Package ‘famTEMsel’

February 5, 2020

Type Package

Title Functional Additive Models for Treatment Effect-Modifier Selection

Version 0.1.0

Author Park, H., Petkova, E., Tarpey, T., Ogden, R.T.

Maintainer Hyung Park <parkh15@nyu.edu>

Description
An implementation of a functional additive regression model which is uniquely modified and constrained to model nonlinear interaction effects between a categorical treatment variable and a potentially large number of functional/scalar pretreatment covariates on their effects on a scalar-valued outcome. The model generalizes functional additive models by incorporating treatment-specific components into additive effect components, however, a structural constraint is imposed on the treatment-specific components, to give a class of orthogonal main and interaction effect additive models. If primary interest is in interactions, one can avoid estimating main effects, obviating the need to specify their form and thereby avoiding the issue of model misspecification. Utilizing this orthogonality, one can effectively conduct treatment effect-modifier variable selection. The selected covariates can be used to make individualized treatment recommendations. We refer to Park, Petkova, Tarpey, and Ogden (2020) <doi:10.1016/j.jspi.2019.05.008> and Park, Petkova, Tarpey, and Ogden (2020) ``Constrained functional additive models for estimating interactions between a treatment and functional covariates'' (pre-print) for detail of the method. The main function of this package is famTEMsel().

License GPL-3

Imports samTEMsel, mgcv, SAM, stats, splines, graphics

Remotes syhyunpark/samTEMsel

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

RemoteType github

RemoteHost api.github.com

RemoteRepo famTEMsel

RemoteUsername syhyunpark

RemoteRef master

RemoteSha b371f2d741f025cc9c2d6551401583059909aa2e

GithubRepo famTEMsel

GithubUsername syhyunpark

GithubRef master

GithubSHA1 b371f2d741f025cc9c2d6551401583059909aa2e

NeedsCompilation no

R topics documented:

`cv.famTEMsel` *Functional Additive Models for Treatment Effect-Modifier Selection (cross-validation function)*

Description

Does k-fold cross-validation for `famTEMsel`, selects an optimal regularization parameter index, `lambda.opt.index`, and returns the estimated constrained functional additive model given the optimal regularization parameter index `lambda.opt.index`.

Usage

```
cv.famTEMsel(y, A, X, Z = NULL, mu.hat = NULL, n.folds = 5, d = 3,
  k = 6, bs = "ps", sp = NULL, lambda.opt.index = NULL,
  lambda = NULL, nlambda = 30, lambda.min.ratio = 0.01,
  lambda.index.grid = 1:floor(nlambda/3), cv1sd = FALSE,
  thol = 1e-05, max.ite = 1e+05, regfunc = "L1", max.iter = 10,
  eps.iter = 0.01, eps.num.deriv = 1e-04, trace.iter = TRUE,
  plots = TRUE)
```

Arguments

`y`	a n-by-1 vector of responses
`A`	a n-by-1 vector of treatment variable; each element represents one of the L(>1) treatment conditions; e.g., c(1,2,1,1,3...); can be a factor-valued
`X`	a length-p list of functional-valued covariates, with its jth element corresponding to a n-by-n.eval[j] matrix of the observed jth functional covariates; n.eval[j] represents the number of evaluation points of the jth functional covariates
`Z`	a n-by-q matrix of scalar-valued covaraites
`mu.hat`	a n-by-1 vector of the fitted (X,Z)-main effect term of the model provided by the user; defult is `NULL`, in which case `mu.hat` is taken to be a vector of zeros; the optimal choice for this vector is E(y|X,Z)
`n.folds`	number of folds for cross-validation; the default is 5.

`d`	number of basis spline functions to be used for each component function; the default value is 3; d=1 corresponds to the linear model
`k`	dimension of the basis for representing each single-index coefficient function; see `mgcv::gam` for detail; the default value is 6.
`bs`	type of basis for representing the single-index coefficient functions; the defult is "ps" (p-splines); any basis supported by mgcv::gam can be used, e.g., "cr" (cubic regression splines).
`sp`	smoothing parameter associated with the single-index coefficient function; the default is `NULL`, in which case the smoothing parameter is estimated based on generalized cross-validation.
`lambda.opt.index`	a user-supplied optimal regularization parameter index to be used; the default is NULL, in which case n.folds cross-validation is performed to select an optimal index.
`lambda`	a user-supplied regularization parameter sequence; typical usage is to have the program compute its own lambda sequence based on nlambda and lambda.min.ratio.
`nlambda`	total number of lambda values; the default value is 30.
`lambda.min.ratio`	the smallest value for lambda, as a fraction of lambda.max, the (data derived) entry value (i.e. the smallest value for which all coefficients are zero); the default is 1e-2.
`lambda.index.grid`	a set of indices of `lambda`, in which the search for an optimal regularization parameter is to be conducted.
`cv1sd`	if `TRUE`, an optimal regularization parameter is chosen based on: the mean cross-validated error + 1 SD of the mean cross-validated error, which typically results in an increase in regularization; the defualt is `FALSE`.
`thol`	stopping precision for the coordinate-descent algorithm; the default value is 1e-5.
`max.ite`	number of maximum iterations for the coordinate-descent procedure used in estimating the component functions; the default value is 1e+5.
`regfunc`	type of the regularizer for variable selection; the default is "L1"; can also be "MCP" or "SCAD".
`max.iter`	number of maximum iterations for the iterative procedure used in estimating the single-index coefficient functions; the default value is 1e+1.
`eps.iter`	a value specifying the convergence criterion for the iterative procedure used in estimating the single-index coefficient functions; the defult is 1e-2.
`eps.num.deriv`	a small value that defines a finite difference used in computing the numerical (1st) derivative of the estimated component function; the default is 1e-4.
`trace.iter`	if `TRUE`, trace the estimation process by printing the difference in the estimated single-index basis coefficients (as compared to the previous iteration), and the functional norms of the estimated component functions, for each iteration.
`plots`	if `TRUE`, produce a cross-validation plot of the estimated mean squared error versus the regulariation parameter index.

Value

a list of information of the fitted constrained functional additive model including

`famTEMsel.obj` an object of class `famTEMsel`, which contains the sequence of the set of fitted component functions `samTEMsel.obj` implied by the sequence of the regularization parameters `lambda` and the corresponding set of fitted single-index coefficient functions `si.fit`; see `famTEMsel` for detail.

`lambda.opt.index` an index number, indicating the index of the estimated optimal regularization parameter in `lambda`.

`nonzero.index` a set of numbers, indicating the indices of estimated nonzero component functions, evalated at the regularization parameter index `lambda.opt.index`.

`nonzero.X.index` a set of numbers, indicating the indices of estimated nonzero component functions associated with the p functional covariates, evalated at the regularization parameter index `lambda.opt.index`.

`func_norm.opt` a p-by-1 vector, indicating the norms of the estimated component functions evaluatd at the regularization parameter index `lambda.opt.index`, with each element corresponding to the norm of each estimated component function.

`cv.storage` a n.folds-by-nlambda matrix of the estimated mean squared errors, with each column corresponding to each of the regularization parameters in `lambda` and each row corresponding to each of the n.folds folds.

`mean.cv` a nlambda-by-1 vector of the estimated mean squared errors, with each element corresponding to each of the regularization parameters in `lambda`.

`sd.cv` a nlambda-by-1 vector of the standard deviation of the estimated mean squared errors, with each element corresponding to each of the regularization parameters in `lambda`.

Author(s)

Park, Petkova, Tarpey, Ogden

See Also

`famTEMsel`, `predict_famTEMsel`, `plot_famTEMsel`

Examples

```
p = q = 10 # p and q are the numbers of functional and scalar pretreatment covariates, respectively.
n.train = 300  # training set sample size
n.test = 1000  # testing set sample size
n = n.train + n.test

# generate p pretreatment functional covariates X by first seting up functional basis:
n.eval = 50; s = seq(0, 1, length.out = n.eval)  # a grid of support points
b1 = sqrt(2)*sin(2*pi*s)
b2 = sqrt(2)*cos(2*pi*s)
b3 = sqrt(2)*sin(4*pi*s)
b4 = sqrt(2)*cos(4*pi*s)
B  = cbind(b1, b2, b3, b4)   # a (n.eval-by-4) basis matrix
# randomly generate basis coefficients, and then add measurement noise
```

```
X = vector("list", length= p);  for(j in 1:p){
  X[[j]] = matrix(rnorm(n*4, 0, 1), n, 4) %*% t(B) +
    matrix(rnorm(n*n.eval, 0, 0.25), n, n.eval)    # measurement noise
}
Z  = matrix(rnorm(n*q, 0, 1), n, q)   # q scalar covariates
A  = rbinom(n, 1, 0.5) + 1  # treatment variable taking a value in {1,2} with equal prob.

# X main effect on y; depends on the first 5 covariates
# the effect is generated randomly; randomly generated basis coefficients, scaled to unit L2 norm.
tmp = apply(matrix(rnorm(4*5), 4,5), 2, function(s) s/sqrt(sum(s^2) ))
main.effect = rep(0, n); for(j in 1:5){
 main.effect = main.effect + cos(X[[j]]%*% B %*%tmp[,j]/n.eval) # nonlinear effect (cosine)
}; rm(tmp)
# Z main effect on y; also depends on first 5 covariates
for(k in 1:5){
  main.effect = main.effect + cos(Z[,k])
}

# define (interaction effect) coefficient functions associted with X[[1]] and X[[2]]
beta1 =  B %*% c(0.5,0.5,0.5,0.5)
beta2 =  B %*% c(0.5,-0.5,0.5,-0.5)
# A-by-X ineraction effect on y; depends only on X[[1]] and X[[2]].
interaction.effect = (A-1.5)*( 2*sin(X[[1]]%*%beta1/n.eval) + 2*sin(X[[2]]%*%beta2/n.eval))
# A-by-Z ineraction effect on y; depends only on Z[,1] and Z[,2].
interaction.effect =  interaction.effect +  (A-1.5)*(Z[,1] + 2*sin(Z[,2]))

# generate outcome y
noise = rnorm(n, 0, 0.5)
y = main.effect  + interaction.effect + noise

var.main <- var(main.effect)
var.interaction <- var(interaction.effect)
var.noise <- var(noise)
SNR <- var.interaction/ (var.main + var.noise)
SNR  # "signal-to-noise" ratio


# train/test set splitting
train.index = 1:n.train
y.train = y[train.index]
X.train= X.test = vector("list", p);  for(j in 1:p){
X.train[[j]] = X[[j]][train.index,]
X.test[[j]]  = X[[j]][-train.index,]
}
A.train = A[train.index]
A.test  = A[-train.index]
y.train = y[train.index]
y.test  = y[-train.index]
Z.train = Z[train.index,]
Z.test  = Z[-train.index,]


# obtain an optimal regularization parameter and the corresponding model by running cv.famTEMsel().
cv.obj = cv.famTEMsel(y.train, A.train, X.train, Z.train)
lambda.opt.index = cv.obj$lambda.opt.index   # optimal regularization parameter index
cv.obj$func_norm.opt  # L2 norm of the component functions, associated with lambda.opt.index.
famTEMsel.obj = cv.obj$famTEMsel.obj  # extract the fitted model associted with lambda.opt.index.
```

```
# see also, famTEMsel() for the detail of famTEMsel.obj.

famTEMsel.obj$nonzero.index  # set of indices for the component functions estimated as nonzero
# plot the component functions estimated as nonzero
plot_famTEMsel(famTEMsel.obj, which.index = famTEMsel.obj$nonzero.index)

# make ITRs for subjects with pretreatment characteristics, X.test and Z.test
trt.rule = make_ITR_famTEMsel(famTEMsel.obj, newX = X.test, newZ = Z.test)$trt.rule
head(trt.rule)

# an (IPWE) estimate of the "value" of this particualr treatment rule, trt.rule:
mean(y.test[A.test==trt.rule])

# compare the above value to the following estimated "values" of "naive" treatment rules:
mean(y.test[A.test==1])   # a rule that assigns everyone to A=1
mean(y.test[A.test==2])   # a rule that assigns everyone to A=2
```

---

famTEMsel *Functional Additive Models for Treatment Effect-Modifier Selection (main function)*

Description

The function famTEMsel implements estimation of a constrained functional additve model.

Usage

```
famTEMsel(y, A, X, Z = NULL, mu.hat = NULL, d = 3, k = 6,
  bs = "ps", sp = NULL, lambda = NULL, nlambda = 30,
  lambda.min.ratio = 0.01, lambda.index = floor(nlambda/3),
  thol = 1e-05, max.ite = 1e+05, regfunc = "L1", eps.iter = 0.01,
  max.iter = 10, eps.num.deriv = 1e-04, trace.iter = TRUE)
```

Arguments

y	a n-by-1 vector of responses
A	a n-by-1 vector of treatment variable; each element represents one of the L(>1) treatment conditions; e.g., c(1,2,1,1,3...); can be a factor-valued
X	a length-p list of functional-valued covariates, with its jth element corresponding to a n-by-n.eval[j] matrix of the observed jth functional covariates; n.eval[j] represents the number of evaluation points of the jth functional covariates
Z	a n-by-q matrix of scalar-valued covaraites
mu.hat	a n-by-1 vector of the fitted (X,Z)-main effect term of the model provided by the user; defult is NULL, in which case mu.hat is taken to be a vector of zeros; the optimal choice for this vector is E(y|X,Z)
d	number of basis spline functions to be used for each component function; the default value is 3; d=1 corresponds to the linear model
k	number of basis spline functions to be used for each single-index coefficient function associated with each functional covariate;

- `bs` type of basis for representing the single-index coefficient functions; the defult is "ps" (p-splines); any basis supported by mgcv::gam can be used, e.g., "cr" (cubic regression splines)
- `sp` smoothing parameter associated with the single-index coefficient function; the default is NULL, in which case the smoothing parameter is estimated based on generalized cross-validation
- `lambda` a user-supplied regularization parameter sequence; typical usage is to have the program compute its own lambda sequence based on nlambda and lambda.min.ratio.
- `nlambda` total number of lambda values; the default value is 30.
- `lambda.min.ratio` the smallest value for lambda, as a fraction of lambda.max, the (data derived) entry value (i.e. the smallest value for which all coefficients are zero); the default is 0.01.
- `lambda.index` a user-supplied regularization parameter index to be used; the default is floor(nlambda/3).
- `thol` stopping precision for the coordinate-descent algorithm; the default value is 1e-5.
- `max.ite` number of maximum iterations for the coordinate-descent procedure in fitting the component functions; the default value is 1e+5.
- `regfunc` type of the regularizer; the default is "L1"; can also be "MCP" or "SCAD".
- `eps.iter` a value specifying the convergence criterion for the iterative procedure in fitting the single-index coefficient functions; the defult is 1e-2.
- `max.iter` number of maximum iterations for the iterative procedure in fitting the single-index coefficient functions; the default value is 1e+1.
- `eps.num.deriv` a small value used in the finite difference method for computing the numerical (1st) derivatives of the estimated component functions; the default is 1e-4.
- `trace.iter` if `TRUE`, trace the estimation process by printing, for each iteration, the difference from the previous iteration in the estimated single-index basis coefficients and the functional norms of the estimated component functions.

Details

A constrained functional model represents the joint effects of treatment, pretreatment p functional covariates and q scalar covariates on an outcome variable via a sum of treatment-specific additive flexible component functions defined over the (p + q) covariates, subject to the constraint that the expected value of the outcome given the covariates equals zero, while leaving the main effects of the covariates unspecified. The p pretreatment functional covariates appear in the model as 1-dimensional projections, via inner products with corresponding single-index coefficient functions. Under this model, the treatment-by-covariates interaction effects are determined by distinct shapes (across treatment levels) of the treatment-specific flexible component functions. Optimized under a penalized least square criterion with a L1 (or SCAD/MCP) penalty, the constrained functional additive model can effectively identify/select treatment effect-modifiers (from the p functional and q scalar covariates) that exhibit possibly nonlinear interactions with the treatment variable; this is achieved by producing a sparse set of estimated component functions of the model. The estimated nonzero component functions and single-index coefficient functions (available from the returned `famTEMsel` object) can be used to make individualized treatment recommendations (ITRs) for future subjects; see also `make_ITR_famTEMsel` for such ITRs.

The regularization path for the component functions is computed at a grid of values for the regularization parameter `lambda`.

Value

a list of information of the fitted constrained functional additive models including

`samTEMsel.obj` an object of class `samTEMsel`, which contains the sequence of the set of fitted component functions implied by the sequence of the regularization parameters `lambda`; the sparse additive models are fitted over the set of the p functional covariates projected onto the estimated single-index coefficient functions (stored in `si.fit`) and the set of q scalar covariates; the object `samTEMsel.obj` includes the residuals of the fitted models and the fitted values for the response variable; see `samTEMsel::samTEMsel` for detail of the `samTEMsel` object.

`si.fit` the length-p list of the single-index coefficient function estimate objects; each element is a `mgcv::gam` object; the jth element corresponds to the estimated single-index coefficient function associated with the jth functional covariate.

`si.coef.path` the length-p list, where the jth element is a (iter-by-k) matrix, with the lth row corresponding to the basis coefficient vector estimate associated with the jth single-index coefficient function at the lth iteration of the fitting procedure.

`mean.fn` the length-p list of mean functions (averaged across n observations), where the jth element is a n.eval[j]-by-1 vector of the evaluation of the estimated mean of the jth functional covariate.

`n.eval` a length-p vector, where its jth element represents the number of evaluation points of the jth functional covariate.

`func_norm.record` the iter-by-(p+q) matrix, with its lth row corresponding to the vector of the estimated (p+q) component functions' L2 norms at the lth iteration.

`func_norm` a length (p+q) vector of the estimated (p+q) component functions' L2 norms, at the final iteration.

`lambda` the sequence of regularization parameters used in the object `samTEMsel.obj`.

`lambda.index` an index number, indicating the index of the regularization parameter in `lambda` used in obtaining the fitted model (including the single-index coefficient functions).

`nonzero.index` a set of numbers, indicating the indices of estimated nonzero component functions of this particular fit under the regularization parameter index `lambda.index`.

`nonzero.X.index` a set of numbers, indicating the indices of estimated nonzero component functions associated with the p functional covariates, based on this particular fit under the regularization parameter index `lambda.index`.

Author(s)

Park, Petkova, Tarpey, Ogden

See Also

`cv.famTEMsel`, `predict_famTEMsel`, `plot_famTEMsel`, `make_ITR_famTEMsel`

Examples

```
p = q = 10 # p and q are the numbers of functional and scalar pretreatment covariates, respectively.
n.train = 300  # training set sample size
```

```
n.test = 1000  # testing set sample size
n = n.train + n.test

# generate p pretreatment functional covariates X by first seting up functional basis:
n.eval = 50; s = seq(0, 1, length.out = n.eval)  # a grid of support points
b1 = sqrt(2)*sin(2*pi*s)
b2 = sqrt(2)*cos(2*pi*s)
b3 = sqrt(2)*sin(4*pi*s)
b4 = sqrt(2)*cos(4*pi*s)
B  = cbind(b1, b2, b3, b4)   # a (n.eval-by-4) basis matrix
# randomly generate basis coefficients, and then add measurement noise
X = vector("list", length= p);  for(j in 1:p){
  X[[j]] = matrix(rnorm(n*4, 0, 1), n, 4) %*% t(B) +
    matrix(rnorm(n*n.eval, 0, 0.25), n, n.eval)    # measurement noise
}
Z  = matrix(rnorm(n*q, 0, 1), n, q)   # q scalar covariates
A  = rbinom(n, 1, 0.5) + 1  # treatment variable taking a value in {1,2} with equal prob.

# X main effect on y; depends on the first 5 covariates
# the effect is generated randomly; randomly generated basis coefficients, scaled to unit L2 norm.
tmp = apply(matrix(rnorm(4*5), 4,5), 2, function(s) s/sqrt(sum(s^2) ))
main.effect = rep(0, n); for(j in 1:5){
 main.effect = main.effect + cos(X[[j]]%*% B%*%tmp[,j]/n.eval) # nonlinear effect (cosine)
}; rm(tmp)
# Z main effect on y; also depends on first 5 covariates
for(k in 1:5){
  main.effect = main.effect + cos(Z[,k])
}

# define (interaction effect) coefficient functions associted with X[[1]] and X[[2]]
beta1 =  B %*% c(0.5,0.5,0.5,0.5)
beta2 =  B %*% c(0.5,-0.5,0.5,-0.5)
# A-by-X ineraction effect on y; depends only on X[[1]] and X[[2]].
interaction.effect = (A-1.5)*( 2*sin(X[[1]]%*%beta1/n.eval) + 2*sin(X[[2]]%*%beta2/n.eval))
# A-by-Z ineraction effect on y; depends only on Z[,1] and Z[,2].
interaction.effect =  interaction.effect + (A-1.5)*(Z[,1] + 2*sin(Z[,2]))

# generate outcome y
noise = rnorm(n, 0, 0.5)
y = main.effect  + interaction.effect + noise

var.main <- var(main.effect)
var.interaction <- var(interaction.effect)
var.noise <- var(noise)
SNR <- var.interaction/ (var.main + var.noise)
SNR  # "signal-to-noise" ratio

# train/test set splitting
train.index = 1:n.train
y.train = y[train.index]
X.train= X.test = vector("list", p);  for(j in 1:p){
  X.train[[j]] = X[[j]][train.index,]
  X.test[[j]]  = X[[j]][-train.index,]
}
A.train = A[train.index]
A.test  = A[-train.index]
y.train = y[train.index]
```

```
y.test  = y[-train.index]
Z.train = Z[train.index,]
Z.test  = Z[-train.index,]


# fit a model with some regularization parameter index, say, lambda.index = 10.
# (an optimal regularization parameter can be estimated by running cv.famTEMsel().)
famTEMsel.obj = famTEMsel(y.train, A.train, X.train, Z.train, lambda.index=10)
famTEMsel.obj$func_norm  # L2 norm of the estimated component functions of the model
famTEMsel.obj$nonzero.index # set of indices for the component functions estimated as nonzero
# plot the component functions estimated as nonzero and the single-index functions
plot_famTEMsel(famTEMsel.obj, which.index = famTEMsel.obj$nonzero.index)

# make ITRs for subjects with pretreatment characteristics, X.test and Z.test
trt.rule = make_ITR_famTEMsel(famTEMsel.obj, newX = X.test, newZ = Z.test)$trt.rule
head(trt.rule)

# an (IPWE) estimate of the "value" of this particualr treatment rule, trt.rule:
mean(y.test[A.test==trt.rule])

# compare the above value to the following estimated "values" of "naive" treatment rules:
mean(y.test[A.test==1])   # a rule that assigns everyone to A=1
mean(y.test[A.test==2])   # a rule that assigns everyone to A=2
```

---

make_ITR_famTEMsel — *make individualized treatment recommendations (ITRs) based on a* famTEMsel *object*

Description

The function make_ITR_famTEMsel returns individualized treatment decision recommendations for subjects with pretreatment characteristics newX and newZ, given a famTEMsel object, famTEMsel.obj, and an (optimal) regularization parameter index, lambda.index.

Usage

```
make_ITR_famTEMsel(famTEMsel.obj, newX = NULL, newZ = NULL,
  lambda.index = NULL, maximize = TRUE)
```

Arguments

famTEMsel.obj	a famTEMsel object, containing the fitted constrained functional additive models.
newX	a length-p list of new values for the functional-valued covariates X, where the jth element is a (n-by-n.eval[j]) matrix of the observed jth function, at which predictions are to be made; if NULL, X from the training set is used.
newZ	a (n-by-q) matrix of new values for the scalar-valued covariates Z at which predictions are to be made; if NULL, Z from the training set is used.
lambda.index	an index of the regularization parameter lambda at which predictions are to be made; one can supply lambda.opt.index obtained from the function cv.famTEMsel(); the default is NULL, in which case the predictions are made at the lambda.index used in obtaining famTEMsel.obj.

`maximize`	default is `TRUE`, assuming a larger value of the outcome is better; if `FALSE`, a smaller value is assumed to be prefered.

Value

`pred.new`	a (n-by-L) matrix of predicted values, with each column representing one of the L treatment options.
`trt.rule`	a (n-by-1) vector of the individualized treatment recommendations

Author(s)

Park, Petkova, Tarpey, Ogden

See Also

`famTEMsel`,`cv.famTEMsel`, `predict_famTEMsel`

`plot_famTEMsel` *plot component functions from a* `famTEMsel` *object*

Description

Produces plots of the component functions and the single-index coefficient functions from a `famTEMsel` object.

Usage

```
plot_famTEMsel(famTEMsel.obj, newX = NULL, newZ = NULL, newA = NULL,
  scatter.plot = TRUE, lambda.index = famTEMsel.obj$lambda.index,
  which.index = famTEMsel.obj$nonzero.index, ylims,
  single.index.plot = TRUE, solution.path = FALSE)
```

Arguments

`famTEMsel.obj`	a `famTEMsel` object
`newX`	a (n by p) list of new values for the functional covariates X at which predictions are to be made; the jth element of the list corresponds to a n-by-n.eval[j] matrix of the observed jth functional covariates; n.eval[j] represents the number of evaluation points of the jth functional covariates; if `NULL`, X from the training set is used.
`newZ`	a (n by q) matrix of new values for the scalar covariates Z at which predictions are to be made; if `NULL`, Z from the training set is used.
`newA`	a (n-by-1) vector of new values for the treatment A at which plots are to be made; the default is `NULL`, in which case A is taken from the training set.
`scatter.plot`	if `TRUE`, draw scatter plots of partial residuals versus the covariates; these scatter plots are made based on the training observations; the default is `TRUE`.
`lambda.index`	an index of the tuning parameter `lambda` at which plots are to be made; one can supply `lambda.opt.index` obtained from the function `cv.samTEMsel`; the default is `NULL`, in which case `plot_samTEMsel` utilizes the most non-sparse model.

which.index	this specifies which component functions are to be plotted; the default is all p component functions, i.e., 1:p.
ylims	this specifies the vertical range of the plots, e.g., c(-10, 10).
single.index.plot	if TRUE, draw the plots of the estimated single-index coefficient functions; the default is TRUE.
solution.path	if TRUE, draw the functional norms of the fitted component functions (based on the training set) versus the regularization parameter; the default is FALSE.

Author(s)

Park, Petkova, Tarpey, Ogden

See Also

famTEMsel,predict_famTEMsel, cv.famTEMsel

---

predict_famTEMsel   famTEMsel *prediction function*

Description

predict_famTEMsel makes predictions given a (new) set of functional covariates newX, a (new) set of scalar covariates newZ and a (new) vector of treatment indicators newA based on a constrained functional additive model famTEMsel.obj. Specifically, predict_famTEMsel predicts the responses y based on the (X,Z)-by-A interaction effect (plus the A main effect) portion of the full model that includes the unspecified X main effect term.

Usage

```
predict_famTEMsel(famTEMsel.obj, newX = NULL, newZ = NULL,
  newA = NULL, type = "response", lambda.index = NULL)
```

Arguments

famTEMsel.obj	a famTEMsel object
newX	a (n by p) list of new values for the functional covariates X at which predictions are to be made; the jth element of the list corresponds to a n-by-n.eval[j] matrix of the observed jth functional covariates; n.eval[j] represents the number of evaluation points of the jth functional covariates; if NULL, X from the training set is used.
newZ	a (n by q) matrix of new values for the scalar covariates Z at which predictions are to be made; if NULL, Z from the training set is used.
newA	a (n by 1) vector of new values for the treatment A at which predictions are to be made; if NULL, A from the training set is used.
type	the type of prediction required; the default "response" gives the predicted responses y based on the whole model; the alternative "terms" gives the component-wise predicted responses from each of the p components (and plus the treatment-specific intercepts) of the model.

`lambda.index` an index of the tuning parameter `lambda` at which predictions are to be made; one can supply `lambda.opt.index` obtained from the function `cv.samTEMsel`; the default is `NULL`, in which case the predictions based on the most non-sparse model is returned.

Value

`predicted` a (n-by-`length(lambda.index)`) matrix of predicted values; a (n-by-`length(lambda.index)`*(p+q-matrix of predicted values if type = "terms", where the last column corresponds to the (treatment-specific) intercept.

`U` a n-by-(p+q) matrix of the index variables; the first p columns correspond to the 1-D projections of the p functional covariates and the last q columns correspond to the q scalar covariates.

Author(s)

Park, Petkova, Tarpey, Ogden

See Also

`famTEMsel`,`cv.famTEMsel`, `plot_famTEMsel`

Index