Package 'Funclustering'

February 19, 2015

Type Package

Title A package for functional data clustering.

Version 1.0.1

Date 2013-12-20

Author Mohamed Soueidatt <mohamed.soueidatt@inria.fr>, <msoueidatt@gmail.com>*.

Maintainer Vincent KUBICKI <vincent.kubicki@inria.fr>

Description This packages proposes a model-based clustering algorithm for multivariate functional data. The parametric mixture model, based on the assumption of normality of the principal components resulting from a multivariate functional PCA, is estimated by an EM-like algorithm. The main advantage of the proposed algorithm is its ability to take into account the dependence among curves.

License GPL (>= 2)

Depends fda (>= 2.2.6), R (>= 2.15.1), methods

Imports Rcpp (>= 0.10.3)

LinkingTo Rcpp, RcppEigen

SystemRequirements GNU make

Collate 'cpp_data.R' 'funclust.r' 'input.R' 'output.R' 'plot.R' 'mfpca.R' 'harmCut.r' 'mfpcaPlot.R'

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-01-15 11:52:11

R topics documented:

`Funclustering-package` *A package for functional data clustering.*

Description

This packages proposes a model-based clustering algorithm for multivariate functional data. The parametric mixture model, based on the assumption of normality of the principal components resulting from a multivariate functional PCA, is estimated by an EM-like algorithm. The main advantage of the proposed algorithm is its ability to take into account the dependence among curves.

Details

This package include a function named funclust which allow to perform functional data clustering. This package allows also to perform a functional PCA (function mfpca) for univariate or multivariate functional data, with possibility to define different weights for each observation. The output of funclust and mfpca are a list, so one can use summary() to display the results. We can also plot the original curves with function plotOC. The function plotfd plots the curves after interpolation or smoothing. See the help of theses functions for more details.

References

J.Jacques and C.Preda (2013), Funclust: a curves clustering method using functional random variable density approximation, Neurocomputing, 112, 164-171.

J.Jacques and C.Preda (2013), Model-based clustering of multivariate functional data, Computational Statistics and Data Analysis, in press DOI 10.1016/j.csda.2012.12.004.

Examples

```
 # Multivariate
# ---------  CanadianWeather (data from the fda package) --------
 CWtime<- 1:365
 CWrange<-c(1,365)
 CWbasis <- create.fourier.basis(CWrange, nbasis=65)
 harmaccelLfd <- vec2Lfd(c(0,(2*pi/365)^2,0), rangeval=CWrange)

 # -- Build the curves --
 temperature=CanadianWeather$dailyAv[,,"Temperature.C"]
 CWfd1 <- smooth.basisPar(
 CWtime, CanadianWeather$dailyAv[,,"Temperature.C"],CWbasis,
```

```
Lfdobj=harmaccelLfd, lambda=1e-2)$fd
precipitation=CanadianWeather$dailyAv[,,"Precipitation.mm"]
CWfd2 <- smooth.basisPar(
CWtime, CanadianWeather$dailyAv[,,"Precipitation.mm"],CWbasis,
Lfdobj=harmaccelLfd, lambda=1e-2)$fd

# -- the multivariate functional data object --
CWfd=list(CWfd1,CWfd2)

# -- clustering in two class --
res=funclust(CWfd,K=2)
summary(res)
```

`cppMultiData` *The C++ code, of this package take to run from the data two main information. the coefficient in the basis expansion of the functional data, and the inner product between theses basis. cppMultidata made this task in the multivariate case.*

Description

The C++ code, of this package take to run from the data two main information. the coefficient in the basis expansion of the functional data, and the inner product between theses basis. cppMultidata made this task in the multivariate case.

Usage

```
cppMultiData(mfd)
```

Arguments

`mfd`	a list containing all dimension of the data

Value

fdData a list that containing the concatenation of the transpose of the coefficients matrix and a block diagonal matrix where each block is the inner product between the basis functions.

`cppUniData` *The C++ code, of this package take to run from the data two main information. the coefficient in the basis expansion of the functional data, and the inner product between theses basis. cppUnidata made this task in the univariate case.*

Description

The C++ code, of this package take to run from the data two main information. the coefficient in the basis expansion of the functional data, and the inner product between theses basis. cppUnidata made this task in the univariate case.

Usage

```
cppUniData(fd)
```

Arguments

`fd`	the functional data

Value

fdData a list that containing the transpose of the coefficients matrix and the inner product between these basis functions.

`funclust` *funclust, clustering multivariate functional data*

Description

This function run clustering algorithm for multivariate functional data.

Usage

```
funclust(fd, K, thd = 0.05, increaseDimension = FALSE,
  hard = FALSE, fixedDimension = integer(0), nbInit = 5,
  nbIterInit = 20, nbIteration = 200, epsilon = 1e-05)
```

Arguments

`fd`	in the univariate case fd is an object from a class fd of fda package. Otherwise, in the multivariate case, fd is a list of fd objects (fd=list(fd1,fd2,..)).
`K`	the number of clusters.
`thd`	the threshold in the Cattell scree test used to select the numbers of principal components retained for the approximation of the probability density of the functional data.

`increaseDimension`
: A logical parameter. If FALSE, the numbers of principal components are selected at each step of the algorithm according to the Cattell scree test. If TRUE, only an increase of the numbers of principal components are allowed.

`hard`
: A logical parameter. If TRUE, the algorithm is randomly initialized with "hard" cluster membership (each curves is randomly assigned to one of the clusters). if FALSE, "soft" cluster membership are used (the cluster membership probabilities are randomly choosen in the K-simplex).

`fixedDimension`
: A vector of size K which contains the numbers of principal components in the case of fixed numbers of principal components.

`epsilon`
: The stoping criterion for the EM-like algorithm: the algorithm is stopped if the difference between two successive loglikelihood is less than epsilon.

`nbInit`
: The number of small-EM used to determine the intialization of the main EM-like algorithm.

`nbIterInit`
: The maximum number of iterations for each small-EM.

`nbIteration`
: The maximum number of iteration in the main EM-like algorithm.

Details

There is multiple kind of running the function funclust. The first one is to run the function with fixed dimensions among all iteration of the algorithm. (parameter fixedDimenstion). fixedDimension must be an integer vector of size K (number of cluster). If the user gives a not integer value in fixedDimension, then this value will be convert automatically to an integer one for examplethe, the algorithm will run with a dimension 2 instead of 2.2 given by user. The second one is to run it with a dimensions variying according to the results of the scree test (parameter increaseDimension). If increaseDimension = true, then the dimensions will be constraint to only increase between to consecutuves iterations of the algorithm. else the values of the dimensions will be the results of the scree test.

Value

a list containing: tik: conditional probabilities of cluster membership for each observation cls: the estimated partition, proportions: the mixing proportions, loglikelihood: the final value of the approximated likelihood, loglikTotal: the values of the approximated likelihood along the EM-like algorithm (not necessarily increasing), aic, bic, icl: model selection criteria, dimensions: number of principal components, used in the functional data probability density approximation, selected for each clusters, dimTotal: values of the number of principal components along the EM-like algorithm, V: principal components variances per cluster

Examples

```
data(growth)
data=cbind(matrix(growth$hgtm,31,39),matrix(growth$hgtf,31,54));
t=growth$age;
splines <- create.bspline.basis(rangeval=c(1, max(t)), nbasis = 20,norder=4);
fd <- Data2fd(data, argvals=t, basisobj=splines);
# with varying dimensions (according to the results of the scree test)
res=funclust(fd,K=2)
```

```
summary(res)

# with fixed dimensions
res=funclust(fd,K=2,fixedDimension=c(2,2))


# Multivariate (deactivated by default to comply with CRAN policies)
# ---------  CanadianWeather (data from the package fda) --------
# CWtime<- 1:365
# CWrange<-c(1,365)
# CWbasis <- create.fourier.basis(CWrange, nbasis=65)
# harmaccelLfd <- vec2Lfd(c(0,(2*pi/365)^2,0), rangeval=CWrange)

# -- Build the curves ---
# temp=CanadianWeather$dailyAv[,,"Temperature.C"]
# CWfd1 <- smooth.basisPar(
# CWtime, CanadianWeather$dailyAv[,,"Temperature.C"],CWbasis,
# Lfdobj=harmaccelLfd, lambda=1e-2)$fd
# precip=CanadianWeather$dailyAv[,,"Precipitation.mm"]
# CWfd2 <- smooth.basisPar(
# CWtime, CanadianWeather$dailyAv[,,"Precipitation.mm"],CWbasis,
# Lfdobj=harmaccelLfd, lambda=1e-2)$fd

# CWfd=list(CWfd1,CWfd2)

# res=funclust(CWfd,K=2)
```

`harmsCut` *Separates the matrices of the coefficients of harmonics*

Description

This function is used in mfpca to cut the harmonic coefficient matrix. In fact mfpca call a c++ mfpca which return (among others) the matrix of the coefficients of the harmonics in the basis expantion. But in the multivariate case the coefficient matrix for all dimension are store in the same matrix, so we use this function to store each dimension in the specific matrix.

Usage

```
harmsCut(harms, nbasis)
```

Arguments

`harms`	the coefficients matrix, see the description for more details
`nbasis`	vector containing the number of basis for each dimension

Value

a list of length dimension of the data which contain the harmonic coefficients matrix for each dimension

Input-class *Constructor of Input class*

Description

This class contains the input paramaters need to run the algorithm.

Details

coefs In the univariate case this will be the transpose of the coefficients matrix. In the multivarite case this matrix will be the concatenation of the coefs matrix for each dimension of the multivariate functinal object.

basisProd In the univariate case this is the matrix of the inner product between the basis functions.In the multivariate case the m_basisProd member will be a block diagonal matrix and each block will be the matrix of the inner product between the basis functions of each dimension of the data.

K the number of clusters.

thd the threshold in the scree test to select the dimensions of the curves.

increaseDimension A logicla paramater, if true the dimensions will be constraint to increase after each iteration. A false mean that the dimensions will take their values according to the scree test.

hard A logical parameter, if true we initialize randomly the model with "hard" weights A logical parameter, if true we initialize randomly the model with hard weights (weights of each curves taking 0 or 1 according to the class membership of the curves). if false we initialize randomly with "soft" weights (weights of each curves taking a probabilities according to the class membership of the curves).

fixedDimension A vector of size "K" which contains the dimensions in the case of running algorithm with fixed dimensions.

epsilon The stoping criterion, we stop run the algorithm if the difference between two successive loglikelihood is less than epsilon.

nbInit The number of initialization to be achieve, befor running the long algorithm.

nbIterInit The maximum number of iterations in each initialization.

nbIteration The maximum number of iteration in the long algorithm.

mfpca *Multivariate functional pca*

Description

This function will run a weighted functional pca in the two cases of uni, and multivariate cases. If the observations (the curves) are given with weights, set up the parameter tik.

Usage

```
mfpca(fd, nharm, tik = numeric(0))
```

Arguments

fd	in the univariate case fd is an object from a class fd. Otherwise in the multivariate case fd is a list of fd object (fd=list(fd1,fd2,..)).
nharm	number of harmonics or principal component to be retain.
tik	the weights of the functional pca which corresponds to the weights of the curves. If don't given, then we will run a classic functional pca (without weighting the curves).

Value

When univarite functional data, the function are returning an object of calss "pca.fd", When multivariate a list of "pca.fd" object by dimension. The "pca.fd" class contains the folowing parameter: harmonics: functional data object storing the eigen function values: the eigenvalues varprop: the normalized eigenvalues (eigenvalues divide by their sum) scores: the scores matrix meanfd: the mean of the functional data object

Examples

```
data(growth)
data=cbind(matrix(growth$hgtm,31,39),matrix(growth$hgtf,31,54));
t=growth$age;
splines <- create.bspline.basis(rangeval=c(1, max(t)), nbasis = 20,norder=4);
fd <- Data2fd(data, argvals=t, basisobj=splines);
pca=mfpca(fd,nharm=2)
summary(pca)
```

---

mfpcaPlot *Plot multivariate functional pca*

Description

This function plots the functional pca.

Usage

```
mfpcaPlot(pca, grid = c())
```

Arguments

pca	is the result of mfpca. In the univariate case mfpcaPlot use the package fda and will be similar to it's function "plot.pca.fd". In multivariate functional pca, we will make a graphic window for each dimension.
grid	specify how to divide the graphics window. grid=c(n,m) divided the widow in to n lines and m columns. If user don't specify grid then he must enter <Enter> to pass to the next graphic.

Examples

```
# Multivariate
# ---------  CanadianWeather (data from the package fda) --------
CWtime<- 1:365
CWrange<-c(1,365)
CWbasis <- create.fourier.basis(CWrange, nbasis=65)
harmaccelLfd <- vec2Lfd(c(0,(2*pi/365)^2,0), rangeval=CWrange)

# -- Build the curves ---
temp=CanadianWeather$dailyAv[,,"Temperature.C"]
CWfd1 <- smooth.basisPar(
CWtime, CanadianWeather$dailyAv[,,"Temperature.C"],CWbasis,
Lfdobj=harmaccelLfd, lambda=1e-2)$fd
precip=CanadianWeather$dailyAv[,,"Precipitation.mm"]
CWfd2 <- smooth.basisPar(
CWtime, CanadianWeather$dailyAv[,,"Precipitation.mm"],CWbasis,
Lfdobj=harmaccelLfd, lambda=1e-2)$fd

CWfd=list(CWfd1,CWfd2)

pca=mfpca(CWfd,nharm=4)
mfpcaPlot(pca,grid=c(2,2))
```

Output-class *Constructor of Output class*

Description

This class contains the parameters in the output after running classification.

Details

tik a matrix of size (number of Curves) x (K), each column contains the weights of the curves in the corresponding class.

cls a vector of size number of curves, containing the index of the class for each curve.

proportion a matrix of size 1xnbClust (number of clusters), containing the estimated mixture proportions.

loglikelihood the estimated log-likelihood.

aic the value of AIC criterion.

bic the value of BIC criterion.

icl the value of ICL criterion.

dimensions a vector of size nbClust of the dimensions of the specifique dimensions of the functional data in each class.

dimTotal a matrix of size nbClust x nbRunIteration, where nbRunIteration is the number of iterations before the algorithm converge. Each column of the dimTotal matrix contain the dimensions on the coresponding iteration.

V principal components variances per cluster

empty logical parameter, and empty=TRUE if we have an empty class

---

`plotfd` *plot a functional data object*

Description

This function plots a functional data object (after smoothing or interpolation). If you want to color the curves according to a cluster's membership, please specify the parmetere col. Note: this function works only for univariate functional data.

Usage

```
plotfd(fd, col = c(1:nrow(fd$coefs)), xlab = "time",
  ylab = "value", main = "Functional data curves")
```

Arguments

`fd`	a functional data object
`col`	the color vector.
`xlab`	label of the horizontal axis
`ylab`	label of the vertical axis
`main`	the title of the graphic

Examples

```
data(growth)
data=cbind(matrix(growth$hgtm,31,39),matrix(growth$hgtf,31,54));
t=growth$age;
splines <- create.bspline.basis(rangeval=c(1, max(t)), nbasis = 20,norder=4);
fd <- Data2fd(data, argvals=t, basisobj=splines);
cls=c(rep(1,39),rep(2,54)) #there is 39 boys and 54 girls in this data
plotfd(fd,col=cls)
```

plotOC *plot Original Curves*

Description

This function plots the observed curves, before any action of smoothing or interpolation.

Usage

```
plotOC(time = 1:nrow(curves), curves, xlab = "time",
  ylab = "value", main = "Original curves")
```

Arguments

time	a vector containing the observation time for the curves. If absent the time param will be set at the vector 1:nrow(curves)
curves	the observations matrix. Each column of this matrix corresponds to one observed curve, and contains the value of the curve at discrete time points.
xlab	label of the horizontal axis
ylab	label of the vertical axis
main	the title of the graphic

Examples

```
data(growth)
curves=matrix(data=cbind(growth$hgtm,growth$hgtf),ncol=93)
time=growth$age
plotOC(time,curves)
```

Index