

Package ‘mboost’

November 18, 2009

Title Model-Based Boosting

Date 2009-11-18

Version 1.1-4

Author Torsten Hothorn, Peter Buhlmann, Thomas Kneib, Matthias Schmid and Benjamin Hofner

Maintainer Torsten Hothorn <Torsten.Hothorn@R-project.org>

Description Functional gradient descent algorithms (boosting) for optimizing general loss functions utilizing componentwise least squares, either of parametric linear form or smoothing splines, or regression trees as base learners for fitting generalized linear, additive and interaction models to potentially high-dimensional data.

Depends R (>= 2.4.0), methods, modeltools (>= 0.2.10), party (>= 0.9-993), splines

Suggests mlbench, ipred, multicore

LazyLoad yes

LazyData yes

License GPL-2

Repository CRAN

Date/Publication 2009-11-18 11:50:51

R topics documented:

baselearners	2
birds	6
blackboost	7
bodyfat	9
boost_control	10
boost_dpp	12
boost_family-class	12
cvrisk	13

Family	15
FP	17
gamboost	18
glmboost	21
IPCweights	23
methods	24
survFit	26
Westbc	28
wpbc	29

Index	31
--------------	-----------

baselearners	<i>Base learners for Gradient Boosting with Smooth Components</i>
--------------	---

Description

Base learners to be utilized in the formula specification of `gamboost()`.

Usage

```

bols(x, z = NULL, xname = NULL, zname = NULL, center = FALSE,
      df = NULL, contrasts.arg = "contr.treatment")
bbs(x, z = NULL, df = 4, knots = 20, degree = 3, differences = 2,
     center = FALSE, xname = NULL, zname = NULL)
bns(x, z = NULL, df = 4, knots = 20, differences = 2,
     xname = NULL, zname = NULL)
bss(x, df = 4, xname = NULL)
bspacial(x, y, z = NULL, df = 5, xknots = 20, yknots = 20,
          degree = 3, differences = 2, center = FALSE, xname = NULL,
          yname = NULL, zname = NULL)
brandom(x, z = NULL, df = 4, xname = NULL, zname = NULL)
btree(..., tree_controls = ctree_control(stump = TRUE,
                                           mincriterion = 0), xname = NULL)

```

Arguments

<code>x</code>	a vector containing data, either numeric, a factor or an ordered factor.
<code>y</code>	a vector containing numeric data.
<code>z</code>	an optional vector containing data. <code>z</code> can be numeric or a binary (factor).
<code>xname</code>	an optional string indicating the name of the variable whose data values are given by the vector <code>x</code> .
<code>yname</code>	an optional string indicating the name of the variable whose data values are given by the vector <code>y</code> .
<code>zname</code>	an optional string indicating the name of the variable whose data values are given by the vector <code>z</code> .

<code>df</code>	trace of the hat matrix for the base learner defining the base learner complexity. Low values of <code>df</code> correspond to a large amount of smoothing and thus to "weaker" base learners. Certain restrictions have to be kept for the specification of <code>df</code> since most of the base learners rely on penalisation approaches with a non-trivial null space. For example, for p-splines fitted with <code>bbs</code> , <code>df</code> has to be larger than the order of differences employed in the construction of the penalty term. However, when option <code>center=TRUE</code> , the effect is centered around its unpenalized part and therefore any positive number is admissible for <code>df</code> .
<code>knots</code>	either the number of (equidistant) interior knots to be used for the regression spline fit or a vector including the positions of the interior knots.
<code>xknots</code>	knots in <i>x</i> -direction when fitting a bivariate surface with <code>bspatial</code> . See <code>knots</code> for details.
<code>yknots</code>	knots in <i>y</i> -direction when fitting a bivariate surface with <code>bspatial</code> . See <code>knots</code> for details.
<code>degree</code>	degree of the regression spline.
<code>differences</code>	natural number between 1 and 3. If <code>differences = k</code> , <i>k</i> -th-order differences are used as a penalty.
<code>center</code>	If <code>center=TRUE</code> in <code>bols</code> , the intercept in the linear model is omitted. If <code>center=TRUE</code> in <code>bbs</code> and <code>bspatial</code> , the corresponding effect is re-parameterized such that the unpenalized part of the fit is subtracted and only the deviation effect is fitted. The unpenalized, parametric part has then to be included in separate base learners using <code>bols</code> (see the examples below).
<code>contrasts.arg</code>	a character suitable for input to the <code>contrasts</code> replacement function.
<code>tree_controls</code>	an object of class " <code>TreeControl</code> ", which can be obtained using <code>ctree_control</code> . Defines hyper-parameters for the trees which are used as base learners, stumps are fitted by default.
<code>...</code>	a number of variables to fit a tree to.

Details

`bols` refers to linear base learners (ordinary least squares fit), while `bbs`, `bns`, and `bss` refer to penalized regression splines, penalized natural splines, and smoothing splines, respectively. `bspatial` fits bivariate surfaces and `brandom` defines random effects base learners. In combination with option `z`, all base learners can be turned into varying coefficient terms.

Linear base learners can be set up using `bols`. The function can deal with both numeric and factor variables `x`. By default, an intercept term is added to the corresponding design matrix (which can be omitted using `center = TRUE`). When `df` is given, and `x` is a factor a ridge estimator with `df` degrees of freedom is used as base learner. When `x` is an ordered factor (`ordered()`) and `df` is given the differences of adjacent parameters are ridge penalized.

With `bbs`, the P-spline approach of Eilers and Marx (1996) is used. `bns` uses the same penalty and interior knots as `bbs`, but operates with a constrained natural spline basis instead of an unconstrained B-spline basis. P-splines use a squared *k*-th-order difference penalty which can be

interpreted as an approximation of the integrated squared k -th derivative of the spline. This approximation is only valid if the knots are equidistant, so it is not recommended to use non-equidistant knots for `bbs` and `bns`. `bss` refers to a smoothing spline based on the `smooth.spline` function.

`bspatial` implements bivariate tensor product P-splines for the estimation of either spatial effects (if `x` and `y` correspond to coordinates) or interaction surfaces. The penalty term is constructed based on bivariate extensions of the univariate penalties in `x` and `y` directions, see Kneib, Hothorn and Tutz (2007) for details. Note that the dimensions of the penalty matrix increase (quickly) with the number of `xknots` and `yknots` with strong impact on computational time. Thus, both should not be chosen to large.

`brandom` specifies a random effects base learner based on a factor variable `x` that defines the grouping structure of the data set. For each level of `x`, a separate random intercept is fitted, where the random effects variance is governed by the specification of the degrees of freedom `df`.

For all base learners the amount of smoothing is determined by the trace of the hat matrix, as indicated by `df`. If `df` is specified in `bols` a ridge penalty or a ridge penalty of the differences of adjacent parameters with the according degrees of freedom is used.

If `z` is specified as an additional argument, a varying coefficients term is estimated, where `z` is the interaction variable and the effect modifier is given by either `x` or `x` and `y`. If one of the non-parametric base learners `bbs` or `bns` is used, this corresponds to the classical situation of varying coefficients, where the effect of `z` varies over the co-domain of `x`. In case of `bspatial` as base learner, the effect of `z` varies with respect to both `x` and `y`, i.e. an interaction surface between `x` and `y` is specified as effect modifier. For `brandom` specification of `z` leads to the estimation of random slopes for covariate `z` with grouping structure defined by factor `x` instead of a simple random intercept.

For `bbs` and `bspatial`, option `center` requests that the fitted effect is centered around its parametric, unpenalized part. For example, with second order difference penalty, a linear effect of `x` remains unpenalized by `bbs` and therefore the degrees of freedom for the base learner have to be larger than 2. To avoid this restriction, option `center=TRUE` subtracts the unpenalized linear effect from the fit, allowing to specify any positive number as `df`. Note that in this case the linear effect `x` should generally be specified as an additional base learner `bols(x)`. For `bspatial` and, for example, second order differences, a linear effect of `x` (`bols(x)`), a linear effect of `y` (`bols(y)`), and their interaction (`bols(x*y)`) are subtracted from the effect and have to be added separately to the model equation. More details on centering can be found in Kneib, Hothorn and Tutz (2007) and Fahrmeir, Kneib and Lang (2004).

By default, all base learners include an intercept term (which can only be removed using `center = TRUE` for `bols`, `bbs` and `bspatial`). In this case, an explicit global intercept term should be added to `gamboost` via `bols` (see example below).

`btree` fits a stump to one or two variables. Note that `blackboost` is more efficient for boosting stumps.

Value

Either a matrix (in case of an ordinary least squares fit) or an object of class `basis` (in case of a regression or smoothing spline fit) with a `dpp` function as an additional attribute. The call of `dpp` returns an object of class `basisdpp`.

References

Paul H. C. Eilers and Brian D. Marx (1996), Flexible smoothing with B-splines and penalties. *Statistical Science*, **11**(2), 89-121.

Ludwig Fahrmeir, Thomas Kneib and Stefan Lang (2004), Penalized structured additive regression for space-time data: a Bayesian perspective. *Statistica Sinica*, **14**, 731-761.

Thomas Kneib, Torsten Hothorn and Gerhard Tutz (2009), Variable selection and model choice in geoaddditive regression models, *Biometrics*, accepted. <http://epub.ub.uni-muenchen.de/2063/>

See Also

[gamboost](#)

Examples

```
x1 <- rnorm(100)
x2 <- rnorm(100) + 0.25*x1
x3 <- as.factor(sample(0:1, 100, replace = TRUE))
x4 <- gl(4, 25)
y <- 3*sin(x1) + x2^2 + rnorm(100)

knots.x2 <- quantile(x2, c(0.25,0.5,0.75))

spline1 <- bbs(x1,knots=20,df=4)
attributes(spline1)
spline2 <- bns(x2,knots=knots.x2,df=5)
attributes(spline2)
olsfit <- bols(x3)
attributes(olsfit)

form1 <- y ~ bbs(x1,knots=20,df=4) + bns(x2,knots=knots.x2,df=5)

# example for factors
attributes(bols(x4))

# example for bspatial

x1 <- runif(250,-pi,pi)
x2 <- runif(250,-pi,pi)

y <- sin(x1)*sin(x2) + rnorm(250, sd = 0.4)

spline3 <- bspatial(x1, x2, xknots=12, yknots=12)
attributes(spline3)

form2 <- y ~ bspatial(x1, x2, xknots=12, yknots=12)

# decompose spatial effect into parametric part and deviation with 1 df
form2 <- y ~ bols(x1) + bols(x2) + bols(x1*x2) +
  bspatial(x1, x2, xknots=12, yknots=12, center = TRUE, df=1)
```

```

# random intercept

id <- factor(rep(1:10, each=5))
raneff <- brandom(id)
attributes(raneff)

# random slope

z <- runif(50)
raneff <- brandom(id, z=z)
attributes(raneff)

# remove intercept from base learner
# and add explicit intercept to the model

tmpdata <- data.frame(x = 1:100, y = rnorm(1:100), int = rep(1, 100))
mod <- gamboost(y ~ bols(int, center = TRUE) + bols(x, center = TRUE),
               data = tmpdata, control = boost_control(mstop = 2500))
cf <- unlist(coef(mod))
cf[1] <- cf[1] + mod$offset
cf
coef(lm(y ~ x, data = tmpdata))

```

birds

Habitat Suitability for Breeding Bird Communities

Description

Environmental variables and bird counts for identifying suitable bird habitats

Usage

```
data(birds)
```

Format

A data frame with 258 observations on the following 10 variables.

GST Growing stock per grid
 DBH Mean diameter of the largest three trees
 AOT Age of oldest tree
 AFS Age of forest stand
 DWC Amount of dead wood of conifers
 LOG Amount of logs per grid
 x_gk grid location, x coordinate

y_gk grid location, y coordinate

SG4 observed number of birds from structural gild 4: Requirement of regeneration (Phylloscopus trochilus, Aegithalos caudatus)

SG5 observed number of birds from structural gild 5: Requirement of regeneration combined with planted conifers (Phylloscopus collybita, Turdus merula, Sylvia atricapilla).

Details

Counts of breeding bird communities collected at 258 observation plots in a northern Bavarian forest district are the response variable of interest. Along with the number of birds in two structural gilds, 6 covariates are given here and one is interested in quantifying their impact on habitat suitability.

Source

Joerg Mueller (2005). Forest structures as key factor for beetle and bird communities in beech forests. PhD thesis, Munich University of Technology. <http://mediatum.ub.tum.de->

References

Thomas Kneib and Joerg Mueller and Torsten Hothorn (2008), Spatial smoothing techniques for the assessment of habitat suitability, *Environmental and Ecological Statistics*, **15**(3), 343–364.

blackboost

Gradient Boosting with Regression Trees

Description

Gradient boosting for optimizing arbitrary loss functions where regression trees are utilized as base learners.

Usage

```
## S3 method for class 'formula':
blackboost(formula, data = list(), weights = NULL, ...)
## S3 method for class 'matrix':
blackboost(x, y, weights = NULL, ...)
blackboost_fit(object, tree_controls =
  ctree_control(teststat = "max",
                testtype = "Teststatistic",
                mincriterion = 0,
                maxdepth = 2),
  fitmem = ctree_memory(object, TRUE), family = GaussReg(),
  control = boost_control(), weights = NULL)
```

Arguments

<code>formula</code>	a symbolic description of the model to be fit.
<code>data</code>	a data frame containing the variables in the model.
<code>weights</code>	an optional vector of weights to be used in the fitting process.
<code>x</code>	design matrix.
<code>y</code>	vector of responses.
<code>object</code>	an object of class <code>boost_data</code> , see <code>boost_dpp</code> .
<code>tree_controls</code>	an object of class <code>"TreeControl"</code> , which can be obtained using <code>ctree_control</code> . Defines hyper-parameters for the trees which are used as base learners. It is wise to make sure to understand the consequences of altering any of its arguments.
<code>fitmem</code>	an object of class <code>TreeFitMemory</code> .
<code>family</code>	an object of class <code>"boost_family"</code> , implementing the negative gradient corresponding to the loss function to be optimized. By default, squared error loss for continuous responses is used.
<code>control</code>	an object of class <code>boost_control</code> which defines the hyper-parameters of the boosting algorithm.
<code>...</code>	additional arguments passed to callies.

Details

This function implements the ‘classical’ gradient boosting utilizing regression trees as base learners. Essentially, the same algorithm is implemented in package `gbm`. The main difference is that arbitrary loss functions to be optimized can be specified via the `family` argument to `blackboost` whereas `gbm` uses hard-coded loss functions. Moreover, the base learners (conditional inference trees, see `ctree`) are a little bit more flexible.

The regression fit is a black box prediction machine and thus hardly interpretable.

Usually, the formula based interface `blackboost` should be used. When necessary (for example for cross-validation), function `blackboost_fit` operating on objects of class `boost_data` is faster alternative.

Value

An object of class `blackboost` with `print` and `predict` methods being available.

References

Jerome H. Friedman (2001), Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, **29**, 1189–1232.

Greg Ridgeway (1999), The state of boosting. *Computing Science and Statistics*, **31**, 172–181.

Peter Buhlmann and Torsten Hothorn (2007), Boosting algorithms: regularization, prediction and model fitting. *Statistical Science*, **22**(4), 477–505.

Torsten Hothorn, Kurt Hornik and Achim Zeileis (2006). Unbiased Recursive Partitioning: A Conditional Inference Framework. *Journal of Computational and Graphical Statistics*, **15**(3), 651–674.

See Also

[glmboost](#) for boosted linear models and [gamboost](#) for boosted smooth models. See [cvrisk](#) for cross-validated stopping iteration. Furthermore see [boost_control](#), [Family](#) and [methods](#)

Examples

```
### a simple two-dimensional example: cars data
cars.gb <- blackboost(dist ~ speed, data = cars,
                      control = boost_control(mstop = 50))
cars.gb

### plot fit
plot(dist ~ speed, data = cars)
lines(cars$speed, predict(cars.gb), col = "red")
```

bodyfat	<i>Prediction of Body Fat by Skinfold Thickness, Circumferences, and Bone Breadths</i>
---------	--

Description

For 71 healthy female subjects, body fat measurements and several anthropometric measurements are available for predictive modelling of body fat.

Usage

```
data("bodyfat")
```

Format

A data frame with 71 observations on the following 10 variables.

age age in years.

DEXfat body fat measured by DXA, response variable.

waistcirc waist circumference.

hipcirc hip circumference.

elbowbreadth breadth of the elbow.

kneebreadth breadth of the knee.

anthro3a sum of logarithm of three anthropometric measurements.

anthro3b sum of logarithm of three anthropometric measurements.

anthro3c sum of logarithm of three anthropometric measurements.

anthro4 sum of logarithm of three anthropometric measurements.

Details

Garcia et al. (2005) report on the development of predictive regression equations for body fat content by means of common anthropometric measurements which were obtained for 71 healthy German women. In addition, the women's body composition was measured by Dual Energy X-Ray Absorptiometry (DXA). This reference method is very accurate in measuring body fat but finds little applicability in practical environments, mainly because of high costs and the methodological efforts needed. Therefore, a simple regression equation for predicting DXA measurements of body fat is of special interest for the practitioner. Backward-elimination was applied to select important variables from the available anthropometrical measurements, and Garcia (2005) report a final linear model utilizing hip circumference, knee breadth and a compound covariate which is defined as the sum of log chin skinfold, log triceps skinfold and log subscapular skinfold.

Source

Ada L. Garcia, Karen Wagner, Torsten Hothorn, Corinna Koebnick, Hans-Joachim F. Zunft and Ulrike Trippo (2005), Improved prediction of body fat by measuring skinfold thickness, circumferences, and bone breadths. *Obesity Research*, **13**(3), 626–634.

Peter Buhlmann and Torsten Hothorn (2007), Boosting algorithms: regularization, prediction and model fitting. *Statistical Science*, **22**(4), 477–505.

Examples

```
data("bodyfat", package = "mboost")

### final model proposed by Garcia et al. (2005)
fmod <- lm(DEXfat ~ hipcirc + anthro3a + kneebreadth, data = bodyfat)
coef(fmod)

### plot additive model for same variables
amod <- gamboost(DEXfat ~ hipcirc + anthro3a + kneebreadth,
                 data = bodyfat, baselearner = "bbs")
layout(matrix(1:3, ncol = 3))
plot(amod[mstop(AIC(amod, "corrected"))], ask = FALSE)
```

Description

Definition of the initial number of boosting iterations, step size and other hyper-parameters for boosting algorithms.

Usage

```
boost_control(mstop = 100, nu = 0.1, constraint = FALSE,
             risk = c("inbag", "oobag", "none"),
             savedata = TRUE, center = FALSE, trace = FALSE,
             save_ensemless=TRUE)
```

Arguments

mstop	a integer giving the number of initial boosting iterations.
nu	a double (between 0 and 1) defining the step size or shrinkage parameter.
constraint	a logical indicating whether the working responses should be restricted to $(-1, +1)$.
risk	a character indicating how the empirical risk should be computed for each boosting iteration. <code>inbag</code> leads to risks computed for the learning sample (i.e., all non-zero weights), <code>oobag</code> to risks based on the out-of-bag (all observations with zero weights) and <code>none</code> to no risk computations at all.
savedata	a logical, should the data be saved in the returned object?
center	a logical indicating if the numerical covariates should be mean centered before fitting. Only implemented for <code>glmboost</code> . In <code>blackboost</code> centering is not needed. In <code>gamboost</code> centering is only needed if <code>bols</code> base-learners are specified without intercept. In this case centering of the covariates is essential and should be done manually (at the moment).
trace	a logical triggering printout of status information during the fitting process.
save_ensemless	a logical indicating if the list of baselearners should be saved and returned. This list is generally needed but can be suppressed to reduce memory usage (not recommended).

Details

Objects returned by this function specify hyper-parameters of the boosting algorithms implemented in `glmboost`, `gamboost` and `blackboost` (via the `control` argument).

Value

An object of class `boost_control`, a list.

See Also

`gamboost`, `glmboost` and `blackboost` for the usage

`boost_dpp`*Data Preprocessing for Gradient Boosting*

Description

Formula parsing and model matrix computations for gradient boosting functions.

Usage

```
boost_dpp(formula, data, weights = NULL, na.action = na.omit, ...)
```

Arguments

<code>formula</code>	a symbolic description of the model to be fit.
<code>data</code>	a data frame containing the variables in the model.
<code>weights</code>	an optional vector of weights to be used in the fitting process.
<code>na.action</code>	a function which indicates what should happen when the data contain NAs.
<code>...</code>	additional arguments passed to callies.

Details

Some very basic formula parsing and model matrix computations.

Value

An object of class `boost_data`.

`boost_family-class` *Class "boost_family": Gradient Boosting Family*

Description

Objects of class "boost_family" define negative gradients of loss functions to be optimized.

Objects from the Class

Objects can be created by calls of the form `Family(...)`

Slots

`ngradient`: a function with arguments `y` and `f` implementing the *negative* gradient of the `loss` function.

`loss`: a loss function with arguments `y` and `f` (to be minimized).

`risk`: a risk function with arguments `y`, `f` and `w`, the weighted mean of the loss function by default.

`offset`: a function with argument `y` and `w` (weights) for computing a *scalar* offset.

`weights`: a logical indicating if weights are allowed.

`fW`: transformation of the fit for the diagonal weights matrix for an approximation of the boosting hat matrix for loss functions other than squared error.

`check_y`: a function for checking the class / mode of a response variable.

`name`: a character giving the name of the loss function for pretty printing.

`charloss`: a character, the deparsed loss function.

See Also

[Family](#)

Examples

```
Laplace()
```

cvrisk

Cross-Validation

Description

Cross-validated estimation of the empirical risk for hyper-parameter selection.

Usage

```
cvrisk(object, folds, grid = c(1:mstop(object)), ...)
```

Arguments

<code>object</code>	an object of class <code>gb</code> .
<code>folds</code>	a weight matrix with number of rows equal to the number of observations. The number of columns corresponds to the number of cross-validation runs.
<code>grid</code>	a vector of stopping parameters the empirical risk is to be evaluated for.
<code>...</code>	additional arguments passed to <code>mclapply</code> eventually.

Details

The number of boosting iterations is a hyper-parameter of the boosting algorithms implemented in this package. Honest, i.e., cross-validated, estimates of the empirical risk for different stopping parameters `mstop` are computed by this function which can be utilized to choose an appropriate number of boosting iterations to be applied.

Different forms of cross-validation can be applied, for example 10-fold cross-validation or bootstrapping. The weights (zero weights correspond to test cases) are defined via the `folds` matrix.

If package `multicore` is available, `cvrisk` runs in parallel on cores/processors available. The scheduling can be changed by the corresponding arguments of `mclapply` (via the dot arguments). No trace output is given when running in parallel.

Value

An object of class `cvrisk`, basically a matrix containing estimates of the empirical risk for a varying number of bootstrap iterations. `plot` and `print` methods are available as well as a `mstop` method.

Note

The model object needs to be fitted with option `savedata = TRUE` in `boost_control`.

References

Torsten Hothorn, Friedrich Leisch, Achim Zeileis and Kurt Hornik (2006), The design and analysis of benchmark experiments. *Journal of Computational and Graphical Statistics*, **14**(3), 675–699.

See Also

`AIC.gamboost` or `AIC.glmboost` for AIC based selection of the stopping iteration. Use `mstop` to extract the optimal stopping iteration from `cvrisk` object.

Examples

```
data("bodyfat", package = "mboost")

### fit linear model to data
model <- glmboost(DEXfat ~ ., data = bodyfat,
                  control = boost_control(center = TRUE))

### AIC-based selection of number of boosting iterations
maic <- AIC(model)
maic

### inspect coefficient path and AIC-based stopping criterion
par(mai = par("mai") * c(1, 1, 1, 1.8))
plot(model)
abline(v = mstop(maic), col = "lightgray")

### 10-fold cross-validation
```

```

n <- nrow(bodyfat)
k <- 10
ntest <- floor(n / k)
cv10f <- matrix(c(rep(c(rep(0, ntest), rep(1, n)), k - 1),
                  rep(0, n * k - (k - 1) * (n + ntest))), nrow = n)
cvm <- cvrisk(model, folds = cv10f)
print(cvm)
mstop(cvm)
plot(cvm)

### 25 bootstrap iterations
set.seed(290875)
bs25 <- rmultinom(25, n, rep(1, n)/n)
cvm <- cvrisk(model, folds = bs25)
print(cvm)
mstop(cvm)

layout(matrix(1:2, ncol = 2))
plot(cvm)

### trees
blackbox <- blackboost(DEXfat ~ ., data = bodyfat)
cvtree <- cvrisk(blackbox, folds = bs25)
plot(cvtree)

```

Family

Gradient Boosting Families

Description

`boost_family` objects provide a convenient way to specify loss functions and corresponding risk functions to be optimized by one of the boosting algorithms implemented in this package.

Usage

```

Family(ngradient, loss = NULL, risk = NULL,
       offset = function(y, w) 0,
       fW = function(f) rep(1, length(f)),
       check_y = function(y) TRUE,
       weights = TRUE, name = "user-specified")
AdaExp()
Binomial()
GaussClass()
GaussReg()
Huber(d = NULL)
Laplace()
Poisson()
CoxPH()
QuantReg(tau = 0.5, qoffset = 0.5)

```

Arguments

<code>ngradient</code>	a function with arguments y , f and w implementing the <i>negative</i> gradient of the loss function (which is to be minimized).
<code>loss</code>	an optional loss function with arguments y and f to be minimized (!).
<code>risk</code>	an optional risk function with arguments y , f and w , the weighted mean of the loss function by default.
<code>offset</code>	a function with argument y and w (weights) for computing a <i>scalar</i> offset.
<code>fW</code>	transformation of the fit for the diagonal weights matrix for an approximation of the boosting hat matrix for loss functions other than squared error.
<code>check_y</code>	a function for checking the class / mode of a response variable.
<code>weights</code>	a logical indicating if weights are allowed.
<code>name</code>	a character giving the name of the loss function for pretty printing.
<code>d</code>	delta parameter for Huber loss function. If omitted, it is chosen adaptively.
<code>tau</code>	the quantile to be estimated, a number strictly between 0 and 1.
<code>qoffset</code>	quantile of response distribution to be used as offset.

Details

The boosting algorithms implemented in `glmboost`, `gamboost` or `blackboost` aim at minimizing the (weighted) empirical risk function $\text{risk}(y, f, w)$ with respect to f . By default, the risk function is the weighted sum of the loss function $\text{loss}(y, f)$ but can be chosen arbitrarily. The `ngradient`(y, f) function is the negative gradient of $\text{loss}(y, f)$ with respect to f . For binary classification problems we assume that the response y is coded by -1 and $+1$.

Pre-fabricated functions for the most commonly used loss functions are available as well.

The `offset` function returns the population minimizers evaluated at the response, i.e., $1/2 \log(p/(1-p))$ for `Binomial()` or `AdaExp()` and $(\sum w_i)^{-1} \sum w_i y_i$ for `GaussReg` and the median for Huber and Laplace.

Boosting for quantile regression with the `QuantReg` family is introduced in Fenske et al. (2009).

Value

An object of class `boost_family`.

References

Nora Fenke, Thomas Kneib, and Torsten Hothorn (2009). Identifying risk factors for severe childhood malnutrition by boosting additive quantile regression. Technical Report Nr. 52, Institut fuer Statistik, LMU Muenchen. <http://epub.ub.uni-muenchen.de/>

See Also

`gamboost`, `glmboost` and `blackboost` for the usage of `Family`s. See `boost_family-class` for objects resulting from a call to `Family`.

Examples

```
Laplace()

Family(ngradient = function(y, f) y - f,
       loss = function(y, f) (y - f)^2,
       name = "My Gauss Variant")
```

 FP

Fractional Polynomials

Description

Fractional polynomials transformation for continuous covariates.

Usage

```
FP(x, p = c(-2, -1, -0.5, 0.5, 1, 2, 3))
```

Arguments

`x` a numeric vector.
`p` all powers of `x` to be included.

Details

A fractional polynomial refers to a model $\sum_{j=1}^k (\beta_j x^{p_j} + \gamma_j x^{p_j} \log(x)) + \beta_{k+1} \log(x) + \gamma_{k+1} \log(x)^2$, where the degree of the fractional polynomial is the number of non-zero regression coefficients β and γ . See [mfp](#) for the reference implementation.

Currently, no scaling of `x` is implemented. However, one may wish to standardize the inputs prior to fitting the model.

Value

A matrix including all powers `p` of `x`, all powers `p` of $\log(x)$, and $\log(x)$.

References

Willi Sauerbrei and Patrick Royston (1999), Building multivariable prognostic and diagnostic models: transformation of the predictors by using fractional polynomials. *Journal of the Royal Statistical Society A*, **162**, 71–94.

See Also

[gamboost](#) to fit smooth models, [bbs](#) for P-spline base-learners

Examples

```

data("bodyfat", package = "mboost")
tbodyfat <- bodyfat

### map covariates into [1, 2]
indep <- names(tbodyfat)[-2]
tbodyfat[indep] <- lapply(bodyfat[indep], function(x) {
  x <- x - min(x)
  x / max(x) + 1
})

### generate formula
fpfm <- as.formula(paste("DEXfat ~ ", paste("FP(", indep, ")",
  collapse = "+")))
fpfm

### fit linear model
bf_fp <- glmboost(fpfm, data = tbodyfat,
  control = boost_control(mstop = 3000))

### when to stop
mstop(aic <- AIC(bf_fp))
plot(aic)

### coefficients
cf <- coef(bf_fp[mstop(aic)])
length(cf)
cf[abs(cf) > 0]

```

gamboost

Gradient Boosting with Smooth Components

Description

Gradient boosting for optimizing arbitrary loss functions, where component-wise smoothing procedures are utilized as base learners.

Usage

```

## S3 method for class 'formula':
gamboost(formula, data = list(), weights = NULL,
  na.action = na.omit, ...)
## S3 method for class 'matrix':
gamboost(x, y, weights = NULL, ...)
gamboost_fit(object, baselearner = c("bbs", "bss", "bols",
  "bns", "btree"), dfbase = 4, family = GaussReg(),
  control = boost_control(), weights = NULL)

```

```
## S3 method for class 'gamboost':
plot(x, which = NULL, ask = TRUE && dev.interactive(),
     type = "b", ylab = expression(f[partial]), add_rug = TRUE, ...)
```

Arguments

<code>formula</code>	a symbolic description of the model to be fit.
<code>data</code>	a data frame containing the variables in the model.
<code>weights</code>	an optional vector of weights to be used in the fitting process.
<code>na.action</code>	a function which indicates what should happen when the data contain NAs.
<code>x</code>	design matrix (for <code>gamboost.matrix</code>) or an object returned by <code>gamboost</code> to be plotted via <code>plot</code> .
<code>y</code>	vector of responses.
<code>object</code>	an object of class <code>boost_data</code> , see <code>boost_dpp</code> .
<code>baselearner</code>	a character specifying the component-wise base learner to be used: <code>bss</code> means smoothing splines (see Buhlmann and Yu 2003), <code>bbs</code> P-splines with a B-spline basis (see Schmid and Hothorn 2007), <code>bns</code> P-splines with a natural spline basis, <code>bols</code> linear models, <code>bspatial</code> bivariate tensor product penalized splines, and <code>brandom</code> random effects. In addition, <code>btree</code> boosts stumps. Component-wise smoothing splines have been considered in Buhlmann and Yu (2003) and Schmid and Hothorn (2007) investigate P-splines with a B-spline basis. Kneib, Hothorn and Tutz (2007) also utilise P-splines with a B-spline basis, supplement them with their bivariate tensor product version to estimate interaction surfaces and spatial effects and also consider random effects base learners.
<code>dfbase</code>	an integer vector giving the degrees of freedom for the smoothing spline, either globally for all variables (when its length is one) or separately for each single covariate.
<code>family</code>	an object of class <code>boost_family-class</code> , implementing the negative gradient corresponding to the loss function to be optimized, by default, squared error loss for continuous responses is used.
<code>control</code>	an object of class <code>boost_control</code> .
<code>which</code>	if a subset of the plots is required, specify a subset of the variables. Only selected variables are plotted by default.
<code>ask</code>	logical; if <code>TRUE</code> , the user is <i>asked</i> before each plot, see <code>par(ask=.)</code> .
<code>type</code>	what type of plot should be drawn: see <code>plot</code> .
<code>ylab</code>	a title for the y axis: see <code>title</code> .
<code>add_rug</code>	logical; if <code>TRUE</code> , <code>rugs</code> are added.
<code>...</code>	additional arguments passed to callies.

Details

A (generalized) additive model is fitted using a boosting algorithm based on component-wise univariate base learners. The base learner can either be specified via the `formula` object or via the

`baselearner` argument (see `bbs` for an example). If the base learners specified in `formula` differ from `baselearner`, the latter argument will be ignored.

The function `gamboost_fit` provides access to the fitting procedure without data pre-processing, e.g. for cross-validation.

Note that penalized B-splines instead of smoothing splines are used as default baselearners as of version 1.1-0.

Value

An object of class `gamboost` with `print`, `AIC` and `predict` methods being available.

References

Peter Buhlmann and Bin Yu (2003), Boosting with the L2 loss: regression and classification. *Journal of the American Statistical Association*, **98**, 324–339.

Peter Buhlmann and Torsten Hothorn (2007), Boosting algorithms: regularization, prediction and model fitting. *Statistical Science*, **22**(4), 477–505.

Thomas Kneib, Torsten Hothorn and Gerhard Tutz (2009), Variable selection and model choice in geoadditive regression models. *Biometrics*, accepted. <http://epub.ub.uni-muenchen.de/2063/>

Matthias Schmid and Torsten Hothorn (2009), Boosting additive models using component-wise P-splines as base-learners. *Computational Statistics & Data Analysis*, accepted. <http://epub.ub.uni-muenchen.de/2057/>

See Also

`glmboost` for boosted linear models and `blackboost` for boosted trees. See e.g. `bbs` for possible base-learners. See `cvrisk` for cross-validated stopping iteration. Furthermore see `boost_control`, `Family` and `methods`

Examples

```
### a simple two-dimensional example: cars data
cars.gb <- gamboost(dist ~ speed, data = cars, dfbase = 4,
                   control = boost_control(mstop = 50))

cars.gb
AIC(cars.gb, method = "corrected")

### plot fit for mstop = 1, ..., 50
plot(dist ~ speed, data = cars)
tmp <- sapply(1:mstop(AIC(cars.gb)), function(i)
             lines(cars$speed, predict(cars.gb[i]), col = "red"))
lines(cars$speed, predict(smooth.spline(cars$speed, cars$dist),
                          cars$speed)$y, col = "green")

### artificial example: sinus transformation
x <- sort(runif(100)) * 10
y <- sin(x) + rnorm(length(x), sd = 0.25)
```

```

plot(x, y)
### linear model
lines(x, fitted(lm(y ~ sin(x) - 1)), col = "red")
### GAM
lines(x, fitted(gamboost(y ~ x - 1,
                        control = boost_control(mstop = 500))),
      col = "green")

```

glmboost

Gradient Boosting with Component-wise Linear Models

Description

Gradient boosting for optimizing arbitrary loss functions where component-wise linear models are utilized as base learners.

Usage

```

## S3 method for class 'formula':
glmboost(formula, data = list(), weights = NULL,
         contrasts.arg = NULL, na.action = na.omit, ...)
## S3 method for class 'matrix':
glmboost(x, y, weights = NULL, ...)
glmboost_fit(object, family = GaussReg(),
             control = boost_control(), weights = NULL)
## S3 method for class 'glmboost':
plot(x, main = deparse(x$call), col = NULL, ...)

```

Arguments

formula	a symbolic description of the model to be fit.
data	a data frame containing the variables in the model.
weights	an optional vector of weights to be used in the fitting process.
contrasts.arg	a list, whose entries are contrasts suitable for input to the <code>contrasts</code> replacement function and whose names are the names of columns of data containing factors. See model.matrix.default .
na.action	a function which indicates what should happen when the data contain NAs.
x	design matrix or an object of class <code>glmboost</code> for plotting.
y	vector of responses.
object	an object of class <code>boost_data</code> , see boost_dpp .
family	an object of class <code>boost_family-class</code> , implementing the negative gradient corresponding to the loss function to be minimized. By default, squared error loss for continuous responses is used.

<code>control</code>	an object of class <code>boost_control</code> .
<code>main</code>	a title for the plot.
<code>col</code>	(a vector of) colors for plotting the lines representing the coefficient paths.
<code>...</code>	additional arguments passed to callies.

Details

A (generalized) linear model is fitted using a boosting algorithm based on component-wise univariate linear models. The fit, i.e., the regression coefficients, can be interpreted in the usual way. The methodology is described in Buhlmann and Yu (2003), Buhlmann (2006), and Buhlmann and Hothorn (2007).

The function `glmboost_fit` provides access to the fitting procedure without data pre-processing, e.g. for cross-validation.

Value

An object of class `glmboost` with `print`, `coef`, `AIC` and `predict` methods being available. For inputs with longer variable names, you might want to change `par("mai")` before calling the `plot` method of `glmboost` objects visualizing the coefficients path.

References

Peter Buhlmann and Bin Yu (2003), Boosting with the L2 loss: regression and classification. *Journal of the American Statistical Association*, **98**, 324–339.

Peter Buhlmann (2006), Boosting for high-dimensional linear models. *The Annals of Statistics*, **34**(2), 559–583.

Peter Buhlmann and Torsten Hothorn (2007), Boosting algorithms: regularization, prediction and model fitting. *Statistical Science*, **22**(4), 477–505.

See Also

`gamboost` for boosted smooth models and `blackboost` for boosted trees. See `cvrisk` for cross-validated stopping iteration. Furthermore see `boost_control`, `Family` and `methods`

Examples

```
### a simple two-dimensional example: cars data
cars.gb <- glmboost(dist ~ speed, data = cars,
                   control = boost_control(mstop = 5000))

cars.gb

### coefficients should coincide
coef(cars.gb) + c(cars.gb$offset, 0)
coef(lm(dist ~ speed, data = cars))

### plot fit
layout(matrix(1:2, ncol = 2))
plot(dist ~ speed, data = cars)
```

```

lines(cars$speed, predict(cars.gb), col = "red")

### alternative loss function: absolute loss
cars.gbl <- glmboost(dist ~ speed, data = cars,
                    control = boost_control(mstop = 5000),
                    family = Laplace())

cars.gbl

coef(cars.gbl) + c(cars.gbl$offset, 0)
lines(cars$speed, predict(cars.gbl), col = "green")

### Huber loss with adaptive choice of delta
cars.gbh <- glmboost(dist ~ speed, data = cars,
                    control = boost_control(mstop = 5000),
                    family = Huber())

lines(cars$speed, predict(cars.gbh), col = "blue")
legend("topleft", col = c("red", "green", "blue"), lty = 1,
      legend = c("Gaussian", "Laplace", "Huber"), bty = "n")

### plot coefficient path of glmboost
par(mai = par("mai") * c(1, 1, 1, 2.5))
plot(cars.gb)

```

IPCweights

Inverse Probability of Censoring Weights

Description

Compute weights for censored regression via the inverted probability of censoring principle.

Usage

```
IPCweights(x, maxweight = 5)
```

Arguments

`x` an object of class `Surv`.
`maxweight` the maximal value of the returned weights.

Details

Inverse probability of censoring weights are one possibility to fit models formulated in the *full data world* in the presence of censoring, i.e., the *observed data world*, see van der Laan and Robins (2003) for the underlying theory and Hothorn et al. (2006) for an application to survival analysis.

Value

A vector of numeric weights.

References

Mark J. van der Laan and James M. Robins (2003), *Unified Methods for Censored Longitudinal Data and Causality*, Springer, New York.

Torsten Hothorn, Peter Buhlmann, Sandrine Dudoit, Annette Molinaro and Mark J. van der Laan (2006), Survival ensembles. *Biostatistics* 7(3), 355–373.

Peter Buhlmann and Torsten Hothorn (2007), Boosting algorithms: regularization, prediction and model fitting. *Statistical Science*, 22(4), 477–505.

methods

Methods for Gradient Boosting Objects

Description

Methods for models fitted by boosting algorithms.

Usage

```
## S3 method for class 'glmboost':
print(x, ...)
## S3 method for class 'gamboost':
print(x, ...)
## S3 method for class 'glmboost':
coef(object, ...)
## S3 method for class 'gamboost':
coef(object, ...)
## S3 method for class 'gamboost':
AIC(object, method = c("corrected", "classical", "gMDL"), ..., k = 2)
## S3 method for class 'glmboost':
AIC(object, method = c("corrected", "classical", "gMDL"),
     df = c("trace", "actset"), ..., k = 2)
## S3 method for class 'gbAIC':
mstop(object, ...)
## S3 method for class 'gb':
mstop(object, ...)
## S3 method for class 'cvrisk':
mstop(object, ...)
## S3 method for class 'blackboost':
mstop(object, ...)
## S3 method for class 'gb':
predict(object, newdata = NULL, type = c("lp", "response"),
        allIterations = FALSE, ...)
## S3 method for class 'blackboost':
predict(object, newdata = NULL, type = c("lp", "response"),
        allIterations = FALSE, ...)
## S3 method for class 'gb':
fitted(object, type = c("lp", "response"), ...)
```

```
## S3 method for class 'gb':
logLik(object, ...)
```

Arguments

<code>object</code>	objects of class <code>glmboost</code> , <code>gamboost</code> , <code>blackboost</code> or <code>gbAIC</code> .
<code>x</code>	objects of class <code>glmboost</code> or <code>gamboost</code> .
<code>newdata</code>	optionally, a data frame in which to look for variables with which to predict.
<code>type</code>	a character indicating whether the fit or the response (classes) should be predicted in case of classification problems.
<code>allIterations</code>	computes the (linear) predictor for all boosting iterations simultaneously and returns a matrix.
<code>method</code>	a character specifying if the corrected AIC criterion or a classical ($-2 \log\text{Lik} + k * \text{df}$) should be computed.
<code>df</code>	a character specifying how degrees of freedom should be computed: <code>trace</code> defines degrees of freedom by the trace of the boosting hat matrix and <code>actset</code> uses the number of non-zero coefficients for each boosting iteration.
<code>k</code>	numeric, the <i>penalty</i> per parameter to be used; the default <code>k = 2</code> is the classical AIC. Only used when <code>method = "classical"</code> .
<code>...</code>	additional arguments passed to callies.

Details

These functions can be used to extract details from fitted models. `print` shows a dense representation of the model fit and `coef` extracts the regression coefficients of a linear model fitted using the `glmboost` function or the `gamboost` function.

The `predict` function can be used to predict the status of the response variable for new observations whereas `fitted` extracts the regression fit for the observations in the learning sample. When `allIterations = TRUE`, the matrix of all (linear) predictors for boosting iterations 1 to `mstop` is returned.

For (generalized) linear and additive models, the AIC function can be used to compute both the classical and corrected AIC (Hurvich et al., 1998, only available when `family = GaussReg()` was used), which is useful for the determination of the optimal number of boosting iterations to be applied (which can be extracted via `mstop`). The degrees of freedom are either computed via the trace of the boosting hat matrix (which is rather slow even for moderate sample sizes) or the number of variables (non-zero coefficients) that entered the model so far (faster but only meaningful for linear models fitted via `gamboost` (see Hastie, 2007).

In addition, the general Minimum Description Length criterion (Buhlmann and Yu, 2006) can be computed using function `AIC`.

Note that `logLik` and `AIC` only make sense when the corresponding `Family` implements the appropriate loss function.

References

Clifford M. Hurvich, Jeffrey S. Simonoff and Chih-Ling Tsai (1998), Smoothing parameter selection in nonparametric regression using an improved Akaike information criterion. *Journal of the Royal Statistical Society, Series B*, **20**(2), 271–293.

Peter Buhlmann and Torsten Hothorn (2007), Boosting algorithms: regularization, prediction and model fitting. *Statistical Science*, **22**(4), 477–505.

Travor Hastie (2007), Discussion of “Boosting Algorithms: Regularization, Prediction and Model Fitting” by Peter Buhlmann and Torsten Hothorn. *Statistical Science*, **22**(4), 505.

Peter Buhlmann and Bin Yu (2006), Sparse Boosting. *Journal of Machine Learning Research*, **7**, 1001–1024.

See Also

[gamboost](#), [glmboost](#) and [blackboost](#) for model fitting. See [cvrisk](#) for cross-validated stopping iteration.

Examples

```
### a simple two-dimensional example: cars data
cars.gb <- glmboost(dist ~ speed, data = cars,
                   control = boost_control(mstop = 2000))
cars.gb

### initial number of boosting iterations
mstop(cars.gb)

### AIC criterion
aic <- AIC(cars.gb, method = "corrected")
aic

### coefficients for optimal number of boosting iterations
coef(cars.gb[mstop(aic)])
plot(cars$dist, predict(cars.gb[mstop(aic)]),
     ylim = range(cars$dist))
abline(a = 0, b = 1)
```

Description

Computes the predicted survivor function for a Cox proportional hazards model.

Usage

```
## S3 method for class 'gb':
survFit(object, newdata = NULL, ...)
## S3 method for class 'blackboost':
survFit(object, newdata = NULL, ...)
## S3 method for class 'survFit':
plot(x, xlab = "Time", ylab = "Probability", ...)
```

Arguments

object	an object of class <code>blackboost</code> , <code>gamboost</code> , or <code>glmboost</code> which is assumed to have a CoxPH family component.
newdata	an optional data frame in which to look for variables with which to predict the survivor function.
x	an object of class <code>survFit</code> for plotting.
xlab	the label of the x axis.
ylab	the label of the y axis.
...	additional arguments passed to callies.

Details

If `newdata = NULL`, the survivor function of the Cox proportional hazards model is computed for the mean of the covariates used in the [blackboost](#), [gamboost](#), or [glmboost](#) call. The Breslow estimator is used for computing the baseline survivor function. If `newdata` is a data frame, the [predict](#) method of `object`, along with the Breslow estimator, is used for computing the predicted survivor function for each row in `newdata`.

Value

An object of class `survFit` containing the following components:

surv	the estimated survival probabilities at the time points given in <code>time</code> .
time	the time points at which the survivor functions are evaluated.
n.event	the number of events observed at each time point given in <code>time</code> .

See Also

[gamboost](#), [glmboost](#) and [blackboost](#) for model fitting.

Examples

```
fm <- Surv(futime, fustat) ~ age + resid.ds + rx + ecog.ps
fit <- glmboost(fm, data = ovarian, family = CoxPH(),
               control=boost_control(mstop = 500))

S1 <- survFit(fit)
S1
newdata <- ovarian[c(1,3,12),]
```

```
S2 <- survFit(fit, newdata = newdata)
S2

plot(S1)
```

Westbc

Breast Cancer Gene Expression

Description

Gene expressions for 7129 genes in 49 breast cancer samples and the status of lymph node involvement.

Usage

```
data("Westbc")
```

Format

An list with two elements to be converted to class `ExpressionSet` (see package `Biobase`).

Details

A full description of the data can be found in West et al. (2001) and an application of boosted linear models is given by Buhlmann (2006).

Source

Mike West, Carrie Blanchette, Holly Dressman, Erich Huang, Seiichi Ishida, Rainer Spang, Harry Zuzan, John A. Olson Jr., Jeffrey R. Marks and Joseph R. Nevins (2001), Predicting the clinical status of human breast cancer by using gene expression profiles, *Proceedings of the National Academy of Sciences*, **98**, 11462-11467. <http://data.cgt.duke.edu/west.php>

References

Peter Buhlmann (2006), Boosting for high-dimensional linear models. *The Annals of Statistics*, **34**(2), 559–583.

Peter Buhlmann and Torsten Hothorn (2007), Boosting algorithms: regularization, prediction and model fitting. *Statistical Science*, **22**(4), 477–505.

Examples

```
## Not run:
library("Biobase")
data("Westbc", package = "mboost")
westbc <- new("ExpressionSet",
             phenoData = new("AnnotatedDataFrame", data = Westbc$pheno),
             assayData = assayDataNew(exprs = Westbc$assay))
```

```
## End(Not run)
```

```
wpbc
```

```
Wisconsin Prognostic Breast Cancer Data
```

Description

Each record represents follow-up data for one breast cancer case. These are consecutive patients seen by Dr. Wolberg since 1984, and include only those cases exhibiting invasive breast cancer and no evidence of distant metastases at the time of diagnosis.

Usage

```
data("wpbc")
```

Format

A data frame with 198 observations on the following 34 variables.

```
status a factor with levels N (nonrecur) and R (recur)
time recurrence time (for status == "R") or disease-free time (for status == "N").
mean_radius radius (mean of distances from center to points on the perimeter) (mean).
mean_texture texture (standard deviation of gray-scale values) (mean).
mean_perimeter perimeter (mean).
mean_area area (mean).
mean_smoothness smoothness (local variation in radius lengths) (mean).
mean_compactness compactness (mean).
mean_concavity concavity (severity of concave portions of the contour) (mean).
mean_concavepoints concave points (number of concave portions of the contour) (mean).
mean_symmetry symmetry (mean).
mean_fractaldim fractal dimension (mean).
SE_radius radius (mean of distances from center to points on the perimeter) (SE).
SE_texture texture (standard deviation of gray-scale values) (SE).
SE_perimeter perimeter (SE).
SE_area area (SE).
SE_smoothness smoothness (local variation in radius lengths) (SE).
SE_compactness compactness (SE).
SE_concavity concavity (severity of concave portions of the contour) (SE).
SE_concavepoints concave points (number of concave portions of the contour) (SE).
SE_symmetry symmetry (SE).
```

SE_fractaldim fractal dimension (SE).
 worst_radius radius (mean of distances from center to points on the perimeter) (worst).
 worst_texture texture (standard deviation of gray-scale values) (worst).
 worst_perimeter perimeter (worst).
 worst_area area (worst).
 worst_smoothness smoothness (local variation in radius lengths) (worst).
 worst_compactness compactness (worst).
 worst_concavity concavity (severity of concave portions of the contour) (worst).
 worst_concavepoints concave points (number of concave portions of the contour) (worst).
 worst_symmetry symmetry (worst).
 worst_fractaldim fractal dimension (worst).
 tsize diameter of the excised tumor in centimeters.
 pnodes number of positive axillary lymph nodes observed at time of surgery.

Details

The first 30 features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.

There are two possible learning problems: predicting `status` or predicting the time to recur.

1) Predicting field 2, outcome: R = recurrent, N = non-recurrent - Dataset should first be filtered to reflect a particular endpoint; e.g., recurrences before 24 months = positive, non-recurrence beyond 24 months = negative. - 86.3 previous version of this data.

2) Predicting Time To Recur (field 3 in recurrent records) - Estimated mean error 13.9 months using Recurrence Surface Approximation.

The data are originally available from the UCI machine learning repository, see <http://www.ics.uci.edu/~mllearn/databases/breast-cancer-wisconsin/>.

Source

W. Nick Street, Olvi L. Mangasarian and William H. Wolberg (1995). An inductive learning approach to prognostic prediction. In A. Prieditis and S. Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning*, pages 522–530, San Francisco, Morgan Kaufmann.

Peter Buhlmann and Torsten Hothorn (2007), Boosting algorithms: regularization, prediction and model fitting. *Statistical Science*, **22**(4), 477–505.

Examples

```
data("wpbc", package = "mboost")

### fit logistic regression model with 100 boosting iterations
coef(glmboost(status ~ ., data = wpbc[,colnames(wpbc) != "time"],
              family = Binomial()))
```

Index

- *Topic **classes**
 - boost_family-class, 12
 - *Topic **datagen**
 - boost_dpp, 11
 - FP, 16
 - *Topic **datasets**
 - birds, 6
 - bodyfat, 9
 - Westbc, 27
 - wpbc, 28
 - *Topic **methods**
 - methods, 23
 - *Topic **misc**
 - boost_control, 10
 - *Topic **models**
 - baselearners, 2
 - blackboost, 7
 - cvrisk, 13
 - Family, 15
 - gamboost, 18
 - glmboost, 20
 - *Topic **nonlinear**
 - gamboost, 18
 - *Topic **regression**
 - blackboost, 7
 - cvrisk, 13
 - glmboost, 20
 - *Topic **survival**
 - IPCweights, 23
- AdaExp (*Family*), 15
AIC, 19, 21
AIC.gamboost, 14
AIC.gamboost (*methods*), 23
AIC.glmboost, 14
AIC.glmboost (*methods*), 23
- baselearners, 2
bbs, 17–20
bbs (*baselearners*), 2
- Binomial (*Family*), 15
birds, 6
blackboost, 4, 7, 11, 16, 20, 22, 25, 27
blackboost_fit (*blackboost*), 7
bns, 18
bns (*baselearners*), 2
bodyfat, 9
bols, 11, 18
bols (*baselearners*), 2
boost_control, 8, 10, 13, 19–22
boost_dpp, 7, 11, 18, 21
boost_family, 8
boost_family-class, 16, 19, 21
boost_family-class, 12
brandom, 18
brandom (*baselearners*), 2
bspacial, 18
bspacial (*baselearners*), 2
bss, 18
bss (*baselearners*), 2
btree, 18
btree (*baselearners*), 2
- coef, 21
coef.gamboost (*methods*), 23
coef.glmboost (*methods*), 23
contrasts, 3
CoxPH, 26
CoxPH (*Family*), 15
ctree, 8
ctree_control, 3, 7
cvrisk, 8, 13, 20, 22, 25
- Family, 8, 12, 15, 20, 22, 25
fitted.gb (*methods*), 23
FP, 16
- gamboost, 4, 8, 11, 16, 17, 18, 22, 25, 27
gamboost_fit (*gamboost*), 18
GaussClass (*Family*), 15

GaussReg (*Family*), 15
gbm, 8
glmboost, 8, 11, 16, 20, 20, 25, 27
glmboost_fit (*glmboost*), 20

Huber (*Family*), 15

IPCweights, 23

Laplace (*Family*), 15
logLik.gb (*methods*), 23

mclapply, 13
methods, 8, 20, 22, 23
mfp, 17
model.matrix.default, 21
mstop (*methods*), 23

plot, 19
plot.gamboost (*gamboost*), 18
plot.glmboost (*glmboost*), 20
plot.survFit (*survFit*), 26
Poisson (*Family*), 15
predict, 8, 19, 21, 27
predict.blackboost (*methods*), 23
predict.gb (*methods*), 23
print, 8, 19, 21
print.gamboost (*methods*), 23
print.glmboost (*methods*), 23

QuantReg (*Family*), 15

rug, 19

show, boost_family-method
 (*boost_family-class*), 12
survFit, 26

title, 19
TreeControl, 3, 7

Westbc, 27
wpsc, 28