Package 'FDboost'

July 2, 2014

Type Package

Title Boosting functional regression models

Version 0.0-5

Date 2013-04-29

Author Sarah Brockhaus

Maintainer Sarah Brockhaus <Sarah.Brockhaus@stat.uni-muenchen.de>

Description Boosting functional regression models

Depends R (>= 3.0.0), mboost

Imports Matrix, mgcv, zoo, nnls

Suggests fda, fields, maps, mapdata

License GPL-2

Collate 'FDboost.R' 'baselearners.R' 'crossvalidation.R' 'methods.R' 'bkroneckerTrafo.R' 'utilityFunctions.R'

NeedsCompilation no

Repository CRAN

Date/Publication 2014-04-22 20:37:15

R topics documented:

---

bbsc *Constrained Base-learners for Scalar Covariates*

Description

Constrained base-learners for fitting effects of scalar covariates in models with functional response

Usage

```
bbsc(..., by = NULL, index = NULL, knots = 10,
  boundary.knots = NULL, degree = 3, differences = 2,
  df = 4, lambda = NULL, center = FALSE, cyclic = FALSE)

bolsc(..., by = NULL, index = NULL, intercept = TRUE,
  df = NULL, lambda = 0, K = NULL,
  contrasts.arg = "contr.treatment")

brandomc(..., contrasts.arg = "contr.dummy", df = 4)
```

Arguments

...	one or more predictor variables or one matrix or data frame of predictor variables.
by	an optional variable defining varying coefficients, either a factor or numeric variable.
index	a vector of integers for expanding the variables in ....
knots	either the number of knots or a vector of the positions of the interior knots (for more details see bbs).

boundary.knots	boundary points at which to anchor the B-spline basis (default the range of the data). A vector (of length 2) for the lower and the upper boundary knot can be specified.
degree	degree of the regression spline.
differences	a non-negative integer, typically 1, 2 or 3. If differences = k, k-th-order differences are used as a penalty (0-th order differences specify a ridge penalty).
df	trace of the hat matrix for the base-learner defining the base-learner complexity. Low values of df correspond to a large amount of smoothing and thus to "weaker" base-learners.
lambda	smoothing penalty, computed from df when df is specified.
K	in bolsc it is possible to specify the penalty matrix K
center	not implemented yet
cyclic	if cyclic = TRUE the fitted values coincide at the boundaries (useful for cyclic covariates such as day time etc.).
contrasts.arg	Note that a special contrasts.arg exists in package mboost, namely "contr.dummy". This contrast is used per default in brandomc. It leads to a dummy coding as returned by model.matrix(~ x - 1) were the intercept is implicitly included but each factor level gets a separate effect estimate (for more details see brandom).
intercept	if intercept = TRUE an intercept is added to the design matrix of a linear base-learner.

Details

The base-learners bbsc, bolsc and brandomc are basically the base-learners bbs, bols and brandom with additional identifiability constraints. Instead of the default identifiability constraints ($\sum_{i,t} \hat{f}(x_i, t) = 0$) implemented in mboost for tensor product terms whose marginal terms include the index of the functional response t constraints that enforce $\sum_i \hat{f}(z_i, x_i, t) = 0$ for all t are used, so that effects varying over t can be interpreted as deviations from the global functional intercept.

Cannot deal with any missing values in the covariates.

Value

Equally to the base-learners of the package mboost:

An object of class blg (base-learner generator) with a dpp function.

The call of dpp returns an object of class bl (base-learner) with a fit function. The call to fit finally returns an object of class bm (base-model).

References

Scheipl, F., Staicu, A.-M., and Greven, S. (2014), Functional Additive Mixed Models, Journal of Computational and Graphical Statistics, in press, DOI 10.1080/10618600.2014.901914. http://arxiv.org/abs/1207.5947

See Also

FDboost for the model fit. bbs, bols and brandom for the corresponding base-learners in mboost.

bsignal *Base-learners for Functional Covarites*

Description

Base-learners that fit effects of functional covariates

Usage

```
bsignal(..., index = NULL, knots = 10,
  boundary.knots = NULL, degree = 3, differences = 2,
  df = 4, lambda = NULL, cyclic = FALSE, Z = NULL)

bconcurrent(..., knots = 10, boundary.knots = NULL,
  degree = 3, differences = 2, df = 4)

bhist(..., index = NULL, knots = 10,
  boundary.knots = NULL, degree = 3, differences = 2,
  df = 4, limits = "s<=t")
```

Arguments

`...`	matrix of functional data and the vector of observation points. The functional covariates have to be supplied as n by <no. of evaluations> matrices, i.e. each row is one functional observation. The base-learner `bhist` expects three arguments: functional covariate, index of functional covariate, index of functional response $[0, 1]$ is assumed.
`index`	a vector of integers for expanding the signal variable in `....` For example, bsignal(X, s, index = index) is equal to bsignal(X[index,], s), where index is an integer of length greater or equal to length(x).
`knots`	either the number of knots or a vector of the positions of the interior knots (for more details see `bbs`).
`boundary.knots`	boundary points at which to anchor the B-spline basis (default the range of the data). A vector (of length 2) for the lower and the upper boundary knot can be specified.
`degree`	degree of the regression spline.
`differences`	a non-negative integer, typically 1, 2 or 3. If `differences` = k, k-th-order differences are used as a penalty (0-th order differences specify a ridge penalty).
`df`	trace of the hat matrix for the base-learner defining the base-learner complexity. Low values of `df` correspond to a large amount of smoothing and thus to "weaker" base-learners.
`lambda`	smoothing penalty
`cyclic`	if cyclic = TRUE the fitted coefficient function coincides at the boundaries (useful for cyclic covariates such as day time etc.).

Z	a transformation matrix for the design-matrix over the index of the covariate. Z can be calculated as the transformation matrix for a sum-to-zero constraint in the case that all trajectories have the same mean (then a shift in the coefficient function is not identifiable).
limits	defaults to "s<=t" for an historical effect with s<=t, otherwise specifies the integration limits s_hi, i, s_lo, i: either one of "s<t" or "s<=t" for (s_hi, i, s_lo, i) = (0, t) or a function that takes s as the first and t as the second argument and returns TRUE for combinations of values (s,t) if s falls into the integration range for the given t. This is an experimental feature and not well tested yet; use at your own risk.

Details

`bsignal` implements a base-learner for functional covariates to estimate an effect of the form $X_i(s)\beta(t,s)ds$. Defaults to a cubic B-spline basis with second difference penalties for $\beta(t,s)$ in the direction of s and numerical integration over the entire range by using trapezoidal Riemann weights.

`bconcurrent` implements a concurrent effect for a functional covariate on a functional response, i.e. an effect of the form $X_i(t)\beta(t)$ for a functional response $Y_i(t)$.

It is recommended to use centered functional covariates with $\sum_i X_i(s) = 0$ for all s in `bconcurrent`- and `bsignal`-terms so that the global functional intercept can be interpreted as the global mean function.

Cannot deal with any missing values in the covariates.

Value

Equally to the base-learners of the package mboost:

An object of class `blg` (base-learner generator) with a `dpp` function.

The call of `dpp` returns an object of class `bl` (base-learner) with a `fit` function. The call to `fit` finally returns an object of class `bm` (base-model).

References

Scheipl, F., Staicu, A.-M., and Greven, S. (2014), Functional Additive Mixed Models, Journal of Computational and Graphical Statistics, in press, DOI 10.1080/10618600.2014.901914. http://arxiv.org/abs/1207.5947

See Also

`FDboost` for the model fit.

Examples

```
### example for scalar response and two functional covariates
data(fuel)
modFuel <- FDboost(heatan ~ bsignal(UVVIS, uvvis.lambda, knots=40, df=4)
           + bsignal(NIR, nir.lambda, knots=40, df=4),
           timeformula=~bols(1), data=fuel)
```

```
summary(modFuel)
## plot(modFuel, rug=FALSE)
```

`coef.FDboost` *Coefficients of boosted functional regression model*

Description

Takes a fitted FDboost-object produced by `FDboost()` and returns estimated coefficient functions/surfaces $\beta(t), \beta(s,t)$ and estimated smooth effects $f(z), f(x,z)$ or $f(x,z,t)$. Not implemented for smooths in more than 3 dimensions.

Usage

```
 ## S3 method for class 'FDboost'
coef(object, raw = FALSE,
   which = NULL, computeCoef = TRUE, n1 = 40, n2 = 40,
   n3 = 20, n4 = 10, ...)
```

Arguments

`object`	a fitted FDboost-object
`raw`	logical defaults to FALSE. If raw=FALSE for each effect the estimated function/surface is calculated If raw=TRUE the coefficients of the model are returned.
`which`	a subset of base-learners for which the coefficients should be computed (numeric vector), defaults to NULL which is the same as `which=1:length(object$baselearner)` In the special case of `which=0`, only the coefficients of the offset are returned.
`computeCoef`	defaults to TRUE, if FALSE only the names of the terms are returned
`n1`	see below
`n2`	see below
`n3`	n1, n2, n3 give the number of grid-points for 1-/2-/3-dimensional smooth terms used in the marginal equidistant grids over the range of the covariates at which the estimated effects are evaluated.
`n4`	gives the number of points for the third dimension in a 3-dimensional smooth term
`...`	other arguments, not used.

Value

If `raw==FALSE`, a list containing

- `pterms` a matrix containing the parametric / non-functional coefficients
- `smterms` a named list with one entry for each smooth term in the model. Each entry contains

- `x, y, z` the unique grid-points used to evaluate the smooth/coefficient function/coefficient surface
- `xlim, ylim, zlim` the extent of the x/y/z-axes
- `xlab, ylab, zlab` the names of the covariates for the x/y/z-axes
- `value` a vector/matrix/list of matrices containing the coefficient values
- `dim` the dimensionality of the effect
- `main` the label of the smooth term (a short label)

`cvMa` *Cross-Validation and Bootstrapping over Curves*

Description

Cross-Validation and bootstrapping over curves to compute the empirical risk for hyper-parameter selection and to compute resampled coefficients and predictions for the models.

Usage

```
cvMa(ydim, weights = NULL,
  type = c("bootstrap", "kfold", "subsampling", "curves"),
  B = ifelse(type == "kfold", 10, 25), prob = 0.5,
  strata = NULL, id = NULL)

cvLong(id, weights = rep(1, l = length(id)),
  type = c("bootstrap", "kfold", "subsampling", "curves"),
  B = ifelse(type == "kfold", 10, 25), prob = 0.5,
  strata = NULL)

validateFDboost(object, response = NULL,
  folds = cv(rep(1, object$ydim[1]), type = "bootstrap"),
  grid = 1:mstop(object), getCoefCV = TRUE,
  riskopt = c("mean", "median"), mrdDelete = 0,
  refitSmoothOffset = TRUE, ...)
```

Arguments

`object`	fitted FDboost-object
`response`	you can specify a response vector to calculate predictions errors. Defaults to NULL which means that the response of the fitted model is used.
`folds`	a weight matrix with number of rows equal to the number of observed trajectories.
`grid`	the grid over which the optimal mstop is searched
`getCoefCV`	logical, defaults to TRUE Should the coefficients and predictions be computed for all the models on the sampled data?

riskopt	how is the optimal stopping iteration determined. Defaults to the mean but median is possible as well.
mrdDelete	Delete values that are mrdDelete percent smaller then the mean of the response. Defaults to 0 which means that only response values being 0 are not used in the calculation of the MRD (= mean relative deviation)
refitSmoothOffset	logical, should the offset be refitted in each learning sample? Defaults to TRUE. In cvrisk the offset of the original model object is used.
ydim	dimensions of response-matrix
weights	a numeric vector of weights for the model to be cross-validated. if weights=NULL all weights are taken to be 1.
type	character argument for specifying the cross-validation method. Currently (stratified) bootstrap, k-fold cross-validation and subsampling are implemented. The argument curves implies that a cross-validation leaving out one curve at a time is performed.
B	number of folds, per default 25 for bootstrap and subsampling and 10 for kfold.
prob	percentage of observations to be included in the learning samples for subsampling.
strata	a factor of the same length as weights for stratification.
id	id-variable to sample upon ids instead of sampling single observations. (Only interesting in the case that several observations per individual were made.)
...	further arguments passed to mclapply

Details

The function validateFDboost calculates honest estimates of prediction errors as the curves/observations that should be predicted are not part of the model fit. The functions cvMa and cvLong can be used to build an appropriate weight matrix to be used with cvrisk. The function cvMa is appropriate for responses observed over a regular grid. The function cvLong for responsese that are observed over a irregular grid. Note, that the function validateFDboost expects folds that give weights per trajectory and that cvrisk expectes folds that give weights for single observations. In cvMa and cvLong trajectories are sampled. The probability for each trajectory to enter a fold is equal over all trajectories. If strata is defined sampling is performed in each stratum separately thus preserving the distribution of the strata variable in each fold. If id is defined sampling is performed on the level of id thus sampling individuals.

Value

cvMa and cvLong return a matrix of weights to be used in cvrisk. validateFDboost returns an validateFDboost-object, which is a named list containing:

response	the response
yind	the observation points of the response
id	the id variable of the response
folds	folds that were used

grid	grid of possible numbers of boosting iterations
coefCV	if getCoefCV is TRUE the estimated coefficient functions in the folds
predCV	if getCoefCV is TRUE the out-of-bag predicted values of the response
oobpreds	if the type of folds is curves the out-of-bag predictions for each trajectory
oobrisk	the out-of-bag risk
oobriskMean	the out-of-bag risk at the minimal mean risk
oobmse	the out-of-bag mean squared error (MSE)
oobrelMSE	the out-of-bag relative mean squared error (relMSE)
oobmrd	the out-of-bag mean relative deviation (MRD)
oobrisk0	the out-of-bag risk without consideration of integration weights
oobmse0	the out-of-bag mean squared error (MSE) without consideration of integration weights

Note

Use argument `mc.cores = 1L` to set the numbers of cores that is used in parallel computation. On Windows only 1 core is possible, `mc.cores = 1`, which is the default.

See Also

`cvrisk` to perform cross-validation.

Examples

```
Ytest <- matrix(rnorm(15), ncol=3) # 5 trajectories, each with 3 observations
cvMa(ydim=c(5,3), type="bootstrap", B=4)

data("viscosity", package = "FDboost")
## set time-interval that should be modeled
interval <- "101"

## model time until "interval" and take log() of viscosity
end <- which(viscosity$timeAll==as.numeric(interval))
viscosity$vis <- log(viscosity$visAll[,1:end])
viscosity$time <- viscosity$timeAll[1:end]
# with(viscosity, funplot(time, vis, pch=16, cex=0.2))

## fit median regression model with 200 boosting iterations,
## step-length 0.2 and
## smooth time-specific offset
mod <- FDboost(vis ~ 1 + bols(T_C) + bols(T_A),
               timeformula=~bbs(time, lambda=100),
               numInt="Riemann", family=QuantReg(),
               offset=NULL, offset_control = o_control(k_min = 9),
               data=viscosity, control=boost_control(mstop = 100, nu = 0.4))

## Not run:
## for the example B is set to a small value so that bootstrap is faster
```

```
val1 <- validateFDboost(mod, folds=cv(rep(1, 64), B=3) )
# plot(val1)
mstop(val1)

## End(Not run)
```

`extract.blg` *Extract information of a base-learner*

Description

Takes a base-learner and extracts information.

Usage

```
 ## S3 method for class 'blg'
extract(object,
   what = c("design", "penalty", "index"),
   asmatrix = FALSE, expand = FALSE, ...)
```

Arguments

`object`	a base-learner
`what`	a character specifying the quantities to extract. This can be a subset of "design" (default; design matrix), "penalty" (penalty matrix) and "index" (index of ties used to expand the design matrix)
`asmatrix`	a logical indicating whether the the returned matrix should be coerced to a matrix (default) or if the returned object stays as it is (i.e., potentially a sparse matrix). This option is only applicable if extract returns matrices, i.e., what = "design" or what = "penalty".
`expand`	a logical indicating whether the design matrix should be expanded (default: FALSE). This is useful if ties where taken into account either manually (via argument index in a base-learner) or automatically for data sets with many observations. expand = TRUE is equivalent to extract(B)[extract(B, what = "index"),] for a base-learner B.
`...`	currently not used

See Also

`extract` for the extract functions of the package mboost

FDboost *Model-based Gradient Boosting for Functional Response*

Description

Gradient boosting for optimizing arbitrary loss functions, where component-wise models are utilized as base-learners in the case of functional response. This function is a wrapper for mboost's mboost and its siblings to fit models of the general form
$E(Y_i(t)) = g(\mu(t) + \int X_i(s)\beta(s,t)ds + f(z_{1i},t) + f(z_{2i}) + z_{3i}\beta_3(t) + \ldots$
with a functional (but not necessarily continuous) response $Y(t)$, response function g, (optional) smooth intercept $\mu(t)$, (multiple) functional covariates $X(t)$ and scalar covariates z_1, z_2, etc.

Usage

```
FDboost(formula, timeformula, id = NULL,
  numInt = "equal", data, weights = NULL, offset = NULL,
  offset_control = o_control(), ...)
```

Arguments

formula	a symbolic description of the model to be fit.
timeformula	formula for the expansion over the index of the response. For a functional response $Y_i(t)$ typically ~bbs(t) to obtain a smooth expansion of the effects along t. In the limiting case that Y_i is a scalar response use ~bols(1), which sets up a base-learner for the scalar 1.
id	defaults to NULL which means that the response is a matrix with a regular time. If the response is given in long format for irregular observations, id contains the information which observations belong together. id should contain numbers 1, 2, 3, ...
numInt	integration scheme for the integration of the loss function. One of c("equal", "Riemann") meaning equal weights of 1 or trapezoidal Riemann weights. Alternatively a vector of length nrow(response) containing arbitrary positive weights can be specified.
data	a data frame or list containing the variables in the model.
weights	(1) a numeric vector of weights for observational units, i.e. length(weights) has to be nrow(response), (2) alternatively weights can be specified for single observations then length(weights) has to be nrow(response)*ncol(response) per default weights is constantly 1.
offset_control	parameters for the calculation of the offset, defaults to offset_control(k_min=20, silent=TRUE).
offset	a numeric vector to be used as offset over the index of the response (optional). If no offset is specified, per default a smooth time-specific offset is calculated and used within the model fit. If you do not want to use an offset you can set offset=0.
...	additional arguments passed to mboost, including offset, family and control.

Details

The functional response and functional covariates have to be supplied as n by <no. of evaluations> matrices, i.e. each row is one functional observation. For the model fit the matrix of the functional response evaluations $Y_i(t)$ are stacked into one long vector. If the functional response is supplied as a vector in long format, the argument `id` has to be specified.

Value

on object of class `FDboost` that inherits from `mboost`. Special `predict.FDboost`, `coef.FDboost` and `plot.FDboost` methods are available. The methods of `mboost` are available as well, e.g. `extract`.

The `FDboost`-object is a named list containing:

`...`	all elements of an `mboost`-object
`yname`	the name of the response
`yind`	the observation points of the response, with its name as attribute
`data`	the data that was used for the model fit
`id`	NULL for a response over a regular grid, otherwise the id variable of the response
`predictOffset`	the function to predict the smooth offset
`offsetVec`	the offset for one trajectory for regular response and otherwise the offset for all trajectories
`callEval`	the evaluated function call
`timeformula`	the time-formula
`formulaFDboost`	the formula with which `FDboost` was called
`formulaMboost`	the formula with which `mboost` was called within `FDboost`

Author(s)

Sarah Brockhaus, Torsten Hothorn

See Also

`mboost` for the help of the wrapped function in package mboost. See `bsignal` and `bbsc` for possible base-learners

Examples

```
data("viscosity", package = "FDboost")
## set time-interval that should be modeled
interval <- "101"

## model time until "interval" and take log() of viscosity
end <- which(viscosity$timeAll==as.numeric(interval))
viscosity$vis <- log(viscosity$visAll[,1:end])
viscosity$time <- viscosity$timeAll[1:end]
```

```
# with(viscosity, funplot(time, vis, pch=16, cex=0.2))

## fit median regression model with 200 boosting iterations,
## step-length 0.2 and smooth time-specific offset
mod <- FDboost(vis ~ 1 + bols(T_C) + bols(T_A),
               timeformula=~bbs(time, lambda=100),
               numInt="Riemann", family=QuantReg(),
               offset=NULL, offset_control = o_control(k_min = 9),
               data=viscosity, control=boost_control(mstop = 100, nu = 0.4))
summary(mod)
```

`fitted.FDboost` *Fitted values of a boosted functional regression model*

Description

Takes a fitted `FDboost`-object and computes the fitted values.

Usage

```
 ## S3 method for class 'FDboost'
fitted(object, ...)
```

Arguments

`object`	a fitted `FDboost`-object
`...`	additional arguments passed on to `predict.FDboost`

Value

matrix of fitted values

See Also

`FDboost` for the model fit.

`fuel` *Spectral data of fossil fuels*

Description

For 129 laboratory samples of fossil fuels the heat value and the humidity were determined together with two spectra. One spectrum is ultraviolet-visible (UV-VIS), measured at 1335 wavelengths in the range of 250.4 to 878.4 nanometer (nm), the other a near infrared spectrum (NIR) measured at 2307 wavelengths in the range of 800.4 to 2779.0 nm.

Usage

```
data("fuel")
```

Format

A data list with 129 observations on the following 7 variables.

`heatan` heat value in mega joule (mJ)

`h2o` humidity in percent

`NIR` near infrared spectrum (NIR)

`UVVIS` ultraviolet-visible spectrum (UV-VIS)

`nir.lambda` wavelength of NIR spectrum in nm

`uvvis.lambda` wavelength of UV-VIS spectrum in nm

`h2o.fit` predicted values of humidity

Details

The aim is to predict the heat value using the spectral data. The variable `h2o.fit` was generated by a functional linear regression model, using both spectra and their derivatives as predictors.

Source

Siemens AG

Fuchs, K., Scheipl, F. & Greven, S. (2013), Penalized scalar-on-functions regression with interaction term. Submitted for publication.

Examples

```
data("fuel", package = "FDboost")

### fit mean regression model with 100 boosting iterations,
### step-length 0.1 and
mod <- FDboost(heatan ~ bsignal(UVVIS, uvvis.lambda, knots=40, df=4)
               + bsignal(NIR, nir.lambda, knots=40, df=4),
               timeformula=~bols(1), data=fuel)
```

---

funMRD *Functional MRD*

Description

Calculates the functional MRD for a fitted FDboost-object

Usage

```
funMRD(object, overTime = TRUE, breaks = object$yind,
  global = FALSE, ...)
```

Arguments

object	fitted FDboost-object with regular response
overTime	per default the functional MRD is calculated over time if overTime=FALSE, the MRD is calculated per curve
breaks	an optional vector or number giving the time-points at which the model is evaluated. Can be specified as number of equidistant time-points or as vector of time-points. Defaults to the index of the response in the model.
global	logical. defaults to FALSE, if TRUE the global MRD like in a normal linear model is calculated
...	currently not used

Details

Formula to calculate MRD over time, overTime=TRUE:
$MRD(t) = n^{-1} \sum_i |(Y_i(t) - \hat{Y}_i(t))^2| / |Y_i(t)|$

Formula to calculate MRD over subjects, overTime=FALSE:
$MRD_i = \int |(Y_i(t) - \hat{Y}_i(t))^2| / |Y_i(t)| dt \approx G^{-1} \sum_g |(Y_i(t_g) - \hat{Y}_i(t_g))^2| / |Y_i(t)|$

Value

Returns a vector with the calculated MRD and some extra information in attributes.

Note

breaks cannot be changed in the case the bsignal() is used over the same domain as the response! In that case you would have to rename the index of the response or that of the covariates.

---

funMSE *Functional MSE*

Description

Calculates the functional MSE for a fitted FDboost-object

Usage

```
funMSE(object, overTime = TRUE, breaks = object$yind,
  global = FALSE, relative = FALSE, root = FALSE, ...)
```

Arguments

object	fitted FDboost-object
overTime	per default the functional R-squared is calculated over time if overTime=FALSE, the R-squared is calculated per curve
breaks	an optional vector or number giving the time-points at which the model is evaluated. Can be specified as number of equidistant time-points or as vector of time-points. Defaults to the index of the response in the model.
global	logical. defaults to FALSE, if TRUE the global R-squared like in a normal linear model is calculated
relative	logical. defaults to FALSE. If TRUE the MSE is standardized by the global variance of the response $n^{-1} \int \sum_i (Y_i(t) - \bar{Y})^2 dt \approx G^{-1} n^{-1} \sum_g \sum_i (Y_i(t_g) - \bar{Y})^2$
root	take the square root of the MSE
...	currently not used

Details

Formula to calculate MSE over time, overTime=TRUE:
$MSE(t) = n^{-1} \sum_i (Y_i(t) - \hat{Y}_i(t))^2$

Formula to calculate MSE over subjects, overTime=FALSE:
$MSE_i = \int (Y_i(t) - \hat{Y}_i(t))^2 dt \approx G^{-1} \sum_g (Y_i(t_g) - \hat{Y}_i(t_g))^2$

Value

Returns a vector with the calculated MSE and some extra information in attributes.

Note

breaks cannot be changed in the case the bsignal() is used over the same domain as the response! In that case you would have to rename the index of the response or that of the covariates.

---

funplot *Plot functional data with linear interpolation of missing values*

Description

Plot functional data with linear interpolation of missing values

Usage

```
funplot(x, y, id = NULL, rug = TRUE, ...)
```

Arguments

x	optional, time-vector for plotting
y	matrix of functional data with functions in rows and measured times in columns
id	defaults to NULL for y matrix, is id-variables for y in long format
rug	logical. Should rugs be plotted? Defaults to TRUE.
...	further arguments passed to matplot.

Details

All observations are marked by a small cross (pch=3). Missing values are imputed by linear interpolation. Parts that are interpolated are plotted by dotted lines, parts with non-missing values as solid lines.

Examples

```
## Not run:
### examples for regular data in wide format
data(viscosity)
with(viscosity, funplot(timeAll, visAll, pch=20))
if(require(fda)){
  with(fda::growth, funplot(age, t(hgtm)))
}

## End(Not run)
```

---

funRsquared *Functional R-squared*

Description

Calculates the functional R-squared for a fitted FDboost-object

Usage

```
funRsquared(object, overTime = TRUE,
  breaks = object$yind, global = FALSE, ...)
```

Arguments

object	fitted FDboost-object
overTime	per default the functional R-squared is calculated over time if overTime=FALSE, the R-squared is calculated per curve
breaks	an optional vector or number giving the time-points at which the model is evaluated. Can be specified as number of equidistant time-points or as vector of time-points. Defaults to the index of the response in the model.
global	logical. defaults to FALSE, if TRUE the global R-squared like in a normal linear model is calculated
...	currently not used

Details

breaks should be set to some grid, if there are many missing values or time-points with very few observations in the dataset. Otherwise at these points of t the variance will be almost 0 (or even 0 if there is only one observation at a time-point), and then the prediction by the local means $\mu(t)$ is locally very good. The observations are interpolated linearly if necessary.

Formula to calculate R-squared over time, overTime=TRUE:
$R^2(t) = 1 - \sum_i (Y_i(t) - \hat{Y}_i(t))^2 / \sum_i (Y_i(t) - \bar{Y}(t))^2$

Formula to calculate R-squared over subjects, overTime=FALSE:
$R_i^2 = 1 - \int (Y_i(t) - \hat{Y}_i(t))^2 dt / \int (Y_i(t) - \bar{Y}_i)^2 dt$

Value

Returns a vector with the calculated R-squared and some extra information in attributes.

Note

breaks cannot be changed in the case the bsignal() is used over the same domain as the response! In that case you would have to rename the index of the response or that of the covariates.

References

Ramsay, J., Silverman, B. (2006). Functional data analysis. Wiley Online Library. chapter 16.3

---

integrationWeights *Functions to compute integration weights*

Description

Computes trapezoidal integration weights for a functional variable X1 on grid xind.

Usage

```
integrationWeights(X1, xind, id = NULL)
```

Arguments

X1	matrix of functional variable
xind	index of functional variable
id	defaults to NULL if X1 is a matrix. identity variable if X1 is in long format.

Details

The function integrationWeights() computes trapezoidal integration weights, that are symmetric. The function integrationWeightsLeft() computes weights, that take into account only the distance to the prior observation point. Those weights are adequate for historical effects.

`mstop.validateFDboost` *Methods for objects of class validateFDboost*

Description

Methods for objects that are fitted to determine the optimal mstop and the prediction error of a model fitted by FDboost.

Usage

```
 ## S3 method for class 'validateFDboost'
mstop(object,
   riskopt = c("mean", "median"), ...)

 ## S3 method for class 'validateFDboost'
plot(x,
   riskopt = c("mean", "median"), which = 1,
   modObject = NULL, predictNA = FALSE, names.arg = NULL,
   ask = TRUE, ...)

 plotPredCoef(x, which = NULL, pers = TRUE,
   commonRange = TRUE, showNumbers = FALSE,
   showQuantiles = TRUE, ask = TRUE, terms = TRUE,
   probs = c(0.05, 0.5, 0.95), ylim = NULL, ...)
```

Arguments

`x`	object of class validateFDboost
`object`	object of class validateFDboost
`riskopt`	how the risk is minimized to obtain the optimal stopping iteration; defaults to the mean, can be changed to the median.
`modObject`	if the original model object of class `FDboost` is given predicted values of the whole model can be compared to the predictions of the cross-validated models
`predictNA`	should missing values in the response be predicted? Defaults to FALSE.
`names.arg`	names of the observed curves
`ask`	par(ask=ask)
`which`	a subset of base-learners to take into account for plotting. In the case of `plot.validateFDboost` the diagnostic plots that are given.
`pers`	plot coefficient surfaces as persp-plots? Defaults to TRUE.
`commonRange,`	plot predicted coefficients on a common range, defaults to TRUE
`showQuantiles`	plot the 0.05 and the 0.95 Quantile of coefficients in 1-dim effects
`showNumbers`	show number of curve in plot of predicted coefficients, defaults to FALSE
`terms,`	logical, defaults to TRUE plot the added terms (default) or the coefficients?

probs	vector of quantiles to be used in the plotting of 2-dimensional coefficients surfaces, defaults to probs=c(0.25, 0.5, 0.75)
ylim	values for limits of y-axis
...	additional arguments passed to callies.

Details

plot.validateFDboost plots cross-validated risk, RMSE, MRD, measured and predicted values and residuals as determined by validateFDboost. mstop.validateFDboost extracts the optimal mstop by minimizing the median risk.

---

o_control *Function to control estimation of smooth offset*

Description

Function to control estimation of smooth offset

Usage

```
o_control(k_min = 20, rule = 2, silent = TRUE,
  cyclic = FALSE, knots = NULL)
```

Arguments

k_min	maximal number of k in s()
rule	which rule to use in approx() of the response before calculating the global mean, rule=1 means no extrapolation, rule=2 means to extrapolate the closest non-missing value, see approx
silent	print error messages of model fit?
cyclic	defaults to FALSE, if TRUE cyclic splines are used
knots	arguments knots passed to gam

---

plot.FDboost *Plot the fit or the coefficients of a boosted functional regression model*

Description

Takes a fitted FDboost-object produced by FDboost() and plots the fitted effects or the coefficient-functions/surfaces.

Usage

```
 ## S3 method for class 'FDboost'
plot(x, raw = FALSE, rug = TRUE,
   which = NULL, includeOffset = TRUE, ask = TRUE,
   n1 = 40, n2 = 40, n3 = 20, n4 = 11,
   onlySelected = TRUE, pers = FALSE, commonRange = FALSE,
   ...)

 plotPredicted(x, subset = NULL, posLegend = "topleft",
   lwdObs = 1, lwdPred = 1, ...)

 plotResiduals(x, subset = NULL, posLegend = "topleft",
   ...)
```

Arguments

x	a fitted FDboost-object
raw	logical defaults to FALSE. If raw=FALSE for each effect the estimated function/surface is calculated If raw=TRUE the coefficients of the model are returned.
rug	when TRUE (default) then the covariate to which the plot applies is displayed as a rug plot at the foot of each plot of a 1-d smooth, and the locations of the covariates are plotted as points on the contour plot representing a 2-d smooth.
which	a subset of base-learners to take into account for plotting. a list is returned.
includeOffset	logical, defaults to TRUE. Should the offset be included in the plot of the intercept (default) or should it be plotted separately.
ask	logical, defaults to TRUE, if several effects are plotted the user has to hit Return to see next plot.
n1	see below
n2	see below
n3	n1, n2, n3 give the number of grid-points for 1-/2-/3-dimensional smooth terms used in the marginal equidistant grids over the range of the covariates at which the estimated effects are evaluated.
n4	gives the number of points for the third dimension in a 3-dimensional smooth term
onlySelected,	logical, defaults to TRUE. Only plot effects that were selected in at least one boosting iteration
pers	logical, defaults to FALSE, If TRUE, perspective plots (persp) for 2- and 3-dimensional effects are drawn. If FALSE, image/contour-plots (image, contour) are drawn for 2- and 3-dimensional effects.
commonRange	logical, defaults to FALSE, if TRUE the range over all effects is the same (so far not implemented)
subset	subset of the observed response curves and their predictions that is plotted. Per default all observations are plotted.

posLegend	location of the legend, if a legend is drawn automatically (only used in plotPredicted). The default is "topleft".
lwdObs	lwd of observed curves (only used in plotPredicted)
lwdPred	lwd of predicted curves (only used in plotPredicted)
...	other arguments, passed to funplot (only used in plotPredicted)

See Also

FDboost for the model fit and coef.FDboost for the calculation of the coefficient functions.

---

predict.FDboost *Prediction for boosted functional regression model*

Description

Takes a fitted FDboost-object produced by FDboost() and produces predictions given a new set of values for the model covariates or the original values used for the model fit. This is a wrapper function for predict.mboost()

Usage

```
## S3 method for class 'FDboost'
predict(object, newdata = NULL,
   which = NULL, unlist = TRUE, ...)
```

Arguments

object	a fitted FDboost-object
newdata	A named list or a data frame containing the values of the model covariates at which predictions are required. If this is not provided then predictions corresponding to the original data are returned. If newdata is provided then it should contain all the variables needed for prediction, in the format supplied to FDboost, i.e., functional predictors must be supplied as matrices with each row corresponding to one observed function.
which	a subset of base-learners to take into account for computing predictions or coefficients. If which is given (as an integer vector corresponding to base-learners) a list is returned.
unlist	logical, defaults to TRUE. Should predictions be returned in matrix form (default) or as a list
...	additional arguments passed on to predict.mboost().

Value

a matrix or list of predictions depending on values of unlist and which

See Also

FDboost for the model fit and plotPredicted for a plot of the observed values and their predictions.

`print.FDboost` *Print a boosted functional regression model*

Description

Takes a fitted FDboost-object and produces a print on the console.

Usage

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

Arguments

x	a fitted FDboost-object
...	currently not used

Value

a list with information on the model

See Also

FDboost for the model fit.

`residuals.FDboost` *Residual values of a boosted functional regression model*

Description

Takes a fitted FDboost-object and computes the residuals.

Usage

```
## S3 method for class 'FDboost'
residuals(object, ...)
```

Arguments

object	a fitted FDboost-object
...	not used

Details

The residual is missing if the corresponding value of the response was missing.

Value

matrix of residual values

See Also

FDboost for the model fit.

`summary.FDboost` *Summary of a boosted functional regression model*

Description

Takes a fitted FDboost-object and produces a summary.

Usage

```
 ## S3 method for class 'FDboost'
summary(object, ...)
```

Arguments

object	a fitted FDboost-object
...	currently not used

Value

a list with summary information

See Also

FDboost for the model fit.

truncateTime *Function to truncate time in functional data*

Description

Function to truncate time in functional data

Usage

```
truncateTime(funVar, time, newtime, data)
```

Arguments

funVar	names of functional variables that should be truncated
time	name of time variable
newtime	new time vector that should be used. Must be part of the old time-line.
data	list containing all the data

Value

A list with the data containing all variables of the original dataset with the variables of funVar truncated according to newtime.

Note

All variables that are not part if funVar, or time are simply copied into the new data list

Examples

```
if(require(fda)){
  dat <- fda::growth
  dat$hgtm <- t(dat$hgtm[,1:10])
  dat$hgtf <- t(dat$hgtf[,1:10])

  ## only use time-points 1:16 of variable age
  datTr <- truncateTime(funVar=c("hgtm","hgtf"), time="age", newtime=1:16, data=dat)

  ## Not run:
  par(mfrow=c(1,2))
  with(dat, funplot(age, hgtm, main="Original data"))
  with(datTr, funplot(age, hgtm, main="Yearly data"))
  par(mfrow=c(1,1))

## End(Not run)
}
```

`viscosity` *Viscosity of resin over time*

Description

In an experimental setting the viscosity of resin was measured over time to asses the curing process depending on 5 binary factors (low-high).

Usage

```
data("viscosity")
```

Format

A data list with 64 observations on the following 7 variables.

`visAll` viscosity measures over all available time points

`timeAll` time points of viscosity measures

`T_C` temperature of tools

`T_A` temperature of resin

`T_B` temperature of curing agent

`rspeed` rotational speed

`mflow` mass flow

Details

The aim is to determine factors that affect the curing process in the mold. The desired viscosity-curve has low values in the beginning followed by a sharp increase. Due to technical reasons the measuring method of the rheometer has to be changed in a certain range of viscosity. The first observations are measured by rotation of a blade giving observations every two seconds, the later observations are measured through oscillation of a blade giving observations every ten seconds. In the later observations the resin is quite hard so the measurements should be interpreted as a qualitative measure of hardening.

Source

Wolfgang Raffelt, Technical University of Munich, Institute for Carbon Composites

Examples

```
data("viscosity", package = "FDboost")

## set time-interval that should be modeled
interval <- "509"

## model time until "interval" and take log() of viscosity
```

```
end <- which(viscosity$timeAll==as.numeric(interval))
viscosity$vis <- log(viscosity$visAll[,1:end])
viscosity$time <- viscosity$timeAll[1:end]

### fit median regression model with 200 boosting iterations,
### step-length 0.2 and
### smooth time-specific offset
mod <- FDboost(vis ~ 1 + bols(T_C) + bols(T_A),
            timeformula=~bbs(time, lambda=100),
            numInt="Riemann", family=QuantReg(),
            offset=NULL, offset_control = o_control(k_min = 9),
            data=viscosity, control=boost_control(mstop = 200, nu = 0.2))
```

Index