

Package ‘gamboostLSS’

February 14, 2012

Type Package

Title Boosting Methods for GAMLSS Models

Version 1.0-3

Date 2011-11-16

Author Benjamin Hofner, Andreas Mayr, Nora Fenske, Matthias Schmid

Maintainer Benjamin Hofner <benjamin.hofner@imbe.med.uni-erlangen.de>

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

Depends R (>= 2.10.0), splines, survival, methods, mboost (>= 2.0-12)

Suggests gamlss

LazyLoad yes

License GPL-2

Repository CRAN

Date/Publication 2011-11-17 11:25:49

R topics documented:

gamboostLSS-package	2
Families	3
mboostLSS	7
methods	11
Index	16

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 <http://gamlss.org>.

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 selection 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. 2010a, b) 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 (`bo1s`), non-linear (`bbs`), random (`brandom`) or spatial effects (`bspatial` or Markov random fields `bmrf`). Each base-learner can be included separately for each predictor. The selection of base-learners 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, Matthias Schmid

Maintainer: Benjamin Hofner <benjamin.hofner@imbe.med.uni-erlangen.de>

References

Mayr, A., Fenske, N., Hofner, B., Kneib, T. and Schmid, M. (2011): GAMLSS for high-dimensional data - a flexible approach based on boosting. *Journal of the Royal Statistical Society, Series C (Applied Statistics)*. Accepted.

Preliminary version: Department of Statistics, Technical Report 98. <http://epub.ub.uni-muenchen.de/11938/>

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. (2010a). 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. (2010b). *mboost*: Model-based boosting 2.0. R package version 2.0-9. <http://cran.r-project.org/web/packages/mboost>

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)
```

Families

Families for GAMLSS models

Description

The package provides some pre-defined GAMLSS families, e.g. `NBinomialLSS`. By using the function `Families`, a new object of the class `families` can be generated.

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.

Usage

```
## Constructor function for new GAMLSS distributions
Families(...)

### Pre-defined GAMLSS distributions ###

## families Object for the negative binomial distribution
NBinomialLSS(mu = NULL, sigma = NULL)
## 'Sub-families' for the two distribution parameters:
NBinomialMu(mu = NULL, sigma = NULL)
NBinomialSigma(mu = NULL, sigma = NULL)

## families Object for Student's t-distribution
StudentTLSS(mu = NULL, sigma = NULL, df = NULL)
## 'Sub-families' for the three distribution parameters:
StudentTMu(mu = NULL, sigma = NULL, df = NULL)
StudentTSigma(mu = NULL, sigma = NULL, df = NULL)
StudentTdf(mu = NULL, sigma = NULL, df = NULL)

## families Object for log-normal accelerated failure time model
LogNormalLSS(mu = NULL, sigma = NULL)
## 'Sub-families' for the two distribution parameters:
LogNormalMu(mu = NULL, sigma = NULL)
LogNormalSigma(mu = NULL, sigma = NULL)

## families Object for log-logistic accelerated failure time model
LogLogLSS(mu = NULL, sigma = NULL)
## 'Sub-families' for the two distribution parameters:
LogLogMu(mu = NULL, sigma = NULL)
LogLogSigma(mu = NULL, sigma = NULL)

## families Object for accelerated failure time model with Weibull distribution
WeibullLSS(mu = NULL, sigma = NULL)
## 'Sub-families' for the two distribution parameters:
WeibullMu(mu = NULL, sigma = NULL)
WeibullSigma(mu = NULL, sigma = NULL)
```

Arguments

mu	offset value for mu.
sigma	offset value for sigma.
df	offset value for df.
...	Sub-families to be passed to constructor.

Details

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 parameters, only `ngradient` differs. Both Sub-families are passed to the `Families` constructor, which returns an object of the class `families`.

The arguments of the final object 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 the three components `mu`, `sigma` and `df` 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*.

Value

An object of class `families`.

References

Mayr, A., Fenske, N., Hofner, B., Kneib, T. and Schmid, M. (2011): GAMLSS for high-dimensional data - a flexible approach based on boosting. *Journal of the Royal Statistical Society, Series C (Applied Statistics)*. Accepted.

Preliminary version: Department of Statistics, Technical Report 98. <http://epub.ub.uni-muenchen.de/11938/>

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

The corresponding constructor function `Family` in `mboost`. For the fitting of GAMLSS distributions see `mboostLSS`.

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
ngradient <- function(y, f, w = 1) {
  (df + 1) * (y - f)/(df + (y - f)^2)
}

# use the Family constructor of 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 w.r.t. df
  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
  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))

### 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)
# shrunked 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)

```

mboostLSS

Fitting GAMLSS by Boosting

Description

Functions for fitting GAMLSS (generalized additive models for location, scale and shape) using boosting techniques. The 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., 2010).

Usage

```

mboostLSS(formula, data = list(), families = list(),
          control = boost_control(), weights = NULL, ...)
glmboostLSS(formula, data = list(), families = list(),
            control = boost_control(), weights = NULL, ...)
gamboostLSS(formula, data = list(), families = list(),
            control = boost_control(), weights = NULL, ...)

```

```
blackboostLSS(formula, data = list(), families = list(),
              control = boost_control(), weights = NULL, ...)

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

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 <code>families</code> . 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).
control	a list of parameters controlling the algorithm. For more details see boost_control .
weights	a numeric vector of weights (optional).
fun	fit function. Either mboost , glmboost , gamboost or blackboost . Specified directly via the corresponding LSS function. E.g. <code>gamboostLSS()</code> calls <code>mboostLSS_fit(..., fun = gamboost)</code> .
...	Further arguments to be passed to <code>mboostLSS_fit</code> . In <code>mboostLSS_fit</code> , ... represent further arguments to be passed to mboost and mboost_fit . So ... can be all arguments of <code>mboostLSS_fit</code> and <code>mboost_fit</code> .

Details

For information on GAMLSS theory see Rigby and Stasinopoulos (2005) or the information provided at <http://gamlss.org>.

`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](#).

In all four fitting functions 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 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. 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).

Value

An object of class `mboostLSS` with corresponding methods to extract information.

References

Mayr, A., Fenske, N., Hofner, B., Kneib, T. and Schmid, M. (2011): GAMLSS for high-dimensional data - a flexible approach based on boosting. *Journal of the Royal Statistical Society, Series C (Applied Statistics)*. Accepted.

Preliminary version: Department of Statistics, Technical Report 98. <http://epub.ub.uni-muenchen.de/11938/>

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.

See Also

`mboost`, `gamboost`, `glmboost`, `blackboost` for the underlying boosting functions contained in `mboost`.

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

# 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)

### 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))

## Not run: ## (only as is takes some time)
### 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)

```

```
## End(Not run)
### Alternatively we can subset model2:
# here it is assumed that the first element in the vector corresponds to
# the first distribution parameter of model2 etc.
model2[c(400, 300)]
par(mfrow = c(1,2))
plot(model2, xlim = c(0, max(mstop(model2))))
## all.equal(coef(model2), coef(model3)) # same!

### WARNING: Subsetting via model[mstopnew] changes the model directly!
### For the original fit one has to subset again: model[mstop]
```

methods

Methods for mboostLSS

Description

Methods for GAMLSS models fitted by boosting algorithms.

Usage

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

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

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

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

## S3 method for class 'mboostLSS'
x[i, return = TRUE, ...]

## S3 method for class 'mboostLSS'
selected(object, parameter = names(object), ...)

## 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),
     ...)

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

```

Arguments

x	an object of the appropriate class (see usage).
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.
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 <i>j</i> th columns of the respective matrix contains the predictions of the base-learner of the <i>j</i> 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 examples 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>i</i> is smaller than the initial <i>mstop</i> , a subset is used. If <i>i</i> is larger than the initial <i>mstop</i> , additional boosting steps are performed until step <i>i</i> is reached. See details for more information.
return	a logical indicating whether the changed object is returned.
main	a title for the plots.
newdata	optionally; A data frame in which to look for variables with which to predict.
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.

... Further arguments to the functions.

Details

These functions can be used to extract details from fitted models. `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. Furthermore, 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 `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).

Warning

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

References

Mayr, A., Fenske, N., Hofner, B., Kneib, T. and Schmid, M. (2011): GAMLSS for high-dimensional data - a flexible approach based on boosting. *Journal of the Royal Statistical Society, Series C (Applied Statistics)*. Accepted.

Preliminary version: Department of Statistics, Technical Report 98. <http://epub.ub.uni-muenchen.de/11938/>

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. 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] <- rbinom(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 = 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)

### subset model for mstop = 300 (one-dimensional)
model[300]

# WARNING: Subsetting via model[mstopnew] changes the model directly!
# For the original fit one has to subset again: model[mstop]

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

```

```
plot(risk(model, parameter = "sigma")[[1]])  
### get back to original fit  
model[400]  
plot(risk(model, parameter = "mu")[[1]])  
plot(risk(model, parameter = "sigma")[[1]])
```

Index

- *Topic **distributions**
 - Families, [3](#)
- *Topic **fitting**
 - mboostLSS, [7](#)
- *Topic **methods**
 - methods, [11](#)
- *Topic **models**
 - Families, [3](#)
 - mboostLSS, [7](#)
- *Topic **nonlinear**
 - mboostLSS, [7](#)
- *Topic **package**
 - gamboostLSS-package, [2](#)
 - [.mboostLSS (methods), [11](#)

- baselearners, [2, 8](#)
- blackboost, [8, 9](#)
- blackboostLSS, [13](#)
- blackboostLSS (mboostLSS), [7](#)
- boost_control, [8, 9](#)

- coef, [9](#)
- coef.glmboostLSS (methods), [11](#)
- coef.mboostLSS (methods), [11](#)

- Families, [2, 3, 8](#)
- Family, [5](#)
- fitted.mboostLSS (methods), [11](#)

- gamboost, [2, 8, 9](#)
- gamboostLSS, [2, 13](#)
- gamboostLSS (mboostLSS), [7](#)
- gamboostLSS-package, [2](#)
- gamlss, [2, 3](#)
- glmboost, [2, 8, 9](#)
- glmboostLSS, [2, 13](#)
- glmboostLSS (mboostLSS), [7](#)

- LogLogLSS (Families), [3](#)
- LogLogMu (Families), [3](#)
- LogLogSigma (Families), [3](#)

- LogNormalLSS (Families), [3](#)
- LogNormalMu (Families), [3](#)
- LogNormalSigma (Families), [3](#)

- mboost, [2, 3, 5, 8, 9, 13](#)
- mboost_fit, [8](#)
- mboostLSS, [4, 5, 7, 9](#)
- mboostLSS_fit (mboostLSS), [7](#)
- methods, [11, 13](#)
- mstop.mboostLSS (methods), [11](#)
- mstop.oobag (methods), [11](#)

- NBinomialLSS (Families), [3](#)
- NBinomialMu (Families), [3](#)
- NBinomialSigma (Families), [3](#)

- plot.gamboostLSS (methods), [11](#)
- plot.glmboostLSS (methods), [11](#)
- predict.mboostLSS (methods), [11](#)

- risk, [9](#)
- risk (methods), [11](#)

- selected (methods), [11](#)
- StudentTDf (Families), [3](#)
- StudentTLSS (Families), [3](#)
- StudentTMu (Families), [3](#)
- StudentTSigma (Families), [3](#)

- WeibullLSS (Families), [3](#)
- WeibullMu (Families), [3](#)
- WeibullSigma (Families), [3](#)