Package: clv (via r-universe)

November 13, 2026

Title Cluster Validation Techniques

Version 0.3.2.5.9001

Description Contains most of the popular internal and external cluster validation methods ready to use for the most of the outputs produced by functions coming from package ``cluster''. Package contains also functions and examples of usage for cluster stability approach that might be applied to algorithms implemented in ``cluster'' package as well as user defined clustering algorithms. Compatibility updates by Renova.

Depends cluster, class

License GPL (>= 2)

NeedsCompilation yes

Author Lukasz Nieweglowski [aut, cre]

Maintainer Lukasz Nieweglowski <wookashn@gmail.com>

Repository https://renova-r.r-universe.dev

Date/Publication 2026-11-13 15:38:31 UTC

RemoteUrl https://github.com/renova-r/clv

RemoteRef renova-0.3.2.5.9001

RemoteSha b0ea9de216dac4c52f3cbc28f0544918de5d6f4f

Contents

cls.attrib 2
cls.scatt.data 3
cls.set.section 5
cls.stab.sim.ind 6
cls.stab.sim.ind.usr 10
clv.Davies.Bouldin 13
clv.Dens_bw 15
clv.Dis 17
clv.Dunn 18
clv.Scatt 20

clv.SD, clv.SDbw 21
confusion.matrix 23
connectivity 24
dot.product 25
similarity.index 28
std.ext 31
wcls/bcls.matrix 33

Index **36**

`cls.attrib` *Mean, cluster size and center - cluster utilities*

Description

Mean, center of each cluster, number of objects in each cluster - informations retrieved from partitioned data using `cls.attrib`.

Usage

```
cls.attrib(data, clust)
```

Arguments

`data`	`numeric matrix` or `data.frame` where columns correspond to variables and rows to observations.
`clust`	integer `vector` with information about cluster id the object is assigned to. If vector is not integer type, it will be coerced with warning.

Value

As a result function returns object of `list` type which contains three objects with information about:
`mean` - `numeric vector` which represents mean of given `data`,
`cluster.center` - `numeric matrix` where columns correspond to variables and rows to observations,
`cluster.size` - `integer vector` with information about size of each cluster.

Author(s)

Lukasz Nieweglowski

See Also

Result of function is mostly used to compute following indicies: `clv.Dis`, `wcls.matrix`, `bcls.matrix`.

Examples

```
# create "data" matrix
mx <- matrix(0,4,2)
mx[2,1] = mx[3,2] = mx[4,1] = mx[4,2] = 1
# give information about cluster assignment
clust = as.integer(c(1,1,2,2))
cls.attrib(mx,clust)
```

`cls.scatt.data` *Intercluster distances and intracluster diameters - Internal Measures*

Description

Two functions which find most popular *intercluster distances* and *intracluster diameters*.

Usage

```
cls.scatt.data(data, clust, dist="euclidean")
cls.scatt.diss.mx(diss.mx, clust)
```

Arguments

`data`	`numeric matrix` or `data.frame` where columns correspond to variables and rows to observations
`diss.mx`	square, symmetric `numeric matrix` or `data.frame`, representation of dissimilarity matrix where infomartion about distances between objects is stored.
`clust`	integer `vector` with information about cluster id the object is assigned to. If vector is not integer type, it will be coerced with warning.
`dist`	chosen metric: "euclidean" (default value), "manhattan", "correlation" (variable enable only in `cls.scatt.data` function).

Details

Six *intercluster distances* and three *intracluster diameters* can be used to calculate such validity indices as *Dunn* and *Davies-Bouldin* like. Let `d(x,y)` be a distance function between two objects comming from our data set.

Intracluster diameters

The *complete diameter* represents the distance between two the most remote objects belonging to the same cluster.

diam1(C) = max{ d(x,y): x,y belongs to cluster C }

The *average diameter* distance defines the average distance between all of the samples belonging to the same cluster.

diam2(C) = 1/|C|(|C|-1) * sum{ forall x,y belongs to cluster C and x != y } d(x,y)

The *centroid diameter* distance reflects the double average distance between all of the samples and the cluster's center (v(C) - cluster center).

diam3(C) = 1/|C| * sum{ forall x belonging to cluster C} d(x,v(C))

Intercluster distances

The *single linkage* distance defines the closest distance between two samples belonging to two different clusters.

dist1(Ci,Cj) = min{ d(x,y): x belongs to Ci and y to Cj cluster }

The *complete linkage* distance represents the distance between the most remote samples belonging to two different clusters.

dist2(Ci,Cj) = max{ d(x,y): x belongs to Ci and y to Cj cluster }

The *average linkage* distance defines the average distance between all of the samples belonging to two different clusters.

dist3(Ci,Cj) = 1/(|Ci|*|Cj|) * sum{ forall x belongs Ci and y to Cj } d(x,y)

The *centroid linkage* distance reflects the distance between the centres of two clusters (v(i), v(j) - clusters' centers).

dist4(Ci,Cj) = d(v(i), V(j))

The *average of centroids linkage* represents the distance between the centre of a cluster and all of samples belonging to a different cluster.

dist5(Ci,Cj) = 1/(|Ci|+|Cj|) * (sum{ forall x belongs Ci } d(x,v(j)) + sum{ forall y belongs Cj } d(y,v(i)))

Hausdorff metrics are based on the discovery of a maximal distance from samples of one cluster to the nearest sample of another cluster.

dist6(Ci,Cj) = max{ distH(Ci,Cj), distH(Cj,Ci) }

where: distH(A,B) = max{ min{ d(x,y): y belongs to B}: x belongs to A }

Value

`cls.scatt.data` returns an object of class `"list"`. Intracluster diameters: `intracls.complete`, `intracls.average`, `intracls.centroid`, are stored in vectors and intercluster distances: `intercls.single`, `intercls.complete`, `intercls.average`, `intercls.centroid`, `intercls.ave_to_cent`, `intercls.hausdorff` in symmetric matrices. Vectors' lengths and both dimensions of each matrix are equal to number of clusters. Additionally in result list `cluster.center` matrix (rows correspond to clusters centers) and `cluster.size` vector is given (information about size of each cluster).

`cls.scatt.diss.mx` returns an object of class `"list"`. Intracluster diameters: `intracls.complete`, `intracls.average`, are stored in vectors and intercluster distances: `intercls.single`, `intercls.complete`, `intercls.average`, `intercls.hausdorff` in symmetric matrices. Vectors' lengths and both dimensions of each matrix are equal to number of clusters. Additionally in result list `cluster.size` vector is given (information about size of each cluster).

Author(s)

Lukasz Nieweglowski

References

J. Handl, J. Knowles and D. B. Kell *Computational cluster validation in post-genomic data analysis*, http://bioinformatics.oxfordjournals.org/cgi/reprint/21/15/3201?ijkey=VbTHU29vqzwkGs2&keytype=ref

N. Bolshakova, F. Azuajeb *Cluster validation techniques for genome expression data*, http://citeseer.ist.psu.edu/552250.html

See Also

Result used in: `clv.Dunn`, `clv.Davies.Bouldin`.

Examples

```
# load and prepare data
library(clv)
data(iris)
iris.data <- iris[,1:4]

# cluster data
pam.mod <- pam(iris.data,5) # create five clusters
v.pred <- as.integer(pam.mod$clustering) # get cluster ids associated to given data objects

# compute intercluster distances and intracluster diameters
cls.scatt1 <- cls.scatt.data(iris.data, v.pred)
cls.scatt2 <- cls.scatt.data(iris.data, v.pred, dist="manhattan")
cls.scatt3 <- cls.scatt.data(iris.data, v.pred, dist="correlation")

# the same using dissimilarity matrix
iris.diss.mx <- as.matrix(daisy(iris.data))
cls.scatt4 <- cls.scatt.diss.mx(iris.diss.mx, v.pred)
```

`cls.set.section` *Section of two subsets - External Measure utilities*

Description

Function finds section of two different subsets comming from the same data set.

Usage

```
cls.set.section(clust1, clust2)
```

Arguments

`clust1`	n x 2 integer `matrix` or `data.frame`. First column gives information about object number in data set in increasing order. Second column store information about cluster id the object is assigned to. If matrix is not an integer type, it will be coerced with warning.

clust2 — n x 2 integer matrix or data.frame. First column gives information about object number in data set in increasing order. Second column store information about cluster id the object is assigned to. If matrix is not an integer type, it will be coerced with warning.

Details

Let A and B be two different subsamples of the same data set. Each subset is partitioned into P(A) and P(B) cluster sets. Information about object and cluster id's for pairs (A,P(A)) and (B,P(B)) are stored in matrices clust1 and clust2. Function creates matrix which represents section of A and B.

Value

cls.set.section returns a n x 3 integer matrix. First column gives information about object number in dataset in increasing order. Second column store information about cluster id the object is assigned to. Information is taken from clust1 vector The same is for the third column but cluster id is taken from vector clust2.

Author(s)

Lukasz Nieweglowski

See Also

Function preapres data for further computation. Result mostly is used in: std.ext, dot.product, confusion.matrix

Examples

```
# create two different subsamples
mx1 <- matrix(as.integer( c(1,2,3,4,5,6,1,1,2,2,3,3) ), 6, 2 )
mx2 <- matrix(as.integer( c(1,2,4,5,6,7,1,1,2,2,3,3) ), 6, 2 )
# find section
m = cls.set.section(mx1,mx2)
```

---

cls.stab.sim.ind *Cluster Stability - Similarity Index and Pattern-wise Stability Approaches*

Description

cls.stab.sim.ind and cls.stab.opt.assign reports validation measures for clustering results. Both functions return lists of cluster stability results computed according to similarity index and pattern-wise stability approaches.

Usage

```
cls.stab.sim.ind( data, cl.num, rep.num, subset.ratio, clust.method,
                  method.type, sim.ind.type, fast, ... )
cls.stab.opt.assign( data, cl.num, rep.num, subset.ratio, clust.method,
                     method.type, fast, ... )
```

Arguments

`data`	`numeric matrix` or `data.frame` where columns correspond to variables and rows to observations.
`cl.num`	integer `vector` with information about numbers of cluster to which `data` will be partitioned. If vector is not an integer type, it will be coerced with warning.
`rep.num`	integer number which tells how many pairs of data subsets will be partitioned for particular number of clusters. The results of partitioning for given pair of subsets is used to compute similarity indices (in case of `cls.stab.sim.ind`) or *pattern-wise stability* (in case of `cls.stab.opt.assign`, for more details see references). By default `rep.num` value is 10. If wrong argument is applied it will be repaced with default value.
`subset.ratio`	a number comming from (0,1) section which tells how big data subsets should be. 0 means empty subset, 1 means all data. By default `subset.ratio` is set to 0.75. If wrong argument is applied it will be repaced with default value.
`clust.method`	string vector with names of cluster algorithms to be used. Available are: "agnes", "diana", "hclust", "kmeans", "pam", "clara". Combinations are also possible. By default `c("agnes","pam")` vector is applied.
`method.type`	string vector with information useful only in context of "agnes" and "hclust" algorithms . Available are: "single", "average", "complete", "ward" and "weighted" (for more details see `agnes`, `hclust`). The last type is applicable only for "agnes". Combinations are also possible. By default `c("single","average")` vector is applied.
`sim.ind.type`	string vector with information useful only for `cls.stab.sim.ind` function. User is able to choose which similarity indices (external measures) to use to compare two partitionings. Available are: "dot.pr", "sim.ind", "rand", "jaccard" (for more details see `similarity.index`, `dot.product`, `std.ext`). Combinations are also possible. By default `c("dot.pr","sim.ind")` vector is applied.
`fast`	logical argument which sets the way of computing cluster stability for hierarchical algorithms. By default it is set to TRUE, which means that each result produced by hierarchical algorithm is partitioned for the number of clusters chosen in `cl.num` argument and given clustering results are put for further computation. In this way computation of cluster stability is faster. If wrong argument is applied it will be repaced with default value.
`...`	additional parameters for clustering algorithms. Note: use with caution! Different clustering methods chosen in `clust.method` have different set of parameter names - mixing them often disallow any cluster algorithm to run.

Details

Both functions realize cluster stability approaches described in *Detecting stable clusters using principal component analysis* (see references).

The `cls.stab.sim.ind` function realizes algorithm given in *chapter 3.1* where only cosine similarity index (see `dot.product`) is introduced as a similarity index between two different partitionings. This function realize this cluster stability approach also for other similarity indices such us `similarity.index`, `clv.Rand` and `clv.Jaccard`. The important thing is that `similarity index` (if chosen) produced by this function is not exactly the same as index produced by `similarity.index` function. The value of the `similarity.index` is a number which depends on number of clusters. Eg. if two "n-clusters" partitionings are compared the value always will be a number which belong to the `[1/n, 1]` section. That means the results produced by this similarity index are not comparable for different number of clusters. That's why each result is scaled thanks to the linear function `f:[1/n, 1] -> [0, 1]` where "n" is a number of clusters. The results' layout is described in *Value* section.

The `cls.stab.opt.assign` function realizes algorithm given in *chapter 3.2* where *pattern-wise agreement* and *pattern-wise stability* was introduced. Function returns the lowest *pattern-wise stability* value for given number of clusters. The results' layout is described in *Value* section.

It often happens that clustering algorithms can't produce amount of clusters that user wants. In this situation only the warning is produced and cluster stability is computed for partitionings with unequal number of clusters.

The cluster stability will not be calculated for all cluster numbers that are bigger than the subset size. For example if `data` contains about 20 objects and the `subset.ratio` equals 0.5 then the highest cluster number to calculate is 10. In that case all elements above 10 will be removed from `cl.num` vector.

Value

`cls.stab.sim.ind` returns a list of lists of matrices. Each matrix consists of the set of external similarity indices (which one similarity index see below) where number of columns is equal to `cl.num` vector length and row number is equal to `rep.num` value what means that each column contain a set of similarity indices computed for fixed number of clusters. The order of the matricides depends on three input arguments: `clust.method`, `method.type`, and `sim.ind.type`. Combination of `clust.method` and `method.type` give a names for elements listed in the first list. Each element of this list is also a list type where each element name correspond to one of similarity index type chosen thanks to `sim.ind.type` argument. The order of the names exactly match to the order given in those arguments description. It is easy to understand after considering the following example.
Let say we are running `cls.stab.sim.ind` with default arguments then the results will be given in the following order: `$agnes.single$dot.pr`, `$agnes.single$sim.ind`, `$agnes.average$dot.pr`, `$agnes.average$sim.ind`, `$pam$dot.pr`, `$pam$sim.ind`.

`cls.stab.opt.assign` returns a list of vectors. Each vector consists of the set of cluster stability indices described in *Detecting stable clusters using principal component analysis* (see references). Vector length is equal to `cl.num` vector length what means that each position in vector is assigned to proper clusters' number given in `cl.num` argument. The order of the vectors depends on two input arguments: `clust.method`, `method.type`. The order of the names exactly match to the order given in arguments description. It is easy to understand after considering the following example.

Let say we are running `cls.stab.opt.assign` with `c("pam", "kmeans", "hclust", "agnes")` as `clust.method` and `c("ward","average")` as `method.type` then the results will be given in the following order: `$hclust.average`, `$hclust.ward`, `$agnes.average`, `$agnes.ward`, `$kmeans`, `$pam`.

Author(s)

Lukasz Nieweglowski

References

A. Ben-Hur and I. Guyon *Detecting stable clusters using principal component analysis*, http://citeseerx.ist.psu.edu/

C. D. Giurcaneanu, I. Tabus, I. Shmulevich, W. Zhang *Stability-Based Cluster Analysis Applied To Microarray Data*, http://citeseerx.ist.psu.edu/.

T. Lange, V. Roth, M. L. Braun and J. M. Buhmann *Stability-Based Validation of Clustering Solutions*, doi:10.1162/089976604773717621

See Also

Advanced cluster stability functions: `cls.stab.sim.ind.usr`, `cls.stab.opt.assign.usr`.

Functions that compare two different partitionings: `clv.Rand`, `dot.product`, `similarity.index`.

Examples

```
# load and prepare data
library(clv)
data(iris)
iris.data <- iris[,1:4]

# fix arguments for cls.stab.* function
iter = c(2,3,4,5,6,7,9,12,15)
smp.num = 5
ratio = 0.8

res1 = cls.stab.sim.ind( iris.data, iter, rep.num=smp.num, subset.ratio=0.7,
                         sim.ind.type=c("rand","dot.pr","sim.ind") )
res2 = cls.stab.opt.assign( iris.data, iter, clust.method=c("hclust","kmeans"),
                            method.type=c("single","average") )

print(res1)
boxplot(res1$agnes.average$sim.ind)
plot(res2$hclust.single)
```

`cls.stab.sim.ind.usr` *Cluster Stability - Similarity Index and Pattern-wise Stability Approaches with User Defined Cluster Algorithms*

Description

`cls.stab.sim.ind.usr` and `cls.stab.opt.assign.usr` reports validation measures for clustering results. Both functions return lists of cluster stability results computed for user defined cluster algorithms according to similarity index and pattern-wise stability approaches.

Usage

```
cls.stab.sim.ind.usr( data, cl.num, clust.alg, sim.ind.type, rep.num, subset.ratio )
cls.stab.opt.assign.usr( data, cl.num, clust.alg, rep.num, subset.ratio )
cls.alg( clust.method, clust.wrap, fast )
```

Arguments

`data`	`numeric matrix` or `data.frame` where columns correspond to variables and rows to observations.
`cl.num`	integer `vector` with information about numbers of cluster to which `data` will be partitioned. If vector is not an integer type, it will be coerced with warning.
`clust.alg`	there are two possible types of input: 1. clustering function that takes two arguments: "data" to be partitioned described in `data` section and "clust.num" that represents number of cluster to which data will be partitioned. Function represents partitioning algorithm. 2. an object of type "cls.alg" returned by `cls.alg` function (see "Details" for explanation). Object represents hierarchical algorithm.
`clust.method`	hierarchical clustering function that takes only one argument named "data" described in `data` section. Function should return hierarchical structure that might be applied as parameter to `clust.wrap` function.
`clust.wrap`	cluster function that takes exactly two arguments: "clust.res" that represents the result of `clust.method` function and "clust.num" which is the number of clusters to which "clust.res" is going to be cut. Function should return integer vector that represents object id (comming from `data` set) to cluster id (integer between 1 and `clust.num`) association.
`sim.ind.type`	string vector with information useful only for `cls.stab.sim.ind.usr` function. User is able to choose which similarity indicies (external measures) to use to compare two partitionings. Available are: "dot.pr", "sim.ind", "rand", "jaccard" (for more details see `similarity.index`, `dot.product`, `std.ext`). Combinations are also possible. By default `c("dot.pr","sim.ind")` vector is applied.
`rep.num`	integer number which tells how many pairs of data subsets will be partitioned for particular number of clusters. The results of partitioning for given pair of subsets is used to compute similarity indices (in case of `cls.stab.sim.ind.usr`) or

pattern-wise stability (in case of `cls.stab.opt.assign.usr`, for more details see references). By default `rep.num` value is 10. If wrong argument is applied it will be repaced with default value.

`subset.ratio` a number comming from (0,1) section which tells how big data subsets should be. 0 means empty subset, 1 means all data. By default `subset.ratio` is set to 0.75. If wrong argument is applied it will be repaced with default value.

`fast` logical argument which sets the way of computing cluster stability for hierarchical algorithms. By default it is set to TRUE, which means that each result produced by hierarchical algorithm is partitioned for the number of clusters chosen in `cl.num` argument and given clustering results are put for further computation. In this way computation of cluster stability is faster. If wrong argument is applied it will be repaced with default value.

Details

Both functions realize cluster stability approaches described in *Detecting stable clusters using principal component analysis* chapters *3.1* and *3.2* (see references).

The `cls.stab.sim.ind.usr` as well as `cls.stab.opt.assign.usr` do the same thing as `cls.stab.sim.ind` and `cls.stab.opt.assign` functions. Main difference is that using this functions user is able to define and apply its own cluster algorithm to measure its cluster stability. For that reason `clust.alg` argument is introduced. This argument may represent partitioning algorithm (by passing it directly as a function) or hierarchical algorithm (by passing an object of "cls.alg" type produced by `cls.alg` function).

If a partitioning algorithm is going to be used the decalration of this function that represents this algorithm should always look like this: `function(data, clust.num) { ... return(integer.vector)}` . As an output function should always return integer vector that represents single clustering result on `data`.

If a hierarchical algorithm is going to be used user has to use helper `cls.alg` function that produces an object of "cls.alg" type. This object encapsulates a pair of methods that are used in hierarchical version (which is faster if the `fast` argument is not FALSE) of cluster stability approach. These methods are:
1. *clust.method* - which builds hierarchical structure that might be cut. The declaration of this function should always look like this one: `function(data) { ... return(hierarchical.struct) }` ,
2. *clust.wrap* - which cuts this hierarchical structure to `clust.num` clusters. This function definition should always look like this one: `function(clust.res, clust.num) { ... return(integer.vector)}` . As an output function should always return integer vector that represents single clustering result on `clust.res`.

`cls.alg` function has also third argument that indicates if fast computation should be taken (when `TRUE`) or if these two methods should be converted to one partitioning algorithm and to be run as a normal partitioning algorithm.

Well defined cluster functions "f" should always follow this rules (size(data) means number of object to be partitioned, res - integer vector with cluster ids):
1. when `data` is empty or `cl.num` is less than 2 or more than `size(data)` then `f(data, cl.num)` returns error. 2. if `f(data, cl.num) -> res` then length(res) == size(data),
3. if `f(data, cl.num) -> res` then for all "elem" in "res" the folowing condition is true: `0 < elem <= cl.num`.

It often happens that clustering algorithms can't produce amount of clusters that user wants. In this situation only the warning is produced and cluster stability is computed for partitionings with unequal number of clusters.

The cluster stability will not be calculated for all cluster numbers that are bigger than the subset size. For example if `data` contains about 20 objects and the `subset.ratio` equals 0.5 then the highest cluster number to calculate is 10. In that case all elements above 10 will be removed from `cl.num` vector.

Value

`cls.stab.sim.ind.usr` returns a lists of matrices. Each matrix consists of the set of external similarity indices (which one similarity index see below) where number of columns is equal to `cl.num` vector length and row number is equal to `rep.num` value what means that each column contain a set of similarity indices computed for fixed number of clusters. The order of the matrices depends on `sim.ind.type` argument. Each element of this list correspond to one of similarity index type chosen thanks to `sim.ind.type` argument. The order of the names exactly match to the order given in those arguments description.

`cls.stab.opt.assign.usr` returns a vector. The vector consists of the set of cluster stability indices described in *Detecting stable clusters using principal component analysis* chapter *3.2* (see references). Vector length is equal to `cl.num` vector length what means that each position in vector is assigned to proper clusters' number given in `cl.num` argument.

Author(s)

Lukasz Nieweglowski

References

A. Ben-Hur and I. Guyon *Detecting stable clusters using principal component analysis*, http://citeseerx.ist.psu.edu/

C. D. Giurcaneanu, I. Tabus, I. Shmulevich, W. Zhang *Stability-Based Cluster Analysis Applied To Microarray Data*, http://citeseerx.ist.psu.edu/.

T. Lange, V. Roth, M. L. Braun and J. M. Buhmann *Stability-Based Validation of Clustering Solutions*, doi:10.1162/089976604773717621

See Also

Other cluster stability methods: `cls.stab.sim.ind`, `cls.stab.opt.assign`.

Functions that compare two different partitionings: `clv.Rand`, `dot.product`,`similarity.index`.

Examples

```
# load and prepare data
library(clv)
data(iris)
iris.data <- iris[,1:4]
```

```
# example of wrapper for partitioning algorithm
pam.clust <- function(data, clust.num) pam(data, clust.num, cluster.only=TRUE)

# example of wrapper for hierarchical algorithm
cutree.wrap <- function(clust.res, clust.num)  cutree(clust.res, clust.num)
agnes.single <- function(data) agnes(data, method="single")

# converting hierarchical algorithm to partitioning one
agnes.part1 <- function(data, clust.num) cutree.wrap( agnes.single(data), clust.num )
# the same using "cls.alg"
agnes.part2 <- cls.alg(agnes.single, cutree.wrap, fast=FALSE)

# fix arguments for cls.stab.* function
iter = c(2,4,5,7,9,12,15)

res1 = cls.stab.sim.ind.usr( iris.data, iter, pam.clust,
    sim.ind.type=c("rand","dot.pr","sim.ind"), rep.num=5, subset.ratio=0.7 )
res2 = cls.stab.opt.assign.usr( iris.data, iter, clust.alg=cls.alg(agnes.single, cutree.wrap) )

res3 = cls.stab.sim.ind.usr( iris.data, iter, agnes.part1,
     sim.ind.type=c("rand","dot.pr","sim.ind"), rep.num=5, subset.ratio=0.7 )
res4 = cls.stab.opt.assign.usr( iris.data, iter, clust.alg=agnes.part2 )

print(res1)
boxplot(res1$sim.ind)
plot(res2)
```

`clv.Davies.Bouldin` *Davies-Bouldin Index - Internal Measure*

Description

Function computes *Dunn index* - internal measure for given data and its partitioning.

Usage

```
clv.Davies.Bouldin( index.list, intracls, intercls)
```

Arguments

`index.list`	object returned by function `cls.scatt.data` or `cls.scatt.diss.mx`.
`intracls`	string `vector` containing one or more names of intra cluster distances. Available are:

1. if `index.list` is produced by `cls.scatt.data`: complete | average | centroid,
2. if `index.list` is produced by `cls.scatt.diss.mx`: complete | average.

`intercls` string `vector` containing one or more names of inter cluster diameters. Available are:

1. if `index.list` is produced by `cls.scatt.data`: single | complete | average | centroid | aveToCent | hausdorff.
2. if `index.list` is produced by `cls.scatt.diss.mx`: single | complete | average | hausdorff.

Details

Davies-Bouldin index is given by equation:

DB = (1/|C|) sum{forall i in 1:|C|} max[i != j] { (diam(Ci) + diam(Cj))/dist(Ci,Cj) }

i,j	- numbers of clusters which come from the same partitioning,
dist(Ck,Cl)	- inter cluster distance between clusters Ck and Cl,
diam(Cm)	- intra cluster diameter computed for cluster Cm,
|C|	- number of clusters.

Value

As output user gets the matrix of *Davies-Bouldin* indices. Matrix dimension depends on how many *diam* and *dist* measures are chosen by the user, normally `dim(D)=c(length(intercls),length(intracls))`. Each pair: (inter-cluster dist, intra-cluster diam) have its own position in result matrix.

Author(s)

Lukasz Nieweglowski

References

M. Halkidi, Y. Batistakis, M. Vazirgiannis *Clustering Validity Checking Methods : Part II*, http://citeseer.ist.psu.edu/537304.html

See Also

Functions which produce *index.list* input argument: `cls.scatt.data`, `cls.scatt.diss.mx`. Related functions: clv.Dunn.

Examples

```
# load and prepare data
library(clv)
data(iris)
iris.data <- iris[,1:4]

# cluster data
agnes.mod <- agnes(iris.data) # create cluster tree
v.pred <- as.integer(cutree(agnes.mod,5)) # "cut" the tree

intraclust = c("complete","average","centroid")
interclust = c("single", "complete", "average","centroid", "aveToCent", "hausdorff")

# compute Davies-Bouldin indicies (also Dunn indicies)
```

```
# 1. optimal solution:

# compute intercluster distances and intracluster diameters
cls.scatt <- cls.scatt.data(iris.data, v.pred, dist="manhattan")

# once computed valuse use in both functions
dunn1 <- clv.Dunn(cls.scatt, intraclust, interclust)
davies1 <- clv.Davies.Bouldin(cls.scatt, intraclust, interclust)

# 2. functional solution:

# define new Dunn and Davies.Bouldin functions
Dunn <- function(data,clust)
  clv.Dunn( cls.scatt.data(data,clust),
     intracls = c("complete","average","centroid"),
     intercls = c("single", "complete", "average","centroid", "aveToCent", "hausdorff")
  )
Davies.Bouldin <- function(data,clust)
  clv.Davies.Bouldin( cls.scatt.data(data,clust),
    intracls = c("complete","average","centroid"),
    intercls = c("single", "complete", "average","centroid", "aveToCent", "hausdorff")
  )

# compute indicies
dunn2 <- Dunn(iris.data, v.pred)
davies2 <- Davies.Bouldin(iris.data, v.pred)
```

`clv.Dens_bw` *Inter-cluster density - Internal Measure*

Description

Function computes *inter-cluster density*.

Usage

```
clv.DensBw(data, clust, scatt.obj, dist="euclidean")
```

Arguments

`data`	`matrix` or `data.frame` where columns correspond to variables and rows to observations
`clust`	integer `vector` with information about cluster id the object is assigned to. If vector is not integer type, it will be coerced with warning.
`scatt.obj`	object returned by `clv.Scatt` function.
`dist`	chosen metric: "euclidean" (default value), "manhattan", "correlation"

Details

The definition of *inter-cluster density* is given by equation:

Dens_bw = 1/(|C|*(|C|-1)) * sum{forall i in 1:|C|} sum{forall j in 1:|C| and j != i} *density*(u(i,j))/max{*density*(v(i)), *density*(v(j))}

where:

|C|	- number of clusters,
v(i), v(j)	- centers of clusters i and j,
u(i,j)	- middle point of the line segment defined by the clusters' centers v(i), v(j),
density(x)	- see below.

Let define function f(x,u):

f(x,u) = 0 if dist(x,u) > *stdev* (*stdev* is defined in clv.Scatt)
f(x,u) = 1 otherwise

Function f is used in definition of *density*(u):

density(u) = sum{forall i in 1:n(i,j)} f(xi,u)

where n(i,j) is the number of objects which belongs to clusters i and j and xi is such object.

This value is used by clv.SDbw.

Value

As result Dens_bw value is returned.

Author(s)

Lukasz Nieweglowski

See Also

clv.SD and clv.SDbw

Examples

```
# load and prepare data
library(clv)
data(iris)
iris.data <- iris[,1:4]

# cluster data
agnes.mod <- agnes(iris.data) # create cluster tree
v.pred <- as.integer(cutree(agnes.mod,5)) # "cut" the tree

# compute Dens_bw index
scatt <- clv.Scatt(iris.data, v.pred)
dens.bw <- clv.DensBw(iris.data, v.pred, scatt)
```

clv.Dis *Total separation between clusters - Internal Measure*

Description

Function computes *total separation between clusters*.

Usage

```
clv.Dis(cluster.center)
```

Arguments

cluster.center	numeric matrix or data.frame where columns correspond to variables and rows cluster centers.

Details

The definition of total separation between clusters is given by equation:

Dis = (Dmax/Dmin) * sum{forall i in 1:|C|} 1 /(sum{forall j in 1:|C|} ||vi - vj||)

where:

|C|	- number of clusters,
vi, vj	- centers of clusters i and j,
Dmax	- defined as: max{||vi - vj||: vi,vj - centers of clusters },
Dmin	- defined as: min{||vi - vj||: vi,vj - centers of clusters },
||x||	- means: sqrt(x*x').

This value is a part of clv.SD and clv.SDbw.

Value

As result Dis value is returned.

Author(s)

Lukasz Nieweglowski

References

M. Haldiki, Y. Batistakis, M. Vazirgiannis *On Clustering Validation Techniques*, http://citeseer.ist.psu.edu/513619.html

See Also

clv.SD and clv.SDbw

Examples

```
# load and prepare data
library(clv)
data(iris)
iris.data <- iris[,1:4]

# cluster data
agnes.mod <- agnes(iris.data) # create cluster tree
v.pred <- as.integer(cutree(agnes.mod,5)) # "cut" the tree

# compute Dis index
scatt <- clv.Scatt(iris.data, v.pred)
dis <- clv.Dis(scatt$cluster.center)
```

`clv.Dunn` *Dunn Index - Internal Measure*

Description

Function computes *Dunn index* - internal measure for given data and its partitioning.

Usage

```
clv.Dunn( index.list, intracls, intercls)
```

Arguments

`index.list`	object returned by function `cls.scatt.data` or `cls.scatt.diss.mx`.
`intracls`	string `vector` containing one or more names of intra cluster distances. Available are: 1. if `index.list` is produced by `cls.scatt.data`: complete | average | centroid, 2. if `index.list` is produced by `cls.scatt.diss.mx`: complete | average.
`intercls`	string `vector` containing one or more names of inter cluster diameters. Available are:

1. if `index.list` is produced by `cls.scatt.data`: single | complete | average | centroid | aveToCent | hausdorff.
2. if `index.list` is produced by `cls.scatt.diss.mx`: single | complete | average | hausdorff.

Details

Dunn index:

D = [min{ k,l - numbers of clusters } *dist*(Ck, Cl)]/[max{ m - cluster number } *diam*(Cm)]

k,l,m	- numbers of clusters which come from the same partitioning,
dist(Ck,Cl)	- inter cluster distance between clusters Ck and Cl,

diam(Cm) - intra cluster diameter computed for cluster Cm.

Value

As output user gets matrix of *Dunn* indices. Matrix dimension depends on how many *diam* and *dist* measures are chosen by the user, normally `dim(D)=c(length(intercls),length(intracls))`. Each pair: (inter-cluster dist, intra-cluster diam) have its own position in result matrix.

Author(s)

Lukasz Nieweglowski

References

M. Halkidi, Y. Batistakis, M. Vazirgiannis *Clustering Validity Checking Methods : Part II*, http://citeseer.ist.psu.edu/537304.html

See Also

Functions which produce *index.list* input argument: `cls.scatt.data`, `cls.scatt.diss.mx`. Related functions: clv.Davies.Bouldin.

Examples

```
# load and prepare data
library(clv)
data(iris)
iris.data <- iris[,1:4]

# cluster data
agnes.mod <- agnes(iris.data) # create cluster tree
v.pred <- as.integer(cutree(agnes.mod,5)) # "cut" the tree

intraclust = c("complete","average","centroid")
interclust = c("single", "complete", "average","centroid", "aveToCent", "hausdorff")

# compute Dunn indicies (also Davies-Bouldin indicies)
# 1. optimal solution:

# compute intercluster distances and intracluster diameters
cls.scatt <- cls.scatt.data(iris.data, v.pred, dist="manhattan")

# once computed valuse use in both functions
dunn1 <- clv.Dunn(cls.scatt, intraclust, interclust)
davies1 <- clv.Davies.Bouldin(cls.scatt, intraclust, interclust)

# 2. functional solution:

# define new Dunn and Davies.Bouldin functions
Dunn <- function(data,clust)
  clv.Dunn( cls.scatt.data(data,clust),
     intracls = c("complete","average","centroid"),
```

```
     intercls = c("single", "complete", "average","centroid", "aveToCent", "hausdorff")
  )
Davies.Bouldin <- function(data,clust)
  clv.Davies.Bouldin( cls.scatt.data(data,clust),
    intracls = c("complete","average","centroid"),
    intercls = c("single", "complete", "average","centroid", "aveToCent", "hausdorff")
  )

# compute indicies
dunn2 <- Dunn(iris.data, v.pred)
davies2 <- Davies.Bouldin(iris.data, v.pred)
```

`clv.Scatt` *Average scattering for clusters - Internal Measure*

Description

Function computes *average scattering for clusters*.

Usage

```
clv.Scatt(data, clust, dist="euclidean")
```

Arguments

`data`	`numeric matrix` or `data.frame` where columns correspond to variables and rows to observations
`clust`	integer `vector` with information about cluster id the object is assigned to. If vector is not integer type, it will be coerced with warning.
`dist`	choosen metric: "euclidean" (default value), "manhattan", "correlation"

Details

Let *scatter for set* X assigned as *sigma*(X) be defined as vector of variances computed for particular dimensions. *Average scattering for clusters* is defined as:

`Scatt` = (1/|C|) * sum{forall i in 1:|C|} ||*sigma*(Ci)||/||*sigma*(X)||

where:

|C|	- number of clusters,
i	- cluster id,
Ci	- cluster with id 'i',
X	- set with all objects,
||x||	- sqrt(x*x').

Standard deviation is defined as:

`stdev` = (1/|C|) * sqrt(sum{forall i in 1:|C|} ||*sigma*(Ci)||)

Value

As result `list` with three values is returned.

`Scatt`	- *average scattering for clusters* value,
`stdev`	- *standard deviation* value,
`cluster.center`	- numeric `matrix` where columns correspond to variables and rows to cluster centers.

Author(s)

Lukasz Nieweglowski

References

M. Haldiki, Y. Batistakis, M. Vazirgiannis *On Clustering Validation Techniques*, http://citeseer.ist.psu.edu/513619.html

See Also

`clv.SD` and `clv.SDbw`

Examples

```
# load and prepare data
library(clv)
data(iris)
iris.data <- iris[,1:4]

# cluster data
agnes.mod <- agnes(iris.data) # create cluster tree
v.pred <- as.integer(cutree(agnes.mod,5)) # "cut" the tree

# compute Scatt index
scatt <- clv.Scatt(iris.data, v.pred)
```

`clv.SD, clv.SDbw` *SD, SDbw - Internal Measures*

Description

Function computes *SD* and *S_Dbw validity indices*.

Usage

```
clv.SD(scatt, dis, alfa)
clv.SDbw(scatt, dens)
```

Arguments

scatt	*average scattering for cluster* value computed using clv.Scatt function.
dis	*total separation between clusters* value computed using clv.Dis function.
dens	*inter-cluster density* value computed using clv.DensBw function.
alfa	weighting factor (normally equal to Dis(cmax) where cmax is the maximum number of input clusters).

Details

SD validity index is defined by equation:

SD = *scatt**alfa + *dis*

where *scatt* means *average scattering for clusters* defined in clv.Scatt. *S_Dbw validity index* is defined by equation:

S_Dbw = *scatt* + *dens*

where *dens* is defined in clv.DensBw.

Value

As result of clv.SD function *SD validity index* is returned. As result of clv.SDbw function *S_Dbw validity index* is returned.

Author(s)

Lukasz Nieweglowski

References

M. Haldiki, Y. Batistakis, M. Vazirgiannis *On Clustering Validation Techniques*, http://citeseer.ist.psu.edu/513619.html

See Also

clv.Scatt, clv.Dis and clv.DensBw

Examples

```
# load and prepare
library(clv)
data(iris)
iris.data <- iris[,1:4]

# cluster data
agnes.mod <- agnes(iris.data) # create cluster tree
v.pred <- as.integer(cutree(agnes.mod,5)) # "cut" the tree

# prepare proper input data for SD and S_Dbw indicies
scatt <- clv.Scatt(iris.data, v.pred)
dis <- clv.Dis(scatt$cluster.center)
```

```
dens.bw <- clv.DensBw(iris.data, v.pred, scatt)

# compute  SD and S_Dbw indicies
SD <- clv.SD(scatt$Scatt, dis, alfa=5) # alfa is equal to number of clusters
SDbw <- clv.SDbw(scatt$Scatt, dens.bw)
```

`confusion.matrix` *Confusion Matrix - External Measures, Cluster Stability*

Description

For two different partitioning function computes *confusion matrix*.

Usage

```
confusion.matrix(clust1, clust2)
```

Arguments

`clust1`	integer `vector` with information about cluster id the object is assigned to. If vector is not integer type, it will be coerced with warning.
`clust2`	integer `vector` with information about cluster id the object is assigned to. If vector is not integer type, it will be coerced with warning.

Details

Let P and P' be two different partitioning of the same data. Partitionings are represent as two vectors `clust1`, `clust2`. Both vectors should have the same length. Confusion matrix measures the size of intersection between clusters comming from P and P' according to equation:

M[i,j] = | intersection of P(i) and P'(j) |

where:

P(i)	- cluster which belongs to partitioning P,
P'(j)	- cluster which belongs to partitioning P',
|A|	- cardinality of set A.

Value

`cls.set.section` returns a n x m integer `matrix` where n = |P| and m = |P'| defined above.

Author(s)

Lukasz Nieweglowski

See Also

Result used in `similarity.index`.

Examples

```
# create two different subsamples
mx1 <- matrix(as.integer( c(1,2,3,4,5,6,1,1,2,2,3,3) ), 6, 2 )
mx2 <- matrix(as.integer( c(1,2,4,5,6,7,1,1,2,2,3,3) ), 6, 2 )
# find section
m = cls.set.section(mx1,mx2)
confusion.matrix(as.integer(m[,2]),as.integer(m[,3]))
```

---

connectivity *Connectivity Index - Internal Measure*

Description

Function evaluates *connectivity* index.

Usage

```
connectivity(data,clust,neighbour.num, dist="euclidean")
connectivity.diss.mx(diss.mx,clust,neighbour.num)
```

Arguments

`data`	`numeric matrix` or `data.frame` where columns correspond to variables and rows to observations
`diss.mx`	square, symetric `numeric matrix` or `data.frame`, representation of dissimilarity matrix where infomartion about distances between objects is stored.
`clust`	integer `vector` with information about cluster id the object is assigned to. If vector is not integer type, it will be coerced with warning.
`neighbour.num`	value which tells how many nearest neighbors for every object should be checked.
`dist`	chosen metric: "euclidean" (default value), "manhattan", "correlation" (variable enable only in `connectivity` function).

Details

For given data and its partitioning *connectivity* index is computed. For choosen pattern `neighbour.num` nearest neighbours are found and sorted from closest to most further. Alghorithm checks if those neighbours are assigned to the same cluster. At the beggining *connectivity* value is equal 0 and increase with value:

1/i	when i-th nearest neighbour is not assigned to the same cluster,
0	otherwise.

Procedure is repeated for all patterns which comming from our data set. All values received for particular pattern are added and creates main *connectivity* index.

Value

`connectivity` returns a *connectivity* value.

Author(s)

Lukasz Nieweglowski

References

J. Handl, J. Knowles and D. B. Kell *Sumplementary material to computational cluster validation in post-genomic data analysis*, http://dbkgroup.org/handl/clustervalidation/supplementary.pdf

Examples

```
# load and prepare data
library(clv)
data(iris)
iris.data <- iris[,1:4]

# cluster data
pam.mod <- pam(iris.data,5) # create five clusters
v.pred <- as.integer(pam.mod$clustering) # get cluster ids associated to gived data objects

# compute connectivity index using data and its clusterization
conn1 <- connectivity(iris.data, v.pred, 10)
conn2 <- connectivity(iris.data, v.pred, 10, dist="manhattan")
conn3 <- connectivity(iris.data, v.pred, 10, dist="correlation")

# the same using dissimilarity matrix
iris.diss.mx <- as.matrix(daisy(iris.data))
conn4 <- connectivity.diss.mx(iris.diss.mx, v.pred, 10)
```

---

dot.product *Cosine similarity measure - External Measure, Cluster Stability*

Description

Similarity index based on dot product is the measure which estimates how those two different partitionings, that comming from one dataset, are different from each other.

Usage

```
dot.product(clust1, clust2)
```

Arguments

clust1	integer vector with information about cluster id the object is assigned to. If vector is not integer type, it will be coerced with warning.
clust2	integer vector with information about cluster id the object is assigned to. If vector is not integer type, it will be coerced with warning.

Details

Two input vectors keep information about two different partitionings of the same subset comming from one data set. For each partitioning (let say P and P') its matrix representation is created. Let P[i,j] and P'[i,j] each defines as:

P[i,j] = 1 when object i and j belongs to the same cluster and i != j
P[i,j] = 0 in other case

Two matrices are needed to compute *dot product* using formula:

<P,P'> = sum(forall i and j) P[i,j]*P'[i,j]

This *dot product* satisfy Cauchy-Schwartz inequality <P,P'> <= <P,P>*<P',P'>. As result we get *cosine similarity measure*: <P,P'>/sqrt(<P,P>*<P',P'>)

Value

dot.product returns a *cosine similarity measure* of two partitionings. NaN is returned when in any partitioning each cluster contains only one object.

Author(s)

Lukasz Nieweglowski

References

A. Ben-Hur and I. Guyon *Detecting stable clusters using principal component analysis*, http://citeseer.ist.psu.edu/528061.html

T. Lange, V. Roth, M. L. Braun and J. M. Buhmann *Stability-Based Validation of Clustering Solutions*, doi:10.1162/089976604773717621

See Also

Other external measures: std.ext, similarity.index

Examples

```
# dot.product function(and also similarity.index) is used to compute
# cluster stability, additional stability functions will be
# defined - as its arguments some additional functions (wrappers)
# will be needed

# define wrappers
pam.wrapp <-function(data)
{
```

```
 return( as.integer(data$clustering) )
}

identity <- function(data) { return( as.integer(data) ) }

agnes.average <- function(data, clust.num)
{
 return( cutree( agnes(data,method="average"), clust.num ) )
}

# define cluster stability function - cls.stabb

# cls.stabb arguments description:
# data - data to be clustered
# clust.num - number of clusters to which data will be clustered
# sample.num - number of pairs of data subsets to be clustered,
#              each clustered pair will be given as argument for
#              dot.product and similarity.index functions
# ratio - value comming from (0,1) section:
#    0 - means sample emtpy subset,
#    1 - means chose all "data" objects
# method - cluster method (see wrapper functions)
# wrapp - function which extract information about cluster id assigned
#         to each clustered object

# as a result mean of dot.product (and similarity.index) results,
# computed for subsampled pairs of subsets is given
cls.stabb <- function( data, clust.num, sample.num , ratio, method, wrapp  )
{
 dot.pr  = 0
 sim.ind = 0
 obj.num = dim(data)[1]

 for( j in 1:sample.num )
 {
  smp1 = sort( sample( 1:obj.num, ratio*obj.num ) )
  smp2 = sort( sample( 1:obj.num, ratio*obj.num ) )

  d1 = data[smp1,]
  cls1 = wrapp( method(d1,clust.num) )

  d2 = data[smp2,]
  cls2 = wrapp( method(d2,clust.num) )

  clsm1 = t(rbind(smp1,cls1))
  clsm2 = t(rbind(smp2,cls2))

  m = cls.set.section(clsm1, clsm2)
  cls1 = as.integer(m[,2])
  cls2 = as.integer(m[,3])
  cnf.mx = confusion.matrix(cls1,cls2)
  std.ms = std.ext(cls1,cls2)
```

```
  # external measures - compare partitioning
  dt = dot.product(cls1,cls2)
  si = similarity.index(cnf.mx)

  if( !is.nan(dt) ) dot.pr = dot.pr + dt/sample.num
  sim.ind = sim.ind + si/sample.num
 }
 return( c(dot.pr, sim.ind) )
}

# load and prepare data
library(clv)
data(iris)
iris.data <- iris[,1:4]

# fix arguments for cls.stabb function
iter = c(2,3,4,5,6,7,9,12,15)
smp.num = 5
sub.smp.ratio = 0.8

# cluster stability for PAM
print("PAM method:")
for( i in iter )
{
 result = cls.stabb(iris.data, clust.num=i, sample.num=smp.num,
           ratio=sub.smp.ratio, method=pam, wrapp=pam.wrapp)
 print(result)
}

# cluster stability for Agnes (average-link)
print("Agnes (single) method:")
for( i in iter )
{
 result = cls.stabb(iris.data, clust.num=i, sample.num=smp.num,
            ratio=sub.smp.ratio, method=agnes.average, wrapp=identity)
 print(result)
}
```

`similarity.index` *Similarity index based on confusion matrix - External Measure, Cluster Stability*

Description

Similarity index based on confusion matrix is the measure which estimates how those two different partitionings, that comming from one dataset, are different from each other. For given `matrix` returned by `confusion.matrix` function *similarity index* is found.

Usage

```
similarity.index(cnf.mx)
```

Arguments

cnf.mx	not negative, integer matrix or data.frame which represents object returned by confusion.matrix function.

Details

Let M is n x m (n <= m) *confusion matrix* for partitionings P and P'. Any one to one function *sigma*: {1,2,...,n} -> {1,2,... ,m}. is called *assignment* (or also *association*). Using set of *assignment* functions, A(P,P') index defined as:

A(P,P') = max{ sum(forall i in 1:length(*sigma*)) M[i,*sigma(i)*]: *sigma* is an *assignment* }

is found. (*Assignment* which satisfy above equation is called *optimal assignment*). Using this value we can compute *similarity index* S(P.P') = (A(P,P') - 1)/(N - 1) where N is quantity of partitioned objects (here is equal to sum(M)).

Value

similarity.index returns value from section [0,1] which is a measure of similarity between two different partitionings. Value 1 means that we have two the same partitionings.

Author(s)

Lukasz Nieweglowski

References

C. D. Giurcaneanu, I. Tabus, I. Shmulevich, W. Zhang *Stability-Based Cluster Analysis Applied To Microarray Data*, http://citeseer.ist.psu.edu/577114.html.

T. Lange, V. Roth, M. L. Braun and J. M. Buhmann *Stability-Based Validation of Clustering Solutions*, doi:10.1162/089976604773717621

See Also

confusion.matrix as matrix representation of two partitionings. Other functions created to compare two different partitionings: std.ext, dot.product

Examples

```
# similarity.index function(and also dot.product) is used to compute
# cluster stability, additional stability functions will be
# defined - as its arguments some additional functions (wrappers)
# will be needed

# define wrappers
pam.wrapp <-function(data)
{
 return( as.integer(data$clustering) )
}

identity <- function(data) { return( as.integer(data) ) }
```

```
agnes.average <- function(data, clust.num)
{
 return( cutree( agnes(data,method="average"), clust.num ) )
}

# define cluster stability function - cls.stabb

# cls.stabb arguments description:
# data - data to be clustered
# clust.num - number of clusters to which data will be clustered
# sample.num - number of pairs of data subsets to be clustered,
#              each clustered pair will be given as argument for
#              dot.product and similarity.index functions
# ratio - value comming from (0,1) section:
#    0 - means sample emtpy subset,
#    1 - means chose all "data" objects
# method - cluster method (see wrapper functions)
# wrapp - function which extract information about cluster id assigned
#         to each clustered object

# as a result mean of similarity.index (and dot.product) results,
# computed for subsampled pairs of subsets is given
cls.stabb <- function( data, clust.num, sample.num , ratio, method, wrapp  )
{
 dot.pr  = 0
 sim.ind = 0
 obj.num = dim(data)[1]

 for( j in 1:sample.num )
 {
  smp1 = sort( sample( 1:obj.num, ratio*obj.num ) )
  smp2 = sort( sample( 1:obj.num, ratio*obj.num ) )

  d1 = data[smp1,]
  cls1 = wrapp( method(d1,clust.num) )

  d2 = data[smp2,]
  cls2 = wrapp( method(d2,clust.num) )

  clsm1 = t(rbind(smp1,cls1))
  clsm2 = t(rbind(smp2,cls2))

  m = cls.set.section(clsm1, clsm2)
  cls1 = as.integer(m[,2])
  cls2 = as.integer(m[,3])
  cnf.mx = confusion.matrix(cls1,cls2)
  std.ms = std.ext(cls1,cls2)

  # external measures - compare partitioning
  dt = dot.product(cls1,cls2)
  si = similarity.index(cnf.mx)
```

```
  if( !is.nan(dt) ) dot.pr = dot.pr + dt/sample.num
  sim.ind = sim.ind + si/sample.num
 }
 return( c(dot.pr, sim.ind) )
}

# load and prepare data
library(clv)
data(iris)
iris.data <- iris[,1:4]

# fix arguments for cls.stabb function
iter = c(2,3,4,5,6,7,9,12,15)
smp.num = 5
sub.smp.ratio = 0.8

# cluster stability for PAM
print("PAM method:")
for( i in iter )
{
 result = cls.stabb(iris.data, clust.num=i, sample.num=smp.num,
            ratio=sub.smp.ratio, method=pam, wrapp=pam.wrapp)
 print(result)
}

# cluster stability for Agnes (average-link)
print("Agnes (single) method:")
for( i in iter )
{
 result = cls.stabb(iris.data, clust.num=i, sample.num=smp.num,
            ratio=sub.smp.ratio, method=agnes.average, wrapp=identity)
 print(result)
}
```

`std.ext` *Standard External Measures: Rand index, Jaccard coefficient etc.*

Description

Group of functions which compute standard external measures such as: Rand statistic and Folkes and Mallows index, Jaccard coefficient etc.

Usage

```
std.ext(clust1, clust2)
clv.Rand(external.ind)
clv.Jaccard(external.ind)
clv.Folkes.Mallows(external.ind)
clv.Phi(external.ind)
clv.Russel.Rao(external.ind)
```

Arguments

clust1	integer vector with information about cluster id the object is assigned to. If vector is not integer type, it will be coerced with warning.
clust2	integer vector with information about cluster id the object is assigned to. If vector is not integer type, it will be coerced with warning.
external.ind	vector or list with four values SS,SD,DS,DD which are result of function std.ext

Details

Two input vectors keep information about two different partitionings (let say P and P') of the same data set X. We refer to a pair of points (xi, xj) (we assume that i != j) from the data set using the following terms:

SS - number of pairs where both points belongs to the same cluster in both partitionings,
SD - number of pairs where both points belongs to the same cluster in partitioning P but in P' do not,
DS - number of pairs where in partitioning P both point belongs to different clusters but in P' do not,
DD - number of pairs where both objects belongs to different clusters in both partitionings.

Those values are used to compute (M = SS + SD + DS +DD):

Rand statistic	R = (SS + DD)/M
Jaccard coefficient	J = SS/(SS + SD + DS)
Folkes and Mallows index	FM = sqrt(SS/(SS + SD))*sqrt(SS/(SS + DS))
Russel and Rao index	RR = SS/M
Phi index	Ph = (SS*DD - SD*DS)/((SS+SD)(SS+DS)(SD+DD)(DS+DD)).

Value

std.ext returns a list containing four values: SS, SD, DS, DD.
clv.Rand returns R value.
clv.Jaccard returns J value.
clv.Folkes.Mallows returns FM value.
clv.Phi returns Ph value.
clv.Russel.Rao returns RR value.

Author(s)

Lukasz Nieweglowski

References

G. Saporta and G. Youness *Comparing two partitions: Some Proposals and Experiments*. http://cedric.cnam.fr/PUBLIS/RC405.pdf

See Also

Other measures created to compare two partitionings: dot.product, similarity.index

Examples

```
# load and prepare data
library(clv)
data(iris)
iris.data <- iris[,1:4]

# cluster data
pam.mod <- pam(iris.data,3) # create three clusters
v.pred <- as.integer(pam.mod$clustering) # get cluster ids associated to given data objects
v.real <- as.integer(iris$Species) # get also real cluster ids

# compare true clustering with those given by the algorithm
# 1. optimal solution:

# use only once std.ext function
std <- std.ext(v.pred, v.real)
# to compute three indicies based on std.ext result
rand1 <- clv.Rand(std)
jaccard1 <- clv.Jaccard(std)
folk.mal1 <- clv.Folkes.Mallows(std)

# 2. functional solution:

# prepare set of functions which compare two clusterizations
Rand <- function(clust1,clust2) clv.Rand(std.ext(clust1,clust2))
Jaccard <- function(clust1,clust2) clv.Jaccard(std.ext(clust1,clust2))
Folkes.Mallows <- function(clust1,clust2) clv.Folkes.Mallows(std.ext(clust1,clust2))

# compute indicies
rand2 <- Rand(v.pred,v.real)
jaccard2 <- Jaccard(v.pred,v.real)
folk.mal2 <- Folkes.Mallows(v.pred,v.real)
```

---

wcls/bcls.matrix *Matrix Cluster Scatter Measures*

Description

Functions compute two base matrix cluster scatter measures.

Usage

```
wcls.matrix(data,clust,cluster.center)
bcls.matrix(cluster.center,cluster.size,mean)
```

Arguments

`data`	`numeric matrix` or `data.frame` where columns correspond to variables and rows to observations
`clust`	integer `vector` with information about cluster id the object is assigned to. If vector is not integer type, it will be coerced with warning.
`cluster.center`	`matrix` or `data.frame` where columns correspond to variables and rows to cluster centers defined by `data` and `clust` parameters.
`cluster.size`	integer `vector` with information about size of each cluster computed using `clust` vector.
`mean`	*mean* of all data objects.

Details

There are two base matrix scatter measures.

1. *within-cluster scatter measure* defined as:

W = sum(forall k in 1:cluster.num) W(k)

where W(k) = sum(forall x) (x - m(k))*(x - m(k))'

x	- object belongs to cluster k,
m(k)	- center of cluster k.

2. *between-cluster scatter measure* defined as:

B = sum(forall k in 1:cluster.num) |C(k)|*(m(k) - m)*(m(k) - m)'

|C(k)|	- size of cluster k,
m(k)	- center of cluster k,
m	- center of all data objects.

Value

`wcls.matrix`	returns W matrix (*within-cluster scatter measure*),
`bcls.matrix`	returns B matrix (*between-cluster scatter measure*).

Author(s)

Lukasz Nieweglowski

References

T. Hastie, R. Tibshirani, G. Walther *Estimating the number of data clusters via the Gap statistic*, http://citeseer.ist.psu.edu/tibshirani00estimating.html

Examples

```
# load and prepare data
library(clv)
data(iris)
iris.data <- iris[,1:4]

# cluster data
pam.mod <- pam(iris.data,5) # create five clusters
v.pred <- as.integer(pam.mod$clustering) # get cluster ids associated to given data objects

# compute cluster sizes, center of each cluster
# and mean from data objects
cls.attr <- cls.attrib(iris.data, v.pred)
center <- cls.attr$cluster.center
size <- cls.attr$cluster.size
iris.mean <- cls.attr$mean

# compute matrix scatter measures
W.matrix <- wcls.matrix(iris.data, v.pred, center)
B.matrix <- bcls.matrix(center, size, iris.mean)
T.matrix <- W.matrix + B.matrix

# example of indices based on W, B i T matrices
mx.scatt.crit1 = sum(diag(W.matrix))
mx.scatt.crit2 = sum(diag(B.matrix))/sum(diag(W.matrix))
mx.scatt.crit3 = det(W.matrix)/det(T.matrix)
```

Index

∗ **cluster**
cls.attrib, 2
cls.scatt.data, 3
cls.set.section, 5
cls.stab.sim.ind, 6
cls.stab.sim.ind.usr, 10
clv.Davies.Bouldin, 13
clv.Dens_bw, 15
clv.Dis, 17
clv.Dunn, 18
clv.Scatt, 20
clv.SD, clv.SDbw, 21
confusion.matrix, 23
connectivity, 24
dot.product, 25
similarity.index, 28
std.ext, 31
wcls/bcls.matrix, 33

agnes, *7*

bcls.matrix, *2*
bcls.matrix *(*wcls/bcls.matrix*)*, 33

cls.alg *(*cls.stab.sim.ind.usr*)*, 10
cls.attrib, 2
cls.scatt.data, 3, *13, 14*, *18, 19*
cls.scatt.diss.mx, *13, 14*, *18, 19*
cls.scatt.diss.mx *(*cls.scatt.data*)*, 3
cls.set.section, 5
cls.stab.opt.assign, *11, 12*
cls.stab.opt.assign *(*cls.stab.sim.ind*)*, 6
cls.stab.opt.assign.usr, *9*
cls.stab.opt.assign.usr *(*cls.stab.sim.ind.usr*)*, 10
cls.stab.sim.ind, 6, *11, 12*
cls.stab.sim.ind.usr, *9*, 10
clv.Davies.Bouldin, *5*, 13, *19*
clv.Dens_bw, 15
clv.DensBw, *22*
clv.DensBw *(*clv.Dens_bw*)*, 15
clv.Dis, *2*, 17, *22*
clv.Dunn, *5*, *14*, 18
clv.Folkes.Mallows *(*std.ext*)*, 31
clv.Jaccard, *8*
clv.Jaccard *(*std.ext*)*, 31
clv.Phi *(*std.ext*)*, 31
clv.Rand, *8, 9*, *12*
clv.Rand *(*std.ext*)*, 31
clv.Russel.Rao *(*std.ext*)*, 31
clv.Scatt, *15, 16*, 20, *22*
clv.SD, *16, 17*, *21*
clv.SD *(*clv.SD, clv.SDbw*)*, 21
clv.SD, clv.SDbw, 21
clv.SDbw, *16, 17*, *21*
clv.SDbw *(*clv.SD, clv.SDbw*)*, 21
confusion.matrix, *6*, 23, *28, 29*
connectivity, 24

dot.product, *6–10*, *12*, 25, *29*, *33*

hclust, *7*

similarity.index, *7–10*, *12*, *23*, *26*, 28, *33*
std.ext, *6, 7*, *10*, *26*, *29*, 31

wcls.matrix, *2*
wcls.matrix *(*wcls/bcls.matrix*)*, 33
wcls/bcls.matrix, 33