Package ‘bst’

October 12, 2022

Type Package

Title Gradient Boosting

Version 0.3-23

Date 2020-10-20

Author Zhu Wang [aut, cre] (<https://orcid.org/0000-0002-0773-0052>),
Torsten Hothorn [ctb]

Maintainer Zhu Wang <wangz1@uthscsa.edu>


Imports rpart, methods, foreach, doParallel, gbm

Suggests hdi, pROC, R.rsp, knitr, gdata

VignetteBuilder R.rsp, knitr

License GPL (>= 2)

LazyLoad yes

NeedsCompilation no

Repository CRAN

Date/Publication 2020-11-09 22:10:07 UTC

R topics documented:

bst ............................................................. 2
bst.sel .................................................... 4
bst_control ............................................. 5
cv.bst ................................................... 7
cv.mada .................................................. 9
cv.mbst .................................................. 10
cv.mhingebst ......................................... 11
cv.mhingeova ......................................... 12
Gradient boosting for optimizing loss functions with componentwise linear, smoothing splines, tree
models as base learners.

Usage
bst(x, y, cost = 0.5, family = c("gaussian", "hinge", "hinge2", "binom", "expo",
"poisson", "tgaussianDC", "tgaussianDC", "binomDC", "binomdDC", "texpoDC", "texpoDC", "texpoDC", "texpoDC",
huber", "thuberDC", "clossR", "clossRMM", "closs", "gloss", "qloss", "clossMM",
"glossMM", "qlossMM", "lar"), ctrl = bst_control(), control.tree = list(maxdepth = 1),
learner = c("ls", "sm", "tree"))
## S3 method for class 'bst'
print(x, ...)## S3 method for class 'bst'
predict(object, newdata=NULL, newy=NULL, mstop=NULL,
type=c("response", "all.res", "class", "loss", "error"), ...)
## S3 method for class 'bst'
plot(x, type = c("step", "norm"), ...)
## S3 method for class 'bst'
coef(object, which=object$ctrl$mstop, ...)
## S3 method for class 'bst'
fpartial(object, mstop=NULL, newdata=NULL)

Arguments

x a data frame containing the variables in the model.
y vector of responses. y must be in \{1,-1\} for family = "hinge".
cost price to pay for false positive, 0 < cost < 1; price of false negative is 1-cost.
family  A variety of loss functions. family = "hinge" for hinge loss and family="gaussian" for squared error loss. Implementing the negative gradient corresponding to the loss function to be minimized. For hinge loss, +1/-1 binary responses is used.

ctrl  an object of class bst_control.
type  type of prediction or plot, see predict, plot
control.tree  control parameters of rpart.
learner  a character specifying the component-wise base learner to be used: lls linear models, sm smoothing splines, tree regression trees.
object  class of bst.
newdata  new data for prediction with the same number of columns as x.
newy  new response.
mstop  boosting iteration for prediction.
which  at which boosting mstop to extract coefficients.
...  additional arguments.

Details

Boosting algorithms for classification and regression problems. In a classification problem, suppose $f$ is a classifier for a response $y$. A cost-sensitive or weighted loss function is

$$ L(y, f, cost) = l(y, f, cost) \max(0, (1 - yf)) $$

For family="hinge",

$$ l(y, f, cost) = 1 - cost, \text{if } y = +1; \text{ cost, if } y = -1 $$

For family="hinge2", $l(y,f,cost)= 1, \text{if } y = +1 \text{ and } f > 0 ; = 1\text{-cost, if } y = +1 \text{ and } f < 0; = \text{cost, if } y = -1 \text{ and } f > 0; = 1, \text{if } y = -1 \text{ and } f < 0$.

For twin boosting if twinboost=TRUE, there are two types of adaptive boosting if learner="ls": for twintype=1, weights are based on coefficients in the first round of boosting; for twintype=2, weights are based on predictions in the first round of boosting. See Buehlmann and Hothorn (2010).

Value

An object of class bst with print, coef, plot and predict methods are available for linear models. For nonlinear models, methods print and predict are available.

x, y, cost, family, learner, control.tree, maxdepth

These are input variables and parameters

ctrl  the input ctrl with possible updated fk if family="thingeDC", "tbinomDC", "binomDC"
yhat  predicted function estimates
ens  a list of length mstop. Each element is a fitted model to the pseudo residuals, defined as negative gradient of loss function at the current estimated function
ml.fit  the last element of ens
ensemble  a vector of length mstop. Each element is the variable selected in each boosting step when applicable
xselect  selected variables in mstop
coef  estimated coefficients in each iteration. Used internally only
Author(s)
Zhu Wang

References


See Also
cv.bst for cross-validated stopping iteration. Furthermore see bst_control

Examples
x <- matrix(rnorm(100*5),ncol=5)
c <- 2*x[,1]
p <- exp(c)/(exp(c)+exp(-c))
y <- rbinom(100,1,p)
y[y != 1] <- -1
x <- as.data.frame(x)
dat.m <- bst(x, y, ctrl = bst_control(mstop=50), family = "hinge", learner = "ls")
predict(dat.m)
dat.m1 <- bst(x, y, ctrl = bst_control(twinboost=TRUE, coefir=coef(dat.m), xselect.init = dat.m$xselect, mstop=50))
dat.m2 <- rbst(x, y, ctrl = bst_control(mstop=50, s=0, trace=TRUE), rfamily = "thinge", learner = "ls")
predict(dat.m2)

bst.sel  Function to select number of predictors

Description
Function to determine the first q predictors in the boosting path, or perform (10-fold) cross-validation and determine the optimal set of parameters

Usage
bst.sel(x, y, q, type=c("firstq", "cv"), ...)

Arguments
x  Design matrix (without intercept).
y  Continuous response vector for linear regression
q  Maximum number of predictors that should be selected if type="firstq".
type if type="firstq", return the first q predictors in the boosting path. if type="cv", perform (10-fold) cross-validation and determine the optimal set of parameters
... Further arguments to be passed to bst.cv.bst.

Details
Function to determine the first q predictors in the boosting path, or perform (10-fold) cross-validation and determine the optimal set of parameters. This may be used for p-value calculation. See below.

Value
Vector of selected predictors.

Author(s)
Zhu Wang

Examples
## Not run:
x <- matrix(rnorm(100*100), nrow = 100, ncol = 100)
y <- x[,1] * 2 + x[,2] * 2.5 + rnorm(100)
se1 <- bst.sel(x, y, q=10)
library("hdi")
fit.multi <- hdi(x, y, method = "multi.split", 
model.selector =bst.sel, 
args.model.selector=list(type="firstq", q=10))
fit.multi
fit.multi$pval[1:10] ## the first 10 p-values
fit.multi <- hdi(x, y, method = "multi.split", 
model.selector =bst.sel, 
args.model.selector=list(type="cv"))
fit.multi
fit.multi$pval[1:10] ## the first 10 p-values
## End(Not run)
Arguments

mstop an integer giving the number of boosting iterations.

nu a small number (between 0 and 1) defining the step size or shrinkage parameter.

twinboost a logical value: TRUE for twin boosting.

twintype for twinboost=TRUE only. For learner="ls", if twintype=1, twin boosting with weights from magnitude of coefficients in the first round of boosting. If twintype=2, weights are correlations between predicted values in the first round of boosting and current predicted values. For learners not componentwise least squares, twintype=2.

threshold if threshold="adaptive", the estimated function ctrl$fk is updated in every boosting step. Otherwise, no update for ctrl$fk in boosting steps. Only used in robust nonconvex loss function.

f.init the estimate from the first round of twin boosting. Only useful when twinboost=TRUE and learner="sm" or "tree".

twintype for twinboost=TRUE only. For learner="ls", if twintype=1, twin boosting with weights from magnitude of coefficients in the first round of boosting. If twintype=2, weights are correlations between predicted values in the first round of boosting and current predicted values. For learners not componentwise least squares, twintype=2.

threshold if threshold="adaptive", the estimated function ctrl$fk is updated in every boosting step. Otherwise, no update for ctrl$fk in boosting steps. Only used in robust nonconvex loss function.

f.init the estimate from the first round of twin boosting. Only useful when twinboost=TRUE and learner="sm" or "tree".

xselect.init the variable selected from the first round of twin boosting. Only useful when twinboost=TRUE.

center a logical value: TRUE to center covariates with mean.

trace a logical value for printout of more details of information during the fitting process.

numsample number of random sample variable selected in the first round of twin boosting. This is potentially useful in the future implementation.

df degree of freedom used in smoothing splines.

s,q nonconvex loss tuning parameter s or frequency q of outliers for robust regression and classification. If s is missing but q is available, s may be computed as the 1-q quantile of robust loss values using conventional software.

sh, qh threshold value or frequency qh of outliers for Huber regression family="huber" or family="rhuberDC". For family="huber", if sh is not provided, sh is then updated adaptively with the median of y-yhat where yhat is the estimated y in the last boosting iteration. For family="rhuberDC", if sh is missing but qh is available, sh may be computed as the 1-qh quantile of robust loss values using conventional software.

fk predicted values at an iteration in the MM algorithm

start a logical value, if start=TRUE and fk is a vector of values, then bst iterations begin with fk. Otherwise, bst iterations begin with the default values. This can be useful, for instance, in rbst for the MM boosting algorithm.

iter number of iteration in the MM algorithm

intercept logical value, if TRUE, estimation of intercept with linear predictor model

trun logical value, if TRUE, predicted value in each boosting iteration is truncated at -1, 1, for family="closs" in bst and rfamily="closs" in rbst
Details

Objects to specify parameters of the boosting algorithms implemented in bst, via the ctrl argument. The s value is for robust nonconvex loss where smaller s value is more robust to outliers with family="closs", "tbinom", "thinge", "tbinomd", and larger s value more robust with family="clossR", "gloss", "qloss".

For family="closs", if s=2, the loss is similar to the square loss; if s=1, the loss function is an approximation of the hinge loss; for smaller values, the loss function approaches the 0-1 loss function if s<1, the loss function is a nonconvex function of the margin.


Value

An object of class bst_control, a list. Note fk may be updated for robust boosting.

See Also

bst

Arguments

x a data frame containing the variables in the model.
y vector of responses. y must be in {1, -1} for binary classifications.
K K-fold cross-validation
cost price to pay for false positive, 0 < cost < 1; price of false negative is 1-cost.
family family = "hinge" for hinge loss and family="gaussian" for squared error loss.
learner a character specifying the component-wise base learner to be used: ls linear models, sm smoothing splines, tree regression trees.

ctrl an object of class bst_control.

type cross-validation criteria. For type="loss", loss function values and type="error" is misclassification error.

plot.it a logical value, to plot the estimated loss or error with cross validation if TRUE.

main title of plot

se a logical value, to plot with standard errors.

n.cores The number of CPU cores to use. The cross-validation loop will attempt to send different CV folds off to different cores.

... additional arguments.

Value

object with

residmat empirical risks in each cross-validation at boosting iterations

mstop boosting iteration steps at which CV curve should be computed.

cv The CV curve at each value of mstop

cv.error The standard error of the CV curve

family loss function types

...

See Also

bst

Examples

## Not run:
x <- matrix(rnorm(100*5),ncol=5)
c <- 2*x[,1]
p <- exp(c)/(exp(c)+exp(-c))
y <- rbinom(100,1,p)
y[y != 1] <- -1
x <- as.data.frame(x)
cv.bst(x, y, ctrl = bst_control(mstop=50), family = "hinge", learner = "ls", type="loss")
cv.bst(x, y, ctrl = bst_control(mstop=50), family = "hinge", learner = "ls", type="error")
dat.m <- bst(x, y, ctrl = bst_control(mstop=50), family = "hinge", learner = "ls")
dat.ml <- cv.bst(x, y, ctrl = bst_control(twinboost=TRUE, coefir=coef(dat.m), xselect.init = dat.m$xselect, mstop=50), family = "hinge", learner="ls")

## End(Not run)
Cross-Validation for one-vs-all AdaBoost with multi-class problem

Description

Cross-validated estimation of the empirical misclassification error for boosting parameter selection.

Usage

cv.mada(x, y, balance=FALSE, K=10, nu=0.1, mstop=200, interaction.depth=1, trace=FALSE, plot.it = TRUE, se = TRUE, ...)

Arguments

x  a data matrix containing the variables in the model.
y  vector of multi class responses. y must be an integer vector from 1 to C for C class problem.
balance  logical value. If TRUE, The K parts were roughly balanced, ensuring that the classes were distributed proportionally among each of the K parts.
K  K-fold cross-validation
nu  a small number (between 0 and 1) defining the step size or shrinkage parameter.
mstop  number of boosting iteration.
interaction.depth  used in gbm to specify the depth of trees.
trace  if TRUE, iteration results printed out.
plot.it  a logical value, to plot the cross-validation error if TRUE.
se  a logical value, to plot with 1 standard deviation curves.
...  additional arguments.

Value

object with
residmat  empirical risks in each cross-validation at boosting iterations
fraction  abscissa values at which CV curve should be computed.
cv  The CV curve at each value of fraction
cv.error  The standard error of the CV curve
...

See Also

mada
Cross-Validation forMulti-class Boosting

**Description**

Cross-validated estimation of the empirical multi-class loss for boosting parameter selection.

**Usage**

```r
cv.mbst(x, y, balance=FALSE, K = 10, cost = NULL,
family = c("hinge","hinge2","thingeDC", "closs", "clossMM"),
learner = c("tree", "ls", "sm"), ctrl = bst_control(),
type = c("loss","error"), plot.it = TRUE, se = TRUE, n.cores=2, ...)
```

**Arguments**

- `x`: a data frame containing the variables in the model.
- `y`: vector of responses. `y` must be integers from 1 to `C` for `C` class problem.
- `balance`: logical value. If TRUE, The K parts were roughly balanced, ensuring that the classes were distributed proportionally among each of the K parts.
- `K`: K-fold cross-validation
- `cost`: price to pay for false positive, 0 < cost < 1; price of false negative is 1-cost.
- `family`: family = "hinge" for hinge loss. "hinge2" is a different hinge loss
- `learner`: a character specifying the component-wise base learner to be used: ls linear models, sm smoothing splines, tree regression trees.
- `ctrl`: an object of class `bst_control`.
- `type`: for family="hinge", type="loss" is hinge risk. For family="thingeDC", type="loss"
- `plot.it`: a logical value, to plot the estimated risks if TRUE.
- `se`: a logical value, to plot with standard errors.
- `n.cores`: The number of CPU cores to use. The cross-validation loop will attempt to send different CV folds off to different cores.
- `...`: additional arguments.

**Value**

object with

- `residmat`: empirical risks in each cross-validation at boosting iterations
- `fraction`: abscissa values at which CV curve should be computed.
- `cv`: The CV curve at each value of fraction
- `cv.error`: The standard error of the CV curve
- `...`
cv.mhingebst

See Also

mbst

cv.mhingebst  Cross-Validation for Multi-class Hinge Boosting

Description

Cross-validated estimation of the empirical multi-class hinge loss for boosting parameter selection.

Usage

cv.mhingebst(x, y, balance=FALSE, K = 10, cost = NULL, family = "hinge", learner = c("tree", "ls", "sm"), ctrl = bst_control(), type = c("loss","error"), plot.it = TRUE, main = NULL, se = TRUE, n.cores=2, ...)

Arguments

x  a data frame containing the variables in the model.
y  vector of responses. y must be integers from 1 to C for C class problem.
balance  logical value. If TRUE, The K parts were roughly balanced, ensuring that the classes were distributed proportionally among each of the K parts.
K  K-fold cross-validation
cost  price to pay for false positive, 0 < cost < 1; price of false negative is 1-cost.
family  family = "hinge" for hinge loss. Implementing the negative gradient corresponding to the loss function to be minimized.
learner  a character specifying the component-wise base learner to be used: ls linear models, sm smoothing splines, tree regression trees.
ctrl  an object of class bst_control.
type  for family="hinge", type="loss" is hinge risk.
plot.it  a logical value, to plot the estimated loss or error with cross validation if TRUE.
main  title of plot
se  a logical value, to plot with standard errors.
n.cores  The number of CPU cores to use. The cross-validation loop will attempt to send different CV folds off to different cores.
...  additional arguments.

Value

object with

residmat empirical risks in each cross-validation at boosting iterations
fraction abscissa values at which CV curve should be computed.
cv  The CV curve at each value of fraction
cv.error  The standard error of the CV curve
...
See Also

mhingebst

cv.mhingeova

Cross-Validation for one-vs-all HingeBoost with multi-class problem

Description

Cross-validated estimation of the empirical misclassification error for boosting parameter selection.

Usage

`cv.mhingeova(x, y, balance=FALSE, K=10, cost = NULL, nu=0.1, learner=c("tree", "ls", "sm"), maxdepth=1, m1=200, twinboost = FALSE, m2=200, trace=FALSE, plot.it = TRUE, se = TRUE, ...)`

Arguments

- `x` a data frame containing the variables in the model.
- `y` vector of multi-class responses. `y` must be an integer vector from 1 to C for C-class problem.
- `balance` logical value. If TRUE, the K parts were roughly balanced, ensuring that the classes were distributed proportionally among each of the K parts.
- `K` K-fold cross-validation
- `cost` price to pay for false positive, 0 < cost < 1; price of false negative is 1-cost.
- `nu` a small number (between 0 and 1) defining the step size or shrinkage parameter.
- `learner` a character specifying the component-wise base learner to be used: `ls` linear models, `sm` smoothing splines, `tree` regression trees.
- `maxdepth` tree depth used in `learner=tree`
- `m1` number of boosting iteration
- `twinboost` logical: twin boosting?
- `m2` number of twin boosting iteration
- `trace` if TRUE, iteration results printed out
- `plot.it` a logical value, to plot the estimated risks if TRUE.
- `se` a logical value, to plot with standard errors.
- `...` additional arguments.
Value

object with

residmat  empirical risks in each cross-validation at boosting iterations
fraction  abscissa values at which CV curve should be computed.
cv       The CV curve at each value of fraction
cv.error  The standard error of the CV curve

Note

The functions for balanced cross validation were from R package pmar.

See Also

mhingeova

Cross-Validation for Nonconvex Loss Boosting

Description

Cross-validated estimation of the empirical risk/error, can be used for tuning parameter selection.

Usage

cv.rbst(x, y, K = 10, cost = 0.5, rfamily = c("tgaussian", "thuber", "thinge",
"tbinom", "binomd", "texpo", "tpoisson", "clossR", "closs", "gloss", "qloss"),
learner = c("ls", "sm", "tree"), ctrl = bst_control(), type = c("loss", "error"),
plot.it = TRUE, main = NULL, se = TRUE, n.cores=2,...)

Arguments

x  a data frame containing the variables in the model.
y  vector of responses. y must be in \{1,-1\} for binary classification
K  K-fold cross-validation
cost  price to pay for false positive, 0 < cost < 1; price of false negative is 1-cost.
rfamily  nonconvex loss function types.
learner  a character specifying the component-wise base learner to be used: ls linear
          models, sm smoothing splines, tree regression trees.
ctrl  an object of class bst_control.
type  cross-validation criteria. For type="loss", loss function values and type="error"
       is misclassification error.
plot.it  a logical value, to plot the estimated loss or error with cross validation if TRUE.
main  title of plot
se  a logical value, to plot with standard errors.
n.cores  The number of CPU cores to use. The cross-validation loop will attempt to send different CV folds off to different cores.
...
additional arguments.

Value

object with

residmat  empirical risks in each cross-validation at boosting iterations
mstop  boosting iteration steps at which CV curve should be computed.
cv  The CV curve at each value of mstop
cv.error  The standard error of the CV curve
rfamily  nonconvex loss function types.
...

Author(s)

Zhu Wang

See Also

rbst

Examples

## Not run:
x <- matrix(rnorm(100*5),ncol=5)
c <- 2*x[,1]
p <- exp(c)/(exp(c)+exp(-c))
y <- rbinom(100,1,p)
y[y != 1] <- -1
x <- as.data.frame(x)
cv.rbst(x, y, ctrl = bst_control(mstop=50), rfamily = "thinge", learner = "ls", type="lose")
cv.rbst(x, y, ctrl = bst_control(mstop=50), rfamily = "thinge", learner = "ls", type="error")
dat.m <- rbst(x, y, ctrl = bst_control(mstop=50), rfamily = "thinge", learner = "ls")
dat.m1 <- cv.rbst(x, y, ctrl = bst_control(twinboost=TRUE, coefir=coef(dat.m), xselect.init = dat.m$xselect, mstop=50), family = "thinge", learner="ls")

## End(Not run)
Cross-Validation for Nonconvex Multi-class Loss Boosting

Description

Cross-validated estimation of the empirical multi-class loss, can be used for tuning parameter selection.

Usage

cv.rmbst(x, y, balance=FALSE, K = 10, cost = NULL, rfamily = c("thinge", "closs"), learner = c("tree", "ls", "sm"), ctrl = bst_control(), type = c("loss","error"), plot.it = TRUE, main = NULL, se = TRUE, n.cores=2, ...)

Arguments

- **x**: a data frame containing the variables in the model.
- **y**: vector of responses. y must be integers from 1 to C for C class problem.
- **balance**: logical value. If TRUE, The K parts were roughly balanced, ensuring that the classes were distributed proportionally among each of the K parts.
- **K**: K-fold cross-validation
- **cost**: price to pay for false positive, 0 < cost < 1; price of false negative is 1-cost.
- **rfamily**: rfamily = "thinge" for truncated multi-class hinge loss. Implementing the negative gradient corresponding to the loss function to be minimized.
- **learner**: a character specifying the component-wise base learner to be used: ls linear models, sm smoothing splines, tree regression trees.
- **ctrl**: an object of class bst_control.
- **type**: loss value or misclassification error.
- **plot.it**: a logical value, to plot the estimated loss or error with cross validation if TRUE.
- **main**: title of plot
- **se**: a logical value, to plot with standard errors.
- **n.cores**: The number of CPU cores to use. The cross-validation loop will attempt to send different CV folds off to different cores.
- **...**: additional arguments.

Value

- **residmat**: empirical risks in each cross-validation at boosting iterations
- **fraction**: abscissa values at which CV curve should be computed.
- **cv**: The CV curve at each value of fraction
- **cv.error**: The standard error of the CV curve
- **...**
ex1data

Author(s)
Zhu Wang

See Also
rmbst

Description
Randomly generate data for a three-class model.

Usage
ex1data(n.data, p=50)

Arguments
n.data number of data samples.
p number of predictors.

Details
The data is generated based on Example 1 described in Wang (2012).

Value
A list with n.data by p predictor matrix x, three-class response y and conditional probabilities.

Author(s)
Zhu Wang

References

Examples
## Not run:
dat <- ex1data(100, p=5)
mhingebst(x=dat$x, y=dat$y)
## End(Not run)
**mada**  
*Multi-class AdaBoost*

**Description**

One-vs-all multi-class AdaBoost

**Usage**

```r
mada(xtr, ytr, xte=NULL, yte=NULL, mstop=50, nu=0.1, interaction.depth=1)
```

**Arguments**

- `xtr`  
  training data matrix containing the predictor variables in the model.
- `ytr`  
  training vector of responses. `ytr` must be integers from 1 to C, for C class problem.
- `xte`  
  test data matrix containing the predictor variables in the model.
- `yte`  
  test vector of responses. `yte` must be integers from 1 to C, for C class problem.
- `mstop`  
  number of boosting iteration.
- `nu`  
  a small number (between 0 and 1) defining the step size or shrinkage parameter.  
  used in gbm to specify the depth of trees.

**Details**

For a C-class problem (C > 2), each class is separately compared against all other classes with AdaBoost, and C functions are estimated to represent confidence for each class. The classification rule is to assign the class with the largest estimate.

**Value**

A list contains variable selected `xselect` and training and testing error `err.tr`, `err.te`.

**Author(s)**

Zhu Wang
See Also

`cv.mada` for cross-validated stopping iteration.

Examples

```r
data(iris)
mada(xtr=iris[,-5], ytr=iris[,5])
```

---

**mbst**

*Boosting for Multi-Classification*

Description

Gradient boosting for optimizing multi-class loss functions with componentwise linear, smoothing splines, tree models as base learners.

Usage

```r
mbst(x, y, cost = NULL, family = c("hinge", "hinge2", "thingeDC", "closs", "clossMM"),
ctrl = bst_control(), control.tree=list(fixed.depth=TRUE,
n.term.node=6, maxdepth = 1), learner = c("ls", "sm", "tree"))
```

Arguments

- `x`: a data frame containing the variables in the model.
- `y`: vector of responses. `y` must be 1, 2, ..., k for a k classification problem.
- `cost`: price to pay for false positive, 0 < cost < 1; price of false negative is 1-cost.
- `family`: family = "hinge" for hinge loss, family="hinge2" for hinge loss but the response is not recoded (see details). family="thingeDC" for DCB loss function, see `rmbst`.
- `ctrl`: an object of class `bst_control`.
- `control.tree`: control parameters of `rpart`.
- `learner`: a character specifying the component-wise base learner to be used: `ls` linear models, `sm` smoothing splines, `tree` regression trees.
- `type`: in `predict` a character indicating whether the response, all responses across the boosting iterations, classes, loss or classification errors should be predicted in case of hinge problems. in `plot`, plot of boosting iteration or $L_1$ norm.
object  class of mbst.
newdata  new data for prediction with the same number of columns as x.
newy  new response.
mstop  boosting iteration for prediction.
...  additional arguments.

Details
A linear or nonlinear classifier is fitted using a boosting algorithm for multi-class responses. This function is different from mhingebst on how to deal with zero-to-sum constraint and loss functions. If family="hinge", the loss function is the same as in mhingebst but the boosting algorithm is different. If family="hinge2", the loss function is different from family="hinge": the response is not recoded as in Wang (2012). In this case, the loss function is

$$\sum I(y_i \neq j)(f_j + 1)_+.$$  

family="thingeDC" for robust loss function used in the DCB algorithm.

Value
An object of class mbst with print, coef, plot and predict methods are available for linear models. For nonlinear models, methods print and predict are available.

x, y, cost, family, learner, control.tree, maxdepth
These are input variables and parameters
ctrl  the input ctrl with possible updated fk if family="thingeDC"
yhat  predicted function estimates
ens  a list of length mstop. Each element is a fitted model to the pseudo residuals, defined as negative gradient of loss function at the current estimated function
ml.fit  the last element of ens
ensemble  a vector of length mstop. Each element is the variable selected in each boosting step when applicable
xselect  selected variables in mstop
coef  estimated coefficients in each iteration. Used internally only

Author(s)
Zhu Wang

References
See Also
cv.mbst for cross-validated stopping iteration. Furthermore see bst_control

Examples

x <- matrix(rnorm(100*5), ncol=5)
c <- quantile(x[,1], prob=c(0.33, 0.67))
y <- rep(1, 100)
y[x[,1] > c[2]] <- 3
x <- as.data.frame(x)
dat.m <- mbst(x, y, ctrl = bst_control(mstop=50), family = "hinge", learner = "ls")
predict(dat.m)
dat.m1 <- mbst(x, y, ctrl = bst_control(twinboost=TRUE, f.init=predict(dat.m), xselect.init = dat.m$xselect, mstop=50))
dat.m2 <- rmbst(x, y, ctrl = bst_control(mstop=50, s=1, trace=TRUE), rfamily = "hinge", learner = "ls")
predict(dat.m2)

mhingebst

Boosting for Multi-class Classification

Description

Gradient boosting for optimizing multi-class hinge loss functions with componentwise linear least squares, smoothing splines and trees as base learners.

Usage

mhingebst(x, y, cost = NULL, family = c("hinge"), ctrl = bst_control(),
control.tree = list(fixed.depth=TRUE, n.term.node=6, maxdepth = 1),
learner = c("ls", "sm", "tree"))
## S3 method for class 'mhingebst'
print(x, ...)
## S3 method for class 'mhingebst'
predict(object, newdata=NULL, newy=NULL, mstop=NULL,
type=c("response", "class", "loss", "error"), ...)  
## S3 method for class 'mhingebst'
fpartial(object, mstop=NULL, newdata=NULL)

Arguments

x        a data frame containing the variables in the model.
y        vector of responses. y must be in {1,-1} for family = "hinge".
cost     equal costs for now and unequal costs will be implemented in the future.
family   family = "hinge" for multi-class hinge loss.
ctrl      an object of class bst_control.
control.tree  control parameters of rpart.
learner  a character specifying the component-wise base learner to be used: ls linear models, sm smoothing splines, tree regression trees.
type  in predict a character indicating whether the response, classes, loss or classification errors should be predicted in case of hinge
object  class of mhingebst.
newdata  new data for prediction with the same number of columns as x.
newy  new response.
mstop  boosting iteration for prediction.
...  additional arguments.

Details

A linear or nonlinear classifier is fitted using a boosting algorithm based on component-wise base learners for multi-class responses.

Value

An object of class mhingebst with print and predict methods being available for fitted models.

Author(s)

Zhu Wang

References


See Also

cv.mhingebst for cross-validated stopping iteration. Furthermore see bst_control

Examples

```r
## Not run:
dat <- ex1data(100, p=5)
res <- mhingebst(x=dat$x, y=dat$y)
```

## End(Not run)
**m hingeova**

**Multi-class HingeBoost**

**Description**

Multi-class algorithm with one-vs-all binary HingeBoost which optimizes the hinge loss functions with componentwise linear, smoothing splines, tree models as base learners.

**Usage**

```r
m hingeova(xtr, ytr, xte=NULL, yte=NULL, cost = NULL, nu=0.1, 
learner=c("tree", "ls", "sm"), maxdepth=1, m1=200, twinboost = FALSE, m2=200) 
## S3 method for class 'm hingeova'
print(x, ...)
```

**Arguments**

- `xtr`: training data containing the predictor variables.
- `ytr`: vector of training data responses. `ytr` must be in `{1,2,...,k}`.
- `xte`: test data containing the predictor variables.
- `yte`: vector of test data responses. `yte` must be in `{1,2,...,k}`.
- `cost`: default is NULL for equal cost; otherwise a numeric vector indicating price to pay for false positive, 0 < `cost` < 1; price of false negative is 1-`cost`.
- `nu`: a small number (between 0 and 1) defining the step size or shrinkage parameter.
- `learner`: a character specifying the component-wise base learner to be used: `ls` linear models, `sm` smoothing splines, `tree` regression trees.
- `maxdepth`: tree depth used in `learner=tree`
- `m1`: number of boosting iteration
- `twinboost`: logical: twin boosting?
- `m2`: number of twin boosting iteration
- `x`: class of `m hingeova`.
- `...`: additional arguments.

**Details**

For a C-class problem (C > 2), each class is separately compared against all other classes with HingeBoost, and C functions are estimated to represent confidence for each class. The classification rule is to assign the class with the largest estimate. A linear or nonlinear multi-class HingeBoost classifier is fitted using a boosting algorithm based on one-against component-wise base learners for +1/-1 responses, with possible cost-sensitive hinge loss function.

**Value**

An object of class `m hingeova` with `print` method being available.
nsel

Author(s)
Zhu Wang

References


See Also
bst for HingeBoost binary classification. Furthermore see cv.bst for stopping iteration selection by cross-validation, and bst_control for control parameters.

Examples
```r
## Not run:
dat2 <- read.table("http://archive.ics.uci.edu/ml/machine-learning-databases/thyroid-disease/ann-test.data")
res <- mhingeova(xtr=dat1[,-22], ytr=dat1[,22], xte=dat2[,-22], yte=dat2[,22],
cost=c(2/3, 0.5, 0.5), nu=0.5, learner="ls", m1=100, K=5, cv1=FALSE,
twinboost=TRUE, m2=200, cv2=FALSE)
res <- mhingeova(xtr=dat1[,-22], ytr=dat1[,22], xte=dat2[,-22], yte=dat2[,22],
cost=c(2/3, 0.5, 0.5), nu=0.5, learner="ls", m1=100, K=5, cv1=FALSE,
twinboost=TRUE, m2=200, cv2=TRUE)
## End(Not run)
```

```
nsel

Find Number of Variables In Multi-class Boosting Iterations

Description
Find Number of Variables In Multi-class Boosting Iterations

Usage
nsel(object, mstop)

Arguments

  object an object of mhingebst, mbst, or rmbst
  mstop boosting iteration number
```
Value

A vector of length mstop indicating number of variables selected in each boosting iteration.

Author(s)

Zhu Wang

Description

MM (majorization/minimization) algorithm based gradient boosting for optimizing nonconvex robust loss functions with componentwise linear, smoothing splines, tree models as base learners.

Usage

```r
rbst(x, y, cost = 0.5, rfamily = c("tgaussian", "thuber","thinge", "tbinom", "binomd", "texpo", "tpoisson", "clossR", "closs", "gloss", "qloss"), ctrl=bst_control(), control.tree=list(maxdepth = 1), learner=c("ls","sm","tree"),del=1e-10)
```

Arguments

- `x`: a data frame containing the variables in the model.
- `y`: vector of responses. `y` must be in \{1, -1\} for classification.
- `cost`: price to pay for false positive, 0 < `cost` < 1; price of false negative is 1-`cost`.
- `rfamily`: robust loss function, see details.
- `ctrl`: an object of class `bst_control`.
- `control.tree`: control parameters of rpart.
- `learner`: a character specifying the component-wise base learner to be used: `ls` linear models, `sm` smoothing splines, `tree` regression trees.
- `del`: convergency criteria

Details

An MM algorithm operates by creating a convex surrogate function that majorizes the nonconvex objective function. When the surrogate function is minimized with gradient boosting algorithm, the desired objective function is decreased. The MM algorithm contains difference of convex (DC) algorithm for \`rfamily=c("tgaussian", "thuber","thinge", "tbinom", "binomd", "texpo", "tpoisson") and quadratic majorization boosting algorithm (QMBA) for \`rfamily=c("clossR", "closs", "gloss", "qloss")

s must be a numeric value to be specified in bst.control. For rfamily="thinge", "tbinom", "texpo" s < 0. For rfamily="binomd", "tpoisson", "closs", "qloss", "clossR", s > 0 and for rfamily="gloss", s > 1. Some suggested s values: "thinge"= -1, "tbinom"= -log(3), "binomd"= log(4), "texpo"= log(0.5), "closs"=1, "gloss"=1.5, "qloss"=2, "clossR"=1.

**Value**

An object of class bst with print, coef, plot and predict methods are available for linear models. For nonlinear models, methods print and predict are available.

x, y, cost, rfamily, learner, control.tree, maxdepth

These are input variables and parameters

ctrl

the input ctrl with possible updated fk if family="tgaussian", "thingeDC", "tbinomDC", "binomdDC" or "tpoisson".

yhat

predicted function estimates

ens

a list of length mstop. Each element is a fitted model to the pseudo residuals, defined as negative gradient of loss function at the current estimated function

ml.fit

the last element of ens

ensemble

a vector of length mstop. Each element is the variable selected in each boosting step when applicable

xselect

selected variables in mstop

coeff

estimated coefficients in mstop

**Author(s)**

Zhu Wang

**References**


**See Also**

cv.rbst for cross-validated stopping iteration. Furthermore see bst_control

**Examples**

```r
x <- matrix(rnorm(100*5),ncol=5)
c <- 2*x[,1]
p <- exp(c)/(exp(c)+exp(-c))
y <- rbinom(100,1,p)
y[y != 1] <- -1
y[1:10] <- -y[1:10]
x <- as.data.frame(x)
```
dat.m <- bst(x, y, ctrl = bst_control(mstop=50), family = "hinge", learner = "ls")
predict(dat.m)
dat.m1 <- bst(x, y, ctrl = bst_control(twinboost=TRUE,
coefir=coef(dat.m), xselect.init = dat.m$xselect, mstop=50))
dat.m2 <- rbst(x, y, ctrl = bst_control(mstop=50, s=0, trace=TRUE),
rfamily = "hinge", learner = "ls")
predict(dat.m2)

---

**rbstpath**

*Robust Boosting Path for Nonconvex Loss Functions*

**Description**

Gradient boosting path for optimizing robust loss functions with componentwise linear, smoothing splines, tree models as base learners. See details below before use.

**Usage**

```r
rbstpath(x, y, rmstop=seq(40, 400, by=20), ctrl=bst_control(), del=1e-16, ...)
```

**Arguments**

- `x`: a data frame containing the variables in the model.
- `y`: vector of responses. `y` must be in `{1,-1}`.
- `rmstop`: vector of boosting iterations
- `ctrl`: an object of class `bst_control`.
- `del`: convergency criteria
- `...`: arguments passed to `rbst`

**Details**

This function invokes `rbst` with `mstop` being each element of vector `rmstop`. It can provide different paths. Thus `rmstop` serves as another hyper-parameter. However, the most important hyper-parameter is the loss truncation point or the point determines the level of nonconvexity. This is an experimental function and may not be needed in practice.

**Value**

A length `rmstop` vector of lists with each element being an object of class `rbst`.

**Author(s)**

Zhu Wang

**See Also**

`rbst`
Examples

```r
x <- matrix(rnorm(100*5), ncol=5)
c <- 2*x[,1]
p <- exp(c)/(exp(c)+exp(-c))
y <- rbinom(100, 1, p)
y[y != 1] <- -1
y[1:10] <- -y[1:10]
x <- as.data.frame(x)
dat.m <- bst(x, y, ctrl = bst_control(mstop=50), family = "hinge", learner = "ls")
predict(dat.m)
dat.m1 <- bst(x, y, ctrl = bst_control(twinboost=TRUE, coefir=coef(dat.m), xselect.init = dat.m$xselect.m, mstop=50))
dat.m2 <- rbst(x, y, ctrl = bst_control(mstop=50, s=0, trace=TRUE), rfamily = "thinge", learner = "ls")
predict(dat.m2)
rmstop <- seq(10, 40, by=10)
dat.m3 <- rbstpath(x, y, rmstop, ctrl=bst_control(s=0), rfamily = "thinge", learner = "ls")
```

---

**rmbst**

*Robust Boosting for Multi-class Robust Loss Functions*

**Description**

MM (majorization/minimization) based gradient boosting for optimizing nonconvex robust loss functions with componentwise linear, smoothing splines, tree models as base learners.

**Usage**

```r
rmbst(x, y, cost = 0.5, rfamily = c("thinge", "closs"), ctrl=bst_control(), control.tree=list(maxdepth = 1), learner=c("ls","sm","tree"), del=1e-10)
```

**Arguments**

- `x` a data frame containing the variables in the model.
- `y` vector of responses. `y` must be in \{1, 2, ..., k\}.
- `cost` price to pay for false positive, 0 < cost < 1; price of false negative is 1-cost.
- `rfamily` family = "thinge" is currently implemented.
- `ctrl` an object of class `bst_control`.
- `control.tree` control parameters of rpart.
- `learner` a character specifying the component-wise base learner to be used: ls linear models, sm smoothing splines, tree regression trees.
- `del` convergency criteria
Details
An MM algorithm operates by creating a convex surrogate function that majorizes the nonconvex objective function. When the surrogate function is minimized with gradient boosting algorithm, the desired objective function is decreased. The MM algorithm contains difference of convex (DC) for rfamily="thinge", and quadratic majorization boosting algorithm (Q MBA) for rfamily="closs".

Value
An object of class bst with print, coef, plot and predict methods are available for linear models. For nonlinear models, methods print and predict are available.

\begin{itemize}
  \item x, y, cost, rfamily, learner, control.tree, maxdepth
    These are input variables and parameters
  \item ctrl
    the input ctrl with possible updated fk if type="adaptive"
  \item yhat
    predicted function estimates
  \item ens
    a list of length mstop. Each element is a fitted model to the pseudo residuals, defined as negative gradient of loss function at the current estimated function
  \item ml.fit
    the last element of ens
  \item ensemble
    a vector of length mstop. Each element is the variable selected in each boosting step when applicable
  \item xselect
    selected variables in mstop
  \item coef
    estimated coefficients in mstop
\end{itemize}

Author(s)
Zhu Wang

References

See Also
cv.rmbst for cross-validated stopping iteration. Furthermore see bst_control

Examples
\begin{verbatim}
x <- matrix(rnorm(100*5),ncol=5)
c <- quantile(x[,1], prob=c(0.33, 0.67))
y <- rep(1, 100)
y[x[,1] > c[2]] <- 3
\end{verbatim}
x <- as.data.frame(x)
x <- as.data.frame(x)
dat.m <- mbst(x, y, ctrl = bst_control(mstop=50), family = "hinge", learner = "ls")
predict(dat.m)
dat.m1 <- mbst(x, y, ctrl = bst_control(twinboost=TRUE, f.init=predict(dat.m), xselect.init = dat.m$xselect, mstop=50))
dat.m2 <- rmbst(x, y, ctrl = bst_control(mstop=50, s=1, trace=TRUE), rfamily = "hinge", learner = "ls")
predict(dat.m2)
## Index

### classification
- bst, 2
- ex1data, 16
- mada, 17
- mbst, 18
- mhingebst, 20
- mhingeova, 22
- rbst, 24
- rbstpath, 26
- rmbst, 27

### models
- bst.sel, 4

### regression
- bst.sel, 4

<table>
<thead>
<tr>
<th>balanced.folds (loss)</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>bst, 2, 3, 5, 7, 8, 23</td>
<td></td>
</tr>
<tr>
<td>bst.sel, 4</td>
<td></td>
</tr>
<tr>
<td>bst_control, 3, 4, 5, 8, 10, 11, 13, 15, 18, 20, 21, 23–28</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>coef, 3, 19, 25, 28</th>
</tr>
</thead>
<tbody>
<tr>
<td>coef.bst (bst), 2</td>
</tr>
<tr>
<td>cv.bst, 4, 5, 7, 23</td>
</tr>
<tr>
<td>cv.mada, 9, 18</td>
</tr>
<tr>
<td>cv.mbst, 10, 20</td>
</tr>
<tr>
<td>cv.mhingebst, 11, 21</td>
</tr>
<tr>
<td>cv.mhingeova, 12</td>
</tr>
<tr>
<td>cv.rbst, 13, 25</td>
</tr>
<tr>
<td>cv.rmbst, 15, 28</td>
</tr>
<tr>
<td>cvfolds (loss), 17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>error.bars (loss), 17</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex1data, 16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>fpartial.bst (bst), 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>fpartial.mbst (mbst), 18</td>
</tr>
<tr>
<td>fpartial.mhingebst (mhingebst), 20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>gradient (loss), 17</th>
</tr>
</thead>
<tbody>
<tr>
<td>hingeloss (loss), 17</td>
</tr>
<tr>
<td>hingengra (loss), 17</td>
</tr>
<tr>
<td>loss, 17</td>
</tr>
<tr>
<td>mada, 9, 17</td>
</tr>
<tr>
<td>mbst, 11, 18, 19, 23</td>
</tr>
<tr>
<td>mbst_fit (loss), 17</td>
</tr>
<tr>
<td>mhingebst, 12, 20, 21, 23</td>
</tr>
<tr>
<td>mhingebst_fit (loss), 17</td>
</tr>
<tr>
<td>mhingeova, 13, 22, 22</td>
</tr>
<tr>
<td>ngradient (loss), 17</td>
</tr>
<tr>
<td>nsel, 23</td>
</tr>
<tr>
<td>permute.rows (loss), 17</td>
</tr>
<tr>
<td>plot, 3, 19, 25, 28</td>
</tr>
<tr>
<td>plot.bst (bst), 2</td>
</tr>
<tr>
<td>plotCVbst (loss), 17</td>
</tr>
<tr>
<td>predict, 3, 19, 21, 25, 28</td>
</tr>
<tr>
<td>predict.bst (bst), 2</td>
</tr>
<tr>
<td>predict.mbst (mbst), 18</td>
</tr>
<tr>
<td>predict.mhingebst (mhingebst), 20</td>
</tr>
<tr>
<td>print, 3, 19, 21, 22, 25, 28</td>
</tr>
<tr>
<td>print.bst (bst), 2</td>
</tr>
<tr>
<td>print.mbst (mbst), 18</td>
</tr>
<tr>
<td>print.mhingebst (mhingebst), 20</td>
</tr>
<tr>
<td>print.mhingeova (mhingeova), 22</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rbst, 14, 24, 26</th>
</tr>
</thead>
<tbody>
<tr>
<td>rbstpath, 26</td>
</tr>
<tr>
<td>rmbst, 16, 23, 27</td>
</tr>
</tbody>
</table>