gmail.com>, Rodney Sparapani: <rsparapa@mcw.edu>.

**References**


**See Also**

\texttt{wbart}

**Examples**

```r
data(ACTG175)

## exclude those who do not have CD4 count at 96 weeks
ex <- is.na(ACTG175$cd496)
table(ex)

## inclusion criteria are CD4 counts between 200 and 500
ACTG175$cd40 <- min(500, max(250, ACTG175$cd40))

## calculate relative CD4 decline
y <- ((ACTG175$cd496-ACTG175$cd40)/ACTG175$cd40)[!ex]
summary(y)

## 0=failure, 1=success
y <- 1*(y > -0.5)

## summarize CD4 outcomes
table(y, ACTG175$arms[!ex])

table(y, ACTG175$arms[!ex])/
  matrix(table(ACTG175$arms[!ex]), nrow=2, ncol=4, byrow=TRUE)
```
## drop unneeded and unwanted variables
## 1: 'pidnum' patient ID number
## 14: 'str2' which will be handled by strat1 below
## 15: 'strat' which will be handled by strat1-strat3 below
## 17: 'treat' handled by arm0-arm3 below
## 18: 'offtrt' indicator of off-treatment before 96 weeks
## 20: 'cd420' CD4 T cell count at 20 weeks
## 21: 'cd496' CD4 T cell count at 96 weeks
## 22: 'r' missing CD4 T cell count at 96 weeks
## 24: 'cd820' CD8 T cell count at 20 weeks
## 25: 'cens' indicator of observing the event in days
## 26: 'days' number of days until the primary endpoint
## 27: 'arms' handled by arm0-arm3 below

```
train <- as.matrix(ACTG175)[!ex, -c(1, 14:15, 17, 18, 20:22, 24:27)]
train <- cbind(1*(ACTG175$strat[!ex]==1), 1*(ACTG175$strat[!ex]==2),
              1*(ACTG175$strat[!ex]==3), train)
dimnames(train)[[2]][1:3] <- paste0('strat', 1:3)
train <- cbind(1*(ACTG175$arms[!ex]==0), 1*(ACTG175$arms[!ex]==1),
              1*(ACTG175$arms[!ex]==2), 1*(ACTG175$arms[!ex]==3), train)
dimnames(train)[[2]][1:4] <- paste0('arm', 0:3)
```

N <- nrow(train)

```
test0 <- train; test0[, 1:4] <- 0; test0[, 1] <- 1
test1 <- train; test1[, 1:4] <- 0; test1[, 2] <- 1
test2 <- train; test2[, 1:4] <- 0; test2[, 3] <- 1
test3 <- train; test3[, 1:4] <- 0; test3[, 4] <- 1

test <- rbind(test0, test1, test2, test3)
```

## test BART with token run to ensure installation works

```
set.seed(21)
post <- pbart(train, y, test, nskip=5, ndpost=5)
```

## Not run:

```
set.seed(21)
post <- pbart(train, y, test)
```

## turn z-scores into probabilities

```
post$prob.test <- pnorm(post$yhat.test)
```

## average over the posterior samples

```
post$prob.test.mean <- apply(post$prob.test, 2, mean)
```

## place estimates for arms 0-3 next to each other for convenience

```
itr <- cbind(post$prob.test.mean[1:N], post$prob.test.mean[N+(1:N)],
             post$prob.test.mean[2*N+(1:N)], post$prob.test.mean[3*N+(1:N)])
```

## find the BART ITR for each patient

```
itr.pick <- integer(N)
for(i in 1:N) itr.pick[i] <- which(itr[i, ]==max(itr[i, ]))-1
```

## arms 0 and 3 (monotherapy) are never chosen
table(itr.pick)

## do arms 1 and 2 show treatment heterogeneity?
diff. <- apply(post$prob.test[, 2*N+(1:N)]-post$prob.test[, N+(1:N)], 2, mean)
plot(sort(diff.), type='h', main='ACTG175 trial: 50% CD4 decline from baseline at 96 weeks',
     xlab='Arm 2 (1) Preferable to the Right (Left)', ylab='Prob.Diff.: Arms 2 - 1')

library(rpart)
library(rpart.plot)

## make data frame for nicer names in the plot
var <- as.data.frame(train[, -(1:4)])

dss <- rpart(diff. ~ var$age+var$gender+var$race+var$wtkg+var$cd40+var$cd80+
             var$karnof+var$symptom+var$hemo+var$homo+var$drugs+var$z30+
             var$zpior+var$oprior+var$strat1+var$strat2+var$strat3,
             method='anova', control=rpart.control(cp=0.1))
rpart.plot(dss, type=3, extra=101)

## if strat1==1 (antiretroviral naive), then arm 2 is better
## otherwise, arm 1
print(dss)

all0 <- apply(post$prob.test[, (1:N)], 1, mean)
all1 <- apply(post$prob.test[, N+(1:N)], 1, mean)
all2 <- apply(post$prob.test[, 2*N+(1:N)], 1, mean)
all3 <- apply(post$prob.test[, 3*N+(1:N)], 1, mean)

## BART ITR
BART.itr <- apply(post$prob.test[, c(N+which(itr.pick==1), 2*N+which(itr.pick==2))], 1, mean)

test <- train
test[, 1:4] <- 0
test[test[, 5]==0, 2] <- 1
test[test[, 5]==1, 3] <- 1

## BART ITR simple
BART.itr.simp <- pwbart(test, post$treedraws)
BART.itr.simp <- apply(pnorm(BART.itr.simp), 1, mean)

plot(density(BART.itr), xlab='Value', xlim=c(0.475, 0.775), lwd=2,
     main='ACTG175 trial: 50% CD4 decline from baseline at 96 weeks')
lines(density(BART.itr.simp), col='brown', lwd=2)
lines(density(all0), col='green', lwd=2)
lines(density(all1), col='red', lwd=2)
lines(density(all2), col='blue', lwd=2)
lines(density(all3), col='yellow', lwd=2)
legend('topleft', legend=c('All Arm 0 (ZDV only)', 'All Arm 1 (ZDV+DDI)', 'All Arm 2 (ZDV+DDC)', 'All Arm 3 (DDI only)',
                          'BART ITR simple', 'BART ITR'),
       col=c('green', 'red', 'blue', 'yellow', 'brown', 'black'), lty=1, lwd=2)
predict.crisk2bart  
Predicting new observations with a previously fitted BART model

Description

BART is a Bayesian “sum-of-trees” model.
For a numeric response $y$, we have $y = f(x) + \epsilon$, where $\epsilon \sim N(0, \sigma^2)$.

$f$ is the sum of many tree models. The goal is to have very flexible inference for the unknown 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

## S3 method for class 'crisk2bart'
predict(object, newdata, newdata2, mc.cores=1, openmp=(mc.cores.openmp()>0), ...)

Arguments

object          object returned from previous BART fit with crisk2.bart or mc.crisk2.bart.
newdata         Matrix of covariates to predict the distribution of $t1$.
newdata2        Matrix of covariates to predict the distribution of $t2$.
cmp.cores      Number of threads to utilize.
openmp          Logical value dictating whether OpenMP is utilized for parallel processing. Of course, this depends on whether OpenMP is available on your system which, by default, is verified with mc.cores.openmp.
...             Other arguments which will be passed on to pwbart.

Details

BART is a Bayesian MCMC method. At each MCMC interaction, 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 $\sigma^*$ 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

Returns an object of type crisk2bart with predictions corresponding to newdata and newdata2.
Author(s)

Robert McCulloch: <robert.e.mcculloch@gmail.com>,
Rodney Sparapani: <rsparapa@mcw.edu>.

References


See Also

crisk2.bart, mc.crisk2.bart, mc.crisk2.pwbart, mc.cores.openmp

Examples

data(transplant)

delta <- (as.numeric(transplant$event)-1)
## recode so that delta=1 is cause of interest; delta=2 otherwise
delta[delta==1] <- 4
delta[delta==2] <- 1
delta[delta>1] <- 2
table(delta, transplant$event)

times <- pmax(1, ceiling(transplant$futime/7)) ## weeks
##times <- pmax(1, ceiling(transplant$futime/30.5)) ## months
table(times)

typeO <- 1*(transplant$abo=='O')
typeA <- 1*(transplant$abo=='A')
typeB <- 1*(transplant$abo=='B')
typeAB <- 1*(transplant$abo=='AB')
table(typeA, typeO)

x.train <- cbind(typeO, typeA, typeB, typeAB)

x.test <- cbind(1, 0, 0, 0)
dimnames(x.test)[[2]] <- dimnames(x.train)[[2]]

## parallel::mcparallel/mccollect do not exist on windows
if(.Platform$OS.type=='unix') {
##test BART with token run to ensure installation works
post <- mc.crisk2.bart(x.train=x.train, times=times, delta=delta,
seed=99, mc.cores=2, nskip=5, ndpost=5,
keepevery=1)
predict.criskbart

predict.criskbart

Predicting new observations with a previously fitted BART model

pre <- surv.pre.bart(x.train=x.train, x.test=x.test,
          times=times, delta=delta)

K <- post$K

pred <- mc.crisk2.pwbart(pre$tx.test, pre$tx.test,
                post$treedraws, post$treedraws2,
                post$binaryOffset, post$binaryOffset2)

## Not run:

## run one long MCMC chain in one process
## set.seed(99)
## post <- crisk2.bart(x.train=x.train, times=times, delta=delta, x.test=x.test)

## in the interest of time, consider speeding it up by parallel processing
## run "mc.cores" number of shorter MCMC chains in parallel processes
post <- mc.crisk2.bart(x.train=x.train,
          times=times, delta=delta,
          x.test=x.test, seed=99, mc.cores=8)

## check <- mc.crisk2.pwbart(post$tx.test, post$tx.test,
## post$treedraws, post$treedraws2,
## post$binaryOffset,
## post$binaryOffset2, mc.cores=8)
check <- predict(post, newdata=post$tx.test, newdata2=post$tx.test2,
     mc.cores=8)

print(c(post$surv.test.mean[1], check$surv.test.mean[1],
        post$surv.test.mean[1]-check$surv.test.mean[1]), digits=22)
print(all(round(post$surv.test.mean, digits=9)==
         round(check$surv.test.mean, digits=9)))

print(c(post$cif.test.mean[1], check$cif.test.mean[1],
        post$cif.test.mean[1]-check$cif.test.mean[1]), digits=22)
print(all(round(post$cif.test.mean, digits=9)==
         round(check$cif.test.mean, digits=9)))

print(c(post$cif.test2.mean[1], check$cif.test2.mean[1],
        post$cif.test2.mean[1]-check$cif.test2.mean[1]), digits=22)
print(all(round(post$cif.test2.mean, digits=9)==
         round(check$cif.test2.mean, digits=9)))

## End(Not run)
**predict.criskbart**

### Description

BART is a Bayesian “sum-of-trees” model.

For a numeric response $y$, we have $y = f(x) + \epsilon$, where $\epsilon \sim N(0, \sigma^2)$.

$f$ is the sum of many tree models. The goal is to have very flexible inference for the unknown 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

```r
## S3 method for class 'criskbart'
predict(object, newdata, newdata2, mc.cores=1, openmp=(mc.cores.openmp()>0), ...)
```

### Arguments

- **object**: object returned from previous BART fit with `crisk.bart` or `mc.crisk.bart`.
- **newdata**: Matrix of covariates to predict the distribution of $t1$.
- **newdata2**: Matrix of covariates to predict the distribution of $t2$.
- **mc.cores**: Number of threads to utilize.
- **openmp**: Logical value dictating whether OpenMP is utilized for parallel processing. Of course, this depends on whether OpenMP is available on your system which, by default, is verified with `mc.cores.openmp`.
- **...**: Other arguments which will be passed on to `pwbart`.

### Details

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

Thus, unlike a lot of other 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 $\sigma^*$ 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

Returns an object of type `criskbart` with predictions corresponding to `newdata` and `newdata2`.

### Author(s)

Robert McCulloch: `<robert.e.mcculloch@gmail.com>`,
Rodney Sparapani: `<rsparapa@mcw.edu>`.
References


See Also

crisk.bart, mc.crisk.bart, mc.crisk.pwbart, mc.cores.openmp

Examples

data(transplant)

delta <- (as.numeric(transplant$event)-1)
## recode so that delta=1 is cause of interest; delta=2 otherwise
delta[delta==1] <- 4
delta[delta==2] <- 1
delta[delta>1] <- 2
table(delta, transplant$event)

times <- pmax(1, ceiling(transplant$futime/7)) ## weeks
##times <- pmax(1, ceiling(transplant$futime/30.5)) ## months
table(times)

typeO <- 1*(transplant$abo=='O')
typeA <- 1*(transplant$abo=='A')
typeB <- 1*(transplant$abo=='B')
typeAB <- 1*(transplant$abo=='AB')
table(typeA, typeO)

x.train <- cbind(typeO, typeA, typeB, typeAB)

x.test <- cbind(1, 0, 0, 0)
dimnames(x.test)[[2]] <- dimnames(x.train)[[2]]

## parallel::mcparallel/mccollect do not exist on windows
if(.Platform$OS.type=='unix') {
##test BART with token run to ensure installation works
post <- mc.crisk.bart(x.train=x.train, times=times, delta=delta,
    seed=99, mc.cores=2, nskip=5, ndpost=5,
    keepevery=1)

pre <- surv.pre.bart(x.train=x.train, x.test=x.test,
    times=times, delta=delta)

K <- post$K
predict.lbart

Predicting new observations with a previously fitted BART model

```r
pred <- mc.crisk.pwbart(pre$tx.test, pre$tx.test,
                         post$treedraws, post$treedraws2,
                         post$binaryOffset, post$binaryOffset2)
}

## Not run:
## run one long MCMC chain in one process
## set.seed(99)
## post <- crisk.bart(x.train=x.train, times=times, delta=delta, x.test=x.test)

## in the interest of time, consider speeding it up by parallel processing
## run "mc.cores" number of shorter MCMC chains in parallel processes
post <- mc.crisk.bart(x.train=x.train,
                      times=times, delta=delta,
                      x.test=x.test, seed=99, mc.cores=8)

## check <- mc.crisk.pwbart(post$tx.test, post$tx.test,
## post$treedraws, post$treedraws2,
## post$binaryOffset,
## post$binaryOffset2, mc.cores=8)
check <- predict(post, newdata=post$tx.test, newdata2=post$tx.test2,
                 mc.cores=8)

print(c(post$surv.test.mean[1], check$surv.test.mean[1],
        post$surv.test.mean[1]-check$surv.test.mean[1]), digits=22)
print(all(round(post$surv.test.mean, digits=9)==
        round(check$surv.test.mean, digits=9)))

print(c(post$cif.test.mean[1], check$cif.test.mean[1],
        post$cif.test.mean[1]-check$cif.test.mean[1]), digits=22)
print(all(round(post$cif.test.mean, digits=9)==
        round(check$cif.test.mean, digits=9)))

print(c(post$cif.test2.mean[1], check$cif.test2.mean[1],
        post$cif.test2.mean[1]-check$cif.test2.mean[1]), digits=22)
print(all(round(post$cif.test2.mean, digits=9)==
        round(check$cif.test2.mean, digits=9)))

## End(Not run)
```
Description

BART is a Bayesian “sum-of-trees” model. For a numeric response \( y \), we have \( y = f(x) + \epsilon \), where \( \epsilon \sim N(0, \sigma^2) \).

\( f \) is the sum of many tree models. The goal is to have very flexible inference for the unknown 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

```r
## S3 method for class 'lbart'
predict(object, newdata, mc.cores=1, openmp=(mc.cores.openmp()>0), ...)
```

Arguments

- `object`: object returned from previous BART fit with `surv.bart` or `mc.surv.bart`.
- `newdata`: Matrix of covariates to predict the distribution of \( t \).
- `mc.cores`: Number of threads to utilize.
- `openmp`: Logical value dictating whether OpenMP is utilized for parallel processing. Of course, this depends on whether OpenMP is available on your system which, by default, is verified with `mc.cores.openmp`.
- `...`: Other arguments which will be passed on to `pwbart`.

Details

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

Thus, unlike a lot of other 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 \( \sigma^* \) 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

Returns an object of type `lbart` with predictions corresponding to `newdata`.

Author(s)

Robert McCulloch: `<robert.e.mcculloch@gmail.com>`.
Rodney Sparapani: `<rsparapa@mcw.edu>`.
References


See Also

surv.bart, mc.surv.bart, surv.pwbart, mc.surv.pwbart, mc.cores.openmp

Examples

```r
## load the advanced lung cancer example
data(lung)

group <- -which(is.na(lung[,7]))  ## remove missing row for ph.karno
times <- lung[group, 2]  ## lung$time
delta <- lung[group, 3]-1  ## lung$status: 1=censored, 2=dead
## delta: 0=censored, 1=dead

## this study reports time in days rather than months like other studies
## coarsening from days to months will reduce the computational burden
																																																																												
times <- ceiling(times/30)

summary(times)
table(delta)

x.train <- as.matrix(lung[group, c(4, 5, 7)])  ## matrix of observed covariates

## lung$age: Age in years
## lung$sex: Male=1 Female=2
## lung$ph.karno: Karnofsky performance score (dead=0:normal=100:by=10)
## rated by physician
dimnames(x.train)[[2]] <- c('age(yr)', 'M(1):F(2)', 'ph.karno(0:100:10)')

summary(x.train[, 1])
table(x.train[, 2])
table(x.train[, 3])

x.test <- matrix(nrow=84, ncol=3)  ## matrix of covariate scenarios
dimnames(x.test)[[2]] <- dimnames(x.train)[[2]]

i <- 1
for(age in 5*(9:15)) for(sex in 1:2) for(ph.karno in 10*(5:10)) {
  x.test[i, ] <- c(age, sex, ph.karno)
  i <- i + 1
}
```
i <- i+1
}

## this x.test is relatively small, but often you will want to
## predict for a large x.test matrix which may cause problems
## due to consumption of RAM so we can predict separately

## mcparallel/mccollect do not exist on windows
if(.Platform$OS.type==quote(unix)) {
  ## test BART with token run to ensure installation works
  set.seed(99)
  post <- surv.bart(x.train=x.train, times=times, delta=delta, nskip=5, ndpost=5, keepevery=1)
  pre <- surv.pre.bart(x.train=x.train, times=times, delta=delta, x.test=x.test)
  pred <- predict(post, pre$tx.test)
  ## pred. <- surv.pwbart(pre$tx.test, post$treedraws, post$binaryOffset)
}

## Not run:
## run one long MCMC chain in one process
set.seed(99)
post <- surv.bart(x.train=x.train, times=times, delta=delta)

## run "mc.cores" number of shorter MCMC chains in parallel processes
## post <- mc.surv.bart(x.train=x.train, times=times, delta=delta,
## # post <- mc.surv.bart(x.train=x.train, times=times, delta=delta,
## # mc.cores=5, seed=99)
pre <- surv.pre.bart(x.train=x.train, times=times, delta=delta, x.test=x.test)

pred <- predict(post, pre$tx.test)

## let's look at some survival curves
## first, a younger group with a healthier KPS
## age 50 with KPS=90: males and females
## males: row 17, females: row 23
x.test[c(17, 23), ]

low.risk.males <- 16*post$K+1:post$K ## K=unique times including censoring
low.risk.females <- 22*post$K+1:post$K

plot(post$times, pred$surv.test.mean[low.risk.males], type='s', col='blue',
     main='Age 50 with KPS=90', xlab='t', ylab='S(t)', ylim=c(0, 1))
points(post$times, pred$surv.test.mean[low.risk.females], type='s', col='red')

## End(Not run)
Description

BART is a Bayesian “sum-of-trees” model. For a numeric response \( y \), we have \( y = f(x) + \epsilon \), where \( \epsilon \sim N(0, \sigma^2) \).

\( f \) is the sum of many tree models. The goal is to have very flexible inference for the unknown 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

```r
## S3 method for class 'mbart'
predict(object, newdata, mc.cores=1, openmp=(mc.cores.openmp()>0), ...)
## S3 method for class 'mbart2'
predict(object, newdata, mc.cores=1, openmp=(mc.cores.openmp()>0), ...)
```

Arguments

- `object` object returned from previous BART fit with `mbart` or `mbart2`.
- `newdata` Matrix of covariates to predict the distribution of \( t \).
- `mc.cores` Number of threads to utilize.
- `openmp` Logical value dictating whether OpenMP is utilized for parallel processing. Of course, this depends on whether OpenMP is available on your system which, by default, is verified with `mc.cores.openmp`.
- `...` Other arguments which will be passed on to `pwbart`.

Details

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

Thus, unlike a lot of other 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 \( \sigma^* \) 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

Returns an object of type `mbart` with predictions corresponding to `newdata`.

Author(s)

Robert McCulloch: <robert.e.mcculloch@gmail.com>,
Rodney Sparapani: <rsparapa@mcw.edu>. 
References


See Also

`mbart, mbart2`

Examples

```r
## load the advanced lung cancer example
data(lung)

group <- which(is.na(lung[, 7]))  ## remove missing row for ph.karno
times <- lung[group, 2]          ## lung$time
delta <- lung[group, 3]-1       ## lung$status: 1=censored, 2=dead
## delta: 0=censored, 1=dead
## this study reports time in days rather than months like other studies
## coarsening from days to months will reduce the computational burden
times <- ceiling(times/30)

summary(times)
table(delta)

x.train <- as.matrix(lung[group, c(4, 5, 7)])  ## matrix of observed covariates

## lung$age: Age in years
## lung$sex: Male=1 Female=2
## lung$ph.karno: Karnofsky performance score (dead=0: normal=100: by=10)
## rated by physician
dimnames(x.train)[[2]] <- c('age(yr)', 'M(1):F(2)', 'ph.karno(0:100:10)')

summary(x.train[, 1])
table(x.train[, 2])
table(x.train[, 3])

x.test <- matrix(nrow=84, ncol=3)  ## matrix of covariate scenarios
dimnames(x.test)[[2]] <- dimnames(x.train)[[2]]

i <- 1
for(age in 5*(9:15)) for(sex in 1:2) for(ph.karno in 10*(5:10)) {
  x.test[i, ] <- c(age, sex, ph.karno)
  i <- i + 1
}
```
predict.pbart

i <- i+1
}

## this x.test is relatively small, but often you will want to
## predict for a large x.test matrix which may cause problems
## due to consumption of RAM so we can predict separately

## mcparallel/mccollect do not exist on windows
if(.Platform$OS.type=='unix') {
## test BART with token run to ensure installation works
set.seed(99)
    post <- surv.bart(x.train=x.train, times=times, delta=delta, nskip=5, ndpost=5, keepevery=1)
    pre <- surv.pre.bart(x.train=x.train, times=times, delta=delta, x.test=x.test)
    pred <- predict(post, pre$tx.test)
    ## pred. <- surv.pwbart(pre$tx.test, post$treedraws, post$binaryOffset)
}

## Not run:
## run one long MCMC chain in one process
set.seed(99)
    post <- surv.bart(x.train=x.train, times=times, delta=delta)

## run "mc.cores" number of shorter MCMC chains in parallel processes
## post <- mc.surv.bart(x.train=x.train, times=times, delta=delta,
##                     mc.cores=5, seed=99)
    pre <- surv.pre.bart(x.train=x.train, times=times, delta=delta, x.test=x.test)
    pred <- predict(post, pre$tx.test)

## let's look at some survival curves
## first, a younger group with a healthier KPS
## age 50 with KPS=90: males and females
## males: row 17, females: row 23
x.test[c(17, 23), ]

low.risk.males <- 16*post$K+1:post$K  ## K=unique times including censoring
low.risk.females <- 22*post$K+1:post$K

plot(post$times, pred$surv.test.mean[low.risk.males], type='s', col='blue',
    main='Age 50 with KPS=90', xlab='t', ylab='S(t)', ylim=c(0, 1))
points(post$times, pred$surv.test.mean[low.risk.females], type='s', col='red')

## End(Not run)
Description

BART is a Bayesian “sum-of-trees” model. For a numeric response $y$, we have $y = f(x) + \epsilon$, where $\epsilon \sim N(0,\sigma^2)$.

$f$ is the sum of many tree models. The goal is to have very flexible inference for the unknown 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

```r
## S3 method for class 'pbart'
predict(object, newdata, mc.cores=1, openmp=(mc.cores.openmp()>0), ...)
```

Arguments

- `object` object returned from previous BART fit with `surv.bart` or `mc.surv.bart`.
- `newdata` Matrix of covariates to predict the distribution of $t$.
- `mc.cores` Number of threads to utilize.
- `openmp` Logical value dictating whether OpenMP is utilized for parallel processing. Of course, this depends on whether OpenMP is available on your system which, by default, is verified with `mc.cores.openmp`.
- `...` Other arguments which will be passed on to `pwbart`.

Details

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

Thus, unlike a lot of other 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 $\sigma^*$ 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

Returns an object of type `pbart` with predictions corresponding to `newdata`.

Author(s)

Robert McCulloch: <robert.e.mcculloch@gmail.com>,
Rodney Sparapani: <rparapa@mcw.edu>.
References


See Also

surv.bart, mc.surv.bart, surv.pwbart, mc.surv.pwbart, mc.cores.openmp

Examples

## load the advanced lung cancer example
data(lung)

group <- -which(is.na(lung[, 7])) ## remove missing row for ph.karno
times <- lung[group, 2]       ##lung$time
delta <- lung[group, 3]-1     ##lung$status: 1=censored, 2=dead
    ##delta: 0=censored, 1=dead

## this study reports time in days rather than months like other studies
## coarsening from days to months will reduce the computational burden
times <- ceiling(times/30)

summary(times)
table(delta)

x.train <- as.matrix(lung[group, c(4, 5, 7)]) ## matrix of observed covariates

# lung$age: Age in years
# lung$sex: Male=1 Female=2
# lung$ph.karno: Karnofsky performance score (dead=0:normal=100:by=10)
# rated by physician
dimnames(x.train)[[2]] <- c('age(yr)', 'M(1):F(2)', 'ph.karno(0:100:10)')

summary(x.train[, 1])
table(x.train[, 2])
table(x.train[, 3])

x.test <- matrix(nrow=84, ncol=3) ## matrix of covariate scenarios
dimnames(x.test)[[2]] <- dimnames(x.train)[[2]]

i <- 1

for(age in 5*(9:15)) for(sex in 1:2) for(ph.karno in 10*(5:10)) {
  x.test[i, ] <- c(age, sex, ph.karno)
i <- i+1
}

## this x.test is relatively small, but often you will want to
## predict for a large x.test matrix which may cause problems
## due to consumption of RAM so we can predict separately

## mcparallel/mccollect do not exist on windows
if(.Platform$OS.type=='unix') {
  ## test BART with token run to ensure installation works
  set.seed(99)
  post <- surv.bart(x.train=x.train, times=times, delta=delta, nskip=5, ndpost=5, keepevery=1)
  pre <- surv.pre.bart(x.train=x.train, times=times, delta=delta, x.test=x.test)
  pred <- predict(post, pre$tx.test)
  ##pred. <- surv.pwbart(pre$tx.test, post$treedraws, post$binaryOffset)
}

## Not run:
## run one long MCMC chain in one process
set.seed(99)
post <- surv.bart(x.train=x.train, times=times, delta=delta)

## run "mc.cores" number of shorter MCMC chains in parallel processes
## post <- mc.surv.bart(x.train=x.train, times=times, delta=delta,
##          mc.cores=5, seed=99)

pre <- surv.pre.bart(x.train=x.train, times=times, delta=delta, x.test=x.test)
pred <- predict(post, pre$tx.test)

## let's look at some survival curves
## first, a younger group with a healthier KPS
## age 50 with KPS=90: males and females
## males: row 17, females: row 23
x.test[c(17, 23), ]

low.risk.males <- 16*post$K+1:post$K ## K=unique times including censoring
low.risk.females <- 22*post$K+1:post$K

plot(post$times, pred$surv.test.mean[low.risk.males], type='s', col='blue',
     main='Age 50 with KPS=90', xlab='t', ylab='S(t)', ylim=c(0, 1))
ponts(post$times, pred$surv.test.mean[low.risk.females], type='s', col='red')

## End(Not run)
BART is a Bayesian “sum-of-trees” model. For a numeric response \( y \), we have \( y = f(x) + \epsilon \), where \( \epsilon \sim N(0, \sigma^2) \).

\( f \) is the sum of many tree models. The goal is to have very flexible inference for the unknown 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**

```r
## S3 method for class 'recurbart'
predict(object, newdata, mc.cores=1, openmp=(mc.cores.openmp()>0), ...)
```

**Arguments**

- `object` object returned from previous BART fit with `recur.bart` or `mc.recur.bart`.
- `newdata` Matrix of covariates to predict the distribution of \( t \).
- `mc.cores` Number of threads to utilize.
- `openmp` Logical value dictating whether OpenMP is utilized for parallel processing. Of course, this depends on whether OpenMP is available on your system which, by default, is verified with `mc.cores.openmp`.
- `...` Other arguments which will be passed on to `pwbart`.

**Details**

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

Thus, unlike a lot of other 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 \( f^\ast(x) \) (and \( \sigma^\ast \) 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**

Returns an object of type `recurbart` with predictions corresponding to `newdata`.

**Author(s)**

Robert McCulloch: <robert.e.mcculloch@gmail.com>,
Rodney Sparapani: <rsparapa@mcw.edu>.
References


See Also

recur.bart, mc.recur.bart, recur.pwbart, mc.recur.pwbart, mc.cores.openmp

Examples

```r
## load 20 percent random sample
data(xdm20.train)
data(xdm20.test)
data(ydm20.train)

## test BART with token run to ensure installation works
## with current technology even a token run will violate CRAN policy
## set.seed(99)
## post <- recur.bart(x.train=xdm20.train, y.train=ydm20.train,
## nskip=1, ndpost=1, keepevery=1)

## Not run:
set.seed(99)
post <- recur.bart(x.train=xdm20.train, y.train=ydm20.train)

## larger data sets can take some time so, if parallel processing
## is available, submit this statement instead
## post <- mc.recur.bart(x.train=xdm20.train, y.train=ydm20.train,
## mc.cores=8, seed=99)

require(rpart)
require(rpart.plot)
dss <- rpart(post$yhat.train.mean~xdm20.train)

rpart.plot(dss)

## for the 20 percent sample, notice that the top splits
## involve cci_pvd and n
## for the full data set, notice that all splits
## involve ca, cci_pud, cci_pvd, ins270 and n
## (except one at the bottom involving a small group)

## compare patients treated with insulin (ins270=1) vs
## not treated with insulin (ins270=0)
N.train <- 50
N.test <- 50
K <- post$K ## 798 unique time points
```
## only testing set, i.e., remove training set
xdm20.test. <- xdm20.test[N.train*K+(1:(N.test*K)), ]
xdm20.test. <- rbind(xdm20.test., xdm20.test.)
xdm20.test.[ , 'ins270'] <- rep(0:1, each=N.test*K)

## multiple threads will be utilized if available
pred <- predict(post, xdm20.test., mc.cores=8)

## create Friedman’s partial dependence function for the
## intensity/hazard by time and ins270
NK.test <- N.test*K
M <- nrow(pred$haz.test) ## number of MCMC samples, typically 1000
RI <- matrix(0, M, K)
for(i in 1:N.test)
  RI <- RI+(pred$haz.test[ , (N.test+i-1)*K+1:K]/pred$haz.test[ , (i-1)*K+1:K])/N.test
RI.lo <- apply(RI, 2, quantile, probs=0.025)
RI.mu <- apply(RI, 2, mean)
RI.hi <- apply(RI, 2, quantile, probs=0.975)
plot(post$times, RI.hi, type='l', lty=2, log='y',
     ylim=c(min(RI.lo, 1/RI.hi), max(1/RI.lo, RI.hi)),
     xlab='t', ylab='RI(t, x)',
     sub='insulin(ins270=1) vs. no insulin(ins270=0)',
     main='Relative intensity of hospital admissions for diabetics')
lines(post$times, RI.mu)
lines(post$times, RI.lo, lty=2)
lines(post$times, rep(1, K), col='darkgray')

## RI for insulin therapy seems fairly constant with time
mean(RI.mu)

## End(Not run)

---

**predict.survbart**

*Predicting new observations with a previously fitted BART model*

**Description**

BART is a Bayesian “sum-of-trees” model. For a numeric response $y$, we have $y = f(x) + \epsilon$, where $\epsilon \sim N(0, \sigma^2)$.

$f$ is the sum of many tree models. The goal is to have very flexible inference for the unknown 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.
## predict.survbart

### Usage

```r
## S3 method for class 'survbart'
predict(object, newdata, mc.cores=1, openmp=(mc.cores.openmp()>0), ...)
```

### Arguments

- **object**: object returned from previous BART fit with `surv.bart` or `mc.surv.bart`.
- **newdata**: Matrix of covariates to predict the distribution of \( t \).
- **mc.cores**: Number of threads to utilize.
- **openmp**: Logical value dictating whether OpenMP is utilized for parallel processing. Of course, this depends on whether OpenMP is available on your system which, by default, is verified with `mc.cores.openmp`.
- **...**: Other arguments which will be passed on to `pwbart`.

### Details

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

Thus, unlike a lot of other 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 \( \sigma^* \) 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

Returns an object of type `survbart` with predictions corresponding to `newdata`.

### Author(s)

Robert McCulloch: `<robert.e.mcculloch@gmail.com>`
Rodney Sparapani: `<rsparapa@mcw.edu>`.

### References


### See Also

`surv.bart`, `mc.surv.bart`, `surv.pwbart`, `mc.surv.pwbart`, `mc.cores.openmp`
Examples

```r
## load the advanced lung cancer example
data(lung)

group <- -which(is.na(lung[, 7])) ## remove missing row for ph.karno
times <- lung[group, 2]  ##lung$time
delta <- lung[group, 3]-1  ##lung$status: 1=censored, 2=dead
##delta: 0=censored, 1=dead
## this study reports time in days rather than months like other studies
## coarsening from days to months will reduce the computational burden
## times <- ceiling(times/30)

summary(times)
table(delta)

x.train <- as.matrix(lung[group, c(4, 5, 7)])  ## matrix of observed covariates
## lung$age: Age in years
## lung$sex: Male=1 Female=2
## lung$ph.karno: Karnofsky performance score (dead=0: normal=100: by=10)
## rated by physician
dimnames(x.train)[[2]] <- c('age(yr)', 'M(1):F(2)', 'ph.karno(0:100:10)')

summary(x.train[, 1])
table(x.train[, 2])
table(x.train[, 3])

x.test <- matrix(nrow=84, ncol=3)  ## matrix of covariate scenarios

dimnames(x.test)[[2]] <- dimnames(x.train)[[2]]
i <- 1

for(age in 5*(9:15)) for(sex in 1:2) for(ph.karno in 10*(5:10)) {
  x.test[i, ] <- c(age, sex, ph.karno)
  i <- i+1
}

## this x.test is relatively small, but often you will want to
## predict for a large x.test matrix which may cause problems
## due to consumption of RAM so we can predict separately

## mcparallel/mccollect do not exist on windows
if(.Platform$OS.type=='unix') {
  ##test BART with token run to ensure installation works
  set.seed(99)
  post <- surv.bart(x.train=x.train, times=times, delta=delta, nskip=5, ndpost=5, keepevery=1)
  pre <- surv.pre.bart(x.train=x.train, times=times, delta=delta, x.test=x.test)
```
```r
pred <- predict(post, pre$tx.test)
##pred. <- surv.pwbart(pre$tx.test, post$treedraws, post$binaryOffset)
}
## Not run:
## run one long MCMC chain in one process
set.seed(99)
post <- surv.bart(x.train=x.train, times=times, delta=delta)
## run "mc.cores" number of shorter MCMC chains in parallel processes
## post <- mc.surv.bart(x.train=x.train, times=times, delta=delta,
## mc.cores=5, seed=99)
pre <- surv.pre.bart(x.train=x.train, times=times, delta=delta, x.test=x.test)
pred <- predict(post, pre$tx.test)
## let's look at some survival curves
## first, a younger group with a healthier KPS
## age 50 with KPS=90: males and females
## males: row 17, females: row 23
x.test[c(17, 23), ]
low.risk.males <- 16*post$K+1:post$K ## K=unique times including censoring
low.risk.females <- 22*post$K+1:post$K
plot(post$times, pred$surv.test.mean[low.risk.males], type='s', col='blue',
main='Age 50 with KPS=90', xlab='t', ylab='S(t)', ylim=c(0, 1))
points(post$times, pred$surv.test.mean[low.risk.females], type='s', col='red')
## End(Not run)
```

### predict.wbart

**Predicting new observations with a previously fitted BART model**

#### Description

BART is a Bayesian “sum-of-trees” model.  
For a numeric response $y$, we have $y = f(x) + \epsilon$, where $\epsilon \sim N(0, \sigma^2)$.  

$f$ is the sum of many tree models. The goal is to have very flexible inference for the unknown 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.
predict.wbart

Usage

```r
## S3 method for class 'wbart'
predict(object, newdata, mc.cores=1, openmp=(mc.cores.openmp()>0), ...)
```

Arguments

- `object`: object returned from previous BART fit.
- `newdata`: Matrix of covariates to predict \( y \) for.
- `mc.cores`: Number of threads to utilize.
- `openmp`: Logical value dictating whether OpenMP is utilized for parallel processing. Of course, this depends on whether OpenMP is available on your system which, by default, is verified with `mc.cores.openmp`.
- `...`: Other arguments which will be passed on to `pwbart`.

Details

BART 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 \( \sigma^* \) 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

Returns a matrix of predictions corresponding to `newdata`.

Author(s)

Robert McCulloch: `<robert.e.mcculloch@gmail.com>`,
Rodney Sparapani: `<rsparapa@mcw.edu>`.

References


See Also

`wbart`, `mc.wbart`, `pwbart`, `mc.pwbart`, `mc.cores.openmp`
Examples

```r
# simulate data (example from Friedman 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
y=f(x)

# test BART with token run to ensure installation works
set.seed(99)
post = wbart(x,y,nskip=5,ndpost=5)
x.test = matrix(runif(500*10),500,10)

# Not run:
## run BART
set.seed(99)
post = wbart(x,y)
x.test = matrix(runif(500*10),500,10)
pred = predict(post, x.test, mu=mean(y))
plot(apply(pred, 2, mean), f(x.test))

## End(Not run)
```

Description

BART is a Bayesian “sum-of-trees” model. For a numeric response $y$, we have $y = f(x) + \epsilon$, where $\epsilon \sim N(0, \sigma^2)$.

$f$ is the sum of many tree models. The goal is to have very flexible inference for the unknown 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

```r
pwbart( x.test, treedraws, mu=0, mc.cores=1L, transposed=FALSE,
        dodraws=TRUE,
        nice=19L  ## mc.pwbart only
)
```
mc.pwbart(  x.test, treedraws, mu=0, mc.cores=2L, transposed=FALSE,  
  dodraws=TRUE,  
  nice=19L ## mc.pwbart only  
)

Arguments

- **x.test**: Matrix of covariates to predict *y* for.
- **treedraws**: $\text{treedraws}$ returned from *wbart* or *pbart*.
- **mu**: Mean to add on to *y* prediction.
- **mc.cores**: Number of threads to utilize.
- **transposed**: When running *pwbart* or *mc.pwbart* in parallel, it is more memory-efficient to transpose *x.test* prior to calling the internal versions of these functions.
- **dodraws**: Whether to return the draws themselves (the default), or whether to return the mean of the draws as specified by *dodraws=FALSE*.
- **nice**: Set the job niceness. The default niceness is 19: niceness goes from 0 (highest) to 19 (lowest).

Details

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

Thus, unlike a lot of other 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 $\sigma^*$ 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

Returns a matrix of predictions corresponding to *x.test*.

Author(s)

Robert McCulloch: <robert.e.mcculloch@gmail.com>,
Rodney Sparapani: <rsparapa@mcw.edu>.

References


See Also

*wbart predict.wbart*
Examples

```r
# simulate data (example from Friedman 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
y=f(x)

## test BART with token run to ensure installation works
set.seed(99)
post = wbart(x,y,nskip=5,ndpost=5)
x.test = matrix(runif(500*10),500,10)

## Not run:
## run BART
set.seed(99)
post = wbart(x,y)
x.test = matrix(runif(500*10),500,10)
pred = pwbart(post$treedraws, x.test, mu=mean(y))
plot(apply(pred, 2, mean), f(x.test))

## End(Not run)
```

### recur.bart

**BART for recurrent events**

**Description**

Here we have implemented a simple and direct approach to utilize BART in survival analysis that is very flexible, and is akin to discrete-time survival analysis. Following the capabilities of BART, we allow for maximum flexibility in modeling the dependence of survival times on covariates. In particular, we do not impose proportional hazards.

To elaborate, consider data in the usual form: \((t_i, \delta_i, x_i)\) where \(t_i\) is the event time, \(\delta_i\) is an indicator distinguishing events \((\delta = 1)\) from right-censoring \((\delta = 0)\), \(x_i\) is a vector of covariates, and \(i = 1, \ldots, N\) indexes subjects.

We denote the \(K\) distinct event/censoring times by \(0 < t_{(1)} < \ldots < t_{(K)} < \infty\) thus taking \(t_{(j)}\) to be the \(j^{th}\) order statistic among distinct observation times and, for convenience, \(t_{(0)} = 0\). Now consider event indicators \(y_{ij}\) for each subject \(i\) at each distinct time \(t_{(j)}\) up to and including the subject’s observation time \(t_i = t_{(n_i)}\) with \(n_i = \sum_j I[t_{(j)} \leq t_i]\). This means \(y_{ij} = 0\) if \(j < n_i\) and \(y_{in_i} = \delta_i\).

We then denote by \(p_{ij}\) the probability of an event at time \(t_{(j)}\) conditional on no previous event. We now write the model for \(y_{ij}\) as a nonparametric probit regression of \(y_{ij}\) on the time \(t_{(j)}\) and the
covariates $x_i$, and then utilize BART for binary responses. Specifically, $y_{ij} = \delta_i I[t_i = t(j)]$, $j = 1, \ldots, n_i$; we have $p_{ij} = F(\mu_{ij})$, $\mu_{ij} = \mu_0 + \delta_j(x_i)$ where $F$ denotes the standard normal cdf (probit link). As in the binary response case, $f$ is the sum of many tree models.

Usage

```r
recur.bart(x.train=matrix(0,0,0),
    y.train=NULL, times=NULL, delta=NULL,
    x.test=matrix(0,0,0), x.test.nogrid=FALSE,
    sparse=FALSE, theta=0, omega=1,
    a=0.5, b=1, augment=FALSE, rho=NULL,
    xinfo=matrix(0,0,0), usequants=FALSE,
    rm.const=TRUE, type='pbart',
    ntype=as.integer(
        factor(type, levels=c('wbart', 'pbart', 'lbart'))),
    k=2, power=2, base=0.95,
    offset=NULL, tau.num=c(NA, 3, 6)[ntype],
    ntree=50, numcut = 100L, ndpost=1000, nskip=250,
    keepevery=10,
    printevery = 100L,
    keeptrainfits = TRUE,
    seed=99, ## mc.recur.bart only
    mc.cores=2, ## mc.recur.bart only
    nice=19L   ## mc.recur.bart only
)
```

```r
mc.recur.bart(x.train=matrix(0,0,0),
    y.train=NULL, times=NULL, delta=NULL,
    x.test=matrix(0,0,0), x.test.nogrid=FALSE,
    sparse=FALSE, theta=0, omega=1,
    a=0.5, b=1, augment=FALSE, rho=NULL,
    xinfo=matrix(0,0,0), usequants=FALSE,
    rm.const=TRUE, type='pbart',
    ntype=as.integer(
        factor(type, levels=c('wbart', 'pbart', 'lbart'))),
    k=2, power=2, base=0.95,
    offset=NULL, tau.num=c(NA, 3, 6)[ntype],
    ntree=50, numcut = 100L, ndpost=1000, nskip=250,
    keepevery=10,
    printevery = 100L,
    keeptrainfits = TRUE,
    seed=99, ## mc.recur.bart only
)
```
mc.cores=2, ## mc.recur.bart only
nice=19L     ## mc.recur.bart only
)

Arguments

x.train  Explanatory variables for training (in sample) data.
          Must be a matrix with (as usual) rows corresponding to observations and columns
to variables.
          recur.bart will generate draws of \( f(t, x) \) for each \( x \) which is a row of x.train
          (note that the definition of x.train is dependent on whether y.train has been
          specified; see below).

y.train   Binary response dependent variable for training (in sample) data.
          If y.train is NULL, then y.train (x.train and x.test, if specified) are generated
          by a call to recur.pre.bart (which require that times and delta be
          provided: see below); otherwise, y.train (x.train and x.test, if specified)
          are utilized as given assuming that the data construction has already been
          performed.

times     The time of event or right-censoring.
          If y.train is NULL, then times (and delta) must be provided.

delta     The event indicator: 1 is an event while 0 is censored.
          If y.train is NULL, then delta (and times) must be provided.

x.test    Explanatory variables for test (out of sample) data.
          Must be a matrix and have the same structure as x.train.
          recur.bart will generate draws of \( f(t, x) \) for each \( x \) which is a row of x.test.

x.test.nogrid Occasionally, you do not need the entire time grid for x.test. If so, then for
          performance reasons, you can set this argument to TRUE.

sparse    Whether to perform variable selection based on a sparse Dirichlet prior rather
          than simply uniform; see Linero 2016.

theta     Set \( \theta \) parameter; zero means random.

omega     Set \( \omega \) parameter; zero means random.
a         Sparse parameter for \( \text{Beta}(a, b) \) prior: \( 0.5 \leq a \leq 1 \) where lower values
          inducing more sparsity.
b         Sparse parameter for \( \text{Beta}(a, b) \) prior; typically, \( b = 1 \).
 rho       Sparse parameter: typically \( \rho = p \) where \( p \) is the number of covariates under
          consideration.

augment   Whether data augmentation is to be performed in sparse variable selection.

xinfo     You can provide the cutpoints to BART or let BART choose them for you. To
          provide them, use the xinfo argument to specify a list (matrix) where the items
          (rows) are the covariates and the contents of the items (columns) are the cut-
          points.

usequants If usequants=FALSE, then the cutpoints in xinfo are generated uniformly; other-
           wise, if TRUE, uniform quantiles are used for the cutpoints.

rm.const  Whether or not to remove constant variables.
Whether to employ Albert-Chib, \texttt{pbart}, or Holmes-Held, \texttt{lbart}.

\texttt{ntype}  
The integer equivalent of \texttt{type} where \texttt{wbart} is 1, \texttt{pbart} is 2 and \texttt{lbart} is 3.

\texttt{k}  
k is the number of prior standard deviations $f(t, x)$ is away from $+/-3$. The bigger \texttt{k} is, the more conservative the fitting will be.

\texttt{power}  
Power parameter for tree prior.

\texttt{base}  
Base parameter for tree prior.

\texttt{offset}  
With binary BART, the centering is $P(Y = 1|x) = F(f(x) + \text{offset})$ where offset defaults to $F^{-1}(\text{mean}(y.\text{train}))$. You can use the offset parameter to over-ride these defaults.

\texttt{tau.num}  
The numerator in the tau definition, i.e., $\text{tau} = \text{tau.num}/(k*\text{sqrt}(\text{ntree}))$.

\texttt{ntree}  
The number of trees in the sum.

\texttt{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.\text{train}) is required, where the $i^{th}$ element gives the number of c used for the $i^{th}$ variable in x.\text{train}. If usequants is false, numcut equally spaced cutoffs are used covering the range of values in the corresponding column of x.\text{train}. If usequants is true, then min(numcut, the number of unique values in the corresponding columns of x.\text{train} - 1) c values are used.

\texttt{ndpost}  
The number of posterior draws returned.

\texttt{nskip}  
Number of MCMC iterations to be treated as burn in.

\texttt{keepevery}  
Every keepevery draw is kept to be returned to the user.

\texttt{printevery}  
As the MCMC runs, a message is printed every printevery draws.

\texttt{keeptrainfits}  
Whether to keep yhat.\text{train} or not.

\texttt{seed}  
\texttt{mc.recur.bart} only: seed required for reproducible MCMC.

\texttt{mc.cores}  
\texttt{mc.recur.bart} only: number of cores to employ in parallel.

\texttt{nice}  
\texttt{mc.recur.bart} only: set the job niceness. The default niceness is 19: niceness goes from 0 (highest) to 19 (lowest).

\textbf{Value}

\texttt{recur.bart} returns an object of type \texttt{recurbart} which is essentially a list. Besides the items listed below, the list has a \texttt{binaryOffset} component giving the value used, a \texttt{times} component giving the unique times, \texttt{K} which is the number of unique times, \texttt{tx.\text{train}} and \texttt{tx.\text{test}}, if any.

\texttt{yhat.train}  
A matrix with \texttt{ndpost} rows and nrow(x.\text{train}) columns. Each row corresponds to a draw $f^*$ from the posterior of $f$ and each column corresponds to a row of \texttt{x.\text{train}}. The $(i, j)$ value is $f^*(t, x)$ for the $i^{th}$ kept draw of $f$ and the $j^{th}$ row of \texttt{x.\text{train}}.

\texttt{Burn-in is dropped.}

\texttt{haz.train}  
The hazard function, $h(t|x)$, where x’s are the rows of the training data.

\texttt{cum.train}  
The cumulative hazard function, $h(t|x)$, where x’s are the rows of the training data.
yhat.test  Same as yhat.train but now the x’s are the rows of the test data.

haz.test  The hazard function, \( h(t|x) \), where x’s are the rows of the test data.

cum.test  The cumulative hazard function, \( h(t|x) \), where x’s are the rows of the test data.

varcount  a matrix with ndpost 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.

Note that yhat.train and yhat.test are \( f(t,x) + \text{binaryOffset} \). If you want draws of the probability \( P(Y = 1|t,x) \) you need to apply the normal cdf (\text{pnorm}) to these values.

Author(s)

Rodney Sparapani: <rsparapa@mcw.edu>

References


See Also

recur.pre.bart, predict.recurbart, recur.pwbart, mc.recur.pwbart

Examples

```r
## load 20 percent random sample
data(xdm20.train)
data(xdm20.test)
data(ydm20.train)

## test BART with token run to ensure installation works
## with current technology even a token run will violate CRAN policy
## set.seed(99)
## post <- recur.bart(x.train=xdm20.train, y.train=ydm20.train,
## nskip=1, ndpost=1, keepevery=1)

## Not run:
```
## set.seed(99)
## post <- recur.bart(x.train=xdm20.train, y.train=ydm20.train,
## keeptrainfits=TRUE)

## larger data sets can take some time so, if parallel processing
## is available, submit this statement instead
post <- mc.recur.bart(x.train=xdm20.train, y.train=ydm20.train,
keeptrainfits=TRUE, mc.cores=8, seed=99)

require(rpart)
require(rpart.plot)

post$yhat.train.mean <- apply(post$yhat.train, 2, mean)
dss <- rpart(post$yhat.train.mean~xdm20.train)
rpart.plot(dss)

## for the 20 percent sample, notice that the top splits
## involve cci_pvd and n
## for the full data set, notice that all splits
## involve ca, cci_pud, cci_pvd, ins270 and n
## (except one at the bottom involving a small group)

## compare patients treated with insulin (ins270=1) vs
## not treated with insulin (ins270=0)
N <- 50 ## 50 training patients and 50 validation patients
K <- post$K ## 798 unique time points
NK <- 50*K

## only testing set, i.e., remove training set
xdm20.test. <- xdm20.test[NK+1:NK, post$rm.const]
xdm20.test. <- rbind(xdm20.test., xdm20.test.)
xdm20.test.[, 'ins270'] <- rep(0:1, each=NK)

## multiple threads will be utilized if available
pred <- predict(post, xdm20.test., mc.cores=8)

## create Friedman's partial dependence function for the
## relative intensity for ins270 by time
M <- nrow(pred$haz.test) ## number of MCMC samples
RI <- matrix(0, M, K)
for(j in 1:K) {
  h <- seq(j, NK, by=K)
  RI[ , j] <- apply(pred$haz.test[ , h+NK]/
pred$haz.test[ , h], 1, mean)
}

RI.lo <- apply(RI, 2, quantile, probs=0.025)
RI.mu <- apply(RI, 2, mean)
RI.hi <- apply(RI, 2, quantile, probs=0.975)

plot(post$times, RI.hi, type='l', lty=2, log='y',
     ylim=c(min(RI.lo, 1/RI.hi), max(1/RI.lo, RI.hi)),
     ...
recur.pre.bart

Data construction for recurrent events with BART

Description

Recurrent event data contained in \((t_1, \delta_1, ..., t_k, \delta_k, x)\) must be translated to data suitable for the BART model; see recur.bart for more details.

Usage

recur.pre.bart( times, delta, x.train=NULL, tstop=NULL, last.value=TRUE )

Arguments

times: Matrix of time to event or right-censoring.
delta: Matrix of event indicators: 1 is an event while 0 is censored.
x.train: Explanatory variables for training (in sample) data. If provided, must be a matrix with (as usual) rows corresponding to observations and columns to variables.
tstop: For non-instantaneous events, this the matrix of event stop times, i.e., between times[i,j] and tstop[i,j] subject i is not in the risk set for a recurrent event. N.B. This is NOT for counting process notation.
last.value: If last.value=TRUE, then the sojourn time, v, and the number of previous events, N, are carried forward assuming that no new events occur beyond censoring. If last.value=FALSE, then these variables are coded NA for easy identification allowing replacement with the desired values.
Value

recur.pre.bart returns a list. Besides the items listed below, the list has a times component giving the unique times and K which is the number of unique times.

- **y.train**: A vector of binary responses.
- **tx.train**: A matrix with the rows of the training data.
- **tx.test**: Generated from x.train (see discussion above included in the argument last.value).

Author(s)

Rodney Sparapani: <rsparapa@mcw.edu>

References


See Also

recur.bart

Examples

data(bladder)
subset <- -which(bladder1$stop==0)
bladder0 <- bladder1[subset,]
id <- unique(sort(bladder0$id))
N <- length(id)
L <- max(bladder0$enum)
times <- matrix(0, nrow=N, ncol=L)
dimnames(times)[[1]] <- paste0(id)
delta <- matrix(0, nrow=N, ncol=L)
dimnames(delta)[[1]] <- paste0(id)
x.train <- matrix(NA, nrow=N, ncol=3+2*L) ## add time-dependent cols too
dimnames(x.train)[[1]] <- paste0(id)
dimnames(x.train)[[2]] <- c('Pl', 'B6', 'Th', rep(c('number', 'size'), L))
for(i in 1:N) {
  h <- id[i]
  for(j in 1:L) {
    k <- which(bladder0$id==h & bladder0$enum==j)
if(length(k)==1) {
  times[i, j] <- bladder0$stop[k]
  delta[i, j] <- (bladder0$status[k]==1)*1

  if(j==1) {
    x.train[i, 1] <- as.numeric(bladder0$treatment[k]==1)
    x.train[i, 2] <- as.numeric(bladder0$treatment[k]==2)
    x.train[i, 3] <- as.numeric(bladder0$treatment[k]==3)
    x.train[i, 4] <- bladder0$number[k]
    x.train[i, 5] <- bladder0$size[k]
  }
  else if(delta[i, j]==1) {
    if(bladder0$rtumor[k]!="Var"
      x.train[i, 2*j+2] <- as.numeric(bladder0$rtumor[k])
    if(bladder0$rsize[k]!="Var"
      x.train[i, 2*j+3] <- as.numeric(bladder0$rsize[k])
  }
  }
}
}

pre <- recur.pre.bart(times=times, delta=delta, x.train=x.train)

J <- nrow(pre$tx.train)
for(j in 1:J) {
  if(pre$tx.train[j, 3]>0) {
    pre$tx.train[j, 7] <- pre$tx.train[j, 7+pre$tx.train[j, 3]*2]
    pre$tx.train[j, 8] <- pre$tx.train[j, 8+pre$tx.train[j, 3]*2]
  }
}

pre$tx.train <- pre$tx.train[, 1:8]

K <- pre$K
NK <- N*K
for(j in 1:NK) {
  if(pre$tx.test[j, 3]>0) {
    pre$tx.test[j, 7] <- pre$tx.test[j, 7+pre$tx.test[j, 3]*2]
    pre$tx.test[j, 8] <- pre$tx.test[j, 8+pre$tx.test[j, 3]*2]
  }
}

pre$tx.test <- pre$tx.test[, 1:8]

## in bladder1 both number and size are recorded as integers
## from 1 to 8 however they are often missing for recurrences
## at baseline there are no missing and 1 is the mode of both
pre$tx.train[which(is.na(pre$tx.train[, 7])), 7] <- 1
pre$tx.train[which(is.na(pre$tx.train[, 8])), 8] <- 1
pre$tx.test[which(is.na(pre$tx.test[, 7])), 7] <- 1
pre$tx.test[which(is.na(pre$tx.test[, 8])), 8] <- 1

## it is a good idea to explore more sophisticated methods
## such as imputing the missing data with Sequential BART


http://biostatistics.oxfordjournals.org/content/early/2016/03/15/biostatistics.kxw009/suppl/DC1

https://cran.r-project.org/package=sbart

```
library(sbart)
set.seed(21)
train <- seqBART(xx=pre$tx.train, yy=NULL, datatype=rep(0, 6),
    type=0, numskip=20, burn=1000)
# coarsen the imputed data same way as observed example data
train$imputed5[which(train$imputed5[, 7]<1), 7] <- 1
train$imputed5[which(train$imputed5[, 7]>8), 7] <- 8
train$imputed5[, 7] <- round(train$imputed5[, 7])
train$imputed5[which(train$imputed5[, 8]<1), 8] <- 1
train$imputed5[which(train$imputed5[, 8]>8), 8] <- 8
train$imputed5[, 8] <- round(train$imputed5[, 8])

for Friedman’s partial dependence, we need to estimate the whole cohort
at each treatment assignment (and, average over those)
```

```
pre$x.test <- rbind(pre$x.test, pre$x.test, pre$x.test)
pre$x.test[, 4] <- c(rep(1, NK), rep(0, 2*NK)) ## Pl
pre$x.test[, 5] <- c(rep(0, NK), rep(1, NK), rep(0, NK))## B6
pre$x.test[, 6] <- c(rep(0, 2*NK), rep(1, NK)) ## Th
```

```
M <- nrow(post$yhat.test)
RI.B6.P1 <- matrix(0, nrow=M, ncol=K)
RI.Th.P1 <- matrix(0, nrow=M, ncol=K)
RI.Th.B6 <- matrix(0, nrow=M, ncol=K)
```

```
for(j in 1:K) {
    h <- seq(j, NK, K)
    RI.B6.P1[, j] <- apply(post$prob.test[, h+NK]/
        post$prob.test[, h], 1, mean)
    RI.Th.P1[, j] <- apply(post$prob.test[, h+2*NK]/
        post$prob.test[, h], 1, mean)
    RI.Th.B6[, j] <- apply(post$prob.test[, h+2*NK]/
        post$prob.test[, h+NK], 1, mean)
}
```

```
RI.B6.P1.mu <- apply(RI.B6.P1, 2, mean)
RI.B6.P1.025 <- apply(RI.B6.P1, 2, quantile, probs=0.025)
RI.B6.P1.975 <- apply(RI.B6.P1, 2, quantile, probs=0.975)
```

```
RI.Th.P1.mu <- apply(RI.Th.P1, 2, mean)
RI.Th.P1.025 <- apply(RI.Th.P1, 2, quantile, probs=0.025)
RI.Th.P1.975 <- apply(RI.Th.P1, 2, quantile, probs=0.975)
```
RI.Th.B6.mu <- apply(RI.Th.B6, 2, mean)
RI.Th.B6.025 <- apply(RI.Th.B6, 2, quantile, probs=0.025)
RI.Th.B6.975 <- apply(RI.Th.B6, 2, quantile, probs=0.975)

plot(post$times, RI.Th.Pl.mu, col='blue',
     log='y', main='Bladder cancer ex: Thiotepa vs. Placebo',
     type='l', ylim=c(0.1, 10), ylab='RI(t)', xlab='t (months)')
lines(post$times, RI.Th.Pl.025, col='red')
lines(post$times, RI.Th.Pl.975, col='red')
abline(h=1)

plot(post$times, RI.B6.Pl.mu, col='blue',
     log='y', main='Bladder cancer ex: Vitamin B6 vs. Placebo',
     type='l', ylim=c(0.1, 10), ylab='RI(t)', xlab='t (months)')
lines(post$times, RI.B6.Pl.025, col='red')
lines(post$times, RI.B6.Pl.975, col='red')
abline(h=1)

plot(post$times, RI.Th.B6.mu, col='blue',
     log='y', main='Bladder cancer ex: Thiotepa vs. Vitamin B6',
     type='l', ylim=c(0.1, 10), ylab='RI(t)', xlab='t (months)')
lines(post$times, RI.Th.B6.025, col='red')
lines(post$times, RI.Th.B6.975, col='red')
abline(h=1)

## End(Not run)

rs.pbart

BART for dichotomous outcomes with parallel computation and stratified random sampling

Description

BART is a Bayesian “sum-of-trees” model.
For 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 unknown 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

rs.pbart(
  x.train, y.train, x.test=matrix(0.0,0,0),
  C=floor(length(y.train)/2000),
  H=10000,
  B=5000,
  T=100000,
  P=1,
  R=2000,
  S=5000000.
)
k=2.0, power=2.0, base=.95,
binaryOffset=0,
ntree=50L, numcut=100L,
ndpost=1000L, nskip=100L,
keepevery=1L, printevery=100,
keeptrainfits=FALSE, transposed=FALSE,
mc.cores = 2L, nice = 19L,
seed = 99L
)

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. pbart 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 continuous 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.

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

C The number of shards to break the data into and analyze separately.

k 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 \(P(Y = 1|x) = F(f(x) + \text{binaryOffset})\). 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.

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.

ndpost The number of posterior draws returned.

nskip Number of MCMC iterations to be treated as burn in.
keepevery  Every keepevery draw is kept to be returned to the user.

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

keeptainfits  Whether to keep yhat.train or not.

transposed  When running pbart in parallel, it is more memory-efficient to transpose x.train and x.test, if any, prior to calling mc.pbart.

seed  Setting the seed required for reproducible MCMC.

mc.cores  Number of cores to employ in parallel.

nice  Set the job niceness. The default niceness is 19: niceness goes from 0 (highest) to 19 (lowest).

Details

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

Thus, unlike a lot of other 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 \(\sigma^*\) 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

\(rs.pbart\) returns an object of type \(pbart\) which is essentially a list.

\(yhat.shard\)  Estimates generated from the individual shards rather than from the whole. This object is only useful for assessing convergence. A matrix with ndpost 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.

\(yhat.train\)  Estimates generated from the whole if \(keeptainfits=TRUE\). A matrix with ndpost 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.

\(yhat.test\)  Estimates generated from the whole if \(x.test\) is provided. Same as \(yhat.train\) but now the \(x\)'s are the rows of the test data.

\(varcount\)  a matrix with ndpost 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.

In addition the list has a binaryOffset component giving the value used.

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

Robert McCulloch: <robert.e.mcculloch@gmail.com>,
Rodney Sparapani: <rsparapa@mcw.edu>.

References


See Also

mc.pbart

Examples

## simulate from Friedman’s five-dimensional test function
## Friedman JH. Multivariate adaptive regression splines
## (with discussion and a rejoinder by the author).

f = function(x) # only the first 5 matter
    sin(pi*x[, 1]*x[, 2]) + 2*(x[, 3]-.5)^2+x[, 4]+0.5*x[, 5]-1.5

sigma = 1.0 # y = f(x) + sigma*z where z~N(0, 1)
k = 50 # number of covariates
thin = 25
ndpost = 2500
nskip = 100
C = 10
m = 10
n = 10000

set.seed(12)
x.train=matrix(runif(n*k), n, k)
Ey.train = f(x.train)
y.train=(Ey.train+sigma*rnorm(n)>0)*1
table(y.train)/n

x <- x.train
x4 <- seq(0, 1, length.out=m)

for(i in 1:m) {
    x[, 4] <- x4[i]
    if(i==1) x.test <- x
    else x.test <- rbind(x.test, x)
}
## parallel::mcparallel/mccollect do not exist on windows
if(.Platform$OS.type=='unix') {
  ## test BART with token run to ensure installation works
  post = rs.pbart(x.train, y.train,
                  C=C, mc.cores=4, keepevery=1,
                  seed=99, ndpost=1, nskip=1)
}

## Not run:
post = rs.pbart(x.train, y.train, x.test=x.test,
               C=C, mc.cores=8, keepevery=thin,
               seed=99, ndpost=ndpost, nskip=nskip)
str(post)

par(mfrow=c(2, 2))

M <- nrow(post$yhat.test)
pred <- matrix(nrow=M, ncol=10)
for(i in 1:m) {
  h <- (i-1)*n+1:n
  pred[, i] <- apply(pnorm(post$yhat.test[, h]), 1, mean)
}
pred <- apply(pred, 2, mean)

plot(x4, qnorm(pred), xlab=expression(x[4]),
     ylab=partial dependence function, type='l')

i <- floor(seq(1, n, length.out=10))
j <- seq(-0.5, 0.4, length.out=10)
for(h in 1:10) {
  auto.corr <- acf(post$yhat.shard[, i[h]], plot=FALSE)
  if(h==1) {
    max.lag <- max(auto.corr$lag[, 1, 1])
    plot(1:max.lag+j[h], auto.corr$acf[1+(1:max.lag), 1, 1],
         type='h', xlim=c(0, max.lag+1), ylim=c(-1, 1),
         ylab='auto-correlation', xlab='lag')
  } else
    lines(1:max.lag+j[h], auto.corr$acf[1+(1:max.lag), 1, 1],
          type='h', col=h)
}

for(j in 1:10) {
  if(j==1)
    plot(pnorm(post$yhat.shard[, i[j]]),
         type='l', ylim=c(0, 1),
         sub=paste0("N: ", n, ", k: ", k),
         ylab=expression(Phi(f(x))), xlab='m')
  else
    # code continues
rtgamma <- gewekediag(post$yhat.shard)

j <- -10^{(log10(n)-1)}
plot(geweke$z, pch='.', cex=2, ylab='z', xlab='i',
     sub=paste0('N: ', n, ', ', k, ': ', k),
     xlim=c(j, n), ylim=c(-5, 5))
lines(1:n, rep(-1.96, n), type='l', col=6)
lines(1:n, rep(+1.96, n), type='l', col=6)
lines(1:n, rep(-2.576, n), type='l', col=5)
lines(1:n, rep(+2.576, n), type='l', col=5)
lines(1:n, rep(-3.291, n), type='l', col=4)
lines(1:n, rep(+3.291, n), type='l', col=4)
lines(1:n, rep(-3.891, n), type='l', col=3)
lines(1:n, rep(+3.891, n), type='l', col=3)
lines(1:n, rep(-4.417, n), type='l', col=2)
lines(1:n, rep(+4.417, n), type='l', col=2)
text(c(1, 1), c(-1.96, 1.96), pos=2, cex=0.6, labels='0.95')
text(c(1, 1), c(-2.576, 2.576), pos=2, cex=0.6, labels='0.99')
text(c(1, 1), c(-3.291, 3.291), pos=2, cex=0.6, labels='0.999')
text(c(1, 1), c(-3.891, 3.891), pos=2, cex=0.6, labels='0.9999')
text(c(1, 1), c(-4.417, 4.417), pos=2, cex=0.6, labels='0.99999')

par(mfrow=c(1, 1))

## dev.copy2pdf(file='geweke.rs.pbart.pdf')

## End(Not run)

---

**Description**

Truncated Gamma draws are needed for the standard deviation of the random effects Gibbs conditional.

**Usage**

```r
rtgamma(n, shape, rate, a)
```

**Arguments**

- `n` Number of samples.
- `shape` Sampling from a truncated Gamma where $E[x] = \text{shape}/\text{rate}$. 
rate

This parameter is the inverse of the scale which is an alternative representation for the Gamma distribution.

a

The truncation point, i.e., $a < x$.

Value

Returns $n$ truncated Gamma, i.e., $\text{Gam}(\text{shape}, \text{rate}) I(a, \infty)$.

Author(s)

Robert McCulloch: <robert.e.mcculloch@gmail.com>, Rodney Sparapani: <rsparapa@mcw.edu> , Robert Gramacy: <rbg@vt.edu>.

References


Examples

```r
set.seed(12)
rtgamma(1, 3, 1, 4)
rtgamma(1, 3, 1, 4)

a=rtgamma(10000, 10, 2, 1)
mean(a)
min(a)
```

---

### rtnorm

Testing truncated Normal sampling

Description

Truncated Normal latents are necessary to transform a binary BART into a continuous BART.

Usage

```
rtnorm(n, mean, sd, tau)
```

Arguments

- `n` Number of samples.
- `mean` Mean.
- `sd` Standard deviation.
- `tau` Truncation point.
spectrum0ar

Value

Returns n truncated Normals, i.e., $N(\text{mean}, \text{sd})I(\tau, \infty)$.

Author(s)

Robert McCulloch: <robert.e.mcculloch@gmail.com>,
Rodney Sparapani: <rsparapa@mcw.edu>,
Robert Gramacy: <rbg@vt.edu>.

References


See Also

pbart, lbart

Examples

```r
set.seed(12)

rtnorm(1, 0, 1, 3)
rtnorm(1, 0, 1, 3)
```

---

### spectrum0ar

Estimate spectral density at zero

**Description**

The spectral density at frequency zero is estimated by fitting an autoregressive model. spectrum0ar(x)/length(x) estimates the variance of mean(x).

**Usage**

`spectrum0ar(x)`

**Arguments**

- `x` Matrix of MCMC chains: the rows are the samples and the columns are different "parameters". For BART, generally, the columns are estimates of $f$. For pbart, they are different subjects. For surv.bart, they are different subjects at a grid of times.
Details
The \texttt{ar()} function to fit an autoregressive model to the time series \texttt{x}. For multivariate time series, separate models are fitted for each column. The value of the spectral density at zero is then given by a well-known formula. Adapted from the \texttt{spectrum0.ar} function of the coda package which passes \texttt{mcmc} objects as arguments rather than matrices.

Value
A list with the following values
- \texttt{spec}: The predicted value of the spectral density at frequency zero.
- \texttt{order}: The order of the fitted model

References

See Also
\texttt{gewekediag}

\begin{itemize}
  \item \texttt{srstepwise} \hspace{2em} \emph{Stepwise Variable Selection Procedure for survreg}
\end{itemize}

Description
This stepwise variable selection procedure can be applied to obtain the best candidates for a \texttt{survreg} fit.

Usage
\begin{itemize}
  \item \texttt{srstepwise(x, times, delta, sle = 0.15, sls = 0.15, dist='lognormal')}
\end{itemize}

Arguments
\begin{itemize}
  \item \texttt{x}: Matrix of variables to consider.
  \item \texttt{times}: The time to an event, if any.
  \item \texttt{delta}: The event indicator: 1 for event, 0 for no event.
  \item \texttt{sle}: The chosen significance level for entering.
  \item \texttt{sls}: The chosen significance level for staying.
  \item \texttt{dist}: The distribution to be used by \texttt{survreg}.
\end{itemize}
stratrs

Details

Unfortunately, no stepwise procedure exists for `survreg` models. Therefore, we provide this brute force method.

Value

Returns a list of indices of variables which have entered and stayed.

See Also

lung

Examples

```r
names. <- names(lung)[-(2:3)]
status1 <- ifelse(lung$status==2,1,0)
X <- as.matrix(lung[, names.]
vars=srstepwise(X, lung$time, status1)
print(names.[vars])
```

---

stratrs  
**Perform stratified random sampling to balance outcomes**

Description

This function is used to perform stratified random sampling to balance outcomes among the shards.

Usage

```r
stratrs(y, C=5, P=0)
```

Arguments

- **y**  
The binary/categorical/continuous outcome.

- **C**  
The number of shards to break the data set into.

- **P**  
For continuous data, we break the range into P segments via the quantiles. Specifying, P=20 seems to work reasonably well.

Details

To perform BART with large data sets, random sampling is employed to break the data into C shards. Each shard should be balanced with respect to the outcome. For binary/categorical outcomes, stratified random sampling is employed with this function.
Value

A vector is returned with each element assigned to a shard.

See Also

rs.pbart

Examples

```r
set.seed(12)
x <- rbinom(25000, 1, 0.1)
a <- stratrs(x)
table(a, x)
z <- pmin(rpois(25000, 0.8), 5)
b <- stratrs(z)
table(b, z)
```

Description

Here we have implemented a simple and direct approach to utilize BART in survival analysis that is very flexible, and is akin to discrete-time survival analysis. Following the capabilities of BART, we allow for maximum flexibility in modeling the dependence of survival times on covariates. In particular, we do not impose proportional hazards.

To elaborate, consider data in the usual form: \((t_i, \delta_i, x_i)\) where \(t_i\) is the event time, \(\delta_i\) is an indicator distinguishing events \((\delta = 1)\) from right-censoring \((\delta = 0)\), \(x_i\) is a vector of covariates, and \(i = 1, \ldots, N\) indexes subjects.

We denote the \(K\) distinct event/censoring times by \(0 < t_{(1)} < \cdots < t_{(K)} < \infty\) thus taking \(t_{(j)}\) to be the \(j^{th}\) order statistic among distinct observation times and, for convenience, \(t_{(0)} = 0\). Now consider event indicators \(y_{ij}\) for each subject \(i\) at each distinct time \(t_{(j)}\) up to and including the subject’s observation time \(t_i = t_{(n_i)}\) with \(n_i = \sum_j I[t_{(j)} \leq t_i]\). This means \(y_{ij} = 0\) if \(j < n_i\) and \(y_{in_i} = \delta_i\).

We then denote by \(p_{ij}\) the probability of an event at time \(t_{(j)}\) conditional on no previous event. We now write the model for \(y_{ij}\) as a nonparametric probit regression of \(y_{ij}\) on the time \(t_{(j)}\) and the covariates \(x_i\), and then utilize BART for binary responses. Specifically, \(y_{ij} = \delta_i I[t_i = t_{(j)}], j = 1, \ldots, n_i\); we have \(p_{ij} = F(\mu_{ij}), \mu_{ij} = \mu_0 + f(t_{(j)}, x_i)\) where \(F\) denotes the standard normal cdf (probit link). As in the binary response case, \(f\) is the sum of many tree models.

Usage

```r
surv.bart( x.train=matrix(0,0,0),
          y.train=NULL, times=NULL, delta=NULL,
          x.test=matrix(0,0,0),
```
K=rep(0,0,0), ntype=as.integer(factor(type, levels=c('wbart', 'pbart', 'lbart'))),
k=2, power=2, base=.95,
offset=rep(0,0,0), tau.num=c(NA, 3, 6)[ntype],
ntree=50, numcut=100, ndpost=1000, nskip=250,
keepevery = 10L,
printevery=100L,
id=rep(0,0,0), seed=99, mc.cores=2,
nice=19L
)
mc.surv.bart(x.train=matrix(0,0,0),
y.train=NULL, times=NULL, delta=NULL,
x.test=matrix(0,0,0),
K=rep(0,0,0), ntype=as.integer(factor(type, levels=c('wbart', 'pbart', 'lbart'))),
k=2, power=2, base=.95,
offset=rep(0,0,0), tau.num=c(NA, 3, 6)[ntype],
ntree=50, numcut=100, ndpost=1000, nskip=250,
keepevery = 10L,
printevery=100L,
id=rep(0,0,0), seed=99, mc.cores=2,
nice=19L
)
Arguments

**x.train**  
Explanatory variables for training (in sample) data.  
Must be a matrix with (as usual) rows corresponding to observations and columns to variables.  
surv.bart will generate draws of $f(t,x)$ for each $x$ which is a row of x.train  
(note that the definition of x.train is dependent on whether y.train has been specified; see below).

**y.train**  
Binary response dependent variable for training (in sample) data.  
If y.train is NULL, then y.train (x.train and x.test, if specified) are generated by a call to surv.pre.bart (which require that times and delta be provided: see below); otherwise, y.train (x.train and x.test, if specified) are utilized as given assuming that the data construction has already been performed.

**times**  
The time of event or right-censoring.  
If y.train is NULL, then times (and delta) must be provided.

**delta**  
The event indicator: 1 is an event while 0 is censored.  
If y.train is NULL, then delta (and times) must be provided.

**x.test**  
Explanatory variables for test (out of sample) data.  
Must be a matrix and have the same structure as x.train.  
surv.bart will generate draws of $f(t,x)$ for each $x$ which is a row of x.test.

**K**  
If provided, then coarsen times per the quantiles $1/K, 2/K, ..., K/K$.

**events**  
If provided, then use for the grid of time points.

**ztimes**  
If provided, then these columns of x.train (and x.test if any) are the times for time-dependent covariates. They will be transformed into time-dependent covariate sojourn times.

**zdelta**  
If provided, then these columns of x.train (and x.test if any) are the delta for time-dependent covariates. They will be transformed into time-dependent covariate binary events.

**sparse**  
Whether to perform variable selection based on a sparse Dirichlet prior rather than simply uniform; see Linero 2016.

**theta**  
Set theta parameter; zero means random.

**omega**  
Set omega parameter; zero means random.

**a**  
Sparse parameter for $Beta(a,b)$ prior: $0.5 \leq a \leq 1$ where lower values inducing more sparsity.

**b**  
Sparse parameter for $Beta(a,b)$ prior; typically, $b = 1$.

**rho**  
Sparse parameter: typically rho = p where p is the number of covariates under consideration.

**augment**  
Whether data augmentation is to be performed in sparse variable selection.
xinfo
You can provide the cutpoints to BART or let BART choose them for you. To provide them, use the xinfo argument to specify a list (matrix) where the items (rows) are the covariates and the contents of the items (columns) are the cutpoints.

usequants
If usequants=FALSE, then the cutpoints in xinfo are generated uniformly; otherwise, if TRUE, uniform quantiles are used for the cutpoints.

rm.const
Whether or not to remove constant variables.

type
Whether to employ Albert-Chib, 'pbart', or Holmes-Held, 'lbart'.

ntype
The integer equivalent of type where 'wbart' is 1, 'pbart' is 2 and 'lbart' is 3.

k
k is the number of prior standard deviations $f(t,x)$ is away from +/-3. The bigger k is, the more conservative the fitting will be.

power
Power parameter for tree prior.

base
Base parameter for tree prior.

offset
With binary BART, the centering is $P(Y = 1|X) = F(f(x) + offset)$ where offset defaults to $F^{-1}(\text{mean}(y.train))$. You can use the offset parameter to over-ride these defaults.

tau.num
The numerator in the tau definition, i.e., $\tau = \tau.num/(k*\sqrt{\text{ntree}})$.

ntree
The number of trees in the sum.

ndpost
The number of posterior draws returned.

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

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

keepevery
Every keepevery draw is kept to be returned to the user.
A “draw” will consist of values $f^*(t,x)$ at $x$ = rows from the train (optionally) and test data, where $f^*$ denotes the current draw of $f$.

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.

id
surv.bart only: unique identifier added to returned list.

seed
mc.surv.bart only: seed required for reproducible MCMC.

mc.cores
mc.surv.bart only: number of cores to employ in parallel.

nice
mc.surv.bart only: set the job niceness. The default niceness is 19: niceness goes from 0 (highest) to 19 (lowest).

Value
surv.bart returns an object of type survbart which is essentially a list. Besides the items listed below, the list has a binaryOffset component giving the value used, a times component giving the unique times, K which is the number of unique times, tx.train and tx.test, if any.
yhat.train  A matrix with \( \text{ndpost} \) rows and \( \text{ncol}(\text{x.train}) \) columns. Each row corresponds to a draw \( f^* \) from the posterior of \( f \) and each column corresponds to a row of \( \text{x.train} \). The \((i,j)\) value is \( f^*(t, x) \) for the \( i^{th} \) kept draw of \( f \) and the \( j^{th} \) row of \( \text{x.train} \). Burn-in is dropped.

yhat.test  Same as yhat.train but now the x’s are the rows of the test data.

surv.test  The survival function, \( S(t|x) \), where x’s are the rows of the test data.

yhat.train.mean  train data fits = mean of yhat.train columns.

yhat.test.mean  test data fits = mean of yhat.test columns.

surv.test.mean  mean of surv.test columns.

varcount  a matrix with \( \text{ndpost} \) rows and \( \text{ncol}(\text{x.train}) \) 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.

Note that yhat.train and yhat.test are \( f(t, x) + \text{binaryOffset} \). If you want draws of the probability \( P(Y = 1|t, x) \) you need to apply the normal cdf (\text{pnorm}) to these values.

Author(s)

Rodney Sparapani: \(<\text{rsparapa@mcw.edu}\>\)

References


See Also

\texttt{surv.pre.bart}

Examples

```r
## load survival package for the advanced lung cancer example
data(lung)
N <- length(lung$status)
```
table(lung$ph.karno, lung$pat.karno)

## if physician's KPS unavailable, then use the patient's
h <- which(is.na(lung$ph.karno))
lung$ph.karno[h] <- lung$pat.karno[h]

times <- lung$time
delta <- lung$status-1  ## lung$status: 1=censored, 2=dead
## delta: 0=censored, 1=dead

## this study reports time in days rather than weeks or months
## coarsening from days to weeks or months will reduce the computational burden
## times <- ceiling(times/30)
times <- ceiling(times/7)  ## weeks

table(times)
table(delta)

## matrix of observed covariates
x.train <- cbind(lung$sex, lung$age, lung$ph.karno)
## lung$sex: Male=1 Female=2
## lung$age: Age in years
## lung$ph.karno: Karnofsky performance score (dead=0: normal=100: by=10)
## rated by physician
dimnames(x.train)[[2]] <- c('M(1):F(2)', 'age(39:82)', 'ph.karno(50:100:10)')

table(x.train[, 1])
summary(x.train[, 2])
table(x.train[, 3])

## test BART with token run to ensure installation works
set.seed(99)
post <- surv.bart(x.train=x.train, times=times, delta=delta,
                   nskip=1, ndpost=1, keepevery=1)

## Not run:
## run one long MCMC chain in one process
## set.seed(99)
## post <- surv.bart(x.train=x.train, times=times, delta=delta, x.test=x.test)

## in the interest of time, consider speeding it up by parallel processing
## run "mc.cores" number of shorter MCMC chains in parallel processes
post <- mc.surv.bart(x.train=x.train, times=times, delta=delta,
                      mc.cores=8, seed=99)

pre <- surv.pre.bart(times=times, delta=delta, x.train=x.train,
                     x.test=x.train)

K <- pre$K
M <- nrow(post$yhat.train)
surv.pre.bart <- rbind(pre$tx.test, pre$tx.test)
pre$tx.test[, 2] <- c(rep(1, N*K), rep(2, N*K))
## sex pushed to col 2, since time is always in col 1

pred <- predict(post, newdata=pre$tx.test, mc.cores=8)

pd <- matrix(nrow=M, ncol=2*K)
for(j in 1:K) {
  h <- seq(j, N*K, by=K)
pd[, j] <- apply(pred$surv.test[, h], 1, mean)
pd[, j+K] <- apply(pred$surv.test[, h+N*K], 1, mean)
}

pd.mu <- apply(pd, 2, mean)
pd.025 <- apply(pd, 2, quantile, probs=0.025)
pd.975 <- apply(pd, 2, quantile, probs=0.975)

males <- 1:K
females <- males+K

plot(c(0, pre$times), c(1, pd.mu[males]), type='s', col='blue',
    ylim=0:1, ylab='S(t, x)', xlab='t (weeks)',
    main=paste('Advanced Lung Cancer ex. (BART::lung)',
               "Friedman's partial dependence function",
               'Male (blue) vs. Female (red)', sep='\n'))
lines(c(0, pre$times), c(1, pd.025[males]), col='blue', type='s', lty=2)
lines(c(0, pre$times), c(1, pd.975[males]), col='blue', type='s', lty=2)
lines(c(0, pre$times), c(1, pd.mu[females]), col='red', type='s')
lines(c(0, pre$times), c(1, pd.025[females]), col='red', type='s', lty=2)
lines(c(0, pre$times), c(1, pd.975[females]), col='red', type='s', lty=2)

## End(Not run)

---

**surv.pre.bart**

*Data construction for survival analysis with BART*

**Description**

Survival data contained in \((t, \delta, x)\) must be translated to data suitable for the BART survival analysis model; see surv.bart for more details.

**Usage**

```r
surv.pre.bart( times, delta, x.train=NULL, x.test=NULL,
               K=NULL, events=NULL, ztimes=NULL, zdelta=NULL )
```
Arguments

times  The time of event or right-censoring.
delta  The event indicator: 1 is an event while 0 is censored.
x.train  Explanatory variables for training (in sample) data.
         If provided, must be a matrix with (as usual) rows corresponding to observations
         and columns to variables.
x.test  Explanatory variables for test (out of sample) data.
         If provided, must be a matrix and have the same structure as x.train.
K  If provided, then coarsen times per the quantiles 1/K, 2/K, ..., K/K.
events  If provided, then use for the grid of time points.
ztimes  If provided, then these columns of x.train (and x.test if any) are the times
         for time-dependent covariates. They will be transformed into time-dependent
         covariate sojourn times.
zdelta  If provided, then these columns of x.train (and x.test if any) are the delta
         for time-dependent covariates. They will be transformed into time-dependent
         covariate binary events.

Value

surv.pre.bart returns a list. Besides the items listed below, the list has a times component giving
the unique times and K which is the number of unique times.

y.train  A vector of binary responses.
tx.train  A matrix with rows consisting of time and the covariates of the training data.
tx.test  A matrix with rows consisting of time and the covariates of the test data, if any.

Author(s)

Rodney Sparapani: <rsparapa@mcw.edu>

References


See Also

surv.bart
Examples

```r
## load the advanced lung cancer example
data(lung)

group <- which(is.na(lung[, 7])) ## remove missing row for ph.karno
times <- lung[group, 2]  ##lung$time
delta <- lung[group, 3]-1  ##lung$status: 1=censored, 2=dead
  ##delta: 0=censored, 1=dead

summary(times)
table(delta)

x.train <- as.matrix(lung[group, c(4, 5, 7)])  ## matrix of observed covariates
  ## lung$age: Age in years
  ## lung$sex: Male=1 Female=2
  ## lung$ph.karno: Karnofsky performance score (dead=0:normal=100:by=10)
  ## rated by physician

dimnames(x.train)[[2]] <- c('age(yr)', 'M(1):F(2)', 'ph.karno(0:100:10)')

summary(x.train[, 1])
table(x.train[, 2])
table(x.train[, 3])

x.test <- matrix(nrow=84, ncol=3)  ## matrix of covariate scenarios

dimnames(x.test)[[2]] <- dimnames(x.train)[[2]]

i <- 1

for(age in 5*(9:15)) for(sex in 1:2) for(ph.karno in 10*(5:10)) {
  x.test[i, ] <- c(age, sex, ph.karno)
  i <- i+1
}

pre <- surv.pre.bart(times=times, delta=delta, x.train=x.train, x.test=x.test)
str(pre)
```

transplant

Liver transplant waiting list

Description

Subjects on a liver transplant waiting list from 1990-1999, and their disposition: received a transplant, died while waiting, withdrew from the list, or censored.

Usage

data("transplant")
**Format**

A data frame with 815 observations on the following 6 variables.

- **age** age at addition to the waiting list
- **sex** m or f
- **abo** blood type: A, B, AB or O
- **year** year in which they entered the waiting list
- **futime** time from entry to final disposition
- **event** final disposition: censored, death, ltx or withdraw

**Details**

This represents the transplant experience in a particular region, over a time period in which liver transplant became much more widely recognized as a viable treatment modality. The number of liver transplants rises over the period, but the number of subjects added to the liver transplant waiting list grew much faster. Important questions addressed by the data are the change in waiting time, who waits, and whether there was an consequent increase in deaths while on the list.

Blood type is an important consideration. Donor livers from subjects with blood type O can be used by patients with A, B, AB or O blood types, whereas a donor liver from the other types will only be transplanted to a matching recipient.

Thus type O subjects on the waiting list are at a disadvantage, since the pool of competitors is larger for type O donor livers.

This data is of historical interest and provides a useful example of competing risks, but it has little relevance to current practice. Liver allocation policies have evolved and now depend directly on each individual patient’s risk and need, assessments of which are regularly updated while a patient is on the waiting list. The overall organ shortage remains acute, however.

**References**


---

**BART for continuous outcomes**

**Description**

BART is a Bayesian “sum-of-trees” model.

For a numeric response \( y \), we have \( y = f(x) + \epsilon \), where \( \epsilon \sim N(0, \sigma^2) \).

\( f \) is the sum of many tree models. The goal is to have very flexible inference for the unknown 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

wbart(
  x.train, y.train, x.test=matrix(0.0,0,0),
  sparse=FALSE, theta=0, omega=1,
  a=0.5, b=1, augment=FALSE, rho=NULL,
  xinfo=matrix(0.0,0,0), usequants=FALSE,
  cont=FALSE, rm.const=TRUE,
  sigest=NA, sigdf=3, sigquant=.90,
  k=2.0, power=2.0, base=.95,
  sigmaf=NA, lambda=NA,
  fmean=mean(y.train), w=rep(1,length(y.train)),
  ntree=200L, numcut=100L,
  ndpost=1000L, nskip=100L, keepevery=1L,
  nkeeptrain=ndpost, nkeeptest=ndpost,
  nkeeptestmean=ndpost, nkeeptreedraws=ndpost,
  printevery=100L, transposed=FALSE
)

Arguments

x.train  Explanatory variables for training (in sample) data.
          May be a matrix or a data frame, with (as usual) rows corresponding to obser-
          vations 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.
          wbart will generate draws of f(x) for each x which is a row of x.train.

y.train  Continuous dependent variable for training (in sample) data.

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

sparse   Whether to perform variable selection based on a sparse Dirichlet prior rather
          than simply uniform; see Linero 2016.

theta    Set theta parameter; zero means random.

omega    Set omega parameter; zero means random.

a        Sparse parameter for Beta(a,b) prior: 0.5 <= a <= 1 where lower values
          inducing more sparsity.

b        Sparse parameter for Beta(a,b) prior; typically, b = 1.

rho      Sparse parameter: typically rho = p where p is the number of covariates under
          consideration.

augment  Whether data augmentation is to be performed in sparse variable selection.

xinfo    You can provide the cutpoints to BART or let BART choose them for you. To
          provide them, use the xinfo argument to specify a list (matrix) where the items
          (rows) are the covariates and the contents of the items (columns) are the cut-
          points.
usequants  If usequants=FALSE, then the cutpoints in xinfo are generated uniformly; otherwise, if TRUE, uniform quantiles are used for the cutpoints.
cont  Whether or not to assume all variables are continuous.
rm.const  Whether or not to remove constant variables.
sigtest  The prior for the error variance ($\sigma^2$) 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 sigtest=NA then the rough estimate will be the usual least squares estimator. Otherwise the supplied value will be used.
sigdf  Degrees of freedom for error variance prior.
sigquant  The quantile of the prior that the rough estimate (see sigest) is placed at. The closer the quantile is to 1, the more aggressive the fit will be as you are putting more prior weight on error standard deviations ($\sigma$) less than the rough estimate.
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 -.5 to .5.

k is the number of prior standard deviations $f(x)$ is away from +/-3.
The bigger k is, the more conservative the fitting will be.
power  Power parameter for tree prior.
base  Base parameter for tree prior.
sigmaf  The SD of f.
lambda  The scale of the prior for the variance.
fmean  BART operates on y.train centered by fmean.
w  Vector of weights which multiply the standard deviation.
ntree  The number of trees in the sum.
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.
ndpost  The number of posterior draws returned.
nskip  Number of MCMC iterations to be treated as burn in.
nkeeptrain  Number of MCMC iterations to be returned for train data.
nkeepittest  Number of MCMC iterations to be returned for test data.
nkeepetestmean  Number of MCMC iterations to be returned for test mean.
nkeepetreedraws  Number of MCMC iterations to be returned for tree draws.
printevery  As the MCMC runs, a message is printed every printevery draws.
keepevery  Every keepevery draw is kept to be returned to the user.
transposed  When running wbart in parallel, it is more memory-efficient to transpose x.train and x.test, if any, prior to calling mc.wbart.
Details

BART is a Bayesian MCMC method. At each MCMC iteration, we produce a draw from the joint posterior \( (f, \sigma) | (x, y) \) in the numeric \( y \) 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 \( \sigma^* \) 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

\texttt{wbart} returns an object of type \texttt{wbart} which is essentially a list. In the numeric \( y \) case, the list has components:

- \texttt{yhat.train} A matrix with \( n \) 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.
- \texttt{yhat.test} Same as \texttt{yhat.train} but now the x’s are the rows of the test data.
- \texttt{yhat.train.mean} train data fits = mean of \texttt{yhat.train} columns.
- \texttt{yhat.test.mean} test data fits = mean of \texttt{yhat.test} columns.
- \texttt{sigma} post burn in draws of sigma, length = \( n \).
- \texttt{first.sigma} burn-in draws of sigma.
- \texttt{varcount} a matrix with \( n \) 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.
- \texttt{sigest} The rough error standard deviation (\( \sigma \)) used in the prior.

Author(s)

Robert McCulloch: <robert.e.mcculloch@gmail.com>, Rodney Sparapani: <rsparapa@mcw.edu>.

References


## simulate data (example from Friedman MARS paper)

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

```r
sigma = 1.0  # y = f(x) + sigma*z , z~N(0,1)
n = 100  #number of observations
```

```r
set.seed(99)
x=matrix(runif(n*10),n,10) #10 variables, only first 5 matter
```

```r
Ey = f(x)
y=Ey+sigma*runif(n)
```

```r
lmFit = lm(y~.,data.frame(x,y)) #compare lm fit to BART later
```

```r
## test BART with token run to ensure installation works
set.seed(99)
bartFit = wbart(x,y,nskip=5,ndpost=5)
```

```r
## test BART with token run to ensure installation works
set.seed(99)
bartFit = wbart(x,y,nskip=5,ndpost=5)
```

```r
## compare BART fit to linear matter and truth = Ey
fitmat = cbind(y,Ey,lmFit$fitted,bartFit$yhat.train.mean)
colnames(fitmat) = c('y','Ey','lm','bart')
print(cor(fitmat))
```

```r
## End(Not run)
```

### Description

A matrix containing a 20% random sample of the testing set for a real data example of recurrent events survival analysis. There are 100 patients in the cohort: 50 in the training set and 50 in the testing set. See the Reference below (and the References therein) for more detailed information; a brief synopsis follows.

xdm20.test contains both the training set and the testing set. There are 798 unique time points so there are 50*798=39900 rows of the training set followed by 50*798=39900 rows of the testing set. For patient’s who died prior to the end of follow-up, their external factors are last value carried forward. Therefore, we can use xdm20.test to estimate the cumulative hazard for all patients for all time points.
The full data set, `xdm.test`, can be obtained online at [http://www.mcw.edu/FileLibrary/Groups/Biostatistics/TechReports/TechReports5175/tr064.tar](http://www.mcw.edu/FileLibrary/Groups/Biostatistics/TechReports/TechReports5175/tr064.tar). There are 488 patients in the full cohort: 235 in the training set and 253 in the testing set.

`xdm.test` contains both the training set and the testing set. There are 798 unique time points so there are 235*798=187530 rows of the training set followed by 253*798=201894 rows of the testing set. For patient’s who died prior to the end of follow-up, their external factors are last value carried forward.

**Usage**

```r
data(xdm20.test)
```

**References**


**See Also**

`xdm20.train`

**Examples**

```r
data(xdm20.test)
head(xdm20.test[, 1:10])
```

---

**Description**

A matrix containing a 20% random sample of the training set for a real data example of recurrent events survival analysis. There are 100 patients in the cohort: 50 in the training set and 50 in the testing set. The full data set, `xdm.train`, can be obtained online at [http://www.mcw.edu/FileLibrary/Groups/Biostatistics/TechReports/TechReports5175/tr064.tar](http://www.mcw.edu/FileLibrary/Groups/Biostatistics/TechReports/TechReports5175/tr064.tar). There are 488 patients in the full cohort: 235 in the training set and 253 in the testing set. See the Reference below (and the References therein) for more detailed information; a brief synopsis follows.

We explored the hospital admissions for a cohort of patients with diabetes cared for by the Froedtert and Medical College of Wisconsin health network. These patients were identified via their Electronic Health Records (EHR) which include vital signs, diagnoses, procedures, laboratory values, pharmacy orders and billing data. This human subjects research and de-identified data release was approved by the Medical College of Wisconsin and Froedtert Hospital joint Institutional Review Board. To maintain patient privacy, roughly one fourth of patients were randomly sampled for inclusion as well as other de-identification procedures.
We identified likely incident diabetes mellitus type 2 patients by tabulating their first diagnosis code of primary diabetes (ICD-9 codes 250.x0 and 250.x2) in 2006 or 2007, i.e., no such codes were found for these patients prior to 2006 for as far back as each patient’s records go which is variable. We restricted the population to adults aged 21 to 90 by 01/01/2008. Among the patients treated in this health system, the vast majority were racially self-identified as either white or black so our inclusion criteria is restricted to these groups. Since our interest is in patients with primary diabetes, we excluded those patients who were diagnosed with either secondary diabetes or gestational diabetes.

For this cohort, we identified every hospital admission between 01/01/2008 and 12/31/2012. For convenience, follow-up begins on 01/01/2008, rather than from each patient’s actual incident diagnosis date which varied over the preceding 2 years. Following all patients concurrently allows us to temporally adapt, via our model, for seasonal/epidemic hospital admissions such as the H1N1 influenza outbreak in the US from April to June 2009.

We investigated the following risk factors: gender, race, age, insurance status (commercial, government or other), diabetes therapy (insulin, metformin and/or sulfonylurea), health care charges, relative value units (RVU), vital signs, laboratory values, comorbidity/complication diagnoses and procedures/surgeries (we will refer to vital signs and laboratory values collectively as signs; and comorbidity/complication diagnoses and procedures/surgeries collectively as conditions). In total, we considered 85 covariates of which 82 are external factors as described above and three are temporal factors: time, $t$, the counting process, $N_i(t^-)$, and the sojourn time, $v_i(t)$. Among these potential predictors only gender, race and age are time-independent. The rest are defined as last value carried forward.

For insulin, metformin and sulfonylurea, we only had access to prescription orders (rather than prescription fills) and self-reported current status of prescription therapy during clinic office visits. Since, generally, orders are only required after every three fills, and each fill can be for up to 90 days, we define insulin, metformin and sulfonylurea as binary indicators which are one if there exists an order or current status indication within the prior 270 days; otherwise zero.

Health care charges and relative value units (RVU) are measures related to the services and procedures delivered. However, they are so closely related that recent charges/RVUs are of no practical value in this analysis. For example, just prior to a patient’s hospital admission on a non-emergent basis, they often have a series of diagnostic tests and imaging. Similarly, for an emergent admission, the patient is often seen in the emergency department just prior to admission where similar services are conducted. We do not consider these charges/RVUs predictive of an admission because we are interested in identifying preventive opportunities. Therefore, we investigate charges/RVUs that are the sum total of the following moving windows of days prior to any given date: 31 to 90, 91 to 180, 181 to 300.

For many patients, some signs were not available for a given date so they were imputed; similarly, if a sign was not observed within the last 180 days, then it was imputed (except for height which never expires, weight extended to 365 days and body mass index which is a deterministic function of the two). We utilized the Sequential BART missing imputation method. However, instead of creating several imputed data sets, we imputed a new sign at each date when it was missing, i.e., in order to properly address uncertainty within one data set, a new value was imputed for each date that it was missing and never carried forward.

Conditions are binary indicators which are zero until the date of the first coding and then they are one from then on. Based on clinical rationale, we identified 26 conditions (23 comorbidities and 3 procedures/surgeries) which are potential risk factors for a hospital admission many of which are possible complications of diabetes; besides clinical merit, these conditions were chosen since they
are present in more than just a few subjects so that they may be informative. Similarly, we employed 15 general conditions which are the Charlson diagnoses and 18 general conditions from the RxRisk adult diagnoses which are defined by prescription orders. Seven conditions are a composite of diagnosis codes and prescription orders.

Usage

data(xdm20.train)

References


See Also

xdm20.test

Examples

data(xdm20.train)
head(xdm20.train[, 1:10])

---

**ydm20.train**

A data set used in example of recur.bart.

**Description**

Two vectors containing the training and testing set outcomes for a 20% random sample for a real data example of recurrent events survival analysis. There are 100 patients in the cohort: 50 in the training set and 50 in the testing set. See the Reference below (and the References therein) for more detailed information; a brief synopsis follows.

**ydm20.train** contains the training set only. **ydm20.test** is provided for completeness; it contains both the training set and the testing set. There are 798 unique time points so there are 50*798=39900 rows of the training set followed by 50*798=39900 rows of the testing set.

The full data sets, **ydm.train** and **ydm.test**, can be obtained online at [http://www.mcw.edu/FileLibrary/Groups/Biostatistics/TechReports/TechReports5175/tr064.tar](http://www.mcw.edu/FileLibrary/Groups/Biostatistics/TechReports/TechReports5175/tr064.tar). There are 488 patients in the full cohort: 235 in the training set and 253 in the testing set. **ydm.train** contains the training set only. **ydm.test** contains both the training set and the testing set. There are 798 unique time points so there are 235*798=187530 rows of the training set followed by 253*798=201894 rows of the testing set.

Usage

data(ydm20.train)
data(ydm20.test)
References


See Also

xdm20.train

Examples

```r
data(ydm20.train)
data(ydm20.test)
table(ydm20.train)
table(ydm20.test)
```
Index

*Topic OpenMP
  mc.cores.openmp, 62
*Topic convergence diagnostics
  gewekediag, 42
  spectrum0ar, 135
  strats, 137
*Topic data construction
  crisk.pre.bart, 22
  recur.pre.bart, 124
  surv.pre.bart, 144
*Topic datasets
  ACTG175, 8
  alligator, 10
  arg, 12
  bladder, 14
  leukemia, 50
  lung, 52
  transplant, 146
  xdm20.test, 151
  xdm20.train, 152
  ydm20.train, 154
*Topic multi-threading
  mc.cores.openmp, 62
*Topic nonlinear
  abart, 3
  gbart, 32
  gbmm, 37
  lbart, 44
  mbart, 52
  mbart2, 57
  mc.crisk.pwbart, 62
  mc.crisk2.pwbart, 66
  mc.lbart, 69
  mc.pbart, 74
  mc.surv.pwbart, 78
  mc.wbart, 82
  pbart, 88
  predict.crisk2bart, 94
  predict.criskbart, 97
  predict.lbart, 99
  predict.mbart, 102
  predict.pbart, 105
  predict.recurbart, 108
  predict.survbart, 111
  predict.wbart, 114
  pwbart, 116
  rs.pbart, 128
  wbart, 147
*Topic nonparametric recurrent
events model
  recur.bart, 118
*Topic nonparametric survival model
  crisk.bart, 16
  crisk2.bart, 24
  surv.bart, 138
*Topic nonparametric
  abart, 3
  gbart, 32
  gbmm, 37
  lbart, 44
  mbart, 52
  mbart2, 57
  mc.crisk.pwbart, 62
  mc.crisk2.pwbart, 66
  mc.lbart, 69
  mc.pbart, 74
  mc.surv.pwbart, 78
  mc.wbart, 82
  pbart, 88
  predict.crisk2bart, 94
  predict.criskbart, 97
  predict.lbart, 99
  predict.mbart, 102
  predict.pbart, 105
  predict.recurbart, 108
  predict.survbart, 111
  predict.wbart, 114
  pwbart, 116
<table>
<thead>
<tr>
<th>Topic</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonproportional hazards</td>
<td>rs.pbart, 128</td>
</tr>
<tr>
<td>variable selection</td>
<td>wbart, 147</td>
</tr>
<tr>
<td>mc.pbart, 74</td>
<td>mc.lbart, 69</td>
</tr>
<tr>
<td></td>
<td>mc.pbart, 74</td>
</tr>
<tr>
<td></td>
<td>mc.mc.lbart, 78</td>
</tr>
<tr>
<td></td>
<td>mc.mc.pbart, 74</td>
</tr>
<tr>
<td></td>
<td>mc.mc.surv.pbart, 78</td>
</tr>
<tr>
<td></td>
<td>mc.mc.wbart, 82</td>
</tr>
<tr>
<td></td>
<td>pwbart, 88</td>
</tr>
<tr>
<td></td>
<td>predict.crisk2bart, 94</td>
</tr>
<tr>
<td></td>
<td>predict.criskbart, 97</td>
</tr>
<tr>
<td></td>
<td>predict.lbart, 99</td>
</tr>
<tr>
<td></td>
<td>predict.mbart, 102</td>
</tr>
<tr>
<td></td>
<td>predict.pbart, 105</td>
</tr>
<tr>
<td></td>
<td>predict.recurbart, 108</td>
</tr>
<tr>
<td></td>
<td>predict.survbart, 111</td>
</tr>
<tr>
<td></td>
<td>predict.wbart, 114</td>
</tr>
<tr>
<td></td>
<td>pwbart, 116</td>
</tr>
<tr>
<td></td>
<td>rs.pbart, 128</td>
</tr>
<tr>
<td></td>
<td>wbart, 147</td>
</tr>
<tr>
<td>package</td>
<td>BART-package, 3</td>
</tr>
<tr>
<td>parallel</td>
<td>mc.cores.openmp, 62</td>
</tr>
<tr>
<td>regression</td>
<td>abart, 3</td>
</tr>
<tr>
<td></td>
<td>gbart, 32</td>
</tr>
<tr>
<td></td>
<td>gbmm, 37</td>
</tr>
<tr>
<td></td>
<td>lbart, 44</td>
</tr>
<tr>
<td></td>
<td>mbart, 52</td>
</tr>
<tr>
<td></td>
<td>mbart2, 57</td>
</tr>
<tr>
<td></td>
<td>mc.crisk2.pbart, 62</td>
</tr>
<tr>
<td></td>
<td>mc.crisk.pbart, 66</td>
</tr>
<tr>
<td></td>
<td>mc.lbart, 69</td>
</tr>
<tr>
<td></td>
<td>mc.pbart, 74</td>
</tr>
<tr>
<td></td>
<td>mc.surv.pbart, 78</td>
</tr>
<tr>
<td></td>
<td>mc.wbart, 82</td>
</tr>
<tr>
<td></td>
<td>predict.crisk2bart, 94</td>
</tr>
<tr>
<td></td>
<td>predict.criskbart, 97</td>
</tr>
<tr>
<td></td>
<td>predict.lbart, 99</td>
</tr>
<tr>
<td></td>
<td>predict.mbart, 102</td>
</tr>
<tr>
<td></td>
<td>predict.pbart, 105</td>
</tr>
<tr>
<td></td>
<td>predict.recurbart, 108</td>
</tr>
<tr>
<td></td>
<td>predict.survbart, 111</td>
</tr>
<tr>
<td></td>
<td>predict.wbart, 114</td>
</tr>
<tr>
<td></td>
<td>pwbart, 116</td>
</tr>
<tr>
<td></td>
<td>rs.pbart, 128</td>
</tr>
<tr>
<td></td>
<td>wbart, 147</td>
</tr>
<tr>
<td>survival</td>
<td>bladder, 14</td>
</tr>
<tr>
<td></td>
<td>draw_lambda_i, 31</td>
</tr>
<tr>
<td>tree</td>
<td>abart, 3</td>
</tr>
<tr>
<td></td>
<td>gbart, 32</td>
</tr>
<tr>
<td></td>
<td>gbmm, 37</td>
</tr>
<tr>
<td></td>
<td>lbart, 44</td>
</tr>
<tr>
<td></td>
<td>mbart, 52</td>
</tr>
<tr>
<td></td>
<td>mbart2, 57</td>
</tr>
<tr>
<td></td>
<td>gbart, 32, 56, 61</td>
</tr>
<tr>
<td></td>
<td>gbmm, 37</td>
</tr>
<tr>
<td></td>
<td>gewekediag, 42, 136</td>
</tr>
<tr>
<td></td>
<td>lbart, 31, 44, 73, 135</td>
</tr>
<tr>
<td></td>
<td>leukemia, 50</td>
</tr>
<tr>
<td></td>
<td>lung, 52, 137</td>
</tr>
<tr>
<td></td>
<td>mbart, 52, 104</td>
</tr>
</tbody>
</table>
INDEX

mbart2, 57, 104
mc.abart (abart), 3
mc.cores.openmp, 62, 95, 98, 101, 107, 110, 112, 115
mc.crisk.bart, 64, 98
mc.crisk.bart (crisk.bart), 16
mc.crisk.pwbart, 21, 62, 98
mc.crisk2.bart, 68, 95
mc.crisk2.bart (crisk2.bart), 24
mc.gbart (gbart), 32
mc.gbart (gbart), 37
mc.mbart (mbart), 52
mc.mbart2 (mbart2), 57
mc.pbart, 74, 131
mc.pwbart, 115
mc.pwbart (pwbart), 116
mc.recur.bart, 110
mc.recur.bart (recur.bart), 118
mc.recur.pwbart, 110, 122
mc.recur.pwbart (mc.surv.pwbart), 78
mc.surv.bart, 101, 107, 112
mc.surv.bart (surv.bart), 138
mc.surv.pwbart, 78, 101, 107, 112
mc.wbart, 82, 87, 115
mc.wbart.gse, 86

pbart, 36, 41, 77, 88, 135, 151
predict.crisk2bart, 29, 94
predict.criskbart, 21, 96
predict.pbart, 99
predict.mbart, 102
predict.mbart2 (predict.mbart), 102
predict.pbart, 105
predict.recurbart, 108, 122
predict.survbart, 111
predict.wbart, 114, 117
pwbart, 62, 64, 68, 80, 115, 116

recur.bart, 110, 118, 125
recur.pre.bart, 122, 124
recur.pwbart, 110, 122
recur.pwbart (mc.surv.pwbart), 78
rs.pbart, 128, 138
rtgamma, 133
rtnorm, 37, 134
spectrum0ar, 43, 135
srstepwise, 136
stratrs, 137
surv.bart, 101, 107, 112, 138, 145
surv.pre.bart, 29, 142, 144
surv.pwbart, 101, 107, 112
surv.pwbart (mc.surv.pwbart), 78
transplant, 146
wbart, 7, 41, 48, 85, 91, 115, 117, 147
xdm20.test, 151
xdm20.train, 152
ydm20.test (ydm20.train), 154
ydm20.train, 154