Package ‘cluster’

May 1, 2019

Version 2.0.9
Date 2019-05-01
Priority recommended
Title "Finding Groups in Data": Cluster Analysis Extended Rousseeuw et al.
Description Methods for Cluster analysis. Much extended the original from Peter Rousseeuw, Anja Struyf and Mia Hubert, based on Kaufman and Rousseeuw (1990) "Finding Groups in Data".
Maintainer Martin Maechler <maechler@stat.math.ethz.ch>
Depends R (>= 3.3.0)
Imports graphics, grDevices, stats, utils
Suggests MASS, Matrix
SuggestsNote MASS: two examples using cov.rob() and mvrnorm(); Matrix tools for testing
LazyLoad yes
LazyData yes
ByteCompile yes
BuildResaveData no
License GPL (>= 2)
URL https://svn.r-project.org/R-packages/trunk/cluster
NeedsCompilation yes
Author Martin Maechler [aut, cre] (https://orcid.org/0000-0002-8685-9910), Peter Rousseeuw [aut] (Fortran original, <https://orcid.org/0000-0002-3807-5353>), Anja Struyf [aut] (S original), Mia Hubert [aut] (S original, <https://orcid.org/0000-0001-6398-4850>), Kurt Hornik [trl, ctb] (port to R; maintenance(1999-2000), <https://orcid.org/0000-0003-4198-9911>), Matthias Studer [ctb], Pierre Roudier [ctb]
Juan Gonzalez [ctb],
Kamil Kozłowski [ctb],
Erich Schubert [ctb] (fastpam options for pam(),
<https://orcid.org/0000-0001-9143-4880>)

Repository CRAN

Date/Publication 2019-05-01 20:30:15 UTC

R topics documented:

agnes ......................................................... 3
agnes.object ............................................... 7
agriculture ............................................... 9
animals ...................................................... 10
bannerplot ............................................... 11
chorSub ................................................... 12
clara ....................................................... 13
clara.object ............................................. 16
clusGap .................................................... 17
clusplot ................................................... 21
clusplot.default ......................................... 23
coefficient.hclust ....................................... 27
daissy ..................................................... 28
diana ....................................................... 32
dissimilarity.object ..................................... 34
ellipsoidhull ............................................. 35
fanny ....................................................... 37
fanny.object ............................................... 40
flower ..................................................... 41
lower.to.upper.tri.inds .................................. 42
mona ....................................................... 43
mona.object ............................................... 45
pam ......................................................... 45
pam.object ............................................... 49
partition.object .......................................... 50
plantTraits ............................................... 52
plot.agnes ............................................... 54
plot.diana ............................................... 56
plot.mona ............................................... 57
plot.partition ............................................ 59
pltree ..................................................... 61
pluton ..................................................... 62
predict.ellipsoid ........................................ 63
print.agnes ............................................... 64
print.clara ............................................... 65
print.diana ............................................... 65
print.dissimilarity ...................................... 66
print.fanny ............................................... 67
Agglomerative Nesting (Hierarchical Clustering)

Description

Computes agglomerative hierarchical clustering of the dataset.

Usage

```r
agnes(x, diss = inherits(x, "dist"), metric = "euclidean", 
      stand = FALSE, method = "average", par.method, 
      keep.diss = n < 100, keep.data = !diss, trace.lev = 0)
```

Arguments

- `x`:
  - Data matrix or data frame, or dissimilarity matrix, depending on the value of the `diss` argument.
  - In case of a matrix or data frame, each row corresponds to an observation, and each column corresponds to a variable. All variables must be numeric. Missing values (NAs) are allowed.
  - In case of a dissimilarity matrix, `x` is typically the output of `daisy` or `dist`. Also a vector with length `n*(n-1)/2` is allowed (where `n` is the number of observations), and will be interpreted in the same way as the output of the above-mentioned functions. Missing values (NAs) are not allowed.

- `diss`:
  - Logical flag: if TRUE (default for `dist` or dissimilarity objects), then `x` is assumed to be a dissimilarity matrix. If FALSE, then `x` is treated as a matrix of observations by variables.
metric character string specifying the metric to be used for calculating dissimilarities between observations. The currently available options are "euclidean" and "manhattan". Euclidean distances are root sum-of-squares of differences, and manhattan distances are the sum of absolute differences. If x is already a dissimilarity matrix, then this argument will be ignored.

stand logical flag: if TRUE, then the measurements in x are standardized before calculating the dissimilarities. Measurements are standardized for each variable (column), by subtracting the variable's mean value and dividing by the variable's mean absolute deviation. If x is already a dissimilarity matrix, then this argument will be ignored.

method character string defining the clustering method. The six methods implemented are "average" (unweighted pair-group arithmetic average method, aka ‘UPGMA’), "single" (single linkage), "complete" (complete linkage), "ward" (Ward's method), "weighted" (weighted average linkage, aka 'WPGMA'), its generalization "flexible" which uses (a constant version of) the Lance-Williams formula and the parNmethod argument, and "gaverage" a generalized "average" aka “flexible UPGMA" method also using the Lance-Williams formula and parNmethod. The default is "average".

parNmethod If method is "flexible" or "gaverage", a numeric vector of length 1, 3, or 4, (with a default for "gaverage"), see in the details section.

keep.diss, keep.data logicals indicating if the dissimilarities and/or input data x should be kept in the result. Setting these to FALSE can give much smaller results and hence even save memory allocation time.

trace.lev integer specifying a trace level for printing diagnostics during the algorithm. Default 0 does not print anything; higher values print increasingly more.

Details

agnes is fully described in chapter 5 of Kaufman and Rousseeuw (1990). Compared to other agglomerative clustering methods such as hclust, agnes has the following features: (a) it yields the agglomerative coefficient (see agnes.object) which measures the amount of clustering structure found; and (b) apart from the usual tree it also provides the banner, a novel graphical display (see plot.agnes).

The agnes-algorithm constructs a hierarchy of clusterings. At first, each observation is a small cluster by itself. Clusters are merged until only one large cluster remains which contains all the observations. At each stage the two nearest clusters are combined to form one larger cluster.

For method="average", the distance between two clusters is the average of the dissimilarities between the points in one cluster and the points in the other cluster.

In method="single", we use the smallest dissimilarity between a point in the first cluster and a point in the second cluster (nearest neighbor method).

When method="complete", we use the largest dissimilarity between a point in the first cluster and a point in the second cluster (furthest neighbor method).

The method = "flexible" allows (and requires) more details: The Lance-Williams formula specifies how dissimilarities are computed when clusters are agglomerated (equation (32) in K&R(1990),...
If clusters $C_1$ and $C_2$ are agglomerated into a new cluster, the dissimilarity between their union and another cluster $Q$ is given by

$$D(C_1 \cup C_2, Q) = \alpha_1 * D(C_1, Q) + \alpha_2 * D(C_2, Q) + \beta * D(C_1, C_2) + \gamma * |D(C_1, Q) - D(C_2, Q)|,$$

where the four coefficients $(\alpha_1, \alpha_2, \beta, \gamma)$ are specified by the vector `par.method`, either directly as a vector of length 4, or (more conveniently) if `par.method` is of length 1, say $\alpha$, `par.method` is extended to give the “Flexible Strategy” (K&R(1990), p.236 f) with Lance-Williams coefficients ($\alpha_1 = \alpha_2 = \alpha$, $\beta = 1 - 2\alpha$, $\gamma = 0$).

Also, if `length(par.method) == 3`, $\gamma = 0$ is set.

Care and expertise is probably needed when using method = "flexible" particularly for the case when `par.method` is specified of longer length than one. Since `cluster` version 2.0, choices leading to invalid merge structures now signal an error (from the C code already). The weighted average (method="weighted") is the same as method="flexible", `par.method = 0.5`. Further, method= "single" is equivalent to method="flexible", `par.method = c(0.5,.5,0,-.5)`, and method="complete" is equivalent to method="flexible", `par.method = c(0.5,.5,0,.5)`.

The method = "gaaverage" is a generalization of "average", aka “flexible UPGMA” method, and is (a generalization of the approach) detailed in Belbin et al. (1992). As "flexible", it uses the Lance-Williams formula above for dissimilarity updating, but with $\alpha_1$ and $\alpha_2$ not constant, but proportional to the sizes $n_1$ and $n_2$ of the clusters $C_1$ and $C_2$ respectively, i.e,

$$\alpha_j = \alpha'_j \frac{n_1}{n_1 + n_2},$$

where $\alpha'_1$, $\alpha'_2$ are determined from `par.method`, either directly as $(\alpha_1, \alpha_2, \beta, \gamma)$ or $(\alpha_1, \alpha_2, \beta)$ with $\gamma = 0$, or (less flexibly, but more conveniently) as follows:

Belbin et al proposed “flexible beta”, i.e. the user would only specify $\beta$ (as `par.method`), sensibly in

$$-1 \leq \beta < 1,$$

and $\beta$ determines $\alpha'_1$ and $\alpha'_2$ as

$$\alpha'_j = 1 - \beta,$$

and $\gamma = 0$.

This $\beta$ may be specified by `par.method` (as length 1 vector), and if `par.method` is not specified, a default value of -0.1 is used, as Belbin et al recommend taking a $\beta$ value around -0.1 as a general agglomerative hierarchical clustering strategy.

Note that method = "gaaverage", `par.method = 0` (or `par.method = c(1,1,0,0)`) is equivalent to the `agnes()` default method "average".

Value

an object of class "agnes" (which extends "twins") representing the clustering. See `agnes.object` for details, and methods applicable.

BACKGROUND

Cluster analysis divides a dataset into groups (clusters) of observations that are similar to each other.

Hierarchical methods like `agnes`, `diana`, and `mona` construct a hierarchy of clusterings, with the number of clusters ranging from one to the number of observations.
Partitioning methods like pam, clara, and fanny require that the number of clusters be given by the user.

Author(s)

Method "gaverage" has been contributed by Pierre Roudier, Landcare Research, New Zealand.

References


See Also

agnes.object, daisy, diana, dist, hclust, plot.agnes, twins.object.

Examples

data(votes.repub)
agn1 <- agnes(votes.repub, metric = "manhattan", stand = TRUE)
plot(agn1)

op <- par(mfrow=c(2,2))
agn2 <- agnes(daisy(votes.repub), diss = TRUE, method = "complete")
plot(agn2)
## alpha = 0.625 ==> beta = -1/4 is "recommended" by some
agn5 <- agnes(votes.repub, method = "flexible", par.meth = 0.625)
plot(agn5)
par(op)

## "show" equivalence of three "flexible" special cases
d.vr <- daisy(votes.repub)
a.wgt <- agnes(d.vr, method = "weighted")
a.sing <- agnes(d.vr, method = "single")
a.comp <- agnes(d.vr, method = "complete")
if (-8:7) # not using 'call' and 'method' for comparisons
stopifnot(
  all.equal(a.wgt[iC], agnes(d.vr, method="flexible", par.meth = 0.5)[iC]),
  all.equal(a.sing[iC], agnes(d.vr, method="flex", par.meth= c(.5,.5,0,-.5))[iC]),
  all.equal(a.comp[iC], agnes(d.vr, method="flex", par.meth= c(.5,.5,0,.5))[iC]))
## Exploring the dendrogram structure

```r
(d2 <- as.dendrogram(agn2)) # two main branches
d2[[1]] # the first branch
d2[[2]] # the 2nd one  ( 8 + 42 = 50 )
d2[[1]][[1]]# first sub-branch of branch 1 .. and shorter form
identical(d2[[c(1,1)]],
        d2[[1]][[1]])
```

## A "textual picture" of the dendrogram:

```r
str(d2)
```

```r
data( agriculture )
```

```r
## Plot similar to Figure 7 in ref
## Not run: plot(agnes( agriculture ), ask = TRUE)
```

```r
data( animals )
aa.a <- agnes( animals ) # default method = "average"
aa.ga <- agnes( animals, method = "gaverage" )
```

```r
op <- par(mfcol=1:2, mgp=c(1.5, 0.6, 0), mar=c(1+ c(4,3,2,1)),
        cex.main=0.8)
plot(aa.a, which.plot = 2)
plot(aa.ga, which.plot = 2)
par(op)
```

```r
## Show how "gaverage" is a "generalized average":
aa.ga.0 <- agnes( animals, method = "gaverage", par.method = 0 )
```

```r
stopifnot(all.equal(aa.ga.0[ic], aa.a[ic]))
```

---

### agnes.object

**Agglomerative Nesting (AGNES) Object**

#### Description

The objects of class "agnes" represent an agglomerative hierarchical clustering of a dataset.

#### Value

A legitimate `agnes` object is a list with the following components:

- **order**: a vector giving a permutation of the original observations to allow for plotting, in the sense that the branches of a clustering tree will not cross.
- **order.lab**: a vector similar to `order`, but containing observation labels instead of observation numbers. This component is only available if the original observations were labelled.
- **height**: a vector with the distances between merging clusters at the successive stages.
the agglomerative coefficient, measuring the clustering structure of the dataset. For each observation \(i\), denote by \(m(i)\) its dissimilarity to the first cluster it is merged with, divided by the dissimilarity of the merger in the final step of the algorithm. The ac is the average of all \(1 - m(i)\). It can also be seen as the average width (or the percentage filled) of the banner plot. Because ac grows with the number of observations, this measure should not be used to compare datasets of very different sizes.

merge
an (n-1) by 2 matrix, where n is the number of observations. Row \(i\) of merge describes the merging of clusters at step \(i\) of the clustering. If a number \(j\) in the row is negative, then the single observation \(|j|\) is merged at this stage. If \(j\) is positive, then the merger is with the cluster formed at stage \(j\) of the algorithm.

diss
an object of class "dissimilarity" (see dissimilarity.object), representing the total dissimilarity matrix of the dataset.

data
a matrix containing the original or standardized measurements, depending on the stand option of the function agnes. If a dissimilarity matrix was given as input structure, then this component is not available.

GENERATION
This class of objects is returned from agnes.

METHODS
The "agnes" class has methods for the following generic functions: print, summary, plot, and as.dendrogram.
In addition, cutree(\(x\), *) can be used to "cut" the dendrogram in order to produce cluster assignments.

INHERITANCE
The class "agnes" inherits from "twins". Therefore, the generic functions pltree and as.hclust are available for agnes objects. After applying as.hclust(), all its methods are available, of course.

See Also
agnes, diana, as.hclust, hclust, plot.agnes, twins.object.
cutree.

Examples
data( agriculture )
ag.ag <- agnes( agriculture )
class( ag.ag )
pltree( ag.ag ) # the dendrogram

## cut the dendrogram -> get cluster assignments:
(ck3 <- cutree( ag.ag, k = 3 ))
(ch6 <- cutree( as.hclust( ag.ag ), h = 6 ))
**agriculture**

stopifnot(identical(unname(ch6), ck3))

---

**European Union Agricultural Workforces**

**Description**

Gross National Product (GNP) per capita and percentage of the population working in agriculture for each country belonging to the European Union in 1993.

**Usage**

data(农业)

**Format**

A data frame with 12 observations on 2 variables:

- [ , 1] `x` numeric  per capita GNP
- [ , 2] `y` numeric  percentage in agriculture

The row names of the data frame indicate the countries.

**Details**

The data seem to show two clusters, the “more agricultural” one consisting of Greece, Portugal, Spain, and Ireland.

**Source**


**References**

see those in `agnes`.

**See Also**

`agnes`, `daisy`, `diana`.

**Examples**

data(农业)

## Compute the dissimilarities using Euclidean metric and without
## standardization
daisy(农业, metric = "euclidean", stand = FALSE)
Animals

## Attributes of Animals

### Description

This data set considers 6 binary attributes for 20 animals.

### Usage

```r
data(animals)
```

### Format

A data frame with 20 observations on 6 variables:

```r
[,1] war warm-blooded
[,2] fly can fly
[,3] ver vertebrate
[,4] end endangered
[,5] gro live in groups
[,6] hai have hair
```

All variables are encoded as 1 = ‘no’, 2 = ‘yes’.

### Details

This dataset is useful for illustrating monothetic (only a single variable is used for each split) hierarchical clustering.

### Source

References

see Struyf, Hubert & Rousseeuw (1996), in agnes.

Examples

data(animals)
apply(animals, 2, table) # simple overview

ma <- mona(animals)
ma
# Plot similar to Figure 10 in Struyf et al (1996)
plot(ma)

**bannerplot**  
*Plot Banner (of Hierarchical Clustering)*

Description

Draws a “banner”, i.e. basically a horizontal *barplot* visualizing the (agglomerative or divisive) hierarchical clustering or an other binary dendrogram structure.

Usage

```r
bannerplot(x, w = rev(x$height), fromLeft = TRUE,
           main = NULL, sub = NULL, xlab = "Height", adj = 0,
           col = c(2, 0), border = 0, axes = TRUE, frame.plot = axes,
           rev.xax = !fromLeft, xax.pretty = TRUE,
           labels = NULL, nmax.lab = 35, max.strlen = 5,
           yax.do = axes && length(x$order) <= nmax.lab,
           yaxRight = fromLeft, y.mar = 2.4 + max.strlen/2.5, ...)
```

Arguments

- **x**: a list with components `order`, `order.lab` and `height` when `w`, the next argument is not specified.
- **w**: non-negative numeric vector of bar widths.
- **fromLeft**: logical, indicating if the banner is from the left or not.
- **main, sub**: main and sub titles, see `title`.
- **xlab**: x axis label (with ‘correct’ default e.g. for `plot.agnes`).
- **adj**: passed to `title(main, sub)` for string adjustment.
- **col**: vector of length 2, for two horizontal segments.
- **border**: color for bar border; now defaults to background (no border).
- **axes**: logical indicating if axes (and labels) should be drawn at all.
- **frame.plot**: logical indicating the banner should be framed; mainly used when `border = 0` (as per default).
Note

This is mainly a utility called from \texttt{plot.agnes}, \texttt{plot.diana} and \texttt{plot.mona}.

Author(s)

Martin Maechler (from original code of Kaufman and Rousseeuw).

Examples

\begin{verbatim}
data(agriculture)
bannerplot(agnes(agriculture), main = "Bannerplot")
\end{verbatim}

\hspace{1cm}

\begin{verbatim}
chorSub Subset of C-horizon of Kola Data
\end{verbatim}

Description

This is a small rounded subset of the C-horizon data \texttt{chorizon} from package \texttt{mvoutlier}.

Usage

\begin{verbatim}
data(chorSub)
\end{verbatim}

Format

A data frame with 61 observations on 10 variables. The variables contain scaled concentrations of chemical elements.
**Details**

This data set was produced from `chorizon` via these statements:

```r
data(chorizon, package = "mvoutlier")
chorSub <- round(100*scale(chorizon[,101:110])[,190:250],]
storage.mode(chorSub) <- "integer"
colnames(chorSub) <- gsub("_.*", ", colnames(chorSub))
```

**Source**


**See Also**

`chorizon` in package `mvoutlier` and other Kola data in the same package.

**Examples**

```r
data(chorSub)
summary(chorSub)
pairs(chorSub, gap= .1)# some outliers
```

---

**clara**

*Clustering Large Applications*

**Description**

Computes a "clara" object, a list representing a clustering of the data into k clusters.

**Usage**

```r
clara(x, k, metric = c("euclidean", "manhattan", "jaccard"),
stand = FALSE, samples = 5,
sampsize = min(n, 40 + 2 * k), trace = 0, medoids.x = TRUE,
keep.data = medoids.x, rngR = FALSE, pamLike = FALSE, correct.d = TRUE)
```

**Arguments**

- **x**
  - data matrix or data frame, each row corresponds to an observation, and each column corresponds to a variable. All variables must be numeric. Missing values (NAs) are allowed.

- **k**
  - integer, the number of clusters. It is required that $0 < k < n$ where $n$ is the number of observations (i.e., `n = nrow(x)`).
clara

metric

character string specifying the metric to be used for calculating dissimilarities between observations. The currently available options are "euclidean", "manhattan", and "jaccard". Euclidean distances are root sum-of-squares of differences, and manhattan distances are the sum of absolute differences.

stand

logical, indicating if the measurements in x are standardized before calculating the dissimilarities. Measurements are standardized for each variable (column), by subtracting the variable's mean value and dividing by the variable's mean absolute deviation.

samples

integer, say $N$, the number of samples to be drawn from the dataset. The default, $N = 5$, is rather small for historical (and now back compatibility) reasons and we recommend to set samples an order of magnitude larger.

sampsize

integer, say $j$, the number of observations in each sample. sampsize should be higher than the number of clusters ($k$) and at most the number of observations ($n = nrow(x)$). While computational effort is proportional to $j^2$, see note below, it may still be advisable to set $j = sampsize$ to a larger value than the (historical) default.

trace

integer indicating a trace level for diagnostic output during the algorithm.

medoids.x

logical indicating if the medoids should be returned, identically to some rows of the input data x. If FALSE, keep.data must be false as well, and the medoid indices, i.e., row numbers of the medoids will still be returned (i.med component), and the algorithm saves space by needing one copy less of x.

keep.data

logical indicating if the (scaled if stand is true) data should be kept in the result. Setting this to FALSE saves memory (and hence time), but disables clusplot()ing of the result. Use medoids.x = FALSE to save even more memory.

rngR

logical indicating if R's random number generator should be used instead of the primitive clara()-builtin one. If true, this also means that each call to clara() returns a different result – though only slightly different in good situations.

pamLike

logical indicating if the "swap" phase (see pam, in C code) should use the same algorithm as pam(). Note that from Kaufman and Rousseeuw's description this should have been true always, but as the original Fortran code and the subsequent port to C has always contained a small one-letter change (a typo according to Martin Maechler) with respect to Pam, the default, pamLike = FALSE has been chosen to remain back compatible rather than "PAM compatible".

correct.d

logical or integer indicating that—only in the case of NAs present in x—the correct distance computation should be used instead of the wrong formula which has been present in the original Fortran code and been in use up to early 2016. Because the new correct formula is not back compatible, for the time being, a warning is signalled in this case, unless the user explicitly specifies correct.d.

Details

clara is fully described in chapter 3 of Kaufman and Rousseeuw (1990). Compared to other partitioning methods such as pam, it can deal with much larger datasets. Internally, this is achieved by considering sub-datasets of fixed size (sampsize) such that the time and storage requirements become linear in $n$ rather than quadratic.
Each sub-dataset is partitioned into $k$ clusters using the same algorithm as in pam. Once $k$ representative objects have been selected from the sub-dataset, each observation of the entire dataset is assigned to the nearest medoid.

The mean (equivalent to the sum) of the dissimilarities of the observations to their closest medoid is used as a measure of the quality of the clustering. The sub-dataset for which the mean (or sum) is minimal, is retained. A further analysis is carried out on the final partition.

Each sub-dataset is forced to contain the medoids obtained from the best sub-dataset until then. Randomly drawn observations are added to this set until sampsize has been reached.

**Value**

an object of class "clara" representing the clustering. See clara.object for details.

**Note**

By default, the random sampling is implemented with a **very simple** scheme (with period $2^{16} = 65536$) inside the Fortran code, independently of R’s random number generation, and as a matter of fact, deterministically. Alternatively, we recommend setting `rngR = TRUE` which uses R’s random number generators. Then, clara() results are made reproducible typically by using `set.seed()` before calling clara.

The storage requirement of clara computation (for small $k$) is about $O(n \times p) + O(j^2)$ where $j = \text{sampsize}$, and $(n, p) = \text{dim}(x)$. The CPU computing time (again assuming small $k$) is about $O(n \times p \times j^2 \times N)$, where $N = \text{samples}$.

For “small” datasets, the function pam can be used directly. What can be considered small, is really a function of available computing power, both memory (RAM) and speed. Originally (1990), “small” meant less than 100 observations; in 1997, the authors said “small (say with fewer than 200 observations)”; as of 2006, you can use pam with several thousand observations.

**Author(s)**

Kaufman and Rousseeuw (see agnes), originally. Metric "jaccard": Kamil Kozlowski and Kamil Jadeszko (from ownedoutcomes.com). All arguments from trace on, and most R documentation and all tests by Martin Maechler.

**See Also**

agne for background and references; clara.object, pam, partition.object, plot.partition.

**Examples**

```r
# generate 500 objects, divided into 2 clusters.
x <- rbind(cbind(rnorm(200,0,8), rnorm(200,0,8)),
           cbind(rnorm(300,50,8), rnorm(300,50,8)))
clarax <- clara(x, 2, samples=50)
clarax
clarax$clusinfo
# using pamLike=TRUE gives the same (apart from the 'call'):
all.equal(clarax[-8],
          clara(x, 2, samples=50, pamLike = TRUE)[-8])
```
plot(clara(x))

## 'xclara' is an artificial data set with 3 clusters of 1000 bivariate
## objects each.
data(xclara)
(clx3 <- clara(xclara, 3))
## "better" number of samples
cl.3 <- clara(xclara, 3, samples=100)
## but that did not change the result here:
stopifnot(cl.3$clustering == clx3$clustering)
## Plot similar to Figure 5 in Struyf et al (1996)
## Not run: plot(clx3, ask = TRUE)

## Try 100 times *different* random samples -- for reliability:
nSim <- 100
nCl <- 3 # = no.classes
set.seed(421)# (reproducibility)
cl <- matrix(NA,nrow(xclara), nSim)
for(i in 1:nSim)
  cl[,i] <- clara(xclara, nCl, medoids.x = FALSE, rngR = TRUE)$cluster
tcl <- apply(cl,1, tabulate, nbins = nCl)
## those that are not always in same cluster (5 out of 3000 for this seed):
(iDoubt <- which(apply(tcl,2, function(n) all(n < nSim))))
if(length(iDoubt)) { # (not for all seeds)
tabD <- tcl[,iDoubt, drop=FALSE]
dimnames(tabD) <- list(cluster = paste(1:nCl), obs = format(iDoubt))
t(tabD) # how many times in which clusters
}

### clara.object

**Clustering Large Applications (CLARA) Object**

**Description**

The objects of class "clara" represent a partitioning of a large dataset into clusters and are typically returned from clara.

**Value**

A legitimate clara object is a list with the following components:

- **sample** labels or case numbers of the observations in the best sample, that is, the sample used by the clara algorithm for the final partition.
- **medoids** the medoids or representative objects of the clusters. It is a matrix with in each row the coordinates of one medoid. Possibly NULL, namely when the object resulted from clara(*, medoids.x=FALSE). Use the following i.med in that case.
clusGap

i.med

the *indices* of the medoids above: medoids <- x[i.med,] where x is the
original data matrix in *clara*(x,*)

clustering

the clustering vector, see *partition.object*.

objective

the objective function for the final clustering of the entire dataset.

clusinfo

matrix, each row gives numerical information for one cluster. These are the
cardinality of the cluster (number of observations), the maximal and average
dissimilarity between the observations in the cluster and the cluster’s medoid.
The last column is the maximal dissimilarity between the observations in the
cluster and the cluster’s medoid, divided by the minimal dissimilarity between
the cluster’s medoid and the medoid of any other cluster. If this ratio is small,
the cluster is well-separated from the other clusters.

diss

dissimilarity (maybe NULL), see *partition.object*.

silinfo

list with silhouette width information for the best sample, see *partition.object*.

call

generating call, see *partition.object*.

data

matrix, possibly standardized, or NULL, see *partition.object*.

Methods, Inheritance

The "clara" class has methods for the following generic functions: print, summary.
The class "clara" inherits from "partition". Therefore, the generic functions plot and clusplot
can be used on a clara object.

See Also

*clara, dissimilarity.object, partition.object, plot.partition*.

---

clusGap

*Gap Statistic for Estimating the Number of Clusters*

Description

clusGap() calculates a goodness of clustering measure, the “gap” statistic. For each number of
clusters *k*, it compares \( \log(W(k)) \) with \( E^\ast[\log(W(k))] \) where the latter is defined via bootstrapping,
i.e., simulating from a reference \((H_0)\) distribution, a uniform distribution on the hypercube
determined by the ranges of *x*, after first centering, and then *svd* (aka ‘PCA’)-rotating them when
(as by default) *spaceH0* = "scaledPCA".

maxSE(f, SE.f) determines the location of the maximum of *f*, taking a “1-SE rule” into account
for the *SE* methods. The default method "firstSEmax" looks for the smallest *k* such that its
value *f(k)* is not more than 1 standard error away from the first local maximum. This is similar but
not the same as "Tibs2001SEmax", Tibshirani et al’s recommendation of determining the number
of clusters from the gap statistics and their standard deviations.
Usage

clusGap(x, FUNcluster, K.max, B = 100, d.power = 1,
        spaceH0 = c("scaledPCA", "original"),
        verbose = interactive(), ...)

maxSE(f, SE.f,
     method = c("firstSEmax", "Tibs2001SEmax", "globalSEmax",
                "firstmax", "globalmax"),
     SE.factor = 1)

## S3 method for class 'clusGap'
print(x, method = "firstSEmax", SE.factor = 1, ...)

## S3 method for class 'clusGap'
plot(x, type = "b", xlab = "k", ylab = expression(Gap[k]),
     main = NULL, do.arrows = TRUE,
     arrowArgs = list(col="red3", length=1/16, angle=90, code=3), ...)

Arguments

x numeric matrix or data.frame.

FUNcluster a function which accepts as first argument a (data) matrix like x, second argument, say k, k \geq 2, the number of clusters desired, and returns a list with a component named (or shortened to) cluster which is a vector of length n = nrow(x) of integers in 1:k determining the clustering or grouping of the n observations.

K.max the maximum number of clusters to consider, must be at least two.

B integer, number of Monte Carlo ("bootstrap") samples.

d.power a positive integer specifying the power p which is applied to the euclidean distances (dist) before they are summed up to give W(k). The default, d.power = 1, corresponds to the "historical" R implementation, whereas d.power = 2 corresponds to what Tibshirani et al had proposed. This was found by Juan Gonzalez, in 2016-02.

spaceH0 a character string specifying the space of the $H_0$ distribution (of no cluster). Both "scaledPCA" and "original" use a uniform distribution in a hyper cube and had been mentioned in the reference; "original" been added after a proposal (including code) by Juan Gonzalez.

verbose integer or logical, determining if “progress” output should be printed. The default prints one bit per bootstrap sample.

... (for clusGap()) optionally further arguments for FUNcluster(), see kmeans example below.

f numeric vector of 'function values', of length K, whose ("1 SE respected") maximum we want.

SE.f numeric vector of length K of standard errors of f.
clusGap

method

character string indicating how the “optimal” number of clusters, \( \hat{k} \), is computed from the gap statistics (and their standard deviations), or more generally how the location \( k \) of the maximum of \( f_k \) should be determined.

“globalmax”: simply corresponds to the global maximum, i.e., is which \( \max(f) \)

“firstmax”: gives the location of the first local maximum.

“Tibs2001SEmax”: uses the criterion, Tibshirani et al (2001) proposed: “the smallest \( k \) such that \( f(k) \geq f(k+1) - s_{k+1} \). Note that this chooses \( k = 1 \) when all standard deviations are larger than the differences \( f(k+1) - f(k) \).

“firstSEmax”: location of the first \( f() \) value which is not smaller than the first local maximum minus \( \text{SE.factor} \times \text{SE.f}[] \), i.e, within an “f S.E.” range of that maximum (see also \( \text{SE.factor} \).

This, the default, has been proposed by Martin Maechler in 2012, when adding \( \text{clusGap()} \) to the \( \text{cluster} \) package, after having seen the “globalSEmax” proposal (in code) and read the "Tibs2001SEmax” proposal.

“globalSEmax”: (used in Dudoit and Fridlyand (2002), supposedly following Tibshirani’s proposition): location of the first \( f() \) value which is not smaller than the global maximum minus \( \text{SE.factor} \times \text{SE.f}[] \), i.e, within an “f S.E.” range of that maximum (see also \( \text{SE.factor} \).

See the examples for a comparison in a simple case.

SE.factor

[When method contains “SE”] Determining the optimal number of clusters, Tibshirani et al. proposed the “1 S.E.”-rule. Using an \( \text{SE.factor} \) \( f \), the “f S.E.”-rule is used, more generally.

type, xlab, ylab, main

arguments with the same meaning as in \( \text{plot.default}() \), with different default.

do.arrows

logical indicating if (1 SE-)“error bars” should be drawn, via \( \text{arrows}() \).

arrowArgs

a list of arguments passed to \( \text{arrows}() \); the default, notably \( \text{angle} \) and \( \text{code} \), provide a style matching usual error bars.

Details

The main result \(<\text{res}>\text{Tab}[, \text{"gap"] of course is from bootstrapping aka Monte Carlo simulation and hence random, or equivalently, depending on the initial random seed (see \( \text{set.seed}() \)). On the other hand, in our experience, using \( B = 500 \) gives quite precise results such that the gap plot is basically unchanged after another run.

Value

\( \text{clusGap(\ldots)} \) returns an object of S3 class “clusGap”, basically a list with components

Tab

a matrix with \( \text{K.max} \) rows and 4 columns, named "logW", "E.logW", "gap", and "SE.sim", where \( \text{gap} = \text{E.logW} - \text{logW} \), and \( \text{SE.sim} \) corresponds to the standard error of gap, \( \text{SE.sim}[k]=s_k \), where \( s_k := \sqrt{1 + \frac{1}{k} \text{sd}^2(\text{gap}_j)} \), and \( \text{sd}^2() \) is the standard deviation of the simulated (“bootstrapped”) gap values.

call

the \( \text{clusGap(\ldots)} \) \( \text{call} \).

spaceH0

the \( \text{spaceH0} \) argument (\( \text{match.arg}() \)ed).

n

number of observations, i.e., \( \text{nrow(x)} \).
clusGap

B input B
FUNcluster input function FUNcluster

Author(s)

This function is originally based on the functions gap of (Bioconductor) package SAGx by Per Broberg, gapStat() from former package SLmisc by Matthias Kohl and ideas from gap() and its methods of package lga by Justin Harrington.

The current implementation is by Martin Maechler.

The implementation of spaceH0 = "original" is based on code proposed by Juan Gonzalez.

References


See Also

silhouette for a much simpler less sophisticated goodness of clustering measure.
cluster.stats() in package fpc for alternative measures.

Examples

```r
### --- maxSE() methods -----------------------------------------------
(mets <- eval(formals(maxSE)$method))
fk <- c(2,3,5,4,7,8,5,4)
sk <- c(1,1,2,1,1,3,1,1)/2
## use plot.clusGap():
plot(structure(class="clusGap", list(Tab = cbind(gap=fk, SE.sim=sk)))))
## Note that 'firstmax' and 'globalmax' are always at 3 and 6:
sapply(c(1,4, 1,2,4), function(SEF)
  sapply(mets, function(M) maxSE(fk, sk, method = M, SE.factor = SEF)))

### --- clusGap() ----------------------------------------------------
## ridiculously nicely separated clusters in 3 D:
x <- rbind(matrix(rnorm(150, sd = 0.1), ncol = 3),
  matrix(rnorm(150, mean = 1, sd = 0.1), ncol = 3),
  matrix(rnorm(150, mean = 2, sd = 0.1), ncol = 3),
  matrix(rnorm(150, mean = 3, sd = 0.1), ncol = 3))
## Slightly faster way to use pam (see below)
pam1 <- function(x,k) list(cluster = pam(x,k, cluster.only=TRUE))
```
**clusplot**

Bivariate Cluster Plot (of a Partitioning Object)

```r
## We do not recommend using hier.clustering here, but if you want,
## there is factoextra::hcut () or a cheap version of it
hclusCut <- function(x, k, d.meth = "euclidean", ...)
    list(cluster = cutree(hclust(dist(x, method=d.meth), ...), k=k))

## You can manually set it before running this:
doExtras <- TRUE  # or FALSE
if(!exists("doExtras") & & is.logical(doExtras))
doExtras <- cluster:::doExtras()

if(doExtras) {
    ## Note we use B = 60 in the following examples to keep them "speedy".
    ## ---- rather keep the default B = 500 for your analysis!
    ## note we can pass 'nstart = 20' to kmeans:
gskmm <- clusGap(x, FUN = kmeans, nstart = 20, K.max = 8, B = 60)
gskmm #-> its print() method
plot(gskmm, main = "clusGap(. , FUN = kmeans, n.start=20, B= 60)")
set.seed(12); system.time(
    gsPam0 <- clusGap(x, FUN = pam, K.max = 8, B = 60)
)
set.seed(12); system.time(
    gsPam1 <- clusGap(x, FUN = pam1, K.max = 8, B = 60)
)

## and show that it gives the "same":
not.eq <- c("call", "FUNcluster"); n <- names(gsPam0)
    eq <- n[[n %in% not.eq]]
stopifnot(identical(gsPam1[eq], gsPam0[eq]))
print(gsPam0, method="globalSEmax")
print(gsPam1, method="globalmax")

print(gsHc <- clusGap(x, FUN = hclusCut, K.max = 8, B = 60))
}
```

```r
gs.pam.RU <- clusGap(ruspini, FUN = pam1, K.max = 8, B = 60)
gs.pam.RU
plot(gs.pam.RU, main = "Gap statistic for the 'ruspini' data")
mtext("k = 4 is best .. and k = 5 pretty close")

## This takes a minute..
## No clustering => k = 1 ("one cluster") should be optimal:
Z <- matrix(rnorm(256*3), 256,3)
gsP.Z <- clusGap(Z, FUN = pam1, K.max = 8, B = 200)
plot(gsP.Z, main = "clusGap(<iid_rnorm_p=3>) => k = 1 cluster is optimal")
gsP.Z
```
Description

Draws a 2-dimensional “clusplot” (clustering plot) on the current graphics device. The generic function has a default and a partition method.

Usage

clusplot(x, ...)

## S3 method for class 'partition'
clusplot(x, main = NULL, dist = NULL, ...)

Arguments

x an \( \mathbb{R} \) object, here, specifically an object of class "partition", e.g. created by one of the functions \texttt{pam}, \texttt{clara}, or \texttt{fanny}.

main title for the plot; when \texttt{NULL} (by default), a title is constructed, using \texttt{x$call}.

dist when \texttt{x} does not have a \texttt{diss} nor a data component, e.g., for \texttt{pam(dist(*), keep.diss=FALSE)}, \texttt{dist} must specify the dissimilarity for the clusplot.

... optional arguments passed to methods, notably the \texttt{clusplot.default} method (except for the \texttt{diss} one) may also be supplied to this function. Many graphical parameters (see \texttt{par}) may also be supplied as arguments here.

Details

The \texttt{clusplot.partition()} method relies on \texttt{clusplot.default}.

If the clustering algorithms \texttt{pam}, \texttt{fanny} and \texttt{clara} are applied to a data matrix of observations-by-variables then a clusplot of the resulting clustering can always be drawn. When the data matrix contains missing values and the clustering is performed with \texttt{pam} or \texttt{fanny}, the dissimilarity matrix will be given as input to \texttt{clusplot}. When the clustering algorithm \texttt{clara} was applied to a data matrix with NAs then \texttt{clusplot} will replace the missing values as described in \texttt{clusplot.default}, because a dissimilarity matrix is not available.

Value

For the partition (and default) method: An invisible list with components Distances and Shading, as for \texttt{clusplot.default}, see there.

Side Effects

a 2-dimensional clusplot is created on the current graphics device.

See Also

\texttt{clusplot.default} for references; \texttt{partition.object}, \texttt{pam}, \texttt{pam.object}, \texttt{clara}, \texttt{clara.object}, \texttt{fanny}, \texttt{fanny.object}, \texttt{par}. 
**Examples**

```r
## For more, see ?clusplot.default

## generate 25 objects, divided into 2 clusters.
x <- rbind(cbind(rnorm(10,0,0.5), rnorm(10,0,0.5)),
           cbind(rnorm(15,5,0.5), rnorm(15,5,0.5)))
clusplot(pam(x, 2))
## add noise, and try again:
x4 <- cbind(x, rnorm(25), rnorm(25))
clusplot(pam(x4, 2))
```

---

### clusplot.default

**Bivariate Cluster Plot (clusplot) Default Method**

**Description**

Creates a bivariate plot visualizing a partition (clustering) of the data. All observations are represented by points in the plot, using principal components or multidimensional scaling. Around each cluster an ellipse is drawn.

**Usage**

```r
## Default S3 method:
clusplot(x, clus, diss = FALSE,
         s.x.2d = mkCheckX(x, diss), stand = FALSE,
         lines = 2, shade = FALSE, color = FALSE,
         labels = 0, plotchar = TRUE,
         col.p = "dark green", col.txt = col.p,
         col.clus = if(color) c(2, 4, 6, 3) else 5, cex = 1, cex.txt = cex,
         span = TRUE,
         add = FALSE,
         xlim = NULL, ylim = NULL,
         main = paste("CLUSPLOT(", sub = paste("These two components explain",
         round(100 * var.dec, digits = 2), ", "%, of the point variability.")",
         xlab = "Component 1", ylab = "Component 2",
         verbose = getOption("verbose"),
         ...
```

**Arguments**

- **x**
  - matrix or data frame, or dissimilarity matrix, depending on the value of the `diss` argument.
  - In case of a matrix (alike), each row corresponds to an observation, and each column corresponds to a variable. All variables must be numeric. Missing values (NA) are allowed. They are replaced by the median of the corresponding variable. When some variables or some observations contain only missing values, the function stops with a warning message.
In case of a dissimilarity matrix, \( x \) is the output of \texttt{daisy} or \texttt{dist} or a symmetric matrix. Also, a vector of length \( n \times (n - 1)/2 \) is allowed (where \( n \) is the number of observations), and will be interpreted in the same way as the output of the above-mentioned functions. Missing values (NAs) are not allowed.

**clus**
- a vector of length \( n \) representing a clustering of \( x \). For each observation the vector lists the number or name of the cluster to which it has been assigned.
- \texttt{clus} is often the clustering component of the output of \texttt{pam}, \texttt{fanny} or \texttt{clara}.

**diss**
- logical indicating if \( x \) will be considered as a dissimilarity matrix or a matrix of observations by variables (see \texttt{x} argument above).

**s.x.2d**
- a \texttt{list} with components named \( x \) (a \( n \times 2 \) matrix; typically something like principal components of original data), \texttt{labs} and \texttt{var.dec}.

**stand**
- logical flag: if true, then the representations of the \( n \) observations in the 2-dimensional plot are standardized.

**lines**
- integer out of 0, 1, 2, used to obtain an idea of the distances between ellipses. The distance between two ellipses \( E_1 \) and \( E_2 \) is measured along the line connecting the centers \( m_1 \) and \( m_2 \) of the two ellipses.
- In case \( E_1 \) and \( E_2 \) overlap on the line through \( m_1 \) and \( m_2 \), no line is drawn.
- Otherwise, the result depends on the value of \texttt{lines}: If
  - \texttt{lines = 0}, no distance lines will appear on the plot;
  - \texttt{lines = 1}, the line segment between \( m_1 \) and \( m_2 \) is drawn;
  - \texttt{lines = 2}, a line segment between the boundaries of \( E_1 \) and \( E_2 \) is drawn (along the line connecting \( m_1 \) and \( m_2 \)).

**shade**
- logical flag: if TRUE, then the ellipses are shaded in relation to their density.
- The density is the number of points in the cluster divided by the area of the ellipse.

**color**
- logical flag: if TRUE, then the ellipses are colored with respect to their density.
- With increasing density, the colors are light blue, light green, red and purple.
- To see these colors on the graphics device, an appropriate color scheme should be selected (we recommend a white background).

**labels**
- integer code, currently one of 0, 1, 2, 3, 4 and 5. If
  - \texttt{labels= 0}, no labels are placed in the plot;
  - \texttt{labels= 1}, points and ellipses can be identified in the plot (see \texttt{identify});
  - \texttt{labels= 2}, all points and ellipses are labelled in the plot;
  - \texttt{labels= 3}, only the points are labelled in the plot;
  - \texttt{labels= 4}, only the ellipses are labelled in the plot;
  - \texttt{labels= 5}, the ellipses are labelled in the plot, and points can be identified.

- The levels of the vector \texttt{clus} are taken as labels for the clusters. The labels of the points are the rownames of \( x \) if \( x \) is matrix like. Otherwise (\texttt{diss = TRUE}), \( x \) is a vector, point labels can be attached to \( x \) as a "Labels" attribute (\texttt{attr(x, "Labels")}), as is done for the output of \texttt{daisy}.
- A possible \texttt{names} attribute of \texttt{clus} will not be taken into account.

**plotchar**
- logical flag: if TRUE, then the plotting symbols differ for points belonging to different clusters.
**clusplot.default**

- **span**
  - logical flag: if TRUE, then each cluster is represented by the ellipse with smallest area containing all its points. (This is a special case of the minimum volume ellipsoid.)
  - If FALSE, the ellipse is based on the mean and covariance matrix of the same points. While this is faster to compute, it often yields a much larger ellipse.
  - There are also some special cases: When a cluster consists of only one point, a tiny circle is drawn around it. When the points of a cluster fall on a straight line, span=FALSE draws a narrow ellipse around it and span=TRUE gives the exact line segment.

- **add**
  - logical indicating if ellipses (and labels if labels is true) should be added to an already existing plot. If false, neither a title or sub title, see sub, is written.

- **col.p**
  - color code(s) used for the observation points.

- **col.txt**
  - color code(s) used for the labels (if labels >= 2).

- **col.clus**
  - color code for the ellipses (and their labels); only one if color is false (as per default).

- **cex, cex.txt**
  - character expansion (size), for the point symbols and point labels, respectively.

- **xlim, ylim**
  - numeric vectors of length 2, giving the x- and y- ranges as in plot.default.

- **main**
  - main title for the plot; by default, one is constructed.

- **sub**
  - sub title for the plot; by default, one is constructed.

- **xlab, ylab**
  - x- and y- axis labels for the plot, with defaults.

- **verbose**
  - a logical indicating, if there should be extra diagnostic output; mainly for ‘de-bugging’.

... Further graphical parameters may also be supplied, see par.

**Details**

clusplot uses function calls princomp(*, cor = (ncol(x) > 2)) or cmdscale(*, add=TRUE), respectively, depending on diss being false or true. These functions are data reduction techniques to represent the data in a bivariate plot.

Ellipses are then drawn to indicate the clusters. The further layout of the plot is determined by the optional arguments.

**Value**

An invisible list with components:

- **Distances**
  - When lines is 1 or 2 we obtain a k by k matrix (k is the number of clusters).
  - The element in [i, j] is the distance between ellipse i and ellipse j.
  - If lines = 0, then the value of this component is NA.

- **Shading**
  - A vector of length k (where k is the number of clusters), containing the amount of shading per cluster.
  - Let y be a vector where element i is the ratio between the number of points in cluster i and the area of ellipse i. When the cluster i is a line segment, y[i] and the density of the cluster are set to NA. Let z be the sum of all the elements of y without the NAs. Then we put shading = y/z * 37 + 3.
Side Effects

A visual display of the clustering is plotted on the current graphics device.

Note

When we have 4 or fewer clusters, then the `color = TRUE` gives every cluster a different color. When there are more than 4 clusters, `clusplot` uses the function `pam` to cluster the densities into 4 groups such that ellipses with nearly the same density get the same color. `col.clus` specifies the colors used.

The `col.p` and `col.txt` arguments, added for R, are recycled to have length the number of observations. If `col.p` has more than one value, using `color = TRUE` can be confusing because of a mix of point and ellipse colors.

References


See Also

`princomp`, `cmdscale`, `pam`, `clara`, `daisy`, `par`, `identify`, `cov.mve`, `clusplot.partition`.

Examples

```R
# plotting votes.diss(dissimilarity) in a bivariate plot and
# partitioning into 2 clusters
data(votes.repub)
votes.diss <- daisy(votes.repub)
pamv <- pam(votes.diss, 2, diss = TRUE)
clusplot(pamv, shade = TRUE)
## is the same as
votes.clus <- pamv$clustering
clusplot(votes.diss, votes.clus, diss = TRUE, shade = TRUE)
## Now look at components 3 and 2 instead of 1 and 2:
str(chmds <- cmdscale(votes.diss, k=3, add=TRUE))
clusplot(pamv, s.x.2d = list(x=chmds$points[, c(3,2)],
                            labs=rownames(votes.repub), var.dec=NA),
                            shade = TRUE, col.p = votes.clus,
                            sub="", xlab = "Component 3", ylab = "Component 2")
clusplot(pamv, col.p = votes.clus, labels = 4)# color points and label ellipses
# "simple" cheap ellipses: larger than minimum volume:
# here they are *added* to the previous plot:
clusplot(pamv, span = FALSE, add = TRUE, col.clus = "midnightblue")
```
## coef.hclust

**Agglomerative / Divisive Coefficient for 'hclust' Objects**

### Description

Computes the “agglomerative coefficient” (aka “divisive coefficient” for `diana`), measuring the clustering structure of the dataset.

For each observation i, denote by \( m(i) \) its dissimilarity to the first cluster it is merged with, divided by the dissimilarity of the merger in the final step of the algorithm. The agglomerative coefficient is the average of all \( 1 - m(i) \). It can also be seen as the average width (or the percentage filled) of the banner plot.

`coefHier()` directly interfaces to the underlying C code, and “proves” that only `object$heights` is needed to compute the coefficient.

Because it grows with the number of observations, this measure should not be used to compare datasets of very different sizes.

### Usage

```r
coefHier(object)
coef.hclust(object, ...)
```
## S3 method for class 'hclust'
coef(object, ...)

## S3 method for class 'twins'
coef(object, ...)

### Arguments

- **object**: An object of class "hclust" or "twins", i.e., typically the result of `hclust()`, `agnes()`, or `diana()`.

  Since `coef.hclust` only uses `object$heights` and `object$merge`, `object` can be any list-like object with appropriate merge and heights components. For `coefHier`, even only `object$heights` is needed.

- **...**: Currently unused potential further arguments

### Value

A number specifying the agglomerative (or divisive for `diana` objects) coefficient as defined by Kaufman and Rousseeuw, see `agnes.object$ac` or `diana.object$dc`.

### Examples

```r
data(agriculture)
aa <- agnes(agriculture)
coef(aa) # really just extracts aa$ac
coef(as.hclust(aa))# recomputes
coefHier(aa)  # ditto
```

## daisy

### Description

Compute all the pairwise dissimilarities (distances) between observations in the data set. The original variables may be of mixed types. In that case, or whenever `metric = "gower"` is set, a generalization of Gower’s formula is used, see ‘Details’ below.

### Usage

```r
daisy(x, metric = c("euclidean", "manhattan", "gower"),
  stand = FALSE, type = list(), weights = rep.int(1, p),
  warnBin = warnType, warnAsym = warnType, warnConst = warnType,
  warnType = TRUE)
```
Arguments

x numeric matrix or data frame, of dimension \(n \times p\), say. Dissimilarities will be computed between the rows of \(x\). Columns of mode numeric (i.e. all columns when \(x\) is a matrix) will be recognized as interval scaled variables, columns of class factor will be recognized as nominal variables, and columns of class ordered will be recognized as ordinal variables. Other variable types should be specified with the \texttt{type} argument. Missing values (\texttt{NAs}) are allowed.

metric character string specifying the metric to be used. The currently available options are "euclidean" (the default), "manhattan" and "gower". Euclidean distances are root sum-of-squares of differences, and manhattan distances are the sum of absolute differences.

"Gower's distance" is chosen by metric "gower" or automatically if some columns of \(x\) are not numeric. Also known as Gower's coefficient (1971), expressed as a dissimilarity, this implies that a particular standardisation will be applied to each variable, and the “distance” between two units is the sum of all the variable-specific distances, see the details section.

stand logical flag: if TRUE, then the measurements in \(x\) are standardized before calculating the dissimilarities. Measurements are standardized for each variable (column), by subtracting the variable’s mean value and dividing by the variable’s mean absolute deviation.

If not all columns of \(x\) are numeric, \texttt{stand} will be ignored and Gower’s standardization (based on the range) will be applied in any case, see argument \texttt{metric}, above, and the details section.

type list for specifying some (or all) of the types of the variables (columns) in \(x\). The list may contain the following components: "ordratio" (ratio scaled variables to be treated as ordinal variables), "logratio" (ratio scaled variables that must be logarithmically transformed), "asymm" (asymmetric binary) and "symm" (symmetric binary variables). Each component’s value is a vector, containing the names or the numbers of the corresponding columns of \(x\). Variables not mentioned in the \texttt{type} list are interpreted as usual (see argument \texttt{x}).

weights an optional numeric vector of length \(p(=\text{ncol}(x))\); to be used in “case 2” (mixed variables, or \texttt{metric = "gower"}), specifying a weight for each variable (\(x[,k]\)) instead of 1 in Gower’s original formula.

\texttt{warnBin, warnAsym, warnConst} logicals indicating if the corresponding type checking warnings should be signalled (when found).

\texttt{warnType} logical indicating if all the type checking warnings should be active or not.

Details

The original version of daisy is fully described in chapter 1 of Kaufman and Rousseeuw (1990). Compared to \texttt{dist} whose input must be numeric variables, the main feature of daisy is its ability to handle other variable types as well (e.g. nominal, ordinal, (a)symmetric binary) even when different types occur in the same data set.

The handling of nominal, ordinal, and (a)symmetric binary data is achieved by using the general dissimilarity coefficient of Gower (1971). If \(x\) contains any columns of these data-types, both
arguments metric and stand will be ignored and Gower's coefficient will be used as the metric. This can also be activated for purely numeric data by `metric = "gower"`. With that, each variable (column) is first standardized by dividing each entry by the range of the corresponding variable, after subtracting the minimum value; consequently the rescaled variable has range $[0, 1]$, exactly.

Note that setting the type to `symm` (symmetric binary) gives the same dissimilarities as using `nominal` (which is chosen for non-ordered factors) only when no missing values are present, and more efficiently.

Note that `daisy` signals a warning when 2-valued numerical variables do not have an explicit type specified, because the reference authors recommend to consider using `asymm`; the warning may be silenced by `warnBin = FALSE`.

In the `daisy` algorithm, missing values in a row of x are not included in the dissimilarities involving that row. There are two main cases,

1. If all variables are interval scaled (and `metric` is not "gower"), the metric is "euclidean", and $n_g$ is the number of columns in which neither row i and j have NAs, then the dissimilarity $d(i,j)$ returned is $\sqrt{\frac{p}{n_g}} \times$ the Euclidean distance between the two vectors of length $n_g$ shortened to exclude NAs. The rule is similar for the "manhattan" metric, except that the coefficient is $\frac{p}{n_g}$. If $n_g = 0$, the dissimilarity is NA.

2. When some variables have a type other than interval scaled, or if `metric = "gower"` is specified, the dissimilarity between two rows is the weighted mean of the contributions of each variable. Specifically,

$$d_{ij} = \frac{\sum_{k=1}^{p} w_k \delta_{ij}^{(k)} d_{ij}^{(k)}}{\sum_{k=1}^{p} w_k \delta_{ij}^{(k)}}.$$

In other words, $d_{ij}$ is a weighted mean of $d_{ij}^{(k)}$ with weights $w_k \delta_{ij}^{(k)}$, where $w_k = \text{weights}[k]$. $\delta_{ij}^{(k)}$ is 0 or 1, and $d_{ij}^{(k)}$, the k-th variable contribution to the total distance, is a distance between $x[i,k]$ and $x[j,k]$, see below.

The 0-1 weight $\delta_{ij}^{(k)}$ becomes zero when the variable $x[,k]$ is missing in either or both rows (i and j), or when the variable is asymmetric binary and both values are zero. In all other situations it is 1.

The contribution $d_{ij}^{(k)}$ of a nominal or binary variable to the total dissimilarity is 0 if both values are equal, 1 otherwise. The contribution of other variables is the absolute difference of both values, divided by the total range of that variable. Note that “standard scoring” is applied to ordinal variables, i.e., they are replaced by their integer codes $1:K$. Note that this is not the same as using their ranks (since there typically are ties).

As the individual contributions $d_{ij}^{(k)}$ are in $[0, 1]$, the dissimilarity $d_{ij}$ will remain in this range. If all weights $w_k \delta_{ij}^{(k)}$ are zero, the dissimilarity is set to NA.

Value

an object of class "dissimilarity" containing the dissimilarities among the rows of x. This is typically the input for the functions `pam`, `fanny`, `agnes` or `diana`. For more details, see `dissimilarity.object`. 
Background

Dissimilarities are used as inputs to cluster analysis and multidimensional scaling. The choice of metric may have a large impact.

Author(s)

Anja Struyf, Mia Hubert, and Peter and Rousseeuw, for the original version. Martin Maechler improved the NA handling and type specification checking, and extended functionality to metric = "gower" and the optional weights argument.

References


See Also

dissimilarity.object, dist, pam, fanny, clara, agnes, diana.

Examples

data(agrCriculture)
## Example 1 in ref:
## Dissimilarities using Euclidean metric and without standardization
d.agr <- daisy(agrCriculture, metric = "euclidean", stand = FALSE)
d.agr
as.matrix(d.agr)[,"DK"] # via as.matrix.dist(.)
## compare with
as.matrix(daisy(agrCriculture, metric = "gower"))

data(flowCer)
## Example 2 in ref
summary(df1L <- daisy(flowCer, type = list(asymm = 3)))
summary(df1L <- daisy(flowCer, type = list(asymm = c(1, 3), ordratio = 7)))
## this failed earlier:
summary(df1L <- daisy(flowCer,
       type = list(asymm = c("V1", "V3"), symm= 2,
                   ordratio= 7, logratio= 8)))
**diana**

*DIVisive ANAlysis Clustering*

**Description**

Computes a divisive hierarchical clustering of the dataset returning an object of class *diana*.

**Usage**

```r
diana(x, diss = inherits(x, "dist"), metric = "euclidean", stand = FALSE,
       stop.at.k = FALSE,
       keep.diss = n < 100, keep.data = !diss, trace.lev = 0)
```

**Arguments**

- **x**
  - data matrix or data frame, or dissimilarity matrix or object, depending on the value of the `diss` argument.
  - In case of a matrix or data frame, each row corresponds to an observation, and each column corresponds to a variable. All variables must be numeric. Missing values (NAs) are allowed.
  - In case of a dissimilarity matrix, `x` is typically the output of `daisy` or `dist`. Also a vector of length n*(n-1)/2 is allowed (where n is the number of observations), and will be interpreted in the same way as the output of the above-mentioned functions. Missing values (NAs) are not allowed.

- **diss**
  - logical flag: if TRUE (default for `dist` or dissimilarity objects), then `x` will be considered as a dissimilarity matrix. If FALSE, then `x` will be considered as a matrix of observations by variables.

- **metric**
  - character string specifying the metric to be used for calculating dissimilarities between observations.
  - The currently available options are "euclidean" and "manhattan". Euclidean distances are root sum-of-squares of differences, and manhattan distances are the sum of absolute differences. If `x` is already a dissimilarity matrix, then this argument will be ignored.

- **stand**
  - logical; if true, the measurements in `x` are standardized before calculating the dissimilarities. Measurements are standardized for each variable (column), by subtracting the variable's mean value and dividing by the variable's mean absolute deviation. If `x` is already a dissimilarity matrix, then this argument will be ignored.

- **stop.at.k**
  - logical or integer, FALSE by default. Otherwise must be integer, say `k`, in \( \{1, 2, \ldots, n\} \), specifying that the `diana` algorithm should stop early.
  - Non-default NOT YET IMPLEMENTED.

- **keep.diss, keep.data**
  - logicals indicating if the dissimilarities and/or input data `x` should be kept in the result. Setting these to FALSE can give much smaller results and hence even save memory allocation time.
trace.lev integer specifying a trace level for printing diagnostics during the algorithm. Default 0 does not print anything; higher values print increasingly more.

Details
diana is fully described in chapter 6 of Kaufman and Rousseeuw (1990). It is probably unique in computing a divisive hierarchy, whereas most other software for hierarchical clustering is agglomerative. Moreover, diana provides (a) the divisive coefficient (see diana.object) which measures the amount of clustering structure found; and (b) the banner, a novel graphical display (see plot.diana).

The diana-algorithm constructs a hierarchy of clusterings, starting with one large cluster containing all n observations. Clusters are divided until each cluster contains only a single observation. At each stage, the cluster with the largest diameter is selected. (The diameter of a cluster is the largest dissimilarity between any two of its observations.) To divide the selected cluster, the algorithm first looks for its most disparate observation (i.e., which has the largest average dissimilarity to the other observations of the selected cluster). This observation initiates the "splinter group". In subsequent steps, the algorithm reassigns observations that are closer to the "splinter group" than to the "old party". The result is a division of the selected cluster into two new clusters.

Value
an object of class "diana" representing the clustering; this class has methods for the following generic functions: print, summary, plot.
Further, the class "diana" inherits from "twins". Therefore, the generic function pltree can be used on a diana object, and as.hclust and as.dendrogram methods are available.
A legitimate diana object is a list with the following components:

order a vector giving a permutation of the original observations to allow for plotting, in the sense that the branches of a clustering tree will not cross.

order.lab a vector similar to order, but containing observation labels instead of observation numbers. This component is only available if the original observations were labelled.

height a vector with the diameters of the clusters prior to splitting.
dc the divisive coefficient, measuring the clustering structure of the dataset. For each observation i, denote by \( d(i) \) the diameter of the last cluster to which it belongs (before being split off as a single observation), divided by the diameter of the whole dataset. The dc is the average of all \( 1 - d(i) \). It can also be seen as the average width (or the percentage filled) of the banner plot. Because dc grows with the number of observations, this measure should not be used to compare datasets of very different sizes.

merge an (n-1) by 2 matrix, where n is the number of observations. Row i of merge describes the split at step n-i of the clustering. If a number \( j \) in row r is negative, then the single observation \( |j| \) is split off at stage n-r. If \( j \) is positive, then the cluster that will be split at stage n-j (described by row j), is split off at stage n-r.
dissimilarity.object

**diss**
an object of class "dissimilarity", representing the total dissimilarity matrix of the dataset.

data
a matrix containing the original or standardized measurements, depending on the stand option of the function agnes. If a dissimilarity matrix was given as input structure, then this component is not available.

See Also

agnes also for background and references; cutree (and as.hclust) for grouping extraction; daisy, dist, plot.diana, twins.object.

Examples

data(votes.repub)
dv <- diana(votes.repub, metric = "manhattan", stand = TRUE)
print(dv)
plot(dv)

### Cut into 2 groups:
dv2 <- cutree(as.hclust(dv), k = 2)
table(dv2) # 8 and 42 group members
rownames(votes.repub)[dv2 == 1]

### For two groups, does the metric matter?
dv0 <- diana(votes.repub, stand = TRUE) # default: Euclidean
dv.2 <- cutree(as.hclust(dv0), k = 2)
table(dv2 == dv.2)# identical group assignments

str(as.dendrogram(dv0)) # (via as.dendrogram.twins() method)

data(agriculture)
### Plot similar to Figure 8 in ref
### Not run: plot(diana(agriculture), ask = TRUE)

---

dissimilarity.object Dissimilarity Matrix Object

**Description**

Objects of class "dissimilarity" representing the dissimilarity matrix of a dataset.

**Value**

The dissimilarity matrix is symmetric, and hence its lower triangle (column wise) is represented as a vector to save storage space. If the object, is called do, and n the number of observations, i.e., n <- attr(do, "Size"), then for $i < j \leq n$, the dissimilarity between (row) i and j is $do[n*(i-1) - i*(i-1)/2 + j-1]$. The length of the vector is $n * (n - 1)/2$, i.e., of order $n^2$. 

"dissimilarity" objects also inherit from class \texttt{dist} and can use \texttt{dist} methods, in particular, \texttt{as.matrix}, such that \(d_{ij}\) from above is just \texttt{as.matrix(do)[i,j]}.

The object has the following attributes:

- **Size**: the number of observations in the dataset.
- **Metric**: the metric used for calculating the dissimilarities. Possible values are "euclidean", "manhattan", "mixed" (if variables of different types were present in the dataset), and "unspecified".
- **Labels**: optionally, contains the labels, if any, of the observations of the dataset.
- **NA.message**: optionally, if a dissimilarity could not be computed, because of too many missing values for some observations of the dataset.
- **Types**: when a mixed metric was used, the types for each variable as one-letter codes (as in the book, e.g. p.54):
  - A: Asymmetric binary
  - S: Symmetric binary
  - N: Nominal (factor)
  - O: Ordinal (ordered factor)
  - I: Interval scaled (numeric)
  - T: Ratio to be log transformed (positive numeric)

**GENERATION**

\texttt{daisy} returns this class of objects. Also the functions \texttt{pam}, \texttt{clara}, \texttt{fanny}, \texttt{agnes}, and \texttt{diana} return a dissimilarity object, as one component of their return objects.

**METHODS**

The "dissimilarity" class has methods for the following generic functions: \texttt{print}, \texttt{summary}.

**See Also**

\texttt{daisy, dist, pam, clara, fanny, agnes, diana}.
Usage

ellipsoidhull(x, tol=0.01, maxit=5000,
    ret.wt = FALSE, ret.sqdist = FALSE, ret.pr = FALSE)
    ## S3 method for class 'ellipsoid'
print(x, digits = max(1, getOption("digits") - 2), ...)

Arguments

x the \( n \times p \) dimensional points as numeric \( n \times p \) matrix.
tol convergence tolerance for Titterington’s algorithm. Setting this to much smaller
    values may drastically increase the number of iterations needed, and you may
    want to increas maxit as well.
maxit integer giving the maximal number of iteration steps for the algorithm.
ret.wt, ret.sqdist, ret.pr
    logicals indicating if additional information should be returned, ret.wt specifying
    the weights, ret.sqdist the squared distances and ret.pr the final
    probabilities in the algorithms.
digits,... the usual arguments to print methods.

Details

The “spanning ellipsoid” algorithm is said to stem from Titterington(1976), in Pison et al (1999)
who use it for clusplot.default.
The problem can be seen as a special case of the “Min.V ol.” ellipsoid of which a more more flexible
and general implementation is cov.mve in the MASS package.

Value

an object of class "ellipsoid", basically a list with several components, comprising at least

cov \( p \times p \) covariance matrix description the ellipsoid.
loc \( p \)-dimensional location of the ellipsoid center.
d2 average squared radius. Further, \( d^2 = t^2 \), where \( t \) is “the value of a t-statistic on
    the ellipse boundary” (from ellipse in the ellipse package), and hence, more
    usefully, \( d^2 = qchisq(alpha, df = p) \), where alpha is the confidence level
    for p-variate normally distributed data with location and covariance loc and cov
to lie inside the ellipsoid.
wt the vector of weights iff ret.wt was true.
sqdist the vector of squared distances iff ret.sqdist was true.
prob the vector of algorithm probabilities iff ret.pr was true.
it number of iterations used.
tol, maxit just the input argument, see above.
eps the achieved tolerance which is the maximal squared radius minus \( p \).
 ierr error code as from the algorithm; \( 0 \) means ok.
 conv logical indicating if the converged. This is defined as it < maxit \&\& ierr == 0.
Author(s)

Martin Maechler did the present class implementation; Rousseeuw et al did the underlying code.

References


See Also

`predict.ellipsoid` which is also the `predict` method for ellipsoid objects. `volume.ellipsoid` for an example of ‘manual’ ellipsoid object construction; further `ellipse` from package `ellipse` and `ellipsePoints` from package `sfsmisc`.

`chull` for the convex hull, `clusplot` which makes use of this; `cov.mve`.

Examples

```r
x <- rnorm(100)
xy <- unname(cbind(x, rnorm(100) + 2*x + 10))
exy <- ellipsoidhull(xy)
exy # >> calling print.ellipsoid()

plot(xy, main = "ellipsoidhull(<Gauss data>) -- 'spanning points'")
lines(predict(exy), col="blue")
points(rbind(exy$loc), col = "red", cex = 3, pch = 13)

exy <- ellipsoidhull(xy, tol = 1e-7, ret.wt = TRUE, ret.sq = TRUE)
str(exy) # had small 'tol', hence many iterations
(ii <- which(zapsmall(exy$wt) > 1e-6))
## --> only about 4 to 6 "spanning ellipsoid" points
round(exy$wt[ii],3); sum(exy$wt[ii]) # weights summing to 1
points(xy[ii,], pch = 21, cex = 2,
      col="blue", bg = adjustcolor("blue",0.25))
```

---

**fanny**

Fuzzy Analysis Clustering

---

Description

Computes a fuzzy clustering of the data into k clusters.
Usage

fanny(x, k, diss = inherits(x, "dist"), memb.exp = 2,
         metric = c("euclidean", "manhattan", "sqEuclidean"),
         stand = FALSE, iniMem.p = NULL, cluster.only = FALSE,
         keep.diss = !diss && !cluster.only && n < 100,
         keep.data = !diss && !cluster.only,
         maxit = 500, tol = 1e-15, trace.lev = 0)

Arguments

x          data matrix or data frame, or dissimilarity matrix, depending on the value of the diss argument.
            In case of a matrix or data frame, each row corresponds to an observation, and each column corresponds to a variable. All variables must be numeric. Missing values (NA) are allowed.
            In case of a dissimilarity matrix, x is typically the output of daisy or dist. Also a vector of length n*(n-1)/2 is allowed (where n is the number of observations), and will be interpreted in the same way as the output of the above-mentioned functions. Missing values (NA) are not allowed.

k          integer giving the desired number of clusters. It is required that 0 < k < n/2 where n is the number of observations.

diss       logical flag: if TRUE (default for dist or dissimilarity objects), then x is assumed to be a dissimilarity matrix. If FALSE, then x is treated as a matrix of observations by variables.

memb.exp   number r strictly larger than 1 specifying the membership exponent used in the fit criterion; see the 'Details' below. Default: 2 which used to be hardwired inside FANNY.

metric     character string specifying the metric to be used for calculating dissimilarities between observations. Options are "euclidean" (default), "manhattan", and "sqEuclidean". Euclidean distances are root sum-of-squares of differences, and manhattan distances are the sum of absolute differences, and "sqEuclidean", the squared euclidean distances are sum-of-squares of differences. Using this last option is equivalent (but somewhat slower) to computing so called “fuzzy C-means". If x is already a dissimilarity matrix, then this argument will be ignored.

stand      logical; if true, the measurements in x are standardized before calculating the dissimilarities. Measurements are standardized for each variable (column), by subtracting the variable’s mean value and dividing by the variable’s mean absolute deviation. If x is already a dissimilarity matrix, then this argument will be ignored.

iniMem.p   numeric n x k matrix or NULL (by default); can be used to specify a starting membership matrix, i.e., a matrix of non-negative numbers, each row summing to one.

cluster.only logical; if true, no silhouette information will be computed and returned, see details.
keep.diss, keep.data  
logicals indicating if the dissimilarities and/or input data x should be kept in 
the result. Setting these to FALSE can give smaller results and hence also save 
memory allocation time.

maxit, tol  
maximal number of iterations and default tolerance for convergence (relative 
convergence of the fit criterion) for the FANNY algorithm. The defaults maxit = 500 
and tol = 1e-15 used to be hardwired inside the algorithm.

trace.lev  
integer specifying a trace level for printing diagnostics during the C-internal 
algorithm. Default 0 does not print anything; higher values print increasingly 
more.

Details

In a fuzzy clustering, each observation is “spread out” over the various clusters. Denote by \( u_{iv} \) the 
membership of observation \( i \) to cluster \( v \).

The memberships are nonnegative, and for a fixed observation \( i \) they sum to 1. The particular 
method fanny stems from chapter 4 of Kaufman and Rousseeuw (1990) (see the references in 
daisy) and has been extended by Martin Maechler to allow user specified memb.exp, iniMem.p, 
maxit, tol, etc.

Fanny aims to minimize the objective function

\[
\sum_{v=1}^{k} \sum_{i=1}^{n} \sum_{j=1}^{n} u_{iv}^r u_{jv}^r d(i,j) = \frac{1}{2} \sum_{j=1}^{n} u_{jv}^r
\]

where \( n \) is the number of observations, \( k \) is the number of clusters, \( r \) is the membership exponent 
memb.exp and \( d(i,j) \) is the dissimilarity between observations \( i \) and \( j \).

Note that \( r \to 1 \) gives increasingly crisper clusterings whereas \( r \to \infty \) leads to complete fuzzyness. 
K&R(1990), p.191 note that values too close to 1 can lead to slow convergence. Further note that 
even the default, \( r = 2 \) can lead to complete fuzzyness, i.e., memberships \( u_{iv} = 1/k \). In that case 
a warning is signalled and the user is advised to chose a smaller memb.exp (= \( r \)).

Compared to other fuzzy clustering methods, fanny has the following features: (a) it also accepts a 
dissimilarity matrix; (b) it is more robust to the spherical cluster assumption; (c) it provides a 
novel graphical display, the silhouette plot (see plot.partition).

Value

an object of class \"fanny\" representing the clustering. See fanny.object for details.

See Also

agnes for background and references; fanny.object, partition.object, plot.partition, daisy, 
dist.

Examples

```r
## generate 10+15 objects in two clusters, plus 3 objects lying 
## between those clusters.
x <- rbind(cbind(rnorm(10, 0, 0.5), rnorm(10, 0, 0.5)),
```
```r
fanny.object

Description

The objects of class "fanny" represent a fuzzy clustering of a dataset.

Value

A legitimate fanny object is a list with the following components:

- **membership**  
  matrix containing the memberships for each pair consisting of an observation and a cluster.

- **memb.exp**  
  the membership exponent used in the fitting criterion.

- **coeff**  
  Dunn’s partition coefficient $F(k)$ of the clustering, where $k$ is the number of clusters.  
  $F(k)$ is the sum of all squared membership coefficients, divided by the number of observations.  
  Its value is between $1/k$ and 1.  
  The normalized form of the coefficient is also given.  
  It is defined as $(F(k) - 1/k)/(1 - 1/k)$, and ranges between 0 and 1.  
  A low value of Dunn's coefficient indicates a very fuzzy clustering, whereas a value close to 1 indicates a near-crisp clustering.

- **clustering**  
  the clustering vector of the nearest crisp clustering, see `partition.object`.

- **k.crisp**  
  integer ($\leq k$) giving the number of crisp clusters; can be less than $k$, where it's recommended to decrease memb.exp.

- **objective**  
  named vector containing the minimal value of the objective function reached by the FANNY algorithm and the relative convergence tolerance tol used.

- **convergence**  
  named vector with iterations, the number of iterations needed and converged indicating if the algorithm converged (in maxit iterations within convergence tolerance tol).
```
flower

8 characteristics for 18 popular flowers.

Usage
data(flower)

Format
A data frame with 18 observations on 8 variables:

```
  [,1] factor winters
  [,2] factor shadow
  [,3] factor tubers
  [,4] factor color
  [,5] ordered soil
  [,6] ordered preference
  [,7] numeric height
  [,8] numeric distance
```

V1 winters, is binary and indicates whether the plant may be left in the garden when it freezes.
V2  shadow, is binary and shows whether the plant needs to stand in the shadow.
V3  tubers, is asymmetric binary and distinguishes between plants with tubers and plants that grow in any other way.
V4  color, is nominal and specifies the flower’s color (1 = white, 2 = yellow, 3 = pink, 4 = red, 5 = blue).
V5  soil, is ordinal and indicates whether the plant grows in dry (1), normal (2), or wet (3) soil.
V6  preference, is ordinal and gives someone’s preference ranking going from 1 to 18.
V7  height, is interval scaled, the plant’s height in centimeters.
V8  distance, is interval scaled, the distance in centimeters that should be left between the plants.

References
Struyf, Hubert and Rousseeuw (1996), see agnes.

Examples
data(flower)
## Example 2 in ref
daisy(flower, type = list(asymm = 3))
daisy(flower, type = list(asymm = c(1, 3), ordratio = 7))

lower.to.upper.tri.ind

Permute Indices for Triangular Matrices

Description
Compute index vectors for extracting or reordering of lower or upper triangular matrices that are stored as contiguous vectors.

Usage
lower.to.upper.tri.ind(n)
upper.to.lower.tri.ind(n)

Arguments
n   integer larger than 1.

Value
integer vector containing a permutation of 1:N where N = n(n - 1)/2.

See Also
upper.tri, lower.tri with a related purpose.
Examples

```r
m5 <- matrix(NA, 5, 5)
m <- m5; m[lower.tri(m)] <- upper.to.lower.tri inds(5); m
m <- m5; m[upper.tri(m)] <- lower.to.upper.tri inds(5); m

stopifnot(lower.to.upper.tri inds(2) == 1,
  lower.to.upper.tri inds(3) == 1:3,
  upper.to.lower.tri inds(3) == 1:3,
  sort(upper.to.lower.tri inds(5)) == 1:10,
  sort(lower.to.upper.tri inds(6)) == 1:15)
```

Description

Returns a list representing a divisive hierarchical clustering of a dataset with binary variables only.

Usage

`mona(x, trace.lev = 0)`

Arguments

- `x` data matrix or data frame in which each row corresponds to an observation, and each column corresponds to a variable. All variables must be binary. A limited number of missing values (NAs) is allowed. Every observation must have at least one value different from NA. No variable should have half of its values missing. There must be at least one variable which has no missing values. A variable with all its non-missing values identical is not allowed.
- `trace.lev` logical or integer indicating if (and how much) the algorithm should produce progress output.

Details

`mona` is fully described in chapter 7 of Kaufman and Rousseeuw (1990). It is “monothetic” in the sense that each division is based on a single (well-chosen) variable, whereas most other hierarchical methods (including agnes and diana) are “polythetic”, i.e. they use all variables together.

The `mona`-algorithm constructs a hierarchy of clusterings, starting with one large cluster. Clusters are divided until all observations in the same cluster have identical values for all variables. At each stage, all clusters are divided according to the values of one variable. A cluster is divided into one cluster with all observations having value 1 for that variable, and another cluster with all observations having value 0 for that variable.

The variable used for splitting a cluster is the variable with the maximal total association to the other variables, according to the observations in the cluster to be splitted. The association between variables f and g is given by a(f,g)*d(f,g) - b(f,g)*c(f,g), where a(f,g), b(f,g), c(f,g), and d(f,g) are the numbers in the contingency table of f and g. [That is, a(f,g) (resp. d(f,g)) is the number of
observations for which f and g both have value 0 (resp. value 1); b(f,g) (resp. c(f,g)) is the number of observations for which f has value 0 (resp. 1) and g has value 1 (resp. 0).] The total association of a variable f is the sum of its associations to all variables.

Value

an object of class "mona" representing the clustering. See mona.object for details.

Missing Values (NAs)

The mona-algorithm requires “pure” 0-1 values. However, mona(x) allows x to contain (not too many) NAs. In a preliminary step, these are “imputed”, i.e., all missing values are filled in. To do this, the same measure of association between variables is used as in the algorithm. When variable f has missing values, the variable g with the largest absolute association to f is looked up. When the association between f and g is positive, any missing value of f is replaced by the value of g for the same observation. If the association between f and g is negative, then any missing value of f is replaced by the value of 1-g for the same observation.

Note

In cluster versions before 2.0.6, the algorithm entered an infinite loop in the boundary case of one variable, i.e., ncol(x) == 1, which currently signals an error (because the algorithm now in C, haes not correctly taken account of this special case).

See Also

agnes for background and references; mona.object, plot.mona.

Examples

data(animals)
ma <- mona(animals)
ma

## Plot similar to Figure 10 in Struyf et al (1996)
plot(ma)

## One place to see if/how error messages are *translated* (to 'de' / 'pl'):
ani.NA <- animals; ani.NA[4,] <- NA
aniNA <- within(animals, { end[2:9] <- NA })
aniN2 <- animals; aniN2[cbind(1:6, c(3, 1, 4:6, 2))] <- NA
ani.non2 <- within(animals, end[7] <- 3)
ani.idNA <- within(animals, end[!is.na(end)] <- 1)
try( mona(ani.NA) ) ## error: .. object with all values missing
try( mona(aniNA) ) ## error: .. more than half missing values
try( mona(aniN2) ) ## error: all have at least one missing
try( mona(ani.non2) ) ## error: all must be binary
try( mona(ani.idNA) ) ## error: ditto
### mona.object

#### Description
The objects of class "mona" represent the divisive hierarchical clustering of a dataset with only binary variables (measurements). This class of objects is returned from mona.

#### Value
A legitimate mona object is a list with the following components:

- **data**: matrix with the same dimensions as the original data matrix, but with factors coded as 0 and 1, and all missing values replaced.
- **order**: a vector giving a permutation of the original observations to allow for plotting, in the sense that the branches of a clustering tree will not cross.
- **order.lab**: a vector similar to order, but containing observation labels instead of observation numbers. This component is only available if the original observations were labelled.
- **variable**: vector of length n-1 where n is the number of observations, specifying the variables used to separate the observations of order.
- **step**: vector of length n-1 where n is the number of observations, specifying the separation steps at which the observations of order are separated.

#### METHODS
The "mona" class has methods for the following generic functions: print, summary, plot.

#### See Also
mona for examples etc, plot.mona.

### pam

#### Partitioning Around Medoids

#### Description
Partitioning (clustering) of the data into k clusters “around medoids”, a more robust version of K-means.
Usage

pam(x, k, diss = inherits(x, "dist"),
    metric = c("euclidean", "manhattan"),
    medoids = NULL, stand = FALSE, cluster.only = FALSE,
    do.swap = TRUE,
    keep.diss = !diss && !cluster.only && n < 100,
    keep.data = !diss && !cluster.only,
    pamonce = FALSE, trace.lev = 0)

Arguments

x            data matrix or data frame, or dissimilarity matrix or object, depending on the
             value of the diss argument.
             In case of a matrix or data frame, each row corresponds to an observation, and
each column corresponds to a variable. All variables must be numeric. Missing
values (NAs) are allowed—as long as every pair of observations has at least one
case not missing.
             In case of a dissimilarity matrix, x is typically the output of daisy or dist. Also
a vector of length n*(n-1)/2 is allowed (where n is the number of observations),
and will be interpreted in the same way as the output of the above-mentioned
functions. Missing values (NAs) are not allowed.

k            positive integer specifying the number of clusters, less than the number of ob-
             servations.

diss         logical flag: if TRUE (default for dist or dissimilarity objects), then x will
             be considered as a dissimilarity matrix. If FALSE, then x will be considered as
a matrix of observations by variables.

metric       character string specifying the metric to be used for calculating dissimilarities
             between observations.
             The currently available options are "euclidean" and "manhattan". Euclidean dis-
stances are root sum-of-squares of differences, and manhattan distances are the
sum of absolute differences. If x is already a dissimilarity matrix, then this argu-
ment will be ignored.

medoids      NULL (default) or length-k vector of integer indices (in 1:n) specifying initial
             medoids instead of using the 'build' algorithm.

stand        logical; if true, the measurements in x are standardized before calculating the
dissimilarities. Measurements are standardized for each variable (column), by
subtracting the variable's mean value and dividing by the variable's mean abso-
lute deviation. If x is already a dissimilarity matrix, then this argument will be
ignored.

cluster.only logical; if true, only the clustering will be computed and returned, see details.

do.swap      logical indicating if the swap phase should happen. The default, TRUE, corre-
spond to the original algorithm. On the other hand, the swap phase is much
more computer intensive than the build one for large n, so can be skipped by
do.swap = FALSE.
keep.diss, keep.data
logicals indicating if the dissimilarities and/or input data x should be kept in the result. Setting these to FALSE can give much smaller results and hence even save memory allocation time.

pamonce
logical or integer in 0:5 specifying algorithmic short cuts as proposed by Reynolds et al. (2006), and Schubert and Rousseeuw (2019) see below.

trace.lev
integer specifying a trace level for printing diagnostics during the build and swap phase of the algorithm. Default 0 does not print anything; higher values print increasingly more.

Details
The basic pam algorithm is fully described in chapter 2 of Kaufman and Rousseeuw(1990). Compared to the k-means approach in kmeans, the function pam has the following features: (a) it also accepts a dissimilarity matrix; (b) it is more robust because it minimizes a sum of dissimilarities instead of a sum of squared euclidean distances; (c) it provides a novel graphical display, the silhouette plot (see plot.partition) (d) it allows to select the number of clusters using mean(silhouette(pr)[, "sil_width"]) on the result pr <- pam(...), or directly its component pr$silinfo$avg.width, see also pam.object.

When cluster.only is true, the result is simply a (possibly named) integer vector specifying the clustering, i.e., pam(x,k, cluster.only=TRUE) is the same as pam(x,k)$clustering but computed more efficiently.

The pam-algorithm is based on the search for k representative objects or medoids among the observations of the dataset. These observations should represent the structure of the data. After finding a set of k medoids, k clusters are constructed by assigning each observation to the nearest medoid. The goal is to find k representative objects which minimize the sum of the dissimilarities of the observations to their closest representative object.

By default, when medoids are not specified, the algorithm first looks for a good initial set of medoids (this is called the build phase). Then it finds a local minimum for the objective function, that is, a solution such that there is no single switch of an observation with a medoid that will decrease the objective (this is called the swap phase).

When the medoids are specified, their order does not matter; in general, the algorithms have been designed to not depend on the order of the observations.

The pamonce option, new in cluster 1.14.2 (Jan. 2012), has been proposed by Matthias Studer, University of Geneva, based on the findings by Reynolds et al. (2006) and was extended by Erich Schubert, TU Dortmund, with the FastPAM optimizations.

The default FALSE (or integer 0) corresponds to the original “swap” algorithm, whereas pamonce = 1 (or TRUE), corresponds to the first proposal .... and pamonce = 2 additionally implements the second proposal as well.

The key ideas of FastPAM (Schubert and Rousseeuw, 2019) are implemented except for the linear approximate build as follows:

pamonce = 3: reduces the runtime by a factor of O(k) by exploiting that points cannot be closest to all current medoids at the same time.

pamonce = 4: additionally allows executing multiple swaps per iteration, usually reducing the number of iterations.
pamonce = 5: adds minor optimizations copied from the pamonce = 2 approach, and is expected
to be the fastest variant.

Value

an object of class "pam" representing the clustering. See ?pam.object for details.

Note

For large datasets, pam may need too much memory or too much computation time since both are
$O(n^2)$. Then, clara() is preferable, see its documentation.
There is hard limit currently, $n \leq 65536$, at $2^{16}$ because for larger $n$, $n(n - 1)/2$ is larger than the
maximal integer (.Machine$integer.max = 2^{31} - 1)$.

Author(s)

Kaufman and Rousseeuw’s orginal Fortran code was translated to C and augmented in several ways,
e.g. to allow cluster.only=TRUE or do.swap=FALSE, by Martin Maechler.
Matthias Studer, Univ.Geneva provided the pamonce (1 and R) implementation.

References

comparison of partitioning and hierarchical clustering algorithms; Journal of Mathematical Mod-
Erich Schubert and Peter J. Rousseeuw (2019) Faster k-Medoids Clustering: Improving the PAM,

See Also

agnes for background and references; pam.object, clara, daisy, partition.object, plot.partition,
dist.

Examples

```r
## generate 25 objects, divided into 2 clusters.
x <- rbind(cbind(rnorm(10, 0, 0.5), rnorm(10, 0, 0.5)),
           cbind(rnorm(15, 5, 0.5), rnorm(15, 5, 0.5)))
pamx <- pam(x, 2)
pamx # Medoids: '7' and '25' ...
summary(pamx)
plot(pamx)
## use obs. 1 & 16 as starting medoids -- same result (typically)
(p2m <- pam(x, 2, medoids = c(1,16)))
## no _build_ *and* no _swap_ phase: just cluster all obs. around (1, 16):
p2.s <- pam(x, 2, medoids = c(1,16), do.swap = FALSE)
p2.s

p3m <- pam(x, 3, trace = 2)
## rather stupid initial medoids:
```
(p3m. <- pam(x, 3, medoids = 3:1, trace = 1))

pam(daisy(x, metric = "manhattan"), 2, diss = TRUE)

data(ruspini)
## Plot similar to Figure 4 in Stryuf et al (1996)
## Not run: plot(pam(ruspini, 4), ask = TRUE)

---

pam.object

**Partitioning Around Medoids (PAM) Object**

**Description**

The objects of class "pam" represent a partitioning of a dataset into clusters.

**Value**

A legitimate pam object is a list with the following components:

- **medoids**: the medoids or representative objects of the clusters. If a dissimilarity matrix was given as input to pam, then a vector of numbers or labels of observations is given, else medoids is a matrix with in each row the coordinates of one medoid.

- **id.med**: integer vector of indices giving the medoid observation numbers.

- **clustering**: the clustering vector, see partition.object.

- **objective**: the objective function after the first and second step of the pam algorithm.

- **isolation**: vector with length equal to the number of clusters, specifying which clusters are isolated clusters (L- or L*-clusters) and which clusters are not isolated.

  A cluster is an L*-cluster iff its diameter is smaller than its separation. A cluster is an L-cluster iff for each observation i the maximal dissimilarity between i and any other observation of the cluster is smaller than the minimal dissimilarity between i and any observation of another cluster. Clearly each L*-cluster is also an L-cluster.

- **clusinfo**: matrix, each row gives numerical information for one cluster. These are the cardinality of the cluster (number of observations), the maximal and average dissimilarity between the observations in the cluster and the cluster's medoid, the diameter of the cluster (maximal dissimilarity between two observations of the cluster), and the separation of the cluster (minimal dissimilarity between an observation of the cluster and an observation of another cluster).

- **silinfo**: list with silhouette width information, see partition.object.

- **diss**: dissimilarity (maybe NULL), see partition.object.

- **call**: generating call, see partition.object.

- **data**: (possibly standardized) see partition.object.
**GENERATION**

These objects are returned from `pam`.

**METHODS**

The "pam" class has methods for the following generic functions: `print`, `summary`.

**INHERITANCE**

The class "pam" inherits from "partition". Therefore, the generic functions `plot` and `clusplot` can be used on a `pam` object.

**See Also**

`pam`, `dissimilarity.object`, `partition.object`, `plot.partition`.

**Examples**

```r
## Use the silhouette widths for assessing the best number of clusters,
## following a one-dimensional example from Christian Hennig:
##
x <- c(rnorm(50), rnorm(50, mean=5), rnorm(30, mean=15))
asw <- numeric(20)
## Note that "k=1" won't work!
for (k in 2:20)
  asw[k] <- pam(x, k, $silinfo $avg.width
k.best <- which.max(asw)
cat("silhouette-optimal number of clusters: ", k.best, "\n")

plot(1:20, asw, type="h", main = "pam() clustering assessment",
    xlab="k (# clusters)", ylab = "average silhouette width")
axis(1, k.best, paste("best", k.best, sep="\n"), col = "red", col.axis = "red")
```

---

**partition.object**

**Partitioning Object**

**Description**

The objects of class "partition" represent a partitioning of a dataset into clusters.

**Value**

A "partition" object is a list with the following (and typically more) components:

- **clustering**: The clustering vector. An integer vector of length $n$, the number of observations, giving for each observation the number ('id') of the cluster to which it belongs.
- **call**: The matched call generating the object.
silinfo

A list with all *silhouette* information, only available when the number of clusters is non-trivial, i.e., \( 1 < k < n \) and then has the following components, see *silhouette*

- **widths**: an \((n \times 3)\) matrix, as returned by *silhouette()*, with for each observation \( i \) the cluster to which \( i \) belongs, as well as the neighbor cluster of \( i \) (the cluster, not containing \( i \), for which the average dissimilarity between its observations and \( i \) is minimal), and the silhouette width \( s(i) \) of the observation.

- **clus.avg.widths**: the average silhouette width per cluster.

- **avg.width**: the average silhouette width for the dataset, i.e., simply the average of \( s(i) \) over all observations \( i \).

This information is also needed to construct a *silhouette plot* of the clustering, see *plot.partition*.

Note that *avg.width* can be maximized over different clusterings (e.g. with varying number of clusters) to choose an optimal clustering.

**objective**

Value of criterion maximized during the partitioning algorithm, may more than one entry for different stages.

diss

An object of class "dissimilarity", representing the total dissimilarity matrix of the dataset (or relevant subset, e.g. for *clara*).

data

A matrix containing the original or standardized data. This might be missing to save memory or when a dissimilarity matrix was given as input structure to the clustering method.

**GENERATION**

These objects are returned from *pam*, *clara* or *fanny*.

**METHODS**

The "partition" class has a method for the following generic functions: *plot*, *clusplot*.

**INHERITANCE**

The following classes inherit from class "partition": "pam", "clara" and "fanny".

See *pam.object*, *clara.object* and *fanny.object* for details.

**See Also**

*pam*, *clara*, *fanny*.
plantTraits  

**Plant Species Traits Data**

**Description**

This dataset constitutes a description of 136 plant species according to biological attributes (morphological or reproductive)

**Usage**

```r
data(plantTraits)
```

**Format**

A data frame with 136 observations on the following 31 variables.

- `pdias` Diaspore mass (mg)
- `longindex` Seed bank longevity
- `durflow` Flowering duration
- `height` Plant height, an ordered factor with levels `1 < 2 < ... < 8`
- `begflow` Time of first flowering, an ordered factor with levels `1 < 2 < 3 < 4 < 5 < 6 < 7 < 8 < 9`
- `mycor` Mycorrhizas, an ordered factor with levels `never < 1 sometimes < 2 always`
- `vegaer` Aerial vegetative propagation, an ordered factor with levels `never < 1 present but limited < 2 important`
- `vegsout` Underground vegetative propagation, an ordered factor with 3 levels identical to `vegaer` above.
- `autopoll` Selfing pollination, an ordered factor with levels `never < 1 rare < 2 often < the rule`
- `insects` Insect pollination, an ordered factor with 5 levels `0 < ... < 4`
- `wind` Wind pollination, an ordered factor with 5 levels `0 < ... < 4`
- `lign` A binary factor with levels `0:1`, indicating if plant is woody.
- `piq` A binary factor indicating if plant is thorny.
- `ros` A binary factor indicating if plant is rosette.
- `semiros` Semi-rosette plant, a binary factor (`0: no; 1: yes`.
- `leafy` Leafy plant, a binary factor.
- `suman` Summer annual, a binary factor.
- `winan` Winter annual, a binary factor.
- `monocarp` Monocarpic perennial, a binary factor.
- `polycarp` Polycarpic perennial, a binary factor.
- `seasaes` Seasonal aestival leaves, a binary factor.
- `seashiv` Seasonal hibernal leaves, a binary factor.
seasver seasonal vernal leaves, a binary factor.
everalw leaves always evergreen, a binary factor.
everparti leaves partially evergreen, a binary factor.
elaio fruits with an elaiosome (dispersed by ants), a binary factor.
dero endozoochorous fruits, a binary factor.
esto epizoochorous fruits, a binary factor.
aquat aquatic dispersal fruits, a binary factor.
windsd wind dispersed fruits, a binary factor.
unsp unspecialized mechanism of seed dispersal, a binary factor.

Details

Most of factor attributes are not disjunctive. For example, a plant can be usually pollinated by insects but sometimes self-pollination can occurred.

Source


Examples

data(plantTraits)

## Calculation of a dissimilarity matrix
library(cluster)
dai.b <- daisy(plantTraits, 
    type = list(ordratio = 4:11, symm = 12:13, asymm = 14:31))

## Hierarchical classification
agn.trts <- agnes(dai.b, method="ward")
plot(agn.trts, which.plots = 2, cex= 0.6)
plot(agn.trts, which.plots = 1)
cutree6 <- cutree(agn.trts, k=6)
cutree6

## Principal Coordinate Analysis
cmdsdai.b <- cmdscale(dai.b, k=6)
plot(cmdsdai.b[, 1:2], asp = 1, col = cutree6)
Description

Creates plots for visualizing an agnes object.

Usage

## S3 method for class 'agnes'
plot(x, ask = FALSE, which.plots = NULL, main = NULL,
     sub = paste("Agglomerative Coefficient = ",round(x$ac, digits = 2)),
     adj = 0, nmax.lab = 35, max.strlen = 5, xax.pretty = TRUE, ...)

Arguments

- **x**: an object of class "agnes", typically created by `agnes(.)`.
- **ask**: logical; if true and which.plots is NULL, `plot.agnes` operates in interactive mode, via menu.
- **which.plots**: integer vector or NULL (default), the latter producing both plots. Otherwise, which.plots must contain integers of 1 for a banner plot or 2 for a dendrogram or "clustering tree".
- **main**, **sub**: main and sub title for the plot, with convenient defaults. See documentation for these arguments in `plot.default`.
- **adj**: for label adjustment in `bannerplot()`.
- **nmax.lab**: integer indicating the number of labels which is considered too large for single-name labelling the banner plot.
- **max.strlen**: positive integer giving the length to which strings are truncated in banner plot labeling.
- **xax.pretty**: logical or integer indicating if `pretty(*, n = xax.pretty)` should be used for the x axis. `xax.pretty = FALSE` is for back compatibility.
- **...**: graphical parameters (see `par`) may also be supplied and are passed to `bannerplot()` or `pltree()` (see `pltree.twins`), respectively.

Details

When `ask = TRUE`, rather than producing each plot sequentially, `plot.agnes` displays a menu listing all the plots that can be produced. If the menu is not desired but a pause between plots is still wanted one must set `par(ask = TRUE)` before invoking the plot command.

The banner displays the hierarchy of clusters, and is equivalent to a tree. See Rousseeuw (1986) or chapter 5 of Kaufman and Rousseeuw (1990). The banner plots distances at which observations and clusters are merged. The observations are listed in the order found by the agnes algorithm, and the numbers in the height vector are represented as bars between the observations.
The leaves of the clustering tree are the original observations. Two branches come together at the
distance between the two clusters being merged.
For more customization of the plots, rather call bannerplot and pltree(), i.e., its method pltree.twins,
respectively.
directly with corresponding arguments, e.g., xlab or ylab.

Side Effects
Appropriate plots are produced on the current graphics device. This can be one or both of the
following choices:
Banner
Clustering tree

Note
In the banner plot, observation labels are only printed when the number of observations is limited
less than nmax.lab (35, by default), for readability. Moreover, observation labels are truncated to
maximally max.strlen (5) characters.
For the dendrogram, more flexibility than via pltree() is provided by dg <- as.dendrogram(x) and plotting
dg via plot.dendrogram.

References
Kaufman, L. and Rousseeuw, P.J. (1990) Finding Groups in Data: An Introduction to Cluster Anal-
Rousseeuw, P.J. (1986). A visual display for hierarchical classification, in Data Analysis and Infor-
matics 4; edited by E. Diday, Y. Escoufier, L. Lebart, J. Pages, Y. Schektman, and R. Tomassone.
North-Holland, Amsterdam, 743–748.
Struyf, A., Hubert, M. and Rousseeuw, P.J. (1997) Integrating Robust Clustering Techniques in
S-PLUS, Computational Statistics and Data Analysis, 26, 17–37.

See Also
agnes and agnes.object; bannerplot, pltree.twins, and par.

Examples
## Can also pass 'labels' to pltree() and bannerplot():
data(iris)
cS <- as.character(Sp <- iris$Species)
cS[Sp == "setosa"] <- "s"
cS[Sp == "versicolor"] <- "v"
cS[Sp == "virginica"] <- "g"
aig <- agnes(iris[, 1:4])
plot(aig, labels = cS, nmax = 150)# bannerplot labels are mess
**Description**

Creates plots for visualizing a `diana` object.

**Usage**

```r
## S3 method for class 'diana'
plot(x, ask = FALSE, which.plots = NULL, main = NULL,
     sub = paste("Divisive Coefficient = ", round(x$dci, digits = 2)),
     adj = 0, nmax.lab = 35, max.strlen = 5, xax.pretty = TRUE, ...)
```

**Arguments**

- `x`: an object of class "diana", typically created by `diana(.)`.
- `ask`: logical; if true and which.plots is NULL, `plot.diana` operates in interactive mode, via menu.
- `which.plots`: integer vector or NULL (default), the latter producing both plots. Otherwise, which.plots must contain integers of 1 for a banner plot or 2 for a dendrogram or "clustering tree".
- `main`, `sub`: main and sub title for the plot, each with a convenient default. See documentation for these arguments in `plot.default`.
- `adj`: for label adjustment in `bannerplot()`.
- `nmax.lab`: integer indicating the number of labels which is considered too large for single-name labelling the banner plot.
- `max.strlen`: positive integer giving the length to which strings are truncated in banner plot labeling.
- `xax.pretty`: logical or integer indicating if `pretty(*, n = xax.pretty)` should be used for the x axis. `xax.pretty = FALSE` is for back compatibility.
- `...`: graphical parameters (see `par`) may also be supplied and are passed to `bannerplot()` or `pltree()`, respectively.

**Details**

When `ask` = `TRUE`, rather than producing each plot sequentially, `plot.diana` displays a menu listing all the plots that can be produced. If the menu is not desired but a pause between plots is still wanted one must set `par(ask= TRUE)` before invoking the plot command.

The banner displays the hierarchy of clusters, and is equivalent to a tree. See Rousseeuw (1986) or chapter 6 of Kaufman and Rousseeuw (1990). The banner plots the diameter of each cluster being splitted. The observations are listed in the order found by the `diana` algorithm, and the numbers in the height vector are represented as bars between the observations.

The leaves of the clustering tree are the original observations. A branch splits up at the diameter of the cluster being splitted.
**Side Effects**

An appropriate plot is produced on the current graphics device. This can be one or both of the following choices:
- Banner
- Clustering tree

**Note**

In the banner plot, observation labels are only printed when the number of observations is limited less than `nmax.lab` (35, by default), for readability. Moreover, observation labels are truncated to maximally `max.strlen` (5) characters.

**References**

see those in `plot.agnes`.

**See Also**

`diana, diana.object, twins.object, par`.

**Examples**

```r
eexample(diana)# -> dv <- diana(....)

plot(dv, which = 1, nmax.lab = 100)

## wider labels :
op <- par(mar = par("mar") + c(0, 2, 0,0))
plot(dv, which = 1, nmax.lab = 100, max.strlen = 12)
par(op)
```

---

**plot.mona**  
*Banner of Monothetic Divisive Hierarchical Clusterings*

**Description**

Creates the banner of a mona object.

**Usage**

```r
## S3 method for class 'mona'
plot(x, main = paste("Banner of ", deparse(x$call)),
     sub = NULL, xlab = "Separation step",
     col = c(2,0), axes = TRUE, adj = 0,
     nmax.lab = 35, max.strlen = 5, ...)
```
Arguments

- **x**: an object of class "mona", typically created by `mona(.)`.
- **main, sub**: main and sub titles for the plot, with convenient defaults. See documentation in `plot.default`.
- **xlab**: x axis label, see `title`.
- **col, adj**: graphical parameters passed to `bannerplot()`.
- **axes**: logical, indicating if (labeled) axes should be drawn.
- **nmax.lab**: integer indicating the number of labels which is considered too large for labeling.
- **max.strlen**: positive integer giving the length to which strings are truncated in labeling.
- **...**: further graphical arguments are passed to `bannerplot()` and `text`.

Details

Plots the separation step at which clusters are split. The observations are given in the order found by the `mona` algorithm, the numbers in the `step` vector are represented as bars between the observations.

When a long bar is drawn between two observations, those observations have the same value for each variable. See chapter 7 of Kaufman and Rousseeuw (1990).

Side Effects

A banner is plotted on the current graphics device.

Note

In the banner plot, observation labels are only printed when the number of observations is limited less than `nmax.lab` (35, by default), for readability. Moreover, observation labels are truncated to maximally `max.strlen` (5) characters.

References

see those in `plot.agnes`.

See Also

`mona`, `mona.object`, `par`
plot.partition

Plot of a Partition of the Data Set

Description

Creates plots for visualizing a partition object.

Usage

## S3 method for class 'partition'
plot(x, ask = FALSE, which.plots = NULL,
     nmax.lab = 40, max.strlen = 5, data = x$data, dist = NULL,
     stand = FALSE, lines = 2,
     shade = FALSE, color = FALSE, labels = 0, plotchar = TRUE,
     span = TRUE, xlim = NULL, ylim = NULL, main = NULL, ...)

Arguments

x an object of class "partition", typically created by the functions pam, clara, or fanny.
ask logical; if true and which.plots is NULL, plot.partition operates in interactive mode, via menu.
which.plots integer vector or NULL (default), the latter producing both plots. Otherwise, which.plots must contain integers of 1 for a clusplot or 2 for silhouette.
nmax.lab integer indicating the number of labels which is considered too large for single-name labeling the silhouette plot.
max.strlen positive integer giving the length to which strings are truncated in silhouette plot labeling.
data numeric matrix with the scaled data; per default taken from the partition object x, but can be specified explicitly.
dist when x does not have a diss component as for pam(*, keep.diss=FALSE), dist must be the dissimilarity if a clusplot is desired.
stand,lines,shade,color,labels,plotchar,span,xlim,ylim,main, ...

All optional arguments available for the clusplot.default function (except for the diss one) and graphical parameters (see par) may also be supplied as arguments to this function.

Details

When ask = TRUE, rather than producing each plot sequentially, plot.partition displays a menu listing all the plots that can be produced. If the menu is not desired but a pause between plots is still wanted, call par(ask = TRUE) before invoking the plot command.

The clusplot of a cluster partition consists of a two-dimensional representation of the observations, in which the clusters are indicated by ellipses (see clusplot.partition for more details).
The silhouette plot of a nonhierarchical clustering is fully described in Rousseeuw (1987) and in chapter 2 of Kaufman and Rousseeuw (1990). For each observation i, a bar is drawn, representing its silhouette width s(i), see silhouette for details. Observations are grouped per cluster, starting with cluster 1 at the top. Observations with a large s(i) (almost 1) are very well clustered, a small s(i) (around 0) means that the observation lies between two clusters, and observations with a negative s(i) are probably placed in the wrong cluster.

A clustering can be performed for several values of k (the number of clusters). Finally, choose the value of k with the largest overall average silhouette width.

**Side Effects**

An appropriate plot is produced on the current graphics device. This can be one or both of the following choices:

- Clusplot
- Silhouette plot

**Note**

In the silhouette plot, observation labels are only printed when the number of observations is less than max.lab (40, by default), for readability. Moreover, observation labels are truncated to maximally maxstrlen (5) characters.

For more flexibility, use plot(silhouette(x), ...), see plot.silhouette.

**References**


Further, the references in plot.agnes.

**See Also**

- partition.object
- clusplot.partition
- clusplot.default
- pam
- pam.object
- clara
- clara.object
- fanny
- fanny.object
- par

**Examples**

```r
## generate 25 objects, divided into 2 clusters.
x <- rbind(cbind(rnorm(10, 0, 0.5), rnorm(10, 0, 0.5)),
          cbind(rnorm(15, 5, 0.5), rnorm(15, 5, 0.5)))
plot(pam(x, 2))

## Save space not keeping data in clus.object, and still clusplot() it:
data(xclara)
xc <- clara(xclara, 3, keep.data = FALSE)
xc$data # is NULL
plot(xc, data = xclara)
```
pltree

Plot Clustering Tree of a Hierarchical Clustering

Description

pltree() Draws a clustering tree ("dendrogram") on the current graphics device. We provide the
	twins method draws the tree of a twins object, i.e., hierarchical clustering, typically resulting from
	agnes() or diana().

Usage

pltree(x, ...)  
  ## S3 method for class 'twins'
  pltree(x, main = paste("Dendrogram of ", deparse(x$call)),
          labels = NULL, ylab = "Height", ...)

Arguments

x     in general, an R object for which a pltree method is defined; specifically, an
main   object of class "twins", typically created by either agnes() or diana().
labels labels to use; the default is constructed from x.
ylab   label for y-axis.
...    graphical parameters (see par) may also be supplied as arguments to this func-

Details

Creates a plot of a clustering tree given a twins object. The leaves of the tree are the original
observations. In case of an agglomerative clustering, two branches come together at the distance
between the two clusters being merged. For a divisive clustering, a branch splits up at the diameter
of the cluster being splitted.

Note that currently the method function simply calls plot(as.hclust(x), ...), which dispatches
to plot.hclust(...). If more flexible plots are needed, consider xx <- as.dendrogram(as.hclust(x))
and plotting xx, see plot.dendrogram.

Value

a NULL value is returned.

See Also

agnes, agnes.object, diana, diana.object, hclust, par, plot.agnes, plot.diana.
Examples

data(votes.repub)
agn <- agnes(votes.repub)
pltree(agn)

dagn <- as.dendrogram(as.hclust(agn))
dagn2 <- as.dendrogram(as.hclust(agn), hang = 0.2)
op <- par(mar = par("mar") + c(0,0,0, 2)) # more space to the right
plot(dagn2, horiz = TRUE)
  plot(dagn, horiz = TRUE, center = TRUE,
       nodePar = list(lab.cex = 0.6, lab.col = "forest green", pch = NA),
       main = deparse(agn$call))
par(op)

pluton

Isotopic Composition Plutonium Batches

Description

The `pluton` data frame has 45 rows and 4 columns, containing percentages of isotopic composition of 45 Plutonium batches.

Usage

data(pluton)

Format

This data frame contains the following columns:

- **Pu238** the percentages of $^{238}$Pu, always less than 2 percent.
- **Pu239** the percentages of $^{239}$Pu, typically between 60 and 80 percent (from neutron capture of Uranium, $^{238}$U).
- **Pu240** percentage of the plutonium 240 isotope.
- **Pu241** percentage of the plutonium 241 isotope.

Details

Note that the percentage of plutonium-242 can be computed from the other four percentages, see the examples.

In the reference below it is explained why it is very desirable to combine these plutonium patches in three groups of similar size.

Source

Available as ‘pluton.dat’ from the archive of the University of Antwerpen, ‘...../datasets/clusplot-examples.tar.gz’, no longer available.
predict.ellipsoid

References

Examples
data(pluton)

```r
hist(apply(pluton,1,sum), col = "gray") # between 94% and 100%
pu5 <- pluton
pu5$Pu242 <- 100 - apply(pluton,1,sum) # the remaining isotope.
pairs(pu5)
```

---

**predict.ellipsoid**  
*Predict Method for Ellipsoid Objects*

**Description**
Compute points on the ellipsoid boundary, mostly for drawing.

**Usage**

```r
predict.ellipsoid(object, n.out=201, ...)
## S3 method for class 'ellipsoid'
predict(object, n.out=201, ...)
ellipsoidPoints(A, d2, loc, n.half = 201)
```

**Arguments**

- **object** an object of class ellipsoid, typically from `ellipsoidhull()`; alternatively any list-like object with proper components, see details below.
- **n.out, n.half** half the number of points to create.
- **A, d2, loc** arguments of the auxiliary `ellipsoidPoints`, see below.
- **...** passed to and from methods.

**Details**
Note `ellipsoidPoints` is the workhorse function of `predict.ellipsoid` a standalone function and method for ellipsoid objects, see `ellipsoidhull`. The class of object is not checked; it must solely have valid components `loc` (length `p`), the `p × p` matrix `cov` (corresponding to `A`) and `d2` for the center, the shape (“covariance”) matrix and the squared average radius (or distance) or `qchisq(*, p)` quantile.

Unfortunately, this is only implemented for `p = 2`, currently; contributions for `p ≥ 3` are very welcome.
print.agnes

Value

a numeric matrix of dimension 2*n.out times p.

See Also

ellipsoidhull, volume.ellipsoid.

Examples

## see also example(ellipsoidhull)

## Robust vs. L.S. covariance matrix
set.seed(143)
x <- rt(200, df=3)
y <- 3*x + rt(200, df=2)
plot(x,y, main="non-normal data (N=200)"
mtext("with classical and robust cov.matrix ellipsoids")
X <- cbind(x,y)
C.ls <- cov(X) ; m.ls <- colMeans(X)
d2.99 <- qchisq(0.99, df = 2)
lines(ellipsoidPoints(C.ls, d2.99, loc=m.ls), col="green")
if(require(MASS)) {
  Cxy <- cov.rob(cbind(x,y))
  lines(ellipsoidPoints(Cxy$cov, d2 = d2.99, loc=Cxy$center), col="red")
}# MASS

print.agnes

Print Method for AGNES Objects

Description

Prints the call, agglomerative coefficient, ordering of objects and distances between merging clusters ('Height') of an agnes object.

This is a method for the generic print() function for objects inheriting from class agnes, see agnes.object.

Usage

## S3 method for class 'agnes'
print(x, ...)

Arguments

x an agnes object.
...
potential further arguments (required by generic).

See Also

summary.agnes producing more output; agnes, agnes.object, print, print.default.
print.clara  

Print Method for CLARA Objects

Description

Prints the best sample, medoids, clustering vector and objective function of clara object.
This is a method for the function print() for objects inheriting from class clara.

Usage

## S3 method for class 'clara'
print(x, ...)

Arguments

x a clara object.

... potential further arguments (require by generic).

See Also

summary.clara producing more output; clara, clara.object, print, print.default.

print.diana  

Print Method for DIANA Objects

Description

Prints the ordering of objects, diameters of splitted clusters, and divisive coefficient of a diana object.
This is a method for the function print() for objects inheriting from class diana.

Usage

## S3 method for class 'diana'
print(x, ...)

Arguments

x a diana object.

... potential further arguments (require by generic).

See Also
diana, diana.object, print, print.default.
**print.dissimilarity**  
*Print and Summary Methods for Dissimilarity Objects*

**Description**

Print or summarize the distances and the attributes of a dissimilarity object.

These are methods for the functions `print()` and `summary()` for dissimilarity objects. See `print`, `print.default`, or `summary` for the general behavior of these.

**Usage**

```r
## S3 method for class 'dissimilarity'
print(x, diag = NULL, upper = NULL,
      digits =getOption("digits"), justify = "none", right = TRUE, ...)
## S3 method for class 'dissimilarity'
summary(object,
         digits = max(3, getOption("digits") - 2), ...)
## S3 method for class 'summary.dissimilarity'
print(x, ...)
```

**Arguments**

- `x`, `object` a dissimilarity object or a `summary.dissimilarity` one for `print.summary.dissimilarity()`.
- `digits` the number of digits to use, see `print.default`.
- `diag, upper, justify, right` optional arguments specifying how the triangular dissimilarity matrix is printed; see `print.dist`.
- `...` potential further arguments (require by generic).

**See Also**

daisy, dissimilarity.object, print, print.default, print.dist.

**Examples**

```r
## See example(daisy)

sd <- summary(daisy(matrix(rnorm(100), 20, 5)))
sd # -> print.summary.dissimilarity(.)
str(sd)
```
**Description**

Prints the objective function, membership coefficients and clustering vector of fanny object. This is a method for the function `print()` for objects inheriting from class `fanny`.

**Usage**

```r
## S3 method for class 'fanny'
print(x, digits =getOption("digits"), ...)
## S3 method for class 'fanny'
summary(object, ...)
## S3 method for class 'summary.fanny'
print(x, digits =getOption("digits"), ...)
```

**Arguments**

- `x`: object a fanny object.
- `digits`: number of significant digits for printing, see `print.default`.
- `...`: potential further arguments (required by generic).

**See Also**

`fanny`, `fanny.object`, `print`, `print.default`.

---

**print.mona**

*Print Method for MONA Objects*

**Description**

Prints the ordering of objects, separation steps, and used variables of a mona object. This is a method for the function `print()` for objects inheriting from class `mona`.

**Usage**

```r
## S3 method for class 'mona'
print(x, ...)
```

**Arguments**

- `x`: a mona object.
- `...`: potential further arguments (require by generic).
See Also

`mona`, `mona.object`, `print`, `print.default`.

---

**print.pam**

*Print Method for PAM Objects*

**Description**

Prints the medoids, clustering vector and objective function of `pam` object.

This is a method for the function `print()` for objects inheriting from class `pam`.

**Usage**

```r
## S3 method for class 'pam'
print(x, ...)
```

**Arguments**

- `x` a `pam` object.
- `...` potential further arguments (require by generic).

**See Also**

`pam`, `pam.object`, `print`, `print.default`.

---

**ruspini**

*Ruspin Data*

**Description**

The Ruspini data set, consisting of 75 points in four groups that is popular for illustrating clustering techniques.

**Usage**

```r
data(ruspini)
```

**Format**

A data frame with 75 observations on 2 variables giving the x and y coordinates of the points, respectively.

**Source**

References

see those in agnes.

Examples

data(ruspini)

## Plot similar to Figure 4 in Stryuf et al (1996)
## Not run: plot(pam(ruspini, 4), ask = TRUE)

## Plot similar to Figure 6 in Stryuf et al (1996)
plot(fanny(ruspini, 5))

silhouette  Compute or Extract Silhouette Information from Clustering

Description

Compute silhouette information according to a given clustering in \( k \) clusters.

Usage

silhouette(x, ...)

# Default S3 method:
silhouette(x, dist, dmatrix, ...)
# S3 method for class 'partition'
silhouette(x, ...)
# S3 method for class 'clara'
silhouette(x, full = FALSE, ...)

sortSilhouette(object, ...)
# S3 method for class 'silhouette'
summary(object, FUN = mean, ...)
# S3 method for class 'silhouette'
plot(x, nmax.lab = 40, max.strlen = 5,
     main = NULL, sub = NULL, xlab = expression("Silhouette width \(^{n[i]}\)),
     col = "gray", do.col.sort = length(col) > 1, border = 0,
     cex.names = par("cex.axis"), do.n.k = TRUE, do.clus.stat = TRUE, ...)

Arguments

x an object of appropriate class; for the default method an integer vector with \( k \) different integer cluster codes or a list with such an x$clustering component.
Note that silhouette statistics are only defined if \( 2 \leq k \leq n - 1 \).

dist a dissimilarity object inheriting from class dist or coercible to one. If not specified, dmatrix must be.
**dmatrix**  
a symmetric dissimilarity matrix \((n \times n)\), specified instead of `dist`, which can be more efficient.

**full**  
logical specifying if a full silhouette should be computed for `clara` object. Note that this requires \(O(n^2)\) memory, since the full dissimilarity (see `daisy`) is needed internally.

**object**  
an object of class `silhouette`.

...  
进一步的参数传递给方法。

**FUN**  
function used to summarize silhouette widths.

**nmax.lab**  
integer indicating the number of labels which is considered too large for single-name labeling the silhouette plot.

**max.strlen**  
positive integer giving the length to which strings are truncated in silhouette plot labeling.

**main, sub, xlab**  
arguments to `title`; have a sensible non-NULL default here.

**col, border, cex.names**  
arguments passed to `barplot`; note that the default used to be `col = heat.colors(n), border = par$bg` instead.

col can also be a color vector of length \(k\) for clusterwise coloring, see also `do.col.sort`.

**do.col.sort**  
logical indicating if the colors col should be sorted “along” the silhouette; this is useful for casewise or clusterwise coloring.

**do.n.k**  
logical indicating if \(n\) and \(k\) “title text” should be written.

**do.clus.stat**  
logical indicating if cluster size and averages should be written right to the silhouettes.

**Details**

For each observation \(i\), the **silhouette width** \(s(i)\) is defined as follows:

Put \(a(i)\) = average dissimilarity between \(i\) and all other points of the cluster to which \(i\) belongs (if \(i\) is the only observation in its cluster, \(s(i) := 0\) without further calculations). For all other clusters \(C\), put \(d(i, C)\) = average dissimilarity of \(i\) to all observations of \(C\). The smallest of these \(d(i, C)\) is \(b(i) := \min_C d(i, C)\), and can be seen as the dissimilarity between \(i\) and its “neighbor” cluster, i.e., the nearest one to which it does not belong. Finally,

\[
s(i) := \frac{b(i) - a(i)}{\max(a(i), b(i))}.
\]

`silhouette.default()` is now based on C code donated by Romain Francois (the R version being still available as `cluster::silhouette.default.R`).

Observations with a large \(s(i)\) (almost 1) are very well clustered, a small \(s(i)\) (around 0) means that the observation lies between two clusters, and observations with a negative \(s(i)\) are probably placed in the wrong cluster.
Value

silhouette() returns an object, sil, of class silhouette which is an \( n \times 3 \) matrix with attributes. For each observation \( i \), \( \text{sil}[i,] \) contains the cluster to which \( i \) belongs as well as the neighbor cluster of \( i \) (the cluster, not containing \( i \), for which the average dissimilarity between its observations and \( i \) is minimal), and the silhouette width \( s(i) \) of the observation. The \texttt{colnames} correspondingly are \texttt{c("cluster", "neighbor", "sil_width")}.

\texttt{summary(sil)} returns an object of class \texttt{summary.silhouette}, a list with components

\begin{itemize}
  \item \texttt{si.summary}: numerical \texttt{summary} of the individual silhouette widths \( s(i) \).
  \item \texttt{clus.avg.widths}: numeric (rank 1) array of clusterwise \textit{means} of silhouette widths where \texttt{mean = FUN} is used.
  \item \texttt{avg.width}: the total mean \texttt{FUN(s)} where \( s \) are the individual silhouette widths.
  \item \texttt{clus.sizes}: \texttt{table} of the \( k \) cluster sizes.
  \item \texttt{call}: if available, the \texttt{call} creating \texttt{sil}.
\end{itemize}

\texttt{Ordered}: logical identical to \texttt{attr(sil, "Ordered")}, see below.

\texttt{sortSilhouette(sil)} orders the rows of \texttt{sil} as in the silhouette plot, by cluster (increasingly) and decreasing silhouette width \( s(i) \).

\texttt{attr(sil, "Ordered")} is a logical indicating if \texttt{sil} is ordered as by \texttt{sortSilhouette()}. In that case, \texttt{rownames(sil)} will contain case labels or numbers, and \texttt{attr(sil, "iOrd")} the ordering index vector.

Note

While \texttt{silhouette()} is \textit{intrinsic} to the \texttt{partition} clusterings, and hence has a (trivial) method for these, it is straightforward to get silhouettes from hierarchical clusterings from \texttt{silhouette.default()} with \texttt{cutree()} and distance as input.

By default, for \texttt{clara()} partitions, the silhouette is just for the best random \textit{subset} used. Use \texttt{full = TRUE} to compute (and later possibly plot) the full silhouette.

References


chapter 2 of Kaufman and Rousseeuw (1990), see the references in \texttt{plot.agnes}.

See Also

\texttt{partition.object, plot.partition}.

Examples

\begin{verbatim}
data(ruspini)
pr4 <- pam(ruspini, 4)
str(si <- silhouette(pr4))
(ssi <- summary(si))
plot(si) # silhouette plot
\end{verbatim}
The `sizeDiss` function returns the number of observations (sample size) corresponding to a dissimilarity like object, or equivalently, the number of rows or columns of a matrix when only the lower or upper triangular part (without diagonal) is given.
It is nothing else but the inverse function of \( f(n) = \frac{n(n - 1)}{2} \).

Usage

```r
sizeDiss(d)
```

Arguments

d any \( \mathbb{R} \) object with length (typically) \( n(n - 1)/2 \).

Value

a number; \( n \) if `length(d) == n(n-1)/2`, NA otherwise.

See Also

dissimilarity.object and also as.dist for class dissimilarity and dist objects which have a Size attribute.

Examples

```r
sizeDiss(1:10) # 5, since 10 == 5 * (5 - 1) / 2
sizeDiss(1:9) # NA

n <- 1:100
stopifnot(n == sapply(n*(n-1)/2, function(n) sizeDiss(logical(n))))
```

---

**summary.agnes**

*Summary Method for 'agnes' Objects*

**Description**

Returns (and prints) a summary list for an agnes object. Printing gives more output than the corresponding `print.agnes` method.

**Usage**

```r
## S3 method for class 'agnes'
summary(object, ...)
## S3 method for class 'summary.agnes'
print(x, ...)
```

**Arguments**

x, object a `agnes` object.

... potential further arguments (require by generic).
summary.clara

See Also

agnes, agnes.object.

Examples

data(agriculture)
summary(agnes(agriculture))

summary.clara

Summary Method for 'clara' Objects

Description

Returns (and prints) a summary list for a clara object. Printing gives more output than the corresponding print.clara method.

Usage

## S3 method for class 'clara'
summary(object, ...)
## S3 method for class 'summary.clara'
print(x, ...)

Arguments

x, object  a clara object.
...  potential further arguments (require by generic).

See Also

clara.object

Examples

## generate 2000 objects, divided into 5 clusters.
set.seed(47)
x <- rbind(cbind(rnorm(400, 0, 4), rnorm(400, 0, 4)),
cbind(rnorm(400,10,8), rnorm(400,40,6)),
cbind(rnorm(400,30,4), rnorm(400, 0,4)),
cbind(rnorm(400,40,4), rnorm(400,20,2)),
cbind(rnorm(400,50,4), rnorm(400,50,4))
)
clx5 <- clara(x, 5)
## Mis'classification' table:

table(rep(1:5, rep(400,5)), clx5$clust)  # -> 1 "error"
summary(clx5)
summary.diana

Summary Method for 'diana' Objects

## Description

Returns (and prints) a summary list for a diana object.

## Usage

```r
## S3 method for class 'diana'
summary(object, ...)
## S3 method for class 'summary.diana'
print(x, ...)
```

## Arguments

- `x, object`: a diana object.
- `...`: potential further arguments (require by generic).

## See Also

diana, diana.object.

summary.mona

Summary Method for 'mona' Objects

## Description

Returns (and prints) a summary list for a mona object.

## Usage

```r
## S3 method for class 'mona'
summary(object, ...)
## S3 method for class 'summary.mona'
print(x, ...)
```
Arguments

`x`, object  a `mona` object.

...  potential further arguments (require by generic).

See Also

`mona`, `mona.object`.

summary.pam  Summary Method for PAM Objects

Description

Summarize a `pam` object and return an object of class `summary.pam`. There’s a `print` method for the latter.

Usage

```r
## S3 method for class 'pam'
summary(object, ...)
## S3 method for class 'summary.pam'
print(x, ...)
```

Arguments

`x`, object  a `pam` object.

...  potential further arguments (require by generic).

See Also

`pam`, `pam.object`.

twins.object  Hierarchical Clustering Object

Description

The objects of class "twins" represent an agglomerative or divisive (polythetic) hierarchical clustering of a dataset.

Value

See `agnes.object` and `diana.object` for details.
**volume.ellipsoid**

**GENERATION**

This class of objects is returned from agnes or diana.

**METHODS**

The "twins" class has a method for the following generic function: pltree.

**INHERITANCE**

The following classes inherit from class "twins": "agnes" and "diana".

**See Also**

agnes, diana.

---

**volume.ellipsoid**

*Compute the Volume of Planar Object*

**Description**

Compute the volume of a planar object. This is a generic function and a method for ellipsoid objects.

**Usage**

```r
## S3 method for class 'ellipsoid'
volume(object)
```

**Arguments**

- `object`: an R object the volume of which is wanted; for the ellipsoid method, an object of that class (see `ellipsoidhull` or the example below).

**Value**

A number, the volume of the given object.

**See Also**

`ellipsoidhull` for spanning ellipsoid computation.

**Examples**

```r
## example(ellipsoidhull) # which defines 'ellipsoid' object <namefoo>
myEl <- structure(list(cov = rbind(c(3,1),1:2), loc = c(0,0), d2 = 10),
  class = "ellipsoid")
volume(myEl)# i.e. "area" here (d = 2)
myEl # also mentions the "volume"```
Description

A data frame with the percents of votes given to the republican candidate in presidential elections from 1856 to 1976. Rows represent the 50 states, and columns the 31 elections.

Usage

data(votes.repub)

Source


Data from 1964 to 1976 is from R. M. Scammon, American Votes 12, Congressional Quarterly.

Description

An artificial data set consisting of 3000 points in 3 quite well-separated clusters.

Usage

data(xclara)

Format

A data frame with 3000 observations on 2 numeric variables (named V1 and V2) giving the $x$ and $y$ coordinates of the points, respectively.

Note

Our version of the xclara is slightly more rounded than the one from read.table("xclara.dat") and the relative difference measured by all.equal is 1.15e-7 for V1 and 1.17e-7 for V2 which suggests that our version has been the result of a options(digits = 7) formatting.

Previously (before May 2017), it was claimed the three cluster were each of size 1000, which is clearly wrong. pam(*, 3) gives cluster sizes of 899, 1149, and 952, which apart from seven “outliers” (or “mislabellings”) correspond to observation indices {1 : 900}, {901 : 2050}, and {2051 : 3000}, see the example.
Source

Sample data set accompanying the reference below (file `xclara.dat` in side `clus_examples.tar.gz`).

References


Examples

```r
## Visualization: Assuming groups are defined as {1:1000}, {1001:2000}, {2001:3000}
plot(xclara, cex = 3/4, col = rep(1:3, each=1000))
p.ID <- c(78, 1411, 2535) ## PAM's medoid indices == pam(xclara, 3)$id.med
text(xclara[p.ID,], labels = 1:3, cex=2, col=1:3)

px <- pam(xclara, 3) ## takes ~2 seconds
cxcl <- px$clustering ; iCl <- split(seq_along(cxcl), cxcl)
boxplot(iCl, range = 0.7, horizontal=TRUE, 
    main = "Indices of the 3 clusters of pam(xclara, 3)")

## Look more closely now:
bxCl <- boxplot(iCl, range = 0.7, plot=FALSE)
## We see 3 + 2 + 2 = 7 clear "outliers" or "wrong group" observations:
with(bxCl, rbind(out, group))
## out 1038 1451 1610 30 327 562 770
## group 1 1 1 2 2 3 3
## Apart from these, what are the robust ranges of indices? -- Robust range:
t(iR <- bxCl$stats[c(1,5),])
## 1 900
## 901 2050
## 2051 3000
gc <- adjustcolor("gray20",1/2)
abline(v = iR, col = gc, lty=3)
axis(3, at = c(0, iR[2,]), padj = 1.2, col=gc, col.axis=gc)
```
Index

*Topic arith
  sizeDiss, 72
*Topic array
  lower.to.upper.tri.ind, 42
*Topic cluster
  agnes, 3
  agnes.object, 7
  bannerplot, 11
  clara, 13
  clara.object, 16
  clusGap, 17
  clusplot, 21
  clusplot.default, 23
  coef.hclust, 27
  daisy, 28
  diana, 32
  dissimilarity.object, 34
  fanny, 37
  fanny.object, 40
  mona, 43
  mona.object, 45
  pam, 45
  pam.object, 49
  partition.object, 50
  plot.agnes, 54
  plot.diana, 56
  plot.mona, 57
  plot.partition, 59
  pltree, 61
  print.agnes, 64
  print.clara, 65
  print.diana, 65
  print.dissimilarity, 66
  print.fanny, 67
  print.mona, 67
  print.pam, 68
  silhouette, 69
  summary.agnes, 73
  summary.clara, 74
  summary.diana, 75
  summary.mona, 75
  twins.object, 76
*Topic datasets
  agriculture, 9
  animals, 10
  chorSub, 12
  flower, 41
  plantTraits, 52
  pluton, 62
  ruspini, 68
  votes.repub, 78
  xclara, 78
*Topic dplot
  ellipsoidhull, 35
  predict.ellipsoid, 63
*Topic hplot
  bannerplot, 11
  clusplot, 21
  clusplot.default, 23
  ellipsoidhull, 35
  plot.agnes, 54
  plot.diana, 56
  plot.mona, 57
  plot.partition, 59
  pltree, 61
*Topic print
  print.agnes, 64
  print.clara, 65
  print.diana, 65
  print.dissimilarity, 66
  print.fanny, 67
  print.mona, 67
  print.pam, 68
  summary.agnes, 73
  summary.clara, 74
  summary.diana, 75
  summary.mona, 75
INDEX

*Topic **utilities**
  - bannerplot, 11
  - lower.to.upper.tri.ind, 42
  - predict.ellipsoid, 63
  - sizeDiss, 72
  - volume.ellipsoid, 77
- Machine, 48
- agnes, 3, 8, 9, 11, 15, 28, 31, 34, 35, 39, 42, 44, 48, 54, 55, 61, 64, 69, 73, 74, 77
  - agnes.object, 4–6, 7, 28, 55, 61, 64, 76
- agriculture, 9
- all.equal, 78
- animals, 10
- arrows, 19
  - as.dendrogram, 8, 33, 55, 61
  - as.dist, 73
  - as.hclust, 8, 33, 34, 61
  - as.matrix, 35
- bannerplot, 11, 54–56, 58
  - barplot, 11, 70
- call, 19, 50, 71
- character, 18
- chorizon, 12, 13
- chorSub, 12
- chull, 37
- clara, 6, 13, 16, 17, 22, 24, 26, 31, 35, 48, 51, 59, 60, 65, 70, 71, 74
  - clara.object, 15, 16, 22, 51, 60, 65, 74
- clusGap, 17
- clusplot, 14, 21, 37
- clusplot.default, 22, 23, 36, 59, 60
- clusplot.partition, 26, 59, 60
- cluster.stats, 20
- cmdscale, 25, 26
- coef.hclust, 27
  - coef.twins (coef.hclust), 27
- coefHier (coef.hclust), 27
- colnames, 71
- cov.mve, 26, 36, 37
- cutree, 8, 34, 71
- daisy, 3, 6, 9, 24, 26, 28, 32, 34, 35, 38, 39, 46, 48, 66, 70
- data.frame, 18
- diana, 5, 6, 8, 9, 27, 28, 31, 32, 35, 56, 57, 61, 65, 73, 77
- diana.object, 28, 57, 61, 65, 75, 76
- dissimilarity.object, 8, 17, 30, 31, 34, 41, 50, 66, 73
- dist, 3, 6, 18, 24, 29, 31, 32, 34, 35, 38, 39, 46, 48, 69
- ellipse, 36, 37
- ellipsePoints, 37
- ellipsoid.hull, 35, 63, 64, 77
- ellipsoidPoints (predict.ellipsoid), 63
- fanny, 6, 22, 24, 31, 35, 37, 41, 51, 59, 60, 67
- fanny.object, 22, 39, 40, 51, 60, 67
- flower, 41
- function, 18
- hclust, 6, 8, 28, 61
- identify, 24, 26
- list, 18, 24, 36, 49
- lower.to.upper.tri.ind, 42
- lower.tri, 42
- match.arg, 19
- matrix, 49
- maxSE (clusGap), 17
- menu, 54, 56, 59
- mona, 5, 43, 45, 58, 67, 68, 76
- mona.object, 44, 45, 58, 68, 76
- NA, 23, 29–32, 43, 44, 46
- names, 24
- options, 78
- pam, 6, 14, 15, 22, 24, 26, 31, 35, 45, 50, 51, 59, 60, 68, 76, 78
- pam.object, 22, 47, 48, 49, 51, 60, 68, 76
- par, 12, 22, 25, 26, 54–61
- partition, 71
- partition (partition.object), 50
- partition.object, 15, 17, 22, 39–41, 48–50, 50, 60, 71
- plantTraits, 52
- plot.agnes, 4, 6, 8, 12, 54, 57, 58, 60, 61, 71
- plot.clusGap (clusGap), 17
- plot.default, 19, 25, 54, 56, 58
- plot.dendrogram, 55, 61
- plot.diana, 12, 34, 56, 61
plot.hclust, 61
plot.mona, 12, 44, 45, 57
plot.partition, 15, 17, 39, 41, 48, 50, 51, 59, 71
plot.silhouette, 60
plot.silhouette(silhouette), 69
pltree, 8, 33, 56, 61
pltree.twins, 54, 55
pluton, 62
predict, 37
predict.ellipsoid, 37, 63
pretty, 12, 54, 56
princomp, 25, 26
print, 36, 64–68, 76
print.agnes, 64, 73
print.clara, 65, 74
print.clusGap(clusGap), 17
print.default, 64–68
print.diana, 65
print.dissimilarity, 66
print.dist, 66
print.ellipsoid(ellipsoidhull), 35
print.fanny, 41, 67
print.mona, 67
print.pam, 68
print.summary.agnes(summary.agnes), 73
print.summary.clara(summary.clara), 74
print.summary.diana(summary.diana), 75
print.summary.dissimilarity
  (print.dissimilarity), 66
print.summary.fanny(print.fanny), 67
print.summary.mona(summary.mona), 75
print.summary.pam(summary.pam), 76
print.summary.silhouette(silhouette), 69

qchisq, 63

range, 29
read.table, 78
ruspini, 68

set.seed, 15, 19
silhouette, 20, 47, 51, 60, 69
sizeDiss, 72
sortSilhouette(silhouette), 69

summary.diana, 75
summary.dissimilarity
  (print.dissimilarity), 66
summary.fanny(print.fanny), 67
summary.mona, 75
summary.pam, 76
summary.silhouette(silhouette), 69
svd, 17

table, 71
text, 58
title, 11, 25, 58, 70
twins(twins.object), 76
twins.object, 6, 8, 34, 57, 76

upper.to.lower.tri,inds
  (lower.to.upper.tri,inds), 42
upper.tri, 42

volume(volume.ellipsoid), 77
volume.ellipsoid, 37, 64, 77
votes.repub, 78

xclara, 78