Package ‘mfaces’

October 13, 2022

Title Fast Covariance Estimation for Multivariate Sparse Functional Data

Version 0.1-4

Date 2022-07-18

Author Cai Li [aut,cre] and Luo Xiao [aut]

Maintainer Cai Li <cai.li.stats@gmail.com>

Description Multivariate functional principal component analysis via fast covariance estimation for multivariate sparse functional data or longitudinal data proposed by Li, Xiao, and Luo (2020) <doi:10.1002/sta4.245>.

Depends R (>= 2.1.4)

Imports stats, splines, Matrix, matrixcalc, mgcv, face

License GPL-3

LazyLoad yes

Repository CRAN

RoxygenNote 7.1.1

NeedsCompilation no

Date/Publication 2022-07-19 07:40:02 UTC

R topics documented:

`mfaces-package` *mfaces*

Description

Fast Covariance Estimation for Multivariate Sparse Functional Data

Details

Package:	mfaces
Type:	Package
Version:	0.1-4
Date:	2022-07-18
License:	GPL-3

Author(s)

Cai Li and Luo Xiao

Maintainer: Cai Li <cai.li.stats@gmail.com>

References

Cai Li, Luo Xiao, and Sheng Luo, 2020. Fast covariance estimation for multivariate sparse functional data. Stat, 9(1), p.e245, doi: 10.1002/sta4.245.

`mface.sparse` *Fast covariance estimation for multivariate sparse functional data*

Description

The function is to estimate the mean and covariance function from a cluster of multivariate functions/longitudinal observations.

Usage

```
mface.sparse(data, newdata = NULL,
             center = TRUE, argvals.new = NULL, knots = 7,
             knots.option = "equally-spaced",
             p = 3, m = 2,
             lambda = NULL, lambda_mean = NULL, lambda_bps = NULL,
             search.length = 14, lower = -3, upper = 10,
             calculate.scores = FALSE, pve = 0.99)
```

Arguments

`data`	a list containing all functional outcomes. Each element is a data frame with three arguments: (1) `argvals`: observation times; (2) `subj`: subject indices; (3) `y`: values of observations for each dimension. Missing values not allowed.
`newdata`	of the same strucutre as `data`; defaults to NULL, then no prediction.
`center`	logical. If TRUE, then Pspline smoothing of the population mean will be conducted and subtracted from the data before covariance smoothing; if FALSE, then the population mean will be just 0s.
`argvals.new`	a vector of observation time points to evaluate mean function, covariance function, error variance and etc. If NULL, then 100 equidistant points in the range of data time points will be used.
`knots`	the number of knots for B-spline basis functions to be used; defaults to 7. The resulting number of basis functions is the number of interior knots plus the degree of B-splines.
`knots.option`	if `knots` specifies the number of knots, then `knots.option` will be used. Default "equally-spaced", then equally-spaced knots in the range of observed time points will be selected; alternatively, "quantile": quantiles of the observed time points will be selected; see details.
`p`	the degrees of B-splines; defaults to 3.
`m`	the order of differencing penalty; defaults to 2.
`lambda`	the value of the smoothing parameter for auto-covariance smoothing; defaults to NULL.
`lambda_mean`	the value of the smoothing parameter for mean smoothing; defaults to NULL.
`lambda_bps`	the value of the smoothing parameter for cross-covariance smoothing; defaults to NULL.
`search.length`	the number of equidistant (log scale) smoothing parameters to search; defaults to 14.
`lower, upper`	bounds for log smoothing parameter; defaults are -3 and 10, respectively.
`calculate.scores`	if TRUE, scores will be calculated.
`pve`	Defaults 0.99. To select the number of eigenvalues by percentage of variance.

Details

This is a generalized version of bivariate P-splines (Eilers and Marx, 2003) for covariance smoothing of multivariate sparse functional or longitudinal data. It uses tensor product B-spline basis functions and employs differencing penalties on the assosciated parameter matrix. The smoothing parameters in the method are selected by leave-one-subject-out cross validation and is implemented with a fast algorithm.

If `center` is TRUE, then the population means will be calculated and are smoothed by univariate P-spline smoothing: `pspline` (Eilers and Marx, 1996). This univariate smoothing uses leave-one-subject-out cross validation to select the smoothing parameter.

If knots.option is "equally-spaced", then the differencing penalty in Eilers and Marx (2003) is used; if knots.option is "quantile" then the integrated squared second order derivative penalty in Wood (2016) is used.

Value

`fit`	Univariate FPCA fit for each function
`y.pred,mu.pred,Chat.diag.pred,se.pred,var.error.pred`	Predicted/estimated objects at `newdata$argvals`
`Theta`	Estimated parameter matrix
`argvals.new`	Vector of time points to evaluate population parameters
`Chat.new,Cor.new,Cor.raw.new,Chat.raw.diag.new,var.error.new`	Estimated objects at `argvals.new`
`eigenfunctions, eigenvalues`	Estimated eigenfunctions (scaled eigenvector) and eigenvalues at `argvals.new`
`var.error.hat`	Estimated objects for each outcome
`calculate.scores,rand_eff`	if `calculate.scores` is TRUE (default to FALSE), then predicted scores `rand_eff$scores` will be calculated.
`...`	...

Author(s)

Cai Li <cli9@ncsu.edu> and Luo Xiao <lxiao5@ncsu.edu>

References

Cai Li, Luo Xiao, and Sheng Luo, 2020. Fast covariance estimation for multivariate sparse functional data. Stat, 9(1), p.e245, doi: 10.1002/sta4.245.

Luo Xiao, Cai Li, William Checkley and Ciprian Crainiceanu, Fast covariance estimation for sparse functional data, Stat. Comput., doi: 10.1007/s1122201797448.

Paul Eilers and Brian Marx, Multivariate calibration with temperature interaction using two-dimensional penalized signal regression, Chemometrics and Intelligent Laboratory Systems 66 (2003), 159-174.

Paul Eilers and Brian Marx, Flexible smoothing with B-splines and penalties, Statist. Sci., 11, 89-121, 1996.

Simon N. Wood, P-splines with derivative based penalties and tensor product smoothing of unevenly distributed data, Stat. Comput., doi: 10.1007/s112220169666x.

See Also

`face.sparse` in `face`

Examples

```
## a toy example
## settings
n <- 25
sigma <- 0.1
seed <- 118

set.seed(seed)
```

```
## data generation
N1 <- sample(3:7,n,replace=TRUE)
N2 <- sample(3:7,n,replace=TRUE)
N3 <- sample(3:7,n,replace=TRUE)


subj1 <- c()
subj2 <- c()
subj3 <- c()
for(i in 1:n){
  subj1 <- c(subj1,rep(i, N1[i]))
  subj2 <- c(subj2,rep(i, N2[i]))
  subj3 <- c(subj3,rep(i, N3[i]))
}
t1 <- runif(sum(N1))
t2 <- runif(sum(N2))
t3 <- runif(sum(N3))
tnew <- seq(0,1,length=100)
y1 <- 5*sin(2*pi*t1)
y2 <- 5*cos(2*pi*t2)
y3 <- 5*(t3-1)^2

x1 <- t(matrix(rep(5*sin(2*pi*tnew),n),length(tnew),n))
x2 <- t(matrix(rep(5*cos(2*pi*tnew),n),length(tnew),n))
x3 <- t(matrix(rep(5*(tnew-1)^2,n),length(tnew),n))

psi11 <- function(x){sqrt(2/3)*sin(2*pi*x)}
psi12 <- function(x){sqrt(2/3)*cos(4*pi*x)}
psi13 <- function(x){sqrt(2/3)*sin(4*pi*x)}

psi21 <- function(x){sqrt(2/3)*sin((1-1/2)*pi*x)}
psi22 <- function(x){sqrt(2/3)*sin((2-1/2)*pi*x)}
psi23 <- function(x){sqrt(2/3)*sin((3-1/2)*pi*x)}

psi31 <- function(x){sqrt(2/3)*sin(1*pi*x)}
psi32 <- function(x){sqrt(2/3)*sin(2*pi*x)}
psi33 <- function(x){sqrt(2/3)*sin(3*pi*x)}

Lambda <- c(2,1,0.5)*3

x <- matrix(NA,nrow=n*length(tnew),ncol=3)
xi <- matrix(NA,nrow=n,ncol=3)
for(k in 1:3){xi[,k] = rnorm(n)*sqrt(Lambda[k])}

for(i in 1:n){
  seq1 <- (sum(N1[1:i])-N1[i]+1):(sum(N1[1:i]))
  seq2 <- (sum(N2[1:i])-N2[i]+1):(sum(N2[1:i]))
  seq3 <- (sum(N3[1:i])-N3[i]+1):(sum(N3[1:i]))

  Xt = xi[i,1]*c(psi11(t1[seq1]),psi21(t2[seq2]),psi31(t3[seq3])) +
    xi[i,2]*c(psi12(t1[seq1]),psi22(t2[seq2]),psi32(t3[seq3])) +
    xi[i,3]*c(psi13(t1[seq1]),psi23(t2[seq2]),psi33(t3[seq3]))
```

```
  y1[seq1] = y1[seq1] + Xt[1:N1[i]]
  y2[seq2] = y2[seq2] + Xt[N1[i]+1:N2[i]]
  y3[seq3] = y3[seq3] + Xt[N1[i]+N2[i]+1:N3[i]]

  x[((i-1)*length(tnew)+1) :(length(tnew)*i),] = c(x1[i,], x2[i,], x3[i,]) +
    xi[i,1]*c(psi11(tnew),psi21(tnew),psi31(tnew)) +
    xi[i,2]*c(psi12(tnew),psi22(tnew),psi32(tnew)) +
    xi[i,3]*c(psi13(tnew),psi23(tnew),psi33(tnew))
}

True_C <- Lambda[1]*c(psi11(tnew),psi21(tnew),psi31(tnew))%x%
  t(c(psi11(tnew), psi21(tnew), psi31(tnew))) +
  Lambda[2]*c(psi12(tnew), psi22(tnew), psi32(tnew))%x%
  t(c(psi12(tnew), psi22(tnew), psi32(tnew))) +
  Lambda[3]*c(psi13(tnew), psi23(tnew), psi33(tnew))%x%
  t(c(psi13(tnew), psi23(tnew), psi33(tnew)))


## observed data
y1 <- y1 + rnorm(sum(N1))*sigma
y2 <- y2 + rnorm(sum(N2))*sigma
y3 <- y3 + rnorm(sum(N3))*sigma

# true trajectories
x1 <- t(matrix(x[,1],length(tnew),n))
x2 <- t(matrix(x[,2],length(tnew),n))
x3 <- t(matrix(x[,3],length(tnew),n))

true_eigenfunctions <- eigen(True_C)$vectors*sqrt(length(tnew))
true_eigenvalues <- eigen(True_C)$values/length(tnew)

## organize data and apply mFACEs
data <- list("y1" = data.frame("subj"= subj1, "argvals" = t1, "y" = y1),
             "y2" = data.frame("subj"= subj2, "argvals" = t2, "y" = y2),
             "y3" = data.frame("subj"= subj3, "argvals" = t3, "y" = y3))
fit <- mface.sparse(data, argvals.new = tnew, knots = 5)


## set calculate.scores to TRUE if want to get scores
fit <- mface.sparse(data, argvals.new = tnew, knots = 5, calculate.scores = TRUE)
scores <- fit$rand_eff$scores


## prediction of several subjects
for(i in 1:2){
  sel <- lapply(data, function(x){which(x$subj==i)})
  dat_i <- mapply(function(data, sel){data[sel,]},
                  data = data, sel = sel, SIMPLIFY = FALSE)
  dat_i_pred <- lapply(dat_i, function(x){
    data.frame(subj=rep(x$subj[1],nrow(x) + length(tnew)),
               argvals = c(rep(NA,nrow(x)),tnew),
               y = rep(NA,nrow(x) + length(tnew)))
```

```
  })
  for(j in 1:length(dat_i)){
    dat_i_pred[[j]][1:nrow(dat_i[[j]]), ] <- dat_i[[j]]
  }
  pred <- predict(fit, dat_i_pred)
  y_pred <- mapply(function(pred_y.pred, dat_i){
    pred_y.pred[nrow(dat_i)+1:length(tnew)]}, pred_y.pred = pred$y.pred,
    dat_i = dat_i, SIMPLIFY = TRUE)

  pre <- pred

  Ylim = c(-12,12)
  Xlim = c(0,1)
  Ylab = bquote(y^(1))
  Xlab = "t"
  main = paste("Subject ", dat_i[[1]][1,1],sep="")
  idx = (nrow(dat_i[[1]])+1):(nrow(dat_i[[1]])+length(tnew))
  plot(dat_i[[1]][,"argvals"],dat_i[[1]][,"y"],ylim=Ylim,xlim=Xlim,ylab=Ylab,xlab=Xlab,
       main=main,cex.lab=2.0,cex.axis = 2.0,cex.main = 2.0,pch=1)
  lines(tnew,pre$y.pred$y1[idx],col="red",lwd=2)
  lines(tnew,pre$y.pred$y1[idx]-1.96*pre$se.pred$y1[idx],col="blue",lwd=2,lty=2)
  lines(tnew,pre$y.pred$y1[idx]+1.96*pre$se.pred$y1[idx],col="blue",lwd=2,lty=2)
  lines(tnew,x1[i,],col="purple",lwd=2)

  Ylab = bquote(y^(2))
  Xlab = "t"
  main = paste("Subject ", dat_i[[1]][1,1],sep="")
  idx = (nrow(dat_i[[2]])+1):(nrow(dat_i[[2]])+length(tnew))
  plot(dat_i[[2]][,"argvals"],dat_i[[2]][,"y"],ylim=Ylim,xlim=Xlim,ylab=Ylab,xlab=Xlab,
       main=main,cex.lab=2.0,cex.axis = 2.0,cex.main = 2.0,pch=1)
  lines(tnew,pre$y.pred$y2[idx],col="red",lwd=2)
  lines(tnew,pre$y.pred$y2[idx]-1.96*pre$se.pred$y2[idx],col="blue",lwd=2,lty=2)
  lines(tnew,pre$y.pred$y2[idx]+1.96*pre$se.pred$y2[idx],col="blue",lwd=2,lty=2)
  lines(tnew,x2[i,],col="purple",lwd=2)

  Ylab = bquote(y^(3))
  Xlab = "t"
  main = paste("Subject ", dat_i[[1]][1,1],sep="")
  idx = (nrow(dat_i[[3]])+1):(nrow(dat_i[[3]])+length(tnew))
  plot(dat_i[[3]][,"argvals"],dat_i[[3]][,"y"],ylim=Ylim,xlim=Xlim,ylab=Ylab,xlab=Xlab,
       main=main,cex.lab=2.0,cex.axis = 2.0,cex.main = 2.0,pch=1)
  lines(tnew,pre$y.pred$y3[idx],col="red",lwd=2)
  lines(tnew,pre$y.pred$y3[idx]-1.96*pre$se.pred$y3[idx],col="blue",lwd=2,lty=2)
  lines(tnew,pre$y.pred$y3[idx]+1.96*pre$se.pred$y3[idx],col="blue",lwd=2,lty=2)
  lines(tnew,x3[i,],col="purple",lwd=2)

}
```

`mfaces-internal` *Internal functions for mfaces package*

Description

Internal function.

Value

No return value, called for internal usage

`predict.mface.sparse` *Subject-specific curve prediction from a mface.sparse fit*

Description

Predict subject-specific curves based on a fit from "mface.sparse".

Usage

```
## S3 method for class 'mface.sparse'
predict(object, newdata, calculate.scores = T, ...)
```

Arguments

`object`	a fitted object from the R function "mface.sparse".
`newdata`	a list containing all functional outcomes. Each element is a data frame with three arguments: (1) `argvals`: observation times; (2) `subj`: subject indices; (3) `y`: values of observations for each dimension. NA values are allowed in "y" but not in the other two.
`calculate.scores`	if TRUE, scores will be calculated.
`...`	further arguments passed to or from other methods.

Details

This function makes prediction based on observed data for each subject. So for each subject, it requires at least one observed data. For the time points prediction is desired but no observation is available, just make the corresponding `data$y` as NA.

Value

`object`	A "mface.sparse" fit
`newdata`	Input data
`y.pred,mu.pred,se.pred,Chat.diag.pred,var.error.pred`	Predicted/estimated objects at the observation time points in `newdata`
`rand_eff`	if `calculate.scores` in `object` is TRUE (typically TRUE), then predicted scores `rand_eff$scores` will be calculated.
`...`	...

Author(s)

Cai Li <cli9@ncsu.edu>

References

Cai Li, Luo Xiao, and Sheng Luo, 2020. Fast covariance estimation for multivariate sparse functional data. Stat, 9(1), p.e245, doi: 10.1002/sta4.245.

Examples

```
# See the examples for "mface.sparse".
```

Index