# Package 'gamboostLSS'

July 22, 2025

**Type** Package

**Title** Boosting Methods for 'GAMLSS'

**Version** 2.1-0

**Date** 2025-02-14

**Description** Boosting models for fitting generalized additive models for location, shape and scale ('GAMLSS') to potentially high dimensional data.

**Depends** R (>= 2.10.0), mboost (>= 2.8-0), stabs (>= 0.5-0), parallel

**Imports** graphics, grDevices, stats, utils

**Suggests** gamlss, gamlss.dist, survival, BayesX, R2BayesX, DirichletReg

**LazyLoad** yes

**LazyData** yes

**License** GPL-2

**URL** https://github.com/boost-R/gamboostLSS

**NeedsCompilation** no

**Author** Benjamin Hofner [aut, cre] (ORCID: <https://orcid.org/0000-0003-2810-3186>), Andreas Mayr [aut], Nora Fenske [aut], Janek Thomas [aut], Matthias Schmid [aut]

**Maintainer** Benjamin Hofner <benjamin.hofner@pei.de>

**Repository** CRAN

**Date/Publication** 2025-02-24 14:30:02 UTC

## Contents

---

gamboostLSS-package *Boosting algorithms for GAMLSS*

---

## Description

Boosting methods for fitting generalized additive models for location, scale and shape (GAMLSS).

## Details

This package uses boosting algorithms for fitting GAMLSS (generalized additive models for location, scale and shape). For information on GAMLSS theory see Rigby and Stasinopoulos (2005), or the information provided at https://www.gamlss.com/. For a tutorial on gamboostLSS see Hofner et al. (2015). Thomas et al. (2018) developed a novel non-cyclic approach to fit gamboostLSS models. This approach is suitable for the combination with stabsel and speeds up model tuning via cvrisk.

The fitting methods glmboostLSS and gamboostLSS, are alternatives for the algorithms provided with gamlss in the gamlss package. They offer shrinkage of effect estimates, intrinsic variable selecion and model choice for potentially high-dimensional data settings.

glmboostLSS (for linear effects) and gamboostLSS (for smooth effects) depend on their analogous companions glmboost and gamboost for generalized additive models (contained in package mboost, see Hothorn et al. 2010, 2015) and are similar in their usage.

The package includes some pre-defined GAMLSS distributions, but the user can also specify new distributions with Families.

A wide range of different base-learners is available for covariate effects (see baselearners) including linear (bols), non-linear (bbs), random (brandom) or spatial effects (bspatial or Markov random fields bmrf). Each bease-learner can be included seperately for each predictor. The selection of base-learnes is crucial as it implies the kind of effect the covariate has on each distribution parameter in the final GAMLSS.

## Author(s)

Benjamin Hofner, Andreas Mayr, Nora Fenske, Janek Thomas, Matthias Schmid

Maintainer: Benjamin Hofner <benjamin.hofner@pei.de>

## References

B. Hofner, A. Mayr, M. Schmid (2016). gamboostLSS: An R Package for Model Building and Variable Selection in the GAMLSS Framework. Journal of Statistical Software, 74(1), 1-31.

Available as `vignette("gamboostLSS_Tutorial")`.

Mayr, A., Fenske, N., Hofner, B., Kneib, T. and Schmid, M. (2012): Generalized additive models for location, scale and shape for high-dimensional data - a flexible approach based on boosting. Journal of the Royal Statistical Society, Series C (Applied Statistics) 61(3): 403-427.

M. Schmid, S. Potapov, A. Pfahlberg, and T. Hothorn. Estimation and regularization techniques for regression models with multidimensional prediction functions. Statistics and Computing, 20(2):139-150, 2010.

Rigby, R. A. and D. M. Stasinopoulos (2005). Generalized additive models for location, scale and shape (with discussion). Journal of the Royal Statistical Society, Series C (Applied Statistics), 54, 507-554.

Stasinopoulos, D. M. and R. A. Rigby (2007). Generalized additive models for location scale and shape (GAMLSS) in R. Journal of Statistical Software 23(7).

Buehlmann, P. and Hothorn, T. (2007). Boosting algorithms: Regularization, prediction and model fitting. Statistical Science, 22(4), 477–505.

Hothorn, T., Buehlmann, P., Kneib, T., Schmid, M. and Hofner, B. (2010). Model-based boosting 2.0. Journal of Machine Learning Research 11(Aug), 2109-2113.

Hothorn, T., Buehlmann, P., Kneib, T., Schmid, M. and Hofner, B. (2015). mboost: Model-based boosting. R package version 2.4-2. https://CRAN.R-project.org/package=mboost

Thomas, J., Mayr, A., Bischl, B., Schmid, M., Smith, A., and Hofner, B. (2018), Gradient boosting for distributional regression - faster tuning and improved variable selection via noncyclical updates. *Statistics and Computing*. 28: 673-687. doi:10.1007/s1122201797546
(Preliminary version: https://arxiv.org/abs/1611.10171).

## See Also

`gamboostLSS` and `glmboostLSS` for model fitting. Available distributions (families) are documented here: `Families`.

See also the `mboost` package for more on model-based boosting, or the `gamlss` package for the original GAMLSS algorithms provided by Rigby and Stasinopoulos.

## Examples

```
# Generate covariates
x1 <- runif(100)
x2 <- runif(100)
eta_mu <-     2 - 2*x1
eta_sigma <-  -1  + 2*x2

# Generate response: Negative Binomial Distribution
y <- numeric(100)
for( i in 1:100)  y[i] <- rnbinom(1, size=exp(eta_sigma[i]), mu=exp(eta_mu[i]))

# Model fitting, 300 boosting steps, same formula for both distribution parameters
```

```
mod1 <- glmboostLSS( y ~ x1 + x2, families=NBinomialLSS(),
        control=boost_control(mstop=300), center = TRUE)

# Shrinked effect estimates
coef(mod1, off2int=TRUE)

# Empirical risk with respect to mu
plot(risk(mod1)$mu)

# Empirical risk with respect to sigma
plot(risk(mod1)$sigma)
```

---

`as.families` *Include* `gamlss` *families in the boosting framework of* `gamboostLSS`

---

## Description

The function `as.families()` provides an interface to apply the available distributions (families) of the `gamlss.dist` package for boosting GAMLSS via `gamboostLSS`.

The function automatically builds sub-families for every distribution parameter and uses the constructor function `Families` to build a `families` object, which can be then included in the fitting functions `gamboostLSS` and `glmboostLSS`.

## Usage

```
as.families(fname = "NO", stabilization = c("none", "MAD", "L2"),
            mu = NULL, sigma = NULL, nu = NULL, tau = NULL,
            mu.link = NULL, sigma.link = NULL, nu.link = NULL,
            tau.link = NULL)

## a wrapper to as.families:
gamlss.Families(...)
```

## Arguments

| | |
|---|---|
| `fname` | name of the distribution in the `gamlss` framework, as specified in the `gamlss.dist` package (e.g., `"NO"` for a normal distribution with parameters `mu` and `sigma`). Alternatively, one can directly specify the function (i.e., `NO`) or the evaluated function `NO()`. |
| `mu` | possible offset value for parameter `mu`. |
| `sigma` | possible offset value for parameter `sigma`. |
| `nu` | possible offset value for parameter `nu`. |
| `tau` | possible offset value for parameter `tau`. |
| `mu.link` | different link function for parameter `mu`. |
| `sigma.link` | different link function for parameter `sigma`. |

| | |
|---|---|
| nu.link | different link function for parameter nu. |
| tau.link | different link function for parameter tau. |
| stabilization | governs if the negative gradient should be standardized in each boosting step. It can be either "none" or "MAD". For details see Families. |
| ... | same arguments as above. |

## Details

The function aims at providing an interface to include all available GAMLSS distributions which are implemented with the original gamlss.dist package in the model-based boosting framework. The user specifies the name of the family (as it is called in gamlss.dist), and the function automatically builds the corresponding mboost-like sub-families and the final families object, which can be then used with the fitting functions gamboostLSS and glmboostLSS.

If no different link functions are specified, the standard links for the corresponding family in gamlss.dist are applied.

To extract the necessary information regarding partial derivatives (for the ngradient - see Family for details) and the log-likelihood (for the loss) the gamlss.dist package is loaded. If the package is not installed yet, this will prompt an error message.

The functions gamlss1parMu, gamlss2parMu, gamlss2parSigma, ... , gamlss4parTau are called internally to construct the sub-families. For one-parametric distributions, the function will prompt a warning and returns a mboost family, which can be then used by the fitting functions of the mboost package.

For information on GAMLSS theory see Rigby and Stasinopoulos (2005), lists of available distributions are provided at https://www.gamlss.com/. For more on details boosting GAMLSS see Mayr et al. (2012). Hofner et al. (2016) provides a worked example and more details on as.families.

To (potentially) stabilize the model estimation by standardizing the negative gradients one can use the argument stabilization of the families. See Families for details.

## Value

An object of class families. If the user specifies a one-parametric distribution, an object of class family is returned.

## Author(s)

The help of Mikis Stasinpoulos during the work on this function is gratefully acknowledged.

## References

B. Hofner, A. Mayr, M. Schmid (2016). gamboostLSS: An R Package for Model Building and Variable Selection in the GAMLSS Framework. Journal of Statistical Software, 74(1), 1-31.

Available as vignette("gamboostLSS_Tutorial").

Mayr, A., Fenske, N., Hofner, B., Kneib, T. and Schmid, M. (2012): Generalized additive models for location, scale and shape for high-dimensional data - a flexible approach based on boosting. Journal of the Royal Statistical Society, Series C (Applied Statistics) 61(3): 403-427.

Rigby, R. A. and D. M. Stasinopoulos (2005). Generalized additive models for location, scale and shape (with discussion). Journal of the Royal Statistical Society, Series C (Applied Statistics), 54, 507-554.

## See Also

`gamlss.dist` for available distributions in the `gamlss` framework.

`Families` for a documentation of pre-implemented distributions for `gamboostLSS`, as well as possibilities for user-defined distributions.

## Examples

```
## simulate small example
set.seed(123)
x <- runif(1000)

y <- rnorm(mean = 2 + 3 * x,        # effect on mu
           sd   = exp( 1 - 1 * x ), # effect on sigma
           n    = 1000)

## boosting
glmss <- glmboostLSS(y ~ x, families = as.families("NO"))
## the same:
if (require("gamlss.dist")) {
    glmss <- glmboostLSS(y ~ x, families = as.families(NO))
    glmss <- glmboostLSS(y ~ x, families = as.families(NO()))
}

coef(glmss, off2int = TRUE)

## compare to gamlss
library(gamlss)
glmss2 <- gamlss(y ~ x, sigma.formula = ~x, family = "NO")
coef(glmss2)
glmss2$sigma.coef
```

---

`cvrisk.mboostLSS` *Cross-Validation*

---

## Description

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

## Usage

```
## S3 method for class 'mboostLSS'
cvrisk(object, folds = cv(model.weights(object)),
       grid = make.grid(mstop(object)), papply = mclapply,
```

```
       trace = TRUE, mc.preschedule = FALSE, fun = NULL, ...)

make.grid(max, length.out = 10, min = NULL, log = TRUE,
          dense_mu_grid = TRUE)

## S3 method for class 'nc_mboostLSS'
cvrisk(object, folds = cv(model.weights(object)),
       grid = 1:sum(mstop(object)), papply = mclapply,
       trace = TRUE, mc.preschedule = FALSE, fun = NULL, ...)

## S3 method for class 'cvriskLSS'
plot(x, type = c("heatmap", "lines"),
     xlab = NULL, ylab = NULL, ylim = range(x),
     main = attr(x, "type"), ...)

## S3 method for class 'nc_cvriskLSS'
plot(x, xlab = "Number of boosting iterations", ylab = NULL,
     ylim = range(x), main = attr(x, "type"), ...)
```

## Arguments

| | |
|---|---|
| object | an object of class mboostLSS (i.e., a boosted GAMLSS model with method = "cyclic") or class nc_mboostLSS (i.e., a boosted GAMLSS model with method = "noncyclic") |
| folds | 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. Can be computed using function cv from package **mboost** and defaults to 25 bootstrap samples. |
| grid | If the model was fitted with method = "cyclic", grid is a matrix of stopping parameters the empirical risk is to be evaluated for. Each row represents a parameter combination. The number of columns must be equal to the number of parameters of the GAMLSS family. Per default, make.grid(mstop(object)) is used.<br>Otherwise (i.e., for method = "noncyclic") grid is a vector of mstop values. Per default all steps up to the intial stopping iteration, i.e., 1:mstop(object) are used. |
| papply | (parallel) apply function, defaults to mclapply. Alternatively, parLapply can be used. In the latter case, usually more setup is needed. To run cvrisk sequentially (i.e. not in parallel), one can use lapply. |
| trace | should status information beein printed during cross-validation? Default: TRUE. |
| mc.preschedule | preschedule tasks if are parallelized using mclapply (default: FALSE)? For details see mclapply. |
| fun | if fun is NULL, the out-of-sample risk is returned. fun, as a function of object, may extract any other characteristic of the cross-validated models. These are returned as is. |
| ... | additional arguments passed to mclapply or the plot function depending on the context. |

| | |
|---|---|
| max | a named vector of length equal to the number of parameters of the GAMLSS family (and names equal to the names of families) that determines the maximal values of the grid. |
| length.out | the number of grid points (default: 10). This can be either a vector of the same length as max (with different values) or a scalar (which is then used as length for all grids). |
| min | minimal value of the grid. Per default the grid starts at 1 but other values (smaller max) are possible. This can be either a vector of the same length as max (with different values) or a scalar (which is then used as min for all grids). |
| log | should the grid be on a logarithmic scale? Default: TRUE. |
| dense_mu_grid | should the grid in the mu component be extended for all values of the mstop values corresponding to mu that are greater or equal to all other parameters in this combination. These values can be computed without or with very little additional computational costs. For details see examples. |
| x | an object of class cvriskLSS (cyclic fitting) or nc_cvriskLSS (non-cyclic fitting), which results from running cvrisk. |
| type | should "lines" or a "heatmap" (default) be plotted? See details. |
| xlab, ylab | user-specified labels for the x-axis and y-axis of the plot (which are usually not needed). The defaults depend on the plot type. |
| ylim | limits of the y-axis. Only applicable for the line plot. |
| main | a title for the plots. |

## 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. For details see cvrisk.mboost.

make.grid eases the creation of an equidistand, integer-valued grids, which can be used with cvrisk. Per default, the grid is equidistant on a logarithmic scale.

The line plot depicts the avarage risk for each grid point and additionally shows information on the variability of the risk from fold to fold. The heatmap shows only the average risk but in a nicer fashion.

For the method = "noncyclic" only the line plot exists.

Hofner et al. (2016) provide a detailed description of cross-validation for gamboostLSS models and show a worked example. Thomas et al. (2018) compare cross-validation for the the cyclic and non-cyclic boosting approach and provide worked examples.

## Value

An object of class cvriskLSS or nc_cvriskLSS for cyclic and non-cyclic fitting, respectively, (when fun wasn't specified); 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 an mstop method.

## References

B. Hofner, A. Mayr, M. Schmid (2016). gamboostLSS: An R Package for Model Building and Variable Selection in the GAMLSS Framework. Journal of Statistical Software, 74(1), 1-31.

Available as `vignette("gamboostLSS_Tutorial")`.

Thomas, J., Mayr, A., Bischl, B., Schmid, M., Smith, A., and Hofner, B. (2018), Gradient boosting for distributional regression - faster tuning and improved variable selection via noncyclical updates. *Statistics and Computing*. 28: 673-687. doi:10.1007/s1122201797546
(Preliminary version: https://arxiv.org/abs/1611.10171).

## See Also

`cvrisk.mboost` and `cv` (both in package **mboost**)

## Examples

```
## Data generating process:
set.seed(1907)
x1 <- rnorm(1000)
x2 <- rnorm(1000)
x3 <- rnorm(1000)
x4 <- rnorm(1000)
x5 <- rnorm(1000)
x6 <- rnorm(1000)
mu    <- exp(1.5 +1 * x1 +0.5 * x2 -0.5 * x3 -1 * x4)
sigma <- exp(-0.4 * x3 -0.2 * x4 +0.2 * x5 +0.4 * x6)
y <- numeric(1000)
for( i in 1:1000)
    y[i] <- rnbinom(1, size = sigma[i], mu = mu[i])
dat <- data.frame(x1, x2, x3, x4, x5, x6, y)

## linear model with y ~ . for both components: 100 boosting iterations
model <- glmboostLSS(y ~ ., families = NBinomialLSS(), data = dat,
                     control = boost_control(mstop = 100),
                     center = TRUE)

## set up a grid
grid <-  make.grid(mstop(model), length.out = 5, dense_mu_grid = FALSE)
plot(grid)

### Do not test the following code per default on CRAN as it takes some time to run:
### a tiny toy example (5-fold bootsrap with maximum stopping value 100)
## (to run it on multiple cores of a Linux or Mac OS computer remove
##  set papply = mclapply (default) and set mc.nodes to the
##  appropriate number of nodes)
cvr <- cvrisk(model, folds = cv(model.weights(model), B = 5),
              papply = lapply, grid = grid)
cvr
## plot the results
par(mfrow = c(1, 2))
plot(cvr)
plot(cvr, type = "lines")
```

```
## extract optimal mstop (here: grid to small)
mstop(cvr)
### END (don't test automatically)


### Do not test the following code per default on CRAN as it takes some time to run:
### a more realistic example
grid <- make.grid(c(mu = 400, sigma = 400), dense_mu_grid = FALSE)
plot(grid)
cvr <- cvrisk(model, grid = grid)
mstop(cvr)
## set model to optimal values:
mstop(model) <- mstop(cvr)
### END (don't test automatically)


### Other grids:
plot(make.grid(mstop(model), length.out = 3, dense_mu_grid = FALSE))
plot(make.grid(c(mu = 400, sigma = 400), log = FALSE, dense_mu_grid = FALSE))
plot(make.grid(c(mu = 400, sigma = 400), length.out = 4,
               min = 100, log = FALSE, dense_mu_grid = FALSE))


### Now use dense mu grids
# standard grid
plot(make.grid(c(mu = 100, sigma = 100), dense = FALSE),
     pch = 20, col = "red")
# dense grid for all mstop_mu values greater than mstop_sigma
grid <- make.grid(c(mu = 100, sigma = 100))
points(grid, pch = 20, cex = 0.2)
abline(0,1)

# now with three parameters
grid <- make.grid(c(mu = 100, sigma = 100, df = 30),
                  length.out = c(5, 5, 2), dense = FALSE)
densegrid <- make.grid(c(mu = 100, sigma = 100, df = 30),
                       length.out = c(5, 5, 2))
par(mfrow = c(1,2))
# first for df = 1
plot(grid[grid$df == 1, 1:2], main = "df = 1", pch = 20, col = "red")
abline(0,1)
abline(v = 1)
# now expand grid for all mu values greater the corresponding sigma
# value (i.e. below the bisecting line) and above df (i.e. 1)
points(densegrid[densegrid$df == 1, 1:2], pch = 20, cex = 0.2)

# now for df = 30
plot(grid[grid$df == 30, 1:2], main = "df = 30", pch = 20, col = "red")
abline(0,1)
abline(v = 30)
# now expand grid for all mu values greater the corresponding sigma
# value (i.e. below the bisecting line) and above df (i.e. 30)
points(densegrid[densegrid$df == 30, 1:2], pch = 20, cex = 0.2)
```

Families *Families for GAMLSS models*

## Description

The package provides some pre-defined GAMLSS families, e.g. `NBionomialLSS`. Objects of the class `families` provide a convenient way to specify GAMLSS distributions to be fitted by one of the boosting algorithms implemented in this package. By using the function `Families`, a new object of the class `families` can be generated.

## Usage

```
############################################################
# Families for continuous response

# Gaussian distribution
GaussianLSS(mu = NULL, sigma = NULL,
            stabilization = c("none", "MAD", "L2"))

# Student's t-distribution
StudentTLSS(mu = NULL, sigma = NULL, df = NULL,
            stabilization = c("none", "MAD", "L2"))

############################################################
# Families for continuous non-negative response

# Gamma distribution
GammaLSS(mu = NULL, sigma = NULL,
         stabilization = c("none", "MAD", "L2"))

############################################################
# Families for fractions and bounded continuous response

# Beta distribution
BetaLSS(mu = NULL, phi = NULL,
        stabilization = c("none", "MAD", "L2"))

############################################################
# Families for count data

# Negative binomial distribution
NBinomialLSS(mu = NULL, sigma = NULL,
             stabilization = c("none", "MAD", "L2"))

# Zero-inflated Poisson distribution
ZIPoLSS(mu = NULL, sigma = NULL,
        stabilization = c("none", "MAD", "L2"))
```

```
# Zero-inflated negative binomial distribution
ZINBLSS(mu = NULL, sigma = NULL, nu = NULL,
        stabilization = c("none", "MAD", "L2"))

############################################################
# Families for survival models (accelerated failure time
# models) for data with right censoring

# Log-normal distribution
LogNormalLSS(mu = NULL, sigma = NULL,
             stabilization = c("none", "MAD", "L2"))

# Log-logistic distribution
LogLogLSS(mu = NULL, sigma = NULL,
          stabilization = c("none", "MAD", "L2"))

# Weibull distribution
WeibullLSS(mu = NULL, sigma = NULL,
           stabilization = c("none", "MAD", "L2"))

############################################################
# Family for Dirichlet regression models
DirichletLSS(K = NULL, stabilization = c("none", "MAD", "L2"))

############################################################
# Constructor function for new GAMLSS distributions
Families(..., qfun = NULL, name = NULL)
```

## Arguments

| | |
|---|---|
| `...` | sub-families to be passed to constructor. |
| `qfun` | quantile function. This function can for example be used to compute (marginal) prediction intervals. See `predint`. |
| `name` | name of the families. |
| `mu` | offset value for mu. |
| `sigma` | offset value for sigma. |
| `phi` | offset value for phi. |
| `df` | offset value for df. |
| `nu` | offset value for nu. |
| `stabilization` | governs if the negative gradient should be standardized in each boosting step. It can be either "none", "MAD" or "L2". See also Details below. |
| `K` | An integer specifying the number of categories in the Dirichlet distribution. This must be provided. |

## Details

The arguments of the families are the offsets for each distribution parameter. Offsets can be either scalar, a vector with length equal to the number of observations or NULL (default). In the latter case, a scalar offset for this component is computed by minimizing the risk function w.r.t. the corresponding distribution parameter (keeping the other parameters fixed).

Note that gamboostLSS is not restricted to three components but can handle an arbitrary number of components (which, of course, depends on the GAMLSS distribution). However, it is important that the names (for the offsets, in the sub-families etc.) are chosen *consistently*.

The ZIPoLSS families can be used to fit zero-inflated Poisson models. Here, mu and sigma refer to the location parameter of the Poisson component (with log link) and the mean of the zero-generating process (with logit link), respectively.

Similarly, ZINBLSS can be used to fit zero-inflated negative binomial models. Here, mu and sigma refer to the location and scale parameters (with log link) of the negative binomial component of the model. The zero-generating process (with logit link) is represented by nu.

The DirichletLSS family can be used to fit Dirichlet regression models. Here, DirichletAlpha corresponds to the distributional parameters in the Dirichlet distribution which are dependent on the number of categories in the respective compositional data (that is, proportions, amounts or rates.) The number of categories, thereby distributional parameters in the data set (K) has to be specified manually in the beginning of the fitting process of the model.

The Families function can be used to implements a new GAMLSS distribution which can be used for fitting by mboostLSS. Thereby, the function builds a list of sub-families, one for each distribution parameter. The sub-families themselves are objects of the class boost_family, and can be constructed via the function Family of the **mboost** Package.

Arguments to be passed to Family: The loss for every distribution parameter (contained in objects of class boost_family) is the negative log-likelihood of the corresponding distribution. The ngradient is the negative partial derivative of the loss function with respect to the distribution parameter. For a two-parameter distribution (e.g. mu and sigma), the user therefore has to specify two sub-families with Family. The loss is basically the same function for both paramters, only ngradient differs. Both sub-families are passed to the Families constructor, which returns an object of the class families.

To (potentially) stabilize the model estimation by standardizing the negative gradients one can use the argument stabilization of the families. If stabilization = "MAD", the negative gradient is divided by its (weighted) median absolute deviation

$$median_i(|u_{k,i} - median_j(u_{k,j})|)$$

in each boosting step. See Hofner et al. (2016) for details. An alternative is stabilization = "L2", where the gradient is divided by its (weighted) mean L2 norm. This results in negative gradient vectors (and hence also updates) of similar size for each distribution parameter, but also for every boosting iteration.

## Value

An object of class families.

## Author(s)

BetaLSS for boosting beta regression was implemented by Florian Wickler. DirichletLSS for boosting Dirichlet regression models was implemented by Michael Balzer.

## References

B. Hofner, A. Mayr, M. Schmid (2016). gamboostLSS: An R Package for Model Building and Variable Selection in the GAMLSS Framework. Journal of Statistical Software, 74(1), 1-31.

Available as vignette("gamboostLSS_Tutorial").

Mayr, A., Fenske, N., Hofner, B., Kneib, T. and Schmid, M. (2012): Generalized additive models for location, scale and shape for high-dimensional data - a flexible approach based on boosting. Journal of the Royal Statistical Society, Series C (Applied Statistics) 61(3): 403-427.

Rigby, R. A. and D. M. Stasinopoulos (2005). Generalized additive models for location, scale and shape (with discussion). Journal of the Royal Statistical Society, Series C (Applied Statistics), 54, 507-554.

## See Also

as.families for applying GAMLSS distributions provided in the framework of the gamlss package.

The functions gamboostLSS and glmboostLSS can be used for model fitting.

See also the corresponding constructor function Family in mboost.

## Examples

```
## Example to define a new distribution:
## Students t-distribution with two parameters, df and mu:

## sub-Family for mu
## -> generate object of the class family from the package mboost
newStudentTMu  <- function(mu, df){

    # loss is negative log-Likelihood, f is the parameter to be fitted with
    # id link -> f = mu
    loss <- function(df,  y, f) {
        -1 * (lgamma((df + 1)/2)  - lgamma(1/2) -
              lgamma(df/2) - 0.5 * log(df) -
              (df + 1)/2 * log(1 + (y - f)^2/(df )))
    }
    # risk is sum of loss
    risk <- function(y, f, w = 1) {
        sum(w * loss(y = y, f = f, df = df))
    }
    # ngradient is the negative derivate w.r.t. mu (=f)
    ngradient <- function(y, f, w = 1) {
        (df + 1) * (y - f)/(df  + (y - f)^2)
    }

    # use the Family constructor of mboost
```

```
    mboost::Family(ngradient = ngradient, risk = risk, loss = loss,
                   response = function(f) f,
                   name = "new Student's t-distribution: mu (id link)")
}

## sub-Family for df
newStudentTDf <- function(mu, df){

    # loss is negative log-Likelihood, f is the parameter to be fitted with
    # log-link: exp(f) = df
    loss <- function( mu, y, f) {
        -1 * (lgamma((exp(f) + 1)/2)  - lgamma(1/2) -
              lgamma(exp(f)/2) - 0.5 * f -
              (exp(f) + 1)/2 * log(1 + (y - mu)^2/(exp(f) )))
    }
    # risk is sum of loss
    risk <- function(y, f, w = 1) {
        sum(w * loss(y = y, f = f,  mu = mu))
    }
    # ngradient is the negative derivate of the loss w.r.t. f
    # in this case, just the derivative of the log-likelihood
    ngradient <- function(y, f, w = 1) {
        exp(f)/2 * (digamma((exp(f) + 1)/2) - digamma(exp(f)/2)) -
            0.5 - (exp(f)/2 * log(1 + (y - mu)^2 / (exp(f) )) -
                   (y - mu)^2 / (1 + (y - mu)^2 / exp(f)) * (exp(-f) + 1)/2)
    }
    # use the Family constructor of mboost
    mboost::Family(ngradient = ngradient, risk = risk, loss = loss,
                   response = function(f) exp(f),
                   name = "Student's t-distribution: df (log link)")
}

## families object for new distribution
newStudentT <- Families(mu= newStudentTMu(mu=mu, df=df),
                        df=newStudentTDf(mu=mu, df=df))

### Do not test the following code per default on CRAN as it takes some time to run:
### usage of the new Student's t distribution:
library(gamlss)   ## required for rTF
set.seed(1907)
n <- 5000
x1  <- runif(n)
x2 <- runif(n)
mu <- 2 -1*x1 - 3*x2
df <- exp(1 + 0.5*x1 )
y <- rTF(n = n, mu = mu, nu = df)

## model fitting
model <- glmboostLSS(y ~ x1 + x2, families = newStudentT,
                     control = boost_control(mstop = 100),
                     center = TRUE)
## shrinked effect estimates
coef(model, off2int = TRUE)
```

```
## compare to pre-defined three parametric t-distribution:
model2 <- glmboostLSS(y ~ x1 + x2, families = StudentTLSS(),
                      control = boost_control(mstop = 100),
                      center = TRUE)
coef(model2, off2int = TRUE)

## with effect on sigma:
sigma <- 3+ 1*x2
y <- rTF(n = n, mu = mu, nu = df, sigma=sigma)
model3 <- glmboostLSS(y ~ x1 + x2, families = StudentTLSS(),
                      control = boost_control(mstop = 100),
                      center = TRUE)
coef(model3, off2int = TRUE)
```

---

india *Malnutrition of Children in India (DHS, 1998-99)*

---

## Description

Data sample from the Standard Demographic and Health Survey, 1998-99, on malnutrition of children in India. The data set contains approximately 12% of the observations in the original data set and only a (very small) subset of variables. Additionally, a boundary file representing the districts of India is provided for spatial analysis.

## Usage

```
data(india)
data(india.bnd)
```

## Format

A data frame with 4000 observations on the following 6 variables:

`stunting` A numeric z-score for malnutrition, stunted growth to be more precise, which ranges from -6 to 6, where negative values represent malnourished children. Children with values below -2 are considered stunted (height-for-age).

`cbmi` BMI of the child.

`cage` Age of the child in months.

`mbmi` BMI of the mother.

`mage` Age of the mother in years.

`mcdist` The district in India, where mother and child live. A factor encoded to match the map `india.bnd`.

`mcdist_lab` The district in India, where mother and child live. A factor with actual district names.

### Details

For details on the boundary file see function `read.bnd` from package **BayesX**.

### Source

The complete data set is provided by the Monitoring and Evaluation to Assess and Use Results Demographic and Health Surveys (MEASURE DHS) which is funded by the U.S. Agency of International Development (USAID). It can be obtained for research purposes (after registration) from https://dhsprogram.com/data/dataset/India_Standard-DHS_1999.cfm (Data set for All-India, Children's Recode: iakr42dt.zip)

### References

For details on the data set see also:

Fahrmeir L and Kneib T (2011), *Bayesian smoothing and regression for longitudinal, spatial and event history data*, Oxford University Press.

### Examples

```
if (require("BayesX")) {
  ## plot distribution of stunting in India
  drawmap(india, map = india.bnd, regionvar = "mcdist", plotvar = "stunting")
}
```

---

## mboostLSS *Fitting GAMLSS by Boosting*

---

### Description

Functions for fitting GAMLSS (generalized additive models for location, scale and shape) using boosting techniques. Two algorithms are implemented: (a) The cyclic algorithm iteratively rotates between the distribution parameters, updating one while using the current fits of the others as offsets (for details see Mayr et al., 2012). (b) The noncyclic algorithm selects in each step the update of a base-learner for the distribution parameter that best fits the negative gradient (algorithm with inner loss of Thomas et al., 2018).

### Usage

```
mboostLSS(formula, data = list(), families = GaussianLSS(),
          control = boost_control(), weights = NULL,
          method = c("cyclic", "noncyclic"), ...)
glmboostLSS(formula, data = list(), families = GaussianLSS(),
            control = boost_control(), weights = NULL,
            method = c("cyclic", "noncyclic"), ...)
gamboostLSS(formula, data = list(), families = GaussianLSS(),
            control = boost_control(), weights = NULL,
            method = c("cyclic", "noncyclic"), ...)
```

```
blackboostLSS(formula, data = list(), families = GaussianLSS(),
              control = boost_control(), weights = NULL,
              method = c("cyclic", "noncyclic"), ...)

## fit function:
mboostLSS_fit(formula, data = list(), families = GaussianLSS(),
              control = boost_control(), weights = NULL,
              fun = mboost, funchar = "mboost", call = NULL, method, ...)
```

## Arguments

| | |
|---|---|
| formula | a symbolic description of the model to be fit. See mboost for details. If formula is a single formula, the same formula is used for all distribution parameters. formula can also be a (named) list, where each list element corresponds to one distribution parameter of the GAMLSS distribution. The names must be the same as in the family (see example for details). |
| data | a data frame containing the variables in the model. |
| families | an object of class families. It can be either one of the pre-defined distributions that come along with the package or a new distribution specified by the user (see Families for details). Per default, we use the two-parametric GaussianLSS family. |
| control | a list of parameters controlling the algorithm. For more details see boost_control. |
| weights | a numeric vector of weights (optional). |
| method | fitting method, currently two methods are supported: "cyclic" (see Mayr et al., 2012) and "noncyclic" (algorithm with inner loss of Thomas et al., 2018). The latter requires a one dimensional mstop value. |
| fun | fit function. Either mboost, glmboost, gamboost or blackboost. Specified directly via the corresponding LSS function. E.g. gamboostLSS() calls mboostLSS_fit(..., fun = gamboost). |
| funchar | character representation of fit function. Either "mboost", "glmboost", "gamboost" or "blackboost". Specified directly via the corresponding LSS function. |
| call | used to forward the call from mboostLSS, glmboostLSS, gamboostLSS and blackboostLSS. This argument should not be directly specified by users! |
| ... | Further arguments to be passed to mboostLSS_fit. In mboostLSS_fit, ... represent further arguments to be passed to mboost and mboost_fit. So ... can be all arguments of mboostLSS_fitand mboost_fit. |

## Details

For information on GAMLSS theory see Rigby and Stasinopoulos (2005) or the information provided at https://www.gamlss.com/. For a tutorial on gamboostLSS see Hofner et al. (2016). Thomas et al. (2018) developed a novel non-cyclic approach to fit gamboostLSS models. This approach is suitable for the combination with stabsel and speeds up model tuning via cvrisk (see also below).

glmboostLSS uses glmboost to fit the distribution parameters of a GAMLSS – a linear boosting model is fitted for each parameter.

gamboostLSS uses gamboost to fit the distribution parameters of a GAMLSS – an additive boosting model (by default with smooth effects) is fitted for each parameter. With the formula argument, a wide range of different base-learners can be specified (see baselearners). The base-learners imply the type of effect each covariate has on the corresponding distribution parameter.

mboostLSS uses mboost to fit the distribution parameters of a GAMLSS. The type of model (linear, tree-based or smooth) is specified by fun.

blackboostLSS uses blackboost to fit the distribution parameters of a GAMLSS – a tree-based boosting model is fitted for each parameter.

mboostLSS, glmboostLSS, gamboostLSS and blackboostLSS all call mboostLSS_fit while fun is the corresponding **mboost** function, i.e., the same function without LSS. For further possible arguments see these functions as well as mboost_fit. Note that mboostLSS_fit is usually not called directly by the user.

For method = "cyclic" it is possible to specify one or multiple mstop and nu values via boost_control. In the case of one single value, this value is used for all distribution parameters of the GAMLSS model. Alternatively, a (named) vector or a (named) list with separate values for each component can be used to specify a separate value for each parameter of the GAMLSS model. The names of the list must correspond to the names of the distribution parameters of the GAMLSS family. If no names are given, the order of the mstop or nu values is assumed to be the same as the order of the components in the families. For one-dimensional stopping, the user therefore can specify, e.g., mstop = 100 via boost_control. For more-dimensional stopping, one can specify, e.g., mstop = list(mu = 100, sigma = 200) (see examples).

If method is set to "noncyclic", mstop has to be a one dimensional integer. Instead of cycling through all distribution parameters, in each iteration only the best base-learner is used. One base-learner of every parameter is selected via RSS, the distribution parameter is then chosen via the loss (in Thomas et. al., 2018, called inner loss). For details on the noncyclic fitting method see Thomas et. al. (2018).

To (potentially) stabilize the model estimation by standardizing the negative gradients one can use the argument stabilization of the families. See Families for details.

## Value

An object of class mboostLSS or nc_mboostLSS (inheriting from class mboostLSS) for models fitted with method = "cyclic" and method = "non-cyclic", respectively, with corresponding methods to extract information. A mboostLSS model object is a named list with one list entry for each modelled distribution parameter. Special "subclasses" inheriting from mboostLSS exist for each of the model-types (with the same name as the function, e.g., gamboostLSS).

## References

B. Hofner, A. Mayr, M. Schmid (2016). gamboostLSS: An R Package for Model Building and Variable Selection in the GAMLSS Framework. Journal of Statistical Software, 74(1), 1-31.

Available as vignette("gamboostLSS_Tutorial").

Mayr, A., Fenske, N., Hofner, B., Kneib, T. and Schmid, M. (2012): Generalized additive models for location, scale and shape for high-dimensional data - a flexible approach based on boosting. Journal of the Royal Statistical Society, Series C (Applied Statistics) 61(3): 403-427.

M. Schmid, S. Potapov, A. Pfahlberg, and T. Hothorn. Estimation and regularization techniques for regression models with multidimensional prediction functions. Statistics and Computing, 20(2):139-150, 2010.

Rigby, R. A. and D. M. Stasinopoulos (2005). Generalized additive models for location, scale and shape (with discussion). Journal of the Royal Statistical Society, Series C (Applied Statistics), 54, 507-554.

Buehlmann, P. and Hothorn, T. (2007), Boosting algorithms: Regularization, prediction and model fitting. Statistical Science, 22(4), 477–505.

Thomas, J., Mayr, A., Bischl, B., Schmid, M., Smith, A., and Hofner, B. (2018), Gradient boosting for distributional regression - faster tuning and improved variable selection via noncyclical updates. *Statistics and Computing*. 28: 673-687. doi:10.1007/s1122201797546
(Preliminary version: https://arxiv.org/abs/1611.10171).

## See Also

Families for a documentation of available GAMLSS distributions.

The underlying boosting functions mboost, gamboost, glmboost, blackboost are contained in the **mboost** package.

See for example risk or coef for methods that can be used to extract information from mboostLSS objects.

## Examples

```
### Data generating process:
set.seed(1907)
x1 <- rnorm(1000)
x2 <- rnorm(1000)
x3 <- rnorm(1000)
x4 <- rnorm(1000)
x5 <- rnorm(1000)
x6 <- rnorm(1000)
mu    <- exp(1.5 +1 * x1 +0.5 * x2 -0.5 * x3 -1 * x4)
sigma <- exp(-0.4 * x3 -0.2 * x4 +0.2 * x5 +0.4 * x6)
y <- numeric(1000)
for( i in 1:1000)
    y[i] <- rnbinom(1, size = sigma[i], mu = mu[i])
dat <- data.frame(x1, x2, x3, x4, x5, x6, y)

### linear model with y ~ . for both components: 400 boosting iterations
model <- glmboostLSS(y ~ ., families = NBinomialLSS(), data = dat,
                     control = boost_control(mstop = 400),
                     center = TRUE)
coef(model, off2int = TRUE)


### estimate model with different formulas for mu and sigma:
names(NBinomialLSS())      # names of the family

### Do not test the following code per default on CRAN as it takes some time to run:
# Note: Multiple formulas must be specified via a _named list_
```

```
#       where the names correspond to the names of the distribution parameters
#       in the family (see above)
model2 <- glmboostLSS(formula = list(mu = y ~ x1 + x2 + x3 + x4,
                                     sigma = y ~ x3 + x4 + x5 + x6),
                      families = NBinomialLSS(), data = dat,
                      control = boost_control(mstop = 400, trace = TRUE),
                      center = TRUE)
coef(model2, off2int = TRUE)
### END (don't test automatically)



### Offset needs to be specified via the arguments of families object:
model <- glmboostLSS(y ~ ., data = dat,
                      families = NBinomialLSS(mu = mean(mu),
                                              sigma = mean(sigma)),
                      control = boost_control(mstop = 10),
                      center = TRUE)
# Note: mu-offset = log(mean(mu)) and sigma-offset = log(mean(sigma))
#       as we use a log-link in both families
coef(model)
log(mean(mu))
log(mean(sigma))

### Do not test the following code per default on CRAN as it takes some time to run:
### use different mstop values for the two distribution parameters
### (two-dimensional early stopping)
### the number of iterations is passed to boost_control via a named list
model3 <- glmboostLSS(formula = list(mu = y ~ x1 + x2 + x3 + x4,
                                     sigma = y ~ x3 + x4 + x5 + x6),
                      families = NBinomialLSS(), data = dat,
                      control = boost_control(mstop = list(mu = 400,
                                                           sigma = 300),
                                              trace  = TRUE),
                      center = TRUE)
coef(model3, off2int = TRUE)

### Alternatively we can change mstop of model2:
# here it is assumed that the first element in the vector corresponds to
# the first distribution parameter of model2 etc.
mstop(model2) <- c(400, 300)
par(mfrow = c(1,2))
plot(model2, xlim = c(0, max(mstop(model2))))
## all.equal(coef(model2), coef(model3)) # same!
### END (don't test automatically)
```

---

methods *Methods for mboostLSS*

---

## Description

Methods for GAMLSS models fitted by boosting algorithms.

## Usage

```
### print model
## S3 method for class 'mboostLSS'
print(x, ...)

### summarize model
## S3 method for class 'mboostLSS'
summary(object, ...)

### extract coefficients
## S3 method for class 'glmboostLSS'
coef(object, which = NULL,
     aggregate = c("sum", "cumsum", "none"),
     off2int = FALSE, parameter = names(object), ...)
## S3 method for class 'mboostLSS'
coef(object, which = NULL,
     aggregate = c("sum", "cumsum", "none"),
     parameter = names(object), ...)

### plot partial effects
## S3 method for class 'glmboostLSS'
plot(x, main = names(x), parameter = names(x),
     off2int = FALSE, ...)
## S3 method for class 'gamboostLSS'
plot(x, main = names(x), parameter = names(x), ...)

### extract and plot marginal prediction intervals
predint(x, which, pi = 0.9, newdata = NULL, ...)
PI(x, which, pi = 0.9, newdata = NULL, ...)
## S3 method for class 'predint'
plot(x, main = "Marginal Prediction Interval(s)",
     xlab = NULL, ylab = NULL, lty = c("solid", "dashed"),
     lcol = c("black", "black"), log = "", ...)

### extract mstop
## S3 method for class 'mboostLSS'
mstop(object, parameter = names(object), ...)
## S3 method for class 'oobag'
mstop(object, parameter = names(object), ...)
## S3 method for class 'cvriskLSS'
mstop(object, parameter = NULL, ...)

### set mstop
## S3 method for class 'mboostLSS'
```

```
x[i, return = TRUE, ...]

### extract risk
## S3 method for class 'mboostLSS'
risk(object, merge = FALSE, parameter = names(object), ...)

### extract selected base-learners
## S3 method for class 'mboostLSS'
selected(object, merge = FALSE, parameter = names(object), ...)

### extract fitted values
## S3 method for class 'mboostLSS'
fitted(object, parameter = names(object), ...)

### make predictions
## S3 method for class 'mboostLSS'
predict(object, newdata = NULL,
        type = c("link", "response", "class"), which = NULL,
        aggregate = c("sum", "cumsum", "none"),
        parameter = names(object), ...)

### update weights of the fitted model
## S3 method for class 'mboostLSS'
update(object, weights, oobweights = NULL,
       risk = NULL, trace = NULL, mstop = NULL, ...)

### extract model weights
## S3 method for class 'mboostLSS'
model.weights(x, ...)
```

## Arguments

| | |
|---|---|
| x, object | an object of the appropriate class (see usage). |
| which | a subset of base-learners to take into account when computing predictions or coefficients. If which is given (as an integer vector or characters corresponding to base-learners), a list or matrix is returned. In plot_PI the argument which must be specified and it must be given as a character string containing the name of the variable. |
| aggregate | a character specifying how to aggregate predictions or coefficients of single base-learners. The default returns the prediction or coefficient for the final number of boosting iterations. "cumsum" returns a matrix with the predictions for all iterations simultaneously (in columns). "none" returns a list with matrices where the $j$th columns of the respective matrix contains the predictions of the base-learner of the $j$th boosting iteration (and zero if the base-learner is not selected in this iteration). |
| parameter | This can be either a vector of indices or a vector of parameter names which |

| | |
|---|---|
| | should be processed. See expamles for details. Per default all distribution parameters of the GAMLSS family are returned. |
| `off2int` | logical indicating whether the offset should be added to the intercept (if there is any) or if the offset is neglected for plotting (default). |
| `merge` | logical. Should the risk vectors of the single components be merged to one risk vector for the model in total? Per default (`merge = FALSE`) a (named) list of risk vectors is returned. |
| `i` | integer. Index specifying the model to extract. If `i` is smaller than the initial `mstop`, a subset is used. If `i` is larger than the initial `mstop`, additional boosting steps are performed until step `i` is reached. One can specify a scalar, a (possibly named) vector or a (possibly named) list with separate values for each component. See the details section of `mboostLSS` for more information. |
| `return` | a logical indicating whether the changed object is returned. |
| `main` | a title for the plots. |
| `xlab, ylab` | x- and y axis labels for the plots. |
| `pi` | the level(s) of the prediction interval(s); Per default a 90% prediction interval is used. |
| `lty` | (vector) of line types to be used for plotting the prediction intervals. The vector should contain `length(pi) + 1` elements. If less elements are specified, the last element is recycled. The first value `lty[1]` is used for the marginal median, the second value `lty[2]` is used for the `pi[1]` prediction interval, etc. |
| `lcol` | (vector) of (line) colors to be used for plotting the prediction intervals. The vector should contain `length(pi) + 1` elements. If less elements are specified, the last element is recycled. The first value `lcol[1]` is used for the marginal median, the second value `lcol[2]` is used for the `pi[1]` prediction interval, etc. |
| `log` | a character string which determines if and if so which axis should be logarithmic. See `plot.default` for details. |
| `newdata` | optional; A data frame in which to look for variables with which to predict or with which to plot the marginal prediction intervals. |
| `type` | the type of prediction required. The default is on the scale of the predictors; the alternative `"response"` is on the scale of the response variable. Thus for a binomial model the default predictions are on the log-odds scale (probabilities on logit scale) and `type = "response"` gives the predicted probabilities. The `"class"` option returns predicted classes. |
| `weights` | a numeric vector of weights for the model |
| `oobweights` | an additional vector of out-of-bag weights (used internally by `cvrisk`. For details see there.). |
| `risk` | a character indicating how the empirical risk should be computed for each boosting iteration. Per default `risk` is set to the risk type specified for model fitting via `boost_control`. For details and alternatives see there. |
| `trace` | a logical triggering printout of status information during the fitting process. |
| `mstop` | number of boosting iterations. |
| `...` | Further arguments to the functions. |

## Details

These functions can be used to extract details from fitted models. For a tutorial with worked examples see Hofner et al. (2016).

`print` shows a dense representation of the model fit.

The function `coef` extracts the regression coefficients of linear predictors fitted using the `glmboostLSS` function or additive predictors fitted using `gamboostLSS`. Per default, only coefficients of selected base-learners are returned for all distribution parameters. However, any desired coefficient can be extracted using the `which` argument. Furhtermore, one can extract only coefficients for a single distribution parameter via the `parameter` argument (see examples for details).

Analogical, the function `plot` per default displays the coefficient paths for the complete GAMLSS but can be restricted to single distribution parameters or covariates (or subsets) using the `parameter` or `which` arguments, respectively.

The function `predint` (or `PI` which is just an alias) computes marginal prediction intervals and returns a data frame with the predictors used for the marginal prediction interval, the computed median prediction and the marginal prediction intervals. A plot function (`plot.predint`) for the resulting object exists. Note that marginal predictions from AFT models (i.e., families `LogLogLSS`, `LogNormalLSS`, and `WeibullLSS`) represent the predicted "true" survival time and not the observed survival time which is possible subject to censoring. Hence, comparing observed survival times with the marginal prediction interval is only sensible for uncensored observations.

The `predict` function can be used for predictions for the distribution parameters depending on new observations whereas `fitted` extracts the regression fits for the observations in the learning sample. For `predict`, `newdata` can be specified – otherwise the fitted values are returned. If `which` is specified, marginal effects of the corresponding base-learner(s) are returned. The argument `type` can be used to make predictions on the scale of the link (i.e., the linear predictor X * beta), the `response` (i.e. h(X * beta), where h is the response function) or the `class` (in case of classification).

The function `update` updates models fit with **gamboostLSS** and is primarily used within `cvrisk`. It updates the weights and refits the model to the altered data. Furthermore, the type of `risk`, the `trace` and the number of boosting iterations `mstop` can be modified.

The function `model.weights` is a generic version of the same function provided by package **stats**, which is required to make `model.weights` work with `mboostLSS` models.

## Warning

The `[.mboostLSS` function changes the original object, i.e., `LSSmodel[10]` changes `LSSmodel` directly!

## References

B. Hofner, A. Mayr, M. Schmid (2016). gamboostLSS: An R Package for Model Building and Variable Selection in the GAMLSS Framework. Journal of Statistical Software, 74(1), 1-31.

Available as `vignette("gamboostLSS_Tutorial")`.

Mayr, A., Fenske, N., Hofner, B., Kneib, T. and Schmid, M. (2012): Generalized additive models for location, scale and shape for high-dimensional data - a flexible approach based on boosting. Journal of the Royal Statistical Society, Series C (Applied Statistics) 61(3): 403-427.

Buehlmann, P. and Hothorn, T. (2007), Boosting algorithms: regularization, prediction and model fitting. Statistical Science, 22(4), 477–505.

Rigby, R. A. and D. M. Stasinopoulos (2005). Generalized additive models for location, scale and shape (with discussion). Journal of the Royal Statistical Society, Series C (Applied Statistics), 54, 507-554.

## See Also

glmboostLSS, gamboostLSS and blackboostLSS for fitting of GAMLSS.

Available distributions (families) are documented here: Families.

See methods in the mboost package for the corresponding methods for mboost objects.

## Examples

```
### generate data
set.seed(1907)
x1 <- rnorm(1000)
x2 <- rnorm(1000)
x3 <- rnorm(1000)
x4 <- rnorm(1000)
x5 <- rnorm(1000)
x6 <- rnorm(1000)
mu    <- exp(1.5 + x1^2 +0.5 * x2 - 3 * sin(x3) -1 * x4)
sigma <- exp(-0.2 * x4 +0.2 * x5 +0.4 * x6)
y <- numeric(1000)
for( i in 1:1000)
    y[i] <- rnbinom(1, size = sigma[i], mu = mu[i])
dat <- data.frame(x1, x2, x3, x4, x5, x6, y)

### fit a model
model <- gamboostLSS(y ~ ., families = NBinomialLSS(), data = dat,
                     control = boost_control(mstop = 100))

### Do not test the following line per default on CRAN as it takes some time to run:
### use a model with more iterations for a better fit
mstop(model) <- 400

### extract coefficients
coef(model)

### only for distribution parameter mu
coef(model, parameter = "mu")

### only for covariate x1
coef(model, which = "x1")


### plot complete model
par(mfrow = c(4, 3))
plot(model)
### plot first parameter only
```

```
par(mfrow = c(2, 3))
plot(model, parameter = "mu")
### now plot only effect of x3 of both parameters
par(mfrow = c(1, 2))
plot(model, which = "x3")
### first component second parameter (sigma)
par(mfrow = c(1, 1))
plot(model, which = 1, parameter = 2)

### Do not test the following code per default on CRAN as it takes some time to run:
### plot marginal prediction interval
pi <- predint(model, pi = 0.9, which = "x1")
pi <- predint(model, pi = c(0.8, 0.9), which = "x1")
plot(pi, log = "y")  # warning as some y values are below 0
## here it would be better to plot x1 against
## sqrt(y) and sqrt(pi)

### set model to mstop = 300 (one-dimensional)
mstop(model) <- 300
### END (don't test automatically)


par(mfrow = c(2, 2))
plot(risk(model, parameter = "mu")[[1]])
plot(risk(model, parameter = "sigma")[[1]])

### Do not test the following code per default on CRAN as it takes some time to run:
### get back to orignal fit
mstop(model) <- 400
plot(risk(model, parameter = "mu")[[1]])
plot(risk(model, parameter = "sigma")[[1]])

### use different mstop values for the components
mstop(model) <- c(100, 200)
## same as
  mstop(model) <- c(mu = 100, sigma = 200)
## or
  mstop(model) <- list(mu = 100, sigma = 200)
## or
  mstop(model) <- list(100, 200)

plot(risk(model, parameter = "mu")[[1]])
plot(risk(model, parameter = "sigma")[[1]])
### END (don't test automatically)
```

---

`stabsel` *Stability Selection*

---

## Description

Selection of influential variables or model components with error control.

## Usage

```
## a method to compute stability selection paths for fitted mboostLSS models
## S3 method for class 'mboostLSS'
stabsel(x, cutoff, q, PFER, mstop = NULL,
        folds = subsample(model.weights(x), B = B),
        B = ifelse(sampling.type == "MB", 100, 50),
        assumption = c("unimodal", "r-concave", "none"),
        sampling.type = c("SS", "MB"),
        papply = mclapply, verbose = TRUE, FWER, eval = TRUE, ...)
## a method to get the selected parameters
## S3 method for class 'stabsel_mboostLSS'
selected(object, parameter = NULL, ...)
```

## Arguments

| | |
|---|---|
| x | an fitted model of class "mboostLSS" or "nc_mboostLSS". |
| cutoff | cutoff between 0.5 and 1. Preferably a value between 0.6 and 0.9 should be used. |
| q | number of (unique) selected variables (or groups of variables depending on the model) that are selected on each subsample. |
| PFER | upper bound for the per-family error rate. This specifies the amount of falsely selected base-learners, which is tolerated. See details. |
| mstop | mstop value to use, if no value is supplied the mstop value of the fitted model is used. |
| folds | a weight matrix with number of rows equal to the number of observations, see cvrisk and subsample. Usually one should not change the default here as subsampling with a fraction of $1/2$ is needed for the error bounds to hold. One usage scenario where specifying the folds by hand might be the case when one has dependent data (e.g. clusters) and thus wants to draw clusters (i.e., multiple rows together) not individuals. |
| assumption | Defines the type of assumptions on the distributions of the selection probabilities and simultaneous selection probabilities. Only applicable for sampling.type = "SS". For sampling.type = "MB" we always use "none". |
| sampling.type | use sampling scheme of of Shah & Samworth (2013), i.e., with complementarty pairs (sampling.type = "SS"), or the original sampling scheme of Meinshausen & Buehlmann (2010). |
| B | number of subsampling replicates. Per default, we use 50 complementary pairs for the error bounds of Shah & Samworth (2013) and 100 for the error bound derived in Meinshausen & Buehlmann (2010). As we use $B$ complementray pairs in the former case this leads to $2B$ subsamples. |
| papply | (parallel) apply function, defaults to mclapply. Alternatively, parLapply can be used. In the latter case, usually more setup is needed (see example of cvrisk for some details). |
| verbose | logical (default: TRUE) that determines wether warnings should be issued. |
| FWER | deprecated. Only for compatibility with older versions, use PFER instead. |

| | |
|---|---|
| eval | logical. Determines whether stability selection is evaluated (eval = TRUE; default) or if only the parameter combination is returned. |
| object | a object of class "stabsel_mboostLSS". |
| parameter | select one or multiple effects. |
| ... | additional arguments to parallel apply methods such as mclapply and to cvrisk. |

## Details

Stability selection is to be preferably used with non-cyclic gamboostLSS models, as proposed by Thomas et al. (2018). In this publication, the combination of package **gamboostLSS** with stability selection was devoloped and is investigated in depth.

For details on stability selection see stabsel in package **stabs** and Hofner et al. (2014).

## Value

An object of class stabsel with a special print method. The object has the following elements:

| | |
|---|---|
| phat | selection probabilities. |
| selected | elements with maximal selection probability greater cutoff. |
| max | maximum of selection probabilities. |
| cutoff | cutoff used. |
| q | average number of selected variables used. |
| PFER | per-family error rate. |
| sampling.type | the sampling type used for stability selection. |
| assumption | the assumptions made on the selection probabilities. |
| call | the call. |

## References

B. Hofner, L. Boccuto and M. Goeker (2015), Controlling false discoveries in high-dimensional situations: Boosting with stability selection. *BMC Bioinformatics*, **16:144**.

N. Meinshausen and P. Buehlmann (2010), Stability selection. *Journal of the Royal Statistical Society, Series B*, **72**, 417–473.

R.D. Shah and R.J. Samworth (2013), Variable selection with error control: another look at stability selection. *Journal of the Royal Statistical Society, Series B*, **75**, 55–80.

Thomas, J., Mayr, A., Bischl, B., Schmid, M., Smith, A., and Hofner, B. (2018), Gradient boosting for distributional regression - faster tuning and improved variable selection via noncyclical updates. *Statistics and Computing*. 28: 673-687. doi:10.1007/s1122201797546
(Preliminary version: https://arxiv.org/abs/1611.10171).

## See Also

stabsel and stabsel_parameters

## Examples

```
### Data generating process:
set.seed(1907)
x1 <- rnorm(500)
x2 <- rnorm(500)
x3 <- rnorm(500)
x4 <- rnorm(500)
x5 <- rnorm(500)
x6 <- rnorm(500)
mu    <- exp(1.5 +1 * x1 +0.5 * x2 -0.5 * x3 -1 * x4)
sigma <- exp(-0.4 * x3 -0.2 * x4 +0.2 * x5 +0.4 * x6)
y <- numeric(500)
for( i in 1:500)
    y[i] <- rnbinom(1, size = sigma[i], mu = mu[i])
dat <- data.frame(x1, x2, x3, x4, x5, x6, y)

### linear model with y ~ . for both components: 400 boosting iterations
model <- glmboostLSS(y ~ ., families = NBinomialLSS(), data = dat,
                     control = boost_control(mstop = 400),
                     center = TRUE, method = "noncyclic")

### Do not test the following code per default on CRAN as it takes some time to run:

#run stability selection
(s <- stabsel(model, q = 5, PFER = 1))
#get selected effects
selected(s)

#visualize selection frequencies
plot(s)

### END (don't test automatically)
```

---

`weighted.median` *Weighted Median*

---

## Description

Function to compute the weighted median.

## Usage

```
weighted.median(x, w = 1, na.rm = FALSE)
```

## Arguments

| | |
|---|---|
| `x` | a numeric vector containing the values whose median is to be computed. |

| | |
|---|---|
| w | weights that are used to compute the median. This can be either a single value (which will be used as weight for all observations) or a numeric vector of the same length as x. |
| na.rm | logical. Should NAs (from weights w and/or data x) be removed? |

## Details

The weighted median is computed as the value where the cumulative relative weights (relative to the sum of all weights) crosses 0.5.

This function is used in the stabilization of the negative gradient via the meadian absolute deviation (MAD). For details see Hofner et al (2015).

## References

B. Hofner, A. Mayr, M. Schmid (2016). gamboostLSS: An R Package for Model Building and Variable Selection in the GAMLSS Framework. Journal of Statistical Software, 74(1), 1-31.

Available as `vignette("gamboostLSS_Tutorial")`.

## See Also

`glmboostLSS`, `gamboostLSS` and `blackboostLSS` for fitting of GAMLSS where the standardization is explained in more detail.

## Examples

```
## compute the weighted median with case weights
x <- c(1, 2, 3, 4)
w <- c(0, 1, 2, 3)
weighted.median(x, w)

## compute the weighted median with arbitrary weights
x <- rnorm(100)
w <- runif(100)
weighted.median(x, w)
```

# Index