Package: fsemipar (via r-universe)

October 19, 2026

Type Package

Title Estimation, Variable Selection and Prediction for Functional Semiparametric Models

Version 1.1.1

Date 2024-05-21

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

Depends R (>= 3.5.0), grpreg

Imports DiceKriging, splines, gtools, stats, parallelly, doParallel, parallel, foreach, ggplot2, gridExtra, tidyr

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

Description Routines for the estimation or simultaneous estimation and variable selection in several functional semiparametric models with scalar responses are provided. These models include the functional single-index model, the semi-functional partial linear model, and the semi-functional partial linear single-index model. Additionally, the package offers algorithms for handling scalar covariates with linear effects that originate from the discretization of a curve. This functionality is applicable in the context of the linear model, the multi-functional partial linear model, and the multi-functional partial linear single-index model.

License GPL (>= 2)

NeedsCompilation no

Config/pak/sysreqs libicu-dev

Repository https://silvianovo.r-universe.dev

Date/Publication 2024-05-22 02:40:16 UTC

RemoteUrl https://github.com/cran/fsemipar

RemoteRef HEAD

RemoteSha 080e003a1e64a2195ce2eddf527c5fb1479db440

Contents

fsemipar-package . . . 3
FASSMR.kernel.fit . . . 4
FASSMR.kNN.fit . . . 9
fsemipar.internal . . . 14
fsim.kernel.fit . . . 15
fsim.kernel.fit.optim . . . 19
fsim.kernel.test . . . 22
fsim.kNN.fit . . . 24
fsim.kNN.fit.optim . . . 28
fsim.kNN.test . . . 31
IASSMR.kernel.fit . . . 33
IASSMR.kNN.fit . . . 39
lm.pels.fit . . . 45
plot.classes . . . 48
predict.fsim . . . 51
predict.IASSMR . . . 52
predict.lm . . . 55
predict.mfplm.PVS . . . 56
predict.sfpl . . . 59
predict.sfplsim.FASSMR . . . 61
print.summary.fsim . . . 64
print.summary.lm . . . 65
print.summary.mfpl . . . 66
print.summary.mfplsim . . . 67
print.summary.sfpl . . . 69
print.summary.sfplsim . . . 70
projec . . . 71
PVS.fit . . . 73
PVS.kernel.fit . . . 77
PVS.kNN.fit . . . 83
semimetric.projec . . . 88
sfpl.kernel.fit . . . 90
sfpl.kNN.fit . . . 94
sfplsim.kernel.fit . . . 97
sfplsim.kNN.fit . . . 102
Sugar . . . 107
Tecator . . . 108

Index **109**

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

Description

This package is dedicated to the estimation and simultaneous estimation and variable selection in several functional semiparametric models with scalar response. These include the functional single-index model, the semi-functional partial linear model, and the semi-functional partial linear single-index model. Additionally, it encompasses algorithms for addressing estimation and variable selection in linear models and bi-functional partial linear models when the scalar covariates with linear effects are derived from the discretisation of a curve. Furthermore, the package offers routines for kernel- and kNN-based estimation using Nadaraya-Watson weights in models with a nonparametric or semiparametric component. It also includes S3 methods (predict, plot, print, summary) to facilitate statistical analysis across all the considered models and estimation procedures.

Details

The package can be divided into several thematic sections:

1. Estimation of the functional single-index model.
   - `projec`.
   - `semimetric.projec`.
   - `fsim.kernel.fit` and `fsim.kNN.fit`.
   - `fsim.kernel.fit.optim` and `fsim.kNN.fit.optim`
   - `fsim.kernel.test` and `fsim.kNN.test`.
   - `predict`, `plot`, `summary` and `print` methods for `fsim.kernel` and `fsim.kNN` classes.
2. Simultaneous estimation and variable selection in linear and semi-functional partial linear models.
   (a) Linear model
      - `lm.pels.fit`.
      - `predict`, `summary`, `plot` and `print` methods for `lm.pels` class.
   (b) Semi-functional partial linear model.
      - `sfpl.kernel.fit` and `sfpl.kNN.fit`.
      - `predict`, `summary`, `plot` and `print` methods for `sfpl.kernel` and `sfpl.kNN` classes.
   (c) Semi-functional partial linear single-index model.
      - `sfplsim.kernel.fit` and `sfplsim.kNN.fit`.
      - `predict`, `summary`, `plot` and `print` methods for `sfplsim.kernel` and `sfplsim.kNN` classes.
3. Algorithms for impact point selection in models with covariates derived from the discretisation of a curve.
   (a) Linear model
      - `PVS.fit`.

- `predict`, `summary`, `plot` and `print` methods for `PVS` class.

(b) Bi-functional partial linear model.

- `PVS.kernel.fit` and `PVS.kNN.fit`.
- `predict`, `summary`, `plot` and `print` methods for `PVS.kernel` and `PVS.kNN` classes.

(c) Bi-functional partial linear single-index model.

- `FASSMR.kernel.fit` and `FASSMR.kNN.fit`.
- `IASSMR.kernel.fit` and `IASSMR.kNN.fit`.
- `predict`, `summary`, `plot` and `print` methods for `FASSMR.kernel`, `FASSMR.kNN`, `IASSMR.kernel` and `IASSMR.kNN` classes.

4. 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` *Impact point selection with FASSMR and kernel estimation*

Description

This function implements the Fast Algorithm for Sparse Semiparametric Multi-functional Regression (FASSMR) with kernel estimation. This algorithm is specifically designed for estimating multi-functional partial linear single-index models, which incorporate multiple scalar variables and a functional covariate as predictors. These scalar variables are derived from the discretisation of a curve and have linear effect while the functional covariate exhibits a single-index effect.

FASSMR selects the impact points of the discretised curve and estimates the model. The algorithm employs a penalised least-squares regularisation procedure, integrated with kernel estimation using Nadaraya-Watson weights. It uses B-spline expansions to represent curves and eligible functional indexes. Additionally, it utilises an objective criterion (`criterion`) to determine 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,  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, 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 = "GCV",
penalty = "grSCAD", max.iter = 1000, n.core = NULL)
```

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 regularly spaced interior knots in the B-spline expansion of θ_0. The default is 3.
`min.q.h`	Minimum quantile order of the distances between curves, which are computed using the projection semi-metric. This value determines the lower endpoint of the range from which the bandwidth is selected. The default is 0.05.
`max.q.h`	Maximum quantile order of the distances between curves, which are computed using the projection semi-metric. This value determines the upper endpoint of the range from 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 bandwidths in the grid. The default is 10.

`kind.of.kernel` The type of kernel function used. Currently, 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 discretisation). If `range.grid=NULL`, then `range.grid=c(1,p)` is considered, where `p` is the discretisation size of `x` (i.e. `ncol(x)`).

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

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

`lambda.min.h` The lower endpoint of the sequence in which `lambda.opt` is selected if the sample size is smaller than `factor.pn` times the number of linear covariates. The default is 0.05.

`lambda.min.l` The lower endpoint of the sequence in which `lambda.opt` is selected if the sample size is larger than `factor.pn` times the number of linear 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 in the sequence from 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)`, resulting in the individual penalization of each scalar covariate.

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

`seed` You may set the seed for the random number generator to ensure reproducible results (applicable when `criterion="k-fold-CV"` is used). The default seed value is 123.

`wn` A vector of positive integers indicating the eligible number of covariates in the reduced model. For more information, refer to the section `Details`. The default is `c(10,15,20)`.

`criterion` The criterion used to select the tuning and regularisation parameters: `wn.opt`, `lambda.opt` and `h.opt` (also `vn.opt` if needed). Options include `"GCV"`, `"BIC"`, `"AIC"`, or `"k-fold-CV"`. The default setting is `"GCV"`.

`penalty` The penalty function applied in the penalised least-squares procedure. Currently, only "grLasso" and "grSCAD" are implemented. The default is "grSCAD".

`max.iter` Maximum number of iterations allowed across the entire path. The default value is 1000.

`n.core` Number of CPU cores designated for parallel execution. The default is `n.core<-availableCores(omit=`

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 assumed to be a curve defined on some interval $[a, b]$ which is observed at the points $a \leq t_1 < \cdots < t_{p_n} \leq b$.
- $\beta_0 = (\beta_{01}, \ldots, \beta_{0p_n})^\top$ 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 it is expected that they contain very similar information about the response).

To explain the algorithm, we assume, without loss 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. This consideration allows us to build a subset of the initial p_n linear covariates, containging only w_n equally spaced discretised observations of ζ covering the entire 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 the real number z.

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

Remark: If the condition $p_n = w_n q_n$ is not met (then p_n/w_n is not an integer number), the function considers variable $q_n = q_{n,k}$ values $k = 1, \ldots, w_n$. 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 the real number z.

The function supports parallel computation. To avoid it, we can set `n.core=1`.

Value

`call`	The matched call.
`fitted.values`	Estimated scalar response.
`residuals`	Differences between `y` and the `fitted.values`.
`beta.est`	$\hat{\beta}$ (i.e. estimate of β_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 θ_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 for 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 entire 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.l=0.03,
        max.q.h=0.35, nknot=20,criterion="BIC",
        max.iter=5000)
proc.time()-ptm
```

FASSMR.kNN.fit *Impact point selection with FASSMR and kNN estimation*

Description

This function implements the Fast Algorithm for Sparse Semiparametric Multi-functional Regression (FASSMR) with kNN estimation. This algorithm is specifically designed for estimating multi-functional partial linear single-index models, which incorporate multiple scalar variables and a functional covariate as predictors. These scalar variables are derived from the discretisation of a curve and have linear effect while the functional covariate exhibits a single-index effect.

FASSMR selects the impact points of the discretised curve and estimates the model. The algorithm employs a penalised least-squares regularisation procedure, integrated with kNN estimation using Nadaraya-Watson weights. It uses B-spline expansions to represent curves and eligible functional indexes. Additionally, it utilises an objective criterion (criterion) to determine 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,  knearest = NULL, min.knn = 2, max.knn = NULL, step = NULL,
kind.of.kernel = "quad",range.grid = NULL, 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 = "GCV",
penalty = "grSCAD", max.iter = 1000, n.core = NULL)
```

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 regularly spaced interior knots in the B-spline expansion of θ_0. The default is 3.
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	A positive integer that represents the minimum value in the sequence for selecting the number of nearest neighbours k.opt. This value should be less than the sample size. The default is 2.
max.knn	A positive integer that represents the maximum value in the sequence for selecting number of nearest neighbours k.opt. This value should be less than the sample size. The default is max.knn <- n%/%5.
step	A positive integer used to construct the sequence of k-nearest neighbours as follows: min.knn, min.knn + step, min.knn + 2*step, min.knn + 3*step,.... The default value for step is step<-ceiling(n/100).
kind.of.kernel	The type of kernel function used. Currently, 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 discretisation). If range.grid=NULL, then range.grid=c(1,p) is considered, where p is the discretisation size of x (i.e. ncol(x)).
nknot	Positive integer indicating the number of interior knots for the B-spline expansion of the functional covariate. The default value is (p - order.Bspline - 1)%/%2.

`lambda.min`	The smallest value for lambda (i. e., the lower endpoint of the sequence in which `lambda.opt` is selected), as fraction of `lambda.max`. The defaults is `lambda.min.l` if the sample size is larger than `factor.pn` times the number of linear covariates and `lambda.min.h` otherwise.
`lambda.min.h`	The lower endpoint of the sequence in which `lambda.opt` is selected if the sample size is smaller than `factor.pn` times the number of linear covariates. The default is 0.05.
`lambda.min.l`	The lower endpoint of the sequence in which `lambda.opt` is selected if the sample size is larger than `factor.pn` times the number of linear 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 in the sequence from 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)`, resulting in the individual penalization of each scalar covariate.
`nfolds`	Positive integer indicating the number of cross-validation folds (used when `criterion="k-fold-CV"`). Default is 10.
`seed`	You may set the seed for the random number generator to ensure reproducible results (applicable when `criterion="k-fold-CV"` is used). The default seed value is 123.
`wn`	A vector of positive integers indicating the eligible number of covariates in the reduced model. For more information, refer to the section `Details`. The default is `c(10,15,20)`.
`criterion`	The criterion used to select the tuning and regularisation parameters: `wn.opt`, `k.opt` and `lambda.opt` (also `vn.opt` if needed). Options include `"GCV"`, `"BIC"`, `"AIC"`, or `"k-fold-CV"`. The default setting is `"GCV"`.
`penalty`	The penalty function applied in the penalised least-squares procedure. Currently, only "grLasso" and "grSCAD" are implemented. The default is "grSCAD".
`max.iter`	Maximum number of iterations allowed across the entire path. The default value is 1000.
`n.core`	Number of CPU cores designated for parallel execution. The default is `n.core<-availableCores(omit=`

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 assumed to be a curve defined on some interval $[a, b]$ which is observed at the points $a \leq t_1 < \cdots < t_{p_n} \leq b$.

- $\beta_0 = (\beta_{01}, \ldots, \beta_{0p_n})^\top$ 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 it is expected that they contain very similar information about the response).

To explain the algorithm, we assume, without loss 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. This consideration allows us to build a subset of the initial p_n linear covariates, containging only w_n equally spaced discretised observations of ζ covering the entire 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 the real number z.

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

Remark: If the condition $p_n = w_n q_n$ is not met (then p_n/w_n is not an integer number), the function considers variable $q_n = q_{n,k}$ values $k = 1, \ldots, w_n$. 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 the real number z.

The function supports parallel computation. To avoid it, we can set `n.core=1`.

Value

`call`	The matched call.
`fitted.values`	Estimated scalar response.
`residuals`	Differences between `y` and the `fitted.values`.
`beta.est`	$\hat{\beta}$ (i.e. estimate of β_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 for 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 entire 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.l=0.03, max.knn=20,nknot=20,criterion="BIC",
        max.iter=5000)
proc.time()-ptm

fit
names(fit)
```

`fsemipar.internal` *Package fsemipar internal functions*

Description

The package includes the following internal functions, based on the code by F. Ferraty, which is available on his website at 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
- 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 and joint LOOCV minimisation*

Description

This function fits a functional single-index model (FSIM) between a functional covariate and a scalar response. It employs kernel estimation with Nadaraya-Watson weights and uses B-spline expansions to represent curves and eligible functional indexes.

The function also utilises the leave-one-out cross-validation (LOOCV) criterion to select the bandwidth (`h.opt`) and the coefficients of the functional index in the spline basis (`theta.est`). It performs a joint minimisation of the LOOCV objective function in both the bandwidth and the functional index.

Usage

```
fsim.kernel.fit(x, y, seed.coeff = c(-1, 0, 1), order.Bspline = 3,
nknot.theta = 3,  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, n.core = 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).
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 regularly spaced interior knots in the B-spline expansion of θ_0. The default is 3.
min.q.h	Minimum quantile order of the distances between curves, which are computed using the projection semi-metric. This value determines the lower endpoint of the range from which the bandwidth is selected. The default is 0.05.
max.q.h	Maximum quantile order of the distances between curves, which are computed using the projection semi-metric. This value determines the upper endpoint of the range from 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 bandwidths in the grid. The default is 10.
kind.of.kernel	The type of kernel function used. Currently, 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 discretisation). If range.grid=NULL, then range.grid=c(1,p) is considered, where p is the discretisation size of x (i.e. ncol(x)).
nknot	Positive integer indicating the number of interior knots for the B-spline expansion of the functional covariate. The default value is (p - order.Bspline - 1)%/%2.
n.core	Number of CPU cores designated for parallel execution.The default is n.core<-availableCores(omit=1

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 an inner product $\langle\cdot,\cdot\rangle$. The term 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 kernel 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

- the real positive number h 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, 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 expansions to represent curves (dimension `nknot+order.Bspline`) and eligible functional indexes (dimension `nknot.theta+order.Bspline`). Then, we build a set Θ_n of eligible functional indexes by calibrating (to ensure the identifiability of the model) the set of initial coefficients given in `seed.coeff`. The larger this set is, the greater the size of Θ_n. Since our approach requires intensive computation, a trade-off between the size of Θ_n and the performance of the estimator is necessary. 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 LOOCV criterion. This function performs a joint minimisation in both parameters, the bandwidth and the functional index, and supports parallel computation. To avoid parallel computation, we can set `n.core=1`.

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.
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=4)
proc.time()-ptm
fit
names(fit)
```

`fsim.kernel.fit.optim` *Functional single-index model fit using kernel estimation and iterative LOOCV minimisation*

Description

This function fits a functional single-index model (FSIM) between a functional covariate and a scalar response. It employs kernel estimation with Nadaraya-Watson weights and uses B-spline expansions to represent curves and eligible functional indexes.

The function also utilises the leave-one-out cross-validation (LOOCV) criterion to select the bandwidth (`h.opt`) and the coefficients of the functional index in the spline basis (`theta.est`). It performs an iterative minimisation of the LOOCV objective function, starting from an initial set of coefficients (`gamma`) for the functional index.

Usage

```
fsim.kernel.fit.optim(x, y, nknot.theta = 3, order.Bspline = 3, gamma = 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, threshold = 0.005)
```

Arguments

`x`	Matrix containing the observations of the functional covariate (i.e. curves) collected by row.
`y`	Vector containing the scalar response.
`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 regularly spaced interior knots in the B-spline expansion of θ_0. The default is 3.
`gamma`	Vector indicating the initial coefficients for the functional index used in the iterative procedure. By default, it is a vector of ones. The size of the vector is determined by the sum `nknot.theta+order.Bspline`.
`min.q.h`	Minimum quantile order of the distances between curves, which are computed using the projection semi-metric. This value determines the lower endpoint of the range from which the bandwidth is selected. The default is 0.05.
`max.q.h`	Maximum quantile order of the distances between curves, which are computed using the projection semi-metric. This value determines the upper endpoint of the range from 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 bandwidths in the grid. The default is 10.
`kind.of.kernel`	The type of kernel function used. Currently, 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 discretisation). If `range.grid=NULL`, then `range.grid=c(1,p)` is considered, where `p` is the discretisation size of `x` (i.e. `ncol(x)`).

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

`threshold` The convergence threshold for the LOOCV function (scaled by the variance of the response). The default is `5e-3`.

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 an inner product $\langle \cdot, \cdot \rangle$. The term 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 kernel 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

- the real positive number h 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, 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 expansions to represent curves (dimension `nknot+order.Bspline`) and eligible functional indexes (dimension `nknot.theta+order.Bspline`). We obtain the estimated coefficients of θ_0 in the spline basis (`theta.est`) and the selected bandwidth (`h.opt`) by minimising the LOOCV criterion. This function performs an iterative minimisation procedure, starting from an initial set of coefficients (`gamma`) for the functional index. Given a functional index, the optimal bandwidth according to the LOOCV criterion is selected. For a given bandwidth, the minimisation in the functional index is performed using the R function `optim`. The procedure is iterated until convergence. For details, see Ferraty et al. (2013).

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.
CV.opt	Minimum value of the LOOCV function, i.e. the value of LOOCV for theta.est and h.opt.
err	Value of the LOOCV function divided by var(y) for each interaction.
H	Hat matrix.
h.seq	Sequence of eligible values for the bandwidth.
CV.hseq	CV values for each h.
...	

Author(s)

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

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

References

Ferraty, F., Goia, A., Salinelli, E., and Vieu, P. (2013) Functional projection pursuit regression. *Test*, **22**, 293–320, doi:10.1007/s1174901203062.

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 predict.fsim.kernel and plot.fsim.kernel.

Alternative procedures fsim.kNN.fit.optim, fsim.kernel.fit and fsim.kNN.fit.

Examples

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

#FSIM fit.
ptm<-proc.time()
fit<-fsim.kernel.fit.optim(y[1:160],x=X[1:160,],max.q.h=0.35, nknot=20,
range.grid=c(850,1050),nknot.theta=4)
proc.time()-ptm
fit
names(fit)
```

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

Description

This function computes predictions for a functional single-index model (FSIM) with a scalar response, which is estimated using the Nadaraya-Watson kernel estimator. It requires a functional index (θ), a global bandwidth (`h`), and the new observations of the functional covariate (`x.test`) as inputs.

Usage

```
fsim.kernel.test(x, y, x.test, y.test=NULL, 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 in 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 in the the testing sample, collected by row.
`y.test`	(optional) Vector or matrix containing the scalar responses in the testing sample.
`theta`	Vector containing the coefficients of θ in a B-spline basis, such that `length(theta)=order.Bspline+nkn`
`nknot.theta`	Number of regularly spaced interior knots in the B-spline expansion of θ_0. The default is 3.
`order.Bspline`	Order of the B-spline basis functions. This is the number of coefficients in each piecewise polynomial segment. The default is 3
`h`	The global bandwidth. The default if 0.5.
`kind.of.kernel`	The type of kernel function used. Currently, 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 discretisation). If `range.grid=NULL`, then `range.grid=c(1,p)` is considered, where `p` is the discretisation size of `x` (i.e. `ncol(x)`).
`nknot`	Number of regularly spaced interior knots for the B-spline expansion 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 an inner product $\langle \cdot, \cdot \rangle$. The term ε denotes the random error, $\theta_0 \in \mathcal{H}$ is the unknown functional index and $r(\cdot)$ denotes the unknown smooth link function; 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 using

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

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.

If the argument `y.test` is provided to the program (i. e. `if(!is.null(y.test))`), the function calculates the mean squared error of prediction (see the value `MSE.test`). This is computed 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`, `fsim.kernel.fit.optim` 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.
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=4)
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=4,nknot=20,
        range.grid=c(850,1050))

#MSEP
test$MSE.test
```

`fsim.kNN.fit` *Functional single-index model fit using kNN estimation and joint LOOCV minimisation*

Description

This function fits a functional single-index model (FSIM) between a functional covariate and a scalar response. It employs kNN estimation with Nadaraya-Watson weights and uses B-spline expansions to represent curves and eligible functional indexes.

The function also utilises the leave-one-out cross-validation (LOOCV) criterion to select the number of neighbours (`k.opt`) and the coefficients of the functional index in the spline basis (`theta.est`). It performs a joint minimisation of the LOOCV objective function in both the number of neighbours and the functional index.

Usage

```
fsim.kNN.fit(x, y, seed.coeff = c(-1, 0, 1), order.Bspline = 3, nknot.theta = 3,
knearest = NULL, min.knn = 2, max.knn = NULL,  step = NULL,
kind.of.kernel = "quad", range.grid = NULL, nknot = NULL, n.core = 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).
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 regularly spaced interior knots in the B-spline expansion of θ_0. The default is 3.
knearest	Vector of positive integers that defines the sequence within which the optimal number of nearest neighbours k.opt is selected. If knearest=NULL, then knearest <- seq(from =min.knn, to = max.knn, by = step).
min.knn	A positive integer that represents the minimum value in the sequence for selecting the number of nearest neighbours k.opt. This value should be less than the sample size. The default is 2.
max.knn	A positive integer that represents the maximum value in the sequence for selecting number of nearest neighbours k.opt. This value should be less than the sample size. The default is max.knn <- n%/%5.
step	A positive integer used to construct the sequence of k-nearest neighbours as follows: min.knn, min.knn + step, min.knn + 2*step, min.knn + 3*step,.... The default value for step is step<-ceiling(n/100).
kind.of.kernel	The type of kernel function used. Currently, 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 discretisation). If range.grid=NULL, then range.grid=c(1,p) is considered, where p is the discretisation size of x (i.e. ncol(x)).
nknot	Positive integer indicating the number of interior knots for the B-spline expansion of the functional covariate. The default value is (p - order.Bspline - 1)%/%2.
n.core	Number of CPU cores designated for parallel execution.The default is n.core<-availableCores(omit=1

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 an inner product $\langle \cdot, \cdot \rangle$. The term 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

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

where

- the positive integer k is a smoothing factor, representing 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.
- $H_{k,x,\hat{\theta}} = \min\{h \in R^+ \text{ such that } \sum_{i=1}^{n} 1_{B_{\hat{\theta}}(x,h)}(X_i) = k\}$, where $1_{B_{\hat{\theta}}(x,h)}(\cdot)$ is the indicator function of the open ball defined by 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 expansions to represent curves (dimension `nknot+order.Bspline`) and eligible functional indexes (dimension `nknot.theta+order.Bspline`). Then, we build a set Θ_n of eligible functional indexes by calibrating (to ensure the identifiability of the model) the set of initial coefficients given in `seed.coeff`. The larger this set is, the greater the size of Θ_n. Since our approach requires intensive computation, a trade-off between the size of Θ_n and the performance of the estimator is necessary. 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 LOOCV criterion. This function performs a joint minimisation in both parameters, the number of neighbours and the functional index, and supports parallel computation. To avoid parallel computation, we can set `n.core=1`.

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 procedures `fsim.kernel.fit`, `fsim.kNN.fit.optim` and `fsim.kernel.fit.optim`

Examples

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

#FSIM fit.
ptm<-proc.time()
fit<-fsim.kNN.fit(y=y[1:160],x=X[1:160,],max.knn=20,nknot.theta=4,nknot=20,
range.grid=c(850,1050))
proc.time()-ptm
fit
names(fit)
```

`fsim.kNN.fit.optim` *Functional single-index model fit using kNN estimation and iterative LOOCV minimisation*

Description

This function fits a functional single-index model (FSIM) between a functional covariate and a scalar response. It employs kNN estimation with Nadaraya-Watson weights and uses B-spline expansions to represent curves and eligible functional indexes.

The function also utilises the leave-one-out cross-validation (LOOCV) criterion to select the bandwidth (`h.opt`) and the coefficients of the functional index in the spline basis (`theta.est`). It performs an iterative minimisation of the LOOCV objective function, starting from an initial set of coefficients (`gamma`) for the functional index.

Usage

```
fsim.kNN.fit.optim(x, y, order.Bspline = 3, nknot.theta = 3, gamma = NULL,
knearest = NULL, min.knn = 2, max.knn = NULL,  step = NULL,
kind.of.kernel = "quad", range.grid = NULL, nknot = NULL, threshold = 0.005)
```

Arguments

`x`	Matrix containing the observations of the functional covariate (i.e. curves) collected by row.
`y`	Vector containing the scalar response.
`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 regularly spaced interior knots in the B-spline expansion of θ_0. The default is 3.
`gamma`	Vector indicating the initial coefficients for the functional index used in the iterative procedure. By default, it is a vector of ones. The size of the vector is determined by the sum `nknot.theta+order.Bspline`.
`knearest`	Vector of positive integers that defines the sequence within which the optimal number of nearest neighbours `k.opt` is selected. If `knearest=NULL`, then `knearest <- seq(from =min.knn, to = max.knn, by = step)`.
`min.knn`	A positive integer that represents the minimum value in the sequence for selecting the number of nearest neighbours `k.opt`. This value should be less than the sample size. The default is 2.
`max.knn`	A positive integer that represents the maximum value in the sequence for selecting number of nearest neighbours `k.opt`. This value should be less than the sample size. The default is `max.knn <- n%/%5`.
`step`	A positive integer used to construct the sequence of k-nearest neighbours as follows: `min.knn`, `min.knn + step`, `min.knn + 2*step`, `min.knn + 3*step`, ... The default value for `step` is `step<-ceiling(n/100)`.

`kind.of.kernel` The type of kernel function used. Currently, 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 discretisation). If `range.grid=NULL`, then `range.grid=c(1,p)` is considered, where `p` is the discretisation size of `x` (i.e. `ncol(x)`).

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

`threshold` The convergence threshold for the LOOCV function (scaled by the variance of the response). The default is `5e-3`.

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 an inner product $\langle \cdot, \cdot \rangle$. The term 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

- the positive integer k is a smoothing factor, representing 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 and $\hat{\theta}$ is an estimate of θ_0.
- $H_{k,x,\hat{\theta}} = \min\{h \in R^+ \text{ such that } \sum_{i=1}^{n} 1_{B_{\hat{\theta}}(x,h)}(X_i) = k\}$, where $1_{B_{\hat{\theta}}(x,h)}(\cdot)$ is the indicator function of the open ball defined by 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 expansions to represent curves (dimension `nknot+order.Bspline`) and eligible functional indexes (dimension `nknot.theta+order.Bspline`). 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 LOOCV criterion. This function performs an iterative minimisation procedure, starting from an initial set of coefficients (`gamma`) for the functional index. Given a functional index, the optimal number of neighbours according to the LOOCV criterion is selected. For a given number of neighbours, the minimisation in the functional index is performed using the R function `optim`. The procedure is iterated until convergence. For details, see Ferraty et al. (2013).

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 neighbours.
r.squared	Coefficient of determination.
var.res	Redidual variance.
df	Residual degrees of freedom.
CV.opt	Minimum value of the LOOCV function, i.e. the value of LOOCV for theta.est and k.opt.
err	Value of the LOOCV function divided by var(y) for each interaction.
H	Hat matrix.
k.seq	Sequence of eligible values for k.
CV.hseq	CV values for each k.
...	

Author(s)

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

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

References

Ferraty, F., Goia, A., Salinelli, E., and Vieu, P. (2013) Functional projection pursuit regression. *Test*, **22**, 293–320, doi:10.1007/s1174901203062.

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 predict.fsim.kNN and plot.fsim.kNN.

Alternative procedures fsim.kernel.fit.optim, fsim.kernel.fit and fsim.kNN.fit.

Examples

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

#FSIM fit.
ptm<-proc.time()
fit<-fsim.kNN.fit.optim(y=y[1:160],x=X[1:160,],max.knn=20,nknot.theta=4,nknot=20,
```

```
range.grid=c(850,1050))
proc.time()-ptm
fit
names(fit)
```

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

Description

This function computes predictions for a functional single-index model (FSIM) with a scalar response, which is estimated using the Nadaraya-Watson kNN estimator. It requires a functional index (θ), a global bandwidth (`h`), and the new observations of the functional covariate (`x.test`) as inputs.

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 in 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 in the the testing sample, collected by row.
`y.test`	(optional) Vector or matrix containing the scalar responses in the testing sample.
`theta`	Vector containing the coefficients of θ in a B-spline basis, such that `length(theta)=order.Bspline+nkn`
`nknot.theta`	Number of regularly spaced interior knots in the B-spline expansion of θ_0. The default is 3.
`order.Bspline`	Order of the B-spline basis functions. This is the number of coefficients in each piecewise polynomial segment. The default is 3
`k`	The number of nearest neighbours. The default is 4.
`kind.of.kernel`	The type of kernel function used. Currently, 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 discretisation). If `range.grid=NULL`, then `range.grid=c(1,p)` is considered, where `p` is the discretisation size of `x` (i.e. `ncol(x)`).
`nknot`	Number of regularly spaced interior knots for the B-spline expansion 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 an inner product $\langle \cdot, \cdot \rangle$. The term ε denotes the random error, $\theta_0 \in \mathcal{H}$ is the unknown functional index and $r(\cdot)$ denotes the unknown smooth link function; 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.
- $H_{k,x,\theta} = \min\left\{h \in R^+ \text{ such that } \sum_{i=1}^{n} 1_{B_\theta(x,h)}(X_i) = k\right\}$, where $1_{B_\theta(x,h)}(\cdot)$ is the indicator function of the open ball defined by the projection semi-metric, with centre $x \in \mathcal{H}$ and radius h.

If the argument `y.test` is provided to the program (i. e. `if(!is.null(y.test))`), the function calculates the mean squared error of prediction (see the value `MSE.test`). This is computed 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`, `fsim.kNN.fit.optim` 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.
ptm<-proc.time()
fit<-fsim.kNN.fit(y=y[train],x=X[train,],max.knn=20,nknot.theta=4,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=4,nknot=20,
        range.grid=c(850,1050))

#MSEP
test$MSE.test
```

`IASSMR.kernel.fit` *Impact point selection with IASSMR and kernel estimation*

Description

This function implements the Improved Algorithm for Sparse Semiparametric Multi-functional Regression (IASSMR) with kernel estimation. This algorithm is specifically designed for estimating multi-functional partial linear single-index models, which incorporate multiple scalar variables and a functional covariate as predictors. These scalar variables are derived from the discretisation of a curve and have linear effects while the functional covariate exhibits a single-index effect.

IASSMR is a two-stage procedure that selects the impact points of the discretised curve and estimates the model. The algorithm employs a penalised least-squares regularisation procedure, integrated with kernel estimation using Nadaraya-Watson weights. It uses B-spline expansions to represent curves and eligible functional indexes. Additionally, it utilises an objective criterion (`criterion`) to determine the initial 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,
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 = "GCV", penalty = "grSCAD",
max.iter = 1000, n.core = NULL)
```

Arguments

x	Matrix containing the observations of the functional covariate (functional single-index component), collected by row .
z	Matrix containing the observations of the functional covariate that is discretised (linear component), collected by row.
y	Vector containing the scalar response.
train.1	Positions of the data that are used as the training sample in the 1st step. The default setting is `train.1<-1:ceiling(n/2)`.
train.2	Positions of the data that are used as the training sample in the 2nd step. The default setting 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 regularly spaced interior knots in the B-spline expansion of θ_0. The default is 3.
min.q.h	Minimum quantile order of the distances between curves, which are computed using the projection semi-metric. This value determines the lower endpoint of the range from which the bandwidth is selected. The default is 0.05.
max.q.h	Maximum quantile order of the distances between curves, which are computed using the projection semi-metric. This value determines the upper endpoint of the range from 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 bandwidths 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 discretisation). If `range.grid=NULL`, then `range.grid=c(1,p)` is considered, where `p` is the discretisation size of `x` (i.e. `ncol(x)`).
kind.of.kernel	The type of kernel function used. Currently, only Epanechnikov kernel (`"quad"`) is available.

nknot	Positive integer indicating the number of interior knots for the B-spline expansion of the functional covariate. The default value is (p - order.Bspline - 1)%/%2.
lambda.min	The smallest value for lambda (i. e., the lower endpoint of the sequence in which lambda.opt is selected), as fraction of lambda.max. The defaults is lambda.min.l if the sample size is larger than factor.pn times the number of linear covariates and lambda.min.h otherwise.
lambda.min.h	The lower endpoint of the sequence in which lambda.opt is selected if the sample size is smaller than factor.pn times the number of linear covariates. The default is 0.05.
lambda.min.l	The lower endpoint of the sequence in which lambda.opt is selected if the sample size is larger than factor.pn times the number of linear 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 in the sequence from 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), resulting in the individual penalization of each scalar covariate.
nfolds	Number of cross-validation folds (used when criterion="k-fold-CV"). Default is 10.
seed	You may set the seed for the random number generator to ensure reproducible results (applicable when criterion="k-fold-CV" is used). The default seed value is 123.
wn	A vector of positive integers indicating the eligible number of covariates in the reduced model. For more information, refer to the section Details. The default is c(10,15,20).
criterion	The criterion used to select the tuning and regularisation parameters: wn.opt, lambda.opt and h.opt (also vn.opt if needed). Options include "GCV", "BIC", "AIC", or "k-fold-CV". The default setting is "GCV".
penalty	The penalty function applied in the penalised least-squares procedure. Currently, only "grLasso" and "grSCAD" are implemented. The default is "grSCAD".
max.iter	Maximum number of iterations allowed across the entire path. The default value is 1000.
n.core	Number of CPU cores designated for parallel execution. The default is n.core<-availableCores(omit=

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 represents 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 assumed to be a curve defined on the interval $[a, b]$, observed at the points $a \leq t_1 < \cdots < t_{p_n} \leq b$.
- $\beta_0 = (\beta_{01}, \ldots, \beta_{0p_n})^\top$ 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, it is assumed that only a few scalar variables from the set $\{\zeta(t_1), \ldots, \zeta(t_{p_n})\}$ are part of the model. Therefore, the relevant variables in the linear component (the impact points of the curve ζ on the response) must be selected, and the model estimated.

In this function, the MFPLSIM is fitted using the IASSMR. The IASSMR is a two-step procedure. For this, we divide the sample into two independent subsamples, each asymptotically half the size of the original sample ($n_1 \sim n_2 \sim n/2$). One subsample is used in the first stage of the method, and the other in the second stage.The subsamples are defined as follows:

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

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

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

To explain the algorithm, we assume 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 being integers.

1. **First step**. The FASSMR (see `FASSMR.kernel.fit`) combined with kernel estimation is applied 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 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 the real number z.The size (cardinality) 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 using the function `sfplsim.kernel.fit`, which requires the remaining arguments (see `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, along with those in their neighborhood, are included. The penalised least-squares procedure, combined with kernel estimation, is carried out again considering 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 $r_n = \sharp(\mathcal{R}_n^{\mathbf{2}})$, the variables in $\mathcal{R}_n^{\mathbf{2}}$ can be renamed 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 penalised least-squares variable selection procedure, with kernel estimation, is applied to this model using the function `sfplsim.kernel.fit`.

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

Remark: If the condition $p_n = w_n q_n$ is not met (then p_n/w_n is not an integer), the function considers variable $q_n = q_{n,k}$ values $k = 1, \ldots, w_n$. 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 the real number z.

The function supports parallel computation. To avoid it, we can set `n.core=1`.

Value

`call`	The matched call.
`fitted.values`	Estimated scalar response.
`residuals`	Differences between `y` and the `fitted.values`.
`beta.est`	$\hat{\beta}$ (i.e. estimate of β_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 $\beta_0^{\mathbf{2}}$ 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 entire set of p_n) used to build $\mathcal{R}_n^{\mathbf{2}}$ for each value of the sequence `wn`.
`beta1`	Estimate of $\beta_0^{\mathbf{1}}$ 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`.
`...`	Further outputs to apply S3 methods.

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, nknot=20,
        criterion="BIC", max.iter=5000)
proc.time()-ptm

fit
names(fit)
```

`IASSMR.kNN.fit` *Impact point selection with IASSMR and kNN estimation*

Description

This function implements the Improved Algorithm for Sparse Semiparametric Multi-functional Regression (IASSMR) with kNN estimation. This algorithm is specifically designed for estimating multi-functional partial linear single-index models, which incorporate multiple scalar variables and a functional covariate as predictors. These scalar variables are derived from the discretisation of a curve and have linear effects while the functional covariate exhibits a single-index effect.

IASSMR is a two-stage procedure that selects the impact points of the discretised curve and estimates the model. The algorithm employs a penalised least-squares regularisation procedure, integrated with kNN estimation using Nadaraya-Watson weights. It uses B-spline expansions to represent curves and eligible functional indexes. Additionally, it utilises an objective criterion (`criterion`) to determine the initial 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, 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 = "GCV", penalty = "grSCAD",
max.iter = 1000, n.core = NULL)
```

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	Positions of the data that are used as the training sample in the 1st step. The default setting is train.1<-1:ceiling(n/2).
train.2	Positions of the data that are used as the training sample in the 2nd step. The default setting 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 regularly spaced interior knots in the B-spline expansion of θ_0. The default is 3.
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	A positive integer that represents the minimum value in the sequence for selecting the number of nearest neighbours k.opt. This value should be less than the sample size. The default is 2.
max.knn	A positive integer that represents the maximum value in the sequence for selecting number of nearest neighbours k.opt. This value should be less than the sample size. The default is max.knn <- n%/%5.
step	A positive integer used to construct the sequence of k-nearest neighbours as follows: min.knn, min.knn + step, min.knn + 2*step, min.knn + 3*step,.... The default value for step 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 discretisation). If range.grid=NULL, then range.grid=c(1,p) is considered, where p is the discretisation size of x (i.e. ncol(x)).
kind.of.kernel	The type of kernel function used. Currently, only Epanechnikov kernel ("quad") is available.

nknot	Positive integer indicating the number of interior knots for the B-spline expansion of the functional covariate. The default value is (p - order.Bspline - 1)%/%2.
lambda.min	The smallest value for lambda (i. e., the lower endpoint of the sequence in which lambda.opt is selected), as fraction of lambda.max. The defaults is lambda.min.l if the sample size is larger than factor.pn times the number of linear covariates and lambda.min.h otherwise.
lambda.min.h	The lower endpoint of the sequence in which lambda.opt is selected if the sample size is smaller than factor.pn times the number of linear covariates. The default is 0.05.
lambda.min.l	The lower endpoint of the sequence in which lambda.opt is selected if the sample size is larger than factor.pn times the number of linear 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 in the sequence from 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), resulting in the individual penalization of each scalar covariate.
nfolds	Number of cross-validation folds (used when criterion="k-fold-CV"). Default is 10.
seed	You may set the seed for the random number generator to ensure reproducible results (applicable when criterion="k-fold-CV" is used). The default seed value is 123.
wn	A vector of positive integers indicating the eligible number of covariates in the reduced model. For more information, refer to the section Details. The default is c(10,15,20).
criterion	The criterion used to select the tuning and regularisation parameters: wn.opt, lambda.opt and k.opt (also vn.opt if needed). Options include "GCV", "BIC", "AIC", or "k-fold-CV". The default setting is "GCV".
penalty	The penalty function applied in the penalised least-squares procedure. Currently, only "grLasso" and "grSCAD" are implemented. The default is "grSCAD".
max.iter	Maximum number of iterations allowed across the entire path. The default value is 1000.
n.core	Number of CPU cores designated for parallel execution. The default is n.core<-availableCores(omit=

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 represents 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 assumed to be a curve defined on the interval $[a, b]$, observed at the points $a \leq t_1 < \cdots < t_{p_n} \leq b$.
- $\beta_0 = (\beta_{01}, \ldots, \beta_{0p_n})^\top$ 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, it is assumed that only a few scalar variables from the set $\{\zeta(t_1), \ldots, \zeta(t_{p_n})\}$ are part of the model. Therefore, the relevant variables in the linear component (the impact points of the curve ζ on the response) must be selected, and the model estimated.

In this function, the MFPLSIM is fitted using the IASSMR. The IASSMR is a two-step procedure. For this, we divide the sample into two independent subsamples, each asymptotically half the size of the original ($n_1 \sim n_2 \sim n/2$). One subsample is used in the first stage of the method, and the other in the second stage.The subsamples are defined as follows:

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

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

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

To explain the algorithm, we assume 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 being integers.

1. **First step**. The FASSMR (see `FASSMR.kNN.fit`) combined with kNN estimation is applied 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 entire 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 the real number z.The size (cardinality) 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 using the function `sfplsim.kNN.fit`, which requires the remaining arguments (see `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, along with those in their neighborhood, are included. The penalised least-squares procedure, combined with kNN estimation, is carried out again considering 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 $r_n = \sharp(\mathcal{R}_n^{\mathbf{2}})$, the variables in $\mathcal{R}_n^{\mathbf{2}}$ can be renamed 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 penalised least-squares variable selection procedure, with kNN estimation, is applied to this model using the function `sfplsim.kNN.fit`.

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

Remark: If the condition $p_n = w_n q_n$ is not met (then p_n/w_n is not an integer number), the function considers variable $q_n = q_{n,k}$ values $k = 1,\ldots,w_n$. 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 the real number z.

The function supports parallel computation. To avoid it, we can set `n.core=1`.

Value

`call`	The matched call.
`fitted.values`	Estimated scalar response.
`residuals`	Differences between `y` and the `fitted.values`.
`beta.est`	$\hat{\beta}$ (i.e. estimate of β_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 θ_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 $\beta_0^{\mathbf{2}}$ 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 entire set of p_n) used to build $\mathcal{R}_n^{\mathbf{2}}$ for each value of the sequence `wn`.

`beta1` Estimate of $\beta_0^{\mathbf{1}}$ 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", max.iter=5000)
proc.time()-ptm

fit
names(fit)
```

`lm.pels.fit` *Regularised fit of sparse linear regression*

Description

This function fits a sparse linear model between a scalar response and a vector of scalar covariates. It employs a penalised least-squares regularisation procedure, with either (group)SCAD or (group)LASSO penalties. The method utilises an objective criterion (`criterion`) to select the optimal regularisation 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 = "GCV", penalty = "grSCAD", 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 lower endpoint of the sequence in which `lambda.opt` is selected), as fraction of `lambda.max`. The defaults is `lambda.min.l` if the sample size is larger than `factor.pn` times the number of linear covariates and `lambda.min.h` otherwise.

`lambda.min.h` The lower endpoint of the sequence in which `lambda.opt` is selected if the sample size is smaller than `factor.pn` times the number of linear covariates. The default is 0.05.

`lambda.min.l` The lower endpoint of the sequence in which `lambda.opt` is selected if the sample size is larger than `factor.pn` times the number of linear 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 in the sequence from 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)`, resulting in the individual penalization of each scalar covariate.

`nfolds` Number of cross-validation folds (used when `criterion="k-fold-CV"`). Default is 10.

`seed` You may set the seed for the random number generator to ensure reproducible results (applicable when `criterion="k-fold-CV"` is used). The default seed value is 123.

`criterion` The criterion used to select the regularisation parameter `lambda.opt` (also `vn.opt` if needed). Options include `"GCV"`, `"BIC"`, `"AIC"`, or `"k-fold-CV"`. The default setting is `"GCV"`.

`penalty` The penalty function applied in the penalised least-squares procedure. Currently, only "grLasso" and "grSCAD" are implemented. The default is "grSCAD".

`max.iter` Maximum number of iterations allowed across the entire path. The default value 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 real covariates. In this equation, $\beta_0 = (\beta_{01}, \ldots, \beta_{0p_n})^\top$ is a vector of unknown real parameters and ε_i represents the random error.

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

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

where $\beta = (\beta_1, \ldots, \beta_{p_n})^\top$, $\mathcal{P}_{\lambda_{j_n}}(\cdot)$ is a penalty function (specified in the argument `penalty`) and $\lambda_{j_n} > 0$ is a tuning parameter. To reduce the number 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. The parameter λ is selected using the objetive criterion specified in the argument `criterion`.

For further details on the estimation procedure of the SLM, see e.g. Fan and Li. (2001). The PeLS objective function is minimised using the R function `grpreg` of the package `grpreg` (Breheny and Huang, 2015).

Remark: It should be noted that if we set `lambda.seq` to $= 0$, we obtain the non-penalised estimation of the model, i.e. the OLS estimation. Using `lambda.seq` with a vaule $\neq 0$ is advisable when suspecting the presence of 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

Breheny, P., and Huang, J. (2015) Group descent algorithms for nonconvex penalized linear and logistic regression models with grouped predictors. *Statistics and Computing*, **25**, 173–187, doi:10.1007/s1122201394242.

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.

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")
proc.time()-ptm

#Results
fit
names(fit)
```

`plot.classes` *Graphical representation of regression model outputs*

Description

`plot` functions to generate visual representations for the outputs of several fitting functions: `FASSMR.kernel.fit`, `FASSMR.kNN.fit`, `fsim.kernel.fit`, `fsim.kernel.fit.optim`, `fsim.kNN.fit`, `fsim.kNN.fit.optim`, `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,ind=1:10, size=15,col1=1,col2=2,col3=4,option=0,...)

## S3 method for class 'FASSMR.kNN'
plot(x,ind=1:10, size=15,col1=1,col2=2,col3=4,option=0, ...)

## S3 method for class 'fsim.kernel'
plot(x,size=15,col1=1,col2=2, ...)

## S3 method for class 'fsim.kNN'
plot(x,size=15,col1=1,col2=2,...)

## S3 method for class 'IASSMR.kernel'
plot(x,ind=1:10, size=15,col1=1,col2=2,col3=4,option=0, ...)
```

```
## S3 method for class 'IASSMR.kNN'
plot(x,ind=1:10, size=15,col1=1,col2=2,col3=4,option=0, ...)

## S3 method for class 'lm.pels'
plot(x,size=15,col1=1,col2=2,col3=4, ...)

## S3 method for class 'PVS'
plot(x,ind=1:10, size=15,col1=1,col2=2,col3=4,option=0, ...)

## S3 method for class 'PVS.kernel'
plot(x,ind=1:10, size=15,col1=1,col2=2,col3=4,option=0, ...)

## S3 method for class 'PVS.kNN'
plot(x,ind=1:10, size=15,col1=1,col2=2,col3=4,option=0, ...)

## S3 method for class 'sfpl.kernel'
plot(x,size=15,col1=1,col2=2,col3=4, ...)

## S3 method for class 'sfpl.kNN'
plot(x,size=15,col1=1,col2=2,col3=4, ...)

## S3 method for class 'sfplsim.kernel'
plot(x,size=15,col1=1,col2=2,col3=4, ...)

## S3 method for class 'sfplsim.kNN'
plot(x,size=15,col1=1,col2=2,col3=4, ...)
```

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).
ind	Indexes of the colors for the curves in the chart of estimated impact points. The default is 1:10
size	The size for title and axis labels in pts. The default is 15.
col1	Color of the points in the charts. Also, color of the estimated functional index representation. The default is black.
col2	Color of the nonparametric fit representation in FSIM functions, and of the straight line in 'Response vs Fitted Values' charts. The default is red.
col3	Color of the nonparametric fit of the residuals in 'Residuals vs Fitted Values' charts. The default is blue.
option	Selection of charts to be plotted. The default, option = 0, means all charts are plotted. See the section Details.
...	Further arguments passed to or from other methods.

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 `lm.pels`, `sfpl.kernel` and `sfpl.kNN`:
  1. The response over the `fitted.values`.
  2. The `residuals` over the `fitted.values`.
- For the classes `sfplsim.kernel` and `sfplsim.kNN`:
  1. The estimated functional index: $\hat{\theta}$.
  2. The response over the `fitted.values`.
  3. The `residuals` over the `fitted.values`.
- For the classes `FASSMR.kernel`, `FASSMR.kNN`, `IASSMR.kernel`, `IASSMR.kNN`, `sfplsim.kernel` and `sfplsim.kNN`:
  1. If `option=1`: The curves with the estimated impact points (in dashed vertical lines).
  2. If `option=2`: The estimated functional index: $\hat{\theta}$.
  3. If `option=3`:
     - The response over the `fitted.values`.
     - The `residuals` over the `fitted.values`.
  4. If `option=0`: All chart are plotted.
- For the classes `PVS`, `PVS.kNN`, `PVS.kernel`:
  1. If `option=1`: The curves with the estimated impact points (in dashed vertical lines).
  2. If `option=2`:
     - The response over the `fitted.values`.
     - The `residuals` over the `fitted.values`.
  3. If `option=0`: All chart are plotted.

All the routines implementing the plot S3 method use internally the R package `ggplot2` to produce elegant and high quality charts.

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 for FSIM*

Description

predict method for the functional single-index model (FSIM) fitted using fsim.kernel.fit, fsim.kernel.fit.optim, fsim.kNN.fit and fsim.kNN.fit.optim.

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, fsim.kernel.fit.optim, fsim.kNN.fit or fsim.kNN.fit.optim 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.
...	Further arguments passed to or from other methods.

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=20,
nknot.theta=4, 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 for MFPLSIM*

Description

predict method for the multi-functional partial linear single-index model (MFPLSIM) 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 derived 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$knearest.
min.knn.n	Only used for objects IASSMR.kNN if option=2 or option=3: minumum value of the sequence in which the number of neighbours k.opt is selected (thus, this number must be smaller than the sample size). The default is object$min.knn.
max.knn.n	Only used for objects IASSMR.kNN if option=2 or option=3: maximum value of the sequence in which the number of neighbours k.opt is selected (thus, this number must be larger than min.kNN and smaller than the sample size). The default is object$max.knn.
step.n	Only used for objects IASSMR.kNN if option=2 or option=3: positive integer used to build the sequence of k-nearest neighbours as follows: min.knn, min.knn + step.n, min.knn + 2*step.n, min.knn + 3*step.n,.... The default is object$step.

Details

Three options are provided to obtain the predictions of the response for newdata.x and newdata.z:

- If option=1, we maintain 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 may not be suitable to predict the functional single-index component of the model.
- If option=2, we maintain theta.est and beta.est, while the tuning parameter (h or k) is selected again to predict the functional single-index component of the model. This selection is performed using the leave-one-out 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 entire 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 and step.n.
- If option=3, we maintain 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 with option=1 for sfplsim.kernel or sfplsim.kNN class.
  - b) The prediction associated with 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 (and two MSEPs) are provided, corresponding to 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,  nknot=20,criterion="BIC",
            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",
          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 for linear models*

Description

predict method for:

- Linear model (LM) fitted using lm.pels.fit.
- Linear model with covariates derived 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 the scalar covariates resulting from the discretisation of a curve. Observations are collected by row.
y.test	(optional) A vector containing the new observations of the response.
...	Further arguments passed to or from other methods.

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), then 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.l=0.01,
      factor.pn=2, max.iter=5000, criterion="BIC")

#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", max.iter=5000)


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

`predict.mfplm.PVS` *Prediction for MFPLM*

Description

predict method for the multi-functional partial linear model (MFPLM) 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 derived from the discretisation of a curve, collected by row.
y.test	(optional) A vector containing the new observations of the response.
option	Allows the selection among the choices 1, 2 and 3 for PVS.kernel objects, and 1, 2, 3, and 4 for PVS.kNN objects. The default setting is 1. See the section Details.
...	Further arguments.
knearest.n	Only used for objects PVS.kNN if option=2, option=3 or option=4: sequence in which the number of nearest neighbours k.opt is selected. The default is object$knearest.
min.knn.n	Only used for objects PVS.kNN if option=2, option=3 or option=4: minumum value of the sequence in which the number of nearest neighbours k.opt is selected (thus, this number must be smaller than the sample size). The default is object$min.knn.
max.knn.n	Only used for objects PVS.kNN if option=2, option=3 or option=4: 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$max.knn.
step.n	Only used for objects PVS.kNN if option=2, option=3 or option=4: positive integer used to build the sequence of k-nearest neighbours in the following way: min.knn, min.knn + step.n, min.knn + 2*step.n, min.knn + 3*step.n, ... The default is object$step.

Details

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

- If `option=1`, we maintain 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` may not be suitable to predict the functional nonparametric component of the model.
- If `option=2`, we maintain `beta.est`, while the tuning parameter (h or k) is selected again to predict the functional nonparametric component of the model. This selection is performed using the leave-one-out cross-validation (LOOCV) criterion in the associated functional nonparametric model 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 entire training sample (not just a subsample of it), the sample size is modified and, as a consequence, the parameters `knearest`, `min.knn`, `max.knn`, and `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` and `step.mn`.
- If `option=3`, we maintain only the indexes of the relevant variables selected by the IASSMR. We estimate again the linear coefficients using `sfpl.kernel.fit` or `sfpl.kNN.fit`, respectively, without penalisation (setting `lambda.seq=0`) and using the entire training sample (`train=c(train.1,train.2)`). The method provides two predictions (and MSEPs):
  - a) The prediction associated with `option=1` for `sfpl.kernel` or `sfpl.kNN` class.
  - b) The prediction associated with `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` (an option only available for the class `PVS.kNN`) we maintain `beta.est`, while the tuning parameter k is selected again to predict the functional nonparametric component of the model. This selection is performed using LOOCV 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 (and two MSEPs) are provided, corresponding to the items a) and b) mentioned in the section `Details`. If `is.null(newdata.x)` or `is.null(newdata.z)`, then 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, nknot=20,criterion="BIC",
              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",
            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 for SFPLM*

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	Matrix containing new observations of the functional covariate 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 selection among the choices 1 and 2 for sfpl.kernel objects, and 1, 2 and 3 for sfpl.kNN objects. The default setting is 1. See the section Details.
...	Further arguments passed to or from other methods.

Details

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

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

In the case of sfpl.kNN objects if option=3, we maintain beta.est, while the tuning parameter k is seleted again to predict the functional nonparametric component of the model. This selection is performed using the LOOCV criterion in the associated functional nonparametric model, 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), then 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.l=0.01, factor.pn=2,
  criterion="BIC", range.grid=c(850,1050), nknot=20, max.iter=5000)
fit.kNN<-sfpl.kNN.fit(y=y[train],x=X[train,], z=z.com[train,],q=2,
  max.knn=20,lambda.min.l=0.01, factor.pn=2,
  criterion="BIC",range.grid=c(850,1050), 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 for SFPLSIM and MFPLSIM (using FASSMR)*

Description

`predict` S3 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 model (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.
...	Further arguments passed to or from other methods.

Details

Two options are provided to obtain the predictions of the response for newdata.x and newdata.z:

- If option=1, we maintain 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 maintain theta.est and beta.est, while the tuning parameter (h or k) is selected again to predict the functional single-index component of the model. This selection is performed using the leave-one-out cross-validation criterion in the associated functional single-index model.

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), then 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.l=0.01, factor.pn=2, nknot.theta=4,
            criterion="BIC", range.grid=c(850,1050),
            nknot=20, max.iter=5000)
s.fit.kNN<-sfplsim.kNN.fit(y=y[train],x=X[train,], z=z.com[train,],
        max.knn=20,lambda.min.l=0.01, factor.pn=2, nknot.theta=4,
        criterion="BIC",range.grid=c(850,1050),
        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.l=0.03, max.q.h=0.35,num.h = 10,
                  nknot=20,criterion="BIC", 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.l=0.03,
            max.knn=20,nknot=20,criterion="BIC",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` *Summarise information from FSIM estimation*

Description

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

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`, `fsim.kernel.fit.optim`, `fsim.kNN.fit` or `fsim.kNN.fit.optim` functions (i.e. an object of the class `fsim.kernel` or `fsim.kNN`).
`...`	Further arguments.
`object`	Output of the `fsim.kernel.fit`, `fsim.kernel.fit.optim`, `fsim.kNN.fit` or `fsim.kNN.fit.optim` 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` *Summarise information from linear models estimation*

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 penalised least-squares function. That is, the value obtained using `beta.est` and `lambda.opt`.
- 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 number of covariates required to construct the reduced model in the first step of the algorithm (`w.opt`). This value is selected using the same criterion employed for selecting 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` *Summarise information from MFPLM estimation*

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 penalised least-squares function. That is, the value obtained using beta.est and lambda.opt.
- The penalty function used.
- The criterion used to select the number of covariates employed to construct the reduced model, the tuning 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 *Summarise information from MFPLSIM estimation*

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).
...	Further arguments passed to or from other methods.
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 penalised least-squares function. That is, the value obtained using theta.est, beta.est and lambda.opt.
- The penalty function used.
- The criterion used to select the number of covariates employed to construct the reduced model, the tuning 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 *Summarise information from SFPLM estimation*

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 penalised least-squares function. That is, the value obtained using `beta.est` and `lambda.opt`.
- The penalty function used.
- The criterion used to select the tuning 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 *Summarise information from SFPLSIM estimation*

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 penalised least-squares function. That is, the value obtained using `theta.est`, `beta.est` and `lambda.opt`.
- The penalty function used.
- The criterion used to select the tuning 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`	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`	Number of regularly spaced interior knots of the B-spline basis. The default is 3.
`range.grid`	Vector of length 2 containing the range of the discretisation of the functional data. If `range.grid=NULL`, then `range.grid=c(1,p)` is considered, where `p` is the discretisation size of `data` (i.e. `ncol(data)`).
`nknot`	Number of regularly spaced interior knots for the B-spline representation of the functional data. The default value is `(p - order.Bspline - 1)%/%2`.

Value

A matrix containing 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 *Impact point selection with PVS*

Description

This function implements the Partitioning Variable Selection (PVS) algorithm. This algorithm is specifically designed for estimating multivarite linear models, where the scalar covariates are derived from the discretisation of a curve.

PVS is a two-stage procedure that selects the impact points of the discretised curve and estimates the model. The algorithm employs a penalised least-squares regularisation procedure. Additionally, it utilises an objective criterion (`criterion`) to determine the initial number of covariates in the reduced model (`w.opt`) of the first stage, and the penalisation parameter (`lambda.opt`).

Usage

```
PVS.fit(z, y, train.1 = NULL, train.2 = 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), range.grid = NULL,
criterion = "GCV", penalty = "grSCAD", max.iter = 1000)
```

Arguments

`z`	Matrix containing the observations of the functional covariate collected by row (linear component).
`y`	Vector containing the scalar response.
`train.1`	Positions of the data that are used as the training sample in the 1st step. The default setting is `train.1<-1:ceiling(n/2)`.
`train.2`	Positions of the data that are used as the training sample in the 2nd step. The default setting is `train.2<-(ceiling(n/2)+1):n`.
`lambda.min`	The smallest value for lambda (i. e., the lower endpoint of the sequence in which `lambda.opt` is selected), as fraction of `lambda.max`. The defaults is `lambda.min.l` if the sample size is larger than `factor.pn` times the number of linear covariates and `lambda.min.h` otherwise.
`lambda.min.h`	The lower endpoint of the sequence in which `lambda.opt` is selected if the sample size is smaller than `factor.pn` times the number of linear covariates. The default is 0.05.
`lambda.min.l`	The lower endpoint of the sequence in which `lambda.opt` is selected if the sample size is larger than `factor.pn` times the number of linear covariates. The default is 0.0001.
`factor.pn`	Positive integer used to set `lambda.min`. The default value is 1.
`nlambda`	Number of values in the sequence from 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)`, resulting in the individual penalization of each scalar covariate.

nfolds	Number of cross-validation folds (used when criterion="k-fold-CV"). Default is 10.
seed	You may set the seed for the random number generator to ensure reproducible results (applicable when criterion="k-fold-CV" is used). The default seed value is 123.
wn	A vector of positive integers indicating the eligible number of covariates in the reduced model. For more information, refer to the section Details. The default is c(10,15,20).
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 discretisation). If range.grid=NULL, then range.grid=c(1,p) is considered, where p is the discretisation size of x (i.e. ncol(x)).
criterion	The criterion used to select the tuning and regularisation parameters: wn.opt and lambda.opt (also vn.opt if needed). Options include "GCV", "BIC", "AIC", or "k-fold-CV". The default setting is "GCV".
penalty	The penalty function applied in the penalised least-squares procedure. Currently, only "grLasso" and "grSCAD" are implemented. The default is "grSCAD".
max.iter	Maximum number of iterations allowed across the entire path. The default value 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 assumed to be a random curve defined on some interval $[a, b]$, which is observed at the points $a \leq t_1 < \cdots < t_{p_n} \leq b$.
- $\beta_0 = (\beta_{01}, \ldots, \beta_{0p_n})^\top$ is a vector of unknown real coefficients.
- ε_i denotes the random error.

In this model, it is assumed that only a few scalar variables from the set $\{\zeta(t_1), \ldots, \zeta(t_{p_n})\}$ are part of the model. Therefore, the relevant variables (the impact points of the curve ζ on the response) must be selected, and the model estimated.

In this function, this model is fitted using the PVS. The PVS is a two-steps procedure. So we divide the sample into two independent subsamples, each asymptotically half the size of the original sample ($n_1 \sim n_2 \sim n/2$). One subsample is used in the first stage of the method, and the other in the second stage.The subsamples are defined as follows:

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

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

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

To explain the algorithm, we assume 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 being integers.

1. **First step**. A reduced model is considered, 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, containing only w_n equally spaced discretized observations of ζ covering the 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 the real number z. The size (cardinality) of this subset is provided to the program in the argument `wn` (which contains a sequence of eligible sizes).
   - Consider the following reduced model involving only the w_n linear covariates from $\mathcal{R}_n^{\mathbf{1}}$: $\mathcal{R}_n^{\mathbf{1}}$:
   $$Y_i = \sum_{k=1}^{w_n} \beta_{0k}^{\mathbf{1}} \zeta_i(t_k^{\mathbf{1}}) + \varepsilon_i^{\mathbf{1}}.$$
   The penalised least-squares variable selection procedure is applied to the reduced model using the function `lm.pels.fit`, which requires the remaining arguments (for details, see the documentation of the function `lm.pels.fit`). The estimates obtained are the outputs of the first step of the algorithm.
2. **Second step**. The variables selected in the first step, along with the variables in their neighborhood, are included. Then the penalised least-squares procedure is carried out again considering 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 penalised least-squares variable selection procedure is applied to this model using `lm.pels.fit`.

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

Remark: If the condition $p_n = w_n q_n$ is not met (then p_n/w_n is not an integer), the function considers variable $q_n = q_{n,k}$ values $k = 1, \ldots, w_n$. 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 the real number z.

Value

`call`	The matched call.
`fitted.values`	Estimated scalar response.
`residuals`	Differences between `y` and the `fitted.values`.
`beta.est`	$\hat{\beta}$ (i. e. estimate of β_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 $\beta_0^{\mathbf{2}}$ 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 entire set of p_n) used to build $\mathcal{R}_n^{\mathbf{2}}$ for each value of the sequence `wn`.
`beta1`	Estimate of $\beta_0^{\mathbf{1}}$ 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 entire 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", max.iter=5000)
proc.time()-ptm

fit
names(fit)
```

PVS.kernel.fit *Impact point selection with PVS and kernel estimation*

Description

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

PVS is a two-stage procedure that selects the impact points of the discretised curve and estimates the model. The algorithm employs a penalised least-squares regularisation procedure, integrated with kernel estimation with Nadaraya-Watson weights. Additionally, it utilises 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 = 1, nlambda = 100,
vn = ncol(z), nfolds = 10, seed = 123, wn = c(10, 15, 20), criterion = "GCV",
penalty = "grSCAD", max.iter = 1000)
```

Arguments

x	Matrix containing the observations of the functional covariate (functional non-parametric component), collected by row.
z	Matrix containing the observations of the functional covariate that is discretised (linear component), collected by row.
y	Vector containing the scalar response.
train.1	Positions of the data that are used as the training sample in the 1st step. The default setting is `train.1<-1:ceiling(n/2)`.
train.2	Positions of the data that are used as the training sample in the 2nd step. The default setting 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	Minimum quantile order of the distances between curves, which are computed using the projection semi-metric. This value determines the lower endpoint of the range from which the bandwidth is selected. The default is 0.05.
max.q.h	Maximum quantile order of the distances between curves, which are computed using the projection semi-metric. This value determines the upper endpoint of the range from 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 bandwidths 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 discretisation). If `range.grid=NULL`, then `range.grid=c(1,p)` is considered, where `p` is the discretisation size of `x` (i.e. `ncol(x)`).
kind.of.kernel	The type of kernel function used. Currently, only Epanechnikov kernel (`"quad"`) is available.
nknot	Positive integer indicating the number of interior knots for the B-spline expansion of the functional covariate. The default value is `(p - order.Bspline - 1)%/%2`.
lambda.min	The smallest value for lambda (i.e. the lower endpoint of the sequence in which `lambda.opt` is selected), as fraction of `lambda.max`. The defaults is `lambda.min.l` if the sample size is larger than `factor.pn` times the number of linear covariates and `lambda.min.h` otherwise.
lambda.min.h	The lower endpoint of the sequence in which `lambda.opt` is selected if the sample size is smaller than `factor.pn` times the number of linear covariates. The default is 0.05.

`lambda.min.l`	The lower endpoint of the sequence in which `lambda.opt` is selected if the sample size is larger than `factor.pn` times the number of linear 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 in the sequence from 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)`, resulting in the individual penalization of each scalar covariate.
`nfolds`	Number of cross-validation folds (used when `criterion="k-fold-CV"`). Default is 10.
`seed`	You may set the seed for the random number generator to ensure reproducible results (applicable when `criterion="k-fold-CV"` is used). The default seed value is 123.
`wn`	A vector of positive integers indicating the eligible number of covariates in the reduced model. For more information, refer to the section `Details`. The default is `c(10,15,20)`.
`criterion`	The criterion used to select the tuning and regularisation parameters: `wn.opt`, `lambda.opt` and `h.opt` (also `vn.opt` if needed). Options include `"GCV"`, `"BIC"`, `"AIC"`, or `"k-fold-CV"`. The default setting is `"GCV"`.
`penalty`	The penalty function applied in the penalised least-squares procedure. Currently, only "grLasso" and "grSCAD" are implemented. The default is "grSCAD".
`max.iter`	Maximum number of iterations allowed across the entire path. The default value 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 assumed to be a curve defined on some interval $[a, b]$, observed at the points $a \leq t_1 < \cdots < t_{p_n} \leq b$.
- $\beta_0 = (\beta_{01}, \ldots, \beta_{0p_n})^\top$ is a vector of unknown real coefficients and $m(\cdot)$ represents a smooth unknown real-valued link function.
- ε_i denotes the random error.

In the MFPLM, it is assumed that only a few scalar variables from the set $\{\zeta(t_1), \ldots, \zeta(t_{p_n})\}$ are part of the model. Therefore, the relevant variables in the linear component (the impact points of the curve ζ on the response) must be selected, and the model estimated.

In this function, the MFPLM is fitted using the PVS procedure, a two-step algorithm. For this, we divide the sample into two two independent subsamples (asymptotically of the same size $n_1 \sim$

$n_2 \sim n/2$). One subsample is used in the first stage of the method, and the other in the second stage.The subsamples are defined as follows:

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

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

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

To explain the algorithm, let's assume 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 being integers.

1. **First step**. A reduced model is considered, 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 containing only w_n equally spaced discretised observations of ζ covering the 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 the real number z. The size (cardinality) of this subset is provided to the program through the argument `wn`, which contains the sequence of eligible sizes.
   - Consider the following reduced model involving only the w_n linear covariates from $\mathcal{R}_n^{\mathbf{1}}$:

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

     The penalised least-squares variable selection procedure, with kernel estimation, is applied to the reduced model using 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, along with those in their neighborhood, are included. Then the penalised least-squares procedure, combined with kernel estimation, is carried out again, considering 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 penalised least-squares variable selection procedure, with kernel estimation, is applied to this model using `sfpl.kernel.fit`.

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

Remark: If the condition $p_n = w_n q_n$ is not met (then p_n/w_n is not an integer), the function considers variable $q_n = q_{n,k}$ values $k = 1, \ldots, w_n$. 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 the real number z.

Value

`call`	The matched call.
`fitted.values`	Estimated scalar response.
`residuals`	Differences between `y` and the `fitted.values`.
`beta.est`	$\hat{\beta}$ (i.e. estimate of β_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 $\beta_0^{\mathbf{2}}$ 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 entire set of p_n) used to construct $\mathcal{R}_n^{\mathbf{2}}$ for each value of the sequence `wn`.
`beta1`	Estimate of $\beta_0^{\mathbf{1}}$ 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 entire set of p_n) used to construct $\mathcal{R}_n^{\mathbf{1}}$ 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, nknot=20,
        criterion="BIC", max.iter=5000)
proc.time()-ptm

fit
names(fit)
```

PVS.kNN.fit *Impact point selection with PVS and kNN estimation*

Description

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

PVS is a two-stage procedure that selects the impact points of the discretised curve and estimates the model. The algorithm employs a penalised least-squares regularisation procedure, integrated with kNN estimation using Nadaraya-Watson weights. Additionally, it utilises 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

```
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 = "GCV",
penalty = "grSCAD", max.iter = 1000)
```

Arguments

`x`	Matrix containing the observations of the functional covariate (functional non-parametric component), collected by row.
`z`	Matrix containing the observations of the functional covariate that is discretised (linear component), collected by row.
`y`	Vector containing the scalar response.
`train.1`	Positions of the data that are used as the training sample in the 1st step. The default setting is `train.1<-1:ceiling(n/2)`.
`train.2`	Positions of the data that are used as the training sample in the 2nd step. The default setting is `train.2<-(ceiling(n/2)+1):n`.
`semimetric`	Semi-metric function. Currently, 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`	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`	A positive integer that represents the minimum value in the sequence for selecting the number of nearest neighbours `k.opt`. This value should be less than the sample size. The default is 2.

max.knn	A positive integer that represents the maximum value in the sequence for selecting number of nearest neighbours k.opt. This value should be less than the sample size. The default is max.knn <- n%/%5.
step	A positive integer used to construct the sequence of k-nearest neighbours as follows: min.knn, min.knn + step, min.knn + 2*step, min.knn + 3*step,.... The default value for step 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 discretisation). If range.grid=NULL, then range.grid=c(1,p) is considered, where p is the discretisation size of x (i.e. ncol(x)).
kind.of.kernel	The type of kernel function used. Currently, only Epanechnikov kernel ("quad") is available.
nknot	Positive integer indicating the number of interior knots for the B-spline expansion of the functional covariate. The default value is (p - order.Bspline - 1)%/%2.
lambda.min	The smallest value for lambda (i.e. the lower endpoint of the sequence in which lambda.opt is selected), as fraction of lambda.max. The defaults is lambda.min.l if the sample size is larger than factor.pn times the number of linear covariates and lambda.min.h otherwise.
lambda.min.h	The lower endpoint of the sequence in which lambda.opt is selected if the sample size is smaller than factor.pn times the number of linear covariates. The default is 0.05.
lambda.min.l	The lower endpoint of the sequence in which lambda.opt is selected if the sample size is larger than factor.pn times the number of linear 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 in the sequence from 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), resulting in the individual penalization of each scalar covariate.
nfolds	Number of cross-validation folds (used when criterion="k-fold-CV"). Default is 10.
seed	You may set the seed for the random number generator to ensure reproducible results (applicable when criterion="k-fold-CV" is used). The default seed value is 123.
wn	A vector of positive integers indicating the eligible number of covariates in the reduced model. For more information, refer to the section Details. The default is c(10,15,20).
criterion	The criterion used to select the tuning and regularisation parameters: wn.opt, lambda.opt and k.opt (also vn.opt if needed). Options include "GCV", "BIC", "AIC", or "k-fold-CV". The default setting is "GCV".
penalty	The penalty function applied in the penalised least-squares procedure. Currently, only "grLasso" and "grSCAD" are implemented. The default is "grSCAD".
max.iter	Maximum number of iterations allowed across the entire path. The default value 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 assumed to be a curve defined on some interval $[a, b]$, observed at the points $a \leq t_1 < \cdots < t_{p_n} \leq b$.
- $\beta_0 = (\beta_{01}, \ldots, \beta_{0p_n})^\top$ is a vector of unknown real coefficients and $m(\cdot)$ represents a smooth unknown real-valued link function.
- ε_i denotes the random error.

In the MFPLM, it is assumed that only a few scalar variables from the set $\{\zeta(t_1), \ldots, \zeta(t_{p_n})\}$ are part of the model. Therefore, the relevant variables in the linear component (the impact points of the curve ζ on the response) must be selected, and the model estimated.

In this function, the MFPLM is fitted using the PVS procedure, a two-step algorithm. For this, we divide the sample into two two independent subsamples (asymptotically of the same size $n_1 \sim n_2 \sim n/2$). One subsample is used in the first stage of the method, and the other in the second stage.The subsamples are defined as follows:

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

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

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

To explain the algorithm, let's assume 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 being integers.

1. **First step**. A reduced model is considered, 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 containing only w_n equally spaced discretised observations of ζ covering the 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 the real number z. The size (cardinality) of this subset is provided to the program through the argument `wn`, which contains the sequence of eligible sizes.
   - Consider the following reduced model involving only the w_n linear covariates from $\mathcal{R}_n^{\mathbf{1}}$:

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

The penalised least-squares variable selection procedure, with kNN estimation, is applied to the reduced model using 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, along with those in their neighborhood, are included. Then the penalised least-squares procedure, combined with kNN estimation, is carried out again, considering 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 penalised least-squares variable selection procedure, with kNN estimation, is applied to this model using `sfpl.kNN.fit`.

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

Remark: If the condition $p_n = w_n q_n$ is not met (then p_n/w_n is not an integer), the function considers variable $q_n = q_{n,k}$ values $k = 1, \ldots, w_n$. 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 the real number z.

Value

`call`	The matched call.
`fitted.values`	Estimated scalar response.
`residuals`	Differences between `y` and the `fitted.values`.
`beta.est`	$\hat{\beta}$ (i.e. estimate of β_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 $\beta_0^{\mathbf{2}}$ 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`.

`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 entire set of p_n) used to build $\mathcal{R}_n^{\mathbf{2}}$ for each value of the sequence `wn`.

`beta1` Estimate of $\beta_0^{\mathbf{1}}$ 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 entire 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",
        max.iter=5000)
proc.time()-ptm

fit
names(fit)
```

`semimetric.projec` *Projection semi-metric computation*

Description

Computes the projection semi-metric between each curve in `data1` and each curve in `data2`, given a functional index θ.

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	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	Number of regularly spaced interior knots of the B-spline basis. The default is 3.
range.grid	Vector of length 2 containing the range of the discretisation of the functional data. If range.grid=NULL, then range.grid=c(1,p) is considered, where p is the discretization size of data (i.e. ncol(data)).
nknot	Number of regularly spaced interior knots for the B-spline representation of the functional data. The default value is (p - order.Bspline - 1)%/%2.

Details

For $x_1, x_2 \in \mathcal{H}$,, where $\mathcal{H}$ is 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 this projection semi-metric using the B-spline representation of the curves and θ. The dimension of the B-spline basis for θ is determined by order.Bspline+nknot.theta.

Value

A matrix containing the projection semi-metrics for 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` *SFPLM regularised fit using kernel estimation*

Description

This function fits a sparse semi-functional partial linear model (SFPLM). It employs a penalised least-squares regularisation procedure, integrated with nonparametric kernel estimation using Nadaraya-Watson weights.

The procedure utilises an objective criterion (`criterion`) to select both the bandwidth (`h.opt`) and the regularisation 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 = 1, nlambda = 100, lambda.seq = NULL,
vn = ncol(z), nfolds = 10, seed = 123, criterion = "GCV", penalty = "grSCAD",
max.iter = 1000)
```

Arguments

`x`	Matrix containing the observations of the functional covariate (functional non-parametric component), collected by row.
`z`	Matrix containing the observations of the scalar covariates (linear component), 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`	Minimum quantile order of the distances between curves, which are computed using the projection semi-metric. This value determines the lower endpoint of the range from which the bandwidth is selected. The default is 0.05.
`max.q.h`	Maximum quantile order of the distances between curves, which are computed using the projection semi-metric. This value determines the upper endpoint of the range from 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 bandwidths 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 discretisation). If `range.grid=NULL`, then `range.grid=c(1,p)` is considered, where `p` is the discretisation size of `x` (i.e. `ncol(x)`).
`kind.of.kernel`	The type of kernel function used. Currently, only Epanechnikov kernel (`"quad"`) is available.
`nknot`	Positive integer indicating the number of interior knots for the B-spline expansion of the functional covariate. The default value is `(p - order.Bspline - 1)%/%2`.
`lambda.min`	The smallest value for lambda (i.e. the lower endpoint of the sequence in which `lambda.opt` is selected), as fraction of `lambda.max`. The defaults is `lambda.min.l` if the sample size is larger than `factor.pn` times the number of linear covariates and `lambda.min.h` otherwise.
`lambda.min.h`	The lower endpoint of the sequence in which `lambda.opt` is selected if the sample size is smaller than `factor.pn` times the number of linear covariates. The default is 0.05.
`lambda.min.l`	The lower endpoint of the sequence in which `lambda.opt` is selected if the sample size is larger than `factor.pn` times the number of linear 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 in the sequence from 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)`, resulting in the individual penalization of each scalar covariate.
`nfolds`	Number of cross-validation folds (used when `criterion="k-fold-CV"`). Default is 10.
`seed`	You may set the seed for the random number generator to ensure reproducible results (applicable when `criterion="k-fold-CV"` is used). The default seed value is 123.
`criterion`	The criterion used to select the tuning and regularisation parameter: `h.opt` and `lambda.opt` (also `vn.opt` if needed). Options include `"GCV"`, `"BIC"`, `"AIC"`, or `"k-fold-CV"`. The default setting is `"GCV"`.
`penalty`	The penalty function applied in the penalised least-squares procedure. Currently, only "grLasso" and "grSCAD" are implemented. The default is "grSCAD".
`max.iter`	Maximum number of iterations allowed across the entire path. The default value is 1000.

Details

The sparse semi-functional partial linear model (SFPLM) 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 real random covariates, and X_i is a functional random covariate valued in a semi-metric space $\mathcal{H}$. In this equation, $\beta_0 = (\beta_{01}, \ldots, \beta_{0p_n})^\top$ and $m(\cdot)$ represent a vector of unknown real parameters and an unknown smooth real-valued function, respectively. Additionally, ε_i is the random error.

In this function, the SFPLM is fitted using a penalised least-squares approach. The approach involves transforming the SFPLM 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 (for details, see Ferraty and Vieu, 2006). This transformation is achieved using kernel estimation with Nadaraya-Watson weights.

An approximate linear model is then obtained:

$$\widetilde{\mathbf{Y}} \approx \widetilde{\mathbf{Z}}\beta_0 + \varepsilon,$$

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

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

where $\beta = (\beta_1, \ldots, \beta_{p_n})^\top$, $\mathcal{P}_{\lambda_{j_n}}(\cdot)$ is a penalty function (specified in the argument `penalty`) and $\lambda_{j_n} > 0$ is a tuning parameter. To reduce the number 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 objective criterion specified in the argument `criterion`.

Finally, after estimating β_0 by minimising (1), we address the estimation of the nonlinear function $m(\cdot)$. For this, we again employ the kernel procedure with Nadaraya-Watson weights to smooth the partial residuals $Y_i - \mathbf{Z}_i^\top\widehat{\beta}$.

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

Remark: It should be noted that if we set `lambda.seq` to 0, we can obtain the non-penalised estimation of the model, i.e. the OLS estimation. Using `lambda.seq` with a value $\neq 0$ is advisable when suspecting the presence of 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.l=0.01,
      max.iter=5000, criterion="BIC", nknot=20)
proc.time()-ptm

#Results
fit
names(fit)
```

`sfpl.kNN.fit` *SFPLM regularised fit using kNN estimation*

Description

This function fits a sparse semi-functional partial linear model (SFPLM). It employs a penalised least-squares regularisation procedure, integrated with nonparametric kNN estimation using Nadaraya-Watson weights.

The procedure utilises an objective criterion (`criterion`) to select both the bandwidth (`h.opt`) and the regularisation 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 = "GCV", penalty = "grSCAD",
max.iter = 1000)
```

Arguments

`x`	Matrix containing the observations of the functional covariate (functional non-parametric component), collected by row.
`z`	Matrix containing the observations of the scalar covariates (linear component), 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`	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`	A positive integer that represents the minimum value in the sequence for selecting the number of nearest neighbours `k.opt`. This value should be less than the sample size. The default is 2.
`max.knn`	A positive integer that represents the maximum value in the sequence for selecting number of nearest neighbours `k.opt`. This value should be less than the sample size. The default is `max.knn <- n%/%5`.
`step`	A positive integer used to construct the sequence of k-nearest neighbours as follows: `min.knn`, `min.knn + step`, `min.knn + 2*step`, `min.knn + 3*step`, ... The default value for `step` 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 discretisation). If range.grid=NULL, then range.grid=c(1,p) is considered, where p is the discretisation size of x (i.e. ncol(x)).
kind.of.kernel	The type of kernel function used. Currently, only Epanechnikov kernel ("quad") is available.
nknot	Positive integer indicating the number of interior knots for the B-spline expansion of the functional covariate. The default value is (p - order.Bspline - 1)%/%2.
lambda.min	The smallest value for lambda (i.e. the lower endpoint of the sequence in which lambda.opt is selected), as fraction of lambda.max. The defaults is lambda.min.l if the sample size is larger than factor.pn times the number of linear covariates and lambda.min.h otherwise.
lambda.min.h	The lower endpoint of the sequence in which lambda.opt is selected if the sample size is smaller than factor.pn times the number of linear covariates. The default is 0.05.
lambda.min.l	The lower endpoint of the sequence in which lambda.opt is selected if the sample size is larger than factor.pn times the number of linear 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 in the sequence from 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), resulting in the individual penalization of each scalar covariate.
nfolds	Number of cross-validation folds (used when criterion="k-fold-CV"). Default is 10.
seed	You may set the seed for the random number generator to ensure reproducible results (applicable when criterion="k-fold-CV" is used). The default seed value is 123.
criterion	The criterion used to select the tuning and regularisation parameter: k.opt and lambda.opt (also vn.opt if needed). Options include "GCV", "BIC", "AIC", or "k-fold-CV". The default setting is "GCV".
penalty	The penalty function applied in the penalised least-squares procedure. Currently, only "grLasso" and "grSCAD" are implemented. The default is "grSCAD".
max.iter	Maximum number of iterations allowed across the entire path. The default value is 1000.

Details

The sparse semi-functional partial linear model (SFPLM) 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 real random covariates, and X_i is a functional random covariate valued in a semi-metric space $\mathcal{H}$. In this equation, $\beta_0 = (\beta_{01}, \ldots, \beta_{0p_n})^\top$ and $m(\cdot)$ represent a vector of unknown real parameters and an unknown smooth real-valued function, respectively. Additionally, ε_i is the random error.

In this function, the SFPLM is fitted using a penalised least-squares approach. The approach involves transforming the SFPLM 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 (for details, see Ferraty and Vieu, 2006). This transformation is achieved using kNN estimation with Nadaraya-Watson weights.

An approximate linear model is then obtained:

$$\widetilde{\mathbf{Y}} \approx \widetilde{\mathbf{Z}}\beta_0 + \varepsilon,$$

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

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

where $\beta = (\beta_1, \ldots, \beta_{p_n})^\top$, $\mathcal{P}_{\lambda_{j_n}}(\cdot)$ is a penalty function (specified in the argument `penalty`) and $\lambda_{j_n} > 0$ is a tuning parameter. To reduce the number 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 objective criterion specified in the argument `criterion`.

Finally, after estimating β_0 by minimising (1), we address the estimation of the nonlinear function $m(\cdot)$. For this, we again employ the kNN procedure with Nadaraya-Watson weights to smooth the partial residuals $Y_i - \mathbf{Z}_i^\top\widehat{\beta}$.

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

Remark: It should be noted that if we set `lambda.seq` to 0, we can obtain the non-penalised estimation of the model, i.e. the OLS estimation. Using `lambda.seq` with a value $\neq 0$ is advisable when suspecting the presence of 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.l=0.01, criterion="BIC",
  range.grid=c(850,1050), nknot=20, max.iter=5000)
proc.time()-ptm

#Results
fit
names(fit)
```

---

sfplsim.kernel.fit *SFPLSIM regularised fit using kernel estimation*

Description

This function fits a sparse semi-functional partial linear single-index (SFPLSIM). It employs a penalised least-squares regularisation procedure, integrated with nonparametric kernel estimation using Nadaraya-Watson weights.

The function uses B-spline expansions to represent curves and eligible functional indexes. It also utilises an objective criterion (`criterion`) to select both the bandwidth (`h.opt`) and the regularisation parameter (`lambda.opt`).

Usage

```
sfplsim.kernel.fit(x, z, y, seed.coeff = c(-1, 0, 1), order.Bspline = 3,
nknot.theta = 3, 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 = "GCV",
penalty = "grSCAD", max.iter = 1000, n.core = NULL)
```

Arguments

x	Matrix containing the observations of the functional covariate (functional single-index component), collected by row.
z	Matrix containing the observations of the scalar covariates (linear component), 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 regularly spaced interior knots in the B-spline expansion of θ_0. The default is 3.
min.q.h	Minimum quantile order of the distances between curves, which are computed using the projection semi-metric. This value determines the lower endpoint of the range from which the bandwidth is selected. The default is 0.05.
max.q.h	Maximum quantile order of the distances between curves, which are computed using the projection semi-metric. This value determines the upper endpoint of the range from 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 bandwidths 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 discretisation). If range.grid=NULL, then range.grid=c(1,p) is considered, where p is the discretisation size of x (i.e. ncol(x)).
kind.of.kernel	The type of kernel function used. Currently, only Epanechnikov kernel ("quad") is available.
nknot	Positive integer indicating the number of interior knots for the B-spline expansion of the functional covariate. The default value is (p - order.Bspline - 1)%/%2.
lambda.min	The smallest value for lambda (i. e., the lower endpoint of the sequence in which lambda.opt is selected), as fraction of lambda.max. The defaults is lambda.min.l if the sample size is larger than factor.pn times the number of linear covariates and lambda.min.h otherwise.

`lambda.min.h`	The lower endpoint of the sequence in which `lambda.opt` is selected if the sample size is smaller than `factor.pn` times the number of linear covariates. The default is 0.05.
`lambda.min.l`	The lower endpoint of the sequence in which `lambda.opt` is selected if the sample size is larger than `factor.pn` times the number of linear 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 in the sequence from 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)`, resulting in the individual penalization of each scalar covariate.
`nfolds`	Number of cross-validation folds (used when `criterion="k-fold-CV"`). Default is 10.
`seed`	You may set the seed for the random number generator to ensure reproducible results (applicable when `criterion="k-fold-CV"` is used). The default seed value is 123.
`criterion`	The criterion used to select the tuning and regularisation parameter: `h.opt` and `lambda.opt` (also `vn.opt` if needed). Options include `"GCV"`, `"BIC"`, `"AIC"`, or `"k-fold-CV"`. The default setting is `"GCV"`.
`penalty`	The penalty function applied in the penalised least-squares procedure. Currently, only "grLasso" and "grSCAD" are implemented. The default is "grSCAD".
`max.iter`	Maximum number of iterations allowed across the entire path. The default value is 1000.
`n.core`	Number of CPU cores designated for parallel execution. The default is `n.core<-availableCores(omit=`

Details

The sparse semi-functional partial linear single-index model (SFPLSIM) 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 real random covariates 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, $\beta_0 = (\beta_{01}, \ldots, \beta_{0p_n})^\top$, $\theta_0 \in \mathcal{H}$ 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 sparse SFPLSIM is fitted using the penalised least-squares approach. The first step 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 transformation is achieved using nonparametric kernel estimation (see, for details, the documentation of the function `fsim.kernel.fit`).

An approximate linear model is then obtained:

$$\widetilde{\mathbf{Y}}_{\theta_0} \approx \widetilde{\mathbf{Z}}_{\theta_0}\beta_0 + \varepsilon,$$

and the penalised least-squares procedure is applied to this model by minimising over the pair (β, θ)

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

where $\beta = (\beta_1, \ldots, \beta_{p_n})^\top$, $\mathcal{P}_{\lambda_{j_n}}(\cdot)$ 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 a B-spline representation to construct 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 greater the size of Θ_n. ue to the intensive computation required by our approach, a balance between the size of Θ_n and the performance of the estimator is necessary. 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 β_0 and θ_0 by minimising (1), we proceed to estimate the nonlinear function $r_{\theta_0}(\cdot) \equiv r(\langle\theta_0, \cdot\rangle)$. For this purporse, we again apply the kernel procedure with Nadaraya-Watson weights to smooth the partial residuals $Y_i - \mathbf{Z}_i^\top\widehat{\beta}$.

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

Remark: It should be noted that if we set `lambda.seq` to 0, we can obtain the non-penalised estimation of the model, i.e. the OLS estimation. Using `lambda.seq` with a value $\neq 0$ is advisable when suspecting the presence of 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.l=0.01,
      max.iter=5000, nknot.theta=4,criterion="BIC",nknot=20)
proc.time()-ptm

#Results
fit
names(fit)
```

`sfplsim.kNN.fit` *SFPLSIM regularised fit using kNN estimation*

Description

This function fits a sparse semi-functional partial linear single-index (SFPLSIM). It employs a penalised least-squares regularisation procedure, integrated with nonparametric kNN estimation using Nadaraya-Watson weights.

The function uses B-spline expansions to represent curves and eligible functional indexes. It also utilises an objective criterion (`criterion`) to select both the number of neighbours (`k.opt`) and the regularisation parameter (`lambda.opt`).

Usage

```
sfplsim.kNN.fit(x, z, y, seed.coeff = c(-1, 0, 1), order.Bspline = 3,
nknot.theta = 3, 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 = "GCV",
penalty = "grSCAD", max.iter = 1000, n.core = NULL)
```

Arguments

`x`	Matrix containing the observations of the functional covariate (functional single-index component), collected by row.

z	Matrix containing the observations of the scalar covariates (linear component), 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 regularly spaced interior knots in the B-spline expansion of θ_0. The default is 3.
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	A positive integer that represents the minimum value in the sequence for selecting the number of nearest neighbours k.opt. This value should be less than the sample size. The default is 2.
max.knn	A positive integer that represents the maximum value in the sequence for selecting number of nearest neighbours k.opt. This value should be less than the sample size. The default is max.knn <- n%/%5.
step	A positive integer used to construct the sequence of k-nearest neighbours as follows: min.knn, min.knn + step, min.knn + 2*step, min.knn + 3*step,.... The default value for step 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 discretisation). If range.grid=NULL, then range.grid=c(1,p) is considered, where p is the discretisation size of x (i.e. ncol(x)).
kind.of.kernel	The type of kernel function used. Currently, only Epanechnikov kernel ("quad") is available.
nknot	Positive integer indicating the number of interior knots for the B-spline expansion of the functional covariate. The default value is (p - order.Bspline - 1)%/%2.
lambda.min	The smallest value for lambda (i. e., the lower endpoint of the sequence in which lambda.opt is selected), as fraction of lambda.max. The defaults is lambda.min.l if the sample size is larger than factor.pn times the number of linear covariates and lambda.min.h otherwise.
lambda.min.h	The lower endpoint of the sequence in which lambda.opt is selected if the sample size is smaller than factor.pn times the number of linear covariates. The default is 0.05.
lambda.min.l	The lower endpoint of the sequence in which lambda.opt is selected if the sample size is larger than factor.pn times the number of linear 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 in the sequence from 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), resulting in the individual penalization of each scalar covariate.
nfolds	Number of cross-validation folds (used when criterion="k-fold-CV"). Default is 10.
seed	You may set the seed for the random number generator to ensure reproducible results (applicable when criterion="k-fold-CV" is used). The default seed value is 123.
criterion	The criterion used to select the tuning and regularisation parameter: h.opt and lambda.opt (also vn.opt if needed). Options include "GCV", "BIC", "AIC", or "k-fold-CV". The default setting is "GCV".
penalty	The penalty function applied in the penalised least-squares procedure. Currently, only "grLasso" and "grSCAD" are implemented. The default is "grSCAD".
max.iter	Maximum number of iterations allowed across the entire path. The default value is 1000.
n.core	Number of CPU cores designated for parallel execution. The default is n.core<-availableCores(omit=

Details

The sparse semi-functional partial linear single-index model (SFPLSIM) 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 real random covariates 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, $\beta_0 = (\beta_{01}, \ldots, \beta_{0p_n})^\top$, $\theta_0 \in \mathcal{H}$ 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 sparse SFPLSIM is fitted using the penalised least-squares approach. The first step 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 transformation is achieved using nonparametric kNN estimation (see, for details, the documentation of the function fsim.kNN.fit).

An approximate linear model is then obtained:

$$\widetilde{\mathbf{Y}}_{\theta_0} \approx \widetilde{\mathbf{Z}}_{\theta_0}\beta_0 + \varepsilon,$$

and the penalised least-squares procedure is applied to this model by minimising over the pair (β, θ)

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

where $\beta = (\beta_1, \ldots, \beta_{p_n})^\top$, $\mathcal{P}_{\lambda_{j_n}}(\cdot)$ 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 a B-spline representation to construct 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 greater the size of Θ_n. ue to the intensive computation required by our approach, a balance between the size of Θ_n and the performance of the estimator is necessary. 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 β_0 and θ_0 by minimising (1), we proceed to estimate the nonlinear function $r_{\theta_0}(\cdot) \equiv r\left(\langle \theta_0, \cdot \rangle\right)$. For this purporse, we again apply the kNN procedure with Nadaraya-Watson weights to smooth the partial residuals $Y_i - \mathbf{Z}_i^\top \widehat{\beta}$.

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

Remark: It should be noted that if we set `lambda.seq` to 0, we can obtain the non-penalised estimation of the model, i.e. the OLS estimation. Using `lambda.seq` with a value $\neq 0$ is advisable when suspecting the presence of irrelevant variables.

Value

`call` The matched call.

`fitted.values` Estimated scalar response.

`residuals` Differences between `y` and the `fitted.values`.

`beta.est` $\hat{\beta}$ (i.e. the estimate of β_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 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.

`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.l=0.01, factor.pn=2,  nknot.theta=4,
    criterion="BIC",range.grid=c(850,1050),
    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 sugar samples. Detailed information about this dataset can be found at https://ucphchemometrics.com/datasets/.

Usage

```
data(Sugar)
```

Format

A `list` containing:

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

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, and moisture content, along with absorbance spectra (including the first and second derivatives), of 215 meat samples. A detailed description of the data can be found at http://lib.stat.cmu.edu/datasets/tecator.

Usage

```
data(Tecator)
```

Format

A `list` containing:

- `fat`: A vector with the fat content.
- `protein`: A vector with the protein content.
- `moisture`: A vector with the moisture content.
- `absor.spectra`: A matrix containing the near-infrared absorbance spectra observed at 100 equally spaced wavelengths in the range of 850-1050nm.
- `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, 107
    Tecator, 108

FASSMR.kernel.fit, 4, *4*, *13*, *36*, *38*, *50*, *62*
FASSMR.kNN.fit, *4*, 9, *9*, *38*, *42*, *44*, *50*, *62*
fsemipar *(*fsemipar-package*)*, 3
fsemipar-package, 3
fsemipar.internal, 14
fsim.kernel.fit, *3*, 15, *21*, *23*, *27*, *30*, *50*, *101*
fsim.kernel.fit.optim, *3*, 19, *23*, *27*, *30*
fsim.kernel.test, *3*, *18*, 22, *33*
fsim.kNN.fit, *3*, *18*, *21*, 24, *30*, *33*, *50*, *106*
fsim.kNN.fit.optim, *3*, *21*, *27*, 28, *33*
fsim.kNN.test, *3*, *23*, *27*, 31

IASSMR.kernel.fit, *4*, *9*, 33, *44*, *50*, *54*
IASSMR.kNN.fit, *4*, *13*, *38*, 39, *50*, *54*

lm.pels.fit, *3*, 45, *50*, *55*, *66*, *75*, *77*

plot.classes, 48
plot.FASSMR.kernel, *9*
plot.FASSMR.kernel *(*plot.classes*)*, 48
plot.FASSMR.kNN, *13*
plot.FASSMR.kNN *(*plot.classes*)*, 48
plot.fsim.kernel, *18*, *21*
plot.fsim.kernel *(*plot.classes*)*, 48
plot.fsim.kNN, *27*, *30*
plot.fsim.kNN *(*plot.classes*)*, 48
plot.IASSMR.kernel, *38*
plot.IASSMR.kernel *(*plot.classes*)*, 48
plot.IASSMR.kNN, *44*
plot.IASSMR.kNN *(*plot.classes*)*, 48
plot.lm.pels *(*plot.classes*)*, 48
plot.PVS *(*plot.classes*)*, 48
plot.PVS.kernel, *82*
plot.PVS.kNN, *87*
plot.sfpl.kernel, *93*
plot.sfpl.kernel *(*plot.classes*)*, 48
plot.sfpl.kNN, *97*
plot.sfpl.kNN *(*plot.classes*)*, 48
plot.sfplsim.kernel, *101*
plot.sfplsim.kernel *(*plot.classes*)*, 48
plot.sfplsim.kNN, *106*
plot.sfplsim.kNN *(*plot.classes*)*, 48
predict.FASSMR.kernel, *9*
predict.FASSMR.kernel *(*predict.sfplsim.FASSMR*)*, 61
predict.FASSMR.kNN, *13*
predict.FASSMR.kNN *(*predict.sfplsim.FASSMR*)*, 61
predict.fsim, 51
predict.fsim.kernel, *18*, *21*, *23*
predict.fsim.kNN, *27*, *30*, *33*
predict.IASSMR, 52
predict.IASSMR.kernel, *38*
predict.IASSMR.kNN, *44*
predict.lm, 55
predict.mfplm.PVS, 56
predict.PVS *(*predict.lm*)*, 55
predict.PVS.kernel, *82*
predict.PVS.kernel *(*predict.mfplm.PVS*)*, 56
predict.PVS.kNN, *87*
predict.PVS.kNN *(*predict.mfplm.PVS*)*, 56
predict.sfpl, 59
predict.sfpl.kernel, *58*, *93*
predict.sfpl.kNN, *58*, *97*
predict.sfplsim.FASSMR, 61
predict.sfplsim.kernel, *101*
predict.sfplsim.kernel *(*predict.sfplsim.FASSMR*)*, 61
predict.sfplsim.kNN, *106*
predict.sfplsim.kNN *(*predict.sfplsim.FASSMR*)*, 61
print.FASSMR.kernel *(*print.summary.mfplsim*)*, 67

print.FASSMR.kNN
(print.summary.mfplsim), 67
print.fsim.kernel (print.summary.fsim), 64
print.fsim.kNN (print.summary.fsim), 64
print.IASSMR.kernel
(print.summary.mfplsim), 67
print.IASSMR.kNN
(print.summary.mfplsim), 67
print.lm.pels (print.summary.lm), 65
print.PVS (print.summary.lm), 65
print.PVS.kernel (print.summary.mfpl), 66
print.PVS.kNN (print.summary.mfpl), 66
print.sfpl.kernel (print.summary.sfpl), 69
print.sfpl.kNN (print.summary.sfpl), 69
print.sfplsim.kernel
(print.summary.sfplsim), 70
print.sfplsim.kNN
(print.summary.sfplsim), 70
print.summary.fsim, 64
print.summary.lm, 65
print.summary.mfpl, 66
print.summary.mfplsim, 67
print.summary.sfpl, 69
print.summary.sfplsim, 70
projec, *3*, 71, *89*
PVS.fit, *3*, *47*, *50*, *55*, *66*, 73
PVS.kernel.fit, *4*, *50*, *58*, 77, *87*
PVS.kNN.fit, *4*, *50*, *58*, *82*, 83

semimetric.projec, *3*, *26*, *72*, 88
sfpl.kernel.fit, *3*, *50*, *58*, *60*, *80*, *82*, 90, *97*
sfpl.kNN.fit, *3*, *50*, *58*, *60*, *86, 87*, *93*, 94
sfplsim.kernel.fit, *3*, *7*, *9*, *36–38*, *50*, *54*, *62*, 97, *106*
sfplsim.kNN.fit, *3*, *12, 13*, *42–44*, *50*, *54*, *62*, *101*, 102
Sugar, *4*, 107
summary.FASSMR.kernel
(print.summary.mfplsim), 67
summary.FASSMR.kNN
(print.summary.mfplsim), 67
summary.fsim.kernel
(print.summary.fsim), 64
summary.fsim.kNN (print.summary.fsim), 64
summary.IASSMR.kernel
(print.summary.mfplsim), 67
summary.IASSMR.kNN
(print.summary.mfplsim), 67
summary.lm.pels (print.summary.lm), 65
summary.PVS (print.summary.lm), 65
summary.PVS.kernel
(print.summary.mfpl), 66
summary.PVS.kNN (print.summary.mfpl), 66
summary.sfpl.kernel
(print.summary.sfpl), 69
summary.sfpl.kNN (print.summary.sfpl), 69
summary.sfplsim.kernel
(print.summary.sfplsim), 70
summary.sfplsim.kNN
(print.summary.sfplsim), 70

Tecator, *4*, 108