Package ‘adamethods’

August 4, 2020

Type Package
Title Archetypoid Algorithms and Anomaly Detection
Version 1.2.1
Date 2020-08-04
Author Guillermo Vinue, Irene Epifanio
Maintainer Guillermo Vinue <Guillermo.Vinue@uv.es>
Description Collection of several algorithms to obtain archetypes with small and large databases, and with both classical multivariate data and functional data (univariate and multivariate). Some of these algorithms also allow to detect anomalies (outliers). Please see Vinue and Epifanio (2020) <doi:10.1007/s11634-020-00412-9>.
License GPL (>= 2)
URL https://www.r-project.org
Depends R (>= 3.4.0)
Imports Anthropometry, archetypes, FNN, foreach, graphics, nnls, parallel, stats, tolerance, univOutl, utils
Suggests doParallel, fda
LazyData true
RoxygenNote 6.1.1
NeedsCompilation no
Repository CRAN
Date/Publication 2020-08-04 12:30:14 UTC

R topics documented:

adalara .......................................................... 2
adalara_no_paral ............................................. 5
archetypes_funct ............................................ 8
archetypes_funct_multiv ................................... 9
archetypes_funct_multiv_robust ........................... 11
Index

adalara  Multivariate parallel archetypoid algorithm for large applications (ADALARA)
**Description**

The ADALARA algorithm is based on the CLARA clustering algorithm. This is the parallel version of the algorithm to try to get faster results. It allows to detect anomalies (outliers). There are two different methods to detect them: the adjusted boxplot (default and most reliable option) and tolerance intervals. If needed, tolerance intervals allow to define a degree of outlierness.

**Usage**

```r
adalara(data, N, m, numArchoid, numRep, huge, prob, type_alg = "ada",
        compare = FALSE, vect_tol = c(0.95, 0.9, 0.85), alpha = 0.05,
        outl_degree = c("outl_strong", "outl_semi_strong", "outl_moderate"),
        method = "adjbox", frame)
```

**Arguments**

- `data` : Data matrix. Each row corresponds to an observation and each column corresponds to a variable. All variables are numeric. The data must have row names so that the algorithm can identify the archetypoids in every sample.
- `N` : Number of samples.
- `m` : Sample size of each sample.
- `numArchoid` : Number of archetypes/archetypoids.
- `numRep` : For each `numArchoid`, run the archetype algorithm `numRep` times.
- `huge` : Penalization added to solve the convex least squares problems.
- `prob` : Probability with values in [0,1].
- `type_alg` : String. Options are 'ada' for the non-robust adalara algorithm and 'ada_rob' for the robust adalara algorithm.
- `compare` : Boolean argument to compute the robust residual sum of squares if `type_alg = "ada"` and the non-robust if `type_alg = "ada_rob"`.
- `vect_tol` : Vector with the tolerance values. Default c(0.95, 0.9, 0.85). Needed if `method = 'toler'`.
- `alpha` : Significance level. Default 0.05. Needed if `method = 'toler'`.
- `outl_degree` : Type of outlier to identify the degree of outlierness. Default c("outl_strong", "outl_semi_strong", "outl_moderate"). Needed if `method = 'toler'`.
- `method` : Method to compute the outliers. Options allowed are 'adjbox' for using adjusted boxplots for skewed distributions, and 'toler' for using tolerance intervals.
- `frame` : Boolean value to indicate whether the frame is computed (Mair et al., 2017) or not. The frame is made up of a subset of extreme points, so the archetypoids are only computed on the frame. Low frame densities are obtained when only small portions of the data were extreme. However, high frame densities reduce this speed-up.

**Value**

A list with the following elements:

- `cases` : Optimal vector of archetypoids.
- `rss` : Optimal residual sum of squares.
- `outliers` : Outliers.
Author(s)

Guillermo Vinue, Irene Epifanio

References


See Also
do_ada, do_ada_robust, adalara_no_paral

Examples

```r
## Not run:
library(Anthropometry)
library(doParallel)

# Prepare parallelization (including the seed for reproducibility):
no_cores <- detectCores() - 1
c1 <- makeCluster(no_cores)
registerDoParallel(c1)
clusterSetRNGStream(c1, iseed = 1)

# Load data:
data(mtcars)
data <- mtcars
n <- nrow(data)

# Arguments for the archetype/archetypoid algorithm:
# Number of archetypoids:
k <- 3
numRep <- 2
huge <- 200

# Size of the random sample of observations:
m <- 10
```
# Number of samples:
\[
N \leftarrow \text{floor}(1 + (n - m)/(m - k))
\]

\[
N
\]

\[
\text{prob} \leftarrow 0.75
\]

# ADALARA algorithm:
\[
\text{preproc} \leftarrow \text{preprocessing}(\text{data}, \text{stand} = \text{TRUE}, \text{percAccomm} = 1)
\]
\[
\text{data1} \leftarrow \text{as.data.frame}(\text{preproc}\$\text{data})
\]
\[
\text{adalara_aux} \leftarrow \text{adalara}(\text{data1}, N, m, k, \text{numRep}, \text{huge}, \text{prob},
\]

"ada_rob", FALSE, method = "adjbox", frame = FALSE)

# Take the minimum RSS, which is in the second position of every sublist:
\[
\text{adalara} \leftarrow \text{adalara_aux}[\text{which.min}(\text{unlist}(\text{sapply}(\text{adalara_aux}, \text{function}(x) x[2])))][[1]]
\]
\[
\text{adalara}
\]

# End parallelization:
\[
\text{stopCluster}(\text{cl})
\]

## End(Not run)

---

### adalara_no_paral

**Multivariate non-parallel archetypoid algorithm for large applications (ADALARA)**

**Description**

The ADALARA algorithm is based on the CLARA clustering algorithm. This is the non-parallel version of the algorithm. It allows to detect anomalies (outliers). There are two different methods to detect them: the adjusted boxplot (default and most reliable option) and tolerance intervals. If needed, tolerance intervals allow to define a degree of outlierness.

**Usage**

\[
\text{adalara_no_paral}(\text{data}, \text{seed}, N, m, \text{numArchoid}, \text{numRep}, \text{huge}, \text{prob}, \text{type_alg} = \text{"ada"},
\]

\[
\text{compare} = \text{FALSE}, \text{verbose} = \text{TRUE}, \text{vect_tol} = c(0.95, 0.9, 0.85),
\]

\[
\text{alpha} = 0.05, \text{outl_degree} = c(\text{"outl_strong", \"outl_semi_strong", \"outl_moderate"},
\]

\[
\text{method} = \text{"adjbox", frame)}
\]
Arguments

data  Data matrix. Each row corresponds to an observation and each column corresponds to a variable. All variables are numeric. The data must have row names so that the algorithm can identify the archetypoids in every sample.

seed  Integer value to set the seed. This ensures reproducibility.

N  Number of samples.

m  Sample size of each sample.

numArchoid  Number of archetypes/archetypoids.

numRep  For each numArchoid, run the archetype algorithm numRep times.

huge  Penalization added to solve the convex least squares problems.

prob  Probability with values in [0,1].

type_alg  String. Options are ‘ada’ for the non-robust adalara algorithm and ‘ada_rob’ for the robust adalara algorithm.

compare  Boolean argument to compute the robust residual sum of squares if type_alg = "ada" and the non-robust if type_alg = "ada_rob".

verbose  Display progress? Default TRUE.

vect_tol  Vector the tolerance values. Default c(0.95, 0.9, 0.85). Needed if method= ‘toler’.

alpha  Significance level. Default 0.05. Needed if method= ‘toler’.

outl_degree  Type of outlier to identify the degree of outlierness. Default c("outl_strong", "outl_semi_strong", "outl_moderate"). Needed if method= ‘toler’.

method  Method to compute the outliers. Options allowed are ‘adjbox’ for using adjusted boxplots for skewed distributions, and ‘toler’ for using tolerance intervals.

frame  Boolean value to indicate whether the frame is computed (Mair et al., 2017) or not. The frame is made up of a subset of extreme points, so the archetypoids are only computed on the frame. Low frame densities are obtained when only small portions of the data were extreme. However, high frame densities reduce this speed-up.

Value

A list with the following elements:

- cases Optimal vector of archetypoids.
- rss Optimal residual sum of squares.
- outliers: Outliers.

Author(s)

Guillermo Vinue, Irene Epifanio
References


See Also
do_ada, do_ada_robust, adalara

Examples

```r
## Not run:
library(Anthropometry)

# Load data:
data(mtcars)
data <- mtcars
n <- nrow(data)

# Arguments for the archetype/archetypoid algorithm:
# Number of archetypoids:
k <- 3
numRep <- 2
huge <- 200

# Size of the random sample of observations:
m <- 10
# Number of samples:
N <- floor(1 + (n - m)/(m - k))
N

prob <- 0.75

# ADALARA algorithm:
preproc <- preprocessing(data, stand = TRUE, percAccomm = 1)
data1 <- as.data.frame(preproc$data)
res_adalara <- adalara_no_paral(data1, 1, N, m, k,
numRep, huge, prob, "ada_rob", FALSE, TRUE,
```
archetypoids_funct

Archetypoid algorithm with the functional Frobenius norm

Description

Archetypoid algorithm with the functional Frobenius norm to be used with functional data.

Usage

archetypoids_funct(numArchoid, data, huge = 200, ArchObj, PM)

Arguments

numArchoid Number of archetypoids.
data Data matrix. Each row corresponds to an observation and each column corresponds to a variable. All variables are numeric.
huge Penalization added to solve the convex least squares problems.
ArchObj The list object returned by the stepArchetypesRawData_funct function.
PM Penalty matrix obtained with eval.penalty.

Value

A list with the following elements:

• cases: Final vector of archetypoids.
• rss: Residual sum of squares corresponding to the final vector of archetypoids.
• archet_ini: Vector of initial archetypoids.
• alphas: Alpha coefficients for the final vector of archetypoids.
• resid: Matrix with the residuals.
Author(s)
Irene Epifanio

References
Epifanio, I., Functional archetype and archetypoid analysis, 2016. *Computational Statistics and Data Analysis* **104**, 24-34, [https://doi.org/10.1016/j.csda.2016.06.007](https://doi.org/10.1016/j.csda.2016.06.007)

See Also
archetypoids

Examples
```r
## Not run:
library(fda)
?growth
str(growth)
hgtm <- t(growth$hgtm)
# Create basis:
basis_fd <- create.bspine.basis(c(1,ncol(hgtm)), 10)
PM <- eval.penalty(basis_fd)
# Make fd object:
temp_points <- 1:ncol(hgtm)
temp_fd <- Data2fd(argvals = temp_points, y = growth$hgtm, basisobj = basis_fd)
data_archs <- t(temp_fd$coefs)

lass <- stepArchetypesRawData_funct(data = data_archs, numArch = 3,
numRep = 5, verbose = FALSE,
saveHistory = FALSE, PM)

af <- archetypoids_funct(3, data_archs, huge = 200, ArchObj = lass, PM)
str(af)
## End(Not run)
```

```
archetypoids_funct_multiv

Archetypoid algorithm with the functional multivariate Frobenius norm

Description
Archetypoid algorithm with the functional multivariate Frobenius norm to be used with functional data.

Usage
archetypoids_funct_multiv(numArchoid, data, huge = 200, ArchObj, PM)
```
Arguments

numArchoid  Number of archetypoids.
data  Data matrix. Each row corresponds to an observation and each column corresponds to a variable. All variables are numeric.
huge  Penalization added to solve the convex least squares problems.
ArchObj  The list object returned by the stepArchetypesRawData_funct function.
PM  Penalty matrix obtained with eval.penalty.

Value

A list with the following elements:

• cases: Final vector of archetypoids.
• rss: Residual sum of squares corresponding to the final vector of archetypoids.
• archet_ini: Vector of initial archetypoids.
• alphas: Alpha coefficients for the final vector of archetypoids.
• resid: Matrix with the residuals.

Author(s)

Irene Epifanio

References

Epifanio, I., Functional archetype and archetypoid analysis, 2016. Computational Statistics and Data Analysis 104, 24-34, https://doi.org/10.1016/j.csda.2016.06.007

See Also

archetypoids

Examples

## Not run:
library(fda)
?growth
str(growth)
hgtm <- growth$hgtm
hgtf <- growth$hgtf[,1:39]

# Create array:
nvars <- 2
data.array <- array(0, dim = c(dim(hgtm), nvars))
data.array[,1] <- as.matrix(hgtm)
data.array[,2] <- as.matrix(hgtf)
rownames(data.array) <- 1:nrow(hgtm)
colnames(data.array) <- colnames(hgtm)
str(data.array)
# Create basis:
nbasis <- 10
basis_fd <- create.bspline.basis(c(1,nrow(hgtm)), nbasis)
PM <- eval.penalty(basis_fd)

# Make fd object:
temp_points <- 1:nrow(hgtm)
temp_fd <- Data2fd(argvals = temp_points, y = data.array, basisobj = basis_fd)

X <- array(0, dim = c(dim(t(temp_fd$coefs[,1])), nvars))
X[,1] <- t(temp_fd$coef[,1])
X[,2] <- t(temp_fd$coef[,2])

# Standardize the variables:
Xs <- X
Xs[,1] <- scale(X[,1])
Xs[,2] <- scale(X[,2])

lass <- stepArchetypesRawData_funct_multiv(data = Xs, numArch = 3,
numRep = 5, verbose = FALSE,
saveHistory = FALSE, PM)

afm <- archetypoids_funct_multiv(3, Xs, huge = 200, ArchObj = lass, PM)
str(afm)

## End(Not run)

---

**archetypoids_funct_multiv_robust**

*Archetypoid algorithm with the functional multivariate robust Frobenius norm*

**Description**

Archetypoid algorithm with the functional multivariate robust Frobenius norm to be used with functional data.

**Usage**

```r
archetypoids_funct_multiv_robust(numArchoid, data, huge = 200, ArchObj, PM, prob)
```

**Arguments**

- `numArchoid` Number of archetypoids.
- `data` Data matrix. Each row corresponds to an observation and each column corresponds to a variable. All variables are numeric.
- `huge` Penalization added to solve the convex least squares problems.
- `ArchObj` The list object returned by the `stepArchetypesRawData_funct` function.
Penalty matrix obtained with `eval.penalty`.

Probability with values in $[0,1]$.

**Value**

A list with the following elements:

- `cases`: Final vector of archetypoids.
- `rss`: Residual sum of squares corresponding to the final vector of archetypoids.
- `archet_ini`: Vector of initial archetypoids.
- `alphas`: Alpha coefficients for the final vector of archetypoids.
- `resid`: Matrix with the residuals.

**Author(s)**

Irene Epifanio

**References**


**See Also**

`archetypoids`

**Examples**

```r
## Not run:
library(fda)
?growth
str(growth)
hgtm <- growth$hgtm
hgtf <- growth$hgtf[,1:39]

# Create array:
nvars <- 2
data.array <- array(0, dim = c(dim(hgtm), nvars))
data.array[,1] <- as.matrix(hgtm)
data.array[,2] <- as.matrix(hgtf)
rownames(data.array) <- 1:nrow(hgtm)
colnames(data.array) <- colnames(hgtm)
str(data.array)

# Create basis:
rbasis <- 10
basis_fd <- create.bspline.basis(c(1,nrow(hgtm)), nbasis)
PM <- eval.penalty(basis_fd)
# Make fd object:
```
temp_points <- 1:nrow(hgtm)
temp_fd <- Data2fd(argvals = temp_points, y = data.array, basisobj = basis_fd)

X <- array(0, dim = c(dim(t(temp_fd$coefs[,,1])), nvars))
X[,,1] <- t(temp_fd$coef[,,1])
X[,,2] <- t(temp_fd$coef[,,2])

# Standardize the variables:
Xs <- X
Xs[,,1] <- scale(X[,,1])
Xs[,,2] <- scale(X[,,2])

lass <- stepArchetypesRawData_funct_multiv_robust(data = Xs, numArch = 3,
numRep = 5, verbose = FALSE,
saveHistory = FALSE, PM, prob = 0.8,
nbasis, nvars)

afmr <- archetypoids_funct_multiv_robust(3, Xs, huge = 200, ArchObj = lass, PM, 0.8)
str(afmr)

## End(Not run)

---

**archetypoids_funct_robust**

*Archetypoid algorithm with the functional robust Frobenius norm*

**Description**

Archetypoid algorithm with the functional robust Frobenius norm to be used with functional data.

**Usage**

```
archetypoids_funct_robust(numArchoid, data, huge = 200, ArchObj, PM, prob)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>numArchoid</td>
<td>Number of archetypoids.</td>
</tr>
<tr>
<td>data</td>
<td>Data matrix. Each row corresponds to an observation and each column corresponds to a variable. All variables are numeric.</td>
</tr>
<tr>
<td>huge</td>
<td>Penalization added to solve the convex least squares problems.</td>
</tr>
<tr>
<td>ArchObj</td>
<td>The list object returned by the <code>stepArchetypesRawData_funct_multiv_robust</code> function.</td>
</tr>
<tr>
<td>PM</td>
<td>Penalty matrix obtained with <code>eval.penalty</code>.</td>
</tr>
<tr>
<td>prob</td>
<td>Probability with values in [0,1].</td>
</tr>
</tbody>
</table>
Value

A list with the following elements:

- cases: Final vector of archetypoids.
- rss: Residual sum of squares corresponding to the final vector of archetypoids.
- archet_ini: Vector of initial archetypoids.
- alphas: Alpha coefficients for the final vector of archetypoids.
- resid: Matrix with the residuals.

Author(s)

Irene Epifanio

References


See Also

archetypoids

Examples

```r
## Not run:
library(fda)
?growth
str(growth)
hgtm <- t(growth$hgtm)
# Create basis:
basis_fd <- create.bspline.basis(c(1,ncol(hgtm)), 10)
PM <- eval.penalty(basis_fd)
# Make fd object:
temp_points <- 1:ncol(hgtm)
temp_fd <- Data2fd(argvals = temp_points, y = growth$hgtm, basisobj = basis_fd)
data_archs <- t(temp_fd$coefs)

lass <- stepArchetypesRawData_funct_robust(data = data_archs, numArch = 3,
    numRep = 5, verbose = FALSE,
    saveHistory = FALSE, PM, prob = 0.8)

afr <- archetypoids_funct_robust(3, data_archs, huge = 200, ArchObj = lass, PM, 0.8)
str(afr)

## End(Not run)
```
archetypoids_norm_frob

Archetypoid algorithm with the Frobenius norm

Description

This function is the same as archetypoids but the 2-norm is replaced by the Frobenius norm. Thus, the comparison with the robust archetypoids can be directly made.

Usage

archetypoids_norm_frob(numArchoid, data, huge = 200, ArchObj)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>numArchoid</td>
<td>Number of archetypoids.</td>
</tr>
<tr>
<td>data</td>
<td>Data matrix. Each row corresponds to an observation and each column cor-</td>
</tr>
<tr>
<td></td>
<td>responds to a variable. All variables are numeric.</td>
</tr>
<tr>
<td>huge</td>
<td>Penalization added to solve the convex least squares problems.</td>
</tr>
<tr>
<td>ArchObj</td>
<td>The list object returned by the stepArchetypesRawData_norm_frob function.</td>
</tr>
</tbody>
</table>

Value

A list with the following elements:

- cases: Final vector of archetypoids.
- rss: Residual sum of squares corresponding to the final vector of archetypoids.
- archet_ini: Vector of initial archetypoids.
- alphas: Alpha coefficients for the final vector of archetypoids.
- resid: Matrix with the residuals.

Author(s)

Irene Epifanio

References


archetypoids_robust

See Also

archetypoids

Examples

data(mtcars)
data <- mtcars

k <- 3
numRep <- 2
huge <- 200

lass <- stepArchetypesRawData_norm_frob(data = data, numArch = k,  
numRep = numRep, verbose = FALSE)

res <- archetypoids_norm_frob(k, data, huge, ArchObj = lass)
str(res)
res$cases
res$rss

archetypoids_robust

Archetypoid algorithm with the robust Frobenius norm

Description

Robust version of the archetypoid algorithm with the Frobenius form.

Usage

archetypoids_robust(numArchoid, data, huge = 200, ArchObj, prob)

Arguments

numArchoid  Number of archetypoids.
data  Data matrix. Each row corresponds to an observation and each column corresponds to a variable. All variables are numeric.
huge  Penalization added to solve the convex least squares problems.
ArchObj  The list object returned by the stepArchetypesRawData_robust function.
prob  Probability with values in [0,1].

Value

A list with the following elements:

• cases: Final vector of archetypoids.
• rss: Residual sum of squares corresponding to the final vector of archetypoids.
bisquare_function

- archet_ini: Vector of initial archetypoids.
- alphas: Alpha coefficients for the final vector of archetypoids.
- resid: Matrix with the residuals.

Author(s)

Irene Epifanio

References


See Also

archetypoids_norm_frob

Examples

data(mtcars)
data <- mtcars

k <- 3
numRep <- 2
huge <- 200

lass <- stepArchetypesRawData_robust(data = data, numArch = k,
numRep = numRep, verbose = FALSE,
saveHistory = FALSE, prob = 0.8)

res <- archetypoids_robust(k, data, huge, ArchObj = lass, 0.8)
str(res)
res$cases
res$rss

bisquare_function

Bisquare function

Description

This function belongs to the bisquare family of loss functions. The bisquare family can better cope with extreme outliers.

Usage

bisquare_function(resid, prob, ...)

---

Description

This function belongs to the bisquare family of loss functions. The bisquare family can better cope with extreme outliers.

Usage

bisquare_function(resid, prob, ...)

---
Arguments

- resid: Vector of residuals, computed from the \( m \times n \) residuals data matrix.
- prob: Probability with values in [0,1].
- ...: Additional possible arguments.

Value

Vector of real numbers.

Author(s)

Irene Epifanio

References


Examples

```r
resid <- c(2.47, 11.85)
bisquare_function(resid, 0.8)
```

---

do_ada

Run the whole classical archetypoid analysis with the Frobenius norm

Description

This function executes the entire procedure involved in the archetypoid analysis. Firstly, the initial vector of archetypoids is obtained using the archetypal algorithm and finally, the optimal vector of archetypoids is returned.

Usage

```r
do_ada(subset, numArchoid, numRep, huge, compare = FALSE,
       vect_tol = c(0.95, 0.9, 0.85), alpha = 0.05,
       outl_degree = c("outl_strong", "outl_semi_strong", "outl_moderate"),
       method = "adjbox", prob)
```
Arguments

- **subset**: Data to obtain archetypes. In ADALARA this is a subset of the entire data frame.
- **numArchoid**: Number of archetypes/archetypoids.
- **numRep**: For each numArch, run the archetype algorithm numRep times.
- **huge**: Penalization added to solve the convex least squares problems.
- **compare**: Boolean argument to compute the robust residual sum of squares to compare these results with the ones provided by `do_ada_robust`.
- **vect_tol**: Vector the tolerance values. Default c(0.95, 0.9, 0.85). Needed if method = `'toler'`.
- **alpha**: Significance level. Default 0.05. Needed if method = `'toler'`.
- **outl_degree**: Type of outlier to identify the degree of outlierness. Default c("outl_strong", "outl_semi_strong", "outl_moderate"). Needed if method = `'toler'`.
- **method**: Method to compute the outliers. Options allowed are 'adjbox' for using adjusted boxplots for skewed distributions, and 'toler' for using tolerance intervals.
- **prob**: If compare=TRUE, probability with values in [0,1].

Value

A list with the following elements:

- cases: Final vector of archetypoids.
- alphas: Alpha coefficients for the final vector of archetypoids.
- rss: Residual sum of squares corresponding to the final vector of archetypoids.
- rss_rob: If compare=TRUE, this is the residual sum of squares using the robust Frobenius norm. Otherwise, NULL.
- resid: Vector with the residuals.
- outliers: Outliers.

Author(s)

Guillermo Vinue, Irene Epifanio

References


See Also

`stepArchetypesRawData_norm_frob, archetypoids_norm_frob`
do_ada_robust

Run the whole robust archetypoid analysis with the robust Frobenius norm

description

This function executes the entire procedure involved in the robust archetypoid analysis. Firstly, the initial vector of archetypoids is obtained using the robust archetypal algorithm and finally, the optimal vector of robust archetypoids is returned.

usage

do_ada_robust(subset, numArchoid, numRep, huge, prob, compare = FALSE, vect_tol = c(0.95, 0.9, 0.85), alpha = 0.05, outl_degree = c("outl_strong", "outl_semi_strong", "outl_moderate"), method = "adjbox")

arguments

subset Data to obtain archetypes. In ADALARA this is a subset of the entire data frame.

numArchoid Number of archetypes/archetypoids.

Examples

library(Anthropometry)
data(mtcars)
#data <- as.matrix(mtcars)
data <- mtcars

k <- 3
numRep <- 2
huge <- 200

preproc <- preprocessing(data, stand = TRUE, percAccomm = 1)
suppressWarnings(RNGversion("3.5.0"))
set.seed(2018)
res_ada <- do_ada(preproc$data, k, numRep, huge, FALSE, method = "adjbox")
str(res_ada)

res_ada1 <- do_ada(preproc$data, k, numRep, huge, FALSE,
vect_tol = c(0.95, 0.9, 0.85), alpha = 0.05,
outl_degree = c("outl_strong", "outl_semi_strong", "outl_moderate"), method = "toler")
str(res_ada1)

res_ada2 <- do_ada(preproc$data, k, numRep, huge, TRUE, method = "adjbox", prob = 0.8)
str(res_ada2)
For each `numArch`, run the archetype algorithm `numRep` times.

Penalization added to solve the convex least squares problems.

Probability with values in [0,1].

Boolean argument to compute the non-robust residual sum of squares to compare these results with the ones provided by `do_ada`.

Vector the tolerance values. Default c(0.95, 0.9, 0.85). Needed if `method='toler'`.

Significance level. Default 0.05. Needed if `method='toler'`.

Type of outlier to identify the degree of outlierness. Default c("outl_strong", "outl_semi_strong", "outl_moderate"). Needed if `method='toler'`.

Method to compute the outliers. Options allowed are 'adjbox' for using adjusted boxplots for skewed distributions, and 'toler' for using tolerance intervals.

A list with the following elements:

- cases: Final vector of archetypoids.
- alphas: Alpha coefficients for the final vector of archetypoids.
- rss: Residual sum of squares corresponding to the final vector of archetypoids.
- rss_non_rob: If `compare=TRUE`, this is the residual sum of squares using the non-robust Frobenius norm. Otherwise, NULL.
- resid: Vector of residuals.
- outliers: Outliers.

Guillermo Vinue, Irene Epifanio


See Also

`stepArchetypesRawData_robust`, `archetypoids_robust`

Examples

```r
## Not run:
library(Anthropometry)
data(mtcars)
data <- as.matrix(mtcars)
data <- mtcars
k <- 3
```
do_alphas_rss

Alphas and RSS of every set of archetypoids

Description

In the ADALARA algorithm, every time that a set of archetypoids is computed using a sample of the data, the alpha coefficients and the associated residual sum of squares (RSS) for the entire data set must be computed.

Usage

```r
do_alphas_rss(data, subset, huge, k_subset, rand_obs, alphas_subset, type_alg = "ada", PM, prob)
```

Arguments

- **data**: Data matrix with all the observations.
- **subset**: Data matrix with a sample of the data observations.
- **huge**: Penalization added to solve the convex least squares problems.
- **k_subset**: Archetypoids obtained from subset.
- **rand_obs**: Sample observations that form subset.
- **alphas_subset**: Alpha coefficients related to k_subset.
- **type_alg**: String. Options are 'ada' for the non-robust multivariate adalara algorithm, 'ada_rob' for the robust multivariate adalara algorithm, 'fada' for the non-robust fda fadalara algorithm and 'fada_rob' for the robust fda fadalara algorithm.
- **PM**: Penalty matrix obtained with `eval.penalty`. Needed when `type_alg = 'fada'` or `type_alg = 'fada_rob'`.
- **prob**: Probability with values in [0,1]. Needed when `type_alg = 'ada_rob'` or `type_alg = 'fada_rob'`.

```r
def do_alphas_rss(data, subset, huge, k_subset, rand_obs, alphas_subset, type_alg = "ada", PM, prob)
```
do_alphas_rss_multiv

Value

A list with the following elements:

- rss Real number of the residual sum of squares.
- resid_rss Matrix with the residuals.
- alphas Matrix with the alpha values.

Author(s)

Guillermo Vinue

See Also

archetypoids_norm_frob

Examples

data(mtcars)
data <- mtcars
n <- nrow(data)
m <- 10

k <- 3
numRep <- 2
huge <- 200

suppressWarnings(RNGversion("3.5.0"))
set.seed(1)
rand_obs_si <- sample(1:n, size = m)

si <- data[rand_obs_si,]
ada_si <- do_ada(si, k, numRep, huge, FALSE)

k_si <- ada_si$cases
alphas_si <- ada_si$alphas
colnames(alphas_si) <- rownames(si)

rss_si <- do_alphas_rss(data, si, huge, k_si, rand_obs_si, alphas_si, "ada")
str(rss_si)

Description

In the ADALARA algorithm, every time that a set of archetypoids is computed using a sample of
the data, the alpha coefficients and the associated residual sum of squares (RSS) for the entire data
set must be computed.
**do_alphas_rss_multiv**

Usage

```r
do_alphas_rss_multiv(data, subset, huge, k_subset, rand_obs, alphas_subset, type_alg = "ada", PM, prob, nbasis, nvars)
```

Arguments

- `data`: Data matrix with all the observations.
- `subset`: Data matrix with a sample of the data observations.
- `huge`: Penalization added to solve the convex least squares problems.
- `k_subset`: Archetypoids obtained from `subset`.
- `rand_obs`: Sample observations that form `subset`.
- `alphas_subset`: Alpha coefficients related to `k_subset`.
- `type_alg`: String. Options are 'ada' for the non-robust multivariate adalara algorithm, 'ada_rob' for the robust multivariate adalara algorithm, 'fada' for the non-robust fda fadalara algorithm and 'fada_rob' for the robust fda fadalara algorithm.
- `PM`: Penalty matrix obtained with `eval.penalty`. Needed when `type_alg = 'fada'` or `type_alg = 'fada_rob'`.
- `prob`: Probability with values in [0,1]. Needed when `type_alg = 'ada_rob'` or `type_alg = 'fada_rob'`.
- `nbasis`: Number of basis.
- `nvars`: Number of variables.

Value

A list with the following elements:

- `rss`: Real number of the residual sum of squares.
- `resid_rss`: Matrix with the residuals.
- `alphas`: Matrix with the alpha values.

Author(s)

Guillermo Vinue

See Also

`archetypoids_norm_frob`

Examples

```r
## Not run:
library(fda)
?growth
str(growth)
hgtm <- growth$hgtm
hgtf <- growth$hgtf[,1:39]
```
# Create array:
  nvars <- 2
  data.array <- array(0, dim = c(dim(hgtm), nvars))
  data.array[,1] <- as.matrix(hgtm)
  data.array[,2] <- as.matrix(hgtf)
  rownames(data.array) <- 1:nrow(hgtm)
  colnames(data.array) <- colnames(hgtm)
  str(data.array)

# Create basis:
  nbasis <- 10
  basis_fd <- create.bspline.basis(c(1,nrow(hgtm)), nbasis)
  PM <- eval.penalty(basis_fd)
# Make fd object:
  temp_points <- 1:nrow(hgtm)
  temp_fd <- Data2fd(argvals = temp_points, y = data.array, basisobj = basis_fd)

  X <- array(0, dim = c(dim(t(temp_fd$coefs[,1])), nvars))
  X[,1] <- t(temp_fd$coef[,1])
  X[,2] <- t(temp_fd$coef[,2])

# Standardize the variables:
  Xs <- X
  Xs[,1] <- scale(X[,1])
  Xs[,2] <- scale(X[,2])

# We have to give names to the dimensions to know the
# observations that were identified as archetypoids.
  dimnames(Xs) <- list(paste("Obs", 1:dim(hgtm)[2], sep = ""),
                       1:nbasis,
                       c("boys", "girls"))

  n <- dim(Xs)[1]
# Number of archetypoids:
  k <- 3
  numRep <- 20
  huge <- 200

# Size of the random sample of observations:
  m <- 15
# Number of samples:
  N <- floor(1 + (n - m)/(m - k))
  N
  prob <- 0.75
  data_alg <- Xs

  nbasis <- dim(data_alg)[2] # number of basis.
  nvars <- dim(data_alg)[3] # number of variables.
  n <- nrow(data_alg)

  suppressWarnings(RNGversion("3.5.0"))
  set.seed(1)
  rand_obs_si <- sample(1:n, size = m)
do_clean

Cleaning outliers

Description
Cleaning of the most remarkable outliers. This improves the performance of the archetypoid algorithm since it is not affected by spurious points.

Usage
do_clean(data, num_pts, range = 1.5, out_perc = 80)

Arguments
- data: Data frame with (temporal) points in the rows and observations in the columns.
- num_pts: Number of temporal points.
- range: Same parameter as in function boxplot. A value of 1.5 is enough to detect amplitude and shift outliers, while a value of 3 is needed to detect isolated outliers.
- out_perc: Minimum number of temporal points (in percentage) to consider the observation as an outlier. Needed when range=1.5.

Value
Numeric vector with the outliers.

Author(s)
Irene Epifanio

See Also
boxplot
Examples

```r
data(mtcars)
data <- mtcars
num_pts <- ncol(data)
do_clean(t(data), num_pts, 1.5, 80)
```

---

**do_clean_multiv** *(Cleaning multivariate functional outliers)*

**Description**

Cleaning of the most remarkable multivariate functional outliers. This improves the performance of the archetypoid algorithm since it is not affected by spurious points.

**Usage**

```r
do_clean_multiv(data, num_pts, range = 1.5, out_perc = 80, nbasis, nvars)
```

**Arguments**

- `data`: Data frame with (temporal) points in the rows and observations in the columns.
- `num_pts`: Number of temporal points.
- `range`: Same parameter as in function `boxplot`. A value of 1.5 is enough to detect amplitude and shift outliers, while a value of 3 is needed to detect isolated outliers.
- `out_perc`: Minimum number of temporal points (in percentage) to consider the observation as an outlier. Needed when `range=1.5`.
- `nbasis`: Number of basis.
- `nvars`: Number of variables.

**Value**

List with the outliers for each variable.

**Author(s)**

Irene Epifanio

**See Also**

`boxplot`
Examples

```r
## Not run:
library(fda)
?growth
str(growth)
hgtm <- growth$hgtm
hgtf <- growth$hgtf[,1:39]

# Create array:
nvars <- 2
data.array <- array(0, dim = c(dim(hgtm), nvars))
data.array[,1] <- as.matrix(hgtm)
data.array[,2] <- as.matrix(hgtf)
rownames(data.array) <- 1:nrow(hgtm)
colnames(data.array) <- colnames(hgtm)
str(data.array)

# Create basis:
nbasis <- 10
basis_fd <- create.bspline.basis(c(1,nrow(hgtm)), nbasis)
PM <- eval.penalty(basis_fd)
# Make fd object:
temp_points <- 1:nrow(hgtm)
temp_fd <- Data2fd(argvals = temp_points, y = data.array, basisobj = basis_fd)

X <- array(0, dim = c(dim(t(temp_fd$coefs[,1])), nvars))
X[,1] <- t(temp_fd$coef[,1])
X[,2] <- t(temp_fd$coef[,2])

# Standardize the variables:
Xs <- X
Xs[,1] <- scale(X[,1])
Xs[,2] <- scale(X[,2])
x1 <- t(Xs[,1])
for (i in 2:nvars) {
x12 <- t(Xs[,i])
x1 <- rbind(x1, x12)
}
data_all <- t(x1)

num_pts <- ncol(data_all) / nvars
range <- 3
outl <- do_clean_multiv(t(data_all), num_pts, range, out_perc, nbasis, nvars)
outl

## End(Not run)
```
do_fada | Run the whole functional archetypoid analysis with the Frobenius norm

**Description**

This function executes the entire procedure involved in the functional archetypoid analysis. Firstly, the initial vector of archetypoids is obtained using the functional archetypal algorithm and finally, the optimal vector of archetypoids is returned.

**Usage**

```r
do_fada(subset, numArchoid, numRep, huge, compare = FALSE, PM,
        vect_tol = c(0.95, 0.9, 0.85), alpha = 0.05,
        outl_degree = c("outl_strong", "outl_semi_strong", "outl_moderate"),
        method = "adjbox", prob)
```

**Arguments**

- `subset`: Data to obtain archetypes. In fadalara this is a subset of the entire data frame.
- `numArchoid`: Number of archetypes/archetypoids.
- `numRep`: For each `numArch`, run the archetype algorithm `numRep` times.
- `huge`: Penalization added to solve the convex least squares problems.
- `compare`: Boolean argument to compute the robust residual sum of squares to compare these results with the ones provided by `do_fada_robust`.
- `PM`: Penalty matrix obtained with `eval.penalty`.
- `vect_tol`: Vector the tolerance values. Default c(0.95, 0.9, 0.85). Needed if `method='toler'`.
- `alpha`: Significance level. Default 0.05. Needed if `method='toler'`.
- `outl_degree`: Type of outlier to identify the degree of outlierness. Default c("outl_strong", "outl_semi_strong", "outl_moderate"). Needed if `method='toler'`.
- `method`: Method to compute the outliers. Options allowed are ‘adjbox’ for using adjusted boxplots for skewed distributions, and ‘toler’ for using tolerance intervals.
- `prob`: If `compare=TRUE`, probability with values in [0,1].

**Value**

A list with the following elements:

- `cases`: Final vector of archetypoids.
- `alphas`: Alpha coefficients for the final vector of archetypoids.
- `rss`: Residual sum of squares corresponding to the final vector of archetypoids.
- `rss_rob`: If `compare_robust=TRUE`, this is the residual sum of squares using the robust Frobenius norm. Otherwise, NULL.
- `resid`: Vector of residuals.
- `outliers`: Outliers.
Author(s)
Guillermo Vinue, Irene Epifanio

References
Epifanio, I. Functional archetype and archetypoid analysis, 2016. Computational Statistics and Data Analysis 104, 24-34, https://doi.org/10.1016/j.csda.2016.06.007

See Also
stepArchetypesRawData_funct, archetypoids_funct

Examples

```r
## Not run:
library(fda)
?growth
str(growth)
hgtm <- t(growth$hgtm)

# Create basis:
basis_fd <- create.bspline.basis(c(1,ncol(hgtm)), 10)
PM <- eval.penalty(basis_fd)
# Make fd object:
temp_points <- 1:ncol(hgtm)
temp_fd <- Data2fd(argvals = temp_points, y = growth$hgtm, basisobj = basis_fd)
data_archs <- t(temp_fd$coefs)

suppressWarnings(RNGversion("3.5.0"))
set.seed(2018)
res_fada <- do_fada(subset = data_archs, numArchoid = 3, numRep = 5, huge = 200,
compare = FALSE, PM = PM, method = "adjbox")
str(res_fada)

suppressWarnings(RNGversion("3.5.0"))
set.seed(2018)
res_fada1 <- do_fada(subset = data_archs, numArchoid = 3, numRep = 5, huge = 200,
compare = FALSE, PM = PM,
vect_tol = c(0.95, 0.9, 0.85), alpha = 0.05,
outl_degree = c("outl_strong", "outl_semi_strong", "outl_moderate"),
method = " toler")
str(res_fada1)

res_fada2 <- do_fada(subset = data_archs, numArchoid = 3, numRep = 5, huge = 200,
compare = TRUE, PM = PM, method = "adjbox", prob = 0.8)
str(res_fada2)

## End(Not run)
```
do_fada_multiv

Run the whole archetypoid analysis with the functional multivariate Frobenius norm

**Description**

This function executes the entire procedure involved in the functional archetypoid analysis. Firstly, the initial vector of archetypoids is obtained using the functional archetypal algorithm and finally, the optimal vector of archetypoids is returned.

**Usage**

```r
do_fada_multiv(subset, numArchoid, numRep, huge, compare = FALSE, PM, 
    method = "adjbox", prob)
```

**Arguments**

- **subset**: Data to obtain archetypes. In fadalara this is a subset of the entire data frame.
- **numArchoid**: Number of archetypes/archetypoids.
- **numRep**: For each numArch, run the archetype algorithm numRep times.
- **huge**: Penalization added to solve the convex least squares problems.
- **compare**: Boolean argument to compute the robust residual sum of squares to compare these results with the ones provided by `do_fada_robust`.
- **PM**: Penalty matrix obtained with `eval.penalty`.
- **method**: Method to compute the outliers. So far the only option allowed is ‘adjbox’ for using adjusted boxplots for skewed distributions. The use of tolerance intervals might also be explored in the future for the multivariate case.
- **prob**: If compare=TRUE, probability with values in [0,1].

**Value**

A list with the following elements:

- **cases**: Final vector of archetypoids.
- **alphas**: Alpha coefficients for the final vector of archetypoids.
- **rss**: Residual sum of squares corresponding to the final vector of archetypoids.
- **rss_rob**: If compare_robust=TRUE, this is the residual sum of squares using the robust Frobenius norm. Otherwise, NULL.
- **resid**: Vector of residuals.
- **outliers**: Outliers.

**Author(s)**

Guillermo Vinue, Irene Epifanio
References


See Also

`stepArchetypesRawData_funct_multiv`, `archetypoids_funct_multiv`

Examples

```r
## Not run:
library(fda)
?growth
str(growth)
hgtm <- growth$hgtm
hgtf <- growth$hgtf[,1:39]

# Create array:
nvars <- 2
data.array <- array(0, dim = c(dim(hgtm), nvars))
data.array[,1] <- as.matrix(hgtm)
data.array[,2] <- as.matrix(hgtf)
rownames(data.array) <- 1:nrow(hgtm)
colnames(data.array) <- colnames(hgtm)
str(data.array)

# Create basis:
nbasis <- 10
basis_fd <- create.bspline.basis(c(1,nrow(hgtm)), nbasis)
PM <- eval.penalty(basis_fd)
# Make fd object:
temp_points <- 1:nrow(hgtm)
temp_fd <- Data2fd(argvals = temp_points, y = data.array, basisobj = basis_fd)

X <- array(0, dim = c(dim(t(temp_fd$coefs[,1])), nvars))
X[,1] <- t(temp_fd$coef[,1])
X[,2] <- t(temp_fd$coef[,2])

# Standardize the variables:
Xs <- X
Xs[,1] <- scale(X[,1])
Xs[,2] <- scale(X[,2])

suppressWarnings(RNGversion("3.5.0"))
set.seed(2018)
res_fada <- do_fada_multiv(subset = Xs, numArchoid = 3, numRep = 5, huge = 200,
                           compare = FALSE, PM = PM, method = "adjbox")
str(res_fada)

## End(Not run)
```
**do_fada_multiv_robust**  
Run the whole archetypoid analysis with the functional multivariate robust Frobenius norm

---

**Description**

This function executes the entire procedure involved in the functional archetypoid analysis. Firstly, the initial vector of archetypoids is obtained using the functional archetypal algorithm and finally, the optimal vector of archetypoids is returned.

**Usage**

```r
do_fada_multiv_robust(subset, numArchoid, numRep, huge, prob, compare = FALSE, PM,  
method = "adjbox")
```

**Arguments**

- **subset**: Data to obtain archetypes. In fadalara this is a subset of the entire data frame.
- **numArchoid**: Number of archetypes/archetypoids.
- **numRep**: For each numArch, run the archetype algorithm numRep times.
- **huge**: Penalization to solve the convex least squares problem, see archetypoids.
- **prob**: Probability with values in [0,1].
- **compare**: Boolean argument to compute the non-robust residual sum of squares to compare these results with the ones provided by do_fada.
- **PM**: Penalty matrix obtained with eval.penalty.
- **method**: Method to compute the outliers. So far the only option allowed is 'adjbox' for using adjusted boxplots for skewed distributions. The use of tolerance intervals might also be explored in the future for the multivariate case.

**Value**

A list with the following elements:

- **cases**: Final vector of archetypoids.
- **alphas**: Alpha coefficients for the final vector of archetypoids.
- **rss**: Residual sum of squares corresponding to the final vector of archetypoids.
- **rss_non_rob**: If compare=TRUE, this is the residual sum of squares using the non-robust Frobenius norm. Otherwise, NULL.
- **resid**: Vector of residuals.
- **outliers**: Outliers.
- **local_rel_imp**: Matrix with the local (casewise) relative importance (in percentage) of each variable for the outlier identification. Only for the multivariate case. It is relative to the outlier observation itself. The other observations are not considered for computing this importance. This procedure works because the functional variables are in the same scale, after standardizing. Otherwise, it couldn’t be interpreted like that.
- margi.rel.imp Matrix with the marginal relative importance of each variable (in percentage) for the outlier identification. Only for the multivariate case. In this case, the other points are considered, since the value of the outlier observation is compared with the remaining points.

Author(s)
Guillermo Vinue, Irene Epifanio

References

See Also
stepArchetypesRawData_funct_multiv_robust, archetypoids_funct_multiv_robust

Examples
## Not run:
library(fda)
?growth
str(growth)
hgtm <- growth$hgtm
hgtf <- growth$hgtf[,1:39]

# Create array:
nvars <- 2
data.array <- array(0, dim = c(dim(hgtm), nvars))
data.array[,1] <- as.matrix(hgtm)
data.array[,2] <- as.matrix(hgtf)
rownames(data.array) <- 1:nrow(hgtm)
colnames(data.array) <- colnames(hgtm)
str(data.array)

# Create basis:
nbasis <- 10
basis_fd <- create.bspline.basis(c(1,nrow(hgtm)), nbasis)
PM <- eval.penalty(basis_fd)
# Make fd object:
temp_points <- 1:nrow(hgtm)
temp_fd <- Data2fd(argvals = temp_points, y = data.array, basisobj = basis_fd)
X <- array(0, dim = c(dim(t(temp_fd$coefs[,1])), nvars))
X[,1] <- t(temp_fd$coef[,1])
X[,2] <- t(temp_fd$coef[,2])

# Standardize the variables:
Xs <- X
Xs[,1] <- scale(X[,1])
Xs[,2] <- scale(X[,2])
do_fada_robust

suppressWarnings(RNGversion("3.5.0"))
set.seed(2018)
res_fada <- do_fada_multiv_robust(subset = Xs, numArchoid = 3, numRep = 5, huge = 200,
prob = 0.75, compare = FALSE, PM = PM, method = "adjbox")
str(res_fada)
res_fada$cases
# [1] 8 24 29
res_fada$rss
# [1] 2.301741
## End(Not run)

---

**do_fada_robust**  
*Run the whole archetypoid analysis with the functional robust Frobenius norm*

**Description**

This function executes the entire procedure involved in the functional archetypoid analysis. Firstly, the initial vector of archetypoids is obtained using the functional archetypal algorithm and finally, the optimal vector of archetypoids is returned.

**Usage**

```r
do_fada_robust(subset, numArchoid, numRep, huge, prob, compare = FALSE, PM,
vect_tol = c(0.95, 0.9, 0.85), alpha = 0.05,
outl_degree = c("outl_strong", "outl_semi_strong", "outl_moderate"),
method = "adjbox")
```

**Arguments**

- **subset**  
  Data to obtain archetypes. In fadalara this is a subset of the entire data frame.
- **numArchoid**  
  Number of archetypes/archetypoids.
- **numRep**  
  For each numArch, run the archetype algorithm numRep times.
- **huge**  
  Penalization added to solve the convex least squares problems.
- **prob**  
  Probability with values in [0,1].
- **compare**  
  Boolean argument to compute the non-robust residual sum of squares to compare these results with the ones provided by do_fada.
- **PM**  
  Penalty matrix obtained with eval.penalty.
- **vect_tol**  
  Vector the tolerance values. Default c(0.95, 0.9, 0.85). Needed if method='toler'.
- **alpha**  
  Significance level. Default 0.05. Needed if method='toler'.
- **outl_degree**  
  Type of outlier to identify the degree of outlierness. Default c("outl_strong", "outl_semi_strong", "outl_moderate"). Needed if method='toler'.
- **method**  
  Method to compute the outliers. Options allowed are 'adjbox' for using adjusted boxplots for skewed distributions, and 'toler' for using tolerance intervals.
Value
A list with the following elements:

- cases: Final vector of archetypoids.
- alphas: Alpha coefficients for the final vector of archetypoids.
- rss: Residual sum of squares corresponding to the final vector of archetypoids.
- rss_non_rob: If compare=TRUE, this is the residual sum of squares using the non-robust Frobenius norm. Otherwise, NULL.
- resid: Vector of residuals.
- outliers: Outliers.

Author(s)
Guillermo Vinue, Irene Epifanio

References

See Also
stepArchetypesRawData_funct_robust, archetypoids_funct_robust

Examples
## Not run:
library(fda)
?growth
str(growth)
hgtm <- t(growth$hgtm)

# Create basis:
basis_fd <- create.bspline.basis(c(1,ncol(hgtm)), 10)
PM <- eval.penalty(basis_fd)
# Make fd object:
temp_points <- 1:ncol(hgtm)
temp_fd <- Data2fd(argvals = temp_points, y = growth$hgtm, basisobj = basis_fd)
data_archs <- t(temp_fd$coefs)
suppressWarnings(RNGversion("3.5.0"))
set.seed(2018)
res_fada_rob <- do_fada_robust(subset = data_archs, numArchoid = 3, numRep = 5, huge = 200,
prob = 0.75, compare = FALSE, PM = PM, method = "adjbox")
str(res_fada_rob)
suppressWarnings(RNGversion("3.5.0"))
set.seed(2018)
res_fada_rob1 <- do_fada_robust(subset = data_archs, numArchoid = 3, numRep = 5, huge = 200,
do_knno

kNN for outlier detection

Description

Ramaswamy et al. proposed the k-nearest neighbors outlier detection method (kNNo). Each point’s anomaly score is the distance to its kth nearest neighbor in the data set. Then, all points are ranked based on this distance. The higher an example’s score is, the more anomalous it is.

Usage

do_knno(data, k, top_n)

Arguments

data Data observations.
k Number of neighbors of a point that we are interested in.
top_n Total number of outliers we are interested in.

Value

Vector of outliers.

Author(s)

Guillermo Vinue

References


Examples

data(mtcars)
data <- as.matrix(mtcars)
outl <- do_knno(data, 3, 2)
outl
data[outl,]
do_outl_degree

Degree of outlierness

Description
Classification of outliers according to their degree of outlierness. They are classified using the
tolerance proportion. For instance, outliers from a 95

Usage

```r
do_outl_degree(vect_tol = c(0.95, 0.9, 0.85), resid_vect, alpha = 0.05,
  outl_degree = c("outl_strong", "outl_semi_strong", "outl_moderate"))
```

Arguments

- `vect_tol`: Vector the tolerance values. Default c(0.95, 0.9, 0.85).
- `resid_vect`: Vector of n residuals, where n was the number of rows of the data matrix.
- `alpha`: Significance level. Default 0.05.
- `outl_degree`: Type of outlier to identify the degree of outlierness. Default c("outl_strong", "outl_semi_strong", "outl_moderate").

Value
List with the type outliers.

Author(s)
Guillermo Vinue

See Also
outl_toler

Examples
```r
do_outl_degree(0.95, 1:100, 0.05, "outl_strong")
```
**Description**

The FADALARA algorithm is based on the CLARA clustering algorithm. This is the parallel version of the algorithm. It allows to detect anomalies (outliers). In the univariate case, there are two different methods to detect them: the adjusted boxplot (default and most reliable option) and tolerance intervals. In the multivariate case, only adjusted boxplots are used. If needed, tolerance intervals allow to define a degree of outlierness.

**Usage**

```r
fadalara(data, N, m, numArchoid, numRep, huge, prob, type_alg = "fada",
          compare = FALSE, PM, vect_tol = c(0.95, 0.9, 0.85), alpha = 0.05,
          outl_degree = c("outl_strong", "outl_semi_strong", "outl_moderate"),
          method = "adjbox", multiv, frame)
```

**Arguments**

- **data**
  Data matrix. Each row corresponds to an observation and each column corresponds to a variable. All variables are numeric. The data must have row names so that the algorithm can identify the archetypoids in every sample.

- **N**
  Number of samples.

- **m**
  Sample size of each sample.

- **numArchoid**
  Number of archetypes/archetypoids.

- **numRep**
  For each numArch, run the archetype algorithm numRep times.

- **huge**
  Penalization added to solve the convex least squares problems.

- **prob**
  Probability with values in [0,1].

- **type_alg**
  String. Options are ‘fada’ for the non-robust fadalara algorithm, whereas ‘fada_rob’ is for the robust fadalara algorithm.

- **compare**
  Boolean argument to compute the robust residual sum of squares if type_alg = “fada” and the non-robust if type_alg = “fada_rob”.

- **PM**
  Penalty matrix obtained with `eval.penalty`.

- **vect_tol**
  Vector the tolerance values. Default c(0.95, 0.9, 0.85). Needed if method='toler'.

- **alpha**
  Significance level. Default 0.05. Needed if method='toler'.

- **outl_degree**
  Type of outlier to identify the degree of outlierness. Default c("outl_strong", "outl_semi_strong", "outl_moderate"). Needed if method='toler'.

- **method**
  Method to compute the outliers. Options allowed are 'adjbox' for using adjusted boxplots for skewed distributions, and 'toler' for using tolerance intervals. The tolerance intervals are only computed in the univariate case, i.e., method='toler' only valid if multiv=FALSE.
multiv  Multivariate (TRUE) or univariate (FALSE) algorithm.
frame  Boolean value to indicate whether the frame is computed (Mair et al., 2017) or not. The frame is made up of a subset of extreme points, so the archetypoids are only computed on the frame. Low frame densities are obtained when only small portions of the data were extreme. However, high frame densities reduce this speed-up.

Value
A list with the following elements:

- cases  Vector of archetypoids.
- rss  Optimal residual sum of squares.
- outliers:  Outliers.
- alphas:  Matrix with the alpha coefficients.
- local_rel_imp  Matrix with the local (casewise) relative importance (in percentage) of each variable for the outlier identification. Only for the multivariate case. It is relative to the outlier observation itself. The other observations are not considered for computing this importance. This procedure works because the functional variables are in the same scale, after standardizing. Otherwise, it couldn’t be interpreted like that.
- margi_rel_imp  Matrix with the marginal relative importance of each variable (in percentage) for the outlier identification. Only for the multivariate case. In this case, the other points are considered, since the value of the outlier observation is compared with the remaining points.

Author(s)
Guillermo Vinue, Irene Epifanio

References
Epifanio, I., Functional archetype and archetypoid analysis, 2016. *Computational Statistics and Data Analysis* **104**, 24-34, [https://doi.org/10.1016/j.csda.2016.06.007](https://doi.org/10.1016/j.csda.2016.06.007)


See Also
do_fada, do_fada_robust
Examples

```r
## Not run:
library(fda)
?growth
str(growth)
hgtm <- growth$hgtm
hgtf <- growth$hgtf[1:39]

# Create array:
# vars <- 2
data.array <- array(0, dim = c(dim(hgtm), nvars))
data.array[,1] <- as.matrix(hgtm)
data.array[,2] <- as.matrix(hgtf)
rownames(data.array) <- 1:nrow(hgtm)
colnames(data.array) <- colnames(hgtm)
str(data.array)

# Create basis:
# basis_fd <- create.bspline.basis(c(1,nrow(hgtm)), nbasis)
PM <- eval.penalty(basis_fd)
# Make fd object:
temp_points <- 1:nrow(hgtm)
temp_fd <- Data2fd(argvals = temp_points, y = data.array, basisobj = basis_fd)

X <- array(0, dim = c(dim(t(temp_fd$coefs[,1])), nvars))
X[,1] <- t(temp_fd$coefs[,1])
X[,2] <- t(temp_fd$coefs[,2])

# Standardize the variables:
Xs <- X
Xs[,1] <- scale(X[,1])
Xs[,2] <- scale(X[,2])
# We have to give names to the dimensions to know the
# observations that were identified as archetypoids.
dimnames(Xs) <- list(paste("Obs", 1:dim(hgtm)[2], sep = ""),
                   1:nbasis,
c("boys", "girls"))

n <- dim(Xs)[1]
# Number of archetypoids:
k <- 3
numRep <- 20
huge <- 200

# Size of the random sample of observations:
m <- 15
# Number of samples:
N <- floor(1 + (n - m)/(m - k))
N
prob <- 0.75
data_alg <- Xs
```
# Parallel:
# Prepare parallelization (including the seed for reproducibility):
library(doParallel)
no_cores <- detectCores() - 1
no_cores
cl <- makeCluster(no_cores)
registerDoParallel(cl)
clusterSetRNGStream(cl, iseed = 2018)
res_fl <- fadalara(data = data_alg, N = N, m = m, numArchoid = k, numRep = numRep,
huge = huge, prob = prob, type_alg = "fada_rob", compare = FALSE,
PM = PM, method = "adjbox", multiv = TRUE, frame = FALSE) # frame = TRUE
stopCluster(cl)
res_fl_copy <- res_fl
res_fl <- res_fl[which.min(unlist(sapply(res_fl, function(x) x[2])))][[1]]
str(res_fl)
res_fl$cases
res_fl$rss
as.vector(res_fl$outliers)

## End(Not run)

---

**fadalara_no_paral**  
*Functional non-parallel archetypoid algorithm for large applications*  
*(FADALARA)*

**Description**

The FADALARA algorithm is based on the CLARA clustering algorithm. This is the non-parallel version of the algorithm. It allows to detect anomalies (outliers). In the univariate case, there are two different methods to detect them: the adjusted boxplot (default and most reliable option) and tolerance intervals. In the multivariate case, only adjusted boxplots are used. If needed, tolerance intervals allow to define a degree of outlierness.

**Usage**

```r
fadalara_no_paral(data, seed, N, m, numArchoid, numRep, huge, prob, type_alg = "fada",
                   compare = FALSE, verbose = TRUE, PM, vect_tol = c(0.95, 0.9, 0.85),
                   alpha = 0.05, outl_degree = c("outl_strong", "outl_semi_strong",
                                               "outl_moderate"), method = "adjbox", multiv, frame)
```

**Arguments**

- **data**  
  Data matrix. Each row corresponds to an observation and each column corresponds to a variable (temporal point). All variables are numeric. The data must have row names so that the algorithm can identify the archetypoids in every sample.
### Parameters

- **seed**: Integer value to set the seed. This ensures reproducibility.
- **N**: Number of samples.
- **m**: Sample size of each sample.
- **numArchoid**: Number of archetypes/archetypoids.
- **numRep**: For each numArch, run the archetype algorithm numRep times.
- **huge**: Penalization added to solve the convex least squares problems.
- **prob**: Probability with values in [0,1].
- **type_alg**: String. Options are 'fada' for the non-robust fadalara algorithm, whereas 'fada_rob' is for the robust fadalara algorithm.
- **compare**: Boolean argument to compute the robust residual sum of squares if type_alg = "fada" and the non-robust if type_alg = "fada_rob".
- **verbose**: Display progress? Default TRUE.
- **PM**: Penalty matrix obtained with `eval.penalty`.
- **vect_tol**: Vector the tolerance values. Default c(0.95, 0.9, 0.85). Needed if method='toler'.
- **alpha**: Significance level. Default 0.05. Needed if method='toler'.
- **outl_degree**: Type of outlier to identify the degree of outlierness. Default c("outl_strong", "outl_semi_strong", "outl_moderate"). Needed if method='toler'.
- **method**: Method to compute the outliers. Options allowed are 'adjbox' for using adjusted boxplots for skewed distributions, and 'toler' for using tolerance intervals. The tolerance intervals are only computed in the univariate case, i.e., method='toler' only valid if multiv = FALSE.
- **multiv**: Multivariate (TRUE) or univariate (FALSE) algorithm.
- **frame**: Boolean value to indicate whether the frame is computed (Mair et al., 2017) or not. The frame is made up of a subset of extreme points, so the archetypoids are only computed on the frame. Low frame densities are obtained when only small portions of the data were extreme. However, high frame densities reduce this speed-up.

### Value

A list with the following elements:

- **cases**: Vector of archetypoids.
- **rss**: Optimal residual sum of squares.
- **outliers**: Vector of outliers.
- **alphas**: Matrix with the alpha coefficients.
- **local_rel_imp**: Matrix with the local (casewise) relative importance (in percentage) of each variable for the outlier identification. Only for the multivariate case. It is relative to the outlier observation itself. The other observations are not considered for computing this importance. This procedure works because the functional variables are in the same scale, after standardizing. Otherwise, it couldn’t be interpreted like that.
- **margi_rel_imp**: Matrix with the marginal relative importance of each variable (in percentage) for the outlier identification. Only for the multivariate case. In this case, the other points are considered, since the value of the outlier observation is compared with the remaining points.
Author(s)
Guillermo Vinue, Irene Epifanio

References
Epifanio, I., Functional archetype and archetypoid analysis, 2016. *Computational Statistics and Data Analysis* **104**, 24-34, [https://doi.org/10.1016/j.csda.2016.06.007](https://doi.org/10.1016/j.csda.2016.06.007)


See Also
fadalara

Examples
```r
## Not run:
library(fda)
?growth
str(growth)
hgtm <- growth$hgtm
hgtf <- growth$hgtf[,1:39]

# Create array:
nvars <- 2
data.array <- array(0, dim = c(dim(hgtm), nvars))
data.array[,1] <- as.matrix(hgtm)
data.array[,2] <- as.matrix(hgtf)ownames(data.array) <- 1:nrow(hgtm)
colnames(data.array) <- colnames(hgtm)
str(data.array)

# Create basis:
nbasis <- 10
basis_fd <- create.bspline.basis(c(1,nrow(hgtm)), nbasis)
PM <- eval.penalty(basis_fd)
# Make fd object:
temp_points <- 1:nrow(hgtm)
temp_fd <- Data2fd(argvals = temp_points, y = data.array, basisobj = basis_fd)
X <- array(0, dim = c(dim(t(temp_fd$coefs[,1])), nvars))
```
X[,1] <- t(temp_fd$coef[,1])
X[,2] <- t(temp_fd$coef[,2])

# Standardize the variables:
Xs <- X
Xs[,1] <- scale(X[,1])
Xs[,2] <- scale(X[,2])

# We have to give names to the dimensions to know the
# observations that were identified as archetypoids.
dimnames(Xs) <- list(paste("Obs", 1:dim(hgtm)[2], sep = ""),
                   1:nbasis,
                   c("boys", "girls"))

n <- dim(Xs)[1]
# Number of archetypoids:
k <- 3
numRep <- 20
huge <- 200

# Size of the random sample of observations:
m <- 15
# Number of samples:
N <- floor(1 + (n - m)/(m - k))
N
prob <- 0.75
data_alg <- Xs

seed <- 2018
res_fl <- fadalara_no_paral(data = data_alg, seed = seed, N = N, m = m,
                            numArchoid = k, numRep = numRep, huge = huge,
                            prob = prob, type_alg = "fada_rob", compare = FALSE,
                            verbose = TRUE, PM = PM, method = "adjbox", multiv = TRUE,
                            frame = FALSE) # frame = TRUE

str(res_fl)
res_fl$cases
res_fl$rss
as.vector(res_fl$outliers)

## End(Not run)

---

**Description**

Computing the frame with the approach by Mair et al. (2017).
Usage

\texttt{frame_in_r(X)}

Arguments

\texttt{X} \hspace{1cm} \text{Data frame.}

Value

Vector with the observations that belong to the frame.

Author(s)

Sebastian Mair, code kindly provided by him.

References


Examples

```r
## Not run:
X <- mtcars
q <- frame_in_r(X)
H <- X[q,]
q
## End(Not run)
```

---

**frobenius_norm**  
*Frobenius norm*

Description

Computes the Frobenius norm.

Usage

\texttt{frobenius_norm(m)}

Arguments

\texttt{m} \hspace{1cm} \text{Data matrix with the residuals. This matrix has the same dimensions as the original data matrix.}
**Details**

Residuals are vectors. If there are p variables (columns), for every observation there is a residual that is a p-dimensional vector. If there are n observations, the residuals are an n times p matrix.

**Value**

Real number.

**Author(s)**

Guillermo Vinue, Irene Epifanio

**References**


**Examples**

```r
mat <- matrix(1:4, nrow = 2)
frobenius_norm(mat)
```

---

**frobenius_norm_funct**  
*Functional Frobenius norm*

**Description**

Computes the functional Frobenius norm.

**Usage**

```r
frobenius_norm_funct(m, PM)
```

**Arguments**

- `m`  
  Data matrix with the residuals. This matrix has the same dimensions as the original data matrix.

- `PM`  
  Penalty matrix obtained with `eval.penalty`.  

```r
dim(m)
```
Details

Residuals are vectors. If there are $p$ variables (columns), for every observation there is a residual that there is a $p$-dimensional vector. If there are $n$ observations, the residuals are an $n$ times $p$ matrix.

Value

Real number.

Author(s)

Irene Epifanio

References

Epifanio, I., Functional archetype and archetypoid analysis, 2016. *Computational Statistics and Data Analysis* **104**, 24-34, [https://doi.org/10.1016/j.csda.2016.06.007](https://doi.org/10.1016/j.csda.2016.06.007)

Examples

```r
library(fda)
mat <- matrix(1:9, nrow = 3)
fbasis <- create.fourier.basis(rangeval = c(1, 32), nbasis = 3)
PM <- eval.penalty(fbasis)
frobenius_norm_funct(mat, PM)
```

---

**frobenius_norm_funct_multiv**

*Functional multivariate Frobenius norm*

Description

Computes the functional multivariate Frobenius norm.

Usage

```r
frobenius_norm_funct_multiv(m, PM)
```

Arguments

- `m` Data matrix with the residuals. This matrix has the same dimensions as the original data matrix.
- `PM` Penalty matrix obtained with `eval.penalty`.

Details

Residuals are vectors. If there are $p$ variables (columns), for every observation there is a residual that there is a $p$-dimensional vector. If there are $n$ observations, the residuals are an $n$ times $p$ matrix.
Value

Real number.

Author(s)

Irene Epifanio

References


Examples

```r
mat <- matrix(1:400, ncol = 20)
PM <- matrix(1:100, ncol = 10)
frobenius_norm_funct_multiv(mat, PM)
```

---

**frobenius_norm_funct_multiv_robust**

*Functional multivariate robust Frobenius norm*

Description

Computes the functional multivariate robust Frobenius norm.

Usage

```r
frobenius_norm_funct_multiv_robust(m, PM, prob, nbasis, nvars)
```

Arguments

- `m`: Data matrix with the residuals. This matrix has the same dimensions as the original data matrix.
- `PM`: Penalty matrix obtained with `eval.penalty`.
- `prob`: Probability with values in [0,1].
- `nbasis`: Number of basis.
- `nvars`: Number of variables.

Details

Residuals are vectors. If there are p variables (columns), for every observation there is a residual that is a p-dimensional vector. If there are n observations, the residuals are an n times p matrix.
frobenius_norm_funct_robust

Value
Real number.

Author(s)
Irene Epifanio

References

Examples
```r
mat <- matrix(1:400, ncol = 20)
PM <- matrix(1:100, ncol = 10)
frobenius_norm_funct_multiv_robust(mat, PM, 0.8, 10, 2)
```

---

frobenius_norm_funct_robust

*Functional robust Frobenius norm*

Description
Computes the functional robust Frobenius norm.

Usage
```r
frobenius_norm_funct_robust(m, PM, prob)
```

Arguments
- `m` Data matrix with the residuals. This matrix has the same dimensions as the original data matrix.
- `PM` Penalty matrix obtained with `eval.penalty`.
- `prob` Probability with values in [0,1].

Details
Residuals are vectors. If there are p variables (columns), for every observation there is a residual that there is a p-dimensional vector. If there are n observations, the residuals are an n times p matrix.

Value
Real number.
Author(s)
Irene Epifanio

References

Examples

```r
library(fda)
mat <- matrix(1:9, nrow = 3)
fbasis <- create.fourier.basis(rangeval = c(1, 32), nbasis = 3)
PM <- eval.penalty(fbasis)
frobenius_norm_funct_robust(mat, PM, 0.8)
```

Description
Computes the robust Frobenius norm.

Usage

```r
frobenius_norm_robust(m, prob)
```

Arguments

- `m` Data matrix with the residuals. This matrix has the same dimensions as the original data matrix.
- `prob` Probability with values in [0,1].

Details
Residuals are vectors. If there are p variables (columns), for every observation there is a residual that is a p-dimensional vector. If there are n observations, the residuals are an n times p matrix.

Value
Real number.

Author(s)
Irene Epifanio
References


Examples

```r
mat <- matrix(1:4, nrow = 2)
frobenius_norm_robust(mat, 0.8)
```

---

**int_prod_mat**

*Interior product between matrices*

Description

Helper function to compute the Frobenius norm.

Usage

```r
int_prod_mat(m)
```

Arguments

- `m` Data matrix.

Value

Data matrix.

Author(s)

Irene Epifanio

References


Examples

```r
mat <- matrix(1:4, nrow = 2)
int_prod_mat(mat)
```
Description

Helper function to compute the Frobenius norm in the functional data analysis (FDA) scenario.

Usage

int_prod_mat_funct(m, PM)

Arguments

m
Data matrix.

PM
Penalty matrix obtained with eval.penalty.

Value

Data matrix.

Author(s)

Irene Epifanio

References


Examples

library(fda)
mat <- matrix(1:9, nrow = 3)
fbasis <- create.fourier.basis(rangeval = c(1, 32), nbasis = 3)
PM <- eval.penalty(fbasis)
int_prod_mat_funct(mat, PM)
int_prod_mat_sq  

**Description**

Helper function to compute the robust Frobenius norm.

**Usage**

```
int_prod_mat_sq(m)
```

**Arguments**

- `m`: Data matrix.

**Value**

Data matrix.

**Author(s)**

Irene Epifanio

**References**


**Examples**

```r
mat <- matrix(1:4, nrow = 2)
int_prod_mat_sq(mat)
```

---

int_prod_mat_sq_funct  

**Description**

Helper function to compute the robust Frobenius norm in the functional data analysis (FDA) scenario.

**Usage**

```
int_prod_mat_sq_funct(m, PM)
```

---
outl_toler

Arguments

- `m` Data matrix.
- `PM` Penalty matrix obtained with `eval.penalty`.

Value

Data matrix.

Author(s)

Irene Epifanio

References


Examples

```r
library(fda)
mat <- matrix(1:9, nrow = 3)
fbasis <- create.fourier.basis(rangeval = c(1, 32), nbasis = 3)
PM <- eval.penalty(fbasis)
int_prod_mat_sq_funct(mat, PM)
```

outl_toler  Tolerance outliers

Description

Outliers according to a tolerance interval. This function is used by the archetypoid algorithms to identify the outliers. See the function `nptol.int` in package `tolerance`.

Usage

```r
outl_toler(p_tol = 0.95, resid_vect, alpha = 0.05)
```

Arguments

- `p_tol` The proportion of observations to be covered by this tolerance interval.
- `resid_vect` Vector of `n` residuals, where `n` was the number of rows of the data matrix.
- `alpha` Significance level.

Value

Vector with the outliers.
Author(s)

Guillermo Vinue

References


See Also

adalara, fadalara, do_outl_degree

Examples

outl_toler(0.95, 1:100, 0.05)

stepArchetypesRawData_funct

Archetype algorithm to raw data with the functional Frobenius norm

Description

This is a slight modification of stepArchetypesRawData to use the functional archetype algorithm with the Frobenius norm.

Usage

stepArchetypesRawData_funct(data, numArch, numRep = 3,
   verbose = TRUE, saveHistory = FALSE, PM)

Arguments

data Data to obtain archetypes.
numArch Number of archetypes to compute, from 1 to numArch.
numRep For each numArch, run the archetype algorithm numRep times.
verbose If TRUE, the progress during execution is shown.
saveHistory Save execution steps.
PM Penalty matrix obtained with eval.penalty.

Value

A list with the archetypes.

Author(s)

Irene Epifanio
References


Epifanio, I., Functional archetype and archetypoid analysis, 2016. Computational Statistics and Data Analysis 104, 24-34, https://doi.org/10.1016/j.csda.2016.06.007


Examples

```r
## Not run:
library(fda)
?growth
str(growth)
htm <- t(growth$hgtm)
# Create basis:
basis_fd <- create.bspline.basis(c(1,ncol(htm)), 10)
PM <- eval.penalty(basis_fd)
# Make fd object:
temp_points <- 1:ncol(htm)
temp_fd <- Data2fd(argvals = temp_points, y = growth$hgtm, basisobj = basis_fd)
data_archs <- t(temp_fd$coefs)
lass <- stepArchetypesRawData_funct(data = data_archs, numArch = 3,
                                   numRep = 5, verbose = FALSE,
                                   saveHistory = FALSE, PM)
str(lass)
length(lass[[1]])
class(lass[[1]])
class(lass[[1]][[5]])
## End(Not run)
```

stepArchetypesRawData_funct_multiv

Archetype algorithm to raw data with the functional multivariate Frobenius norm

Description

This is a slight modification of `stepArchetypesRawData` to use the functional archetype algorithm with the multivariate Frobenius norm.

Usage

```r
stepArchetypesRawData_funct_multiv(data, numArch, numRep = 3,
                                   verbose = TRUE, saveHistory = FALSE, PM)
```
Arguments

- `data`: Data to obtain archetypes.
- `numArch`: Number of archetypes to compute, from 1 to `numArch`.
- `numRep`: For each `numArch`, run the archetype algorithm `numRep` times.
- `verbose`: If TRUE, the progress during execution is shown.
- `saveHistory`: Save execution steps.
- `PM`: Penalty matrix obtained with `eval.penalty`.

Value

A list with the archetypes.

Author(s)

Irene Epifanio

References


Epifanio, I., Functional archetype and archetypoid analysis, 2016. *Computational Statistics and Data Analysis* **104**, 24-34, [https://doi.org/10.1016/j.csda.2016.06.007](https://doi.org/10.1016/j.csda.2016.06.007)


Examples

```R
## Not run:
library(fda)
?growth
str(growth)

hgtm <- growth$hgtm
hgtf <- growth$hgtf[,1:39]

# Create array:
nvars <- 2
data.array <- array(0, dim = c(dim(hgtm), nvars))
data.array[,1] <- as.matrix(hgtm)
data.array[,2] <- as.matrix(hgtf)
rownames(data.array) <- 1:nrow(hgtm)
colnames(data.array) <- colnames(hgtm)
str(data.array)

# Create basis:
nbasis <- 10
basis_fd <- create.bspline.basis(c(1,nrow(hgtm)), nbasis)
PM <- eval.penalty(basis_fd)
# Make fd object:
```
temp_points <- 1:nrow(hgtm)
temp_fd <- Data2fd(argvals = temp_points, y = data.array, basisobj = basis_fd)

X <- array(0, dim = c(dim(t(temp_fd$coefs[,1])), nvars))
X[,1] <- t(temp_fd$coef[,1])
X[,2] <- t(temp_fd$coef[,2])

# Standardize the variables:
Xs <- X
Xs[,1] <- scale(X[,1])
Xs[,2] <- scale(X[,2])

lass <- stepArchetypesRawData_funct_multiv_robust(data = Xs, numArch = 3, numRep = 5, verbose = FALSE, saveHistory = FALSE, PM)

str(lass)
length(lass[[1]])
class(lass[[1]])
class(lass[[1]][[5]])

## End(Not run)

---

**stepArchetypesRawData_funct_multiv_robust**

Archetype algorithm to raw data with the functional multivariate robust Frobenius norm

---

**Description**

This is a slight modification of `stepArchetypesRawData` to use the functional archetype algorithm with the multivariate Frobenius norm.

**Usage**

```
stepArchetypesRawData_funct_multiv_robust(data, numArch, numRep = 3, verbose = TRUE, saveHistory = FALSE, PM, prob, nbasis, nvars)
```

**Arguments**

- `data` Data to obtain archetypes.
- `numArch` Number of archetypes to compute, from 1 to numArch.
- `numRep` For each numArch, run the archetype algorithm numRep times.
- `verbose` If TRUE, the progress during execution is shown.
- `saveHistory` Save execution steps.
- `PM` Penalty matrix obtained with `eval.penalty`.
stepArchetypesRawData_funct_multiv_robust

prob Probability with values in [0,1].
nbasis Number of basis.
nvars Number of variables.

Value
A list with the archetypes.

Author(s)
Irene Epifanio

References


Examples
```r
## Not run:
library(fda)
?growth
str(growth)
hgtm <- growth$hgtm
hgtf <- growth$hgtf[,1:39]

# Create array:
nvars <- 2
data.array <- array(0, dim = c(dim(hgtm), nvars))
data.array[,1] <- as.matrix(hgtm)
data.array[,2] <- as.matrix(hgtf)ownames(data.array) <- 1:nrow(hgtm)
colnames(data.array) <- colnames(hgtm)
str(data.array)

# Create basis:
nbasis <- 10
basis_fd <- create.bspline.basis(c(1,nrow(hgtm)), nbasis)
PM <- eval.penalty(basis_fd)
# Make fd object:
temp_points <- 1:nrow(hgtm)
temp_fd <- Data2Fd(argvals = temp_points, y = data.array, basisobj = basis_fd)
```
X <- array(0, dim = c(dim(t(temp_fd$coefs[,1])), nvars))
X[,1] <- t(temp_fd$coef[,1])
X[,2] <- t(temp_fd$coef[,2])

# Standardize the variables:
Xs <- X
Xs[,1] <- scale(X[,1])
Xs[,2] <- scale(X[,2])

lass <- stepArchetypesRawData_funct_multiv_robust(data = Xs, numArch = 3,
numRep = 5, verbose = FALSE,
saveHistory = FALSE, PM, prob = 0.8,
nbasis, nvars)

str(lass)
length(lass[[1]])
class(lass[[1]])
class(lass[[1]][[5]])

## End(Not run)

---

**stepArchetypesRawData_funct_robust**

*Archetype algorithm to raw data with the functional robust Frobenius norm*

**Description**

This is a slight modification of `stepArchetypesRawData` to use the functional archetype algorithm with the functional robust Frobenius norm.

**Usage**

```r
stepArchetypesRawData_funct_robust(data, numArch, numRep = 3,
verbose = TRUE, saveHistory = FALSE, PM, prob)
```

**Arguments**

- `data`: Data to obtain archetypes.
- `numArch`: Number of archetypes to compute, from 1 to `numArch`.
- `numRep`: For each `numArch`, run the archetype algorithm `numRep` times.
- `verbose`: If TRUE, the progress during execution is shown.
- `saveHistory`: Save execution steps.
- `PM`: Penalty matrix obtained with `eval.penalty`.
- `prob`: Probability with values in [0,1].
Value

A list with the archetypes.

Author(s)

Irene Epifanio

References


Epifanio, I., Functional archetype and archetypoid analysis, 2016. *Computational Statistics and Data Analysis* **104**, 24-34, [https://doi.org/10.1016/j.csda.2016.06.007](https://doi.org/10.1016/j.csda.2016.06.007)


Examples

```r
## Not run:
library(fda)
?growth
growth
hgtm <- t(growth$hgtm)
# Create basis:
basis_fd <- create.bspline.basis(c(1,ncol(hgtm)), 10)
PM <- eval.penalty(basis_fd)
# Make fd object:
temp_points <- 1:ncol(hgtm)
temp_fd <- Data2fd(argvals = temp_points, y = growth$hgtm, basisobj = basis_fd)
data_archs <- t(temp_fd$coefs)

lass <- stepArchetypesRawData_funct_robust(data = data_archs, numArch = 3,
                                          numRep = 5, verbose = FALSE,
                                          saveHistory = FALSE, PM, prob = 0.8)

str(lass)
length(lass[[1]])
class(lass[[1]])
class(lass[[1]][[5]])
```

## End(Not run)
Description

This is a slight modification of `stepArchetypesRawData` to use the archetype algorithm with the Frobenius norm.

Usage

```r
stepArchetypesRawData_norm_frob(data, numArch, numRep = 3,
   verbose = TRUE, saveHistory = FALSE)
```

Arguments

- `data`  
  Data to obtain archetypes.
- `numArch`  
  Number of archetypes to compute, from 1 to `numArch`.
- `numRep`  
  For each `numArch`, run the archetype algorithm `numRep` times.
- `verbose`  
  If TRUE, the progress during execution is shown.
- `saveHistory`  
  Save execution steps.

Value

A list with the archetypes.

Author(s)

Irene Epifanio

References


See Also

`stepArchetypesRawData`, `stepArchetypes`
Examples

```r
data(mtcars)
data <- as.matrix(mtcars)
numArch <- 5
numRep <- 2
lass <- stepArchetypesRawData_norm_frob(data = data, numArch = 1:numArch,
numRep = numRep, verbose = FALSE)
str(lass)
length(lass[[1]])
class(lass[[1]])
```

---

**stepArchetypesRawData_robust**

*Archetype algorithm to raw data with the robust Frobenius norm*

Description

This is a slight modification of `stepArchetypesRawData` to use the archetype algorithm with the robust Frobenius norm.

Usage

```r
stepArchetypesRawData_robust(data, numArch, numRep = 3,
   verbose = TRUE, saveHistory = FALSE, prob)
```

Arguments

- `data` Data to obtain archetypes.
- `numArch` Number of archetypes to compute, from 1 to `numArch`.
- `numRep` For each `numArch`, run the archetype algorithm `numRep` times.
- `verbose` If TRUE, the progress during execution is shown.
- `saveHistory` Save execution steps.
- `prob` Probability with values in [0,1].

Value

A list with the archetypes.

Author(s)

Irene Epifanio
References


See Also

stepArchetypesRawData_norm_frob

Examples

data(mtcars)
data <- as.matrix(mtcars)
numArch <- 5
numRep <- 2

lass <- stepArchetypesRawData_robust(data = data, numArch = 1:numArch,
numRep = numRep, verbose = FALSE,
saveHistory = FALSE, prob = 0.8)

str(lass)
length(lass[[1]])
class(lass[[1]])
Index

adalara, 2, 7, 56
adalara_no_paral, 4, 5
archetypoids, 9, 10, 12, 14–16, 33
archetypoids_func, 8, 30
archetypoids_func_multiv, 9, 32
archetypoids_func_multiv_robust, 11, 34
archetypoids_func_robust, 13, 36
archetypoids_norm_frob, 15, 17, 19, 23, 24
archetypoids_robust, 16, 21
bisquare_function, 17
boxplot, 26, 27
do_ada, 4, 7, 18, 21
do_ada_robust, 4, 7, 19, 20
do_alphas_rss, 22
do_alphas_rss_multiv, 23
do_clean, 26
do_clean_multiv, 27
do_fada, 28, 33, 35, 40
do_fada_multiv, 31
do_fada_multiv_robust, 33
do_fada_robust, 29, 31, 35, 40
do_knno, 37
do_outl_degree, 38, 56
eval.penalty, 8, 10, 12, 13, 22, 24, 29, 31, 33, 35, 39, 43, 47–50, 53, 55, 56, 58, 59, 61
fadalara, 39, 44, 56
fadalara_no_paral, 42
frame_in_r, 45
frobenius_norm, 46
frobenius_norm_func, 47
frobenius_norm_func_multiv, 48
frobenius_norm_func_multiv_robust, 49
frobenius_norm_func_robust, 50
frobenius_norm_robust, 51
int_prod_mat, 52
int_prod_mat_func, 53
int_prod_mat_sq, 54
int_prod_mat_sq_func, 54
outl_toler, 38, 55
stepArchetypes, 63
stepArchetypesRawData, 56, 57, 59, 61, 63
stepArchetypesRawData_func, 8, 10, 11, 30, 56
stepArchetypesRawData_func_multiv, 32, 57
stepArchetypesRawData_func_multiv_robust, 34, 59
stepArchetypesRawData_func_robust, 13, 36, 61
stepArchetypesRawData_norm_frob, 15, 19, 63, 65
stepArchetypesRawData_robust, 16, 21, 64