# Package: cba (via r-universe)

February 13, 2025

**Type** Package

**Title** Clustering for Business Analytics

**Version** 0.2-25

**Description** Implements clustering techniques such as Proximus and Rock, utility functions for efficient computation of cross distances and data manipulation.

**Depends** R (>= 3.4.0), grid, proxy

**Imports** stats, graphics, grDevices, methods

**Suggests** gclus, colorspace

**Encoding** UTF-8

**License** GPL-2

**NeedsCompilation** yes

**Date/Publication** 2024-08-16 17:59:19 UTC

**Author** Christian Buchta [aut, cre], Michael Hahsler [aut]

**Maintainer** Christian Buchta <christian.buchta@wu.ac.at>

**Repository** https://mhahsler.r-universe.dev

**RemoteUrl** https://github.com/cran/cba

**RemoteRef** HEAD

**RemoteSha** 91d4934548ce14f13b3eae6a2f3b2669f4c25338

## Contents

---

`ccfkms` *Clustering with Conjugate Convex Functions*

---

## Description

Partition a data set into convex sets using conjugate convex functions.

## Usage

```
ccfkms(x, n, p = NULL, par = 2, max.iter = 100, opt.std = FALSE,
       opt.retry = 0, debug = FALSE)
```

## Arguments

| | |
|---|---|
| `x` | a data matrix. |
| `n` | optional number of prototypes. |
| `p` | a matrix of initial prototypes. |
| `par` | type or parameter of conjugate convex function. |
| `max.iter` | maximum number of iterations. |
| `opt.std` | optionally standardize the data. |
| `opt.retry` | number of retries. |
| `debug` | optionally turn on debugging output. |

## Details

Two types of conjugate convex functions are available: one that is based on powers of the norm of the prototype vectors and another that is based on a logarithmic transformation of the norm. Both are intended to obtain more robust partitions.

Using `par = 2` is equivalent to performing ordinary k-means with Euclidean distances. `par = 1` is equivalent to LVQ of Kohonen type (the directions of the prototypes from the center of the data are used), and `par = 0` is equivalent to using 2*ln(cosh(|pl))/2.

Internally the algorithm uses sparse data structures and avoids computations with zero data values. Thus, the data must not be centered (the algorithm does this internally with the option to further standardize the data). For dense data this is slightly inefficient.

If initial prototypes are omitted the number of prototypes must be specified. In this case the initial prototypes are drawn from the data (without replacement).

If the number of retries is greater than zero the best among that number of trial solutions is returned. Note that the number of prototypes must be specified as the initial prototypes are sampled from the data.

The debugging output shows the iteration number, the inverted information and the variance of the current partition as a percentage of the total (if each data point were a cluster), and the number of active prototypes (those with at least one member, i.e. a data point that is not closer to any other prototype).

Note that the algorithm uses tie-breaking when it determines the cluster memberships of the samples.

## Value

A list with the following components:

| | |
|---|---|
| `centers` | a matrix of cluster means (final prototypes). |
| `size` | a vector of cluster sizes. |
| `cl` | a factor of cluster labels (indexes). |
| `inv.inf` | the inverted information of the partition. |
| `par` | see above. |
| `opt.std` | see above. |

## Note

Support for data matrices `x` in sparse `dgTMatrix` and `dgCMatrix` format (see package **Matrix**) is experimental. Support for the `dgRMatrix` format is currently suspended due to problems with package **Matrix**.

## Author(s)

Christian Buchta

## References

Helmut Strasser and Klaus Poetzelberger. Data Compression by Unsupervised Classification. *SFB Report Series*, No. 10, 1997.

## See Also

kmeans, cmeans, kkmeans for similar or related clustering techniques.

## Examples

```
### extend proximus example
x <- rlbmat()
rownames(x) <- seq(dim(x)[1])
cm <- ccfkms(x, n=4, opt.retry=10)
pcm <- predict(cm, x)
## Not run:
### using sparse data may be more time-efficient
### depending on the goodness of the implementation
### of subset, etc. in package Matrix.
require(Matrix)
#sx <- as(x, "dgRMatrix")    # currently broken
sx <- as(x, "dgCMatrix")
system.time(scm <- ccfkms(sx, n=4, opt.retry=50))
system.time(cm <- ccfkms(x, n=4, opt.retry=50))

## End(Not run)
```

---

# circleplot.dist *Plotting Distance Graphs*

---

## Description

Function for visualizing distance graphs using a circular layout.

## Usage

```
circleplot.dist(x, cutoff = 0.5, col = 1, circle = FALSE,
                scale = 1.4)
```

## Arguments

| | |
|---|---|
| x | an object of class dist. |
| cutoff | a numeric value specifying the threshold for edge exclusion. |
| col | a number or string specifying the edge color to use. |
| circle | a logical value specifying if a circle connecting the nodes should be drawn. |
| scale | a numeric value specifying the plot range (the default accommodates node labels). |

## Details

Plots the distance graph of x placing its nodes on a circle such that the number of crossing edges is approximately minimized. This is achieved by using order.dist for seriation.

## Author(s)

Christian Buchta

## See Also

order.dist.

## Examples

```
##
data(iris)
d <- dist(iris[,-5])[[1:26]]
circleplot.dist(d, col = 2, scale = 1)

dimnames(d) <- LETTERS[1:26]
circleplot.dist(d)
```

---

clmplot *Plotting Logical Matrices*

---

## Description

A wrapper function to image that produces a level plot with the option to color the rows (or columns) based on a clustering and/or classification of the data, and the option to reorder the rows and columns for better presentation.

## Usage

```
clmplot(x, col, col.bycol = FALSE, order=FALSE,
        dist.method = "binary", hclust.method = "average",
        axes = FALSE, xlab = "", ylab = "", ...)
```

## Arguments

| | |
|---|---|
| x | an logical matrix. |
| col | an optional vector defining a coloring. |
| col.bycol | option to color by columns. |
| order | option to (pre)order the rows and columns. |
| dist.method | method to be used by dist. |
| hclust.method | method to be used by hclust. |
| axes | option to plot axes. |
| xlab, ylab | labels for the x and y axis. |
| ... | further arguments to image. |

## Details

For dummy coded data the level FALSE is assumed to carry no information and is colored white. Thus, the level TRUE can be colored according to some classification of the rows (or columns) of the data matrix. If no color specification is supplied the default color black is used.

If col is of type character it is assumed to contain color codes. Otherwise, it must be a factor and the levels are assigned colors from heat.colors.

If order is TRUE the rows and columns are ordered by hclust where the distances are computed by dist.

Note that an axis is only plotted if the number of elements (rows or columns) is less than 100.

## Value

A list with the following components:

| | |
|---|---|
| rows | the row order. |
| cols | the column order. |

## Author(s)

Christian Buchta

## See Also

lmplot for plotting of logical data at reduced resolutions, heatmap for ordered plotting of real-valued data, and package **gclus** for ordering functions.

## Examples

```
x <- matrix(sample(c(FALSE,TRUE),100,rep=TRUE),ncol=10)
clmplot(x, order=TRUE, axes=TRUE)
clmplot(x, col=rep(c(1,2),each=5))
clmplot(x, col=rep(c("red","blue"),each=5))
clmplot(x, col=rep(c("red","blue"),each=5), col.bycol=TRUE)
## Not run:
example("rockCluster")
### continue example (see rockCluster)
col <- Votes$Class                                  # color by party
levels(col) <- c("red","blue")
op <- par(mfrow=c(1,2), pty="s")
clmplot(x, order=TRUE, col=as.character(col), main="Parties")
col <- rf$cl                                        # color by cluster
levels(col) <- c("blue","red","green", "black")     # map NA to black
clmplot(x, order=TRUE, col=as.character(col), main="Clusters")
par(op)

## End(Not run)
```

`cluster.dist` *Clustering a Sparse Symmetric Distance Matrix*

## Description

Compute a clustering on a sparse symmetric distance matrix using graph cutting.

## Usage

```
cluster.dist(x, beta)
```

## Arguments

| | |
|---|---|
| `x` | an object of class `dist`. |
| `beta` | the distance threshold. |

## Details

This function computes a clustering on an object of class `dist` by cutting the graph induced by the threshold `beta` into all disconnected subgraphs (the clusters). Two nodes are connected by a link if their distance is below the specified threshold. Note that the threshold is not strict, i.e. >=.

Note that distances of value `NA` and `NaN` are ignored. This is not strictly correct but avoids computing $2^k$ possible solutions if there are k `NA` values.

The time complexity is $O(n^2)$ with n the number of rows/columns.

## Value

A factor of cluster labels (indexed 1,2,...,k).

## Note

Fixme: can the time complexity be improved?

## Author(s)

Christian Buchta

## See Also

`dist` and `sdists` for distance computation.

## Examples

```
## 3 clusters (1 = connected)
x <- matrix(c(1,1,0,0,0,0,
       1,1,0,0,0,0,
       0,0,1,1,0,0,
       0,0,1,1,0,0,
       0,0,0,0,1,1,
       0,0,0,0,1,1), ncol=6)
c <- cluster.dist(as.dist(!x), beta = 0) # invert and note that 0 >= 0
c
```

---

`coding` *Dummy Coding*

---

## Description

Functions that convert R objects to a dummy coded matrix (or a list of matrices).

## Usage

```
as.dummy(x, ...)

## S3 method for class 'matrix'
as.dummy(x, sep = " ", drop = FALSE, ...)
## S3 method for class 'data.frame'
as.dummy(x, sep = " ", drop = FALSE, ...)
```

## Arguments

| | |
|---|---|
| `x` | an R object (see below). |
| `sep` | separator used for construction of colnames. |
| `drop` | drop factors with less than two levels. |
| `...` | other (unused) arguments. |

## Details

The generic is applicable to factor and to R objects that can be converted to factor, i.e. `logical`, `integer`, or `character`. For numeric data a discretization method has to be used.

A factor is converted to as many logical variables as there are levels where the value `TRUE` indicates the presence of a level.

The colnames are made of the concatenation of a variable name and the level, separated by `sep`. For `matrix` and `data.frame` variable names are created if necessary.

A value of `NA` is mapped to `FALSE` across all levels.

## Value

A `matrix` with a `levels` attribute which contains a list of the levels of the coded variables.

### Warning

This is experimental code which may change in the future.

### Author(s)

Christian Buchta

### See Also

as.logical.

### Examples

```
###
x <- as.integer(sample(3,10,rep=TRUE))
as.dummy(x)
is.na(x) <- c(3,5)
as.dummy(x)
x <- as.data.frame(x)
as.dummy(x)
```

---

## cut.ordered *Converting Ordered Factors*

---

### Description

Reduce the levels of an ordered factor.

### Usage

```
## S3 method for class 'ordered'
cut(x, breaks, ...)
```

### Arguments

| | |
|---|---|
| x | an ordered factor. |
| breaks | a logical, character, or index vector of cut points. |
| ... | further (unused) arguments. |

### Details

If breaks is of class logical it must have the same length as the number of levels of x.

If breaks is of class character partial matching with the levels of x is attempted.

Otherwise breaks is assumed to index the levels.

### Value

An ordered factor.

### Author(s)

Christian Buchta

### References

Functions Missing in R: A Never Ending Story ;-)

### See Also

`cut` for converting numeric vectors to factor.

### Examples

```
x <- ordered(sample(letters[1:3],10,rep=TRUE))
cut(x, c(FALSE,TRUE,FALSE))
cut(x, "b")
cut(x, 2)
```

---

## fitted.proximus *Extract from a Proximus Object*

---

### Description

Get the full storage representation of the approximated matrix and the pattern labels of the original data samples from an object of class `proximus`.

### Usage

```
## S3 method for class 'proximus'
fitted(object, drop = FALSE, ...)
```

### Arguments

| | |
|---|---|
| `object` | an object of class `proximus`. |
| `drop` | optionally drop patterns that do not meet the mining criteria. |
| `...` | further (unused) arguments. |

### Details

If option `drop` is `TRUE` only patterns that satisfy the minimum size and maximum radius constraint are extracted.

### Value

A list with the following components:

| | |
|---|---|
| `x` | the fitted data matrix. |
| `pl` | a factor of pattern (cluster) labels. The indexes of the original data samples are provided as attribute `Index`. |

### Author(s)

Christian Buchta

### See Also

proximus for pattern mining with the Proximus algorithm.

### Examples

```
### see proximus
```

---

## gknn *Generalized k-Nearest Neighbor Classification*

---

### Description

Compute the k-nearest neighbor classification given a matrix of cross-distances and a factor of class values. For each row the majority class is found, where ties are broken at random (default). If there are ties for the kth nearest neighbor, all candidates are included in the vote (default).

### Usage

```
gknn(x, y, k = 1, l = 0, break.ties = TRUE, use.all = TRUE,
     prob = FALSE)
```

### Arguments

| | |
|---|---|
| x | a cross-distances matrix. |
| y | a factor of class values of the columns of x. |
| k | number of nearest neighbors to consider. |
| l | minimum number of votes for a definite decision. |
| break.ties | option to break ties. |
| use.all | option to consider all neighbors that are tied with the kth neighbor. |
| prob | optionally return proportions of winning votes. |

### Details

The rows of the cross-distances matrix are interpreted as referencing the test samples and the columns as referencing the training samples.

The options are fashioned after knn in package **class** but are extended for tie breaking of votes, e.g. if only definite (majority) votes are of interest.

Missing class values are not allowed because that would collide with a missing classification result.

Missing distance values are ignored but with the possible consequence of missing classification results. Note that this depends on the options settings, e.g.

## Value

Returns a factor of class values (of the rows of x) which may be NA in the case of doubt (no definite decision), ties, or missing neighborhood information.

The proportions of winning votes are returned as attribute prob (if option prob was used).

## Author(s)

Christian Buchta

## See Also

dist for efficient computation of cross-distances.

## Examples

```
## Not run:
### extend Rock example
example("rockCluster")
k <- sample(nrow(x), 100)
y <- rf$cl[k]
levels(y)[3:4] <- 0
gk <- gknn(dist(x, x[k,], method="binary"), y, k=3)
attr(gk, "levels")[3] <- levels(rf$cl)[4]
table(cl = rf$cl, gk)

## End(Not run)
```

---

image *Matrix Image Plots*

---

## Description

Implements a wrapper function to image for proper plotting of objects of class matrix and dist.

## Usage

```
implot(x, xlab = "", ylab = "", axes = FALSE, ticks = 10,
       las = 2, ...)
```

## Arguments

| | |
|---|---|
| x | an object of class matrix or dist. |
| xlab, ylab | labels for the x and y axis. |
| axes | logical, indicating whether dimnames(x) should be drawn on the plot. |
| ticks | the number of tick-marks to use. |
| las | the axis style to use (see par). |
| ... | further arguments to image. |

### Details

Plots an object of class `matrix` in its original row and column orientation. This means, in a plot the columns become the x-coordinates and the reversed rows the y-coordinates.

If `x` is of class `dist` it is coerced to full-storage `matrix` representation.

### Value

Returns the transformed `x` *invisibly*.

### Author(s)

Christian Buchta

### See Also

`image` and `par` for details.

### Examples

```
x <- matrix(sample(c(FALSE, TRUE),100,rep=TRUE),ncol=10,
     dimnames=list(1:10,LETTERS[1:10]))
implot(x)
implot(x, col=c("white","black"), axes = TRUE)
```

---

## `lminter` *Interpolating Logical Matrices*

---

### Description

Interpolate a logical matrix into a lower-resolution representation.

### Usage

```
lminter(x, block.size = 1, nbin = 0)
```

### Arguments

| | |
|---|---|
| `x` | a logical matrix. |
| `block.size` | the interpolation block size. |
| `nbin` | the number of density bins. |

### Details

Partitions a binary matrix into square blocks of specified size (length) and interpolates the number of TRUE values per block using the specified number of bins.

Note that the effective number of bins is one greater than the specified number because the zero bin is always included. Excess rows and columns at the lower or right margins of the matrix are ignored.

If the number of bins is null counts are mapped to zero and one thresholding at half of the number of distinct count values including zero. Thus, for even numbered block sizes there is a bias towards zero.

### Value

An integer matrix of bin numbers.

### Note

Package internal function.

### Author(s)

Christian Buchta

### See Also

lmplot for plotting logical matrices.

### Examples

```
## Not run:
x <- matrix(sample(c(FALSE, TRUE), 4 ,rep=TRUE), ncol=2)
cba:::lminter(x, block.size=2, nbin=2)

## End(Not run)
```

---

## lmplot *Plotting Logical Matrices*

---

### Description

Implements a wrapper function to image that produces a black and white or gray-scale plot of a logical matrix.

### Usage

```
lmplot(x, block.size = 1, gray = FALSE, xlab = "", ylab = "",
       axes = FALSE, ...)
```

### Arguments

| | |
|---|---|
| x | a logical matrix. |
| block.size | the interpolation block size. |
| gray | optionally use a gray scale. |
| xlab | title for the x axis. |
| ylab | title for the y axis. |
| axes | option to plot axes. |
| ... | further arguments to image. |

### Details

TRUE is represented by the color white and FALSE by the color black.

A lower resolution can be obtained by specifying an (interpolation) block size greater than one. Block densities can then be visualized by using the gray scale option. The number of levels of the palette corresponds to the block size but is capped to 8 levels (excluding white). Note that the opacity (blackness) corresponds with density (as on photographic film).

### Author(s)

Christian Buchta

### See Also

lminter for interpolating logical matrices and image for further plotting options

### Examples

```
###
x <- matrix(sample(c(FALSE, TRUE), 64, rep=TRUE), ncol=8)
lmplot(x)
### use lower resolution
lmplot(x, block.size=2)
### use gray scale
lmplot(x, block.size=2, gray=TRUE)
```

---

## Mushroom *Mushroom Data Set*

---

### Description

A data set with descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family, classified according to their edibility as (definitely) 'edible' or 'poisonous' (definitely poisonous, or of unknown edibility and not recommended).

### Usage

```
data("Mushroom")
```

## Format

A data frame with 8124 observations on the following 23 variables.

`class` a factor with levels `edible` and `poisonous`.

`cap-shape` a factor with levels `bell`, `conical`, `convex`, `flat`, `knobbed`, `sunken`.

`cap-surface` a factor with levels `fibrous`, `grooves`, `scaly`, `smooth`.

`cap-color` a factor with levels `brown`, `buff`, `cinnamon`, `gray`, `green`, `pink`, `purple`, `red`, `white`, `yellow`.

`bruises?` a factor with levels `bruises` and `no`.

`odor` a factor with levels `almond`, `anise`, `creosote`, `fishy`, `foul`, `musty`, `none`, `pungent`, `spicy`.

`gill-attachment` a factor with levels `attached` and `free`.

`gill-spacing` a factor with levels `close` and `crowded`.

`gill-size` a factor with levels `broad` and `narrow`.

`gill-color` a factor with levels `black`, `brown`, `buff`, `chocolate`, `gray`, `green`, `orange`, `pink`, `purple`, `red`, `white`, and `yellow`.

`stalk-shape` a factor with levels `enlarging` and `tapering`.

`stalk-root` a factor with levels `bulbous`, `club`, `equal`, and `rooted`.

`stalk-surface-above-ring` a factor with levels `fibrous`, `scaly`, `silky`, and `smooth`.

`stalk-surface-below-ring` a factor with levels `fibrous`, `scaly`, `silky`, and `smooth`.

`stalk-color-above-ring` a factor with levels `brown`, `buff`, `cinnamon`, `gray`, `orange`, `pink`, `red`, `white`, and `yellow`.

`stalk-color-below-ring` a factor with levels `brown`, `buff`, `cinnamon`, `gray`, `orange`, `pink`, `red`, `white`, and `yellow`.

`veil-type` a factor with levels `partial`.

`veil-color` a factor with levels `brown`, `orange`, `white`, and `yellow`.

`ring-number` a factor with levels `one`, `one`, and `two`.

`ring-type` a factor with levels `evanescent`, `flaring`, `large`, `none`, and `pendant`.

`spore-print-color` a factor with levels `black`, `brown`, `buff`, `chocolate`, `green`, `orange`, `purple`, `white`, and `yellow`.

`population` a factor with levels `abundant`, `clustered`, `numerous`, `scattered`, `several`, and `solitary`.

`habitat` a factor with levels `grasses`, `leaves`, `meadows`, `paths`, `urban`, `waste`, and `woods`.

## Details

The records are drawn from G. H. Lincoff (1981) (Pres.), *The Audubon Society Field Guide to North American Mushrooms*. New York: Alfred A. Knopf. (See pages 500–525 for the Agaricus and Lepiota Family.)

The Guide clearly states that there is no simple rule for determining the edibility of a mushroom; no rule like "leaflets three, let it be" for Poisonous Oak and Ivy.

Unused levels in the original data were dropped.

The current version of the UC Irvine Machine Learning Repository Mushroom data set is available from doi:10.24432/C5959T

Blake, C.L. & Merz, C.J. (1998). UCI Repository of Machine Learning Databases. Irvine, CA: University of California, Department of Information and Computer Science. Formerly available from 'http://www.ics.uci.edu/~mlearn/MLRepository.html'.

## Examples

```
data("Mushroom")
summary(Mushroom)
```

---

`order` *Improving the Presentation of Matrix Objects*

---

## Description

High-level functions that improve the presentation of a matrix or data frame by reordering their rows and columns.

## Usage

```
order.dist(x, index = FALSE)
order.matrix(x, type = "neumann", by = c("both","rows","cols"),
             index = FALSE)
order.data.frame(x, type = "neumann", by = c("both","rows","cols"),
                 index = FALSE)
```

## Arguments

| | |
|---|---|
| `x` | an object of class `dist`, `matrix`, or `data.frame`. |
| `type` | the type of stress measure to use (see details). |
| `by` | option to order either by rows, or columns, or both. |
| `index` | option to return the order index(ex) instead of the reordered object. |

## Details

These functions try to improve the presentation of an object of class `dist`, `matrix`, or `data.frame` by reordering the rows and columns such that similar entries are grouped together.

`order.dist` uses a simple heuristic to solve the TSP problem of finding an ordering of minimum length (see `order.length`) for an object of class `dist`. Note that the heuristic used is quick but more elaborate TSP algorithms will produce better orderings.

`order.matrix` tries to minimize the stress measure of a matrix (see `stress` by using the same TSP heuristic as above, once for the column and once for the row ordering (while the other dimension is fixed) if `by = "both"`.

`order.data.frame` uses attributes of type `numeric` and `logical` only, combines them into a normalized matrix and finds an ordering as above.

### Value

Either the reordered object supplied, or a vector of subscripts (for `reorder.dist`), or a list with components `rows` and `columns` containing the order indexes (for `reorder.matrix` and `reorder.data.frame`).

### Note

This is experimental code that may be integrated in a separate packages in the future.

### Author(s)

Christian Buchta

### See Also

`dist`, `stress`, `stress.dist`.

### Examples

```
## not a hard problem
data(iris)
d <- dist(iris[1:4])
implot(order.dist(d))
data(townships)
x <- order.data.frame(townships)
x
```

---

## `order.greedy` *Hierarchical Greedy Ordering*

---

### Description

Compute a hierarchical greedy ordering of a data matrix.

### Usage

```
order.greedy(dist)
```

### Arguments

| | |
|---|---|
| `dist` | an object of class `dist`. |

### Details

A single cluster is constructed by merging in each step the leaf closest to one of the two endpoints of the cluster. The algorithm starts with a random leaf and uses tie-breaking.

Clearly, the algorithm is more an ordering than a cluster algorithm. However, it constructs a binary merge tree so that the linear ordering of its leaves could be further improved.

## Value

A list with the following components:

| | |
|---|---|
| `merge` | a matrix containing the merge tree. |
| `order` | a vector containing the leaf ordering. |
| `height` | a vector containing the merge heights. |

## Note

The merge heights may not be monotonic.

## Author(s)

Christian Buchta

## References

F. Murtagh (1985). Multidimensional Cluster Algorithms. *Lectures in Computational Statistics*, Physica Verlag, pp. 15.

## See Also

`hclust` for hierarchical clustering, `order.optimal` for optimal leaf ordering, and `order.length` for computing the objective value of a leaf ordering.

## Examples

```
d <- dist(matrix(runif(20), ncol=2))
hc <- hclust(d)
co <- order.optimal(d, hc$merge)
md <- -as.dist(crossprod(as.matrix(d, diag = 0)))   # Murtagh's distances
hg <- order.greedy(md)
go <- order.optimal(md, hg$merge)
### compare images
op <- par(mfrow=c(2,2), pty="s")
implot(d[[hc$order]], main="hclust")
implot(d[[co$order]], main="hlcust + optimal")
implot(d[[hg$order]], main="greedy")
implot(d[[go$order]], main="greedy + optimal")
par(op)
# compare lengths
order.length(d, hc$order)
order.length(d, co$order)
order.length(d, hg$order)
order.length(d, go$order)
```

`order.length` *Conciseness of Presentation Measures*

## Description

Compute the length of a Hamilton path through a distance matrix.

## Usage

```
order.length(dist, order)
```

## Arguments

| | |
|---|---|
| `dist` | an object of class `dist`. |
| `order` | an optional permutation of the row (column) indexes. |

## Details

Ordering a distance matrix such that low distance values are placed close to the diagonal may improve its presentation. The length of an order is the corresponding objective measure.

The order corresponds to a path through a graph where each node is visited only once, i.e. a Hamilton path. The length of a path is defined as the sum of the edge weights, i.e. distances.

If `order` is missing the identity order is used.

If `order` is not unique `NA` is returned.

If there are non-finite distance values `NA` is returned.

## Value

A scalar real value.

## Author(s)

Christian Buchta

## References

R. Sedgewick. (2002). *Algorithms in C. Part 5. Graph Algorithms*. 3rd Edition, Addison-Wesley.

## Examples

```
d <- dist(matrix(runif(10),ncol=2))
order.length(d)
o <- sample(5,5) # random order
order.length(d, o)
```

`order.optimal` *Optimal Leaf Ordering of Binary Trees.*

## Description

Find an optimal linear leaf ordering of a binary merge tree as produced by a hierarchical cluster algorithm.

## Usage

```
order.optimal(dist, merge)
```

## Arguments

| | |
|---|---|
| `dist` | an object of class `dist`. |
| `merge` | a binary merge tree (see `hclust`). |

## Details

A binary tree has $2^{n-1}$ internal nodes (subtrees) and the same number of leaf orderings. That is, at each internal node the left and right subtree (or leaves) can be swapped, or, in terms of a dendrogram, be flipped.

An objective measure of a leaf ordering is the sum of the distances along the path connecting the leaves in the given order. An ordering with a minimal path length is defined to be an optimal ordering.

This function provides an interface to the optimal leaf ordering algorithm (see references) for tree representations that are used by hierarchical cluster algorithms such as `hclust`.

Note that non-finite distance values are not allowed.

## Value

A list with the following components:

| | |
|---|---|
| `merge` | a matrix containing the merge tree corresponding with the optimal leaf order. |
| `order` | a vector containing the optimal leaf order. |
| `length` | the length of the ordering. |

## Note

The time complexity of the algorithm is $O(n^3)$.

## Author(s)

Christian Buchta

## References

Z. Bar-Joseph, E. D. Demaine, D. K. Gifford, and T. Jaakkola. (2001). Fast Optimal Leaf Ordering for Hierarchical Clustering. *Bioinformatics*, Vol. 17 Suppl. 1, pp. 22-29.

## See Also

`hclust` for hierarchical clustering and `order.length` for computing the objective value of a leaf ordering.

## Examples

```
d <- dist(matrix(runif(30), ncol=2))
hc <- hclust(d)
co <- order.optimal(d, hc$merge)
### compare dendrograms
ho <- hc
ho$merge <- co$merge
ho$order <- co$order
op <- par(mfrow=c(2,2), pty="s")
plot(hc, main="hclust")
plot(ho, main="optimal")
# compare images
implot(d[[hc$order]])
implot(d[[co$order]])
par(op)
### compare lengths
order.length(d, hc$order)
order.length(d, co$order)
cat("compare: ",co$length,"\n")
```

---

# plot.sdists.graph *Plotting Edit Transcripts and Sequence Alignments*

---

## Description

Function for visualizing the optimal transformation (alignment) graph for two symbol sequences.

## Usage

```
## S3 method for class 'sdists.graph'
plot(x, circle.col = 1, graph.col = 2,
     circle.scale = c("mean", "max", "last", "text"), main = "", ...)
```

## Arguments

| | |
|---|---|
| `x` | an object of class `sdists.graph`. |
| `circle.col` | color to be used for circles. |
| `graph.col` | color to be used for the graph. |

| | |
|---|---|
| `circle.scale` | scaling to be used for circles. |
| `main` | plot title. |
| `...` | further unused arguments. |

## Details

This function plots the dynamic programming table, the (back)pointers and the combined graph of optimal edit transcripts (alignments) computed with `sdist.trace`. The fist sequence is represented by the y-axis and the second by the x-axis.

The circumference of a circle is proportional to the minimum cost (maximum weight) of edit (alignment) operations leading to a table cell. `circle.scale` specifies the type of normalization performed where `last` means the last table entry (containing the optimum value), and `text` draws the values instead of circles.

The (back)pointers, defining possible (traceback) paths, are plotted as `dotted` edges. Note that a traceback starts in the last cell of the table and ends at the origin.

For the edges of the graph that is the union of all optimal paths, two line types are used: `solid` for insert, delete, and replace operations, and `dashed` for a match. The line width indicates the number of times an edge is on a path, but note that the interpretation is device-specific (compare `par`).

## Note

Some issues with grid were fixed in R.2.4.x (Fixme ?).

## Author(s)

Christian Buchta

## References

D. Gusfield (1997). *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, Chapter 11.

Inspired by: http://home.uchicago.edu/~aabbott/.

## See Also

`sdists.trace`, `plot`

## Examples

```
## continue example in sdists.trace
x1 <- "vintner"
y1 <- "writers"
b11 <- sdists.trace(x1, y1, weight=c(1,1,0,1), graph = TRUE)
b11
plot(b11)
plot(b11, circle.scale = "text")
## partial
b12 <- sdists.trace(x1, y1, weight=c(1,1,0,1), graph = TRUE, partial = TRUE)
b12
```

```
plot(b12)
```

---

`predict.ccfkms` *Clustering with Conjugate Convex Functions.*

---

## Description

Classify the rows of a data matrix using conjugate convex functions.

## Usage

```
## S3 method for class 'ccfkms'
predict(object, x, drop = 1, ...)
```

## Arguments

| | |
|---|---|
| `object` | an object of class `ccfkms`. |
| `x` | a data matrix containing test or new samples. |
| `drop` | cluster size threshold. |
| `...` | other (unused) arguments. |

## Details

This is a wrapper to `ccfkms` which uses a single iteration for classifying the data.

In the case a `drop` value greater than zero is specified, all clusters with size equal or less than this value are removed from the classifier.

## Value

A list with the following components:

| | |
|---|---|
| `centers` | a matrix of cluster means. |
| `size` | a vector of cluster sizes. |
| `cl` | a factor of cluster labels (indexes). |
| `inv.inf` | the inverted information of the partition. |

## Author(s)

Christian Buchta

## See Also

`ccfkms` for obtaining a classifier.

## Examples

```
### see ccfkms
```

predict.rock *Rock Clustering*

## Description

Classify the rows of a data matrix using the Rock classifier.

## Usage

```
## S3 method for class 'rock'
predict(object, x, drop = 1, ...)

## S3 method for class 'rock'
fitted(object, ...)
```

## Arguments

| | |
|---|---|
| object | an object of class rock. |
| x | a data matrix containing test or new samples. |
| drop | cluster size threshold. |
| ... | further (unused) arguments. |

## Details

Provides a wrapper function to the Rock classifier: cluster memberships of the (row) samples are determined by majority vote using the size (of the cluster) weighted number of links.

The classifier uses random tie-breaking and assigns samples with zero link counts to NA. This allows for detection of possible outliers, or interesting patterns.

In the case a drop value greater than zero is specified, all clusters with size equal or less than this value are removed from the classifier. Especially, fitted uses a threshold of one because for singleton clusters the neighborhood is empty.

Note that for the training data the predicted memberships need not necessarily be the same as those obtained from the cluster algorithm.

## Value

A list with the following components:

| | |
|---|---|
| cl | a factor of cluster memberships of the samples. |
| size | a vector of cluster sizes. |

## See Also

rockCluster for obtaining a rock classifier and gknn for k-nearest neighbor classification.

## Examples

```
### example from paper
data(Votes)
x <- as.dummy(Votes[-17])
rc <- rockCluster(x, n=2, theta=0.73, debug=TRUE)
rp <- predict(rc, x)
table(rp$cl)
```

---

`proximus` *Proximus*

---

## Description

Cluster the rows of a logical matrix using the Proximus algorithm. The compression rate of the algorithm can be influenced by the choice of the maximum cluster radius and the minimum cluster size.

## Usage

```
proximus(x, max.radius = 2, min.size = 1, min.retry = 10,
         max.iter = 16, debug = FALSE)
```

## Arguments

| | |
|---|---|
| `x` | a logical matrix. |
| `max.radius` | the maximum number of bits a member in a row set may deviate from its dominant pattern. |
| `min.size` | the minimum split size of a row set. |
| `min.retry` | number of retries to split a pure rank-one approximation (translates into a resampling rate). |
| `max.iter` | the maximum number of iterations for finding a local rank-one approximation. |
| `debug` | optional debugging output. |

## Details

The intended area of application is the compression of high-dimensional binary data into representative patterns. For instance, purchase incidence (market basket data) or term-document matrices may be preprocessed by Proximus for later association rule mining.

The algorithm is of a recursive partitioning type. Specifically, at each step a binary split is attempted using a local rank-one approximation of the current submatrix (row set). That is a specialization of principal components to binary data which represents a matrix as the outer product of two binary vectors. The node expansion stops if a submatrix is pure, i.e., the column (presence set) vector indicates all the rows and the Hamming distances from the row (dominant attribute set) pattern vector, or the size of the row set, are less than or equal the specified threshold. In the case the rank-one approximation does not result in a split but the radius constraint is violated, the matrix is split using a random row and the radius constraint.

The debug option can be used to gain some insight into how the algorithm proceeds: a right angle bracket indicates a split and the return to a recursion level is indicated by a left one. Leafs in the recursion tree are indicated by an asterisk and retries by a plus sign. The number of retries is bounded by the size of the current set divided by `min.retry`. Double angle brackets indicate a random split (see above). The numbers between square brackets indicate the current set size, the size of the presence (sub)set, and its radius. The adjoining numbers indicate the depth of the recursion and the count of retries. Finally, a count of the leaf nodes found so far is shown to the right of an asterisk.

## Value

An object of class `proximus` with the following components:

| | |
|---|---|
| `nr` | the number of rows of the data matrix. |
| `nc` | the number of columns of the data matrix. |
| `a` | a list containing the approximations (patterns). |
| `a$x` | a vector of row (presence set) indexes. |
| `a$y` | a vector of column (dominant attribute set) indexes. |
| `a$n` | the number of ones in the approximated submatrix. |
| `a$c` | the absolute error reduction by the approximation. |
| `max.radius` | see arguments. |
| `min.size` | see arguments. |
| `rownames` | rownames of the data matrix. |
| `colnames` | colnames of the data matrix. |

## Warning

Deep recursions may exhaust your computer.

## Note

The size of a set need not be equal or greater than the user defined threshold.

## Author(s)

Christian Buchta

## References

M. Koyutürk, A. Graham, and N. Ramakrishnan. Compression, Clustering, and Pattern Discovery in Very High-Dimensional Discrete-Attribute Data Sets. *IEEE Transactions On Knowledge and Data Engineering*, Vol. 17, No. 4, (April) 2005.

## See Also

`summary.proximus` for summaries, `fitted` for obtaining the approximated matrix and the pattern labels of the samples, and `lmplot` for plotting logical matrices.

## Examples

```
x <- matrix(sample(c(FALSE, TRUE), 200, rep=TRUE), ncol=10)
pr <- proximus(x, max.radius=8)
summary(pr)
### example from paper
x <- rlbmat()
pr <- proximus(x, max.radius=8, debug=TRUE)
op <- par(mfrow=c(1,2), pty="s")
lmplot(x, main="Data")
box()
lmplot(fitted(pr)$x, main="Approximation")
box()
par(op)
```

---

`rlbmat` *Block Uniform Logical Matrix Deviates*

---

## Description

Generate a uniform logical matrix deviate with a possibly overlapping block structure.

## Usage

```
rlbmat(npat = 4, rows = 20, cols = 12, over = 4, noise = 0.01,
       prob = 0.8, perfect = FALSE)
```

## Arguments

| | |
|---|---|
| `npat` | number of patterns. |
| `rows` | number of rows per pattern. |
| `cols` | number of columns per pattern. |
| `over` | number of additional columns per pattern that overlap. |
| `noise` | the probability of observing a one in the background (non-pattern) matrix. |
| `prob` | the probability of observing `TRUE` in a pattern. |
| `perfect` | option for overlap of the first and the last pattern. |

## Details

Implements a test case for `proximus`.

## Value

A logical matrix

## Author(s)

Christian Buchta

## See Also

lmplot and clmplot for plotting a logical matrix

## Examples

```
x <- rlbmat()
lmplot(x)
```

---

rockCluster *Rock Clustering*

---

## Description

Cluster a data matrix using the Rock algorithm.

## Usage

```
rockCluster(x, n, beta = 1-theta, theta = 0.5, fun = "dist",
            funArgs = list(method="binary"), debug = FALSE)

rockLink(x, beta = 0.5)
```

## Arguments

| | |
|---|---|
| x | a data matrix; for rockLink an object of class dist. |
| n | the number of desired clusters. |
| beta | optional distance threshold. |
| theta | neighborhood parameter in the range [0,1). |
| fun | distance function to use. |
| funArgs | a list of named parameter arguments to fun. |
| debug | turn on/off debugging output. |

## Details

The intended area of application is the clustering of binary (logical) data. For instance in a preprocessing step in data mining. However, arbitrary distance metrics could be used (see dist).

According to the reference (see below) the distance threshold and the neighborhood parameter are coupled. Thus, higher values of the neighborhood parameter theta pose a tighter constraint on the neighborhood. For any two data points the latter is defined as the number of other data points that are neighbors to both. Further, points only are neighbors (or linked) if their distance is less than or equal beta.

Note that for a tight neighborhood specification the algorithm may be running out of clusters to merge, i.e. may terminate with more than the desired number of clusters.

The debug option can help in determining the proper settings by examining lines suffixed with a plus which indicates that non-singleton clusters were merged.

Note that tie-breaking is not implemented, i.e. the first max encountered is used. However, permuting the order of the data can help in determining the dependence of a solution on ties.

Function `rockLink` is provided for applications that need to compute link count distances efficiently. Note that `NA` and `NaN` distances are ignored but supplying such values for the threshold `beta` results in an error.

## Value

`rockCluster` returns an object of class `rock`, a list with the following components:

| | |
|---|---|
| `x` | the data matrix or a subset of it. |
| `cl` | a factor of cluster labels. |
| `size` | a vector of cluster sizes. |
| `beta` | see above. |
| `theta` | see above. |

`rockLink` returns an object of class `dist`.

## Author(s)

Christian Buchta

## References

S. Guha, R. Rastogi, and K. Shim. ROCK: A Robust Clustering Algorithm for Categorical Attributes. *Information Science*, Vol. 25, No. 5, 2000.

## See Also

`dist` for common distance functions, `predict` for classifying new data samples, and `fitted` for classifying the clustered data samples.

## Examples

```
### example from paper
data(Votes)
x <- as.dummy(Votes[-17])
rc <- rockCluster(x, n=2, theta=0.73, debug=TRUE)
print(rc)
rf <- fitted(rc)
table(Votes$Class, rf$cl)
## Not run:
### large example from paper
data("Mushroom")
x <- as.dummy(Mushroom[-1])
rc <- rockCluster(x[sample(dim(x)[1],1000),], n=10, theta=0.8)
print(rc)
rp <- predict(rc, x)
table(Mushroom$class, rp$cl)
```

```
## End(Not run)
### real valued example
gdist <- function(x, y=NULL) 1-exp(-dist(x, y)^2)
xr <- matrix(rnorm(200, sd=0.6)+rep(rep(c(1,-1),each=50),2), ncol=2)
rcr <- rockCluster(xr, n=2, theta=0.75, fun=gdist, funArgs=NULL)
print(rcr)
```

---

`sdists` *Sequence Distance Computation*

---

## Description

This function computes and returns the auto-distance matrix between the vectors of a list or between the character strings of a vector treating them as sequences of symbols, as well as the cross-distance matrix between two such lists or vectors.

## Usage

```
sdists(x, y = NULL, method = "ow", weight = c(1, 1, 0, 2),
       exclude = c(NA, NaN, Inf, -Inf), pairwise = FALSE)
```

## Arguments

| | |
|---|---|
| `x, y` | a list (of vectors) or a vector of character. |
| `method` | a mnemonic string referencing a distance measure. |
| `weight` | vector or matrix of parameter values. |
| `exclude` | argument to factor. |
| `pairwise` | compute distances for the parallel pairs of `x` and `y` only. |

## Details

This function provides a common interface to different methods for computation of distances between sequences, such as the edit a.k.a. Levenshtein distance. Conversely, in the context of sequence alignment the similarity of the maximizing alignment is computed.

Note that negative similarities are returned as distances. So be careful to use a proper weighting (scoring) scheme.

The following methods are currently implemented:

`ow`**:** operation-weight edit distance. Weights have to be specified for deletion, insertion, match, and replacement. Other weights for initial operations can be specified as `weight[5:6]`.

`aw`**:** alphabet-weight sequential alignment similarity. A matrix of weights (scores) for all possible symbol replacements needs to be specified with the convention that the first row/column defines the replacement with the empty (space) symbol. The colnames of this matrix are used as the levels argument for the encoding as `factor`. Consequently, unspecified symbols are mapped to `NA`.

`awl`: alphabet-weight local sequential alignment similarity. The weight matrix must be as described above. However, note that zero acts as threshold for a 'restart' of the search for a local alignment and at the same time indicates that the solution is the empty substring. Thus, you normally would use non-negative scores for matches and non-positive weights otherwise.

Missing (and non-finite) values should be avoided, i.e. either be removed or recoded (and appropriately weighted). By default they are excluded when coercing to factor and therefore mapped to `NA`. The result is then defined to be `NA` as we cannot determine a match!

The time complexity is O(n*m) for two sequences of length n and m.

Note that in the case of auto-distances the weight matrix must be (exactly) symmetric. Otherwise, for asymmetric weights `y` must not be `NULL`. For instance, `x` may be supplied twice (see the examples).

## Value

Auto distances are returned as an object of class `dist` and cross-distances as an object of class `matrix`.

## Warning

The interface is experimental and may change in the future

## Author(s)

Christian Buchta

## References

D. Gusfield (1997). *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, Chapter 11.

## See Also

`sdists.trace` for computation of edit transcripts and sequence alignments, `dist` for computation of common distances, `agrep` for searches for approximate matches.

## Examples

```
### numeric data
sdists(list(c(2,2,3),c(2,4,3)))   # 2
sdists(list(c(2,2,3),c(2,4,3)),weight=c(1,1,0,1)) # 1

### character data
w <- matrix(-1,nrow=8,ncol=8)   # weight/score matrix for
diag(w) <- 0     # longest common subsequence
colnames(w) <- c("",letters[1:7])
x <- sapply(rbinom(3,64,0.5),function(n,x)
    paste(sample(x,n,rep=TRUE),collapse=""),
    colnames(w)[-1])
x
```

```
sdists(x,method="aw",weight=w)
sdists(x,x,method="aw",weight=w)  # check
## pairwise
sdists(x,rev(x),method="aw",weight=w,pairwise = TRUE)
diag(w) <- seq(0,7)
sdists(x,method="aw", weight=w)   # global alignment
sdists(x,method="awl",weight=w)   # local alignment

## empty strings
sdists("", "FOO")
sdists("", list(c("F","O","O")))
sdists("", list(""))    # space symbol
sdists("", "abc", method="aw", weight=w)
sdists("", list(""), method="aw", weight=w)

### asymmetric weights
w[] <- matrix(-sample(0:5,64,TRUE),ncol=8)
diag(w) <- seq(0,7)
sdists(x,x,method="aw", weight=w)
sdists(x,x,method="awl",weight=w)

### missing values
sdists(list(c(2,2,3),c(2,NA,3)),exclude=NULL) # 2 (include anything)
sdists(list(c(2,2,3),c(2,NA,3)),exclude=NA) # NA
```

---

`sdists.center` *Centroid Sequences*

---

## Description

Find centroid sequences among a collection of sequences.

## Usage

```
sdists.center(x, d = NULL, method = "ow", weight = c(1, 1, 0, 2),
              exclude = c(NA, NaN, Inf, -Inf), FUN = NULL, ...,
              unique = FALSE)
```

## Arguments

| | |
|---|---|
| `x` | a list (of vectors) of a vector of character. |
| `d` | a matrix or an object of class `dist`. |
| `method` | argument to `sdists`. |
| `weight` | argument to `sdists`. |
| `exclude` | argument to `sdists`. |
| `FUN` | a function to rank distances. |
| `...` | additional arguments to `FUN`. |
| `unique` | a logical specifying whether to return a unique set of sequences. |

## Details

This function provides a wrapper to computing the distances among the sequences in x, unless d is supplied, and the subsequent selection of a set of centroid sequences with minimum sum of distances to any other sequence.

## Value

A subset of x.

## Author(s)

Christian Buchta

## See Also

sdists for distance computation.

## Examples

```
x <- c("ABCD", "AD", "BCD", "ACF", "CDF", "BC")
sdists.center(x)
```

---

# sdists.center.align *Align Sequences to a Center*

---

## Description

Find a global alignment of a collection of sequences using the center-star-tree heuristic.

## Usage

```
sdists.center.align(x, center, method = "ow", weight = c(1, 1, 0, 2),
                    exclude = c(NA, NaN, Inf, -Inf),
                    break.ties = TRUE, transitive = FALSE,
                    to.data.frame = FALSE)
```

## Arguments

| | |
|---|---|
| x | a list (of vectors) or a vector of character. |
| center | a vector |
| method | argument to sdists. |
| weight | argument to sdists. |
| exclude | arguments to sdists. |
| break.ties | a logical specifying whether random tie-breaking should be performed. Otherwise the first alignment is used. |
| transitive | a logical specifying whether the sequences in x should be aligned with each other, too. |
| to.data.frame | a logical specifying whether the result should be converted to data.frame. |

### Details

Each component of x is aligned with center in turn such that the latter is aligned with all sequences processed so far.

If center is missing isdists.center is used to compute an initial center.

### Value

Either a list of sequences with attributes center and ties, or a data.frame with the sequences in the columns.

### Note

The global alignment may depend on the order of x.

### Author(s)

Christian Buchta

### References

D. Gusfield (1997). *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, Chapter XX.

### See Also

sdists for computation of distances, sdists.center for computation of centroids.

### Examples

```
## continue example
x <- c("ABCD", "AD", "BCD", "ACF", "CDF", "BC")
sdists.center.align(x)
sdists.center.align(x, transitive = TRUE, to.data.frame = TRUE)
```

---

## sdists.trace *Edit Transcripts and Sequence Alignments*

---

### Description

This function computes and returns the set of all optimal but equivalent edit transcripts that transforms one sequences into another at minimum cost, as well as the corresponding aligned sequences, or, alternatively a combined edit graph.

### Usage

```
sdists.trace(x, y, method = "ow", weight = c(1, 1, 0, 2),
             exclude = c(NA, NaN, Inf, -Inf), graph = FALSE,
      partial = FALSE)
```

## Arguments

| | |
|---|---|
| x, y | a numeric or string vector. |
| method | a mnemonic string referencing a distance measure. |
| weight | vector or matrix of parameter values. |
| exclude | argument to factor. |
| graph | option to compute the combined edit graph. |
| partial | option to compute an approximate substring match. |

## Details

Function sdists.trace complements the distance computation between sequences by sdists. So, please, see the details of method, weight, and exclude there. However, note the following differences: 1) you can supply only two sequences, either as vectors of numeric symbol codes, factors, or as strings, i.e. scalar vectors of type character. 2) you can supply a weight matrix with the rownames and colnames representing the symbol sets of the first and second sequence. For instance, this allows you to align a sequence with the profile of a multiple alignment. 3) if method = "ow" the space symbol "" is included in the factor levels so that you can conveniently replace NA in the aligned sequences.

A transcript uses the character codes I, D, R, and M, for insert, delete, replace, and match operations, which transform the first into the second sequence. Thus, conceptually a symbol has to be inserted into the first, deleted from the second, replaced in the first sequence, or matched in both, to obtain the second sequence. However, in the aligned sequences you will see NA, where an insert or delete would take place, indicating space.

In the case of a local alignment different symbols are used for the prefix and/or suffix of the alignment: i, d, and ? for insert, delete, and replace or match operations. However, note that their sole purpose is to obtain a common representation of the two sequences. Finally, only alignments of maximal length are reported.

The time complexity of finding a transcript is $O(n + m)$ for two sequences of length n and m, respectively $O(n * m)$ for the local alignment problem. However, note that the runtime for generating all transcripts can be $O((n * m)^3)$ in the worst case.

If partial = FALSE computes an approximate substring match of x (the pattern) in y, for method = "ow" only. Returns the subset of paths which require the maximum number of match and initial and final insert operations.

## Value

A list with components each a list of two factors, the aligned sequences. The names of the components are the edit transcripts, and the attribute value contains the minimum cost, i.e. the distance (or negative similarity).

If graph = TRUE a vector of edit transcripts is returned with attributes value, table, pointer, and graph. The second contains the values of the dynamic programming table and the third a list of vectors x0, y0, x1, y1 representing the (back)pointers. Similarly, the fourth attribute is a list of vectors x0, y0, x1, y1, weight representing the edge set of all optimal paths. That is, each tuple contains the from and to coordinates as used by segments, each representing a pair of indexes into the first and second sequence, and the number of times an edge occurs on a path. Note that

the origin of the coordinate system (0,0) corresponds to the element of `table` indexed by (`""`,`""`), where `""` indicates the space symbol. Thus, if used as subscripts the coordinates have to be offset by one.

## Warning

The interface is experimental and may change in the future

## Author(s)

Christian Buchta

## References

D. Gusfield (1997). *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, Chapter 11.

## See Also

`sdists` for computation of distances between sequences, `segments` for plotting of edge sets, `plot.sdists.graph` for visualizing alignments.

## Examples

```
### from the book
x1 <- "vintner"
y1 <- "writers"
b1 <- sdists.trace(x1, y1, weight=c(1,1,0,1))
b1
## longest common subsequence ?
sdists.trace("a","b", weight=c(0,0,-1,0))
## from the book
w2 <- matrix(-2,ncol=13,nrow=13)
w2[1,] <- w2[,1] <- -1
diag(w2) <- c(0,rep(2,12))
x2 <- "pqraxabcstvq"
y2 <- "xyaxbacsll"
colnames(w2) <- c("",unique(strsplit(paste(x2, y2, sep = ""),"")[[1]]))
b2 <- sdists.trace(x2, y2, method="awl", weight=w2)
b2
## alignment with different symbol sets
x3 <- "121314"
y3 <- "ABACAD"
w3 <- matrix(-1,nrow=5,ncol=5)
diag(w3) <- 0
rownames(w3) <- c("","1","2","3","4")
colnames(w3) <- c("","A","B","C","D")
b3 <- sdists.trace(x3, y3, method="aw", weight=w3)
b3
## partial
b4 <- sdists.trace(x1, y1, weight=c(1,1,0,1), partial = TRUE)
b4
```

stress *Conciseness of Presentation Measures*

## Description

Compute different types of conciseness of presentation measures.

## Usage

```
stress(x, rows = NULL, cols = NULL, type = "moore")

stress.dist(x, rows = NULL, cols = NULL, bycol = FALSE,
            type = "moore")
```

## Arguments

| | |
|---|---|
| x | a matrix object. |
| rows | a subscript vector indexing the rows. |
| cols | a subscript vector indexing the columns. |
| bycol | logical for computation over the columns. |
| type | the type of neighborhood to use. |

## Details

Function stress computes the sum of squared distances of each matrix entry from its adjacent entries. The following types of neighborhoods are available:

moore: comprises the eight adjacent entries (five at the margins and three at the corners).

neumann: comprises the four adjacent entries (three at the margins and two at the corners).

Function stress.dist computes the auto-distance matrix for each pair of rows (or columns) given one of the above stress measures. Note that the result depends on the ordering of the unused dimension.

As the computation can be reduced to summing the edge distances between any two neighboring points, only half of the value of the proposed measures is reported.

Row and/or column indexes (or labels) can be supplied to test specific orderings, as well as subsets of indexes (labels).

Note that the matrix should be normalized so that the distance computation is meaningful.

## Value

stress returns a scalar real, i.e. half of the global stress measure.

stress.dist returns an object of class dist, i.e. a lower triangular matrix in column format.

### Author(s)

Christian Buchta

### See Also

`dist` for general distance computation.

### Examples

```
##
x1 <- matrix(sample(c(FALSE,TRUE),25,rep=TRUE),ncol=5)
stress(x1)
stress(x1, type="neumann")
##
x2 <- cbind(rbind(matrix(1,4,4),matrix(0,4,4)),
     rbind(matrix(0,4,4),matrix(1,4,4)))
stress.dist(x2)
stress.dist(x2, bycol=TRUE)
stress.dist(x2, type="neumann")
## Not run:
##
(res <- stress(x2, rows=c(1,8)))
rownames(x2) <- c(letters[1:7], "ä")
stopifnot(identical(res, stress(x2, rows=c("a","ä"))))
stopifnot(identical(res, stress(x2, rows=c("a",iconv("ä", to="latin1")))))

## End(Not run)
```

---

## summary.proximus *Summarizing Proximus Objects*

---

### Description

`summary` method for an object of class `proximus`.

### Usage

```
## S3 method for class 'proximus'
summary(object, ...)
```

### Arguments

| | |
|---|---|
| `object` | an object of class `proximus`. |
| `...` | further (unused) arguments. |

## Value

An object of class `summary.proximus` with the following elements:

| | |
|---|---|
| `nr` | the number of rows of the approximated matrix. |
| `nc` | the number of columns of the approximated matrix. |
| `error` | the relative error of the total approximation. |
| `fnorm` | the Frobenius norm of the total approximation. |
| `jsim` | the Jaccard similarity of the total approximation. |
| `valid` | the number of patterns that satisfy the mining constraints. |
| `pattern` | a `data.frame` of pattern summaries: |
| `pattern$Size` | the absolute size of the presence set. |
| `pattern$Length` | the number of `TRUE` values of the dominant pattern. |
| `pattern$Radius` | the Hamming radius of the presence set. |
| `pattern$Error` | the relative error of the presence set. |
| `pattern$Fnorm` | the Frobenius norm of the presence set. |
| `pattern$Jsim` | the Jaccard similarity of the presence set. |
| `pattern$Valid` | a logical indicating if the constraints are satisfied. |

## Warning

The function may change in future releases

## Author(s)

Christian Buchta

## See Also

`proximus` for obtaining a result object.

## Examples

```
### see proximus
```

townships *Bertin's Characteristics and Townships Data Set*

## Description

This data set was used to illustrate that the conciseness of presentation can be improved by reordering the rows and columns.

## Usage

```
data(townships)
```

## Format

A data frame with 16 observations on the following 10 variables.

`Township` a factor with levels `A B C D E F G H I J K L M N O P`

`High.School` a logical vector

`Agricultural.Coop.` a logical vector

`Railway.Station` a logical vector

`One.Room.School` a logical vector

`Veterinary` a logical vector

`No.Doctor` a logical vector

`No.Water.Supply` a logical vector

`Police.Station` a logical vector

`Land.Reallocation` a logical vector

## Details

`townships` is a data set with 16 `logical` variables indicating the presence (`TRUE`) or absence (`FALSE`) of characteristics of townships.

## References

Bertin, J. (1981) *Graphics and Graphic Information Processing*. Berlin, Walter de Gruyter.

## Examples

```
## see order.data.frame
```

Votes *Congressional Votes 1984 Data Set*

## Description

This data set includes votes for each of the U.S. House of Representatives Congressmen on the 16 key votes identified by the CQA. The CQA lists nine different types of votes: voted for, paired for, and announced for (these three simplified to yea), voted against, paired against, and announced against (these three simplified to nay), voted present, voted present to avoid conflict of interest, and did not vote or otherwise make a position known (these three simplified to an unknown disposition).

## Usage

```
data(Votes)
```

## Format

A data frame with 435 observations on the following 17 variables.

`handicapped-infants` a factor with levels `n` and `y`

`water-project-cost-sharing` a factor with levels `n` and `y`

`adoption-of-the-budget-resolution` a factor with levels `n` and `y`

`physician-fee-freeze` a factor with levels `n` and `y`

`el-salvador-aid` a factor with levels `n` and `y`

`religious-groups-in-schools` a factor with levels `n` and `y`

`anti-satellite-test-ban` a factor with levels `n` and `y`

`aid-to-nicaraguan-contras` a factor with levels `n` and `y`

`mx-missile` a factor with levels `n` and `y`

`immigration` a factor with levels `n` and `y`

`synfuels-corporation-cutback` a factor with levels `n` and `y`

`education-spending` a factor with levels `n` and `y`

`superfund-right-to-sue` a factor with levels `n` and `y`

`crime` a factor with levels `n` and `y`

`duty-free-exports` a factor with levels `n` and `y`

`export-administration-act-south-africa` a factor with levels `n` and `y`

`Class` a factor with levels `democrat` and `republican`

## Details

The records are drawn from:

*Congressional Quarterly Almanac*, 98th Congress, 2nd session 1984, Volume XL: Congressional Quarterly Inc. Washington, D.C., 1985.

It is important to recognize that NA in this database does not mean that the value of the attribute is unknown. It means simply, that the value is not "yea" or "nay" (see above).

The current version of the UC Irvine Machine Learning Repository Congressional Voting Records data set is available from doi:10.24432/C5C01P.

Blake, C.L. & Merz, C.J. (1998). UCI Repository of Machine Learning Databases. Irvine, CA: University of California, Department of Information and Computer Science. Formerly available from 'http://www.ics.uci.edu/~mlearn/MLRepository.html'.

## Examples

```
data(Votes)
summary(Votes)
## maybe str(Votes) ; plot(Votes) ...
```

# Index