Package ‘nFunNN’

April 28, 2024

Title Nonlinear Functional Principal Component Analysis using Neural Networks

Version 1.0

Description Implementation for 'nFunNN' method, which is a novel nonlinear functional principal component analysis method using neural networks. The crucial function of this package is nFunNNmodel().

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.1.1

Imports fda, splines, stats, torch

NeedsCompilation no

Author Rou Zhong [aut, cre],
Jingxiao Zhang [aut]

Maintainer Rou Zhong <zhong_rou@163.com>

Repository CRAN

Date/Publication 2024-04-28 09:40:02 UTC

R topics documented:

`nFunNNmodel` *Nonlinear FPCA using neural networks*

Description

Nonlinear functional principal component analysis using a transformed functional autoassociative neural network.

Usage

```
nFunNNmodel(
  X_ob,
  t_grid,
  t_grid_est,
  L_smooth,
  L,
  J,
  K,
  R,
  lr = 0.001,
  batch_size,
  n_epoch
)
```

Arguments

X_ob A matrix denoting the observed data.

t_grid A vector denoting the observation time grids on [0, 1].

t_grid_est A vector denoting the time grids that have to be predicted on [0, 1].

L_smooth An integer denoting the number of B-spline basis functions that used to smooth the observed data for the computation of the loss function.

L An integer denoting the number of B-spline basis functions for the parameters in the network.

J An integer denoting the number of neurons in the first hidden layer.

K An integer denoting the number of principal components.

R An integer denoting the number of neurons in the third hidden layer.

lr A scalar denoting the learning rate. (default: 0.001)

batch_size An integer denoting the batch size.

n_epoch An integer denoting the number of epochs.

Value

A list containing the following components:

model The resulting neural network trained by the observed data.

loss A vector denoting the averaged loss in each epoch.

Comp_time An object of class "difftime" denoting the computation time in seconds.

Examples

```
n <- 2000
m <- 51
t_grid <- seq(0, 1, length.out = m)
m_est <- 101
```

```
t_grid_est <- seq(0, 1, length.out = m_est)
err_sd <- 0.1
Z_1a <- stats::rnorm(n, 0, 3)
Z_2a <- stats::rnorm(n, 0, 2)
Z_a <- cbind(Z_1a, Z_2a)
Phi <- cbind(sin(2 * pi * t_grid), cos(2 * pi * t_grid))
Phi_est <- cbind(sin(2 * pi * t_grid_est), cos(2 * pi * t_grid_est))
X <- Z_a %*% t(Phi)
X_to_est <- Z_a %*% t(Phi_est)
X_ob <- X + matrix(stats::rnorm(n * m, 0, err_sd), nr = n, nc = m)
L_smooth <- 10
L <- 10
J <- 20
K <- 2
R <- 20
nFunNN_res <- nFunNNmodel(X_ob, t_grid, t_grid_est, L_smooth,
L, J, K, R, lr = 0.001, n_epoch = 1500, batch_size = 100)
```

nFunNN_CR *Curve reconstruction*

Description

Curve reconstruction by the trained transformed functional autoassociative neural network.

Usage

```
nFunNN_CR(model, X_ob, L, t_grid)
```

Arguments

`model` The trained transformed functional autoassociative neural network obtained from `nFunNNmodel`.

`X_ob` A `matrix` denoting the observed data from subjects that we aim to predict.

`L` An `integer` denoting the number of B-spline basis functions for the parameters in the network.

`t_grid` A `vector` denoting the observation time grids on `[0, 1]`.

Value

A torch tensor denoting the predicted values.

Examples

```
n <- 2000
m <- 51
t_grid <- seq(0, 1, length.out = m)
```

```
m_est <- 101
t_grid_est <- seq(0, 1, length.out = m_est)
err_sd <- 0.1
Z_1a <- stats::rnorm(n, 0, 3)
Z_2a <- stats::rnorm(n, 0, 2)
Z_a <- cbind(Z_1a, Z_2a)
Phi <- cbind(sin(2 * pi * t_grid), cos(2 * pi * t_grid))
Phi_est <- cbind(sin(2 * pi * t_grid_est), cos(2 * pi * t_grid_est))
X <- Z_a %*% t(Phi)
X_to_est <- Z_a %*% t(Phi_est)
X_ob <- X + matrix(stats::rnorm(n * m, 0, err_sd), nr = n, nc = m)
L_smooth <- 10
L <- 10
J <- 20
K <- 2
R <- 20
nFunNN_res <- nFunNNmodel(X_ob, t_grid, t_grid_est, L_smooth,
L, J, K, R, lr = 0.001, n_epoch = 1500, batch_size = 100)
model <- nFunNN_res$model
X_pre <- nFunNN_CR(model, X_ob, L, t_grid)
sqrt(torch::nnf_mse_loss(X_pre, torch::torch_tensor(X_to_est))$item())
```

Index