Package 'briKmeans'

July 22, 2025

Version 1.0

Date 2022-07-20

Title Package for Brik, Fabrik and Fdebrik Algorithms to Initialise Kmeans

Author Javier Albert Smet <javas@kth.se> and Aurora Torrente <etorrent@est-econ.uc3m.es>. Alice Parodi, Mirco Patriarca, Laura Sangalli, Piercesare Secchi, Simone Vantini and Valeria Vitelli, as contributors.

Maintainer Aurora Torrente <etorrent@est-econ.uc3m.es>

Depends R (>= 3.1.0), boot, cluster, depthTools, splines, splines2, stats

Imports methods

Description Implementation of the BRIk, FABRIk and FDEBRIk algorithms to initialise k-means. These methods are intended for the clustering of multivariate and functional data, respectively. They make use of the Modified Band Depth and bootstrap to identify appropriate initial seeds for k-means, which are proven to be better options than many techniques in the literature. Torrente and Romo (2021) <doi:10.1007/s00357-020-09372-3> It makes use of the functions kma and kma.similarity, from the archived package fdakma, by Alice Parodi et al.

License GPL (>= 3)

NeedsCompilation no

Repository CRAN

Date/Publication 2022-07-21 08:40:10 UTC

Contents

`brik` *Computation of Initial Seeds and Kmeans Results*

Description

`brik` computes appropriate seeds –based on bootstrap and the MBD depth– to initialise k-means, which is then run.

Usage

```
brik(x, k, method="Ward", nstart=1, B=10, J = 2, ...)
```

Arguments

`x`	a data matrix containing `N` observations (individuals) by rows and `d` variables (features) by columns
`k`	number of clusters
`method`	clustering algorithm used to cluster the cluster centres from the bootstrapped replicates; `Ward`, by default. Currently, only `pam` and randomly initialised `kmeans` are implemented
`nstart`	number of random initialisations when using the `kmeans` method to cluster the cluster centres
`B`	number of bootstrap replicates to be generated
`J`	number of observations used to build the bands for the MBD computation. Currently, only the value J=2 can be used
`...`	additional arguments to be passed to the `kmeans` function for the final clustering; at this stage `nstart` is set to 1, as the initial seeds are fixed

Details

The brik algorithm is a simple, computationally feasible method, which provides k-means with a set of initial seeds to cluster datasets of arbitrary dimensions. It consists of two stages: first, a set of cluster centers is obtained by applying k-means to bootstrap replications of the original data to be, next, clustered; the deepest point in each assembled cluster is returned as initial seeds for k-means.

Value

`seeds`	a matrix of size `k` x `d` containing the initial seeds obtained with the BRIk algorithm
`km`	an object of class `kmeans` corresponding to the run of kmeans on `x` with starting points `seeds`

Author(s)

Javier Albert Smet <javas@kth.se> and Aurora Torrente <etorrent@est-econ.uc3m.es>

References

Torrente, A. and Romo, J. (2020). Initializing k-means Clustering by Bootstrap and Data Depth. *J Classif* (2020). https://doi.org/10.1007/s00357-020-09372-3.

Examples

```
## brik algorithm
    ## simulated data
    set.seed(0)
    g1 <- matrix(rnorm(200,0,3), 25, 8) ; g1[,1]<-g1[,1]+4;
    g2 <- matrix(rnorm(200,0,3), 25, 8) ; g2[,1]<-g2[,1]+4; g2[,3]<-g2[,3]-4
    g3 <- matrix(rnorm(200,0,3), 25, 8) ; g3[,1]<-g3[,1]+4; g3[,3]<-g3[,3]+4

    x <- rbind(g1,g2,g3)
    labels <-c(rep(1,25),rep(2,25),rep(3,25))

    C1 <- kmeans(x,3)
    C2 <- brik(x,3,B=25)

    table(C1$cluster, labels)
    table(C2$km$cluster, labels)
```

---

elbowRule *Selection of Appropriate DF Parameter Based on an Elbow Rule for the Distortion*

Description

`elbowRule` runs the FABRIk algorithm for different degrees of freedom (DF) and suggests the best of such values as the one where the minimum distortion is obtained. An optional visualization of the computed values allows the choice of alternative suitable DF values based on an elbow-like rule.

Usage

```
elbowRule(x, k, method="Ward", nstart=1, B = 10, J = 2, x.coord = NULL, OSF = 1,
    vect = NULL, intercept = TRUE, degPolyn = 3, degFr = 4:20, knots = NULL,
    plot = FALSE, ...)
```

Arguments

`x`	a data matrix containing `N` observations (individuals) by rows and `d` variables (features) by columns
`k`	number of clusters

method	clustering algorithm used to cluster the cluster centres from the bootstrapped replicates; Ward, by default. Currently, only pam and randomly initialised kmeans are implemented
nstart	number of random initialisations when using the kmeans method to cluster the cluster centres
B	number of bootstrap replicates to be generated
J	number of observations used to build the bands for the MBD computation. Currently, only the value J=2 can be used
x.coord	initial x coordinates (time points) where the functional data is observed; if not provided, it is assumed to be 1:d
OSF	oversampling factor for the smoothed data; an OSF of m means that the number of (equally spaced) time points observed in the approximated function is m times the number of original number of features, d
vect	optional collection of x coordinates (time points) where to assess the smoothed data; if provided, it ignores the OSF
intercept	if TRUE, an intercept is included in the basis; default is FALSE
degPolyn	degree of the piecewise polynomial; 3 by default (cubic splines)
degFr	a vector containing tentative values of the degrees of freedom, to be tested
knots	the internal breakpoints that define the spline
plot	a Boolean parameter; it allows plotting the distortion against the degrees of freedom. Set to FALSE by default
...	additional arguments to be passed to the kmeans function for the final clustering; at this stage nstart is set to 1, as the initial seeds are fixed

Details

The function implements a simple elbow-like rule that allows selecting an appropriate value for the DF parameter among the tested ones. It computes the distortion obtained for each of these values and returns the one yielding to the smallest distortion. By setting the parameter plot to TRUE the distortion is plotted against the degrees of freedom and elbows or minima can be visually detected.

Value

df	the original vector of DF values to be tested
tot.withinss	a vector containing the distortion obtained for each tested DF value
optimal	DF value producing the smallest distortion among the tested df

Author(s)

Javier Albert Smet <javas@kth.se> and Aurora Torrente <etorrent@est-econ.uc3m.es>

References

Torrente, A. and Romo, J. (2020). Initializing Kmeans Clustering by Bootstrap and Data Depth. *J Classif* (2020). https://doi.org/10.1007/s00357-020-09372-3. Albert-Smet, J., Torrente, A. and Romo J. (2021). Modified Band Depth Based Initialization of Kmeans for Functional Data Clustering. Submitted to Computational Statistics and Data Analysis.

Examples

```
## simulated data
set.seed(1)
x.coord = seq(0,1,0.01)
x <- matrix(ncol = length(x.coord), nrow = 80)
labels <- matrix(ncol = 100, nrow = 1)

centers <-  matrix(ncol = length(x.coord), nrow = 4)
centers[1, ] <- abs(x.coord)-0.5
centers[2, ] <- (abs(x.coord-0.5))^2 - 0.8
centers[3, ] <- -(abs(x.coord-0.5))^2 + 0.7
centers[4, ] <- 0.75*sin(8*pi*abs(x.coord))

for(i in 1:4){
    for(j in 1:20){
        labels[20*(i-1) + j] <- i
        if(i == 1){x[20*(i-1) + j, ] <- abs(x.coord)-0.5 +
            rnorm(length(x.coord),0,1.5)}
        if(i == 2){x[20*(i-1) + j, ] <- (abs(x.coord-0.5))^2 - 0.8 +
            rnorm(length(x.coord),0,1.5)}
        if(i == 3){x[20*(i-1) + j, ] <- -(abs(x.coord-0.5))^2 + 0.7 +
            rnorm(length(x.coord),0,1.5)}
        if(i == 4){x[20*(i-1) + j, ] <- 0.75*sin(8*pi*abs(x.coord)) +
            rnorm(length(x.coord),0,1.5)}
        }
    }

# ER <- elbowRule(x, 4, B=25, degFr = 5:12, plot=FALSE)
ER <- elbowRule(x, 4, B=25, degFr = 5:12, plot=TRUE)
```

`fabrik` *Computation of Initial Seeds for Kmeans and Clustering of Functional Data*

Description

`fabrik` fits splines to the multivariate dataset and runs the BRIk algorithm on the smoothed data. For functional data, this is just a straight forward application of BRIk to the k-means algorithm; for multivariate data, the result corresponds to an alternative clustering method where the objective function is not necessarily minimised, but better allocations are obtained in general.

Usage

```
fabrik(x, k, method="Ward", nstart=1, B = 10, J = 2, x.coord = NULL, OSF = 1,
    vect = NULL, intercept = TRUE, degPolyn = 3, degFr = 5, knots = NULL, ...)
```

Arguments

x	a data matrix containing N observations (individuals) by rows and d variables (features) by columns
k	number of clusters
method	clustering algorithm used to cluster the cluster centres from the bootstrapped replicates; Ward, by default. Currently, only pam and randomly initialised kmeans are implemented
nstart	number of random initialisations when using the kmeans method to cluster the cluster centres
B	number of bootstrap replicates to be generated
J	number of observations used to build the bands for the MBD computation. Currently, only the value J=2 can be used
x.coord	initial x coordinates (time points) where the functional data is observed; if not provided, it is assumed to be 1:d
OSF	oversampling factor for the smoothed data; an OSF of m means that the number of (equally spaced) time points observed in the approximated function is m times the number of original number of features, d
vect	optional collection of x coordinates (time points) where to assess the smoothed data; if provided, it ignores the OSF
intercept	if TRUE, an intercept is included in the basis; default is FALSE
degPolyn	degree of the piecewise polynomial; 3 by default (cubic splines)
degFr	degrees of freedom, as in the bs function
knots	the internal breakpoints that define the spline
...	additional arguments to be passed to the kmeans function for the final clustering; at this stage nstart is set to 1, as the initial seeds are fixed

Details

The FABRIk algorithm extends the BRIk algorithm to the case of longitudinal functional data by adding a step that includes B-splines fitting and evaluation of the curve at specific x coordinates. Thus, it allows handling issues such as noisy or missing data. It identifies smoothed initial seeds that are used as starting points of kmeans on the smoothed data. The resulting clustering does not optimise the distortion (sum of squared distances of each data point to its nearest centre) in the original data space but it provides in general a better allocation of datapoints to real groups.

Value

seeds	a matrix of size k x D, where D is either m x d or the length of vect . It contains the initial smoothed seeds obtained with the BRIk algorithm
km	an object of class kmeans corresponding to the run of kmeans on the smoothed data, with starting points seeds

Author(s)

Javier Albert Smet <javas@kth.se> and Aurora Torrente <etorrent@est-econ.uc3m.es>

References

Torrente, A. and Romo, J. (2020). Initializing Kmeans Clustering by Bootstrap and Data Depth. *J Classif* (2020). https://doi.org/10.1007/s00357-020-09372-3. Albert-Smet, J., Torrente, A. and Romo J. (2021). Modified Band Depth Based Initialization of Kmeans for Functional Data Clustering. Submitted to Computational Statistics and Data Analysis.

Examples

```
## fabrik algorithm
    ## simulated data
    set.seed(1)
    x.coord = seq(0,1,0.01)
    x <- matrix(ncol = length(x.coord), nrow = 100)
    labels <- matrix(ncol = 100, nrow = 1)

    centers <-  matrix(ncol = length(x.coord), nrow = 4)
    centers[1, ] <- abs(x.coord)-0.5
    centers[2, ] <- (abs(x.coord-0.5))^2 - 0.8
    centers[3, ] <- -(abs(x.coord-0.5))^2 + 0.7
    centers[4, ] <- 0.75*sin(8*pi*abs(x.coord))

    for(i in 1:4){
        for(j in 1:25){
            labels[25*(i-1) + j] <- i
            if(i == 1){x[25*(i-1) + j, ] <- abs(x.coord)-0.5 +
                rnorm(length(x.coord),0,1.5)}
            if(i == 2){x[25*(i-1) + j, ] <- (abs(x.coord-0.5))^2 - 0.8 +
                rnorm(length(x.coord),0,1.5)}
            if(i == 3){x[25*(i-1) + j, ] <- -(abs(x.coord-0.5))^2 + 0.7 +
                rnorm(length(x.coord),0,1.5)}
            if(i == 4){x[25*(i-1) + j, ] <- 0.75*sin(8*pi*abs(x.coord)) +
                rnorm(length(x.coord),0,1.5)}
            }
        }

    C1 <- kmeans(x,4)
    C2 <- fabrik(x,4,B=25)

    table(C1$cluster, labels)
    table(C2$km$cluster, labels)
```

`fdebrik` *Computation of Initial Seeds for Kmeans with a Functional Extension of Brik*

Description

`fdebrik` first fits splines to the multivariate dataset; then it identifies functional centers that form tighter groups, by means of the kma algorithm; finally, it converts these into a multivariate data set

in a selected dimension, clusters them and finds the deepest point of each cluster to be used as initial seeds. The multivariate objective function is not necessarily minimised, but better allocations are obtained in general.

Usage

```
fdebrik(x, k, method="Ward", nstart=1, B = 10, J = 2, x.coord = NULL,
    functionalDist="d0.pearson", OSF = 1, vect = NULL, intercept = TRUE,
    degPolyn = 3, degFr = 5, knots = NULL, ...)
```

Arguments

x	a data matrix containing N observations (individuals) by rows and d variables (features) by columns
k	number of clusters
method	clustering algorithm used to cluster the cluster centres from the bootstrapped replicates; Ward, by default. Currently, only pam and randomly initialised kmeans with nstart initializations are implemented
nstart	number of random initialisations when using the kmeans method to cluster the cluster centres
B	number of bootstrap replicates to be generated
J	number of observations used to build the bands for the MBD computation. Currently, only the value J=2 can be used
x.coord	initial x coordinates (time points) where the functional data is observed; if not provided, it is assumed to be 1:d
functionalDist	similarity measure between functions to be used. Currently, only the cosine of the angles between functions ("d0.pearson") and between their derivatives ("d1.pearson") can be used
OSF	oversampling factor for the smoothed data; an OSF of m means that the number of (equally spaced) time points observed in the approximated function is m times the number of original number of features, d
vect	optional collection of x coordinates (time points) where to assess the smoothed data; if provided, it ignores the OSF
intercept	if TRUE, an intercept is included in the basis; default is FALSE
degPolyn	degree of the piecewise polynomial; 3 by default (cubic splines)
degFr	degrees of freedom, as in the bs function
knots	the internal breakpoints that define the spline
...	additional arguments to be passed to the kmeans function for the final clustering; at this stage nstart is set to 1, as the initial seeds are fixed

Details

The FDEBRIk algorithm extends the BRIk algorithm to the case of longitudinal functional data by adding a B-spline fitting step, a collection of functional centers by means of the kma algorithm and the evaluation of these at specific x coordinates. Thus, it allows handling issues such as noisy or

missing data. It identifies smoothed initial seeds that are used as starting points of kmeans on the smoothed data. The resulting clustering does not optimise the distortion (sum of squared distances of each data point to its nearest centre) in the original data space but it provides in general a better allocation of datapoints to real groups.

Value

`seeds`	a matrix of size `k` x `D`, where `D` is either `m` x `d` or the length of `vect` . It contains the initial smoothed seeds obtained with the FDEBRIk algorithm
`km`	an object of class `kmeans` corresponding to the run of kmeans on the smoothed data, with starting points `seeds`

Author(s)

Javier Albert Smet <javas@kth.se> and Aurora Torrente <etorrent@est-econ.uc3m.es>

References

Torrente, A. and Romo, J. Initializing Kmeans Clustering by Bootstrap and Data Depth. *J Classif* (2021) 38(2):232-256. DOI: 10.1007/s00357-020-09372-3 Albert-Smet, J., Torrente, A. and Romo, J. Modified Band Depth Based Initialization of Kmeans for Functional Data Clustering. Submitted to *Adv. Data Anal. Classif.* (2022). Sangalli, L.M., Secchi, P., Vantini, V.S. and Vitelli, V. K-mean alignment for curve clustering. *Comput. Stat. Data Anal.* (2010) 54(5):1219-1233. DOI:10.1016/j.csda.2009.12.008

Examples

```
## fdebrik algorithm
    ## Not run:
    ## simulated data
    set.seed(1)
    x.coord = seq(0,1,0.05)
    x <- matrix(ncol = length(x.coord), nrow = 40)
    labels <- matrix(ncol = 100, nrow = 1)

    centers <-  matrix(ncol = length(x.coord), nrow = 4)
    centers[1, ] <- abs(x.coord)-0.5
    centers[2, ] <- (abs(x.coord-0.5))^2 - 0.8
    centers[3, ] <- -(abs(x.coord-0.5))^2 + 0.7
    centers[4, ] <- 0.75*sin(8*pi*abs(x.coord))

    for(i in 1:4){
        for(j in 1:10){
            labels[10*(i-1) + j] <- i
            if(i == 1){x[10*(i-1) + j, ] <- abs(x.coord)-0.5 +
                rnorm(length(x.coord),0,1.5)}
            if(i == 2){x[10*(i-1) + j, ] <- (abs(x.coord-0.5))^2 - 0.8 +
                rnorm(length(x.coord),0,1.5)}
            if(i == 3){x[10*(i-1) + j, ] <- -(abs(x.coord-0.5))^2 + 0.7 +
                rnorm(length(x.coord),0,1.5)}
            if(i == 4){x[10*(i-1) + j, ] <- 0.75*sin(8*pi*abs(x.coord)) +
```

```
                rnorm(length(x.coord),0,1.5)}
            }
        }

    C1 <- kmeans(x,4)
    C2 <- fdebrik(x,4,B=5)

    table(C1$cluster, labels)
    table(C2$km$cluster, labels)

## End(Not run)
```

---

kma *Clustering and alignment of functional data*

Description

kma jointly performs clustering and alignment of a functional dataset (multidimensional or unidimensional functions).

Usage

```
kma(x, y0 = NULL, y1 = NULL, n.clust = 1, warping.method = "affine",
similarity.method = "d1.pearson", center.method = "k-means", seeds = NULL,
optim.method = "L-BFGS-B", span = 0.15, t.max = 0.1, m.max = 0.1, n.out = NULL,
tol = 0.01, fence = TRUE, iter.max = 100, show.iter = 0, nstart=2, return.all=FALSE,
check.total.similarity=FALSE)
```

Arguments

x	matrix *n.func* X *grid.size* or vector *grid.size*: the abscissa values where each function is evaluated. *n.func*: number of functions in the dataset. *grid.size*: maximal number of abscissa values where each function is evaluated. The abscissa points may be unevenly spaced and they may differ from function to function. x can also be a vector of length *grid.size*. In this case, x will be used as abscissa grid for all functions.
y0	matrix *n.func* X *grid.size* or array *n.func* X *grid.size* X *d*: evaluations of the set of original functions on the abscissa grid x. *n.func*: number of functions in the dataset. *grid.size*: maximal number of abscissa values where each function is evaluated. *d*: (only if the sample is multidimensional) number of function components, i.e. each function is a *d*-dimensional curve. Default value of y0 is NULL. The parameter y0 must be provided if the chosen similarity.method concerns original functions.
y1	matrix *n.func* X *grid.size* or array *n.func* X *grid.size* X *d*: evaluations of the set of original functions first derivatives on the abscissa grid x. Default value of y1 is NULL. The parameter y1 must be provided if the chosen similarity.method concerns original function first derivatives.

`n.clust` scalar: required number of clusters. Default value is `1`. Note that if `n.clust=1` kma performs only alignment without clustering.

`warping.method` character: type of alignment required. If `warping.method='NOalignment'` kma performs only k-mean clustering (without alignment). If `warping.method='affine'` kma performs alignment (and possibly clustering) of functions using linear affine transformation as warping functions, i.e., *x.final* = *dilation*x* + *shift*. If `warping.method='shift'` kma allows only shift, i.e., *x.final* = *x* + *shift*. If `warping.method='dilation'` kma allows only dilation, i.e., *x.final* = *dilation*x*. Default value is `'affine'`.

`similarity.method` character: required similarity measure. Possible choices are: `'d0.pearson'`, `'d1.pearson'`, `'d0.L2'`, `'d1.L2'`, `'d0.L2.centered'`, `'d1.L2.centered'`. Default value is `'d1.pearson'`. See kma.similarity for details.

`center.method` character: type of clustering method to be used. Possible choices are: `'k-means'` and `'k-medoids'`. Default value is `'k-means'`.

`seeds` vector *max(n.clust)* or matrix *nstart* X *n.clust*: indexes of the functions to be used as initial centers. If it is a matrix, each row contains the indexes of the initial centers of one of the `nstart` initializations. In the case where not all the values of `seeds` are provided, those not provided are randomly chosen among the `n.func` original functions. If `seeds=NULL` all the centers are randomly chosen. Default value of `seeds` is `NULL`

.

`optim.method` character: optimization method chosen to find the best warping functions at each iteration. Possible choices are: `'L-BFGS-B'` and `'SANN'`. See optim function for details. Default method is `'L-BFGS-B'`.

`span` scalar: the span to be used for the loess procedure in the center estimation step when `center.method='k-means'`. Default value is 0.15. If `center.method='k-medoids'` value of `span` is ignored.

`t.max` scalar: `t.max` controls the maximal allowed shift, at each iteration, in the alignment procedure with respect to the range of curve domains. `t.max` must be such that *0<t.max<1* (e.g., `t.max=0.1` means that shift is bounded, at each iteration, between *-0.1*range(x)* and *+0.1*range(x)*). Default value is `0.1`. If `warping.method='dilation'` value of `t.max` is ignored.

`m.max` scalar: `m.max` controls the maximal allowed dilation, at each iteration, in the alignment procedure. `m.max` must be such that *0<m.max<1* (e.g., `m.max=0.1` means that dilation is bounded, at each iteration, between *1-0.1* and *1+0.1*). Default value is `0.1`. If `warping.method='shift'` value of `m.max` is ignored.

`n.out` scalar: the desired length of the abscissa for computation of the similarity indexes and the centers. Default value is `round(1.1*grid.size)`.

`tol` scalar: the algorithm stops when the increment of similarity of each function with respect to the corrispondent center is lower than `tol`. Default value is `0.01`.

`fence` boolean: if `fence=TRUE` a control is activated at the end of each iteration. The aim of the control is to avoid shift/dilation outlighers with respect to their computed distributions. If `fence=TRUE` the running time can increase considerably. Default value of `fence` is TRUE.

`iter.max` scalar: maximum number of iterations in the k-mean alignment cycle. Default value is `100`.

`show.iter` boolean: if `show.iter=TRUE` kma shows the current iteration of the algorithm. Default value is `FALSE`.

`nstart` scalar: number of initializations with different seeds. Default value is `2`. This parameter is used only if `center.method` is `'k-medoids'`. When `center.method = 'k-means'` one initialization is performed.

`return.all` boolean: if `return.all=TRUE` the results of all the `nstart` initializations are returned; the output is a list of length `nstart`. If `return.all=FALSE` only the best result is provided (the one with higher mean similarity if `similarity.method` is `'d0.pearson'` or`'d1.pearson'`, or the one with lower distance if `similarity.method` is `'d0.L2'`, `'d1.L2'`, `'d0.L2.centered'` or `'d1.L2.centered'`). Default value is `FALSE`.

`check.total.similarity` boolean: if `check.total.similarity=TRUE` at each iteration the algorithm checks if there is a decrease of the total similarity and stops. In the affermative case the result obtained in the penultimate iteration is returned. Default value is `FALSE`

Value

The function output is a list containing the following elements:

`iterations` scalar: total number of iterations performed by kma function.

`x` as input.

`y0` as input.

`y1` as input.

`n.clust` as input.

`warping.method` as input.

`similarity.method` as input.

`center.method` as input.

`x.center.orig` vector *n.out*: abscissa of the original center.

`y0.center.orig` matrix *1* X *n.out*: the unique row contains the evaluations of the original function center. If `warping.method='k-means'` there are two scenarios: if `similarity.method='d0.pearson'` or `'d0.L2'` or `d0.L2.centered` the original function center is computed via loess procedure applied to original data; if `similarity.method='d1.pearson'` or `'d1.L2'` or `d1.L2.centered` it is computed by integration of first derivatives center `y1.center.orig` (the integration constant is computed minimizing the sum of the weighed L2 distances between the center and the original functions). If `warping.method='k-medoids'` the original function center is the medoid of original functions.

`y1.center.orig` matrix *1* X *n.out*: the unique row contains the evaluations of the original function first derivatives center. If `warping.method='k-means'` the original center is computed via loess procedure applied to original function first derivatives. If `warping.method='k-medoids'` the original center is the medoid of original functions.

`similarity.orig`
: vector: original similarities between the original functions and the original center.

`x.final`
: matrix *n.func* X *grid.size*: aligned abscissas.

`n.clust.final`
: scalar: final number of clusters. Note that, when `center.method='k.means'`, the parameter `n.clust.final` may differ from initial number of clusters (i.e., from `n.clust`) if some clusters are found to be empty. In this case a warning message is issued.

`x.centers.final`
: vector *n.out*: abscissas of the final function centers and/or of the final function first derivatives centers.

`y0.centers.final`
: matrix *n.clust.final* X *n.out*: rows contain the evaluations of the final functions centers. `y0.centers.final` is `NULL` if `y0` is not given as input.

`y1.centers.final`
: matrix *n.clust.final* X *n.out*: rows contains the evaluations of the final derivatives centers. `y1.centers.final` is `NULL` if the chosen similarity measure does not concern function first derivatives.

`labels`
: vector: cluster assignments.

`similarity.final`
: vector: similarities between each function and the center of the cluster the function is assigned to.

`dilation.list`
: list: dilations obtained at each iteration of kma function.

`shift.list`
: list: shifts obtained at each iteration of kma function.

`dilation`
: vector: dilation applied to the original abscissas `x` to obtain the aligned abscissas `x.final`.

`shift`
: vector: shift applied to the original abscissas `x` to obtain the aligned abscissas `x.final`.

Author(s)

Alice Parodi, Mirco Patriarca, Laura Sangalli, Piercesare Secchi, Simone Vantini, Valeria Vitelli.

References

Sangalli, L.M., Secchi, P., Vantini, S., Vitelli, V., 2010. *"K-mean alignment for curve clustering"*. Computational Statistics and Data Analysis, 54, 1219-1233.

Sangalli, L.M., Secchi, P., Vantini, S., 2014. *"Analysis of AneuRisk65 data: K-mean Alignment"*. Electronic Journal of Statistics, Special Section on "Statistics of Time Warpings and Phase Variations", Vol. 8, No. 2, 1891-1904.

See Also

`kma.similarity`

Examples

```
## simulated data
set.seed(1)
x.coord = seq(0,1,0.01)
x <- matrix(ncol = length(x.coord), nrow = 100)
labels <- matrix(ncol = 100, nrow = 1)

centers <-  matrix(ncol = length(x.coord), nrow = 4)
centers[1, ] <- abs(x.coord)-0.5
centers[2, ] <- (abs(x.coord-0.5))^2 - 0.8
centers[3, ] <- -(abs(x.coord-0.5))^2 + 0.7
centers[4, ] <- 0.75*sin(8*pi*abs(x.coord))

for(i in 1:4){
    for(j in 1:25){
        labels[25*(i-1) + j] <- i
        if(i == 1){x[25*(i-1) + j, ] <- abs(x.coord)-0.5 +
            rnorm(length(x.coord),0,0.1)}
        if(i == 2){x[25*(i-1) + j, ] <- (abs(x.coord-0.5))^2 - 0.8 +
            rnorm(length(x.coord),0,0.1)}
        if(i == 3){x[25*(i-1) + j, ] <- -(abs(x.coord-0.5))^2 + 0.7 +
            rnorm(length(x.coord),0,0.1)}
        if(i == 4){x[25*(i-1) + j, ] <- 0.75*sin(8*pi*abs(x.coord)) +
            rnorm(length(x.coord),0,0.1)}
        }
    }
C <- kma(x.coord, x, n.clust = 4,
        warping.method = "NOalignment", similarity.method = "d0.pearson")
table(C$labels, labels)
```

`kma.similarity` *Similarity/dissimilarity index between two functions*

Description

kma.similarity computes a similarity/dissimilarity measure between two functions f and g. Users can choose among different types of measures.

Usage

```
kma.similarity(x.f = NULL, y0.f = NULL, y1.f = NULL,
x.g = NULL, y0.g = NULL, y1.g = NULL, similarity.method, unif.grid = TRUE)
```

Arguments

`x.f`	vector *length.f*: abscissa grid where function f and his first derivatives f' is evaluated. *length.f*: numbrt of abscissa values where f is evaluated. `x.f` must always be provided.

`y0.f` vector *length.f* or matrix *length.f* X d: evaluations of function f on the abscissa grid `x.f`. *length.f*: number of abscissa values where f is evaluated. d (only if f and g are multidimensional) number of function's components, i.e. f is d-dimensional curve. Default value of `y0.f` is `NULL`. The vectory`0.f` must be provided if the chosen `similarity.method` concerns original functions.

`y1.f` vector *length.f* or matrix *length.f* X d: evaluations of f first derivative, i.e., f', on the abscissa grid `x.f`. Default value of `y1.f` is `NULL`. The vector `y1.f` must be provided if the chosen `similarity.method` concerns function first derivatives.

`x.g` vector *length.g*: abscissa grid where function g and his first derivatives g' is evaluated. *length.g*: numbrt of abscissa values where g is evaluated. `x.g` must always be provided.

`y0.g` vector *length.g* or matrix *length.g* X d: evaluations of function g on the abscissa grid `x.g`. *length.g*: number of abscissa values where g is evaluated. d (only if f and g are multidimensional) number of function's components, i.e. g is d-dimensional curve. Default value of `y0.g` is `NULL`. The vector `y0.g` must be provided if the chosen `similarity.method` concerns original functions.

`y1.g` vector *length.g* or matrix *length.g* X d: evaluations of g first derivative, i.e., g', on the abscissa grid `x.g`. Default value is of `y1.g` `NULL`. The vector `y1.g` must be provided if the chosen `similarity.method` concerns function first derivatives.

`similarity.method`
character: similarity/dissimilarity between f and g. Possible choices are: `'d0.pearson'`, `'d1.pearson'`, `'d0.L2'`, `'d1.L2'`, `'d0.L2.centered'`, `'d1.L2.centered'`. Default value is `'d1.pearson'`. See details.

`unif.grid` boolean: if equal to `TRUE` the similarity measure is computed over an uniform grid built in the intersection domain of the two functions, that is an additional discretization is performed. If equal to `FALSE` the additional discretization is not performed, so the functions are supposed to be already defined on the same abscissa grid and the grid is supposed to be fine enough to well compute similarity.

Details

We report the list of the currently available similarities/dissimilarities. Note that all norms and inner products are computed over D, that is the intersection of the domains of f and g. $\overline{f}$ and $\overline{g}$ denote the mean value, respectively, of functions f and g.

1. `'d0.pearson'`: this similarity measure is the cosine of the angle between the two functions f and g.

$$\frac{< f, g >_{L^2}}{\|f\|_{L^2}\|g\|_{L^2}}$$

2. `'d1.pearson'`: this similarity measure is the cosine of the angle between the two function derivatives f' and g'.

$$\frac{< f', g' >_{L^2}}{\|f'\|_{L^2}\|g'\|_{L^2}}$$

3. `'d0.L2'`: this dissimilarity measure is the L2 distance of the two functions f and g normalized by the length of the common domain D.

$$\frac{\|f-g\|_{L^2}}{|D|}$$

4. `'d1.L2'`: this dissimilarity measure is the L2 distance of the two function first derivatives f' and g' normalized by the length of the common domain D.

$$\frac{\|f'-g'\|_{L^2}}{|D|}$$

5. `'d0.L2.centered'`: this dissimilarity measure is the L2 distance of $f-\overline{f}$ and $g-\overline{g}$ normalized by the length of the common domain D.

$$\frac{\|(f-\overline{f})-(g-\overline{g})\|_{L^2}}{|D|}$$

6. `'d1.L2.centered'`: this dissimilarity measure is the L2 distance of $f'-\overline{f'}$ and $g'-\overline{g'}$ normalized by the length of the common domain D.

$$\frac{\|(f'-\overline{f'})-(g'-\overline{g'})\|_{L^2}}{|D|}$$

For multidimensional functions, if `similarity.method='d0.pearson'` or `'d1.pearson'` the similarity/dissimilarity measure is computed via the average of the indexes in all directions.

The coherence properties specified in Sangalli et al. (2010) implies that if `similarity.method` is set to `'d0.L2'`, `'d1.L2'`, `'d0.L2.centered'` or `'d1.L2.centered'`, value of `warping.method` must be `'shift'` or `'NOalignment'`. If `similarity.method` is set to `'d0.pearson'` or `'d1.pearson'` all values for `warping.method` are allowed.

Value

scalar: similarity/dissimilarity measure between the two functions f and g computed via the similarity/dissimilarity measure specified.

Author(s)

Alice Parodi, Mirco Patriarca, Laura Sangalli, Piercesare Secchi, Simone Vantini, Valeria Vitelli.

References

Sangalli, L.M., Secchi, P., Vantini, S., Vitelli, V., 2010. *"K-mean alignment for curve clustering"*. Computational Statistics and Data Analysis, 54, 1219-1233.

Sangalli, L.M., Secchi, P., Vantini, S., 2014. *"Analysis of AneuRisk65 data: K-mean Alignment"*. Electronic Journal of Statistics, Special Section on "Statistics of Time Warpings and Phase Variations", Vol. 8, No. 2, 1891-1904.

See Also

kma

plotKmeansClustering *Kmeans Clustering Plot*

Description

plotKmeansClustering represents, in different subpanels, each of the clusters obtained after running k-means. The corresponding centroid is highlighted.

Usage

```
plotKmeansClustering(x, kmeansObj, col=c(8,2), lty=c(2,1), x.coord = NULL,
    no.ticks = 5, ...)
```

Arguments

x	a data matrix containing N observations (individuals) by rows and d variables (features) by columns
kmeansObj	an object of class kmeans, containing the cluster labels output by kmeans
col	a vector containing colors for the elements in x and for the centroid. The last one is used for the centroid, whereas the previous ones are recycled
lty	a vector containing the line type for the elements in x and for the centroid. The last one is used for the centroid, whereas the previous ones are recycled
x.coord	initial x coordinates (time points) where the functional data is observed; if not provided, it is assumed to be 1:d
no.ticks	number of ticks to be displayed in the X axis
...	additional arguments to be passed to the plot function

Details

The function creates a suitable grid where to plot the different clusters independently. In the i-th cell of the grid, the data points corresponding to the i-th cluster are represented in parallel coordinates and the final centroid is highlighted.

Value

the function returns invisibly a list with the following components:

clusters	a list containing one cluster per component; observations are given by rows
centroids	a list with the centroid of each cluster

Author(s)

Javier Albert Smet <javas@kth.se> and Aurora Torrente <etorrent@est-econ.uc3m.es>

Examples

```
## simulated data
set.seed(1)
x.coord = seq(0,1,0.01)
x <- matrix(ncol = length(x.coord), nrow = 100)
labels <- matrix(ncol = 100, nrow = 1)

centers <-  matrix(ncol = length(x.coord), nrow = 4)
centers[1, ] <- abs(x.coord)-0.5
centers[2, ] <- (abs(x.coord-0.5))^2 - 0.8
centers[3, ] <- -(abs(x.coord-0.5))^2 + 0.7
centers[4, ] <- 0.75*sin(8*pi*abs(x.coord))

for(i in 1:4){
    for(j in 1:25){
        labels[25*(i-1) + j] <- i
        if(i == 1){x[25*(i-1) + j, ] <- abs(x.coord)-0.5 +
            rnorm(length(x.coord),0,1.5)}
        if(i == 2){x[25*(i-1) + j, ] <- (abs(x.coord-0.5))^2 - 0.8 +
            rnorm(length(x.coord),0,1.5)}
        if(i == 3){x[25*(i-1) + j, ] <- -(abs(x.coord-0.5))^2 + 0.7 +
            rnorm(length(x.coord),0,1.5)}
        if(i == 4){x[25*(i-1) + j, ] <- 0.75*sin(8*pi*abs(x.coord)) +
            rnorm(length(x.coord),0,1.5)}
        }
    }

plotKmeansClustering(x, kmeans(x,4))
plotKmeansClustering(x, brik(x,4)$km)
plotKmeansClustering(x, fabrik(x,4)$km)
plotKmeansClustering(x, fabrik(x,4,degFr=10)$km)
```

Index