Package ‘fonfDNN’

June 24, 2025

Type Package

Title Hybrid deep tensor network for function-on-function regression with mixed predictors

Version 1.0

Depends R (>= 3.5.0), fda, keras, tensorflow, fda.usc, plot3D

Imports goffda

LazyLoad yes

ByteCompile TRUE

Encoding UTF-8

Maintainer Ufuk Beyaztas <ufukbeyaztas@gmail.com>

Description Functions for implementing deep tensor network for function-on-function regression with mixed predictors.

Contents

`dgp_mixed` *Simulate Mixed Functional–Scalar Data for Function–on–Function Regression*

Description

Generates synthetic datasets that mimic the structure analysed in *Beyaztas, Bakicierler Sezer, (2025)*: a functional response observed on a dense grid, multiple functional predictors, and multiple scalar predictors. Two regimes are available:

- `"linear"`: response driven solely by linear integral effects.
- `"nonlinear"`: adds strong, smooth nonlinear interactions (functional $\times$ functional, functional $\times$ scalar, scalar $\times$ scalar) scaled to dominate the linear part.

Every dataset also includes Ornstein–Uhlenbeck (OU) process noise to emulate realistic autocorrelated measurement error.

Usage

```
dgp_mixed(n, j,
          model   = c("linear", "nonlinear"),
          n_func  = 5L,
          n_scl   = 3L,
          seed    = NULL)
```

Arguments

`n`	Number of subjects/curves.
`j`	Grid length M; all functional objects are evaluated on `seq(0, 1, length.out = j)`.
`model`	Character string choosing the data-generating mechanism; partial matching is supported.
`n_func`	Number P of functional predictors to simulate.
`n_scl`	Number d_z of scalar predictors.
`seed`	Optional integer for reproducibility (`set.seed(seed)`).

Details

Let $X_i^{(p)}(s)$ denote functional predictor $p = 1, \dots, P$ and $\mathbf{Z}_i \in \mathbb{R}^{d_z}$ the scalar predictor vector for subject $i \in \{1, \dots, n\}$. The latent *signal* generating the functional response on grid points $t_1, \dots, t_M$ is

$$\eta_i(t) = \underbrace{\sum_{p=1}^{P} \int_0^1 X_i^{(p)}(s)\, \beta_p(s,t)\, ds +}_{\text{linear functional effects}} \underbrace{\mathbf{Z}_i^\top \gamma(t)}_{\text{scalar effects}} + \mathcal{N}L_i(t),$$

where $P = 3$ "strong" functional predictors drive the linear component, β_p are pre-specified wavy B-spline surfaces (see code), and $\gamma(t)$ are smooth one-dimensional bases. If `model == "linear"` we set $\mathcal{N}L_i(t) \equiv 0$; otherwise $\mathcal{N}L_i(t)$ equals the *sum* of five carefully designed nonlinear terms (quadratic functional interactions, sinusoidal transforms, scalar–functional products, etc.) rescaled so that

$$\mathrm{SD}\{\mathcal{N}L\} \approx 2\,\mathrm{SD}\{\text{linear signal}\}.$$

The *observed* response is

$$Y_i(t_m) = \eta_i(t_m) + \varepsilon_i(t_m), \qquad \varepsilon_i \sim \mathrm{OU}(0, \alpha = 3, \sigma = 0.7),$$

i.e. an Ornstein–Uhlenbeck process discretised on the same grid. Noise is scaled so that its empirical standard deviation equals 10% of the signal standard deviation.

Value

A named `list`:

`y`	$n \times j$ matrix of noisy responses $Y_i(t_m)$.
`yt`	$n \times j$ matrix of true signals $\eta_i(t_m)$.
`x`	`list` of length `P`; each element is an $n \times j$ matrix of functional predictors $X_i^{(p)}(s_m)$.
`x.scl`	$n \times d_z$ numeric matrix of scalar predictors $\mathbf{Z}_i$.
`meta`	List containing grids `sx`, `sy`, true beta surfaces (`beta`), and the `model` flag.

Note

Requires the suggested package **goffda** for OU noise generation.

Author(s)

Ufuk Beyaztas, Gizel Bakicierler Sezer

References

Kokoszka, P. and Reimherr, M.~(2017). *Introduction to Functional Data Analysis*. Chapman and Hall/CRC.

See Also

`fonf_fit`, `r_ou`

Examples

```
## Not run:
# --------------------------------------------------------------
# Simulate a nonlinear training set and inspect its structure
# --------------------------------------------------------------
# n <- 100                      # sample size
# simdata <- dgp_mixed(n, 101, model = "nonlinear")

# y    <- simdata$y             # noisy response curves
# yt   <- simdata$yt            # true (noise-free) curves
# x    <- simdata$x             # list of functional predictors
# xscl <- simdata$x.scl         # scalar predictors
## End(Not run)
```

fonf_fit *Fit a Hybrid Deep Tensor Network for Function-on-Function Regression with Mixed Predictors*

Description

Trains the hybrid deep tensor network (HDTN) of *Beyaztas, Bakicierler Sezer, (2025)* to predict a *functional response* from multiple functional and scalar covariates. The model combines a first-layer tensor-product B-spline representation (capturing linear functional effects) with fully-connected dense layers (capturing higher-order nonlinear interactions) and supplies finite-sample, distribution-free prediction bands via conformal inference.

Usage

```
fonf_fit(resp,
         func_cov,
         scalar_cov           = NULL,
         nbasis_y             = NULL,
         nbasis_x             = NULL,
         hidden_layers        = 2,
         neurons_per_layer    = c(32, 32),
         activations_in_layers = c("relu", "linear"),
         epochs               = 100,
         batch_size           = 32,
         val_split            = 0.1,
         learning_rate        = 1e-3,
         patience_param       = 15,
         dropout_rate         = 0.1,
         l2_lambda            = 1e-4,
         cal_prop             = 0.2,
         alpha                = 0.2,
         verbose              = 1)
```

Arguments

resp: $n \times M$ numeric matrix; row i stores the functional response $Y_i(t_m)$ evaluated on a common grid $t_1, \dots, t_M \subset \mathcal{I}_y$.

func_cov: List of length P. Element p is an $n \times G_p$ matrix that holds the functional predictor $X_i^{(p)}(s_{pj})$ on its own grid $s_{p1}, \dots, s_{pG_p} \subset \mathcal{I}_{x_p}$.

scalar_cov: Optional $n \times d_z$ numeric matrix of scalar covariates $\mathbf{Z}_i \in \mathbb{R}^{d_z}$.

nbasis_y: Number of B-spline basis functions for the response domain (K_y); chosen automatically when `NULL`.

nbasis_x: Integer vector of length P; element p gives the number of input-domain basis functions $K_x^{(p)}$. Automatically selected when `NULL`.

hidden_layers: Number R of fully-connected hidden layers.

neurons_per_layer: Vector of length `hidden_layers` giving the width (D_r) of each dense layer.

activations_in_layers: Character vector of length `hidden_layers` with Keras activation names ("relu", "tanh", etc.).

`epochs`	Maximum training epochs.
`batch_size`	Mini-batch size for stochastic optimisation.
`val_split`	Proportion of the *training rows* (not subjects) held out for on-line validation during training.
`learning_rate`	Initial learning rate for the Adam optimiser (with cosine decay scheduler).
`patience_param`	Early-stopping patience; training stops when validation loss fails to improve for this many epochs.
`dropout_rate`	Dropout probability applied after every dense hidden layer.
`l2_lambda`	ℓ_2 (ridge) penalty applied to dense-layer weights.
`cal_prop`	Proportion of subjects set aside for the conformal *calibration* set.
`alpha`	Mis-coverage level for conformal prediction bands (e.g. `0.2` yields 80% bands).
`verbose`	Passed to `keras`; larger values give more console output.

Details

Let $Y_i(t) \in L^2(\mathcal{I}_y)$ be the response curve for subject i, $X_i^{(p)}(s) \in L^2(\mathcal{I}_{x_p})$, $p \in \{1, \ldots, P\}$, the functional predictors, and $\mathbf{Z}_i \in \mathbb{R}^{d_z}$ scalar predictors. The HDTN targets the nonlinear FoFR model

$$Y_i(t) \;=\; g\left\{\beta_0(t) \;+\; \sum_{p=1}^{P} \langle X_i^{(p)}, \beta_p(\cdot, t)\rangle_{L^2} \;+\; \mathbf{Z}_i^\top \theta(t)\right\} \;+\; \varepsilon_i(t), \quad t \in \mathcal{I}_y,$$

with identity link $g(u) = u$ in the present implementation.

Tensor-product layer. Each bivariate coefficient surface is expanded

$$\beta_p(s,t) = \sum_{k=1}^{K_x^{(p)}} \sum_{\ell=1}^{K_y} w_{k\ell}^{(p)}\, \phi_k^{(p)}(s) \psi_\ell(t),$$

yielding first-layer weights $w_{k\ell}^{(p)}$ to be learned. Subject-specific functional features are $\tilde{\varphi}_{ik}^{(p)} = \langle X_i^{(p)}, \phi_k^{(p)}\rangle_{L^2}$ and $u_{i\ell}^{(p)} = \sum_k \tilde{\varphi}_{ik}^{(p)}\, w_{k\ell}^{(p)}$.

Dense layers. The concatenated feature vector $\mathbf{h}_i^{(0)} = \big(u_{i\cdot}^{(1)\top}, \ldots, u_{i\cdot}^{(P)\top}, \mathbf{Z}_i^\top\big)^\top$ is propagated through R fully-connected layers $\mathbf{h}_i^{(r)} = \sigma_r\big(W^{(r)}\mathbf{h}_i^{(r-1)} + b^{(r)}\big)$ with dropout and ridge penalty $\lambda\|W^{(r)}\|_F^2/2$. The final linear layer outputs $\hat{\eta}_i(t_m)$, giving $\hat{Y}_i(t_m) = \hat{\eta}_i(t_m)$ for identity link.

Loss and optimisation. Training minimises mean integrated squared error plus the ridge penalty, $\mathcal{L} = \frac{1}{nM}\sum_{i=1}^{n}\sum_{m=1}^{M}\big\{Y_i(t_m) - \hat{Y}_i(t_m)\big\}^2 + \frac{\lambda}{2}\sum_{r=1}^{R}\|W^{(r)}\|_F^2$, via Adam with cosine-decay learning rate.

Conformal prediction. A calibration subset C (size `cal_prop * n` subjects) yields residuals $e_{jm} = |Y_j(t_m) - \hat{Y}_j(t_m)|$. The empirical $1-\alpha$ quantile $q_{1-\alpha}$ of `length(C)*M` pooled residuals provides a *constant-width* $(1-\alpha)$ band $\big[\hat{Y}_i(t_m) - q_{1-\alpha},\ \hat{Y}_i(t_m) + q_{1-\alpha}\big]$ for every new subject i and grid point t_m. Finite-sample marginal coverage is guaranteed (Lei, G'Sell, *et al.*, 2018).

Value

An object of class `"fonf_dl"` to be consumed by `fonf_predict`:

`model`	Trained Keras model.
`center, scale`	Vectors used to standardise the design matrix.

py Number of response grid points (M).

nbasis_y, nbasis_x
Basis dimensions actually used.

q_hat Half-width $q_{1-\alpha}$ of the conformal prediction band.

alpha User-supplied mis-coverage level.

history keras_training_history object returned by fit().

Note

Requires TensorFlow/Keras (tested with TensorFlow >= 2.16).

Author(s)

Ufuk Beyaztas, Gizel Bakicierler Sezer.

References

Lei, J., G'Sell, M., Rinaldo, A., Tibshirani, R. and Wasserman, L. (2018). *Distribution-Free Predictive Inference for Regression*. *Journal of the American Statistical Association*, 113(523), 1094-1111.

See Also

fonf_predict, keras_model_sequential

Examples

```
## Not run:
# --------------------------------------------------------------
# 1. Simulate training data
# --------------------------------------------------------------
# n <- 100
# simdata <- dgp_mixed(n, 101, model = "nonlinear")

# y    <- simdata$y
# yt   <- simdata$yt          # (true curves, if needed)
# x    <- simdata$x           # list of functional predictors
# xscl <- simdata$x.scl       # scalar predictors

# --------------------------------------------------------------
# 2. Fit the hybrid deep tensor network
# --------------------------------------------------------------
# nfof_model <- fonf_fit(resp       = y,
#                        func_cov   = x,
#                        scalar_cov = xscl)
## End(Not run)
```

`fonf_fnc` *Extract and Visualise Estimated Coefficient Surfaces*

Description

Given a fitted `fonf_fit` model, this helper recovers the bivariate coefficient surfaces $\widehat{\beta}_p(t, s)$ that link functional predictor $X^{(p)}(s)$ to the functional response $Y(t)$, evaluates them on user-supplied grids, and—optionally—renders compact, printer-friendly 3-D perspective plots via **plot3D**.

Usage

```
fonf_fnc(model,
         y_grid      = NULL,
         x_grid      = NULL,
         agg_fun     = mean,
         plot        = TRUE,
         grid_len    = 101,
         view_theta  = 40,
         view_phi    = 2,
         surface_col = "royalblue",
         border_col  = "black",
         shade_fac   = 0.5,
         title_mgp   = c(2.8, 0.8, 0),
         ...)
```

Arguments

`model`	Object of class `"fonf_dl"` returned by `fonf_fit`.
`y_grid`	Numeric vector of evaluation points $t_1, \dots, t_M$ in the interval $[0, 1]$. Defaults to `seq(0, 1, length.out = grid_len)`.
`x_grid`	List of length P; element p is the grid $s_{k1}, \dots, s_{kL_k}$ for predictor p. Defaults to a length-`grid_len` equi-spaced grid for every predictor.
`agg_fun`	Function used to collapse the first-layer tensor weights into a single set of B-spline coefficients. With the default `mean`: $\widehat{\beta}_p(t, s) = \frac{1}{D_1} \sum_{d=1}^{D_1} w_d^{(p)}\ B_y(t)\ B_x^{(p)}(s).$
`plot`	Logical; if `TRUE` (default) one 3-D plot is produced for each predictor.
`grid_len`	Length of the default equi-spaced grids when `y_grid` or an element of `x_grid` is `NULL`.
`view_theta, view_phi`	Viewing angles (in degrees) passed to `persp3D`.
`surface_col, border_col, shade_fac`	Graphical parameters for the surfaces.
`title_mgp`	Margin settings for titles (argument passed to `par(mgp = ...)` inside each plot).
`...`	Currently ignored, reserved for future extensions.

Details

Let $B_y(t)$ be the vector of K_y B-spline basis functions for t and let $B_x^{(p)}(s)$ be the vector of $K_x^{(p)}$ basis functions for s in predictor p. The first layer of the network stores a weight matrix $W^{(1)}$ whose rows correspond to the spline products $B_y(t)\,B_x^{(p)}(s)$. For each predictor p we:

1. Extract the contiguous block of $K_y K_x^{(p)}$ weights.
2. Fold it into a $K_y \times K_x^{(p)}$ matrix C_k.
3. Evaluate $\widehat{\beta}_p(t,s) = B_y(t)^\top C_k B_x^{(p)}(s)$ on the requested grids.

If `plot = TRUE` every surface is drawn with `persp3D`; otherwise the numeric matrices are returned silently.

Value

Invisibly returns a `list` with components:

`beta_hat`	List of length P; element p is a `length(y_grid)` by `length(x_grid[[p]])` matrix containing $\widehat{\beta}_p(t,s)$.
`y_grid`	Evaluation grid for t.
`x_grid`	Evaluation grids for s.
`plots`	List of closure functions; calling `plots[[p]]()` re-draws the p-th surface.

Note

Requires the suggested packages **fda**, **keras**, and **plot3D**. An error is thrown if any are missing.

Author(s)

Ufuk Beyaztas, Gizel Bakicierler Sezer

See Also

`fonf_fit`, `persp3D`, `eval.basis`

Examples

```
## Not run:
# ---------------------------------------------------------------
# Fit a small model and visualise its coefficient surfaces
# ---------------------------------------------------------------
# simdata <- dgp_mixed(100, 101, model = "nonlinear")
#
# y <- simdata$y
# x <- simdata$x
# xscl <- simdata$x.scl
#
# mdl <- fonf_fit(resp = y, func_cov = x, scalar_cov = xscl)
#
# Plot the 2-D functional weights
# surf_obj <- fonf_fnc(model = mdl)
## End(Not run)
```

`fonf_param_grid` *Create a Parameter Grid Without Factors*

Description

Generates a data frame from all combinations of input vectors, ensuring that character variables are not converted to factors. This is a convenient wrapper around `expand.grid(..., stringsAsFactors = FALSE)`.

Usage

```
fonf_param_grid(...)
```

Arguments

`...`	One or more vectors, factors, or lists to be combined into a data frame of parameter combinations.

Details

This function simplifies the creation of parameter grids for tuning models. Unlike the base R `expand.grid`, it ensures that character vectors remain as characters, which is often desirable when building machine learning or neural network models where hyperparameter names or tags are string-based.

Value

A data frame containing one row for each combination of the supplied vectors.

Note

This function is mainly used to build cross-product grids of model parameters when performing grid search for tuning.

Author(s)

Ufuk Beyaztas, Gizel Bakicierler Sezer

`fonf_predict` *Predict With a Trained Hybrid Deep Tensor Network*

Description

Generates point predictions and, optionally, distribution-free conformal prediction bands for a `fonf_fit` model given new functional and scalar covariates.

Usage

```
fonf_predict(object,
             func_cov_new,
             scalar_cov_new = NULL,
             interval = c("none", "conformal"))
```

Arguments

`object`	An object of class `"fonf_dl"` returned by `fonf_fit`.
`func_cov_new`	List of length P; element p is an $n_{\text{new}} \times G_p$ matrix that stores the new subjects' functional predictor $X^{(p)}_{\text{new},i}(s_{pj})$.
`scalar_cov_new`	Optional $n_{\text{new}} \times d_z$ numeric matrix of scalar covariates $Z_{\text{new},i}$. If the original model was fitted without scalar covariates, set this to `NULL`.
`interval`	Type of prediction to return: `"none"` Point estimates only. `"conformal"` Point estimates plus lower/upper bands that achieve marginal coverage $1-\alpha$ (see Details).

Details

Let f_{hat} denote the trained network and let (x_i, z_i) be the design-matrix rows for a new subject, built with the same tensor-product bases used in training. `fonf_predict`:

1. Builds the design matrix with `design_matrix_build`.
2. Standardises it with the training means and scales stored in the fitted object.
3. Computes the predictions, then reshapes the vector into an $n_{\text{new}} \times M$ matrix, where $M =$ `object$py`.

Conformal bands: If `interval == "conformal"` the returned bands are $\hat{Y}_i(t_m) \pm q_{1-\alpha}$, where $q_{1-\alpha}$ is the $(1-\alpha)$-quantile of the absolute residuals on the calibration set chosen during training. Split-conformal theory (Lei *et al.*, 2018) guarantees marginal coverage $1-\alpha$ at each grid point, without distributional assumptions beyond independent rows.

Value

`interval = "none"` A numeric matrix with $nrow = n_{new}$ and `ncol = py` containing point predictions.

`interval = "conformal"` A list with components `mean`, `lower`, and `upper`, each a matrix whose columns are named `"t1"`, `"t2"`, ..., `"tM"`.

Note

The feature grids supplied here must match those used at training time, and the length of the prediction vector must be divisible by `object$py`; otherwise an error is raised.

Author(s)

Ufuk Beyaztas, Gizel Bakicierler Sezer

References

Lei, J., G'Sell, M., Rinaldo, A., Tibshirani, R. & Wasserman, L. (2018). Distribution-Free Predictive Inference for Regression. *Journal of the American Statistical Association*, 113(523), 1094–1111.

See Also

`fonf_fit`, `dgp_mixed`

Examples

```
## Not run:
# ---------------------------------------------------------------
# 1. Train a model on simulated data
# ---------------------------------------------------------------
# simdata <- dgp_mixed(100, 101, model = "nonlinear")
# mdl <- fonf_fit(resp = simdata$y,
#                 func_cov = simdata$x,
#                 scalar_cov = simdata$x.scl)
#
# ---------------------------------------------------------------
# 2. Predict on new subjects with conformal bands
# ---------------------------------------------------------------
# test <- dgp_mixed(250, 101, model = "nonlinear")
# band <- fonf_predict(mdl,
#                      func_cov_new   = test$x,
#                      scalar_cov_new = test$x.scl,
#                      interval       = "conformal")
# pred_mean <- band$mean
## End(Not run)
```

`fonf_tune` *Simple Grid–Search Hyper-parameter Tuner for* `fonf_fit`

Description

Evaluates every row of a user-supplied hyper-parameter grid via K-fold cross-validation (`fonf_cv`), reports the cross-validated mean-squared error (CV–MSE), and then refits the best combination on the *full* data set. The implementation is deliberately lightweight—single-core, base R only—so it runs on any system where **fonf** installs.

Usage

```
fonf_tune(grid,
          resp, func_cov, scalar_cov = NULL,
          nfolds = 5)
```

Arguments

`grid`	A data frame created by `fonf_param_grid` describing the hyper-parameter combinations to be tested. Each column corresponds to an argument of `fonf_fit`.
`resp`	Numeric matrix of size $n \times p_y$; functional response curves sampled on a common grid.
`func_cov`	List of length P; element p is an $n \times G_p$ matrix containing the p-th functional predictor.
`scalar_cov`	Optional $n \times q$ numeric matrix of scalar predictors. Omit or set to `NULL` if none.
`nfolds`	Number of folds used by `fonf_cv` (default 5).

Details

How it works: For each row $g = 1, \ldots, G$ of grid:

1. The row is coerced to a named list of arguments compatible with fonf_fit (vectors such as neurons_per_layer are replicated to the correct length).
2. fonf_cv performs K-fold CV and stores the average test MSE in cv_vec[g].

After all rows are processed the index of the minimum $CV-MSE$ is selected, the corresponding parameter list is re-sanitised by sanitize_basis(), and a final model is fitted on the whole data via fonf_fit with verbose = TRUE.

Value

results Data frame combining the original grid and a new column CV_MSE (lower is better).

best_params Named list containing the best-performing hyper-parameters in a format ready for fonf_fit.

best_model Object of class "fonf_dl" fitted on the full data with best_params.

Note

- This function is single-threaded. For large grids consider a parallel wrapper (e.g. **future.apply**).
- Internal helpers flatten1() and sanitize_basis() are not exported but are documented in the source code.

Author(s)

Ufuk Beyaztas, Gizel Bakicierler Sezer

Examples

```
## Not run:
# ------------------------------------------------------------
# 1. Simulate training data
# ------------------------------------------------------------
# n <- 100
# simdata <- dgp_mixed(n, 101, model = "nonlinear")
#
# y    <- simdata$y
# x    <- simdata$x
# xscl <- simdata$x.scl
#
# -------------------------------------------------------------
# 2. Construct a small hyper-parameter grid
# -------------------------------------------------------------
# grid <- fonf_param_grid(
#   hidden_layers         = c(1, 2),
#   neurons_per_layer     = list(32, c(64, 32)),
#   activations_in_layers = list("relu", c("relu", "linear")),
#   learning_rate         = c(1e-3, 5e-4),
#   epochs                = 25
# )
#
# -------------------------------------------------------------
# 3. Tune and refit
# -------------------------------------------------------------
```

```
# tune_out <- fonf_tune(grid,
#                       resp       = y,
#                       func_cov   = x,
#                       scalar_cov = xscl,
#                       nfolds     = 5)
#
# tune_out$results           # CV table
# best_mod <- tune_out$best_model
## End(Not run)
```

---

load_air_data *Download and Load the External Air Quality Dataset*

Description

Downloads the large air_data.RData file (47 MB) from a Dropbox URL and loads it into the global environment. This is a workaround to bypass GitHub's file size limit for storing large datasets within an R package.

Usage

```
load_air_data(destdir = tempdir(), verbose = TRUE)
```

Arguments

destdir	A directory path where the downloaded air_data.RData file will be saved locally. Defaults to a temporary directory.
verbose	Logical. If TRUE, status messages are printed during download and loading.

Details

This function is intended for use with the air_data dataset used in the HDTN (Hybrid Deep Tensor Network) model for function-on-function regression. The dataset includes:

- y: a matrix of functional response curves (daily mean $PM_{2.5}$ levels),
- x: a list of 5 functional predictor matrices,
- xscl: a matrix of scalar covariates.

Due to the large size of the dataset, it is not bundled with the package and must be downloaded separately via this function.

Value

Loads the object air_data into the global environment. This object is a list with elements:

y	A matrix of dimension $n \times M$ representing functional responses.
x	A list of 5 $n \times M$ functional predictor matrices.
xscl	A matrix of dimension $n \times 3$ for scalar covariates.

Note

This function downloads a 47 MB file from Dropbox. A stable internet connection is required. The file is cached locally but may be removed between sessions unless a permanent directory is specified.

Author(s)

Ufuk Beyaztas, Gizel Bakicierler Sezer

References

Maranzano, M., and Algieri, C. (2024). ARPALData: An R package for retrieving and analyzing air quality and weather data from ARPA Lombardia (Italy). *Environmental and Ecological Statistics*, 31(2), 187–218.

Examples

```
# Download and load the dataset into the workspace
# load_air_data()

## NOTE: This example involves ultra high-dimensional functional data.
## Running the model may require a PC with at least 64 GB of RAM.

# y <- air_data$y
# x <- air_data$x
# xscl <- air_data$xscl

# ntot <- dim(y)[1]
# ntrain <- 1000
# ntest <- ntot - ntrain

# train_ind <- sample(1:ntot, ntrain, replace = FALSE)

# Training sample
# y_train <- y[train_ind, ]
# y_test  <- y[-train_ind, ]
# xscl_train <- xscl[train_ind, ]
# xscl_test  <- xscl[-train_ind, ]

# Test sample
# x_train <- x_test <- vector("list", length = 5)
# for (j in 1:5) {
#   x_train[[j]] <- x[[j]][train_ind, ]
#   x_test[[j]]  <- x[[j]][-train_ind, ]
# }

# Train HDTN approach
# nfof_model <- fonf_fit(resp = y_train, func_cov = x_train, scalar_cov = xscl_train)

# Obtain predictions with conformal prediction intervals
# band <- fonf_predict(nfof_model,
#                      func_cov_new   = x_test,
#                      scalar_cov_new = xscl_test,
#                      interval       = "conformal")
```

Index