Package 'fsemipar'

January 23, 2023

Type Package

Title Estimation, Variable Selection and Prediction for Functional Semiparametric Models

Version 1.0

Date 2023-01-7

Author German Aneiros [aut],
Silvia Novo [aut, cre]

Depends R (>= 3.5.0), splines, gtools, grpreg, DiceKriging, graphics, stats

Maintainer Silvia Novo <snovo@est-econ.uc3m.es>

Description Routines for estimation or simultaneous estimation and variable selection of several functional semiparametric models with scalar response, such as the functional single-index model,
the semi-functional partial linear model or the semi-functional partial linear single-index model.
In addition, it includes algorithms for dealing with scalar covariates with linear effect coming from the discretization
of a curve in the cases of the linear model, the multi-functional partial linear model
and the multi-functional partial linear single-index model.

License GPL (>= 2)

NeedsCompilation no

Repository CRAN

Date/Publication 2023-01-23 09:50:06 UTC

R topics documented:

fsemipar-package 2
FASSMR.kernel.fit 4
FASSMR.kNN.fit 9
fsemipar.internal 14
fsim.kernel.fit 15
fsim.kernel.test 18
fsim.kNN.fit 21

fsim.kNN.test . . . 24
IASSMR.kernel.fit . . . 27
IASSMR.kNN.fit . . . 33
lm.pels.fit . . . 39
plot.classes . . . 41
predict.fsim . . . 44
predict.IASSMR . . . 45
predict.lm . . . 48
predict.mfplm . . . 50
predict.sfpl . . . 52
predict.sfplsim.FASSMR . . . 54
print.summary.fsim . . . 57
print.summary.lm . . . 58
print.summary.mfpl . . . 59
print.summary.mfplsim . . . 60
print.summary.sfpl . . . 62
print.summary.sfplsim . . . 63
projec . . . 64
PVS.fit . . . 66
PVS.kernel.fit . . . 70
PVS.kNN.fit . . . 76
semimetric.projec . . . 81
sfpl.kernel.fit . . . 83
sfpl.kNN.fit . . . 87
sfplsim.kernel.fit . . . 90
sfplsim.kNN.fit . . . 95
Sugar . . . 100
Tecator . . . 101

Index **103**

`fsemipar-package` *Estimation, Variable Selection and Prediction for Functional Semiparametric Models*

Description

This package is devoted to estimation or simultaneous estimation and variable selection of several functional semiparametric models with scalar response, such us the functional single-index model, the semi-functional partial linear model or the semi-functional partial linear single-index model. It also includes algortithms for dealing with scalar covariates with linear effect coming from the discretization of a curve in the cases of the linear model, the multi-functional partial linear model and the multi-functional partial linear single-index model.

In many cases, the estimation procedures involve nonparametric regression techniques. Then, routines for both kernel-based and kNN-based estimation with Nadaraya-Watson weights are provided for all of them. In addition, the package contains routines to compute predictions from all the models and estimation procedures.

Details

The package could be divided in several tematic sections:

1. Estimation and prediction of the functional single-index model.
   - `projec`.
   - `semimetric.projec`.
   - `fsim.kernel.fit` and `fsim.kNN.fit`.
   - `fsim.kernel.test` and `fsim.kNN.test`.
   - `predict`, `summary` and `print` methods for `fsim.kernel` and `fsim.kNN` classes.
2. Variable selection, estimation and prediction of the semi-functional partial linear single-index model.
   - `sfplsim.kernel.fit` and `sfplsim.kNN.fit`.
   - `predict`, `summary` and `print` methods for `sfplsim.kernel` and `sfplsim.kNN` classes.
3. Variable selection, estimation and prediction of the semi-functional partial linear model.
   - `sfpl.kernel.fit` and `sfpl.kNN.fit`.
   - `predict`, `summary` and `print` methods for `sfpl.kernel` and `sfpl.kNN` classes.
4. Variable selection, estimation and prediction of the linear model.
   - `lm.pels.fit`.
   - `predict`, `summary` and `print` methods for `lm.pels` class.
5. Variable selection, estimation and prediction of the linear model with covariates coming from the discretisation of a curve.
   - `PVS.fit`.
   - `predict`, `summary` and `print` methods for `PVS` class.
6. Variable selection, estimation and prediction of the multi-functional partial linear model.
   - `PVS.kernel.fit` and `PVS.kNN.fit`.
   - `predict`, `summary` and `print` methods for `PVS.kernel` and `PVS.kNN` classes.
7. Variable selection, estimation and prediction of the multi-functional partial linear single-index model.
   - `FASSMR.kernel.fit` and `FASSMR.kNN.fit`.
   - `IASSMR.kernel.fit` and `IASSMR.kNN.fit`.
   - `predict`, `summary` and `print` methods for `FASSMR.kernel`, `FASSMR.kNN`, `IASSMR.kernel` and `IASSMR.kNN` classes.
8. Two datasets: `Tecator` and `Sugar`.

Author(s)

German Aneiros [aut], Silvia Novo [aut, cre]

Maintainer: Silvia Novo <snovo@est-econ.uc3m.es>

References

Aneiros, G. and Vieu, P., (2014) Variable selection in infinite-dimensional problems, *Statistics and Probability Letters*, **94**, 12–20. doi:10.1016/j.spl.2014.06.025.

Aneiros, G., Ferraty, F., and Vieu, P., (2015) Variable selection in partial linear regression with functional covariate, *Statistics*, **49** 1322–1347, doi:10.1080/02331888.2014.998675.

Aneiros, G., and Vieu, P., (2015) Partial linear modelling with multi-functional covariates. *Computational Statistics*, **30**, 647–671. doi:10.1007/s0018001505688.

Novo S., Aneiros, G., and Vieu, P., (2019) Automatic and location-adaptive estimation in functional single-index regression, *Journal of Nonparametric Statistics*, **31(2)**, 364–392, doi:10.1080/10485252.2019.1567726.

Novo, S., Aneiros, G., and Vieu, P., (2021) Sparse semiparametric regression when predictors are mixture of functional and high-dimensional variables, *TEST*, **30**, 481–504, doi:10.1007/s11749020-00728w.

Novo, S., Aneiros, G., and Vieu, P., (2021) A kNN procedure in semiparametric functional data analysis, *Statistics and Probability Letters*, **171**, 109028, doi:10.1016/j.spl.2020.109028.

Novo, S., Vieu, P., and Aneiros, G., (2021) Fast and efficient algorithms for sparse semiparametric bi-functional regression, *Australian and New Zealand Journal of Statistics*, **63**, 606–638, doi:10.1111/anzs.12355.

`FASSMR.kernel.fit` *FASSMR with kernel estimation*

Description

This function computes the fast algorithm for sparse semiparametric multi-functional regression (FASSMR) with kernel estimation.

This algorithm involves the penalised least-squares regularization procedure combined with kernel estimation with Nadaraya-Watson weights. The procedure requires the B-spline representation to estimate the functional index θ_0 and an objective criterion (`criterion`) to select the initial number of covariates in the reduced model (`w.opt`), the bandwidth (`h.opt`) and the penalisation parameter (`lambda.opt`).

Usage

```
FASSMR.kernel.fit(x, z, y, seed.coeff = c(-1, 0, 1), order.Bspline = 3,
  nknot.theta = 3, t0 = NULL, min.q.h = 0.05,max.q.h = 0.5,
  h.seq = NULL, num.h = 10, range.grid = NULL, kind.of.kernel = "quad",
  nknot = NULL, lambda.min = NULL,lambda.min.h = NULL,
  lambda.min.l = NULL, factor.pn = 1, nlambda = 100, vn = ncol(z),
  nfolds = 10, seed = 123,wn = c(10, 15, 20), criterion = c("GCV", "BIC",
  "AIC", "k-fold-CV"), penalty = c("grLasso", "grMCP",
  "grSCAD", "gel", "cMCP", "gBridge", "gLasso", "gMCP"),
   max.iter = 1000)
```

Arguments

x	Matrix containing the observations of the functional covariate collected by row (functional single-index component).
z	Matrix containing the observations of the functional covariate that is discretised collected by row (linear component).
y	Vector containing the scalar response.
seed.coeff	Vector of initial values used to build the set Θ_n (see section Details). The coefficients for the B-spline representation of each eligible functional index $\theta \in \Theta_n$ are obtained from seed.coeff. The default is c(-1,0,1).
order.Bspline	Positive integer giving the order of the B-spline basis functions. This is the number of coefficients in each piecewise polynomial segment. The default is 3.
nknot.theta	Positive integer indicating the number of uniform interior knots of the B-spline basis for the B-spline representation of θ_0. The default is 3.
t0	Value in the domain of the functional indexes at which we evaluate them to build the set Θ_n. We assume $\theta_0(t_0) > 0$ for some arbitrary t_0 in the domain to ensure model identifiability. If t0=NULL, then mean(range.grid) is considered.
min.q.h	Order of the quantile of the set of distances between curves (computed with the projection semi-metric) which gives the lower end of the sequence in which the bandwidth is selected. The default is 0.05.
max.q.h	Order of the quantile of the set of distances between curves (computed with the projection semi-metric) which gives the upper end of the sequence in which the bandwidth is selected. The default is 0.5.
h.seq	Vector containing the sequence of bandwidths. The default is a sequence of num.h equispaced bandwidths in the range constructed using min.q.h and max.q.h.
num.h	Positive integer indicating the number of bandwiths in the grid. The default is 10.
range.grid	Vector of length 2 containing the endpoints of the grid at which the observations of the functional covariate x are evaluated (i.e. the range of the discretization). If range.grid=NULL, then range.grid=c(1,p) is considered, where p is the size of the discretization size of x (i.e. ncol(x)).
kind.of.kernel	The type of kernel function used. Only Epanechnikov kernel ("quad") is available.
nknot	Positive integer indicating the number of interior knots for the B-spline representation of the functional covariate. The default value is (p - order.Bspline - 1)%/%2.
lambda.min	The smallest value for lambda (i. e., the smallest value of the sequence in which lambda.opt is selected), as fraction of lambda.max. The defaults is lambda.min.l if the number of observations is larger than factor.pn times the number of covariates and lambda.min.h otherwise.
lambda.min.h	The smallest value of the sequence in which lambda.opt is selected if the number of observations is smaller than factor.pn times the number of scalar covariates. The default is 0.05.

`lambda.min.l`	The smallest value of the sequence in which `lambda.opt` is selected if the number of observations is larger than `factor.pn` times the number of scalar covariates. The default is 0.0001.
`factor.pn`	Positive integer used to set `lambda.min`. The default value is 1.
`nlambda`	Positive integer indicating the number of values of the sequence in which `lambda.opt` is selected. The default is 100.
`vn`	Positive integer or vector of positive integers indicating the number of groups of consecutive variables to be penalised together. The default value is `vn=ncol(z)`, which leads to the individual penalisation of each scalar covariate.
`nfolds`	Positive integer indicating the number of cross-validation folds (used if `criterion="k-fold-CV"`). Default is 10.
`seed`	You may set the seed of the random number generator to obtain reproducible results (used if `criterion="k-fold-CV"`). Default is 123.
`wn`	A vector of positive integers indicating the eligible number of covariates of the reduced model. See the section `Details`. The default is `c(10,15,20)`.
`criterion`	The criterion by which to select the regularization parameter `lambda.opt` and `k.opt`. One of `"GCV"`, `"BIC"`, `"AIC"` or `"k-fold-CV"`. The default is `"GCV"`.
`penalty`	The penalty function to be applied in the penalized least squares procedure. Only "grLasso" and "grSCAD" are implemented.
`max.iter`	Maximum number of iterations (total across entire path). Default is 1000.

Details

The multi-functional partial linear single-index model (MFPLSIM) is given by the expression

$$Y_i = \sum_{j=1}^{p_n} \beta_{0j}\zeta_i(t_j) + r\left(\langle\theta_0, X_i\rangle\right) + \varepsilon_i, \quad (i = 1, \ldots, n).$$

where

- Y_i is a real random response and X_i denotes a random element belonging to some separable Hilbert space $\mathcal{H}$ with inner product denoted by $\langle\cdot,\cdot\rangle$. The second functional predictor ζ_i is supposed to be a random curve defined on some interval $[a, b]$ which is observed at the points $a \leq t_1 < \cdots < t_{p_n} \leq b$.
- $\boldsymbol{\beta}_0 = (\beta_{01}, \ldots, \beta_{0p_n})^\top$ is a vector of unknown real coefficients and $r(\cdot)$ denotes a smooth unknown link function. In addition, θ_0 is an unknown functional direction in $\mathcal{H}$.
- ε_i denotes the random error.

In the MFPLSIM, we assume that only a few scalar variables from the set $\{\zeta(t_1), \ldots, \zeta(t_{p_n})\}$ form part of the model. Therefore, we must select the relevant variables in the linear component (the impact points of the curve ζ on the response) and estimate the model.

In this function, the MFPLSIM is fitted using the FASSMR algorithm. The main idea of this algorithm is to consider a reduced model, with only some (very few) linear covariates (but covering the entire discretization interval of ζ), and discarding directly the other linear covariates (since one expect that they contain very similar information about the response).

To explain the algorithm we assume, without lost of generality, that the number p_n of linear covariates can be expressed as follows: $p_n = q_n w_n$ with q_n and w_n integers. The previous consideration allows to build a subset of the initial p_n linear covariates, which contains only w_n equally spaced discretized observations of ζ covering the whole interval $[a, b]$. This subset is the following:

$$\mathcal{R}_n^{\mathbf{1}} = \left\{\zeta\left(t_k^{\mathbf{1}}\right),\ \ k = 1, \ldots, w_n\right\},$$

where $t_k^{\mathbf{1}} = t_{[(2k-1)q_n/2]}$ and $[z]$ denotes the smallest integer not less than $z \in \mathbb{R}$.

In this way, we consider the following reduced model, which involves only the linear covariates belonging to $\mathcal{R}_n^{\mathbf{1}}$:

$$Y_i = \sum_{k=1}^{w_n} \beta_{0k}^{\mathbf{1}} \zeta_i(t_k^{\mathbf{1}}) + r^{\mathbf{1}}\left(\left\langle \theta_0^{\mathbf{1}}, \mathcal{X}_i \right\rangle\right) + \varepsilon_i^{\mathbf{1}}.$$

The eligible numbers of linear covariates to build the reduced model are provided to the program in the argument `wn`. Then, the penalised least-squares variable selection procedure, with kernel estimation, is applied to the reduced model. This is done by means of the function `sfplsim.kernel.fit`, which requires remaining arguments (for details, see the documentation of the function `sfplsim.kernel.fit`). The estimates obtained after that are the outputs of the FASSMR algorithm. For further details on this algorithm, see Novo et al. (2021).

Remark: If the condition $p_n = w_n q_n$ fails, the function considers not fixed $q_n = q_{n,k}$ values $k = 1, \ldots, w_n$, when p_n/w_n is not an integer number. Specifically:

$$q_{n,k} = \begin{cases} [p_n/w_n] + 1 & k \in \{1, \ldots, p_n - w_n[p_n/w_n]\}, \\ [p_n/w_n] & k \in \{p_n - w_n[p_n/w_n] + 1, \ldots, w_n\}, \end{cases}$$

where $[z]$ denotes the integer part of $z \in \mathbb{R}$.

Value

`call`	The matched call.
`fitted.values`	Estimated scalar response.
`residuals`	Differences between `y` and the `fitted.values`
`beta.est`	$\hat{\boldsymbol{\beta}}$ (i. e. estimate of $\boldsymbol{\beta}_0$ when the optimal tuning parameters `w.opt`, `lambda.opt`, `h.opt` and `vn.opt` are used).
`beta.red`	Estimate of $\beta_0^{\mathbf{1}}$ in the reduced model when the optimal tuning parameters `w.opt`, `lambda.opt`, `h.opt` and `vn.opt` are used.
`theta.est`	Coefficients of $\hat{\theta}$ in the B-spline basis (i. e. estimate of $\theta_0$when the optimal tuning parameters `w.opt`, `lambda.opt`, `h.opt` and `vn.opt` are used): a vector of `length(order.Bspline+nknot.theta)`.
`indexes.beta.nonnull`	Indexes of the non-zero $\hat{\beta}_j$.
`h.opt`	Selected bandwidth (when `w.opt` is considered).
`w.opt`	Selected size for $\mathcal{R}_n^{\mathbf{1}}$.
`lambda.opt`	Selected value of the penalisation parameter (when `w.opt` is considered).
`IC`	Value of the criterion function considered to select `w.opt`, `lambda.opt`, `h.opt` and `vn.opt`.

`vn.opt`	Selected value of `vn` (when `w.opt` is considered).
`beta.w`	Estimate of $\beta_0^{\mathbf{1}}$ for each value of the sequence `wn`.
`theta.w`	Estimate of $\theta_0^{\mathbf{1}}$ for each value of the sequence `wn` (i.e. its coefficients in the B-spline basis).
`IC.w`	Value of the criterion function for each value of the sequence `wn`.
`indexes.beta.nonnull.w`	Indexes of the non-zero linear coefficients for each value of the sequence `wn`.
`lambda.w`	Selected value of penalisation parameter for each value of the sequence `wn`.
`h.w`	Selected bandwidth for each value of the sequence `wn`.
`index01`	Indexes of the covariates (in the whole set of p_n) used to build $\mathcal{R}_n^{\mathbf{1}}$ for each value of the sequence `wn`.
`...`	

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

References

Novo, S., Vieu, P., and Aneiros, G., (2021) Fast and efficient algorithms for sparse semiparametric bi-functional regression. *Australian and New Zealand Journal of Statistics*, **63**, 606–638, doi:10.1111/anzs.12355.

See Also

See also `sfplsim.kernel.fit`, `predict.FASSMR.kernel`, `plot.FASSMR.kernel` and `IASSMR.kernel.fit`.

Alternative method `FASSMR.kNN.fit`.

Examples

```
data(Sugar)

y<-Sugar$ash
x<-Sugar$wave.290
z<-Sugar$wave.240

#Outliers
index.y.25 <- y > 25
index.atip <- index.y.25
(1:268)[index.atip]


#Dataset to model
x.sug <- x[!index.atip,]
z.sug<- z[!index.atip,]
```

```
y.sug <- y[!index.atip]

train<-1:216

ptm=proc.time()
fit <- FASSMR.kernel.fit(x=x.sug[train,],z=z.sug[train,], y=y.sug[train],
        nknot.theta=2,lambda.min.h=0.03, lambda.min.l=0.03,
        max.q.h=0.35,num.h = 10, nknot=20,criterion="BIC",
        penalty="grSCAD",max.iter=5000)
proc.time()-ptm

fit
names(fit)
```

`FASSMR.kNN.fit` *FASSMR with kNN estimation*

Description

This function computes the fast algorithm for sparse semiparametric multi-functional regression (FASSMR) with kNN estimation.

This algorithm involves the penalised least-squares regularization procedure combined with k-nearest neighbours (kNN) estimation with Nadaraya-Watson weights. The procedure requires the B-spline representation to estimate the functional index θ_0 and an objective criterion (`criterion`) to select the initial number of covariates in the reduced model (`w.opt`), the number of neighbours (`k.opt`) and the penalisation parameter (`lambda.opt`).

Usage

```
FASSMR.kNN.fit(x, z, y, seed.coeff = c(-1, 0, 1), order.Bspline = 3,
  nknot.theta = 3, t0 = NULL, knearest = NULL, min.knn = 2, max.knn = NULL,
  step = NULL, range.grid = NULL, kind.of.kernel = "quad", nknot = NULL,
  lambda.min = NULL, lambda.min.h= NULL, lambda.min.l = NULL,
  factor.pn = 1, nlambda = 100, vn = ncol(z), nfolds = 10, seed = 123,
  wn = c(10, 15, 20), criterion = c("GCV", "BIC", "AIC", "k-fold-CV"),
  penalty = c("grLasso", "grMCP", "grSCAD", "gel", "cMCP",
  "gBridge", "gLasso", "gMCP"),  max.iter = 1000)
```

Arguments

`x`	Matrix containing the observations of the functional covariate collected by row (functional single-index component).
`z`	Matrix containing the observations of the functional covariate that is discretised collected by row (linear component).
`y`	Vector containing the scalar response.

`seed.coeff` Vector of initial values used to build the set Θ_n (see section `Details`). The coefficients for the B-spline representation of each eligible functional index $\theta \in \Theta_n$ are obtained from `seed.coeff`. The default is `c(-1,0,1)`.

`order.Bspline` Positive integer giving the order of the B-spline basis functions. This is the number of coefficients in each piecewise polynomial segment. The default is 3.

`nknot.theta` Positive integer indicating the number of uniform interior knots of the B-spline basis for the B-spline representation of θ_0. The default is 3.

`t0` Value in the domain of the functional indexes at which we evaluate them to build the set Θ_n. We assume $\theta_0(t_0) > 0$ for some arbitrary t_0 in the domain to ensure model identifiability. If `t0=NULL`, then `mean(range.grid)` is considered.

`knearest` Vector of positive integers containing the sequence in which the number of nearest neighbours `k.opt` is selected. If `knearest=NULL`, then `knearest <- seq(from =min.knn, to = max.knn, by = step)`.

`min.knn` Positive integer indicating the minumum value of the sequence in which the number of nearest neighbours `k.opt` is selected (thus, this number must be smaller than the sample size). The default is 2.

`max.knn` Positive integer indicating the maximum value of the sequence in which the number of nearest neighbours `k.opt` is selected (thus, this number must be larger than `min.kNN` and smaller than the sample size, `n`). The default is `max.knn <- n%/%2`.

`step` Positive integer used to build the sequence of k-nearest neighbours in the following way: `min.knn`, `min.knn + step`, `min.knn + 2*step`, `min.knn + 3*step`, ... The default is `step<-ceiling(n/100)`.

`range.grid` Vector of length 2 containing the endpoints of the grid at which the observations of the functional covariate `x` are evaluated (i.e. the range of the discretization). If `range.grid=NULL`, then `range.grid=c(1,p)` is considered, where `p` is the size of the discretization size of `x` (i.e. `ncol(x)`).

`kind.of.kernel` The type of kernel function used. Only Epanechnikov kernel (`"quad"`) is available.

`nknot` Positive integer indicating the number of interior knots for the B-spline representation of the functional covariate. The default value is `(p - order.Bspline - 1)%/%2`.

`lambda.min` The smallest value for lambda (i. e., the smallest value of the sequence in which `lambda.opt` is selected), as fraction of `lambda.max`. The defaults is `lambda.min.l` if the number of observations is larger than `factor.pn` times the number of covariates and `lambda.min.h` otherwise.

`lambda.min.h` The smallest value of the sequence in which `lambda.opt` is selected if the number of observations is smaller than `factor.pn` times the number of scalar covariates. The default is 0.05.

`lambda.min.l` The smallest value of the sequence in which `lambda.opt` is selected if the number of observations is larger than `factor.pn` times the number of scalar covariates. The default is 0.0001.

`factor.pn` Positive integer used to set `lambda.min`. The default value is 1.

`nlambda` Positive integer indicating the number of values of the sequence in which `lambda.opt` is selected. The default is 100.

`vn`	Positive integer or vector of positive integers indicating the number of groups of consecutive variables to be penalised together. The default value is `vn=ncol(z)`, which leads to the individual penalisation of each scalar covariate.
`nfolds`	Positive integer indicating the number of cross-validation folds (used if `criterion="k-fold-CV"`). Default is 10.
`seed`	You may set the seed of the random number generator to obtain reproducible results (used if `criterion="k-fold-CV"`). Default is 123.
`wn`	A vector of positive integers indicating the eligible number of covariates of the reduced model. See the section `Details`. The default is `c(10,15,20)`.
`criterion`	The criterion by which to select the regularization parameter `lambda.opt` and `k.opt`. One of `"GCV"`, `"BIC"`, `"AIC"` or `"k-fold-CV"`. The default is `"GCV"`.
`penalty`	The penalty function to be applied in the penalized least squares procedure. Only "grLasso" and "grSCAD" are implemented.
`max.iter`	Maximum number of iterations (total across entire path). Default is 1000.

Details

The multi-functional partial linear single-index model (MFPLSIM) is given by the expression

$$Y_i = \sum_{j=1}^{p_n} \beta_{0j}\zeta_i(t_j) + r\left(\langle\theta_0, X_i\rangle\right) + \varepsilon_i, \quad (i = 1, \ldots, n).$$

where

- Y_i is a real random response and X_i denotes a random element belonging to some separable Hilbert space $\mathcal{H}$ with inner product denoted by $\langle\cdot,\cdot\rangle$. The second functional predictor ζ_i is supposed to be a random curve defined on some interval $[a, b]$ which is observed at the points $a \leq t_1 < \cdots < t_{p_n} \leq b$.
- $\boldsymbol{\beta}_0 = (\beta_{01}, \ldots, \beta_{0p_n})^\top$ is a vector of unknown real coefficients and $r(\cdot)$ denotes a smooth unknown link function. In addition, θ_0 is an unknown functional direction in $\mathcal{H}$.
- ε_i denotes the random error.

In the MFPLSIM, we assume that only a few scalar variables from the set $\{\zeta(t_1), \ldots, \zeta(t_{p_n})\}$ form part of the model. Therefore, we must select the relevant variables in the linear component (the impact points of the curve ζ on the response) and estimate the model.

In this function, the MFPLSIM is fitted using the FASSMR algorithm. The main idea of this algorithm is to consider a reduced model, with only some (very few) linear covariates (but covering the entire discretization interval of ζ), and discarding directly the other linear covariates (since one expect that they contain very similar information about the response).

To explain the algorithm we assume, without lost of generality, that the number p_n of linear covariates can be expressed as follows: $p_n = q_n w_n$ with q_n and w_n integers. The previous consideration allows to build a subset of the initial p_n linear covariates, which contains only w_n equally spaced discretized observations of ζ covering the whole interval $[a, b]$. This subset is the following:

$$\mathcal{R}_n^{\mathbf{1}} = \left\{\zeta\left(t_k^{\mathbf{1}}\right),\ \ k = 1, \ldots, w_n\right\},$$

where $t_k^{\mathbf{1}} = t_{[(2k-1)q_n/2]}$ and $[z]$ denotes the smallest integer not less than $z \in \mathbb{R}$.

In this way, we consider the following reduced model, which involves only the linear covariates belonging to $\mathcal{R}_n^{\mathbf{1}}$:

$$Y_i = \sum_{k=1}^{w_n} \beta_{0k}^{\mathbf{1}} \zeta_i(t_k^{\mathbf{1}}) + r^{\mathbf{1}}\left(\langle \theta_0^{\mathbf{1}}, \mathcal{X}_i \rangle\right) + \varepsilon_i^{\mathbf{1}}.$$

The eligible numbers of linear covariates to build the reduced model are provided to the program in the argument `wn`. Then, the penalised least-squares variable selection procedure, with kNN estimation, is applied to the reduced model. This is done by means of the function `sfplsim.kNN.fit`, which requires remaining arguments (for details, see the documentation of the function `sfplsim.kNN.fit`). The estimates obtained after that are the outputs of the FASSMR algorithm. For further details on this algorithm, see Novo et al. (2021).

Remark: If the condition $p_n = w_n q_n$ fails, the function considers not fixed $q_n = q_{n,k}$ values $k = 1, \ldots, w_n$, when p_n/w_n is not an integer number. Specifically:

$$q_{n,k} = \begin{cases} [p_n/w_n] + 1 & k \in \{1, \ldots, p_n - w_n[p_n/w_n]\}, \\ [p_n/w_n] & k \in \{p_n - w_n[p_n/w_n] + 1, \ldots, w_n\}, \end{cases}$$

where $[z]$ denotes the integer part of $z \in \mathbb{R}$.

Value

`call`	The matched call.
`fitted.values`	Estimated scalar response.
`residuals`	Differences between `y` and the `fitted.values`
`beta.est`	$\hat{\boldsymbol{\beta}}$ (i. e. estimate of $\boldsymbol{\beta}_0$ when the optimal tuning parameters `w.opt`, `lambda.opt`, `k.opt` and `vn.opt` are used).
`beta.red`	Estimate of $\beta_0^{\mathbf{1}}$ in the reduced model when the optimal tuning parameters `w.opt`, `lambda.opt`, `k.opt` and `vn.opt` are used.
`theta.est`	Coefficients of $\hat{\theta}$ in the B-spline basis (i. e. estimate of θ_0 when the optimal tuning parameters `w.opt`, `lambda.opt`, `k.opt` and `vn.opt` are used): a vector of `length(order.Bspline+nknot.theta)`.
`indexes.beta.nonnull`	Indexes of the non-zero $\hat{\beta}_j$.
`k.opt`	Selected number of nearest neighbours (when `w.opt` is considered).
`w.opt`	Selected size for $\mathcal{R}_n^{\mathbf{1}}$.
`lambda.opt`	Selected value of the penalisation parameter (when `w.opt` is considered).
`IC`	Value of the criterion function considered to select `w.opt`, `lambda.opt`, `k.opt` and `vn.opt`.
`vn.opt`	Selected value of `vn` (when `w.opt` is considered).
`beta.w`	Estimate of $\beta_0^{\mathbf{1}}$ for each value of the sequence `wn` (i.e. for each number of covariates in the reduced model).
`theta.w`	Estimate of $\theta_0^{\mathbf{1}}$ for each value of the sequence `wn` (i.e. its coefficients in the B-spline basis).
`IC.w`	Value of the criterion function for each value of the sequence `wn`.

`indexes.beta.nonnull.w`
: Indexes of the non-zero linear coefficients for each value of the sequence `wn`.

`lambda.w`
: Selected value of penalisation parameter for each value of the sequence `wn`.

`k.w`
: Selected number of neighbours for each value of the sequence `wn`.

`index01`
: Indexes of the covariates (in the whole set of p_n) used to build $\mathcal{R}_n^{\mathbf{1}}$ for each value of the sequence `wn`.

`...`

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

References

Novo, S., Vieu, P., and Aneiros, G., (2021) Fast and efficient algorithms for sparse semiparametric bi-functional regression. *Australian and New Zealand Journal of Statistics*, **63**, 606–638, doi:10.1111/anzs.12355.

See Also

See also `sfplsim.kNN.fit`, `predict.FASSMR.kNN`, `plot.FASSMR.kNN` and `IASSMR.kNN.fit`.

Alternative method `FASSMR.kernel.fit`

Examples

```
data(Sugar)


y<-Sugar$ash
x<-Sugar$wave.290
z<-Sugar$wave.240

#Outliers
index.y.25 <- y > 25
index.atip <- index.y.25
(1:268)[index.atip]


#Dataset to model
x.sug <- x[!index.atip,]
z.sug<- z[!index.atip,]
y.sug <- y[!index.atip]

train<-1:216

ptm=proc.time()
fit<- FASSMR.kNN.fit(x=x.sug[train,],z=z.sug[train,], y=y.sug[train],
```

```
        nknot.theta=2,lambda.min.h=0.03, lambda.min.l=0.03,
        max.knn=20,nknot=20,criterion="BIC", penalty="grSCAD",max.iter=5000)
proc.time()-ptm

fit
names(fit)
```

`fsemipar.internal` *Package fsemipar internal functions*

Description

List of the internal functions. The construction of this code is based on that by F. Ferraty, which is available on his website https://www.math.univ-toulouse.fr/~ferraty/SOFTWARES/NPFDA/index.html.

Details

- approx.spline.deriv
- Bspline.ini
- fnp.kernel.fit
- fnp.kernel.fit.test
- fnp.kernel.test
- fnp.kNN.fit
- fnp.kNN.fit.test
- fnp.kNN.fit.test.loc
- fnp.kNN.GCV
- fnp.kNN.test
- fsim.kernel.fit.fixedtheta
- fsim.kNN.fit.fixedtheta
- fun.kernel
- fun.kernel.fixedtheta
- fun.kNN
- fun.kNN.fixedtheta
- funopare.kNN
- H.fnp.kernel
- H.fnp.kNN
- H.fsim.kernel
- H.fsim.kNN

- interp.spline.deriv
- normaliza
- quad
- semimetric.deriv
- semimetric.interv
- semimetric.pca
- sfplsim.kernel.fit.fixedtheta
- sfplsim.kNN.fit.fixedtheta
- Splinemlf
- symsolve

`fsim.kernel.fit` *Functional single-index model fit using kernel estimation*

Description

This function fits a functional single-index model (FSIM) between a functional explanatory variable and scalar response. The function uses kernel estimation with Nadaraya-Watson weights, a B-spline representation to estimate the functional index θ_0 and the cross-validation (CV) criterion to select the bandwidth (`h.opt`) and the coefficients of the functional index in the spline basis (`theta.est`).

Usage

```
fsim.kernel.fit(x, y, seed.coeff = c(-1, 0, 1), nknot.theta = 3,
  order.Bspline = 3, t0 = NULL, min.q.h = 0.05, max.q.h = 0.5,
  h.seq = NULL, num.h = 10, kind.of.kernel = "quad", range.grid = NULL,
  nknot = NULL)
```

Arguments

`x`	Matrix containing the observations of the functional covariate (i.e. curves) collected by row.
`y`	Vector containing the scalar response.
`seed.coeff`	Vector of initial values used to build the set Θ_n (see section `Details`). The coefficients for the B-spline representation of each eligible functional index $\theta \in \Theta_n$ are obtained from `seed.coeff`. The default is `c(-1,0,1)`.
`nknot.theta`	Positive integer indicating the number of uniform interior knots of the B-spline basis for the B-spline representation of θ_0. The default is 3.
`order.Bspline`	Positive integer giving the order of the B-spline basis functions. This is the number of coefficients in each piecewise polynomial segment. The default is 3.
`t0`	Value in the domain of the functional indexes at which we evaluate them to build the set Θ_n. We assume $\theta_0(t_0) > 0$ for some arbitrary t_0 in the domain to ensure model identifiability. If `t0=NULL`, then `mean(range.grid)` is considered.

`min.q.h`	Order of the quantile of the set of distances between curves (computed with the projection semi-metric) which gives the lower end of the sequence in which the bandwidth is selected. The default is 0.05.
`max.q.h`	Order of the quantile of the set of distances between curves (computed with the projection semi-metric) which gives the upper end of the sequence in which the bandwidth is selected. The default is 0.5.
`h.seq`	Vector containing the sequence of bandwidths. The default is a sequence of `num.h` equispaced bandwidths in the range constructed using `min.q.h` and `max.q.h`.
`num.h`	Positive integer indicating the number of bandwiths in the grid. The default is 10.
`kind.of.kernel`	The type of kernel function used. Only Epanechnikov kernel (`"quad"`) is available.
`range.grid`	Vector of length 2 containing the endpoints of the grid at which the observations of the functional covariate `x` are evaluated (i.e. the range of the discretization). If `range.grid=NULL`, then `range.grid=c(1,p)` is considered, where `p` is the size of the discretization size of `x` (i.e. `ncol(x)`).
`nknot`	Positive integer indicating the number of interior knots for the B-spline representation of the functional covariate. The default value is `(p - order.Bspline - 1)%/%2`.

Details

The functional single-index model (FSIM) is given by the expression:

$$Y_i = r(\langle \theta_0, X_i \rangle) + \varepsilon_i, \quad i = 1, \ldots, n,$$

where Y_i denotes a scalar response, X_i is a functional covariate valued in a separable Hilbert space $\mathcal{H}$ with inner product $\langle \cdot, \cdot \rangle$, ε denotes the random error, $\theta_0 \in \mathcal{H}$ is the unknown functional index and $r(\cdot)$ denotes the unknown smooth link function.

The FSIM is fitted using the kNN estimator

$$\widehat{r}_{h,\hat{\theta}}(x) = \sum_{i=1}^{n} w_{n,h,\hat{\theta}}(x, X_i) Y_i, \quad \forall x \in \mathcal{H},$$

with Nadaraya-Watson weights

$$w_{n,h,\hat{\theta}}(x, X_i) = \frac{K\left(h^{-1} d_{\hat{\theta}}\left(X_i, x\right)\right)}{\sum_{i=1}^{n} K\left(h^{-1} d_{\hat{\theta}}\left(X_i, x\right)\right)},$$

where

- $h \in \mathbb{R}^+$ is the bandwidth.
- K is a kernel function (see the argument `kind.of.kernel`).
- $d_{\hat{\theta}}(x_1, x_2) = |\langle \hat{\theta}, x_1 - x_2 \rangle|$ is the projection semi-metric, computed using `semimetric.projec` and $\hat{\theta}$ is an estimate of θ_0.

The procedure requires the estimation of the function-parameter θ_0. Therefore, we use B-spline representation to build a set Θ_n of eligible functional indexes. The dimension of the B-spline basis is `order.Bspline+nknot.theta` and the set of eligible coefficients is obtained by calibrating (to ensure the identifiability of the model) the set of initial coefficients given in `seed.coeff`. The larger this set, the higher the size of Θ_n. Since our approach requires intensive computation, we need a trade-off between the size of Θ_n and the performance of the estimator. For that, Ait-Saidi et al. (2008) suggested considering `order.Bspline=3` and `seed.coeff=c(-1,0,1)`. For details on the construction of Θ_n see Novo et al. (2019).

We obtain the estimated coefficients of θ_0 in the spline basis (`theta.est`) and the selected bandwidth (`h.opt`) by minimising the CV criterion.

Value

`call`	The matched call.
`fitted.values`	Estimated scalar response.
`residuals`	Differences between `y` and the `fitted.values`
`theta.est`	Coefficients of $\hat{\theta}$ in the B-spline basis: a vector of `length(order.Bspline+nknot.theta)`.
`h.opt`	Selected bandwidth.
`r.squared`	Coefficient of determination.
`var.res`	Redidual variance.
`df`	Residual degrees of freedom.
`yhat.cv`	Predicted values for the scalar response using leave-one-out samples.
`CV.opt`	Minimum value of the CV function, i.e. the value of CV for `theta.est` and `h.opt`.
`CV.values`	Vector containing CV values for each functional index in Θ_n and the value of h that minimises the CV for such index (i.e. `CV.values[j]` contains the value of the CV function corresponding to `theta.seq.norm[j,]` and the best value of the `h.seq` for this functional index according to the CV criterion).
`H`	Hat matrix.
`m.opt`	Index of $\hat{\theta}$ in the set Θ_n.
`theta.seq.norm`	The vector `theta.seq.norm[j,]` contains the coefficientes in the B-spline basis of the jth functional index in Θ_n.
`h.seq`	Sequence of eligible values for h.
`...`	

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

References

Ait-Saidi, A., Ferraty, F., Kassa, R., and Vieu, P. (2008) Cross-validated estimations in the single-functional index model. *Statistics*, **42(6)**, 475–494, doi:10.1080/02331880801980377.

Novo S., Aneiros, G., and Vieu, P., (2019) Automatic and location-adaptive estimation in functional single–index regression. *Journal of Nonparametric Statistics*, **31(2)**, 364–392, doi:10.1080/10485252.2019.1567726.

See Also

See also `fsim.kernel.test`, `predict.fsim.kernel`, `plot.fsim.kernel`.

Alternative procedure `fsim.kNN.fit`.

Examples

```
data(Tecator)
y<-Tecator$fat
X<-Tecator$absor.spectra2


#FSIM fit. With nknot.theta=2 and range.grid=c(850,1050),
#Theta_n contains 108 thetas.
ptm<-proc.time()
fit<-fsim.kernel.fit(y[1:160],x=X[1:160,],max.q.h=0.35, nknot=20,
range.grid=c(850,1050),nknot.theta=2)
proc.time()-ptm
fit
names(fit)
```

`fsim.kernel.test` *Functional single-index kernel predictor*

Description

Provides predictions when we compute a functional single-index model (FSIM) using the nonparametric kernel procedure between a scalar response and a functional covariate given a functional index (θ), a global bandwidth (`h`) and new observations of the functional covariate (`x.test`).

Usage

```
fsim.kernel.test(x, y, x.test, y.test, theta = theta, nknot.theta = 3,
  order.Bspline = 3, h = 0.5, kind.of.kernel = "quad", range.grid = NULL,
  nknot = NULL)
```

Arguments

x	Matrix containing the observations of the functional covariate that correspond to the training sample collected by row.
y	Vector containing the scalar responses in the training sample.
x.test	Matrix containing the observations of the functional covariate that correspond to the testing sample collected by row.
y.test	(optional) Vector/matrix containing the scalar responses in the testing sample.
theta	Vector containing the coefficients of θ in a B-spline basis, so that length(theta)=order.Bspline+nknot
nknot.theta	Positive integer indicating the number of uniform interior knots of the B-spline basis for B-spline representation of θ. The default is 3.
order.Bspline	Positive integer giving the order of the B-spline basis functions for the B-spline representation of θ. This is the number of coefficients in each piecewise polynomial segment. The default is 3.
h	Positive real number indicating the global bandwidth.
kind.of.kernel	The type of kernel function used. Only Epanechnikov kernel ("quad") is available.
range.grid	Vector of length 2 containing the endpoints of the grid at which the observations of the functional covariate x are evaluated (i.e. the range of the discretization). If range.grid=NULL, then range.grid=c(1,p) is considered, where p is the size of the discretization size of x (i.e. ncol(x)).
nknot	Positive integer indicating the number of interior knots for the B-spline representation of the functional covariate. The default value is (p - order.Bspline - 1)%/%2.

Details

The functional single-index model (FSIM) is given by the expression:

$$Y_i = r(\langle \theta_0, X_i \rangle) + \varepsilon_i, \quad i = 1, \ldots, n,$$

where Y_i denotes a scalar response, X_i is a functional covariate valued in a separable Hilbert space $\mathcal{H}$ with inner product $\langle \cdot, \cdot \rangle$, ε denotes the random error, $\theta_0 \in \mathcal{H}$ is the unknown functional index, $r(\cdot)$ denotes the unknown smooth link function and n is the training sample size.

Given $\theta \in \mathcal{H}$, $h > 0$ and a testing sample $\{X_j,\ j = 1, \ldots, n_{test}\}$, the predicted responses (see the value y.estimated.test) can be computed using the kernel procedure by means of

$$\widehat{r}_{h,\theta}(X_j) = \sum_{i=1}^{n} w_{n,h,\theta}(X_j, X_i) Y_i, \quad j = 1, \ldots, n_{test},$$

with Nadaraya-Watson weights

$$w_{n,h,\theta}(X_j, X_i) = \frac{K\left(h^{-1} d_\theta\left(X_i, X_j\right)\right)}{\sum_{i=1}^{n} K\left(h^{-1} d_\theta\left(X_i, X_j\right)\right)},$$

where

- K is a kernel function (see the argument `kind.of.kernel`).
- for $x_1, x_2 \in \mathcal{H}$, $d_\theta(x_1, x_2) = |\langle \theta, x_1 - x_2 \rangle|$ is the projection semi-metric, computed using `semimetric.projec`.

If the argument `y.test` is given to the program (i. e. `if(!is.null(y.test))`), the function provides the mean squared error of prediction (see the value `MSE.test`) calculated as `mean((y.test-y.estimated.test)^2)`.

Value

`y.estimated.test` Predicted responses.

`MSE.test` Mean squared error between predicted and observed responses in the testing sample.

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

References

Novo S., Aneiros, G., and Vieu, P., (2019) Automatic and location-adaptive estimation in functional single–index regression. *Journal of Nonparametric Statistics*, **31(2)**, 364–392, doi:10.1080/10485252.2019.1567726.

See Also

See also `fsim.kernel.fit` and `predict.fsim.kernel`.

Alternative procedure `fsim.kNN.test`.

Examples

```
data(Tecator)
y<-Tecator$fat
X<-Tecator$absor.spectra2

train<-1:160
test<-161:215

#FSIM fit. With nknot.theta=2 and range.grid=c(850,1050),
#Theta_n contains 108 thetas.
ptm<-proc.time()
fit<-fsim.kernel.fit(y=y[train],x=X[train,],max.q.h=0.35, nknot=20,
        range.grid=c(850,1050),nknot.theta=2)
proc.time()-ptm
fit

#FSIM prediction
test<-fsim.kernel.test(y=y[train],x=X[train,],x.test=X[test,],y.test=y[test],
```

```
        theta=fit$theta.est,h=fit$h.opt,nknot.theta=2,nknot=20,
        range.grid=c(850,1050))

#MSEP
test$MSE.test
```

`fsim.kNN.fit` *Functional single-index model fit using kNN estimation*

Description

This function fits a functional single-index model (FSIM) between a functional explanatory variable and scalar response. The function uses k-nearest neighbours (kNN) estimation with Nadaraya-Watson weights, a B-spline representation to estimate the functional index θ_0 and the cross-validation (CV) criterion to select the number of neighbours (`k.opt`) and the coefficients of the functional index in the spline basis (`theta.est`).

Usage

```
fsim.kNN.fit(x, y, seed.coeff = c(-1, 0, 1), order.Bspline = 3,
  nknot.theta = 3, t0 = NULL, min.knn = 2, max.knn = NULL, knearest = NULL,
  step = NULL, kind.of.kernel = "quad", range.grid = NULL, nknot = NULL)
```

Arguments

`x`	Matrix containing the observations of the functional covariate collected by row.
`y`	Vector containing the scalar response.
`seed.coeff`	Vector of initial values used to build the set Θ_n (see section **Details**). The coefficients for the B-spline representation of each eligible functional index $\theta \in \Theta_n$ are obtained from `seed.coeff`. The default is `c(-1,0,1)`.
`order.Bspline`	Positive integer giving the order of the B-spline basis functions. This is the number of coefficients in each piecewise polynomial segment. The default is 3.
`nknot.theta`	Positive integer indicating the number of uniform interior knots of the B-spline basis for the B-spline representation of θ_0. The default is 3.
`t0`	Value in the domain of the functional indexes at which we evaluate them to build the set Θ_n. We assume $\theta_0(t_0) > 0$ for some arbitrary t_0 in the domain to ensure model identifiability. If `t0=NULL`, then `mean(range.grid)` is considered.
`min.knn`	Positive integer indicating the smallest value of the sequence in which the number of nearest neighbours `k.opt` is selected (thus, this number must be smaller than the sample size). The default is 2.
`max.knn`	Positive integer indicating the largest value of the sequence in which the number of nearest neighbours `k.opt` is selected (thus, this number must be larger than `min.kNN` and smaller than the sample size, `n`). The default is `max.knn <- n%/%2`.

knearest	Vector of positive integers containing the sequence in which the number of nearest neighbours k.opt is selected. If knearest=NULL, then knearest <- seq(from =min.knn, to = max.knn, by = step).
step	Positive integer used to build the sequence of k-nearest neighbours in the following way: min.knn, min.knn + step, min.knn + 2*step, min.knn + 3*step, ... The default is step<-ceiling(n/100).
kind.of.kernel	The type of kernel function used. Only Epanechnikov kernel ("quad") is available.
range.grid	Vector of length 2 containing the endpoints of the grid at which the observations of the functional covariate x are evaluated (i.e. the range of the discretization). If range.grid=NULL, then range.grid=c(1,p) is considered, where p is the size of the discretization size of x (i.e. ncol(x)).
nknot	Positive integer indicating the number of interior knots for the B-spline representation of the functional covariate. The default value is (p - order.Bspline - 1)%/%2.

Details

The functional single-index model (FSIM) is given by the expression:

$$Y_i = r(\langle \theta_0, X_i \rangle) + \varepsilon_i, \quad i = 1, \ldots, n,$$

where Y_i denotes a scalar response, X_i is a functional covariate valued in a separable Hilbert space $\mathcal{H}$ with inner product $\langle \cdot, \cdot \rangle$, ε denotes the random error, $\theta_0 \in \mathcal{H}$ is the unknown functional index and $r(\cdot)$ denotes the unknown smooth link function.

The FSIM is fitted using the kNN estimator

$$\widehat{r}_{k,\hat{\theta}}(x) = \sum_{i=1}^{n} w_{n,k,\hat{\theta}}(x, X_i) Y_i, \quad \forall x \in \mathcal{H},$$

with Nadaraya-Watson weights

$$w_{n,k,\hat{\theta}}(x, X_i) = \frac{K\left(H_{k,x,\hat{\theta}}^{-1} d_{\hat{\theta}}\left(X_i, x\right)\right)}{\sum_{i=1}^{n} K\left(H_{k,x,\hat{\theta}}^{-1} d_{\hat{\theta}}\left(X_i, x\right)\right)},$$

where

- $k \in \mathbb{Z}^+$ is a smoothing factor (the number of nearest neighbours).
- K is a kernel function (see the argument kind.of.kernel).
- $d_{\hat{\theta}}(x_1, x_2) = |\langle \hat{\theta}, x_1 - x_2 \rangle|$ is the projection semi-metric, computed using semimetric.projec and $\hat{\theta}$ is an estimate of θ_0.
- $H_{k,x,\hat{\theta}} = \min\{h \in \mathbb{R}^+ \text{ such that } \sum_{i=1}^{n} 1_{B_{\hat{\theta}}(x,h)}(X_i) = k\}$, where $1_{B_{\hat{\theta}}(x,h)}(\cdot)$ is the indicator function of the open ball created with the projection semi-metric with centre $x \in \mathcal{H}$ and radius h.

The procedure requires the estimation of the function-parameter θ_0. Therefore, we use B-spline representation to build a set Θ_n of eligible functional indexes. The dimension of the B-spline basis is `order.Bspline+nknot.theta` and the set of eligible coefficients is obtained by calibrating (to ensure the identifiability of the model) the set of initial coefficients given in `seed.coeff`. The larger this set, the higher the size of Θ_n. Since our approach requires intensive computation, we need a trade-off between the size of Θ_n and the performance of the estimator. For that, Ait-Saidi et al. (2008) suggested considering `order.Bspline=3` and `seed.coeff=c(-1,0,1)`. For details on the construction of Θ_n see Novo et al. (2019).

We obtain the estimated coefficients of θ_0 in the spline basis (`theta.est`) and the selected number of neighbours (`k.opt`) by minimising the CV criterion.

Value

`call`	The matched call.
`fitted.values`	Estimated scalar response.
`residuals`	Differences between `y` and the `fitted.values`
`theta.est`	Coefficients of $\hat{\theta}$ in the B-spline basis: a vector of `length(order.Bspline+nknot.theta)`.
`k.opt`	Selected number of nearest neighbours.
`r.squared`	Coefficient of determination.
`var.res`	Redidual variance.
`df`	Residual degrees of freedom.
`yhat.cv`	Predicted values for the scalar response using leave-one-out samples.
`CV.opt`	Minimum value of the CV function, i.e. the value of CV for `theta.est` and `k.opt`.
`CV.values`	Vector containing CV values for each functional index in Θ_n and the value of k that minimises the CV for such index (i.e. `CV.values[j]` contains the value of the CV function corresponding to `theta.seq.norm[j,]` and the best value of the `k.seq` for this functional index according to the CV criterion).
`H`	Hat matrix.
`m.opt`	Index of $\hat{\theta}$ in the set Θ_n.
`theta.seq.norm`	The vector `theta.seq.norm[j,]` contains the coefficientes in the B-spline basis of the jth functional index in Θ_n.
`k.seq`	Sequence of eligible values for k.
`...`	

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

References

Ait-Saidi, A., Ferraty, F., Kassa, R., and Vieu, P. (2008) Cross-validated estimations in the single-functional index model, *Statistics*, **42(6)**, 475–494, doi:10.1080/02331880801980377.

Novo S., Aneiros, G., and Vieu, P., (2019) Automatic and location-adaptive estimation in functional single–index regression, *Journal of Nonparametric Statistics*, **31(2)**, 364–392, doi:10.1080/10485252.2019.1567726.

See Also

See also `fsim.kNN.test`, `predict.fsim.kNN`, `plot.fsim.kNN`.

Alternative procedure `fsim.kernel.fit`.

Examples

```
data(Tecator)
y<-Tecator$fat
X<-Tecator$absor.spectra2

#FSIM fit. With nknot.theta=2 and range.grid=c(850,1050),
#Theta_n contains 108 thetas.
ptm<-proc.time()
fit<-fsim.kNN.fit(y=y[1:160],x=X[1:160,],max.knn=20,nknot.theta=2,nknot=20,
range.grid=c(850,1050))
proc.time()-ptm
fit
names(fit)
```

`fsim.kNN.test` *Functional single-index kNN predictor*

Description

Provides predictions when we compute a functional single-index model (FSIM) using the kNN procedure between a scalar response and a functional covariate given a functional index (θ), a global number of neighbours (`k`) and new observations of the functional covariate (`x.test`).

Usage

```
fsim.kNN.test(x, y, x.test, y.test = NULL, theta, order.Bspline = 3,
  nknot.theta = 3, k = 4, kind.of.kernel = "quad", range.grid = NULL,
  nknot = NULL)
```

Arguments

x	Matrix containing the observations of the functional covariate that correspond to the training sample collected by row.
y	Vector containing the scalar responses in the training sample.
x.test	Matrix containing the observations of the functional covariate that correspond to the testing sample collected by row.
y.test	(optional) Vector/matrix containing the scalar responses in the testing sample.
theta	Vector containing the coefficients of θ in a B-spline basis, so that length(theta)=order.Bspline+nknot
order.Bspline	Positive integer giving the order of the B-spline basis functions for the B-spline representation of θ. This is the number of coefficients in each piecewise polynomial segment. The default is 3.
nknot.theta	Positive integer indicating the number of uniform interior knots of the B-spline basis for B-spline representation of θ. The default is 3.
k	Positive integer indicating the global number of neighbours.
kind.of.kernel	The type of kernel function used. Only Epanechnikov kernel ("quad") is available.
range.grid	Vector of length 2 containing the endpoints of the grid at which the observations of the functional covariate x are evaluated (i.e. the range of the discretization). If range.grid=NULL, then range.grid=c(1,p) is considered, where p is the size of the discretization size of x (i.e. ncol(x)).
nknot	Positive integer indicating the number of interior knots for the B-spline representation of the functional covariate. The default value is (p - order.Bspline - 1)%/%2.

Details

The functional single-index model (FSIM) is given by the expression:

$$Y_i = r(\langle \theta_0, X_i \rangle) + \varepsilon_i, \quad i = 1, \ldots, n,$$

where Y_i denotes a scalar response, X_i is a functional covariate valued in a separable Hilbert space $\mathcal{H}$ with inner product $\langle \cdot, \cdot \rangle$, ε denotes the random error, $\theta_0 \in \mathcal{H}$ is the unknown functional index, $r(\cdot)$ denotes the unknown smooth link function and n is the training sample size.

Given $\theta \in \mathcal{H}$, $1 < k < n$ and a testing sample $\{X_j,\ j = 1, \ldots, n_{test}\}$, the predicted responses (see the value y.estimated.test) can be computed using the kNN procedure by means of

$$\widehat{r}_{k,\theta}(X_j) = \sum_{i=1}^{n} w_{n,k,\theta}(X_j, X_i) Y_i, \quad j = 1, \ldots, n_{test},$$

with Nadaraya-Watson weights

$$w_{n,k,\theta}(X_j, X_i) = \frac{K\left(H_{k,X_j,\theta}^{-1} d_\theta\left(X_i, X_j\right)\right)}{\sum_{i=1}^{n} K\left(H_{k,X_j,\theta}^{-1} d_\theta\left(X_i, X_j\right)\right)},$$

where

- K is a kernel function (see the argument `kind.of.kernel`).
- for $x_1, x_2 \in \mathcal{H}$, $d_\theta(x_1, x_2) = |\langle \theta, x_1 - x_2 \rangle|$ is the projection semi-metric, computed using `semimetric.projec`.
- $H_{k,x,\theta} = \min \left\{ h \in \mathbb{R}^+ \text{ such that } \sum_{i=1}^{n} 1_{B_\theta(x,h)}(X_i) = k \right\}$, where $1_{B_\theta(x,h)}(\cdot)$ is the indicator function of the open ball created with the projection semi-metric with centre $x \in \mathcal{H}$ and radius h.

If the argument `y.test` is given to the program (i. e. `if(!is.null(y.test))`), the function provides the mean squared error of prediction (see the value `MSE.test`) calculated as `mean((y.test-y.estimated.test)^2)`.

Value

`y.estimated.test`
: Predicted responses.

`MSE.test`
: Mean squared error between predicted and observed responses in the testing sample.

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

References

Novo S., Aneiros, G., and Vieu, P., (2019) Automatic and location-adaptive estimation in functional single–index regression. *Journal of Nonparametric Statistics*, **31(2)**, 364–392, doi:10.1080/10485252.2019.1567726.

See Also

See also `fsim.kNN.fit` and `predict.fsim.kNN`.

Alternative procedure `fsim.kernel.test`.

Examples

```
data(Tecator)
y<-Tecator$fat
X<-Tecator$absor.spectra2


train<-1:160
test<-161:215

#FSIM fit. With nknot.theta=2 and range.grid=c(850,1050),
#Theta_n contains 108 thetas.
ptm<-proc.time()
fit<-fsim.kNN.fit(y=y[train],x=X[train,],max.knn=20,nknot.theta=2,nknot=20,
      range.grid=c(850,1050))
proc.time()-ptm
```

```
fit

#FSIM prediction
test<-fsim.kNN.test(y=y[train],x=X[train,],x.test=X[test,],y.test=y[test],
        theta=fit$theta.est,k=fit$k.opt,nknot.theta=2,order.Bspline=3,nknot=20,
        range.grid=c(850,1050))

#MSEP
test$MSE.test
```

IASSMR.kernel.fit *IASSMR with kernel estimation*

Description

This function computes the improved algorithm for sparse semiparametric multi-functional regression (IASSMR) with kernel estimation.

This algorithm involves the penalised least-squares regularization procedure combined with kernel estimation with Nadaraya-Watson weights. The procedure requires the B-spline representation to estimate the functional index θ_0 and an objective criterion (`criterion`) to select the number of covariates in the reduced model (`w.opt`), the bandwidth (`h.opt`) and the penalisation parameter (`lambda.opt`).

Usage

```
IASSMR.kernel.fit(x, z, y, train.1=NULL, train.2=NULL, seed.coeff = c(-1, 0, 1),
  order.Bspline = 3, nknot.theta = 3, t0 = NULL,min.q.h = 0.05,
  max.q.h = 0.5, h.seq = NULL, num.h = 10, range.grid = NULL,
  kind.of.kernel = "quad", nknot = NULL, lambda.min = NULL,
  lambda.min.h = NULL, lambda.min.l = NULL, factor.pn = 1,
  nlambda = 100, vn = ncol(z), nfolds = 10, seed = 123, wn = c(10, 15, 20),
  criterion = c("GCV", "BIC", "AIC", "k-fold-CV"),
  penalty = c("grLasso", "grMCP", "grSCAD", "gel", "cMCP", "gBridge",
  "gLasso", "gMCP"),max.iter = 1000)
```

Arguments

`x`	Matrix containing the observations of the functional covariate collected by row (functional single-index component).
`z`	Matrix containing the observations of the functional covariate that is discretised collected by row (linear component).
`y`	Vector containing the scalar response.
`train.1`	Indexes of the data used as the training sample in the 1st step. The default is `train.1<-1:ceiling(n/2)`.

`train.2` Indexes of the data used as the training sample in the 2nd step. The default is `train.2<-(ceiling(n/2)+1):n`.

`seed.coeff` Vector of initial values used to build the set Θ_n (see section `Details`). The coefficients for the B-spline representation of each eligible functional index $\theta \in \Theta_n$ are obtained from `seed.coeff`. The default is `c(-1,0,1)`.

`order.Bspline` Positive integer giving the order of the B-spline basis functions. This is the number of coefficients in each piecewise polynomial segment. The default is 3.

`nknot.theta` Positive integer indicating the number of uniform interior knots of the B-spline basis for the B-spline representation of θ_0. The default is 3.

`t0` Value in the domain of the functional indexes at which we evaluate them to build the set Θ_n. We assume $\theta_0(t_0) > 0$ for some arbitrary t_0 in the domain to ensure model identifiability. If `t0=NULL`, then `mean(range.grid)` is considered.

`min.q.h` Order of the quantile of the set of distances between curves (computed with the projection semi-metric) which gives the lower end of the sequence in which the bandwidth is selected. The default is 0.05.

`max.q.h` Order of the quantile of the set of distances between curves (computed with the projection semi-metric) which gives the upper end of the sequence in which the bandwidth is selected. The default is 0.5.

`h.seq` Vector containing the sequence of bandwidths. The default is a sequence of `num.h` equispaced bandwidths in the range constructed using `min.q.h` and `max.q.h`.

`num.h` Positive integer indicating the number of bandwiths in the grid. The default is 10.

`range.grid` Vector of length 2 containing the endpoints of the grid at which the observations of the functional covariate `x` are evaluated (i.e. the range of the discretization). If `range.grid=NULL`, then `range.grid=c(1,p)` is considered, where `p` is the size of the discretization size of `x` (i.e. `ncol(x)`).

`kind.of.kernel` The type of kernel function used. Only Epanechnikov kernel (`"quad"`) is available.

`nknot` Positive integer indicating the number of interior knots for the B-spline representation of the functional covariate. The default value is `(p - order.Bspline - 1)%/%2`.

`lambda.min` The smallest value for lambda (i. e., the smallest value of the sequence in which `lambda.opt` is selected), as fraction of `lambda.max`. The defaults is `lambda.min.l` if the number of observations is larger than `factor.pn` times the number of covariates and `lambda.min.h` otherwise.

`lambda.min.h` The smallest value of the sequence in which `lambda.opt` is selected if the number of observations is smaller than `factor.pn` times the number of scalar covariates. The default is 0.05.

`lambda.min.l` The smallest value of the sequence in which `lambda.opt` is selected if the number of observations is larger than `factor.pn` times the number of scalar covariates. The default is 0.0001.

`factor.pn` Positive integer used to set `lambda.min`. The default value is 1.

`nlambda` Positive integer indicating the number of values of the sequence in which `lambda.opt` is selected. The default is 100.

vn	Positive integer or vector of positive integers indicating the number of groups of consecutive variables to be penalised together. The default value is vn=ncol(z), which leads to the individual penalisation of each scalar covariate.
nfolds	Positive integer indicating the number of cross-validation folds (used if criterion="k-fold-CV"). The default is 10.
seed	You may set the seed of the random number generator to obtain reproducible results (used if criterion="k-fold-CV"). The default is 123.
wn	A vector of positive integers indicating the eligible number of covariates of the reduced model. See the section Details. The default is c(10,15,20).
criterion	The criterion by which to select the regularization parameter lambda.opt and k.opt. One of "GCV", "BIC", "AIC" or "k-fold-CV". The default is "GCV".
penalty	The penalty function to be applied in the penalized least squares procedure. Only "grLasso" and "grSCAD" are implemented.
max.iter	Maximum number of iterations (total across entire path). Default is 1000.

Details

The multi-functional partial linear single-index model (MFPLSIM) is given by the expression

$$Y_i = \sum_{j=1}^{p_n} \beta_{0j}\zeta_i(t_j) + r\left(\langle\theta_0, X_i\rangle\right) + \varepsilon_i, \quad (i = 1, \ldots, n)$$

where

- Y_i is a real random response and X_i denotes a random element belonging to some separable Hilbert space $\mathcal{H}$ with inner product denoted by $\langle\cdot,\cdot\rangle$. The second functional predictor ζ_i is supposed to be a random curve defined on some interval $[a, b]$ which is observed at the points $a \leq t_1 < \cdots < t_{p_n} \leq b$.
- $\boldsymbol{\beta}_0 = (\beta_{01}, \ldots, \beta_{0p_n})^\top$ is a vector of unknown real coefficients and $r(\cdot)$ denotes a smooth unknown link function. In addition, θ_0 is an unknown functional index in $\mathcal{H}$.
- ε_i denotes the random error.

In the MFPLSIM, we assume that only a few scalar variables from the set $\{\zeta(t_1), \ldots, \zeta(t_{p_n})\}$ form part of the model. Therefore, we must select the relevant variables in the linear component (the impact points of the curve ζ on the response) and estimate the model.

In this function, the MFPLSIM is fitted using the IASSMR. The IASSMR (version of the PVS algorithm for semiparametric regression) is an algorithm with two steps, so we split the sample into two independent subsamples (asymptotically of the same size $n_1 \sim n_2 \sim n/2$), one of them to be used in the first stage of the method and the other in the second stage.

$$\mathcal{E}^{\mathbf{1}} = \{(\zeta_i, \mathcal{X}_i, Y_i), \quad i = 1, \ldots, n_1\},$$

$$\mathcal{E}^{\mathbf{2}} = \{(\zeta_i, \mathcal{X}_i, Y_i), \quad i = n_1 + 1, \ldots, n_1 + n_2 = n\}.$$

Note that these two subsamples are specified to the programme by means of the arguments train.1 and train.2. The superscript $\boldsymbol{s}$ with $\boldsymbol{s} = \mathbf{1}, \mathbf{2}$ indicates the stage of the method in which the sample, function, variable or parameter is involved.

To explain the algorithm we assume, without lost of generality, that the number p_n of linear covariates can be expressed as follows: $p_n = q_n w_n$ with q_n and w_n integers.

1. **First step**. The fast algorithm for sparse semiparametric multi-functional regression (FASSMR) combined with kernel estimation is applied using only the subsample $\mathcal{E}^{\mathbf{1}}$ (see the documentation of the function `FASSMR.kernel.fit`). Specifically:
   - Consider a subset of the initial p_n linear covariates, which contains only w_n equally spaced discretized observations of ζ covering the whole interval $[a, b]$. This subset is the following:
   $$\mathcal{R}_n^{\mathbf{1}} = \left\{\zeta\left(t_k^{\mathbf{1}}\right),\ \ k = 1, \ldots, w_n\right\},$$
   where $t_k^{\mathbf{1}} = t_{[(2k-1)q_n/2]}$ and $[z]$ denotes the smallest integer not less than $z \in \mathbb{R}$.The size (cardinal) of this subset is provided to the program in the argument `wn` (which contains a sequence of eligible sizes).
   - Consider the following reduced model, which involves only the w_n linear covariates belonging to $\mathcal{R}_n^{\mathbf{1}}$:
   $$Y_i = \sum_{k=1}^{w_n} \beta_{0k}^{\mathbf{1}} \zeta_i(t_k^{\mathbf{1}}) + r^{\mathbf{1}}\left(\left\langle \theta_0^{\mathbf{1}}, X_i \right\rangle\right) + \varepsilon_i^{\mathbf{1}}.$$
   The penalised least-squares variable selection procedure, with kernel estimation, is applied to the reduced model. This is done by means of the function `sfplsim.kernel.fit`, which requires the remaining arguments (for details, see the documentation of the function `sfplsim.kernel.fit`). The estimates obtained after that are the outputs of the first step of the algorithm.
2. **Second step**. The variables selected in the first step and the variables in the neighbourhood of the ones selected are included. Then the penalised least-squares procedure, combined with kernel estimation, is carried out again. For that, we consider only the subsample $\mathcal{E}^{\mathbf{2}}$. Specifically:
   - Consider a new set of variables :
   $$\mathcal{R}_n^{\mathbf{2}} = \bigcup_{\left\{k, \widehat{\beta}_{0k}^{\mathbf{1}} \neq 0\right\}} \left\{\zeta(t_{(k-1)q_n+1}), \ldots, \zeta(t_{kq_n})\right\}.$$
   Denoting by $r_n = \sharp(\mathcal{R}_n^{\mathbf{2}})$, we can rename the variables in $\mathcal{R}_n^{\mathbf{2}}$ as follows:
   $$\mathcal{R}_n^{\mathbf{2}} = \left\{\zeta(t_1^{\mathbf{2}}), \ldots, \zeta(t_{r_n}^{\mathbf{2}})\right\},$$
   - Consider the following model, which involves only the linear covariates belonging to $\mathcal{R}_n^{\mathbf{2}}$
   $$Y_i = \sum_{k=1}^{r_n} \beta_{0k}^{\mathbf{2}} \zeta_i(t_k^{\mathbf{2}}) + r^{\mathbf{2}}\left(\left\langle \theta_0^{\mathbf{2}}, X_i \right\rangle\right) + \varepsilon_i^{\mathbf{2}}.$$
   The penalized least-squares variable selection procedure, with kernel estimation, is applied to this model by means of the function `sfplsim.kernel.fit`.

The outputs of the second step are the estimates of the MFPLSIM obtained with the IASSMR algorithm. For further details on this algorithm, see Novo et al. (2021).

Remark: If the condition $p_n = w_n q_n$ fails, the function considers not fixed $q_n = q_{n,k}$ values $k = 1, \ldots, w_n$, when p_n/w_n is not an integer number. Specifically:

$$q_{n,k} = \begin{cases} [p_n/w_n] + 1 & k \in \{1, \ldots, p_n - w_n[p_n/w_n]\}, \\ [p_n/w_n] & k \in \{p_n - w_n[p_n/w_n] + 1, \ldots, w_n\}, \end{cases}$$

where $[z]$ denotes the integer part of $z \in \mathbb{R}$.

Value

`call`	The matched call.
`fitted.values`	Estimated scalar response.
`residuals`	Differences between `y` and the `fitted.values`
`beta.est`	$\hat{\boldsymbol{\beta}}$ (i. e. estimate of $\boldsymbol{\beta}_0$ when the optimal tuning parameters `w.opt`, `lambda.opt`, `h.opt` and `vn.opt` are used).
`theta.est`	Coefficients of $\hat{\theta}$ in the B-spline basis (i. e. estimate of $\theta_0$when the optimal tuning parameters `w.opt`, `lambda.opt`, `h.opt` and `vn.opt` are used): a vector of `length(order.Bspline+nknot.theta)`.
`indexes.beta.nonnull`	Indexes of the non-zero $\hat{\beta}_j$.
`h.opt`	Selected bandwidth (when `w.opt` is considered).
`w.opt`	Selected size for $\mathcal{R}_n^{\mathbf{1}}$.
`lambda.opt`	Selected value of the penalisation parameter λ (when `w.opt` is considered).
`IC`	Value of the criterion function considered to select `w.opt`, `lambda.opt`, `h.opt` and `vn.opt`.
`vn.opt`	Selected value of `vn` in the second step (when `w.opt` is considered).
`beta2`	Estimate of $\boldsymbol{\beta}_0^{\mathbf{2}}$ for each value of the sequence `wn`.
`theta2`	Estimate of $\theta_0^{\mathbf{2}}$ for each value of the sequence `wn` (i.e. its coefficients in the B-spline basis).
`indexes.beta.nonnull2`	Indexes of the non-zero linear coefficients after the step 2 of the method for each value of the sequence `wn`.
`h2`	Selected bandwidth in the second step of the algorithm for each value of the sequence `wn`.
`IC2`	Optimal value of the criterion function in the second step for each value of the sequence `wn`.
`lambda2`	Selected value of penalisation parameter in the second step for each value of the sequence `wn`.
`index02`	Indexes of the covariates (in the whole set of p_n) used to build $\mathcal{R}_n^{\mathbf{2}}$ for each value of the sequence `wn`.
`beta1`	Estimate of $\boldsymbol{\beta}_0^{\mathbf{1}}$ for each value of the sequence `wn`.
`theta1`	Estimate of $\theta_0^{\mathbf{1}}$ for each value of the sequence `wn` (i.e. its coefficients in the B-spline basis).
`h1`	Selected bandwidth in the first step of the algorithm for each value of the sequence `wn`.
`IC1`	Optimal value of the criterion function in the first step for each value of the sequence `wn`.
`lambda1`	Selected value of penalisation parameter in the first step for each value of the sequence `wn`.
`index01`	Indexes of the covariates (in the whole set of p_n) used to build $\mathcal{R}_n^{\mathbf{1}}$ for each value of the sequence `wn`.

index1 Indexes of the non-zero linear coefficients after the step 1 of the method for each value of the sequence wn.

...

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

References

Novo, S., Vieu, P., and Aneiros, G., (2021) Fast and efficient algorithms for sparse semiparametric bi-functional regression. *Australian and New Zealand Journal of Statistics*, **63**, 606–638, doi:10.1111/anzs.12355.

See Also

See also sfplsim.kernel.fit, predict.IASSMR.kernel, plot.IASSMR.kernel and FASSMR.kernel.fit.

Alternative methods IASSMR.kNN.fit, FASSMR.kernel.fit and FASSMR.kNN.fit.

Examples

```
data(Sugar)


y<-Sugar$ash
x<-Sugar$wave.290
z<-Sugar$wave.240

#Outliers
index.y.25 <- y > 25
index.atip <- index.y.25
(1:268)[index.atip]


#Dataset to model
x.sug <- x[!index.atip,]
z.sug<- z[!index.atip,]
y.sug <- y[!index.atip]

train<-1:216

ptm=proc.time()
fit<- IASSMR.kernel.fit(x=x.sug[train,],z=z.sug[train,], y=y.sug[train],
        train.1=1:108,train.2=109:216,nknot.theta=2,lambda.min.h=0.03,
        lambda.min.l=0.03,  max.q.h=0.35, num.h = 10, nknot=20,
        criterion="BIC", penalty="grSCAD", max.iter=5000)
proc.time()-ptm
```

```
fit
names(fit)
```

IASSMR.kNN.fit *IASSMR with kNN estimation*

Description

This function computes the improved algorithm for sparse semiparametric multi-functional regression (IASSMR) with kNN estimation.

This algorithm involves the penalised least-squares regularization procedure combined with k-nearest neighbours (kNN) estimation with Nadaraya-Watson weights. The procedure requires the B-spline representation to estimate the functional index θ_0 and an objective criterion (`criterion`) to select the number of covariates in the reduced model (`w.opt`), the number of neighbours (`k.opt`) and the penalisation parameter (`lambda.opt`).

Usage

```
IASSMR.kNN.fit(x, z, y, train.1=NULL, train.2=NULL, seed.coeff = c(-1, 0, 1),
  order.Bspline = 3, nknot.theta = 3, t0 = NULL, knearest = NULL,
  min.knn = 2, max.knn = NULL, step = NULL, range.grid = NULL,
  kind.of.kernel = "quad", nknot = NULL, lambda.min = NULL,
  lambda.min.h = NULL, lambda.min.l = NULL, factor.pn = 1,
  nlambda = 100,vn = ncol(z), nfolds = 10, seed = 123, wn = c(10, 15, 20),
  criterion = c("GCV", "BIC", "AIC", "k-fold-CV"),
  penalty = c("grLasso", "grMCP", "grSCAD", "gel", "cMCP",
  "gBridge", "gLasso", "gMCP"), max.iter = 1000)
```

Arguments

`x`	Matrix containing the observations of the functional covariate collected by row (functional single-index component).
`z`	Matrix containing the observations of the functional covariate that is discretised collected by row (linear component).
`y`	Vector containing the scalar response.
`train.1`	Indexes of the data used as the training sample in the 1st step. The default is `train.1<-1:ceiling(n/2)`.
`train.2`	Indexes of the data used as the training sample in the 2nd step. The default is `train.2<-(ceiling(n/2)+1):n`.
`seed.coeff`	Vector of initial values used to build the set Θ_n (see section `Details`). The coefficients for the B-spline representation of each eligible functional index $\theta \in \Theta_n$ are obtained from `seed.coeff`. The default is `c(-1,0,1)`.
`order.Bspline`	Positive integer giving the order of the B-spline basis functions. This is the number of coefficients in each piecewise polynomial segment. The default is 3.

`nknot.theta` Positive integer indicating the number of uniform interior knots of the B-spline basis for the B-spline representation of θ_0. The default is 3.

`t0` Value in the domain of the functional indexes at which we evaluate them to build the set Θ_n. We assume $\theta_0(t_0) > 0$ for some arbitrary t_0 in the domain to ensure model identifiability. If `t0=NULL`, then `mean(range.grid)` is considered.

`knearest` Vector of positive integers containing the sequence in which the number of nearest neighbours `k.opt` is selected. If `knearest=NULL`, then `knearest <- seq(from =min.knn, to = max.knn, by = step)`.

`min.knn` Positive integer indicating the minumum value of the sequence in which the number of nearest neighbours `k.opt` is selected (thus, this number must be smaller than the sample size). The default is 2.

`max.knn` Positive integer indicating the maximum value of the sequence in which the number of nearest neighbours `k.opt` is selected (thus, this number must be larger than `min.kNN` and smaller than the sample size). The default is `max.knn <- n%/%2`, being $n = n_1$ in the 1st step and $n = n_2$ in the 2nd step of the method (see section `Details`).

`step` Positive integer used to build the sequence of k-nearest neighbours in the following way: `min.knn`, `min.knn + step`, `min.knn + 2*step`, `min.knn + 3*step`, ... The default is `step<-ceiling(n/100)`.

`range.grid` Vector of length 2 containing the endpoints of the grid at which the observations of the functional covariate `x` are evaluated (i.e. the range of the discretization). If `range.grid=NULL`, then `range.grid=c(1,p)` is considered, where `p` is the size of the discretization size of `x` (i.e. `ncol(x)`).

`kind.of.kernel` The type of kernel function used. Only Epanechnikov kernel (`"quad"`) is available.

`nknot` Positive integer indicating the number of interior knots for the B-spline representation of the functional covariate. The default value is `(p - order.Bspline - 1)%/%2`.

`lambda.min` The smallest value for lambda (i. e., the smallest value of the sequence in which `lambda.opt` is selected), as fraction of `lambda.max`. The defaults is `lambda.min.l` if the number of observations is larger than `factor.pn` times the number of covariates and `lambda.min.h` otherwise.

`lambda.min.h` The smallest value of the sequence in which `lambda.opt` is selected if the number of observations is smaller than `factor.pn` times the number of scalar covariates. The default is 0.05.

`lambda.min.l` The smallest value of the sequence in which `lambda.opt` is selected if the number of observations is larger than `factor.pn` times the number of scalar covariates. The default is 0.0001.

`factor.pn` Positive integer used to set `lambda.min`. The default value is 1.

`nlambda` Positive integer indicating the number of values of the sequence in which `lambda.opt` is selected. The default is 100.

`vn` Positive integer or vector of positive integers indicating the number of groups of consecutive variables to be penalised together. The default value is `vn=ncol(z)`, which leads to the individual penalisation of each scalar covariate.

`nfolds`	Positive integer indicating the number of cross-validation folds (used if `criterion="k-fold-CV"`). Default is 10.
`seed`	You may set the seed of the random number generator to obtain reproducible results (used if `criterion="k-fold-CV"`). Default is 123.
`wn`	A vector of positive integers indicating the eligible number of covariates of the reduced model. See the section `Details`. The default is `c(10,15,20)`.
`criterion`	The criterion by which to select the regularization parameter `lambda.opt` and `k.opt`. One of `"GCV"`, `"BIC"`, `"AIC"` or `"k-fold-CV"`. The default is `"GCV"`.
`penalty`	The penalty function to be applied in the penalized least squares procedure. Only "grLasso" and "grSCAD" are implemented.
`max.iter`	Maximum number of iterations (total across entire path). Default is 1000.

Details

The multi-functional partial linear single-index model (MFPLSIM) is given by the expression

$$Y_i = \sum_{j=1}^{p_n} \beta_{0j}\zeta_i(t_j) + r\left(\langle\theta_0, X_i\rangle\right) + \varepsilon_i, \quad (i = 1, \ldots, n)$$

where

- Y_i is a real random response and X_i denotes a random element belonging to some separable Hilbert space $\mathcal{H}$ with inner product denoted by $\langle\cdot,\cdot\rangle$. The second functional predictor ζ_i is supposed to be a random curve defined on some interval $[a, b]$ which is observed at the points $a \leq t_1 < \cdots < t_{p_n} \leq b$.
- $\boldsymbol{\beta}_0 = (\beta_{01}, \ldots, \beta_{0p_n})^\top$ is a vector of unknown real coefficients and $r(\cdot)$ denotes a smooth unknown link function. In addition, θ_0 is an unknown functional index in $\mathcal{H}$.
- ε_i denotes the random error.

In the MFPLSIM, we assume that only a few scalar variables from the set $\{\zeta(t_1), \ldots, \zeta(t_{p_n})\}$ form part of the model. Therefore, we must select the relevant variables in the linear component (the impact points of the curve ζ on the response) and estimate the model.

In this function, the MFPLSIM is fitted using the IASSMR. The IASSMR (version of the PVS algorithm for semiparametric regression) is an algorithm with two steps, so we split the sample into two independent subsamples (asymptotically of the same size $n_1 \sim n_2 \sim n/2$), one of them to be used in the first stage of the method and the other in the second stage.

$$\mathcal{E}^{\mathbf{1}} = \{(\zeta_i, \mathcal{X}_i, Y_i), \quad i = 1, \ldots, n_1\},$$

$$\mathcal{E}^{\mathbf{2}} = \{(\zeta_i, \mathcal{X}_i, Y_i), \quad i = n_1 + 1, \ldots, n_1 + n_2 = n\}.$$

Note that these two subsamples are specified to the programme by means of the arguments `train.1` and `train.2`. The superscript $\boldsymbol{s}$ with $\boldsymbol{s} = \mathbf{1}, \mathbf{2}$ indicates the stage of the method in which the sample, function, variable or parameter is involved.

To explain the algorithm we assume, without lost of generality, that the number p_n of linear covariates can be expressed as follows: $p_n = q_n w_n$ with q_n and w_n integers.

1. **First step**. The fast algorithm for sparse semiparametric multi-functional regression (FASSMR) combined with kNN estimation is applied using only the subsample $\mathcal{E}^{\mathbf{1}}$ (see the documentation of the function `FASSMR.kNN.fit`). Specifically:
   - Consider a subset of the initial p_n linear covariates, which contains only w_n equally spaced discretized observations of ζ covering the whole interval $[a, b]$. This subset is the following:
   $$\mathcal{R}_n^{\mathbf{1}} = \left\{\zeta\left(t_k^{\mathbf{1}}\right),\ \ k = 1, \ldots, w_n\right\},$$
   where $t_k^{\mathbf{1}} = t_{[(2k-1)q_n/2]}$ and $[z]$ denotes the smallest integer not less than $z \in \mathbb{R}$.The size (cardinal) of this subset is provided to the program in the argument `wn` (which contains a sequence of eligible sizes).
   - Consider the following reduced model, which involves only the w_n linear covariates belonging to $\mathcal{R}_n^{\mathbf{1}}$:
   $$Y_i = \sum_{k=1}^{w_n} \beta_{0k}^{\mathbf{1}} \zeta_i(t_k^{\mathbf{1}}) + r^{\mathbf{1}}\left(\left\langle \theta_0^{\mathbf{1}}, X_i \right\rangle\right) + \varepsilon_i^{\mathbf{1}}.$$
   The penalised least-squares variable selection procedure, with kNN estimation, is applied to the reduced model. This is done by means of the function `sfplsim.kNN.fit`, which requires the remaining arguments (for details, see the documentation of the function `sfplsim.kNN.fit`). The estimates obtained after that are the outputs of the first step of the algorithm.
2. **Second step**. The variables selected in the first step and the variables in the neighbourhood of the ones selected are included. Then the penalised least-squares procedure, combined with kNN estimation, is carried out again. For that, we consider only the subsample $\mathcal{E}^{\mathbf{2}}$. Specifically:
   - Consider a new set of variables:
   $$\mathcal{R}_n^{\mathbf{2}} = \bigcup_{\left\{k, \widehat{\beta}_{0k}^{\mathbf{1}} \neq 0\right\}} \left\{\zeta(t_{(k-1)q_n+1}), \ldots, \zeta(t_{kq_n})\right\}.$$
   Denoting by $r_n = \sharp(\mathcal{R}_n^{\mathbf{2}})$, we can rename the variables in $\mathcal{R}_n^{\mathbf{2}}$ as follows:
   $$\mathcal{R}_n^{\mathbf{2}} = \left\{\zeta(t_1^{\mathbf{2}}), \ldots, \zeta(t_{r_n}^{\mathbf{2}})\right\},$$
   - Consider the following model, which involves only the linear covariates belonging to $\mathcal{R}_n^{\mathbf{2}}$
   $$Y_i = \sum_{k=1}^{r_n} \beta_{0k}^{\mathbf{2}} \zeta_i(t_k^{\mathbf{2}}) + r^{\mathbf{2}}\left(\left\langle \theta_0^{\mathbf{2}}, X_i \right\rangle\right) + \varepsilon_i^{\mathbf{2}}.$$
   The penalized least-squares variable selection procedure, with kNN estimation, is applied to this model by means of the function `sfplsim.kNN.fit`.

The outputs of the second step are the estimates of the MFPLSIM obtained with the IASSMR algorithm. For further details on this algorithm, see Novo et al. (2021).

Remark: If the condition $p_n = w_n q_n$ fails, the function considers not fixed $q_n = q_{n,k}$ values $k = 1, \ldots, w_n$, when p_n/w_n is not an integer number. Specifically:

$$q_{n,k} = \begin{cases} [p_n/w_n] + 1 & k \in \{1, \ldots, p_n - w_n[p_n/w_n]\}, \\ [p_n/w_n] & k \in \{p_n - w_n[p_n/w_n] + 1, \ldots, w_n\}, \end{cases}$$

where $[z]$ denotes the integer part of $z \in \mathbb{R}$.

Value

`call`	The matched call.
`fitted.values`	Estimated scalar response.
`residuals`	Differences between `y` and the `fitted.values`
`beta.est`	$\hat{\boldsymbol{\beta}}$ (i.e. estimate of $\boldsymbol{\beta}_0$ when the optimal tuning parameters `w.opt`, `lambda.opt`, `vn.opt` and `k.opt` are used).
`theta.est`	Coefficients of $\hat{\theta}$ in the B-spline basis (i. e. estimate of $\theta_0$when the optimal tuning parameters `w.opt`, `lambda.opt`, `vn.opt` and `k.opt` are used): a vector of `length(order.Bspline+nknot.theta)`.
`indexes.beta.nonnull`	Indexes of the non-zero $\hat{\beta}_j$.
`k.opt`	Selected number of nearest neighbours (when `w.opt` is considered).
`w.opt`	Selected initial number of covariates in the reduced model.
`lambda.opt`	Selected value of the penalisation parameter λ (when `w.opt` is considered).
`IC`	Value of the criterion function considered to select `w.opt`, `lambda.opt`, `vn.opt` and `k.opt`.
`vn.opt`	Selected value of `vn` in the second step (when `w.opt` is considered).
`beta2`	Estimate of $\boldsymbol{\beta}_0^{\mathbf{2}}$ for each value of the sequence `wn`.
`theta2`	Estimate of $\theta_0^{\mathbf{2}}$ for each value of the sequence `wn` (i.e. its coefficients in the B-spline basis).
`indexes.beta.nonnull2`	Indexes of the non-zero linear coefficients after the step 2 of the method for each value of the sequence `wn`.
`knn2`	Selected number of neighbours in the second step of the algorithm for each value of the sequence `wn`.
`IC2`	Optimal value of the criterion function in the second step for each value of the sequence `wn`.
`lambda2`	Selected value of penalisation parameter in the second step for each value of the sequence `wn`.
`index02`	Indexes of the covariates (in the whole set of p_n) used to build $\mathcal{R}_n^{\mathbf{2}}$ for each value of the sequence `wn`.
`beta1`	Estimate of $\boldsymbol{\beta}_0^{\mathbf{1}}$ for each value of the sequence `wn`.
`theta1`	Estimate of $\theta_0^{\mathbf{1}}$ for each value of the sequence `wn` (i.e. its coefficients in the B-spline basis).
`knn1`	Selected number of neighbours in the first step of the algorithm for each value of the sequence `wn`.
`IC1`	Optimal value of the criterion function in the first step for each value of the sequence `wn`.
`lambda1`	Selected value of penalisation parameter in the first step for each value of the sequence `wn`.
`index01`	Indexes of the covariates (in the whole set of p_n) used to build $\mathcal{R}_n^{\mathbf{1}}$ for each value of the sequence `wn`.

index1 Indexes of the non-zero linear coefficients after the step 1 of the method for each value of the sequence wn.

...

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

References

Novo, S., Vieu, P., and Aneiros, G., (2021) Fast and efficient algorithms for sparse semiparametric bi-functional regression. *Australian and New Zealand Journal of Statistics*, **63**, 606–638, doi:10.1111/anzs.12355.

See Also

See also sfplsim.kNN.fit, predict.IASSMR.kNN, plot.IASSMR.kNN and FASSMR.kNN.fit.

Alternative method IASSMR.kernel.fit

Examples

```
data(Sugar)


y<-Sugar$ash
x<-Sugar$wave.290
z<-Sugar$wave.240

#Outliers
index.y.25 <- y > 25
index.atip <- index.y.25
(1:268)[index.atip]


#Dataset to model
x.sug <- x[!index.atip,]
z.sug<- z[!index.atip,]
y.sug <- y[!index.atip]

train<-1:216

ptm=proc.time()
fit<- IASSMR.kNN.fit(x=x.sug[train,],z=z.sug[train,], y=y.sug[train],
        train.1=1:108,train.2=109:216,nknot.theta=2,lambda.min.h=0.07,
        lambda.min.l=0.07, max.knn=20, nknot=20,criterion="BIC",
        penalty="grSCAD", max.iter=5000)
proc.time()-ptm

fit
```

```
names(fit)
```

`lm.pels.fit` *Linear model fit*

Description

This function fits a sparse linear model between a scalar response and a vector of scalar covariates. The function uses the penalised least-squares regularization procedure. The method requires an objective criterion (`criterion`) to select the regularization parameter (`lambda.opt`).

Usage

```
lm.pels.fit(z, y, lambda.min = NULL, lambda.min.h = NULL,
  lambda.min.l = NULL, factor.pn =1, nlambda = 100,lambda.seq = NULL,
  vn = ncol(z), nfolds = 10, seed = 123, criterion = c("GCV", "BIC",
  "AIC", "k-fold-CV"), penalty = c("grLasso", "grMCP",
  "grSCAD", "gel", "cMCP", "gBridge", "gLasso", "gMCP"),
  max.iter = 1000)
```

Arguments

`z`	Matrix containing the observations of the covariates collected by row.
`y`	Vector containing the scalar response.
`lambda.min`	The smallest value for lambda (i. e., the smallest value of the sequence in which `lambda.opt` is selected), as fraction of `lambda.max`. The defaults is `lambda.min.l` if the number of observations is larger than `factor.pn` times the number of covariates and `lambda.min.h` otherwise.
`lambda.min.h`	The smallest value of the sequence in which `lambda.opt` is selected if the number of observations is smaller than `factor.pn` times the number of scalar covariates. The default is 0.05.
`lambda.min.l`	The smallest value of the sequence in which `lambda.opt` is selected if the number of observations is larger than `factor.pn` times the number of scalar covariates. The default is 0.0001.
`factor.pn`	Positive integer used to set `lambda.min`. The default is 1.
`nlambda`	Positive integer indicating the number of values of the sequence in which `lambda.opt` is selected. The default is 100.
`lambda.seq`	Sequence of values in which `lambda.opt` is selected. If `lambda.seq=NULL`, then the programme builds the sequence automatically using `lambda.min` and `nlambda`.
`vn`	Positive integer or vector of positive integers indicating the number of groups of consecutive variables to be penalised together. The default value is `vn=ncol(z)`, which leads to the individual penalisation of each scalar covariate.

nfolds	Positive integer indicating the number of cross-validation folds (used if criterion="k-fold-CV"). The default is 10.
seed	You may set the seed of the random number generator to obtain reproducible results (used if criterion="k-fold-CV"). The default is 123.
criterion	The criterion by which to select the regularization parameter lambda.opt and k.opt. One of "GCV", "BIC", "AIC" or "k-fold-CV". The default is "GCV".
penalty	The penalty function to be applied in the penalized least squares procedure. Only "grLasso" and "grSCAD" are implemented.
max.iter	Maximum number of iterations (total across entire path). The default is 1000.

Details

The sparse linear model (SLM) is given by the expression:

$$Y_i = Z_{i1}\beta_{01} + \cdots + Z_{ip_n}\beta_{0p_n} + \varepsilon_i \quad i = 1, \ldots, n,$$

where Y_i denotes a scalar response, $Z_{i1}, \ldots, Z_{ip_n}$ are random covariates taking values in $\mathbb{R}$. In this equation, $\boldsymbol{\beta}_0 = (\beta_{01}, \ldots, \beta_{0p_n})^\top \in \mathbb{R}^{p_n}$ is a vector of unknown real parameters and ε_i is the random error.

In this function, the SLM is fitted using the penalised least-squares approach by minimising

$$\mathcal{Q}(\boldsymbol{\beta}) = \frac{1}{2}(\boldsymbol{Y} - \boldsymbol{Z}\boldsymbol{\beta})^\top (\boldsymbol{Y} - \boldsymbol{Z}\boldsymbol{\beta}) + n\sum_{j=1}^{p_n} \mathcal{P}_{\lambda_{j_n}}(|\beta_j|), \quad (1)$$

where $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_{p_n})^\top$, $\mathcal{P}_{\lambda_{j_n}}(\cdot)$ is a penalty function (specified in the argument penalty) and $\lambda_{j_n} > 0$ is a tuning parameter. To reduce the quantity of tuning parameters, λ_j, to be selected for each sample, we consider $\lambda_j = \lambda\widehat{\sigma}_{\beta_{0,j,OLS}}$, where $\beta_{0,j,OLS}$ denotes the OLS estimate of $\beta_{0,j}$ and $\widehat{\sigma}_{\beta_{0,j,OLS}}$ is the estimated standard deviation; λ is selected using the objetive criterion specified in the argument criterion.

For further details on the estimation procedure of the SLM, see, for instance, Fan and Li. (2001) or Fan and Lv (2011).

Remark: We should note that if we set lambda.seq$= 0$, we can obtain the non-penalised estimation of the model, i.e. the OLS estimation. It is convenient to use lambda.seq$\neq 0$ when one suspects there are irrelevant variables.

Value

call	The matched call.
fitted.values	Estimated scalar response.
residuals	Differences between y and the fitted.values
beta.est	Estimate of β_0 when the optimal penalisation parameter lambda.opt and vn.opt are used.
indexes.beta.nonnull	Indexes of the non-zero $\hat{\beta}_j$.
lambda.opt	Selected value of lambda.
IC	Value of the criterion function considered to select lambda.opt and vn.opt.
vn.opt	Selected value of vn.
...	

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

References

Fan, J., and Li, R. (2001) Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American Statistical Association*, **96**, 1348–1360, doi:10.1198/016214501753382273.

Fan, J., and Lv, J. (2011) Nonconcave penalized likelihood with NP-dimensionality, *IEEE Transactions on Information Theory*, **57(8)**, 5467–5484, https://ieeexplore.ieee.org/document/5961830.

See Also

See also PVS.fit.

Examples

```
data("Tecator")
y<-Tecator$fat
z1<-Tecator$protein
z2<-Tecator$moisture

#Quadratic, cubic and interaction effects of the scalar covariates.
z.com<-cbind(z1,z2,z1^2,z2^2,z1^3,z2^3,z1*z2)
train<-1:160


#LM fit.
ptm=proc.time()
fit<-lm.pels.fit(z=z.com[train,], y=y[train],lambda.min.h=0.02,
      lambda.min.l=0.01,factor.pn=2, max.iter=5000, criterion="BIC",
      penalty="grSCAD")
proc.time()-ptm

#Results
fit
names(fit)
```

---

plot.classes *Plot outputs from regression estimation methods*

Description

plot function for FASSMR.kernel.fit, FASSMR.kNN.fit, fsim.kernel.fit, fsim.kNN.fit, IASSMR.kernel.fit, IASSMR.kNN.fit, lm.pels.fit, PVS.fit, PVS.kernel.fit, PVS.kNN.fit, sfpl.kernel.fit, sfpl.kNN.fit,sfplsim.kernel.fit and sfplsim.kNN.fit.

Usage

```
## S3 method for class 'FASSMR.kernel'
plot(x, cex.axis = 1.5, cex.lab = 1.5, cex = 2, col = 1, cex.main = 1.5, ...)

## S3 method for class 'FASSMR.kNN'
plot(x, cex.axis = 1.5, cex.lab = 1.5, cex = 2, col = 1, cex.main = 1.5, ...)

## S3 method for class 'fsim.kernel'
plot(x, cex.axis = 1.5, cex.lab = 1.5, cex = 2, col = 1, cex.main = 1.5, ...)

## S3 method for class 'fsim.kNN'
plot(x, cex.axis = 1.5, cex.lab = 1.5, cex = 2, col = 1, cex.main = 1.5, ...)

## S3 method for class 'IASSMR.kernel'
plot(x, cex.axis = 1.5, cex.lab = 1.5, cex = 2, col = 1, cex.main = 1.5, ...)

## S3 method for class 'IASSMR.kNN'
plot(x, cex.axis = 1.5, cex.lab = 1.5, cex = 2, col = 1, cex.main = 1.5, ...)

## S3 method for class 'lm.pels'
plot(x, cex.axis = 1.5, cex.lab = 1.5, cex = 2, col = 1, cex.main = 1.5, ...)

## S3 method for class 'PVS'
plot(x, cex.axis = 1.5, cex.lab = 1.5, cex = 2, col = 1, cex.main = 1.5, ...)

## S3 method for class 'PVS.kernel'
plot(x, cex.axis = 1.5, cex.lab = 1.5, cex = 2, col = 1, cex.main = 1.5, ...)

## S3 method for class 'PVS.kNN'
plot(x, cex.axis = 1.5, cex.lab = 1.5, cex = 2, col = 1, cex.main = 1.5, ...)

## S3 method for class 'sfpl.kernel'
plot(x, cex.axis = 1.5, cex.lab = 1.5, cex = 2, col = 1, cex.main = 1.5, ...)

## S3 method for class 'sfpl.kNN'
plot(x, cex.axis = 1.5, cex.lab = 1.5, cex = 2, col = 1, cex.main = 1.5, ...)

## S3 method for class 'sfplsim.kernel'
plot(x, cex.axis = 1.5, cex.lab = 1.5, cex = 2, col = 1, cex.main = 1.5, ...)

## S3 method for class 'sfplsim.kNN'
plot(x, cex.axis = 1.5, cex.lab = 1.5, cex = 2, col = 1, cex.main = 1.5, ...)
```

Arguments

x	Output of the functions mentioned in the Description (i.e. an object of the class FASSMR.kernel, FASSMR.kNN, fsim.kernel,fsim.kNN, IASSMR.kernel, IASSMR.kNN, lm.pels, PVS, PVS.kernel, PVS.kNN, sfpl.kernel,sfpl.kNN, sfplsim.kernel or sfplsim.kNN).
cex.axis	The magnification to be used for axis annotation relative to the current setting of cex. The default is 1.5.
cex.lab	The magnification to be used for x and y labels relative to the current setting of cex. The default is 1.5.
cex	A numerical value giving the amount by which plotting text and symbols should be magnified. The default is 2.
col	A specification for the default plotting color. The default is color=1.
cex.main	The magnification to be used for main titles relative to the current setting of cex. The default is 1.5.
...	

Value

The functions return different graphical representations.

- For the classes fsim.kNN and fsim.kernel:
  1. The estimated functional index: $\hat{\theta}$.
  2. The regression fit.
- For the classes FASSMR.kernel, FASSMR.kNN, IASSMR.kernel, IASSMR.kNN, sfplsim.kernel and sfplsim.kNN.
  1. The response over the fitted.values.
  2. The residuals over the fitted.values.
  3. The estimated functional index: $\hat{\theta}$.
- For the classes lm.pels, PVS, PVS.kernel, PVS.kNN, sfpl.kernel and sfpl.kNN.
  1. The response over the fitted.values.
  2. The residuals over the fitted.values.

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

See Also

FASSMR.kernel.fit, FASSMR.kNN.fit, fsim.kernel.fit, fsim.kNN.fit, IASSMR.kernel.fit, IASSMR.kNN.fit, lm.pels.fit, PVS.fit, PVS.kernel.fit, PVS.kNN.fit, sfpl.kernel.fit, sfpl.kNN.fit, sfplsim.kernel.fit and sfplsim.kNN.fit.

predict.fsim *Prediction from functional single-index model estimates*

Description

predict method for functional single-index regression fitted using fsim.kernel.fit or fsim.kNN.fit.

Usage

```
## S3 method for class 'fsim.kernel'
predict(object, newdata = NULL, y.test = NULL, ...)
## S3 method for class 'fsim.kNN'
predict(object, newdata = NULL, y.test = NULL, ...)
```

Arguments

object	Output of the fsim.kernel.fit or fsim.kNN.fit functions (i.e. an object of the class fsim.kernel or fsim.kNN).
newdata	A matrix containing new observations of the functional covariate collected by row.
y.test	(optional) A vector containing the new observations of the response.
...	

Details

The prediction is computed using the functions fsim.kernel.test and fsim.kernel.fit, respectively.

Value

The function returns the predicted values of the response (y) for newdata. If !is.null(y.test), it also provides the mean squared error of prediction (MSEP) computed as mean((y-y.test)^2). If is.null(newdata) the function returns the fitted values.

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

See Also

fsim.kernel.fit and fsim.kernel.test or fsim.kNN.fit and fsim.kNN.test.

Examples

```
data(Tecator)
y<-Tecator$fat
X<-Tecator$absor.spectra2

train<-1:160
test<-161:215


#FSIM fit.
fit.kernel<-fsim.kernel.fit(y[train],x=X[train,],max.q.h=0.35, nknot=20,
range.grid=c(850,1050),nknot.theta=4)
fit.kNN<-fsim.kNN.fit(y=y[train],x=X[train,],max.knn=20,nknot.theta=4,
nknot=20,range.grid=c(850,1050))

test<-161:215

pred.kernel<-predict(fit.kernel,newdata=X[test,],y.test=y[test])
pred.kernel$MSEP
pred.kNN<-predict(fit.kNN,newdata=X[test,],y.test=y[test])
pred.kNN$MSEP
```

`predict.IASSMR` *Prediction from multi-functional partial linear single-index model*

Description

`predict` method for the multi-functional partial linear single-index model fitted using `IASSMR.kernel.fit` or `IASSMR.kNN.fit`.

Usage

```
## S3 method for class 'IASSMR.kernel'
predict(object, newdata.x = NULL, newdata.z = NULL,
  y.test = NULL, option = NULL, ...)
## S3 method for class 'IASSMR.kNN'
predict(object, newdata.x = NULL, newdata.z = NULL,
  y.test = NULL, option = NULL, knearest.n = object$knearest,
  min.knn.n = object$min.knn, max.knn.n = object$max.knn.n,
  step.n = object$step, ...)
```

Arguments

`object`	Output of the functions mentioned in the `Description` (i.e. an object of the class `IASSMR.kernel` or `IASSMR.kNN`).

newdata.x	A matrix containing new observations of the functional covariate in the functional single-index component collected by row.
newdata.z	Matrix containing the new observations of the scalar covariates coming from the discretisation of a curve collected by row.
y.test	(optional) A vector containing the new observations of the response.
option	Allows the choice between 1, 2 and 3. The default is 1. See the section Details.
...	Further arguments.
knearest.n	Only used for objects IASSMR.kNN if option=2 or option=3: vector of positive integers containing the sequence in which the number of nearest neighbours k.opt is selected. The default is object$knearest.
min.knn.n	Only used for objects IASSMR.kNN if option=2 or option=3: positive integer indicating the minumum value of the sequence in which the number of nearest neighbours k.opt is selected (thus, this number must be smaller than the sample size). The default is object$min.knn.
max.knn.n	Only used for objects IASSMR.kNN if option=2 or option=3: positive integer indicating the maximum value of the sequence in which the number of nearest neighbours k.opt is selected (thus, this number must be larger than min.kNN and smaller than the sample size). The default is object$max.knn.
step.n	Only used for objects IASSMR.kNN if option=2 or option=3: positive integer used to build the sequence of k-nearest neighbours in the following way: min.knn, min.knn + step, min.knn + 2*step, min.knn + 3*step,.... The default is object$step.

Details

To obtain the predictions of the response for newdata.x and newdata.z, three options are provided:

- If option=1, we mantain all the estimates (k.opt or h.opt, theta.est and beta.est) to predict the functional single-index component of the model. As we use the estimates of the second step of the algorithm, only the train.2 is used as training sample to predict. Then, it should be noted that k.opt or h.opt could not be suitable to predict the functional single-index component of the model.
- If option=2, we mantain theta.est and beta.est, while the tunning parameter (h or k) is seleted again to predict the functional single-index component of the model. This selection is performed using the cross-validation criterion in the functional single-index model associated and the complete training sample (i.e. train=c(train.1,train.2)). As we use the whole training sample (not just a subsample of it), the sample size is modified and, as a consequence, the parameters knearest, min.knn, max.knn, step given to the function IASSMR.kNN.fit may need to be provided again to compute predictions. For that, we add the arguments knearest.n, min.knn.n, max.knn.n, step.mn.
- If option=3, we mantain only the indexes of the relevant variables selected by the IASSMR. We estimate again the linear coefficients and the functional index by means of sfplsim.kernel.fit or sfplsim.kNN.fit, respectively, without penalisation (setting lambda.seq=0) and using the whole training sample (train=c(train.1,train.2)). The method provides two predictions (and MSEPs):
  - a) The prediction associated to option=1 for sfplsim.kernel or sfplsim.kNN class.

- b) The prediction associated to option=2 for sfplsim.kernel or sfplsim.kNN class.

(see the documentation of the functions predict.sfplsim.kernel and predict.sfplsim.kNN)

Value

The function returns the predicted values of the response (y) for newdata.x and newdata.z. If !is.null(y.test), it also provides the mean squared error of prediction (MSEP) computed as mean((y-y.test)^2). If option=3 two sets of predictions are provided (and two MSEP), in correspondence with the items a) and b) mentioned in the section Details. If is.null(newdata.x) or is.null(newdata.z), the function returns the fitted values.

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

See Also

sfplsim.kernel.fit, sfplsim.kNN.fit, IASSMR.kernel.fit, IASSMR.kNN.fit.

Examples

```
data(Sugar)


y<-Sugar$ash
x<-Sugar$wave.290
z<-Sugar$wave.240

#Outliers
index.y.25 <- y > 25
index.atip <- index.y.25
(1:268)[index.atip]


#Dataset to model
x.sug <- x[!index.atip,]
z.sug<- z[!index.atip,]
y.sug <- y[!index.atip]

train<-1:216
test<-217:266

#Fit
fit.kernel<-IASSMR.kernel.fit(x=x.sug[train,],z=z.sug[train,], y=y.sug[train],
            train.1=1:108,train.2=109:216,nknot.theta=2,lambda.min.h=0.03,
            lambda.min.l=0.03,  max.q.h=0.35, num.h = 10, nknot=20,
            criterion="BIC", penalty="grSCAD", max.iter=5000)

fit.kNN<- IASSMR.kNN.fit(x=x.sug[train,],z=z.sug[train,], y=y.sug[train],
```

```
            train.1=1:108,train.2=109:216,nknot.theta=2,lambda.min.h=0.07,
            lambda.min.l=0.07, max.knn=20, nknot=20,criterion="BIC",
            penalty="grSCAD", max.iter=5000)


#Predictions
predict(fit.kernel,newdata.x=x.sug[test,],newdata.z=z.sug[test,],y.test=y.sug[test],option=2)
predict(fit.kNN,newdata.x=x.sug[test,],newdata.z=z.sug[test,],y.test=y.sug[test],option=2)
```

`predict.lm` *Prediction from linear model estimates*

Description

`predict` method for:

- Linear model (LM) fitted using `lm.pels.fit`.
- Linear model with covariates coming from the discretization of a curve fitted using `PVS.fit`.

Usage

```
## S3 method for class 'lm.pels'
predict(object, newdata = NULL, y.test = NULL, ...)
## S3 method for class 'PVS'
predict(object, newdata = NULL, y.test = NULL, ...)
```

Arguments

`object`	Output of the `lm.pels.fit` or `PVS.fit` functions (i.e. an object of the class `lm.pels` or `PVS`)
`newdata`	Matrix containing the new observations of the scalar covariates (LM) or of the scalar covariates coming from the discretisation of a curve, collected by row.
`y.test`	(optional) A vector containing the new observations of the response.
`...`	

Value

The function returns the predicted values of the response (`y`) for `newdata`. If `!is.null(y.test)`, it also provides the mean squared error of prediction (`MSEP`) computed as `mean((y-y.test)^2)`. If `is.null(newdata)` the function returns the fitted values.

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

See Also

lm.pels.fit and PVS.fit.

Examples

```
data("Tecator")
y<-Tecator$fat
z1<-Tecator$protein
z2<-Tecator$moisture

#Quadratic, cubic and interaction effects of the scalar covariates.
z.com<-cbind(z1,z2,z1^2,z2^2,z1^3,z2^3,z1*z2)
train<-1:160
test<-161:215

#LM fit.
fit<-lm.pels.fit(z=z.com[train,], y=y[train],lambda.min.h=0.02,lambda.min.l=0.01,
      factor.pn=2, max.iter=5000, criterion="BIC", penalty="grSCAD")

#Predictions
predict(fit,newdata=z.com[test,],y.test=y[test])


data(Sugar)

y<-Sugar$ash
z<-Sugar$wave.240

#Outliers
index.y.25 <- y > 25
index.atip <- index.y.25
(1:268)[index.atip]


#Dataset to model
z.sug<- z[!index.atip,]
y.sug <- y[!index.atip]

train<-1:216
test<-217:266

#Fit
fit.pvs<-PVS.fit(z=z.sug[train,], y=y.sug[train],train.1=1:108,train.2=109:216,
          lambda.min.h=0.2,criterion="BIC", penalty="grSCAD", max.iter=5000)


#Predictions
predict(fit.pvs,newdata=z.sug[test,],y.test=y.sug[test])
```

predict.mfplm *Prediction from multi-functional partial linear model*

Description

predict method for the multi-functional partial linear model fitted using PVS.kernel.fit or PVS.kNN.fit.

Usage

```
## S3 method for class 'PVS.kernel'
predict(object, newdata.x = NULL, newdata.z = NULL,
  y.test = NULL, option = NULL, ...)
## S3 method for class 'PVS.kNN'
predict(object, newdata.x = NULL, newdata.z = NULL,
  y.test = NULL, option = NULL, knearest.n = object$knearest,
  min.knn.n = object$min.knn, max.knn.n = object$max.knn.n,
  step.n = object$step, ...)
```

Arguments

object	Output of the functions mentioned in the Description (i.e. an object of the class PVS.kernel or PVS.kNN).
newdata.x	A matrix containing new observations of the functional covariate in the functional nonparametric component collected by row.
newdata.z	Matrix containing the new observations of the scalar covariates coming from the discretisation of a curve collected by row.
y.test	(optional) A vector containing the new observations of the response.
option	Allows the choice between 1, 2 and 3 in the case of PVS.kernel objects and between 1, 2, 3, and 4 in PVS.kNN objects. The default is 1. See the section Details.
...	Further arguments.
knearest.n	Only used for objects PVS.kNN if option=2, option=3 or option=4: vector of positive integers containing the sequence in which the number of nearest neighbours k.opt is selected. The default is object$knearest.
min.knn.n	Only used for objects PVS.kNN if option=2, option=3 or option=4: positive integer indicating the minumum value of the sequence in which the number of nearest neighbours k.opt is selected (thus, this number must be smaller than the sample size). The default is object$min.knn.
max.knn.n	Only used for objects PVS.kNN if option=2, option=3 or option=4: positive integer indicating the maximum value of the sequence in which the number of nearest neighbours k.opt is selected (thus, this number must be larger than min.kNN and smaller than the sample size). The default is object$max.knn.

step.n — Only used for objects PVS.kNN if option=2, option=3 or option=4: positive integer used to build the sequence of k-nearest neighbours in the following way: min.knn, min.knn + step, min.knn + 2*step, min.knn + 3*step,.... The default is object$step.

Details

To obtain the predictions of the response for newdata.x and newdata.z, the following options are provided:

- If option=1, we mantain all the estimates (k.opt or h.opt and beta.est) to predict the functional nonparametric component of the model. As we use the estimates of the second step of the algorithm, only the train.2 is used as training sample to predict. Then, it should be noted that k.opt or h.opt could not be suitable to predict the functional nonparametric component of the model.
- If option=2, we mantain beta.est, while the tunning parameter (h or k) is seleted again to predict the functional nonparametric component of the model. This selection is performed using the cross-validation criterion in the functional nonparametric model associated and the complete training sample (i.e. train=c(train.1,train.2)), obtaining a global selection for h or k. As we use the whole training sample (not just a subsample of it), the sample size is modified and, as a consequence, the parameters knearest, min.knn, max.knn, step given to the function IASSMR.kNN.fit may need to be provided again to compute predictions. For that, we add the arguments knearest.n, min.knn.n, max.knn.n, step.mn.
- If option=3, we mantain only the indexes of the relevant variables selected by the IASSMR. We estimate again the linear coefficients by means of sfpl.kernel.fit or sfpl.kNN.fit, respectively, without penalisation (setting lambda.seq=0) and using the whole training sample (train=c(train.1,train.2)). The method provides two predictions (and MSEPs):
  - a) The prediction associated to option=1 for sfpl.kernel or sfpl.kNN class.
  - b) The prediction associated to option=2 for sfpl.kernel or sfpl.kNN class.

  (see the documentation of the functions predict.sfpl.kernel and predict.sfpl.kNN)
- If option=4 (option only available for the class PVS.kNN) we mantain beta.est, while the tunning parameter k is seleted again to predict the functional nonparametric component of the model. This selection is performed using the cross-validation criterion in the functional nonparametric model associated and the complete training sample (i.e. train=c(train.1,train.2)), obtaining a local selection for k.

Value

The function returns the predicted values of the response (y) for newdata.x and newdata.z. If !is.null(y.test), it also provides the mean squared error of prediction (MSEP) computed as mean((y-y.test)^2). If option=3, two sets of predictions are provided (and two MSEP), in correspondence with the items a) and b) mentioned in the section Details. If is.null(newdata.x) or is.null(newdata.z), the function returns the fitted values.

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

See Also

PVS.kernel.fit, sfpl.kernel.fit and predict.sfpl.kernel or PVS.kNN.fit, sfpl.kNN.fit and predict.sfpl.kNN.

Examples

```
data(Sugar)


y<-Sugar$ash
x<-Sugar$wave.290
z<-Sugar$wave.240

#Outliers
index.y.25 <- y > 25
index.atip <- index.y.25
(1:268)[index.atip]


#Dataset to model
x.sug <- x[!index.atip,]
z.sug<- z[!index.atip,]
y.sug <- y[!index.atip]

train<-1:216
test<-217:266

#Fit
fit.kernel<- PVS.kernel.fit(x=x.sug[train,],z=z.sug[train,],
              y=y.sug[train],train.1=1:108,train.2=109:216,
              lambda.min.h=0.03,lambda.min.l=0.03,
              max.q.h=0.35,num.h = 10, nknot=20,criterion="BIC",
              penalty="grSCAD",max.iter=5000)
fit.kNN<- PVS.kNN.fit(x=x.sug[train,],z=z.sug[train,], y=y.sug[train],
            train.1=1:108,train.2=109:216,lambda.min.h=0.07,
            lambda.min.l=0.07, nknot=20,criterion="BIC", penalty="grSCAD",
            max.iter=5000)

#Preditions
predict(fit.kernel,newdata.x=x.sug[test,],newdata.z=z.sug[test,],y.test=y.sug[test],option=2)
predict(fit.kNN,newdata.x=x.sug[test,],newdata.z=z.sug[test,],y.test=y.sug[test],option=2)
```

`predict.sfpl` *Predictions from semi-functional partial linear regression*

Description

`predict` method for the semi-functional partial linear model (SFPLM) fitted using `sfpl.kernel.fit` or `sfpl.kNN.fit`.

Usage

```
## S3 method for class 'sfpl.kernel'
predict(object, newdata.x = NULL, newdata.z = NULL,
  y.test = NULL, option = NULL, ...)
## S3 method for class 'sfpl.kNN'
predict(object, newdata.x = NULL, newdata.z = NULL,
  y.test = NULL, option = NULL, ...)
```

Arguments

object	Output of the functions mentioned in the Description (i.e. an object of the class sfpl.kernel or sfpl.kNN.
newdata.x	A matrix containing new observations of the functional covariate in the functional-single index component collected by row.
newdata.z	Matrix containing the new observations of the scalar covariate collected by row.
y.test	(optional) A vector containing the new observations of the response.
option	Allows the choice between 1 and 2 in sfpl.kernel objects, and 1, 2 and 3 in sfpl.kNN objects. The default is 1. See the section Details.
...	

Details

To obtain the predictions of the response for newdata.x and newdata.z, the following options are provided:

- If option=1, we mantain all the estimations (k.opt or h.opt and beta.est) to predict the functional nonparametric component of the model.
- If option=2, we mantain beta.est, while the tunning parameter (h or k) is seleted again to predict the functional nonparametric component of the model. This selection is performed using the cross-validation criterion in the functional nonparametric model associated, obtaining a global selection for h or k.

In the case of sfpl.kNN objects if option=3, we mantain beta.est, while the tunning parameter k is seleted again to predict the functional nonparametric component of the model. This selection is performed using the cross-validation criterion in the functional nonparametric model associated, performing a local selection for k.

Value

The function returns the predicted values of the response (y) for newdata.x and newdata.z. If !is.null(y.test), it also provides the mean squared error of prediction (MSEP) computed as mean((y-y.test)^2). If is.null(newdata.x) or is.null(newdata.z), the function returns the fitted values.

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

See Also

sfpl.kernel.fit and sfpl.kNN.fit

Examples

```
data("Tecator")
y<-Tecator$fat
X<-Tecator$absor.spectra
z1<-Tecator$protein
z2<-Tecator$moisture

#Quadratic, cubic and interaction effects of the scalar covariates.
z.com<-cbind(z1,z2,z1^2,z2^2,z1^3,z2^3,z1*z2)
train<-1:160
test<-161:215


#Fit
fit.kernel<-sfpl.kernel.fit(x=X[train,], z=z.com[train,], y=y[train],q=2,
  max.q.h=0.35,lambda.min.h=0.02,lambda.min.l=0.01,
  factor.pn=2, max.iter=5000, criterion="BIC", penalty="grSCAD",nknot=20)
fit.kNN<-sfpl.kNN.fit(y=y[train],x=X[train,], z=z.com[train,],q=2,
  max.knn=20,lambda.min.h=0.02,lambda.min.l=0.01, factor.pn=2,
  criterion="BIC",range.grid=c(850,1050), penalty="grSCAD",
  nknot=20, max.iter=5000)

#Predictions
predict(fit.kernel,newdata.x=X[test,],newdata.z=z.com[test,],y.test=y[test],
  option=2)
predict(fit.kNN,newdata.x=X[test,],newdata.z=z.com[test,],y.test=y[test],
  option=2)
```

---

predict.sfplsim.FASSMR

Prediction from functional semiparametric model estimates

Description

predict method for:

- Semi-functional partial linear single-index model (SFPLSIM) fitted using sfplsim.kernel.fit or sfplsim.kNN.fit.
- Multi-functional partial linear single-index regression (MFPLSIM) fitted using FASSMR.kernel.fit or FASSMR.kNN.fit.

Usage

```
## S3 method for class 'sfplsim.kernel'
predict(object, newdata.x = NULL, newdata.z = NULL,
  y.test = NULL, option = NULL, ...)
## S3 method for class 'sfplsim.kNN'
predict(object, newdata.x = NULL, newdata.z = NULL,
  y.test = NULL, option = NULL, ...)
## S3 method for class 'FASSMR.kernel'
predict(object, newdata.x = NULL, newdata.z = NULL,
  y.test = NULL, option = NULL, ...)
## S3 method for class 'FASSMR.kNN'
predict(object, newdata.x = NULL, newdata.z = NULL,
  y.test = NULL, option = NULL, ...)
```

Arguments

`object`	Output of the functions mentioned in the `Description` (i.e. an object of the class `sfplsim.kernel`, `sfplsim.kNN`, `FASSMR.kernel` or `FASSMR.kNN`).
`newdata.x`	A matrix containing new observations of the functional covariate in the functional-single index component collected by row.
`newdata.z`	Matrix containing the new observations of the scalar covariates (SFPLSIM) or of the scalar covariates coming from the discretisation of a curve (MFPLSIM), collected by row.
`y.test`	(optional) A vector containing the new observations of the response.
`option`	Allows the choice between 1 and 2. The default is 1. See the section `Details`.
`...`	

Details

To obtain the predictions of the response for `newdata.x` and `newdata.z`, two options are provided:

- If `option=1`, we mantain all the estimations (`k.opt` or `h.opt`, `theta.est` and `beta.est`) to predict the functional single-index component of the model.
- If `option=2`, we mantain `theta.est` and `beta.est`, while the tunning parameter (h or k) is seleted again to predict the functional single-index component of the model. This selection is performed using the cross-validation criterion in the functional single-index model associated.

Value

The function returns the predicted values of the response (`y`) for `newdata.x` and `newdata.z`. If `!is.null(y.test)`, it also provides the mean squared error of prediction (`MSEP`) computed as `mean((y-y.test)^2)`. If `is.null(newdata.x)` or `is.null(newdata.z)`, the function returns the fitted values.

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

See Also

sfplsim.kernel.fit, sfplsim.kNN.fit, FASSMR.kernel.fit or FASSMR.kNN.fit.

Examples

```
data("Tecator")
y<-Tecator$fat
X<-Tecator$absor.spectra2
z1<-Tecator$protein
z2<-Tecator$moisture

#Quadratic, cubic and interaction effects of the scalar covariates.
z.com<-cbind(z1,z2,z1^2,z2^2,z1^3,z2^3,z1*z2)
train<-1:160
test<-161:215

#SFPLSIM fit. Convergence errors for some theta are obtained.
s.fit.kernel<-sfplsim.kernel.fit(x=X[train,], z=z.com[train,], y=y[train],
            max.q.h=0.35,lambda.min.h=0.02,lambda.min.l=0.01,
            factor.pn=2, max.iter=5000, nknot.theta=4,criterion="BIC",
            penalty="grSCAD",nknot=20)
s.fit.kNN<-sfplsim.kNN.fit(y=y[train],x=X[train,], z=z.com[train,],
        max.knn=20,lambda.min.h=0.02,lambda.min.l=0.01, factor.pn=2,
        nknot.theta=4,criterion="BIC",range.grid=c(850,1050), penalty="grSCAD",
        nknot=20, max.iter=5000)


predict(s.fit.kernel,newdata.x=X[test,],newdata.z=z.com[test,],
  y.test=y[test],option=2)
predict(s.fit.kNN,newdata.x=X[test,],newdata.z=z.com[test,],
  y.test=y[test],option=2)



data(Sugar)
y<-Sugar$ash
x<-Sugar$wave.290
z<-Sugar$wave.240

#Outliers
index.y.25 <- y > 25
index.atip <- index.y.25
(1:268)[index.atip]


#Dataset to model
x.sug <- x[!index.atip,]
z.sug<- z[!index.atip,]
y.sug <- y[!index.atip]

train<-1:216
```

```
test<-217:266

m.fit.kernel <- FASSMR.kernel.fit(x=x.sug[train,],z=z.sug[train,],
                  y=y.sug[train],  nknot.theta=2,lambda.min.h=0.03,
                  lambda.min.l=0.03, max.q.h=0.35,num.h = 10,
                  nknot=20,criterion="BIC", penalty="grSCAD",max.iter=5000)


m.fit.kNN<- FASSMR.kNN.fit(x=x.sug[train,],z=z.sug[train,], y=y.sug[train],
            nknot.theta=2,lambda.min.h=0.03, lambda.min.l=0.03,
            max.knn=20,nknot=20,criterion="BIC", penalty="grSCAD",max.iter=5000)


predict(m.fit.kernel,newdata.x=x.sug[test,],newdata.z=z.sug[test,],
  y.test=y.sug[test],option=2)
predict(m.fit.kNN,newdata.x=x.sug[test,],newdata.z=z.sug[test,],
  y.test=y.sug[test],option=2)
```

`print.summary.fsim` *Summarize information from functional single-index model (FSIM) estimation methods*

Description

`summary` and `print` functions for `fsim.kNN.fit` and `fsim.kernel.fit`.

Usage

```
## S3 method for class 'fsim.kernel'
print(x, ...)
## S3 method for class 'fsim.kNN'
print(x, ...)
## S3 method for class 'fsim.kernel'
summary(object, ...)
## S3 method for class 'fsim.kNN'
summary(object, ...)
```

Arguments

`x`	Output of the `fsim.kernel.fit` or `fsim.kNN.fit` functions (i.e. an object of the class `fsim.kernel` or `fsim.kNN`).
`...`	Further arguments.
`object`	Output of the `fsim.kernel.fit` or `fsim.kNN.fit` functions (i.e. an object of the class `fsim.kernel` or `fsim.kNN`).

Value

- The matched call.
- The optimal value of the tunning parameter (`h.opt` or `k.opt`).
- Coefficients of $\hat{\theta}$ in the B-spline basis (`theta.est`: a vector of `length(order.Bspline+nknot.theta)`.
- Minimum value of the CV function, i.e. the value of CV for `theta.est` and `h.opt`/`k.opt`.
- R squared.
- Residual variance.
- Residual degrees of freedom.

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

See Also

`fsim.kernel.fit` and `fsim.kNN.fit`.

`print.summary.lm` *Summarize information from linear model estimation methods*

Description

`summary` and `print` functions for `lm.pels.fit` and `PVS.fit`.

Usage

```
## S3 method for class 'lm.pels'
print(x, ...)
## S3 method for class 'PVS'
print(x, ...)
## S3 method for class 'lm.pels'
summary(object, ...)
## S3 method for class 'PVS'
summary(object, ...)
```

Arguments

`x`	Output of the `lm.pels.fit` or `PVS.fit` functions (i.e. an object of the class `lm.pels` or `PVS`).
`...`	Further arguments.
`object`	Output of the `lm.pels.fit` or `PVS.fit` functions (i.e. an object of the class `lm.pels` or `PVS`).

Value

- The matched call.
- The estimated intercept of the model.
- The estimated vector of linear coefficients (`beta.est`).
- The number of non-zero components in `beta.est`.
- The indexes of the non-zero components in `beta.est`.
- The optimal value of the penalisation parameter (`lambda.opt`).
- The optimal value of the criterion function, i. e. the value obtained with `lambda.opt` and `vn.opt` (and `w.opt` in the case of the PVS).
- Minimum value of the penalized profile least-squares function. That is, the value obtained using `beta.est`.
- The penalty function used.
- The criterion used to select the penalisation parameter and `vn`.
- The optimal value of `vn` in the case of the `lm.pels` object.

In the case of the `PVS` objects, these functions also return the optimal initial number of covariates to build the reduced model in the first step of the algorithm (`w.opt`). This value is also selected by means of the criterion used to select the penalisation parameter.

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

See Also

`lm.pels.fit` and `PVS.fit`.

`print.summary.mfpl` *Summarize information from multi-functional partial linear model (MFPLM) estimation methods*

Description

`summary` and `print` functions for `PVS.kernel.fit` and `PVS.kNN.fit`.

Usage

```
## S3 method for class 'PVS.kernel'
print(x, ...)
## S3 method for class 'PVS.kNN'
print(x, ...)
## S3 method for class 'PVS.kernel'
summary(object, ...)
## S3 method for class 'PVS.kNN'
summary(object, ...)
```

Arguments

x	Output of the PVS.kernel.fit or PVS.kNN.fit functions (i.e. an object of the class PVS.kernel or PVS.kNN).
...	Further arguments.
object	Output of the PVS.kernel.fit or PVS.kNN.fit functions (i.e. an object of the class PVS.kernel or PVS.kNN).

Value

- The matched call.
- The optimal value of the tunning parameter (h.opt or k.opt).
- The optimal initial number of covariates to build the reduced model (w.opt).
- The estimated vector of linear coefficients (beta.est).
- The number of non-zero components in beta.est.
- The indexes of the non-zero components in beta.est.
- The optimal value of the penalisation parameter (lambda.opt).
- The optimal value of the criterion function, i. e. the value obtained with w.opt, lambda.opt, vn.opt and h.opt/k.opt
- Minimum value of the penalized profile least-squares function. That is, the value obtained using beta.est.
- The penalty function used.
- The criterion used to select the initial number of covariates used to build the reduced model, the tunning parameter, the penalisation parameter and vn.

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

See Also

PVS.kernel.fit and PVS.kNN.fit.

---

print.summary.mfplsim *Summarize information from multi-functional partial linear single-index model (MFPLSIM) estimation methods*

Description

summary and print functions for FASSMR.kernel.fit, FASSMR.kNN.fit, IASSMR.kernel.fit and IASSMR.kNN.fit.

Usage

```
## S3 method for class 'FASSMR.kernel'
print(x, ...)
## S3 method for class 'FASSMR.kNN'
print(x, ...)
## S3 method for class 'IASSMR.kernel'
print(x, ...)
## S3 method for class 'IASSMR.kNN'
print(x, ...)
## S3 method for class 'FASSMR.kernel'
summary(object, ...)
## S3 method for class 'FASSMR.kNN'
summary(object, ...)
## S3 method for class 'IASSMR.kernel'
summary(object, ...)
## S3 method for class 'IASSMR.kNN'
summary(object, ...)
```

Arguments

x	Output of the FASSMR.kernel.fit, FASSMR.kNN.fit, IASSMR.kernel.fit or IASSMR.kNN.fit functions (i.e. an object of the class FASSMR.kernel, FASSMR.kNN, IASSMR.kernel or IASSMR.kNN).
...	
object	Output of the FASSMR.kernel.fit, FASSMR.kNN.fit, IASSMR.kernel.fit or IASSMR.kNN.fit functions (i.e. an object of the class FASSMR.kernel, FASSMR.kNN, IASSMR.kernel or IASSMR.kNN).

Value

- The matched call.
- The optimal value of the tunning parameter (h.opt or k.opt).
- The optimal initial number of covariates to build the reduced model (w.opt).
- Coefficients of $\hat{\theta}$ in the B-spline basis (theta.est): a vector of length(order.Bspline+nknot.theta).
- The estimated vector of linear coefficients (beta.est).
- The number of non-zero components in beta.est.
- The indexes of the non-zero components in beta.est.
- The optimal value of the penalisation parameter (lambda.opt).
- The optimal value of the criterion function, i. e. the value obtained with w.opt, lambda.opt, vn.opt and h.opt/k.opt
- Minimum value of the penalized profile least-squares function. That is, the value obtained using theta.est and beta.est.
- The penalty function used.
- The criterion used to select the initial number of covariates used to build the reduced model, the tunning parameter, the penalisation parameter and Vn.

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

See Also

FASSMR.kernel.fit, FASSMR.kNN.fit, IASSMR.kernel.fit and IASSMR.kNN.fit.

---

print.summary.sfpl *Summarize information from semi-functional partial linear model (SFPLM) estimation methods*

Description

summary and print functions for sfpl.kNN.fit and sfpl.kernel.fit.

Usage

```
## S3 method for class 'sfpl.kernel'
print(x, ...)
## S3 method for class 'sfpl.kNN'
print(x, ...)
## S3 method for class 'sfpl.kernel'
summary(object, ...)
## S3 method for class 'sfpl.kNN'
summary(object, ...)
```

Arguments

x	Output of the sfpl.kernel.fit or sfpl.kNN.fit functions (i.e. an object of the class sfpl.kernel or sfpl.kNN).
...	Further arguments.
object	Output of the sfpl.kernel.fit or sfpl.kNN.fit functions (i.e. an object of the class sfpl.kernel or sfpl.kNN).

Value

- The matched call.
- The optimal value of the tunning parameter (h.opt or k.opt).
- The estimated vector of linear coefficients (beta.est).
- The number of non-zero components in beta.est.
- The indexes of the non-zero components in beta.est.
- The optimal value of the penalisation parameter (lambda.opt).

- The optimal value of the criterion function, i. e. the value obtained with `lambda.opt`, `vn.opt` and `h.opt`/`k.opt`
- Minimum value of the penalized profile least-squares function. That is, the value obtained using `beta.est`.
- The penalty function used.
- The criterion used to select the tunning parameter, the penalisation parameter and `vn`.
- The optimal value of `vn`.

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

See Also

`sfpl.kernel.fit` and `sfpl.kNN.fit`.

`print.summary.sfplsim` *Summarize information from semi-functional partial linear single-index model (SFPLSIM) estimation methods*

Description

`summary` and `print` functions for `sfplsim.kNN.fit` and `sfplsim.kernel.fit`.

Usage

```
## S3 method for class 'sfplsim.kernel'
print(x, ...)
## S3 method for class 'sfplsim.kNN'
print(x, ...)
## S3 method for class 'sfplsim.kernel'
summary(object, ...)
## S3 method for class 'sfplsim.kNN'
summary(object, ...)
```

Arguments

`x`	Output of the `sfplsim.kernel.fit` or `sfplsim.kNN.fit` functions (i.e. an object of the class `sfplsim.kernel` or `sfplsim.kNN`).
`...`	Further arguments.
`object`	Output of the `sfplsim.kernel.fit` or `sfplsim.kNN.fit` functions (i.e. an object of the class `sfplsim.kernel` or `sfplsim.kNN`).

Value

- The matched call.
- The optimal value of the tunning parameter (`h.opt` or `k.opt`).
- Coefficients of $\hat{\theta}$ in the B-spline basis (`theta.est`): a vector of `length(order.Bspline+nknot.theta)`.
- The estimated vector of linear coefficients (`beta.est`).
- The number of non-zero components in `beta.est`.
- The indexes of the non-zero components in `beta.est`.
- The optimal value of the penalisation parameter (`lambda.opt`).
- The optimal value of the criterion function, i. e. the value obtained with `lambda.opt`, `vn.opt` and `h.opt`/`k.opt`
- Minimum value of the penalized profile least-squares function. That is, the value obtained using `theta.est` and `beta.est`.
- The penalty function used.
- The criterion used to select the tunning parameter, the penalisation parameter and `vn`.
- The optimal value of `vn`.

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

See Also

`sfplsim.kernel.fit` and `sfplsim.kNN.fit`.

`projec` *Inner product computation*

Description

Computes the inner product between each curve collected in `data` and a particular curve θ.

Usage

```
projec(data, theta, order.Bspline = 3, nknot.theta = 3, range.grid = NULL, nknot = NULL)
```

Arguments

data	Matrix containing functional data collected by row
theta	Vector containing the coefficients of θ in a B-spline basis, so that length(theta)=order.Bspline+nknot
order.Bspline	Positive integer giving the order of the B-spline basis functions for the B-spline representation of θ. This is the number of coefficients in each piecewise polynomial segment. The default is 3.
nknot.theta	Positive integer indicating the number of uniform interior knots of the B-spline basis. The default is 3.
range.grid	Vector of length 2 containing the range of the discretization of the functional data. If range.grid=NULL, then range.grid=c(1,p) is considered, where p is the size of the discretization size of data (i.e. ncol(data)).
nknot	Positive integer indicating the number of interior knots for the B-spline representation of the functional data. The default value is (p - order.Bspline - 1)%/%2.

Value

A matrix with the inner products.

Note

The construction of this code is based on that by Frederic Ferraty, which is available on his website https://www.math.univ-toulouse.fr/~ferraty/SOFTWARES/NPFDA/index.html.

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

References

Novo S., Aneiros, G., and Vieu, P., (2019) Automatic and location-adaptive estimation in functional single–index regression. *Journal of Nonparametric Statistics*, **31(2)**, 364–392, doi:10.1080/10485252.2019.1567726.

See Also

See also semimetric.projec.

Examples

```
data("Tecator")
names(Tecator)
y<-Tecator$fat
X<-Tecator$absor.spectra

#length(theta)=6=order.Bspline+nknot.theta
projec(X,theta=c(1,0,0,1,1,-1),nknot.theta=3,nknot=20,range.grid=c(850,1050))
```

PVS.fit *PVS estimation*

Description

This function computes the partitioning variable selection algorithm (PVS) for sparse linear regression with covariates with functional origin.

This algorithm involves the penalised least-squares regularization procedure, which requires an objective criterion (`criterion`) to select the number of covariates in the reduced model `w.opt` and the penalisation parameter `lambda.opt`.

Usage

```
PVS.fit(z, y, train.1, train.2, lambda.min = NULL, lambda.min.h = NULL,
  lambda.min.l = NULL, factor.pn = 1, nlambda = 100, vn = ncol(z),
  nfolds = 10, seed = 123, wn = c(10,15,20), criterion = c("GCV", "BIC", "AIC",
  "k-fold-CV"), penalty = c("grLasso", "grMCP",
  "grSCAD", "gel", "cMCP", "gBridge", "gLasso", "gMCP"), max.iter = 1000)
```

Arguments

`z`	Matrix containing the observations of the functional covariate that is discretised collected by row.
`y`	Vector containing the scalar response.
`train.1`	Indexes of the data used as the training sample in the 1st step. The default is `train.1<-1:ceiling(n/2)`.
`train.2`	Indexes of the data used as the training sample in the 2nd step. The default is `train.2<-(ceiling(n/2)+1):n`.
`lambda.min`	The smallest value for lambda (i. e., the smallest value of the sequence in which `lambda.opt` is selected), as fraction of `lambda.max`. The defaults is `lambda.min.l` if the number of observations is larger than `factor.pn` times the number of covariates and `lambda.min.h` otherwise.
`lambda.min.h`	The smallest value of the sequence in which `lambda.opt` is selected if the number of observations is smaller than `factor.pn` times the number of scalar covariates. The default is 0.05.
`lambda.min.l`	The smallest value of the sequence in which `lambda.opt` is selected if the number of observations is larger than `factor.pn` times the number of scalar covariates. The default is 0.0001.
`factor.pn`	Positive integer used to set `lambda.min`. The default value is 1.
`nlambda`	Positive integer indicating the number of values of the sequence in which `lambda.opt` is selected. The default is 100.
`vn`	Positive integer or vector of positive integers indicating the number of groups of consecutive variables to be penalised together. The default value is `vn=ncol(z)`, which leads to the individual penalisation of each scalar covariate.

`nfolds`	Positive integer indicating the number of cross-validation folds (used if `criterion="k-fold-CV"`). Default is 10.
`seed`	You may set the seed of the random number generator to obtain reproducible results (used if `criterion="k-fold-CV"`). Default is 123.
`wn`	A vector of positive integers indicating the eligible number of covariates of the reduced model. See the section `Details`. The default is `c(10,15,20)`.
`criterion`	The criterion by which to select the regularization parameter `lambda.opt` and `k.opt`. One of `"GCV"`, `"BIC"`, `"AIC"` or `"k-fold-CV"`. The default is `"GCV"`.
`penalty`	The penalty function to be applied in the penalized least squares procedure. Only "grLasso" and "grSCAD" are implemented.
`max.iter`	Maximum number of iterations (total across entire path). Default is 1000.

Details

The sparse linear model with covariates coming from the discretization of a curve is given by the expression

$$Y_i = \sum_{j=1}^{p_n} \beta_{0j}\zeta_i(t_j) + \varepsilon_i, \quad (i = 1, \ldots, n)$$

where

- Y_i is a real random response and ζ_i is supposed to be a random curve defined on some interval $[a, b]$ which is observed at the points $a \leq t_1 < \cdots < t_{p_n} \leq b$.
- $\boldsymbol{\beta}_0 = (\beta_{01}, \ldots, \beta_{0p_n})^\top$ is a vector of unknown real coefficients.
- ε_i denotes the random error.

In this model, we assume that only a few scalar variables from the set $\{\zeta(t_1), \ldots, \zeta(t_{p_n})\}$ form part of the model. Therefore, we must select the relevant variables (the impact points of the curve ζ on the response) and estimate the model.

In this function, this model is fitted using the PVS. The PVS is an algorithm with two steps, so we split the sample into two independent subsamples (asymptotically of the same size $n_1 \sim n_2 \sim n/2$), one of them to be used in the first stage of the method and the other in the second stage.

$$\mathcal{E}^{\mathbf{1}} = \{(\zeta_i, \mathcal{X}_i, Y_i), \quad i = 1, \ldots, n_1\},$$

$$\mathcal{E}^{\mathbf{2}} = \{(\zeta_i, \mathcal{X}_i, Y_i), \quad i = n_1 + 1, \ldots, n_1 + n_2 = n\}.$$

Note that these two subsamples are specified to the programme by means of the arguments `train.1` and `train.2`. The superscript $\boldsymbol{s}$ with $\boldsymbol{s} = \mathbf{1}, \mathbf{2}$ indicates the stage of the method in which the sample, function, variable or parameter is involved.

To explain the algorithm we assume, without lost of generality, that the number p_n of linear covariates can be expressed as follows: $p_n = q_n w_n$ with q_n and w_n integers.

1. **First step**. A reduced model is consider, discarding many linear covariates. The penalised least-squares procedure is applied to the reduced model using only the subsample $\mathcal{E}^{\mathbf{1}}$. Specifically:

- Consider a subset of the initial p_n linear covariates, which contains only w_n equally spaced discretized observations of ζ covering the whole interval $[a, b]$. This subset is the following:

$$\mathcal{R}_n^{\mathbf{1}} = \left\{ \zeta\left(t_k^{\mathbf{1}}\right),\ \ k = 1, \ldots, w_n \right\},$$

where $t_k^{\mathbf{1}} = t_{[(2k-1)q_n/2]}$ and $[z]$ denotes the smallest integer not less than $z \in \mathbb{R}$.The size (cardinal) of this subset is provided to the program in the argument `wn` (which contains a sequence of eligible sizes).

- Consider the following reduced model, which involves only the w_n linear covariates belonging to $\mathcal{R}_n^{\mathbf{1}}$:

$$Y_i = \sum_{k=1}^{w_n} \beta_{0k}^{\mathbf{1}} \zeta_i(t_k^{\mathbf{1}}) + \varepsilon_i^{\mathbf{1}}.$$

The penalised least-squares variable selection procedure is applied to the reduced model. This is done by means of the function `lm.pels.fit`, which requires the remaining arguments (for details, see the documentation of the function `lm.pels.fit`). The estimates obtained after that are the outputs of the first step of the algorithm.

2. **Second step**. The variables selected in the first step and the variables in the neighbourhood of the ones selected are included. Then the penalised least-squares procedure is performed again. For that, we consider only the subsample $\mathcal{E}^{\mathbf{2}}$. Specifically:

- Consider a new set of variables :

$$\mathcal{R}_n^{\mathbf{2}} = \bigcup_{\left\{k, \widehat{\beta}_{0k}^{\mathbf{1}} \neq 0\right\}} \left\{ \zeta(t_{(k-1)q_n+1}), \ldots, \zeta(t_{kq_n}) \right\}.$$

Denoting by $r_n = \sharp(\mathcal{R}_n^{\mathbf{2}})$, we can rename the variables in $\mathcal{R}_n^{\mathbf{2}}$ as follows:

$$\mathcal{R}_n^{\mathbf{2}} = \left\{ \zeta(t_1^{\mathbf{2}}), \ldots, \zeta(t_{r_n}^{\mathbf{2}}) \right\},$$

- Consider the following model, which involves only the linear covariates belonging to $\mathcal{R}_n^{\mathbf{2}}$

$$Y_i = \sum_{k=1}^{r_n} \beta_{0k}^{\mathbf{2}} \zeta_i(t_k^{\mathbf{2}}) + \varepsilon_i^{\mathbf{2}}.$$

The penalized least-squares variable selection procedure, with kernel estimation, is applied to this model by means of the function `lm.pels.fit`.

The outputs of the second step are the estimates of the model obtained with the PVS algorithm. For further details on this algorithm, see Aneiros and Vieu (2014).

Remark: If the condition $p_n = w_n q_n$ fails, the function considers not fixed $q_n = q_{n,k}$ values $k = 1, \ldots, w_n$, when p_n/w_n is not an integer number. Specifically:

$$q_{n,k} = \begin{cases} [p_n/w_n] + 1 & k \in \{1, \ldots, p_n - w_n[p_n/w_n]\}, \\ [p_n/w_n] & k \in \{p_n - w_n[p_n/w_n] + 1, \ldots, w_n\}, \end{cases}$$

where $[z]$ denotes the integer part of $z \in \mathbb{R}$.

Value

`call`	The matched call.
`fitted.values`	Estimated scalar response.
`residuals`	Differences between `y` and the `fitted.values`
`beta.est`	$\hat{\boldsymbol{\beta}}$ (i. e. estimate of $\boldsymbol{\beta}_0$ when the optimal tuning parameters `w.opt` and `lambda.opt` are used).
`indexes.beta.nonnull`	Indexes of the non-zero $\hat{\beta}_j$.
`w.opt`	Selected size for $\mathcal{R}_n^{\mathbf{1}}$.
`lambda.opt`	Selected value of the penalisation parameter λ (when `w.opt` is considered).
`IC`	Value of the criterion function considered to select `w.opt` and `lambda.opt`.
`beta2`	Estimate of $\boldsymbol{\beta}_0^{\mathbf{2}}$ for each value of the sequence `wn`.
`indexes.beta.nonnull2`	Indexes of the non-zero linear coefficients after the step 2 of the method for each value of the sequence `wn`.
`IC2`	Optimal value of the criterion function in the second step for each value of the sequence `wn`.
`lambda2`	Selected value of penalisation parameter in the second step for each value of the sequence `wn`.
`index02`	Indexes of the covariates (in the whole set of p_n) used to build $\mathcal{R}_n^{\mathbf{2}}$ for each value of the sequence `wn`.
`beta1`	Estimate of $\boldsymbol{\beta}_0^{\mathbf{1}}$ for each value of the sequence `wn`.
`IC1`	Optimal value of the criterion function in the first step for each value of the sequence `wn`.
`lambda1`	Selected value of penalisation parameter in the first step for each value of the sequence `wn`.
`index01`	Indexes of the covariates (in the whole set of p_n) used to build $\mathcal{R}_n^{\mathbf{1}}$ for each value of the sequence `wn`.
`index1`	Indexes of the non-zero linear coefficients after the step 1 of the method for each value of the sequence `wn`.
`...`	

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

References

Aneiros, G. and Vieu, P. (2014) Variable selection in infinite-dimensional problems. *Statistics & Probability Letters*, **94**, 12–20, doi:10.1016/j.spl.2014.06.025.

See Also

See also lm.pels.fit.

Examples

```
data(Sugar)

y<-Sugar$ash
z<-Sugar$wave.240

#Outliers
index.y.25 <- y > 25
index.atip <- index.y.25
(1:268)[index.atip]


#Dataset to model
z.sug<- z[!index.atip,]
y.sug <- y[!index.atip]

train<-1:216

ptm=proc.time()
fit<- PVS.fit(z=z.sug[train,], y=y.sug[train],train.1=1:108,train.2=109:216,
        lambda.min.h=0.2,criterion="BIC", penalty="grSCAD", max.iter=5000)
proc.time()-ptm

fit
names(fit)
```

PVS.kernel.fit *PVS with kernel estimation*

Description

This function computes the partitioning variable selection algorithm (PVS) for sparse multi-functional partial linear regression.

This algorithm involves the penalised least-squares regularization procedure combined with kernel estimation with Nadaraya-Watson weights. The procedure requires an objective criterion (`criterion`) to select the number of covariates in the reduced model (`w.opt`), the bandwidth (`h.opt`) and the penalisation parameter (`lambda.opt`).

Usage

```
PVS.kernel.fit(x, z, y, train.1=NULL, train.2=NULL, semimetric = "deriv", q = NULL,
  min.q.h = 0.05, max.q.h = 0.5, h.seq = NULL, num.h = 10,
  range.grid = NULL, kind.of.kernel = "quad", nknot = NULL, lambda.min = NULL,
```

```
lambda.min.h = NULL, lambda.min.l = NULL, factor.pn = NULL,
nlambda = 100, vn = ncol(z), nfolds = 10, seed = 123, wn = c(10, 15, 20),
criterion = c("GCV", "BIC", "AIC", "k-fold-CV"),
penalty = c("grLasso", "grMCP", "grSCAD", "gel", "cMCP", "gBridge",
"gLasso", "gMCP"), max.iter = 1000)
```

Arguments

x	Matrix containing the observations of the functional covariate collected by row (functional nonparametric component).
z	Matrix containing the observations of the functional covariate that is discretised collected by row (linear component).
y	Vector containing the scalar response.
train.1	Indexes of the data used as the training sample in the 1st step. The default is train.1<-1:ceiling(n/2).
train.2	Indexes of the data used as the training sample in the 2nd step. The default is train.2<-(ceiling(n/2)+1):n.
semimetric	Semi-metric function. Only "deriv" and "pca" are implemented. By default semimetric="deriv".
q	Order of the derivative (if semimetric="deriv") or number of principal components (if semimetric="pca"). The default values are 0 and 2, respectively.
min.q.h	Order of the quantile of the set of distances between curves (computed with the provided semimetric) which gives the lower end of the sequence in which the bandwidth is selected. The default is 0.05.
max.q.h	Order of the quantile of the set of distances between curves (computed with the provided semimetric) which gives the upper end of the sequence in which the bandwidth is selected. The default is 0.5.
h.seq	Vector containing the sequence of bandwidths. The default is a sequence of num.h equispaced bandwidths in the range constructed using min.q.h and max.q.h.
num.h	Positive integer indicating the number of bandwiths in the grid. The default is 10.
range.grid	Vector of length 2 containing the endpoints of the grid at which the observations of the functional covariate x are evaluated (i.e. the range of the discretization). If range.grid=NULL, then range.grid=c(1,p) is considered, where p is the size of the discretization size of x (i.e. ncol(x)).
kind.of.kernel	The type of kernel function used. Only Epanechnikov kernel ("quad") is available.
nknot	Positive integer indicating the number of interior knots for the B-spline representation of the functional covariate. The default value is (p - order.Bspline - 1)%/%2.
lambda.min	The smallest value for lambda (i. e., the smallest value of the sequence in which lambda.opt is selected), as fraction of lambda.max. The defaults is lambda.min.l if the number of observations is larger than factor.pn times the number of covariates and lambda.min.h otherwise.

`lambda.min.h`	The smallest value of the sequence in which `lambda.opt` is selected if the number of observations is smaller than `factor.pn` times the number of scalar covariates. The default is 0.05.
`lambda.min.l`	The smallest value of the sequence in which `lambda.opt` is selected if the number of observations is larger than `factor.pn` times the number of scalar covariates. The default is 0.0001.
`factor.pn`	Positive integer used to set `lambda.min`. The default value is 1.
`nlambda`	Positive integer indicating the number of values of the sequence in which `lambda.opt` is selected. The default is 100.
`vn`	Positive integer or vector of positive integers indicating the number of groups of consecutive variables to be penalised together. The default value is `vn=ncol(z)`, which leads to the individual penalisation of each scalar covariate.
`nfolds`	Positive integer indicating the number of cross-validation folds (used if `criterion="k-fold-CV"`). The default is 10.
`seed`	You may set the seed of the random number generator to obtain reproducible results (used if `criterion="k-fold-CV"`). The default is 123.
`wn`	A vector of positive integers indicating the eligible number of covariates of the reduced model. See the section `Details`. The default is `c(10,15,20)`.
`criterion`	The criterion by which to select the regularization parameter `lambda.opt` and `k.opt`. One of `"GCV"`, `"BIC"`, `"AIC"` or `"k-fold-CV"`. The default is `"GCV"`.
`penalty`	The penalty function to be applied in the penalized least squares procedure. Only "grLasso" and "grSCAD" are implemented.
`max.iter`	Maximum number of iterations (total across entire path). Default is 1000.

Details

The multi-functional partial linear model (MFPLM) is given by the expression

$$Y_i = \sum_{j=1}^{p_n} \beta_{0j}\zeta_i(t_j) + m\left(X_i\right) + \varepsilon_i, \quad (i = 1, \ldots, n)$$

where

- Y_i is a real random response and X_i denotes a random element belonging to some semi-metric space $\mathcal{H}$. The second functional predictor ζ_i is supposed to be a random curve defined on some interval $[a, b]$ which is observed at the points $a \leq t_1 < \cdots < t_{p_n} \leq b$.
- $\boldsymbol{\beta}_0 = (\beta_{01}, \ldots, \beta_{0p_n})^\top$ is a vector of unknown real coefficients and $m(\cdot)$ denotes a smooth unknown real-valued link function.
- ε_i denotes the random error.

In the MFPLM, we assume that only a few scalar variables from the set $\{\zeta(t_1), \ldots, \zeta(t_{p_n})\}$ form part of the model. Therefore, we must select the relevant variables in the linear component (the impact points of the curve ζ on the response) and estimate the model.

In this function, the MFPLM is fitted using the PVS. The PVS is an algorithm with two steps, so we split the sample into two independent subsamples (asymptotically of the same size $n_1 \sim n_2 \sim n/2$), one of them to be used in the first stage of the method and the other in the second stage.

$$\mathcal{E}^{\mathbf{1}} = \{(\zeta_i, \mathcal{X}_i, Y_i), \quad i = 1, \ldots, n_1\},$$

$$\mathcal{E}^{\mathbf{2}} = \{(\zeta_i, \mathcal{X}_i, Y_i), \quad i = n_1 + 1, \ldots, n_1 + n_2 = n\}.$$

Note that these two subsamples are specified to the programme by means of the arguments `train.1` and `train.2`. The superscript $\boldsymbol{s}$ with $\boldsymbol{s} = \mathbf{1}, \mathbf{2}$ indicates the stage of the method in which the sample, function, variable or parameter is involved.

To explain the algorithm we assume, without lost of generality, that the number p_n of linear covariates can be expressed as follows: $p_n = q_n w_n$ with q_n and w_n integers.

1. **First step**. A reduced model is consider, discarding many linear covariates. The penalised least-squares procedure is applied to the reduced model using only the subsample $\mathcal{E}^{\mathbf{1}}$. Specifically:

   - Consider a subset of the initial p_n linear covariates, which contains only w_n equally spaced discretized observations of ζ covering the whole interval $[a, b]$. This subset is the following:
   $$\mathcal{R}_n^{\mathbf{1}} = \left\{\zeta\left(t_k^{\mathbf{1}}\right), \ k = 1, \ldots, w_n\right\},$$
   where $t_k^{\mathbf{1}} = t_{[(2k-1)q_n/2]}$ and $[z]$ denotes the smallest integer not less than $z \in \mathbb{R}$.The size (cardinal) of this subset is provided to the program in the argument `wn` (which contains a sequence of eligible sizes).
   - Consider the following reduced model, which involves only the w_n linear covariates belonging to $\mathcal{R}_n^{\mathbf{1}}$:
   $$Y_i = \sum_{k=1}^{w_n} \beta_{0k}^{\mathbf{1}} \zeta_i(t_k^{\mathbf{1}}) + m^{\mathbf{1}}(X_i) + \varepsilon_i^{\mathbf{1}}.$$
   The penalised least-squares variable selection procedure, with kernel estimation, is applied to the reduced model. This is done by means of the function `sfpl.kernel.fit`, which requires the remaining arguments (for details, see the documentation of the function `sfpl.kernel.fit`). The estimates obtained after that are the outputs of the first step of the algorithm.

2. **Second step**. The variables selected in the first step and the variables in the neighbourhood of the ones selected are included. Then the penalised least-squares procedure, combined with kernel estimation, is carried out again. For that, we consider only the subsample $\mathcal{E}^{\mathbf{2}}$. Specifically:

   - Consider a new set of variables :
   $$\mathcal{R}_n^{\mathbf{2}} = \bigcup_{\left\{k, \widehat{\beta}_{0k}^{\mathbf{1}} \neq 0\right\}} \left\{\zeta(t_{(k-1)q_n+1}), \ldots, \zeta(t_{kq_n})\right\}.$$
   Denoting by $r_n = \sharp(\mathcal{R}_n^{\mathbf{2}})$, we can rename the variables in $\mathcal{R}_n^{\mathbf{2}}$ as follows:
   $$\mathcal{R}_n^{\mathbf{2}} = \left\{\zeta(t_1^{\mathbf{2}}), \ldots, \zeta(t_{r_n}^{\mathbf{2}})\right\},$$
   - Consider the following model, which involves only the linear covariates belonging to $\mathcal{R}_n^{\mathbf{2}}$
   $$Y_i = \sum_{k=1}^{r_n} \beta_{0k}^{\mathbf{2}} \zeta_i(t_k^{\mathbf{2}}) + m^{\mathbf{2}}(X_i) + \varepsilon_i^{\mathbf{2}}.$$
   The penalized least-squares variable selection procedure, with kernel estimation, is applied to this model by means of the function `sfpl.kernel.fit`.

The outputs of the second step are the estimates of the MFPLM obtained with the PVS algorithm. For further details on this algorithm, see Aneiros and Vieu (2015).

Remark: If the condition $p_n = w_n q_n$ fails, the function considers not fixed $q_n = q_{n,k}$ values $k = 1, \ldots, w_n$, when p_n/w_n is not an integer number. Specifically:

$$q_{n,k} = \begin{cases} [p_n/w_n] + 1 & k \in \{1, \ldots, p_n - w_n[p_n/w_n]\}, \\ [p_n/w_n] & k \in \{p_n - w_n[p_n/w_n] + 1, \ldots, w_n\}, \end{cases}$$

where $[z]$ denotes the integer part of $z \in \mathbb{R}$.

Value

`call`	The matched call.
`fitted.values`	Estimated scalar response.
`residuals`	Differences between `y` and the `fitted.values`
`beta.est`	$\hat{\boldsymbol{\beta}}$ (i. e. estimate of $\boldsymbol{\beta}_0$ when the optimal tuning parameters `w.opt`, `lambda.opt`, `vn.opt` and `h.opt` are used).
`indexes.beta.nonnull`	Indexes of the non-zero $\hat{\beta}_j$.
`h.opt`	Selected bandwidth (when `w.opt` is considered).
`w.opt`	Selected size for $\mathcal{R}_n^{\mathbf{1}}$.
`lambda.opt`	Selected value of the penalisation parameter λ (when `w.opt` is considered).
`IC`	Value of the criterion function considered to select `w.opt`, `lambda.opt`, `vn.opt` and `h.opt`.
`vn.opt`	Selected value of `vn` in the second step (when `w.opt` is considered).
`beta2`	Estimate of $\boldsymbol{\beta}_0^{\mathbf{2}}$ for each value of the sequence `wn`.
`indexes.beta.nonnull2`	Indexes of the non-zero linear coefficients after the step 2 of the method for each value of the sequence `wn`.
`h2`	Selected bandwidth in the second step of the algorithm for each value of the sequence `wn`.
`IC2`	Optimal value of the criterion function in the second step for each value of the sequence `wn`.
`lambda2`	Selected value of penalisation parameter in the second step for each value of the sequence `wn`.
`index02`	Indexes of the covariates (in the whole set of p_n) used to build $\mathcal{R}_n^{\mathbf{2}}$ for each value of the sequence `wn`.
`beta1`	Estimate of $\boldsymbol{\beta}_0^{\mathbf{1}}$ for each value of the sequence `wn`.
`h1`	Selected bandwidth in the first step of the algorithm for each value of the sequence `wn`.
`IC1`	Optimal value of the criterion function in the first step for each value of the sequence `wn`.

`lambda1` Selected value of penalisation parameter in the first step for each value of the sequence `wn`.

`index01` Indexes of the covariates (in the whole set of p_n) used to build $\mathcal{R}_n^{\mathbf{1}}$ for each value of the sequence `wn`.

`index1` Indexes of the non-zero linear coefficients after the step 1 of the method for each value of the sequence `wn`.

`...`

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

References

Aneiros, G., and Vieu, P. (2015) Partial linear modelling with multi-functional covariates. *Computational Statistics*, **30**, 647–671, doi:10.1007/s0018001505688.

See Also

See also `sfpl.kernel.fit`, `predict.PVS.kernel` and `plot.PVS.kernel`.

Alternative method `PVS.kNN.fit`.

Examples

```
data(Sugar)


y<-Sugar$ash
x<-Sugar$wave.290
z<-Sugar$wave.240

#Outliers
index.y.25 <- y > 25
index.atip <- index.y.25
(1:268)[index.atip]


#Dataset to model
x.sug <- x[!index.atip,]
z.sug<- z[!index.atip,]
y.sug <- y[!index.atip]

train<-1:216

ptm=proc.time()
fit<- PVS.kernel.fit(x=x.sug[train,],z=z.sug[train,], y=y.sug[train],
        train.1=1:108,train.2=109:216,lambda.min.h=0.03,
        lambda.min.l=0.03,  max.q.h=0.35, num.h = 10, nknot=20,
```

```
        criterion="BIC", penalty="grSCAD", max.iter=5000)
proc.time()-ptm

fit
names(fit)
```

PVS.kNN.fit *PVS with kNN estimation*

Description

This function computes the partitioning variable selection algorithm (PVS) for sparse multi-functional partial linear regression.

This algorithm involves the penalised least-squares regularization procedure combined with k-nearest neighbours (kNN) estimation with Nadaraya-Watson weights. The procedure requires an objective criterion (`criterion`) to select the number of covariates in the reduced model (`w.opt`), the bandwidth (`k.opt`) and the penalisation parameter (`lambda.opt`).

Usage

```
PVS.kNN.fit(x, z, y, train.1=NULL, train.2=NULL, semimetric = "deriv", q = NULL,
  knearest = NULL, min.knn = 2, max.knn = NULL, step = NULL,
  range.grid = NULL, kind.of.kernel = "quad", nknot = NULL, lambda.min = NULL,
  lambda.min.h = NULL,lambda.min.l = NULL, factor.pn = 1,
  nlambda = 100, vn = ncol(z), nfolds = 10, seed = 123, wn = c(10, 15, 20),
  criterion = c("GCV", "BIC", "AIC", "k-fold-CV"),
  penalty = c("grLasso", "grMCP", "grSCAD", "gel", "cMCP", "gBridge",
  "gLasso", "gMCP"), max.iter = 1000)
```

Arguments

`x`	Matrix containing the observations of the functional covariate collected by row (functional nonparametric component).
`z`	Matrix containing the observations of the functional covariate that is discretised collected by row (linear component).
`y`	Vector containing the scalar response.
`train.1`	Indexes of the data used as the training sample in the 1st step. The default is `train.1<-1:ceiling(n/2)`.
`train.2`	Indexes of the data used as the training sample in the 2nd step. The default is `train.2<-(ceiling(n/2)+1):n`.
`semimetric`	Semi-metric function. Only `"deriv"` and `"pca"` are implemented. By default `semimetric="deriv"`.
`q`	Order of the derivative (if `semimetric="deriv"`) or number of principal components (if `semimetric="pca"`). The default values are 0 and 2, respectively.

knearest	Sequence of eligible values for k considered to seek for k.opt. If knearest=NULL, then knearest <- seq(from =min.knn, to = max.knn, by = step).
min.knn	Positive integer indicating the minumum value of the sequence in which the number of nearest neighbours k.opt is selected (thus, this number must be smaller than the sample size). The default is 2.
max.knn	Positive integer indicating the maximum value of the sequence in which the number of nearest neighbours k.opt is selected (thus, this number must be larger than min.kNN and smaller than the sample size). The default is max.knn <- n%/%2, being $n = n_1$ in the first step and $n = n_2$ in the second step of the method (see section Details).
step	Positive integer used to build the sequence of k-nearest neighbours in the following way: min.knn, min.knn + step, min.knn + 2*step, min.knn + 3*step, ... The default is step<-ceiling(n/100), being $n = n_1$ in the 1st step and $n = n_2$ in the 2nd step of the method.
range.grid	Vector of length 2 containing the endpoints of the grid at which the observations of the functional covariate x are evaluated (i.e. the range of the discretization). If range.grid=NULL, then range.grid=c(1,p) is considered, where p is the size of the discretization size of x (i.e. ncol(x)).
kind.of.kernel	The type of kernel function used. Only Epanechnikov kernel ("quad") is available.
nknot	Positive integer indicating the number of interior knots for the B-spline representation of the functional covariate. The default value is (p - order.Bspline - 1)%/%2.
lambda.min	The smallest value for lambda (i. e., the smallest value of the sequence in which lambda.opt is selected), as fraction of lambda.max. The defaults is lambda.min.l if the number of observations is larger than factor.pn times the number of covariates and lambda.min.h otherwise.
lambda.min.h	The smallest value of the sequence in which lambda.opt is selected if the number of observations is smaller than factor.pn times the number of scalar covariates. The default is 0.05.
lambda.min.l	The smallest value of the sequence in which lambda.opt is selected if the number of observations is larger than factor.pn times the number of scalar covariates. The default is 0.0001.
factor.pn	Positive integer used to set lambda.min. The default value is 1.
nlambda	Positive integer indicating the number of values of the sequence in which lambda.opt is selected. The default is 100.
vn	Positive integer or vector of positive integers indicating the number of groups of consecutive variables to be penalised together. The default value is vn=ncol(z), which leads to the individual penalisation of each scalar covariate.
nfolds	Positive integer indicating the number of cross-validation folds (used if criterion="k-fold-CV"). The default is 10.
seed	You may set the seed of the random number generator to obtain reproducible results (used if criterion="k-fold-CV"). Default is 123.
wn	A vector of positive integers indicating the eligible number of covariates of the reduced model. See the section Details. The default is c(10,15,20).

`criterion`	The criterion by which to select the regularization parameter `lambda.opt` and `k.opt`. One of `"GCV"`, `"BIC"`, `"AIC"` or `"k-fold-CV"`. The default is `"GCV"`.
`penalty`	The penalty function to be applied in the penalized least squares procedure. Only "grLasso" and "grSCAD" are implemented.
`max.iter`	Maximum number of iterations (total across entire path). The default is 1000.

Details

The multi-functional partial linear model (MFPLM) is given by the expression

$$Y_i = \sum_{j=1}^{p_n} \beta_{0j}\zeta_i(t_j) + m\left(X_i\right) + \varepsilon_i, \quad (i = 1, \ldots, n)$$

where

- Y_i is a real random response and X_i denotes a random element belonging to some semi-metric space $\mathcal{H}$. The second functional predictor ζ_i is supposed to be a random curve defined on some interval $[a, b]$ which is observed at the points $a \leq t_1 < \cdots < t_{p_n} \leq b$.
- $\boldsymbol{\beta}_0 = (\beta_{01}, \ldots, \beta_{0p_n})^\top$ is a vector of unknown real coefficients and $m(\cdot)$ denotes a smooth unknown real-valued link function.
- ε_i denotes the random error.

In the MFPLM, we assume that only a few scalar variables from the set $\{\zeta(t_1), \ldots, \zeta(t_{p_n})\}$ form part of the model. Therefore, we must select the relevant variables in the linear component (the impact points of the curve ζ on the response) and estimate the model.

In this function, the MFPLM is fitted using the PVS. The PVS is an algorithm with two steps, so we split the sample into two independent subsamples (asymptotically of the same size $n_1 \sim n_2 \sim n/2$), one of them to be used in the first stage of the method and the other in the second stage.

$$\mathcal{E}^{\mathbf{1}} = \{(\zeta_i, \mathcal{X}_i, Y_i), \quad i = 1, \ldots, n_1\},$$

$$\mathcal{E}^{\mathbf{2}} = \{(\zeta_i, \mathcal{X}_i, Y_i), \quad i = n_1 + 1, \ldots, n_1 + n_2 = n\}.$$

Note that these two subsamples are specified to the programme by means of the arguments `train.1` and `train.2`. The superscript $\boldsymbol{s}$ with $\boldsymbol{s} = \mathbf{1}, \mathbf{2}$ indicates the stage of the method in which the sample, function, variable or parameter is involved.

To explain the algorithm we assume, without lost of generality, that the number p_n of linear covariates can be expressed as follows: $p_n = q_n w_n$ with q_n and w_n integers.

1. **First step**. A reduced model is consider, discarding many linear covariates. The penalised least-squares procedure is applied to the reduced model using only the subsample $\mathcal{E}^{\mathbf{1}}$. Specifically:
   - Consider a subset of the initial p_n linear covariates, which contains only w_n equally spaced discretized observations of ζ covering the whole interval $[a, b]$. This subset is the following:
   $$\mathcal{R}_n^{\mathbf{1}} = \left\{\zeta\left(t_k^{\mathbf{1}}\right), \ k = 1, \ldots, w_n\right\},$$
   where $t_k^{\mathbf{1}} = t_{[(2k-1)q_n/2]}$ and $[z]$ denotes the smallest integer not less than $z \in \mathbb{R}$.The size (cardinal) of this subset is provided to the program in the argument `wn` (which contains a sequence of eligible sizes).

- Consider the following reduced model, which involves only the w_n linear covariates belonging to $\mathcal{R}_n^{\mathbf{1}}$:

$$Y_i = \sum_{k=1}^{w_n} \beta_{0k}^{\mathbf{1}} \zeta_i(t_k^{\mathbf{1}}) + m^{\mathbf{1}}\left(X_i\right) + \varepsilon_i^{\mathbf{1}}.$$

  The penalised least-squares variable selection procedure, with kNN estimation, is applied to the reduced model. This is done by means of the function `sfpl.kNN.fit`, which requires the remaining arguments (for details, see the documentation of the function `sfpl.kNN.fit`). The estimates obtained after that are the outputs of the first step of the algorithm.

2. **Second step**. The variables selected in the first step and the variables in the neighbourhood of the ones selected are included. Then the penalised least-squares procedure, combined with kernel estimation, is carried out again. For that, we consider only the subsample $\mathcal{E}^{\mathbf{2}}$. Specifically:

   - Consider a new set of variables :

$$\mathcal{R}_n^{\mathbf{2}} = \bigcup_{\left\{k, \widehat{\beta}_{0k}^{\mathbf{1}} \neq 0\right\}} \left\{\zeta(t_{(k-1)q_n+1}), \ldots, \zeta(t_{kq_n})\right\}.$$

     Denoting by $r_n = \sharp(\mathcal{R}_n^{\mathbf{2}})$, we can rename the variables in $\mathcal{R}_n^{\mathbf{2}}$ as follows:

$$\mathcal{R}_n^{\mathbf{2}} = \left\{\zeta(t_1^{\mathbf{2}}), \ldots, \zeta(t_{r_n}^{\mathbf{2}})\right\},$$

   - Consider the following model, which involves only the linear covariates belonging to $\mathcal{R}_n^{\mathbf{2}}$

$$Y_i = \sum_{k=1}^{r_n} \beta_{0k}^{\mathbf{2}} \zeta_i(t_k^{\mathbf{2}}) + m^{\mathbf{2}}\left(X_i\right) + \varepsilon_i^{\mathbf{2}}.$$

     The penalized least-squares variable selection procedure, with kNN estimation, is applied to this model by means of the function `sfpl.kNN.fit`.

The outputs of the second step are the estimates of the MFPLM obtained with the PVS algorithm. For further details on this algorithm, see Aneiros and Vieu (2015).

Remark: If the condition $p_n = w_n q_n$ fails, the function considers not fixed $q_n = q_{n,k}$ values $k = 1, \ldots, w_n$, when p_n/w_n is not an integer number. Specifically:

$$q_{n,k} = \begin{cases} [p_n/w_n] + 1 & k \in \{1, \ldots, p_n - w_n[p_n/w_n]\}, \\ [p_n/w_n] & k \in \{p_n - w_n[p_n/w_n] + 1, \ldots, w_n\}, \end{cases}$$

where $[z]$ denotes the integer part of $z \in \mathbb{R}$.

Value

`call`	The matched call.
`fitted.values`	Estimated scalar response.
`residuals`	Differences between `y` and the `fitted.values`
`beta.est`	$\hat{\boldsymbol{\beta}}$ (i.e. estimate of $\boldsymbol{\beta}_0$ when the optimal tuning parameters `w.opt`, `lambda.opt`, `vn.opt` and `k.opt` are used).

`indexes.beta.nonnull`
: Indexes of the non-zero $\hat{\beta}_j$.

`k.opt`
: Selected number of nearest neighbours (when `w.opt` is considered).

`w.opt`
: Selected initial number of covariates in the reduced model.

`lambda.opt`
: Selected value of the penalisation parameter λ (when `w.opt` is considered).

`IC`
: Value of the criterion function considered to select `w.opt`, `lambda.opt`, `vn.opt` and `k.opt`.

`vn.opt`
: Selected value of `vn` in the second step (when `w.opt` is considered).

`beta2`
: Estimate of $\boldsymbol{\beta_0^2}$ for each value of the sequence `wn`.

`indexes.beta.nonnull2`
: Indexes of the non-zero linear coefficients after the step 2 of the method for each value of the sequence `wn`.

`knn2`
: Selected number of neighbours in the second step of the algorithm for each value of the sequence `wn`.

`IC2`
: Optimal value of the criterion function in the second step for each value of the sequence `wn`.

`lambda2`
: Selected value of penalisation parameter in the second step for each value of the sequence `wn`.

`index02`
: Indexes of the covariates (in the whole set of p_n) used to build $\mathcal{R}_n^{\mathbf{2}}$ for each value of the sequence `wn`.

`beta1`
: Estimate of $\boldsymbol{\beta_0^1}$ for each value of the sequence `wn`.

`knn1`
: Selected number of neighbours in the first step of the algorithm for each value of the sequence `wn`.

`IC1`
: Optimal value of the criterion function in the first step for each value of the sequence `wn`.

`lambda1`
: Selected value of penalisation parameter in the first step for each value of the sequence `wn`.

`index01`
: Indexes of the covariates (in the whole set of p_n) used to build $\mathcal{R}_n^{\mathbf{1}}$ for each value of the sequence `wn`.

`index1`
: Indexes of the non-zero linear coefficients after the step 1 of the method for each value of the sequence `wn`.

`...`

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

References

Aneiros, G., and Vieu, P. (2015) Partial linear modelling with multi-functional covariates. *Computational Statistics*, **30**, 647–671, doi:10.1007/s0018001505688.

See Also

See also sfpl.kNN.fit, predict.PVS.kNN and plot.PVS.kNN.

Alternative method PVS.kernel.fit.

Examples

```
data(Sugar)


y<-Sugar$ash
x<-Sugar$wave.290
z<-Sugar$wave.240

#Outliers
index.y.25 <- y > 25
index.atip <- index.y.25
(1:268)[index.atip]


#Dataset to model
x.sug <- x[!index.atip,]
z.sug<- z[!index.atip,]
y.sug <- y[!index.atip]

train<-1:216

ptm=proc.time()
fit<- PVS.kNN.fit(x=x.sug[train,],z=z.sug[train,], y=y.sug[train],
        train.1=1:108,train.2=109:216,lambda.min.h=0.07,
        lambda.min.l=0.07, nknot=20,criterion="BIC", penalty="grSCAD",
        max.iter=5000)
proc.time()-ptm

fit
names(fit)
```

---

semimetric.projec *Projection semi-metric computation*

Description

Computes the projection semi-metric in a direction θ between each curve in data1 and each curve in data2.

Usage

```
semimetric.projec(data1, data2, theta, order.Bspline = 3, nknot.theta = 3,
  range.grid = NULL, nknot = NULL)
```

Arguments

data1	Matrix containing functional data collected by row.
data2	Matrix containing functional data collected by row.
theta	Vector containing the coefficients of θ in a B-spline basis, so that length(theta)=order.Bspline+nknot
order.Bspline	Positive integer giving the order of the B-spline basis functions for the B-spline representation of θ. This is the number of coefficients in each piecewise polynomial segment. The default is 3.
nknot.theta	Positive integer indicating the number of uniform interior knots of the B-spline basis. The default is 3.
range.grid	Vector of length 2 containing the range of the discretization of the functional data. If range.grid=NULL, then range.grid=c(1,p) is considered, where p is the size of the discretization size of data (i.e. ncol(data)).
nknot	Positive integer indicating the number of interior knots for the B-spline representation of the functional data. The default value is (p - order.Bspline - 1)%/%2.

Details

For $x_1, x_2 \in \mathcal{H}$, being $\mathcal{H}$ a separable Hilbert space, the projection semi-metric in the direction $\theta \in \mathcal{H}$ is defined as

$$d_\theta(x_1, x_2) = |\langle \theta, x_1 - x_2 \rangle|.$$

The function semimetric.projec computes the projection semi-metric using the B-spline representation of the curves and θ. The dimension of the B-spline basis for θ is order.Bspline+nknot.theta.

Value

A matrix with the projection semi-semimetrics of each pair of curves.

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

References

Novo S., Aneiros, G., and Vieu, P., (2019) Automatic and location-adaptive estimation in functional single–index regression. *Journal of Nonparametric Statistics*, **31(2)**, 364–392, doi:10.1080/10485252.2019.1567726.

See Also

See also projec.

Examples

```
data("Tecator")
names(Tecator)
y<-Tecator$fat
X<-Tecator$absor.spectra

#length(theta)=6=order.Bspline+nknot.theta
semimetric.projec(data1=X[1:5,], data2=X[5:10,],theta=c(1,0,0,1,1,-1),
  nknot.theta=3,nknot=20,range.grid=c(850,1050))
```

---

sfpl.kernel.fit *Sparse semi-functional partial linear model fit using kernel estimation*

Description

This function fits a sparse semi-functional partial linear model between a scalar response, a functional explanatory variable and a vector of scalar covariates. The function uses the penalised least-squares regularization procedure combined with nonparametric kernel estimation with Nadaraya-Watson weights.

The procedure requires an objective criterion (`criterion`) to select the bandwidth (`h.opt`) and the regularization parameter (`lambda.opt`).

Usage

```
sfpl.kernel.fit(x, z, y, semimetric = "deriv", q = NULL, min.q.h = 0.05,
  max.q.h = 0.5, h.seq = NULL,num.h = 10,range.grid = NULL,
  kind.of.kernel = "quad", nknot = NULL, lambda.min = NULL,
  lambda.min.h = NULL, lambda.min.l = NULL, factor.pn = NULL,
  nlambda = 100, lambda.seq = NULL, vn = ncol(z), nfolds = 10, seed = 123,
  criterion = c("GCV", "BIC", "AIC", "k-fold-CV"),
  penalty = c("grLasso", "grMCP", "grSCAD", "gel", "cMCP", "gBridge", "gLasso",
  "gMCP"),max.iter = 1000)
```

Arguments

`x`	Matrix containing the observations of the functional covariate collected by row.
`z`	Matrix containing the observations of the scalar covariates collected by row.
`y`	Vector containing the scalar response.
`semimetric`	Semi-metric function. Only `"deriv"` and `"pca"` are implemented. By default `semimetric="deriv"`.
`q`	Order of the derivative (if `semimetric="deriv"`) or number of principal components (if `semimetric="pca"`). The default values are 0 and 2, respectively.

`min.q.h` Order of the quantile of the set of distances between curves (computed with the provided `semimetric`) which gives the lower end of the sequence in which the bandwidth is selected. The default is 0.05.

`max.q.h` Order of the quantile of the set of distances between curves (computed with the provided `semimetric`) which gives the upper end of the sequence in which the bandwidth is selected. The default is 0.5.

`h.seq` Vector containing the sequence of bandwidths. The default is a sequence of `num.h` equispaced bandwidths in the range constructed using `min.q.h` and `max.q.h`.

`num.h` Positive integer indicating the number of bandwiths in the grid. The default is 10.

`range.grid` Vector of length 2 containing the endpoints of the grid at which the observations of the functional covariate `x` are evaluated (i.e. the range of the discretization). If `range.grid=NULL`, then `range.grid=c(1,p)` is considered, where `p` is the size of the discretization size of `x` (i.e. `ncol(x)`).

`kind.of.kernel` The type of kernel function used. Only Epanechnikov kernel (`"quad"`) is available.

`nknot` Positive integer indicating the number of interior knots for the B-spline representation of the functional covariate. The default value is `(p - order.Bspline - 1)%/%2`.

`lambda.min` The smallest value for lambda (i. e., the smallest value of the sequence in which `lambda.opt` is selected), as fraction of `lambda.max`. The defaults is `lambda.min.l` if the number of observations is larger than `factor.pn` times the number of covariates and `lambda.min.h` otherwise.

`lambda.min.h` The smallest value of the sequence in which `lambda.opt` is selected if the number of observations is smaller than `factor.pn` times the number of scalar covariates. The default is 0.05.

`lambda.min.l` The smallest value of the sequence in which `lambda.opt` is selected if the number of observations is larger than `factor.pn` times the number of scalar covariates. The default is 0.0001.

`factor.pn` Positive integer used to set `lambda.min`. The default is 1.

`nlambda` Positive integer indicating the number of values of the sequence in which `lambda.opt` is selected. The default is 100.

`lambda.seq` Sequence of values in which `lambda.opt` is selected. If `lambda.seq=NULL`, then the programme builds the sequence automatically using `lambda.min` and `nlambda`.

`vn` Positive integer or vector of positive integers indicating the number of groups of consecutive variables to be penalised together. The default value is `vn=ncol(z)`, which leads to the individual penalisation of each scalar covariate.

`nfolds` Positive integer indicating the number of cross-validation folds (used if `criterion="k-fold-CV"`). The default is 10.

`seed` You may set the seed of the random number generator to obtain reproducible results (used if `criterion="k-fold-CV"`). The default is 123.

`criterion` The criterion by which to select the regularization parameter `lambda.opt` and the bandwidth `h.opt`. One of `"GCV"`, `"BIC"`, `"AIC"` or `"k-fold-CV"`. The default is `"GCV"`.

`penalty`	The penalty function to be applied in the penalized least squares procedure. Only "grLasso" and "grSCAD" are implemented.
`max.iter`	Maximum number of iterations (total across entire path). The default is 1000.

Details

The sparse semi-functional partial linear model (SSFPLM) is given by the expression:

$$Y_i = Z_{i1}\beta_{01} + \cdots + Z_{ip_n}\beta_{0p_n} + m(X_i) + \varepsilon_i \quad i = 1, \ldots, n,$$

where Y_i denotes a scalar response, $Z_{i1}, \ldots, Z_{ip_n}$ are random covariates taking values in $\mathbb{R}$ and X_i is a functional random covariate valued in some semi-metric space $\mathcal{H}$. In this equation, $\boldsymbol{\beta}_0 = (\beta_{01}, \ldots, \beta_{0p_n})^\top \in \mathbb{R}^{p_n}$ and $m(\cdot)$ are a vector of unknown real parameters and an unknown smooth real-valued function, respectively. In addition, ε_i is the random error.

In this function the SSFPLM is fitted using the penalised least-squares approach. The first idea is to transform the SSFPLM into a linear model by extracting from Y_i and Z_{ij} $(j = 1, \ldots, p_n)$ the effect of the functional covariate X_i using functional nonparametric regression (see, for details, Ferraty and Vieu, 2006). This is made using kernel estimation with Nadaraya-Watson weights.

Then, an approximate linear model is obtained:

$$\widetilde{\boldsymbol{Y}} \approx \widetilde{\boldsymbol{Z}}\boldsymbol{\beta}_0 + \boldsymbol{\varepsilon},$$

and the penalised least-squares procedure is applied to this model by minimising

$$\mathcal{Q}(\boldsymbol{\beta}) = \frac{1}{2}\left(\widetilde{\boldsymbol{Y}} - \widetilde{\boldsymbol{Z}}\boldsymbol{\beta}\right)^\top \left(\widetilde{\boldsymbol{Y}} - \widetilde{\boldsymbol{Z}}\boldsymbol{\beta}\right) + n\sum_{j=1}^{p_n} \mathcal{P}_{\lambda_{j_n}}(|\beta_j|), \quad (1)$$

where $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_{p_n})^\top$, $\mathcal{P}_{\lambda_{j_n}}(\cdot)$ is a penalty function (specified in the argument `penalty`) and $\lambda_{j_n} > 0$ is a tuning parameter. To reduce the quantity of tuning parameters, λ_j, to be selected for each sample, we consider $\lambda_j = \lambda\widehat{\sigma}_{\beta_{0,j,OLS}}$, where $\beta_{0,j,OLS}$ denotes the OLS estimate of $\beta_{0,j}$ and $\widehat{\sigma}_{\beta_{0,j,OLS}}$ is the estimated standard deviation. Both λ and h (in the kernel estimation) are selected using the objetive criterion specified in the argument `criterion`.

Finally, after estimating $\boldsymbol{\beta}_0$ by minimising (1), we deal with the estimation of the nonlinear function $m(\cdot)$. For that, we employ again the kernel procedure with Nadaraya-Watson weights to smooth the partial residuals $Y_i - \boldsymbol{Z}_i^\top\widehat{\boldsymbol{\beta}}$.

For further details on the estimation procedure of the SSFPLM, see Aneiros et al. (2015).

Remark: We should note that if we set `lambda.seq`$= 0$, we can obtain the non-penalised estimation of the model, i.e. the OLS estimation. It is convenient to use `lambda.seq`$\neq 0$ when one suspects there are irrelevant variables.

Value

`call`	The matched call.
`fitted.values`	Estimated scalar response.
`residuals`	Differences between `y` and the `fitted.values`
`beta.est`	Estimate of β_0 when the optimal tuning parameters `lambda.opt`, `h.opt` and `vn.opt` are used.

`indexes.beta.nonnull`
: Indexes of the non-zero $\hat{\beta}_j$.

`h.opt`
: Selected bandwidth.

`lambda.opt`
: Selected value of lambda.

`IC`
: Value of the criterion function considered to select `lambda.opt`, `h.opt` and `vn.opt`.

`h.min.opt.max.mopt`
: `h.opt=h.min.opt.max.mopt[2]` (used by `beta.est`) was seeked between `h.min.opt.max.mopt[1]` and `h.min.opt.max.mopt[3]`.

`vn.opt`
: Selected value of `vn`.

`...`

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

References

Aneiros, G., Ferraty, F., Vieu, P. (2015) Variable selection in partial linear regression with functional covariate. *Statistics*, **49**, 1322–1347, doi:10.1080/02331888.2014.998675.

Ferraty, F. and Vieu, P. (2006) *Nonparametric Functional Data Analysis*. Springer Series in Statistics, New York.

See Also

See also `predict.sfpl.kernel` and `plot.sfpl.kernel`.

Alternative method `sfpl.kNN.fit`.

Examples

```
data("Tecator")
y<-Tecator$fat
X<-Tecator$absor.spectra
z1<-Tecator$protein
z2<-Tecator$moisture

#Quadratic, cubic and interaction effects of the scalar covariates.
z.com<-cbind(z1,z2,z1^2,z2^2,z1^3,z2^3,z1*z2)
train<-1:160


#SFPLM fit.
ptm=proc.time()
fit<-sfpl.kernel.fit(x=X[train,], z=z.com[train,], y=y[train],q=2,
      max.q.h=0.35,lambda.min.h=0.02,lambda.min.l=0.01,
      factor.pn=2, max.iter=5000, criterion="BIC", penalty="grSCAD",nknot=20)
proc.time()-ptm
```

```
#Results
fit
names(fit)
```

`sfpl.kNN.fit` *Sparse semi-functional partial linear model fit using kNN estimation*

Description

This function fits a sparse semi-functional partial linear model between a scalar response, a functional explanatory variable and a vector of scalar covariates. The function uses the penalised least-squares regularization procedure combined with k-nearest neighbours (kNN) estimation with Nadaraya-Watson weights.

The procedure requires an objective criterion (`criterion`) to select the number of nearest neighbours (`k.opt`) and the regularization parameter (`lambda.opt`).

Usage

```
sfpl.kNN.fit(x, z, y, semimetric = "deriv", q = NULL, knearest = NULL, min.knn = 2,
  max.knn = NULL, step = NULL,range.grid = NULL, kind.of.kernel = "quad",
  nknot = NULL, lambda.min = NULL, lambda.min.h = NULL,
  lambda.min.l = NULL, factor.pn = 1, nlambda = 100, lambda.seq = NULL,
  vn = ncol(z), nfolds = 10, seed = 123,criterion = c("GCV", "BIC",
  "AIC", "k-fold-CV"),penalty = c("grLasso", "grMCP",
  "grSCAD", "gel", "cMCP", "gBridge", "gLasso", "gMCP"),
  max.iter = 1000)
```

Arguments

`x`	Matrix containing the observations of the functional covariate collected by row.
`z`	Matrix containing the observations of the scalar covariates collected by row.
`y`	Vector containing the scalar response.
`semimetric`	Semi-metric function. Only `"deriv"` and `"pca"` are implemented. By default `semimetric="deriv"`.
`q`	Order of the derivative (if `semimetric="deriv"`) or number of principal components (if `semimetric="pca"`). The default values are 0 and 2, respectively.
`knearest`	Sequence of eligible values for k considered to seek for `k.opt`. If `knearest=NULL`, then `knearest <- seq(from = min.knn, to = max.knn, by = step)`.
`min.knn`	Positive integer indicating the minumum value of the sequence in which the number of nearest neighbours `k.opt` is selected (thus, this number must be smaller than the sample size). The default is 2.
`max.knn`	Positive integer indicating the maximum value of the sequence in which the number of nearest neighbours `k.opt` is selected (thus, this number must be larger than `min.kNN` and smaller than the sample size, `n`). The default is `max.knn <- n%/%2`.

step	Positive integer used to build the sequence of k-nearest neighbours in the following way: min.knn, min.knn + step, min.knn + 2*step, min.knn + 3*step, ... The default is step<-ceiling(n/100).
range.grid	Vector of length 2 containing the endpoints of the grid at which the observations of the functional covariate x are evaluated (i.e. the range of the discretization). If range.grid=NULL, then range.grid=c(1,p) is considered, where p is the size of the discretization size of x (i.e. ncol(x)).
kind.of.kernel	The type of kernel function used. Only Epanechnikov kernel ("quad") is available.
nknot	Positive integer indicating the number of interior knots for the B-spline representation of the functional covariate. The default value is (p - order.Bspline - 1)%/%2.
lambda.min	The smallest value for lambda (i. e., the smallest value of the sequence in which lambda.opt is selected), as fraction of lambda.max. The defaults is lambda.min.l if the number of observations is larger than factor.pn times the number of covariates and lambda.min.h otherwise.
lambda.min.h	The smallest value of the sequence in which lambda.opt is selected if the number of observations is smaller than factor.pn times the number of scalar covariates. The default is 0.05.
lambda.min.l	The smallest value of the sequence in which lambda.opt is selected if the number of observations is larger than factor.pn times the number of scalar covariates. The default is 0.0001.
factor.pn	Positive integer used to set lambda.min. The default value is 1.
nlambda	Positive integer indicating the number of values of the sequence in which lambda.opt is selected. The default is 100.
lambda.seq	Sequence of values in which lambda.opt is selected. If lambda.seq=NULL, then the programme builds the sequence automatically using lambda.min and nlambda.
vn	Positive integer or vector of positive integers indicating the number of groups of consecutive variables to be penalised together. The default value is vn=ncol(z), which leads to the individual penalisation of each scalar covariate.
nfolds	Positive integer indicating the number of cross-validation folds (used if criterion="k-fold-CV"). The default is 10.
seed	You may set the seed of the random number generator to obtain reproducible results (used if criterion="k-fold-CV"). The default is 123.
criterion	The criterion by which to select the regularization parameter lambda.opt and k.opt. One of "GCV", "BIC", "AIC" or "k-fold-CV". The default is "GCV".
penalty	The penalty function to be applied in the penalized least squares procedure. Only "grLasso" and "grSCAD" are implemented.
max.iter	Maximum number of iterations (total across entire path). The default is 1000.

Details

The sparse semi-functional partial linear model (SSFPLM) is given by the expression:

$$Y_i = Z_{i1}\beta_{01} + \cdots + Z_{ip_n}\beta_{0p_n} + m(X_i) + \varepsilon_i \quad i = 1, \ldots, n,$$

where Y_i denotes a scalar response, $Z_{i1}, \ldots, Z_{ip_n}$ are random covariates taking values in $\mathbb{R}$ and X_i is a functional random covariate valued in some semi-metric space $\mathcal{H}$. In this equation, $\boldsymbol{\beta}_0 = (\beta_{01}, \ldots, \beta_{0p_n})^\top \in \mathbb{R}^{p_n}$ and $m(\cdot)$ are a vector of unknown real parameters and an unknown smooth real-valued function, respectively. In addition, ε_i is the random error.

In this function the SSFPLM is fitted using the penalised least-squares approach. The first idea is to transform the SSFPLM into a linear model by extracting from Y_i and Z_{ij} $(j = 1, \ldots, p_n)$ the effect of the functional covariate X_i using functional nonparametric regression (see, for details, Ferraty and Vieu, 2006). This is made using kNN estimation with Nadaraya-Watson weights.

Then, an approximate linear model is obtained:

$$\widetilde{\boldsymbol{Y}} \approx \widetilde{\boldsymbol{Z}}\boldsymbol{\beta}_0 + \boldsymbol{\varepsilon},$$

and the penalised least-squares procedure is applied to this model by minimising

$$\mathcal{Q}(\boldsymbol{\beta}) = \frac{1}{2}\left(\widetilde{\boldsymbol{Y}} - \widetilde{\boldsymbol{Z}}\boldsymbol{\beta}\right)^\top \left(\widetilde{\boldsymbol{Y}} - \widetilde{\boldsymbol{Z}}\boldsymbol{\beta}\right) + n\sum_{j=1}^{p_n} \mathcal{P}_{\lambda_{j_n}}(|\beta_j|), \quad (1)$$

where $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_{p_n})^\top$, $\mathcal{P}_{\lambda_{j_n}}(\cdot)$ is a penalty function (specified in the argument `penalty`) and $\lambda_{j_n} > 0$ is a tuning parameter. To reduce the quantity of tuning parameters, λ_j, to be selected for each sample, we consider $\lambda_j = \lambda\widehat{\sigma}_{\beta_{0,j,OLS}}$, where $\beta_{0,j,OLS}$ denotes the OLS estimate of $\beta_{0,j}$ and $\widehat{\sigma}_{\beta_{0,j,OLS}}$ is the estimated standard deviation. Both λ and k (in the kNN estimation) are selected using the objetive criterion specified in the argument `criterion`.

Finally, after estimating $\boldsymbol{\beta}_0$ by minimising (1), we deal with the estimation of the nonlinear function $m(\cdot)$. For that, we employ again the kNN procedure with Nadaraya-Watson weights to smooth the partial residuals $Y_i - \boldsymbol{Z}_i^\top\widehat{\boldsymbol{\beta}}$.

For further details on the estimation procedure of the SSFPLM, see Aneiros et al. (2015).

Remark: We should note that if we set `lambda.seq`$= 0$, we can obtain the non-penalised estimation of the model, i.e. the OLS estimation. It is convenient to use `lambda.seq`$\neq 0$ when one suspects there are irrelevant variables.

Value

`call`	The matched call.
`fitted.values`	Estimated scalar response.
`residuals`	Differences between `y` and the `fitted.values`
`beta.est`	Estimate of β_0 when the optimal tuning parameters `lambda.opt`, `k.opt` and `vn.opt` are used.
`indexes.beta.nonnull`	Indexes of the non-zero $\hat{\beta}_j$.
`k.opt`	Selected number of nearest neighbours.
`lambda.opt`	Selected value of lambda.
`IC`	Value of the criterion function considered to select both `lambda.opt`, `h.opt` and `vn.opt`.
`vn.opt`	Selected value of `vn`.
`...`	

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

References

Aneiros, G., Ferraty, F., Vieu, P. (2015) Variable selection in partial linear regression with functional covariate. *Statistics*, **49**, 1322–1347, doi:10.1080/02331888.2014.998675.

See Also

See also `predict.sfpl.kNN` and `plot.sfpl.kNN`.

Alternative method `sfpl.kernel.fit`.

Examples

```
data("Tecator")
y<-Tecator$fat
X<-Tecator$absor.spectra
z1<-Tecator$protein
z2<-Tecator$moisture

#Quadratic, cubic and interaction effects of the scalar covariates.
z.com<-cbind(z1,z2,z1^2,z2^2,z1^3,z2^3,z1*z2)
train<-1:160


#SFPLM fit.
ptm=proc.time()
fit<-sfpl.kNN.fit(y=y[train],x=X[train,], z=z.com[train,],q=2, max.knn=20,
  lambda.min.h=0.02,lambda.min.l=0.01, factor.pn=2, criterion="BIC",
  range.grid=c(850,1050), penalty="grSCAD",nknot=20, max.iter=5000)
proc.time()-ptm

#Results
fit
names(fit)
```

`sfplsim.kernel.fit` *Sparse semi-functional partial linear single-index model fit using kernel estimation*

Description

This function fits a sparse semi-functional partial linear single-index model between a scalar response, a functional explanatory variable and a vector of scalar covariates. The function uses the penalised least-squares regularization procedure combined with nonparametric kernel estimation with Nadaraya-Watson weights.

The procedure requires the B-spline representation to estimate the functional index θ_0 and an objective criterion (`criterion`) to select the bandwidth (`h.opt`) and the regularization parameter (`lambda.opt`).

Usage

```
sfplsim.kernel.fit(x, z, y, seed.coeff = c(-1, 0, 1), order.Bspline = 3,
  nknot.theta = 3,t0 = NULL, min.q.h = 0.05, max.q.h = 0.5,
  h.seq = NULL, num.h = 10, range.grid = NULL, kind.of.kernel = "quad",
  nknot = NULL, lambda.min = NULL, lambda.min.h = NULL,
  lambda.min.l = NULL, factor.pn = 1, nlambda = 100, lambda.seq = NULL,
  vn = ncol(z), nfolds = 10, seed = 123, criterion = c("GCV", "BIC", "AIC",
  "k-fold-CV"), penalty = c("grLasso", "grMCP",
  "grSCAD", "gel", "cMCP", "gBridge", "gLasso", "gMCP"),
  max.iter = 1000)
```

Arguments

`x`	Matrix containing the observations of the functional covariate collected by row.
`z`	Matrix containing the observations of the scalar covariates collected by row.
`y`	Vector containing the scalar response.
`seed.coeff`	Vector of initial values used to build the set Θ_n (see section `Details`). The coefficients for the B-spline representation of each eligible functional index $\theta \in \Theta_n$ are obtained from `seed.coeff`. The default is `c(-1,0,1)`.
`order.Bspline`	Positive integer giving the order of the B-spline basis functions. This is the number of coefficients in each piecewise polynomial segment. The default is 3.
`nknot.theta`	Positive integer indicating the number of uniform interior knots of the B-spline basis for the B-spline representation of θ_0. The default is 3.
`t0`	Value in the domain of the functional indexes at which we evaluate them to build the set Θ_n. We assume $\theta_0(t_0) > 0$ for some arbitrary t_0 in the domain to ensure model identifiability. If `t0=NULL`, then `mean(range.grid)` is considered.
`min.q.h`	Order of the quantile of the set of distances between curves (computed with the projection semi-metric) which gives the lower end of the sequence in which the bandwidth is selected. The default is 0.05.
`max.q.h`	Order of the quantile of the set of distances between curves (computed with the projection semi-metric) which gives the upper end of the sequence in which the bandwidth is selected. The default is 0.5.
`h.seq`	Vector containing the sequence of bandwidths. The default is a sequence of `num.h` equispaced bandwidths in the range constructed using `min.q.h` and `max.q.h`.
`num.h`	Positive integer indicating the number of bandwiths in the grid. The default is 10.
`range.grid`	Vector of length 2 containing the endpoints of the grid at which the observations of the functional covariate `x` are evaluated (i.e. the range of the discretization). If `range.grid=NULL`, then `range.grid=c(1,p)` is considered, where `p` is the size of the discretization size of `x` (i.e. `ncol(x)`).

`kind.of.kernel` The type of kernel function used. Only Epanechnikov kernel (`"quad"`) is available.

`nknot` Positive integer indicating the number of interior knots for the B-spline representation of the functional covariate. The default value is `(p - order.Bspline - 1)%/%2`.

`lambda.min` The smallest value for lambda (i. e., the smallest value of the sequence in which `lambda.opt` is selected), as fraction of `lambda.max`. The default is `lambda.min.l` if the number of observations is larger than `factor.pn` times the number of covariates and `lambda.min.h` otherwise.

`lambda.min.h` The smallest value of the sequence in which `lambda.opt` is selected if the number of observations is smaller than `factor.pn` times the number of scalar covariates. The default is 0.05.

`lambda.min.l` The smallest value of the sequence in which `lambda.opt` is selected if the number of observations is larger than `factor.pn` times the number of scalar covariates. The default is 0.0001.

`factor.pn` Positive integer used to set `lambda.min`. The default value is 1.

`nlambda` Positive integer indicating the number of values of the sequence in which `lambda.opt` is selected. The default is 100.

`lambda.seq` Sequence of values in which `lambda.opt` is selected. If `lambda.seq=NULL`, then the programme builds the sequence automatically using `lambda.min` and `nlambda`. For non-penalized estimation, i. e. ordinary least squares estimation (OLS), set `lambda.seq=0`.

`vn` Positive integer or vector of positive integers indicating the number of groups of consecutive variables to be penalised together. The default value is `vn=ncol(z)`, which leads to the individual penalisation of each scalar covariate.

`nfolds` Positive integer indicating the number of cross-validation folds (used if `criterion="k-fold-CV"`). The default is 10.

`seed` You may set the seed of the random number generator to obtain reproducible results (used if `criterion="k-fold-CV"`). The default is 123.

`criterion` The criterion by which to select the regularization parameter `lambda.opt` and `h.opt`. One of `"GCV"`, `"BIC"`, `"AIC"` or `"k-fold-CV"`. The default is `"GCV"`.

`penalty` The penalty function to be applied in the penalized least squares procedure. Only "grLasso" and "grSCAD" are implemented.

`max.iter` Maximum number of iterations (total across entire path). The default is 1000.

Details

The sparse semi-functional partial linear single-index model (SSFPLSIM) is given by the expression:

$$Y_i = Z_{i1}\beta_{01} + \cdots + Z_{ip_n}\beta_{0p_n} + r(\langle\theta_0, X_i\rangle) + \varepsilon_i \quad i = 1, \ldots, n,$$

where Y_i denotes a scalar response, $Z_{i1}, \ldots, Z_{ip_n}$ are random covariates taking values in $\mathbb{R}$ and X_i is a functional random covariate valued in a separable Hilbert space $\mathcal{H}$ with inner product $\langle\cdot,\cdot\rangle$. In this equation, $\boldsymbol{\beta}_0 = (\beta_{01}, \ldots, \beta_{0p_n})^\top \in \mathbb{R}^{p_n}$, $\theta_0 \in \mathcal{H}$ and $r(\cdot)$ are a vector of unknown real parameters, an unknown functional direction and an unknown smooth real-valued function, respectively. In addition, ε_i is the random error.

The SSFPLSIM is fitted using the penalised least-squares approach. The first idea is to transform the SSFPLSIM into a linear model by extracting from Y_i and Z_{ij} $(j = 1, \ldots, p_n)$ the effect of the functional covariate X_i using functional single-index regression. This is made using nonparametric kernel estimation (see, for details, the documentation of the function `fsim.kernel.fit`).

Then, an approximate linear model is obtained:

$$\widetilde{\boldsymbol{Y}}_{\theta_0} \approx \widetilde{\boldsymbol{Z}}_{\theta_0}\boldsymbol{\beta}_0 + \boldsymbol{\varepsilon},$$

and the penalised least-squares procedure is applied to this model by minimising over the pair $(\boldsymbol{\beta}, \theta)$

$$\mathcal{Q}(\boldsymbol{\beta}, \theta) = \frac{1}{2}\left(\widetilde{\boldsymbol{Y}}_\theta - \widetilde{\boldsymbol{Z}}_\theta\boldsymbol{\beta}\right)^\top \left(\widetilde{\boldsymbol{Y}}_\theta - \widetilde{\boldsymbol{Z}}_\theta\boldsymbol{\beta}\right) + n\sum_{j=1}^{p_n} \mathcal{P}_{\lambda_{j_n}}(|\beta_j|), \quad (1)$$

where $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_{p_n})^\top$, $\mathcal{P}_{\lambda_{j_n}}(\cdot)$ is a penalty function (specified in the argument `penalty`) and $\lambda_{j_n} > 0$ is a tuning parameter. To reduce the quantity of tuning parameters, λ_j, to be selected for each sample, we consider $\lambda_j = \lambda\widehat{\sigma}_{\beta_{0,j,OLS}}$, where $\beta_{0,j,OLS}$ denotes the OLS estimate of $\beta_{0,j}$ and $\widehat{\sigma}_{\beta_{0,j,OLS}}$ is the estimated standard deviation. Both λ and h (in the kernel estimation) are selected using the objetive criterion specified in the argument `criterion`.

In addition, the function uses B-spline representation to build a set Θ_n of eligible functional indexes θ. The dimension of the B-spline basis is `order.Bspline+nknot.theta` and the set of eligible coefficients is obtained by calibrating (to ensure the identifiability of the model) the set of initial coefficients given in `seed.coeff`. The larger this set, the higher the size of Θ_n. Since our approach requires intensive computation, we need a trade-off between the size of Θ_n and the performance of the estimator. For that, Ait-Saidi et al. (2008) suggested considering `order.Bspline=3` and `seed.coeff=c(-1,0,1)`. For details on the construction of Θ_n see Novo et al. (2019).

Finally, after estimating $\boldsymbol{\beta}_0$ and θ_0 by minimising (1), we deal with the estimation of the nonlinear function $r_{\theta_0}(\cdot) \equiv r(\langle\theta_0, \cdot\rangle)$. For that, we employ again the kernel procedure with Nadaraya-Watson weights to smooth the partial residuals $Y_i - \boldsymbol{Z}_i^\top\widehat{\boldsymbol{\beta}}$.

For further details on the estimation procedure of the SSFPLSIM, see Novo et al. (2021).

Remark: We should note that if we set `lambda.seq`$= 0$, we can obtain the non-penalised estimation of the model, i.e. the OLS estimation. It is convenient to use `lambda.seq`$\neq 0$ when one suspects there are irrelevant variables.

Value

`call`	The matched call.
`fitted.values`	Estimated scalar response.
`residuals`	Differences between `y` and the `fitted.values`
`beta.est`	Estimate of β_0 when the optimal tuning parameters `lambda.opt`, `h.opt` and `vn.opt` are used.
`theta.est`	Coefficients of $\hat{\theta}$ in the B-spline basis (when the optimal tuning parameters `lambda.opt`, `h.opt` and `vn.opt` are used): a vector of `length(order.Bspline+nknot.theta)`.
`indexes.beta.nonnull`	Indexes of the non-zero $\hat{\beta}_j$.
`h.opt`	Selected bandwidth.

`lambda.opt` Selected value of the penalisation parameter λ.

`IC` Value of the criterion function considered to select `lambda.opt`, `h.opt` and `vn.opt`.

`Q.opt` Minimum value of the penalized criterion used to estimate β_0 and θ_0. That is, the value obtained using `theta.est` and `beta.est`.

`Q` Vector of dimension equal to the cardinal of Θ_n, containing the values of the penalized criterion for each functional index in Θ_n.

`m.opt` Index of $\hat{\theta}$ in the set Θ_n.

`lambda.min.opt.max.mopt`
A grid of values in [`lambda.min.opt.max.mopt[1]`, `lambda.min.opt.max.mopt[3]`] is considered to seek for the `lambda.opt` (`lambda.opt=lambda.min.opt.max.mopt[2]`).

`lambda.min.opt.max.m`
A grid of values in [`lambda.min.opt.max.m[m,1]`, `lambda.min.opt.max.m[m,3]`] is considered to seek for the optimal λ (`lambda.min.opt.max.m[m,2]`) used by the optimal β for each θ in Θ_n.

`h.min.opt.max.mopt`
`h.opt=h.min.opt.max.mopt[2]` (used by `theta.est` and `beta.est`) was seeked between `h.min.opt.max.mopt[1]` and `h.min.opt.max.mopt[3]`.

`h.min.opt.max.m`
For each θ in Θ_n, the optimal h (`h.min.opt.max.m[m,2]`) used by the optimal β for this θ was seeked between `h.min.opt.max.m[m,1]` and `h.min.opt.max.m[m,3]`.

`h.seq.opt` Sequence of eligible values for h considered to seek for `h.opt`.

`theta.seq.norm` The vector `theta.seq.norm[j,]` contains the coefficientes in the B-spline basis of the jth functional index in Θ_n.

`vn.opt` Selected value of `vn`.

`...`

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

References

Ait-Saidi, A., Ferraty, F., Kassa, R., and Vieu, P. (2008) Cross-validated estimations in the single-functional index model. *Statistics*, **42(6)**, 475–494, doi:10.1080/02331880801980377.

Novo S., Aneiros, G., and Vieu, P., (2019) Automatic and location-adaptive estimation in functional single-index regression. *Journal of Nonparametric Statistics*, **31(2)**, 364–392, doi:10.1080/10485252.2019.1567726.

Novo, S., Aneiros, G., and Vieu, P., (2021) Sparse semiparametric regression when predictors are mixture of functional and high-dimensional variables. *TEST*, **30**, 481–504, doi:10.1007/s11749-02000728w.

Novo, S., Aneiros, G., and Vieu, P., (2021) A kNN procedure in semiparametric functional data analysis. *Statistics and Probability Letters*, **171**, 109028, doi:10.1016/j.spl.2020.109028.

See Also

See also fsim.kernel.fit, predict.sfplsim.kernel and plot.sfplsim.kernel

Alternative procedure sfplsim.kNN.fit.

Examples

```
data("Tecator")
y<-Tecator$fat
X<-Tecator$absor.spectra2
z1<-Tecator$protein
z2<-Tecator$moisture

#Quadratic, cubic and interaction effects of the scalar covariates.
z.com<-cbind(z1,z2,z1^2,z2^2,z1^3,z2^3,z1*z2)
train<-1:160


#SSFPLSIM fit. Convergence errors for some theta are obtained.
ptm=proc.time()
fit<-sfplsim.kernel.fit(x=X[train,], z=z.com[train,], y=y[train],
      max.q.h=0.35,lambda.min.h=0.02,lambda.min.l=0.01,
      factor.pn=2, max.iter=5000, nknot.theta=4,criterion="BIC",
      penalty="grSCAD",nknot=20)
proc.time()-ptm

#Results
fit
names(fit)
```

---

sfplsim.kNN.fit — *Sparse semi-functional partial linear single-index model fit using kNN estimation*

Description

This function fits a sparse semi-functional partial linear single-index model between a scalar response, a functional explanatory variable and a vector of scalar covariates. The function uses the penalised least-squares regularization procedure combined with k-nearest neighbours (kNN) estimation with Nadaraya-Watson weights.

The procedure requires the B-spline representation to estimate the functional index θ_0 and an objective criterion (`criterion`) to select the number of neighbours (`k.opt`) and the regularization parameter (`lambda.opt`).

Usage

```
sfplsim.kNN.fit(x, z, y, seed.coeff = c(-1, 0, 1), order.Bspline = 3,
  nknot.theta = 3,t0 = NULL,knearest = NULL, min.knn = 2, max.knn = NULL,
  step = NULL,range.grid = NULL, kind.of.kernel = "quad", nknot = NULL,
  lambda.min = NULL, lambda.min.h = NULL, lambda.min.l = NULL,
  factor.pn = 1, nlambda = 100, lambda.seq = NULL,vn = ncol(z),
  nfolds = 10, seed = 123, criterion = c("GCV", "BIC", "AIC", "k-fold-CV"),
  penalty = c("grLasso", "grMCP", "grSCAD", "gel", "cMCP",
  "gBridge", "gLasso", "gMCP"), max.iter = 1000)
```

Arguments

x	Matrix containing the observations of the functional covariate collected by row.
z	Matrix containing the observations of the scalar covariates collected by row.
y	Vector containing the scalar response.
seed.coeff	Vector of initial values used to build the set Θ_n (see section Details). The coefficients for the B-spline representation of each eligible functional index $\theta \in \Theta_n$ are obtained from seed.coeff. The default is c(-1,0,1).
order.Bspline	Positive integer giving the order of the B-spline basis functions. This is the number of coefficients in each piecewise polynomial segment. The default is 3.
nknot.theta	Positive integer indicating the number of uniform interior knots of the B-spline basis for the B-spline representation of θ_0. The default is 3.
t0	Value in the domain of the functional indexes at which we evaluate them to build the set Θ_n. We assume $\theta_0(t_0) > 0$ for some arbitrary t_0 in the domain to ensure model identifiability. If t0=NULL, then mean(range.grid) is considered.
knearest	Vector of positive integers containing the sequence in which the number of nearest neighbours k.opt is selected. If knearest=NULL, then knearest <- seq(from=min.knn, to=max.knn, by=step).
min.knn	Positive integer indicating the minumum value of the sequence in which the number of nearest neighbours k.opt is selected (thus, this number must be smaller than the sample size). The default is 2.
max.knn	Positive integer indicating the maximum value of the sequence in which the number of nearest neighbours k.opt is selected (thus, this number must be larger than min.kNN and smaller than the sample size, n). The default is max.knn <- n%/%2.
step	Positive integer used to build the sequence of k-nearest neighbours in the following way: min.knn, min.knn + step, min.knn + 2*step, min.knn + 3*step,... The default is step<-ceiling(n/100).
range.grid	Vector of length 2 containing the endpoints of the grid at which the observations of the functional covariate x are evaluated (i.e. the range of the discretization). If range.grid=NULL, then range.grid=c(1,p) is considered, where p is the size of the discretization size of x (i.e. ncol(x)).
kind.of.kernel	The type of kernel function used. Only Epanechnikov kernel ("quad") is available.

nknot	Positive integer indicating the number of interior knots for the B-spline representation of the functional covariate. The default value is (p - order.Bspline - 1)%/%2.
lambda.min	The smallest value for lambda (i. e., the smallest value of the sequence in which lambda.opt is selected), as fraction of lambda.max. The defaults is lambda.min.l if the number of observations is larger than factor.pn times the number of covariates and lambda.min.h otherwise.
lambda.min.h	The smallest value of the sequence in which lambda.opt is selected if the number of observations is smaller than factor.pn times the number of scalar covariates. The default is 0.05.
lambda.min.l	The smallest value of the sequence in which lambda.opt is selected if the number of observations is larger than factor.pn times the number of scalar covariates. The default is 0.0001.
factor.pn	Positive integer used to set lambda.min. The default value is 1.
nlambda	Positive integer indicating the number of values of the sequence in which lambda.opt is selected. The default is 100.
lambda.seq	Sequence of values in which lambda.opt is selected. If lambda.seq=NULL, then the programme builds the sequence automatically using lambda.min and nlambda. For non-penalized estimation, i. e. ordinary least squares estimation, set lambda.seq=0.
vn	Positive integer or vector of positive integers indicating the number of groups of consecutive variables to be penalised together. The default value is vn=ncol(z), which leads to the individual penalisation of each scalar covariate.
nfolds	Positive integer indicating the number of cross-validation folds (used if criterion="k-fold-CV"). The default is 10.
seed	You may set the seed of the random number generator to obtain reproducible results (used if criterion="k-fold-CV"). The default is 123.
criterion	The criterion by which to select the regularization parameter lambda.opt and k.opt. One of "GCV", "BIC", "AIC" or "k-fold-CV". The default is "GCV".
penalty	The penalty function to be applied in the penalized least squares procedure. Only "grLasso" and "grSCAD" are implemented.
max.iter	Maximum number of iterations (total across entire path). Default is 1000.

Details

The sparse semi-functional partial linear single-index model (SSFPLSIM) is given by the expression:

$$Y_i = Z_{i1}\beta_{01} + \cdots + Z_{ip_n}\beta_{0p_n} + r(\langle\theta_0, X_i\rangle) + \varepsilon_i \quad i = 1, \ldots, n,$$

where Y_i denotes a scalar response, $Z_{i1}, \ldots, Z_{ip_n}$ are random covariates taking values in $\mathbb{R}$ and X_i is a functional random covariate valued in a separable Hilbert space $\mathcal{H}$ with inner product $\langle\cdot,\cdot\rangle$. In this equation, $\boldsymbol{\beta}_0 = (\beta_{01}, \ldots, \beta_{0p_n})^\top \in \mathbb{R}^{p_n}$, $\theta_0 \in \mathcal{H}$ and $r(\cdot)$ are a vector of unknown real parameters, an unknown functional direction and an unknown smooth real-valued function, respectively. In addition, ε_i is the random error.

The SSFPLSIM is fitted using the penalised least-squares approach. The first idea is to transform the SSFPLSIM into a linear model by extracting from Y_i and Z_{ij} $(j = 1, \ldots, p_n)$ the effect of the

functional covariate X_i using functional single-index regression. This is made using nonparametric kNN estimation (see, for details, the documentation of the function `fsim.kNN.fit`).

Then, an approximate linear model is obtained:

$$\widetilde{\boldsymbol{Y}}_{\theta_0} \approx \widetilde{\boldsymbol{Z}}_{\theta_0}\boldsymbol{\beta}_0 + \boldsymbol{\varepsilon},$$

and the penalised least-squares procedure is applied to this model by minimising over the pair $(\boldsymbol{\beta}, \theta)$

$$\mathcal{Q}(\boldsymbol{\beta}, \theta) = \frac{1}{2}\left(\widetilde{\boldsymbol{Y}}_\theta - \widetilde{\boldsymbol{Z}}_\theta\boldsymbol{\beta}\right)^\top \left(\widetilde{\boldsymbol{Y}}_\theta - \widetilde{\boldsymbol{Z}}_\theta\boldsymbol{\beta}\right) + n\sum_{j=1}^{p_n} \mathcal{P}_{\lambda_{j_n}}(|\beta_j|), \quad (1)$$

where $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_{p_n})^\top$, $\mathcal{P}_{\lambda_{j_n}}(\cdot)$ is a penalty function (specified in the argument `penalty`) and $\lambda_{j_n} > 0$ is a tuning parameter. To reduce the quantity of tuning parameters, λ_j, to be selected for each sample, we consider $\lambda_j = \lambda\widehat{\sigma}_{\beta_{0,j,OLS}}$, where $\beta_{0,j,OLS}$ denotes the OLS estimate of $\beta_{0,j}$ and $\widehat{\sigma}_{\beta_{0,j,OLS}}$ is the estimated standard deviation. Both λ and k (in the kNN estimation) are selected using the objetive criterion specified in the argument `criterion`.

In addition, the function uses B-spline representation to build a set Θ_n of eligible functional indexes θ. The dimension of the B-spline basis is `order.Bspline+nknot.theta` and the set of eligible coefficients is obtained by calibrating (to ensure the identifiability of the model) the set of initial coefficients given in `seed.coeff`. The larger this set, the higher the size of Θ_n. Since our approach requires intensive computation, we need a trade-off between the size of Θ_n and the performance of the estimator. For that, Ait-Saidi et al. (2008) suggested considering `order.Bspline=3` and `seed.coeff=c(-1,0,1)`. For details on the construction of Θ_n see Novo et al. (2019).

Finally, after estimating $\boldsymbol{\beta}_0$ and θ_0 by minimising (1), we deal with the estimation of the nonlinear function $r_{\theta_0}(\cdot) \equiv r(\langle\theta_0, \cdot\rangle)$. For that we employ again the kNN procedure with Nadaraya-Watson weights to smooth the partial residuals $Y_i - \boldsymbol{Z}_i^\top\widehat{\boldsymbol{\beta}}$.

For further details on the estimation procedure of the SSFPLSIM, see Novo et al. (2021).

Remark: We should note that if we set `lambda.seq`$= 0$, we can obtain the non-penalised estimation of the model, i.e. the OLS estimation. It is convenient to use `lambda.seq`$\neq 0$ when one suspects there are irrelevant variables.

Value

`call`	The matched call.
`fitted.values`	Estimated scalar response.
`residuals`	Differences between `y` and the `fitted.values`
`beta.est`	$\hat{\boldsymbol{\beta}}$ (i. e. the estimate of $\boldsymbol{\beta}_0$ when the optimal tuning parameters `lambda.opt`, `k.opt` and `vn.opt` are used).
`theta.est`	Coefficients of $\hat{\theta}$ in the B-spline basis (when the optimal tuning parameters `lambda.opt`, `k.opt` and `vn.opt`) are used): a vector of `length(order.Bspline+nknot.theta)`.
`indexes.beta.nonnull`	Indexes of the non-zero $\hat{\beta}_j$.
`k.opt`	Selected number of nearest neighbours.
`lambda.opt`	Selected value of the penalisation parameter λ.

`IC` Value of the criterion function considered to select `lambda.opt`, `k.opt` and `vn.opt`.

`Q.opt` Minimum value of the penalized criterion used to estimate $\boldsymbol{\beta}_0$ and θ_0. That is, the value obtained using `theta.est` and `beta.est`.

`Q` Vector of dimension equal to the cardinal of Θ_n, containing the values of the penalized criterion for each functional index in Θ_n.

`m.opt` Index of $\hat{\theta}$ in the set Θ_n.

`lambda.min.opt.max.mopt`
A grid of values in [`lambda.min.opt.max.mopt[1]`, `lambda.min.opt.max.mopt[3]`] is considered to seek for the `lambda.opt` (`lambda.opt=lambda.min.opt.max.mopt[2]`).

`lambda.min.opt.max.m`
A grid of values in [`lambda.min.opt.max.m[m,1]`, `lambda.min.opt.max.m[m,3]`] is considered to seek for the optimal λ (`lambda.min.opt.max.m[m,2]`) used by the optimal $\boldsymbol{\beta}$ for each θ in Θ_n.

`knn.min.opt.max.mopt`
`k.opt=knn.min.opt.max.mopt[2]` (used by `theta.est` and `beta.est`) was seeked between `knn.min.opt.max.mopt[1]` and `knn.min.opt.max.mopt[3]` (no necessarly the step was 1).

`knn.min.opt.max.m`
For each θ in Θ_n, the optimal k (`knn.min.opt.max.m[m,2]`) used by the optimal β for this θ was seeked between `knn.min.opt.max.m[m,1]` and `knn.min.opt.max.m[m,3]` (no necessarly the step was 1).

`knearest` Sequence of eligible values for k considered to seek for `k.opt`.

`theta.seq.norm` The vector `theta.seq.norm[j,]` contains the coefficientes in the B-spline basis of the jth functional index in Θ_n.

`vn.opt` Selected value of `vn`.

`...`

Author(s)

German Aneiros Perez <german.aneiros@udc.es>

Silvia Novo Diaz <snovo@est-econ.uc3m.es>

References

Ait-Saidi, A., Ferraty, F., Kassa, R., and Vieu, P., (2008) Cross-validated estimations in the single-functional index model. *Statistics*, **42(6)**, 475–494, doi:10.1080/02331880801980377.

Novo S., Aneiros, G., and Vieu, P., (2019) Automatic and location-adaptive estimation in functional single-index regression. *Journal of Nonparametric Statistics*, **31(2)**, 364–392, doi:10.1080/10485252.2019.1567726.

Novo, S., Aneiros, G., and Vieu, P., (2021) Sparse semiparametric regression when predictors are mixture of functional and high-dimensional variables. *TEST*, **30**, 481–504, doi:10.1007/s11749-02000728w.

Novo, S., Aneiros, G., and Vieu, P., (2021) A kNN procedure in semiparametric functional data analysis. *Statistics and Probability Letters*, **171**, 109028, doi:10.1016/j.spl.2020.109028

See Also

See also fsim.kNN.fit, predict.sfplsim.kNN and plot.sfplsim.kNN

Alternative procedure sfplsim.kernel.fit.

Examples

```
data("Tecator")
y<-Tecator$fat
X<-Tecator$absor.spectra2
z1<-Tecator$protein
z2<-Tecator$moisture

#Quadratic, cubic and interaction effects of the scalar covariates.
z.com<-cbind(z1,z2,z1^2,z2^2,z1^3,z2^3,z1*z2)
train<-1:160


#SSFPLSIM fit. Convergence errors for some theta are obtained.
ptm=proc.time()
fit<-sfplsim.kNN.fit(y=y[train],x=X[train,], z=z.com[train,], max.knn=20,
    lambda.min.h=0.02,lambda.min.l=0.01, factor.pn=2,  nknot.theta=4,
    criterion="BIC",range.grid=c(850,1050), penalty="grSCAD",
    nknot=20, max.iter=5000)
proc.time()-ptm

#Results
fit
names(fit)
```

Sugar *Sugar data*

Description

Ash content and absorbance spectra at two different excitation wavelengths of 268 samples of sugar. Detailed information about this dataset can be found in https://ucphchemometrics.com/datasets/.

Usage

```
data(Sugar)
```

Format

A `list` containing:

- `ash`: A vector with the ash contents.
- `wave.290`: A matrix containing the absorbance spectra observed on 571 equally spaced wavelengths in the range 275-560 nm at excitation wavelengths 290 nm.
- `wave.240`: A matrix containing the absorbance spectra observed on 571 equally spaced wavelengths in the range 275-560 nm at excitation wavelengths 240 nm.

References

Aneiros, G., and Vieu, P. (2015) Partial linear modelling with multi-functional covariates. *Computational Statistics*, **30**, 647–671, doi:10.1007/s0018001505688.

Novo, S., Vieu, P., and Aneiros, G., (2021) Fast and efficient algorithms for sparse semiparametric bi-functional regression. *Australian and New Zealand Journal of Statistics*, **63**, 606–638, doi:10.1111/anzs.12355.

Examples

```
data(Sugar)
names(Sugar)
Sugar$ash
dim(Sugar$wave.290)
dim(Sugar$wave.240)
```

Tecator *Tecator data*

Description

Fat, protein, moisture content and absorbance spectra (with the first and the second derivative) of 215 samples of meat. A detailed description of the data can be seen in http://lib.stat.cmu.edu/datasets/tecator.

Usage

```
data(Tecator)
```

Format

A `list` containing:

- `fat`: A vector with the fat contents.
- `protein`: A vector with the protein contents.
- `moisture`: A vector with the moisture contents.
- `absor.spectra`: A matrix containing the near-infrared absorbance spectra observed on 100 equally spaced wavelengths in the range 850-1050 nm.

- absor.spectra1: Fist derivative of the absorbance spectra (computed using B-spline representation of the curves).
- absor.spectra2: Second derivative of the absorbance spectra (computed using B-spline representation of the curves).

References

Ferraty, F. and Vieu, P. (2006) *Nonparametric functional data analysis*, Springer Series in Statistics, New York.

Examples

```
data(Tecator)
names(Tecator)
Tecator$fat
Tecator$protein
Tecator$moisture
dim(Tecator$absor.spectra)
```

Index

∗ **datasets**
    Sugar, 100
    Tecator, 101

FASSMR.kernel.fit, *3*, 4, *13*, *30*, *32*, *43*, *56*
FASSMR.kNN.fit, *3*, *8*, 9, *32*, *36*, *38*, *43*, *56*
fsemipar *(*fsemipar-package*)*, 2
fsemipar-package, 2
fsemipar.internal, 14
fsim.kernel.fit, *3*, 15, *20*, *24*, *43*, *95*
fsim.kernel.test, *3*, *18*, 18, *26*
fsim.kNN.fit, *3*, *18*, 21, *26*, *43*, *100*
fsim.kNN.test, *3*, *20*, *24*, 24

IASSMR.kernel.fit, *3*, *8*, 27, *38*, *43*, *47*
IASSMR.kNN.fit, *3*, *13*, *32*, 33, *43*, *47*

lm.pels.fit, *3*, 39, *43*, *49*, *59*, *68*, *70*

plot.classes, 41
plot.FASSMR.kernel, *8*
plot.FASSMR.kernel *(*plot.classes*)*, 41
plot.FASSMR.kNN, *13*
plot.FASSMR.kNN *(*plot.classes*)*, 41
plot.fsim.kernel, *18*
plot.fsim.kernel *(*plot.classes*)*, 41
plot.fsim.kNN, *24*
plot.fsim.kNN *(*plot.classes*)*, 41
plot.IASSMR.kernel, *32*
plot.IASSMR.kernel *(*plot.classes*)*, 41
plot.IASSMR.kNN, *38*
plot.IASSMR.kNN *(*plot.classes*)*, 41
plot.lm.pels *(*plot.classes*)*, 41
plot.PVS *(*plot.classes*)*, 41
plot.PVS.kernel, *75*
plot.PVS.kNN, *81*
plot.sfpl.kernel, *86*
plot.sfpl.kernel *(*plot.classes*)*, 41
plot.sfpl.kNN, *90*
plot.sfpl.kNN *(*plot.classes*)*, 41
plot.sfplsim.kernel, *95*
plot.sfplsim.kernel *(*plot.classes*)*, 41
plot.sfplsim.kNN, *100*
plot.sfplsim.kNN *(*plot.classes*)*, 41
predict.FASSMR.kernel, *8*
predict.FASSMR.kernel
        *(*predict.sfplsim.FASSMR*)*, 54
predict.FASSMR.kNN, *13*
predict.FASSMR.kNN
        *(*predict.sfplsim.FASSMR*)*, 54
predict.fsim, 44
predict.fsim.kernel, *18*, *20*
predict.fsim.kNN, *24*, *26*
predict.IASSMR, 45
predict.IASSMR.kernel, *32*
predict.IASSMR.kNN, *38*
predict.lm, 48
predict.mfplm, 50
predict.PVS *(*predict.lm*)*, 48
predict.PVS.kernel, *75*
predict.PVS.kernel *(*predict.mfplm*)*, 50
predict.PVS.kNN, *81*
predict.PVS.kNN *(*predict.mfplm*)*, 50
predict.sfpl, 52
predict.sfpl.kernel, *52*, *86*
predict.sfpl.kNN, *52*, *90*
predict.sfplsim.FASSMR, 54
predict.sfplsim.kernel, *95*
predict.sfplsim.kernel
        *(*predict.sfplsim.FASSMR*)*, 54
predict.sfplsim.kNN, *100*
predict.sfplsim.kNN
        *(*predict.sfplsim.FASSMR*)*, 54
print.FASSMR.kernel
        *(*print.summary.mfplsim*)*, 60
print.FASSMR.kNN
        *(*print.summary.mfplsim*)*, 60
print.fsim.kernel *(*print.summary.fsim*)*,
        57

print.fsim.kNN *(print.summary.fsim)*, 57
print.IASSMR.kernel
(print.summary.mfplsim), 60
print.IASSMR.kNN
(print.summary.mfplsim), 60
print.lm.pels *(print.summary.lm)*, 58
print.PVS *(print.summary.lm)*, 58
print.PVS.kernel *(print.summary.mfpl)*,
59
print.PVS.kNN *(print.summary.mfpl)*, 59
print.sfpl.kernel *(print.summary.sfpl)*,
62
print.sfpl.kNN *(print.summary.sfpl)*, 62
print.sfplsim.kernel
(print.summary.sfplsim), 63
print.sfplsim.kNN
(print.summary.sfplsim), 63
print.summary.fsim, 57
print.summary.lm, 58
print.summary.mfpl, 59
print.summary.mfplsim, 60
print.summary.sfpl, 62
print.summary.sfplsim, 63
projec, *3*, 64, *82*
PVS.fit, *3*, *41*, *43*, *49*, *59*, 66
PVS.kernel.fit, *3*, *43*, *52*, 70, *81*
PVS.kNN.fit, *3*, *43*, *52*, *75*, 76

semimetric.projec, *3*, *16*, *20*, *22*, *26*, *65*, 81
sfpl.kernel.fit, *3*, *43*, *52*, *54*, *73*, *75*, 83, *90*
sfpl.kNN.fit, *3*, *43*, *52*, *54*, *79*, *81*, *86*, 87
sfplsim.kernel.fit, *3*, *7, 8*, *30*, *32*, *43*, *47*,
56, 90, *100*
sfplsim.kNN.fit, *3*, *12, 13*, *36*, *38*, *43*, *47*,
56, *95*, 95
Sugar, *3*, 100
summary.FASSMR.kernel
(print.summary.mfplsim), 60
summary.FASSMR.kNN
(print.summary.mfplsim), 60
summary.fsim.kernel
(print.summary.fsim), 57
summary.fsim.kNN *(print.summary.fsim)*,
57
summary.IASSMR.kernel
(print.summary.mfplsim), 60
summary.IASSMR.kNN
(print.summary.mfplsim), 60
summary.lm.pels *(print.summary.lm)*, 58
summary.PVS *(print.summary.lm)*, 58
summary.PVS.kernel
(print.summary.mfpl), 59
summary.PVS.kNN *(print.summary.mfpl)*, 59
summary.sfpl.kernel
(print.summary.sfpl), 62
summary.sfpl.kNN *(print.summary.sfpl)*,
62
summary.sfplsim.kernel
(print.summary.sfplsim), 63
summary.sfplsim.kNN
(print.summary.sfplsim), 63

Tecator, *3*, 101