Package ‘extracat’

February 19, 2015

Type Package
Title Categorical Data Analysis and Visualization
Version 1.7-1
Date 2014-10-21
Author Alexander Pilhoefer <alexander.pilhoefer@math.uni-augsburg.de>
Maintainer Alexander Pilhoefer <alexander.pilhoefer@math.uni-augsburg.de>
Description Categorical Data Analysis and Visualization
Depends R (>= 3.1.0)
Suggests iplots, ca, amap, MASS, proto
Imports grid, colorspace, hexbin, scales, ggplot2, reshape2, plyr, TSP
LazyLoad yes
LazyData yes
License GPL (>= 2)
NeedsCompilation yes
Repository CRAN
Date/Publication 2014-10-21 13:34:48

R topics documented:

ahist ................................................................. 3
approx.dcor ........................................................ 4
arsim ................................................................. 6
barysort ............................................................. 7
BCC ................................................................. 8
BCI ................................................................. 9
boxplot2g .......................................................... 10
Burt ................................................................. 11
carcustomers ....................................................... 12
CBCI ............................................................... 14
cfcl ................................................................. 15
cfuctile ............................................................ 16
<table>
<thead>
<tr>
<th>R topics documented:</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmat</td>
</tr>
<tr>
<td>cohen</td>
</tr>
<tr>
<td>combcl</td>
</tr>
<tr>
<td>CPScluster</td>
</tr>
<tr>
<td>cutbw</td>
</tr>
<tr>
<td>dcorMVdata</td>
</tr>
<tr>
<td>dcorMVtable</td>
</tr>
<tr>
<td>dendro</td>
</tr>
<tr>
<td>dmc</td>
</tr>
<tr>
<td>extracat</td>
</tr>
<tr>
<td>facetshade</td>
</tr>
<tr>
<td>fluctile</td>
</tr>
<tr>
<td>GeneEx</td>
</tr>
<tr>
<td>getbw</td>
</tr>
<tr>
<td>getcolors</td>
</tr>
<tr>
<td>getIs</td>
</tr>
<tr>
<td>getIs2</td>
</tr>
<tr>
<td>getpath</td>
</tr>
<tr>
<td>gsac</td>
</tr>
<tr>
<td>heattile</td>
</tr>
<tr>
<td>hexpie</td>
</tr>
<tr>
<td>idat</td>
</tr>
<tr>
<td>imat</td>
</tr>
<tr>
<td>innerval</td>
</tr>
<tr>
<td>itab</td>
</tr>
<tr>
<td>JBCI</td>
</tr>
<tr>
<td>kendalls</td>
</tr>
<tr>
<td>ME</td>
</tr>
<tr>
<td>olives</td>
</tr>
<tr>
<td>optile</td>
</tr>
<tr>
<td>optME</td>
</tr>
<tr>
<td>plants</td>
</tr>
<tr>
<td>qBCI</td>
</tr>
<tr>
<td>quickfechner</td>
</tr>
<tr>
<td>regmax</td>
</tr>
<tr>
<td>rmrb</td>
</tr>
<tr>
<td>rmbrmat</td>
</tr>
<tr>
<td>scpccp</td>
</tr>
<tr>
<td>setcover</td>
</tr>
<tr>
<td>sortandcut</td>
</tr>
<tr>
<td>steptile</td>
</tr>
<tr>
<td>subtree</td>
</tr>
<tr>
<td>subtable</td>
</tr>
<tr>
<td>tfluctile</td>
</tr>
<tr>
<td>untableSet</td>
</tr>
<tr>
<td>USR</td>
</tr>
<tr>
<td>visid</td>
</tr>
<tr>
<td>WBCI</td>
</tr>
</tbody>
</table>
ahist

Histogram using active bins

Description

A standard histogram using getbw to compute the binwidth and breakpoints.

Usage

ahist(x, k = NULL, m = NULL, fun = "qplot", col = "grey", ival = NULL)

Arguments

x 
A numeric vector.

k 
The desired number of active bins. A bin is active if it contains at least min_n observations. The default is $k \leftarrow 1 + 2 \cdot \text{ceiling}(\log(N)/\log(2))$.

m 
The minimum number of observations necessary for a bin to count as an active bin. Defaults to $m = \max(\log(N/10)/\log(10),1)$.

fun 
Either "qplot" or "hist".

col 
The color for the bars.

ival 
If this is set to a numeric value in $(0, 1)$ then $x$ is trimmed according to interval(x, p = ival).

Value

The ggplot object.

Note

This is purely experimental at this time.

Author(s)

Alexander Pilhoefer

See Also

getbw, cutbw
Examples

```r
ahist(rnorm(100))
ahist(rnorm(1000))
ahist(rnorm(10000))

ahist(rexp(100))
ahist(rexp(1000))
ahist(rexp(10000))
```

```r
# Not run:
ahist(rcauchy(1000))
ahist(rcauchy(1000), ival = 0.95)
x <- c(rnorm(400), rnorm(200, mean=6))
ahist(x)
x <- c(rnorm(400), rnorm(200, mean=16))
ahist(x)
x <- c(rnorm(400), rnorm(200, mean=32))
ahist(x)
```

```r
# End(Not run)
```

approx.dcor  

*Distance Correlation Approximation*

Description

Computes the distance correlation for two variables using an approximation based on binning and `wdcor.table`. The approximation underestimates the true value by a small error depending on the number of bins. (In simulations with the default of 50 bins the average error was about 0.001.)

Usage

```r
approx.dcor(x, y, n = 50, ep = 1, bin = "eq")
```

Arguments

- `x`: A numeric vector.
- `y`: A numeric vector.
- `n`: The number of bins per variable.
- `ep`: The euclidean distances are taken to the power of ep.
- `bin`: Either "eq" or "q" for equidistant breakpoints or quantile breakpoints.
**Value**

The correlation value which is between 0 and 1.

**Author(s)**

Alexander Pilhoefer

**References**


**Examples**

```r
## Not run:

# The straightforward way of approximating the distance correlation fails:
# for instance the computation of dcov for a random sample with 4000 observations
# takes pretty long but drawing samples of 500, 1000 or even 2000 observations
# leads to relatively big errors.
# The approximation via approx.dcov is very fast and for
# n = 50 or n=100 the results are very close to the truth

require(energy)
x <- rnorm(4000, mean=10, sd=3)
y <- rnorm(1, sd=0.01)*x**3 + rnorm(1, sd=0.1)*x**2 + rnorm(1)*x + rnorm(4000, sd=4)

system.time(dd <- dcov(x, y))

system.time(dd$ <- wdcov(x, y))[[3]]

dd - dd$

system.time(da100 <- approx.dcov(x, y, 100))[[3]]
da100-dd$

# For a smaller sample size we can try out how good the approximation really is:
test <- replicate(100,{
N <- 1000
x <- rnorm(N, mean=10, sd=3)
y <- rnorm(1, sd=0.01)*x**3 + rnorm(1, sd=0.1)*x**2 + rnorm(1)*x + rnorm(N, sd=4)

dd <- wdcov(x, y)
dd$25 <- approx.dcov(x, y, 25)
dd$50 <- approx.dcov(x, y, 50)
dd$100 <- approx.dcov(x, y, 100)
dd$75 <- approx.dcov(x, y, 75)

dd$25c <- approx.dcov(x, y, 25, correct = TRUE)
dd$50c <- approx.dcov(x, y, 50, correct = TRUE)
dd$100c <- approx.dcov(x, y, 100, correct = TRUE)
```

```r
```
dd75c <- approx.dcor(x, y, 75, correct = TRUE)
c(2*dd, dd25, dd50, dd75, dd100, dd25c, dd50c, dd75c, dd100c)-dd

rm<-apply(test, 1, mean)

plot( seq(25, 100, 25), rm[2:5], type="l",
ylim = c(min(rm), abs(min(rm))), xlab = "No. of bins per axis", ylab = "error")
points( seq(25, 100, 25), rm[2:5], pch=19)
points( seq(25, 100, 25), rm[6:9], type="1", col=2)
points( seq(25, 100, 25), rm[6:9], pch=19, col=2)
abline(h=0,lwd=3)
legend(25, abs(min(rm)), legend=c("raw value after binning","corrected value"),
col=1:2,lwd=3)

## End(Not run)

arsim

block-structured arrays

Description

Generates an array or matrix that includes k fully separated block-clusters.

Usage

arsim(n, dim, k, noise = 0, shuffle = TRUE, v = 0.1, minc = 1,
exp.prop = NULL, min.prop = 1/dim/4, noise.type = "s",
dimnames=list(LETTERS,1:max(dim)))

Arguments

n The number of observations in the array.
dim The dimension of the array.
k The number of clusters. 1 for no clusters.
noise The proportion of noise among the observations. There are two choices for noise.type.
shuffle Whether or not to shuffle the original category orders randomly.
v A variability parameter for the assignment of the observations to the block clusters. Small values lea
minc The minimum number of categories each cluster must have in each variable. E.g. minc = 2 means, that each block cluster covers at least 2 categories in each dimension.
exp.prop Optional: expected proportions of the observations which fall into the block clusters.
min.prop Minimum proportion of observations in each cluster.
noise.type Either "s" or "I". The noise type "s" means that n*noise observations are drawn at random from the block-diagonal matrix. Then for these observations the category labels are permuted at random. "I" adds noise in form of a random sample from the independence matrix with the same marginal totals as the block matrix.

dimnames A list of 2: The first entry defines the variable labels (default: A,B,C,...) and the second entry defines the category labels (default 1:k).

Details
Not a very sophisticated way of generating random arrays but it serves for tests and illustrations of the other functions.

Value
A simulated data array.

Examples
A <- arsim(1000, c(12,12), 3, shuffle = FALSE)
fluctile(A)
A <- arsim(1000, c(12,12), 3, shuffle = FALSE, dimnames = list(NULL,letters))
dimnames(A)
A <- arsim(4000, c(11,7,5), 3, shuffle = TRUE, dimnames = list(0:2,letters))
dimnames(A)

## Not run:
A2<- arsim(1000, c(12,12,12), 3, shuffle = FALSE)
fluctile3d(A2, shape ="oct")

## End(Not run)
Value

The reordered matrix.

Examples

```r
# a good and quick solution:
a <- arsim(2000, c(24, 24), 6, noise=0.4)
fluctile(a2<-barysort(a))
BCI(a2)

# which is near
a3 <- optile(a, iter=100)
BCI(a3)

## Not run:
a <- arsim(64000, c(256, 256), 16, noise=0.4)
s1 <- system.time( bci1 <- BCI(a1<-optile(a, fun = "barysort", foreign=".Call", iter = 1)) )[[3]]
s2 <- system.time( bci2 <- BCI(a2<-optile(a, iter=1)))[[3]]
s3 <- system.time( bci3 <- BCI(a3<-optile(a, fun="IBCC",iter=1)))[[3]]

## End(Not run)
```

---

**BCC**

*The Bertin Classification Criterion*

Description

Computes the Bertin Classification Criterion for a contingency table of any dimensions.

Usage

```r
BCC(x)
```

Arguments

- `x` A data matrix, table or array.

Details

The BCC counts the number of observation pairs which differ in all variables but are not fully concordant, (i.e. neither of the two observations of each pair is larger than the other in all variables).

Value

The criterion value.
BCI

Author(s)
Alexander Pilhoefer

See Also
kendalls

Examples
M <- arsim(1000, c(12,12), 3)
BCC(M)

M2 <- optile(M, iter = 100)
BCC(M2)

Description
Computes the Bertin Classification Index for a contingency table of any dimensions.

Usage
BCI(x)

Arguments
x A data matrix, table or array.

Details
The BCI is the Bertin Classification Criterion (BCC) normalized by the BCC value under independence.

Value
The criterion value.

Author(s)
Alexander Pilhoefer

See Also
kendalls
Examples

# for an unoptimized matrix we take the minimum of BCI(M) and BCI(M[,12:1])
M <- arsim(1000, c(12,12), 3)
min(BCI(M), BCI(M[,12:1]))

# an strongly related alternative (for two-way data)
kendalls(M)

M2 <- optile(M, iter = 100)
BCI(M2)
kendalls(M2)

M3 <- arsim(100000, c(12,13,15), 4, noise=0.2, shuffle=FALSE)
BCI(M3)

---

**boxplot2g**

*2D projection boxplots*

---

**Description**

Contours derived from the boxplot marks of data projections.

**Usage**

```r
boxplot2g(x, y = NULL, groups = NULL, smooth = loess,
           smooth.args = list(span = 0.1), colv = NULL, alpha = 1, n = 360, ...)
```

**Arguments**

- **x**: Numeric x variable.
- **y**: Numeric y variable.
- **groups**: A grouping variable or NULL.
- **smooth**: A smoothing function such as `loess` or FALSE.
- **smooth.args**: Additional arguments for the smoother, such as `span`.
- **colv**: A color vector with one color for each group.
- **alpha**: The alpha-blending value for the points of the scatterplot.
- **n**: The number of rotation vectors. Defaults to 360 which means 1-degree steps.
- **...**: Further args.

**Details**

The data is first centered and standardized via multiplication with the inverse of the covariance matrix. Then projections of the data onto different direction vectors (0 to 360 degree) are computed. The boxplot marks (median, box, whisker) are computed for each such projection and finally re-transformed again using the covariance matrix. Optionally the projected marks are then smoothed, e.g. via `smooth=loess`. 
**Value**

The ggplot object.

**Note**

Please note that I do not claim that the resulting contours have any statistically useful characteristics! Usage at one’s own risk!

**Author(s)**

Alexander Pilhoefer

**See Also**

contour

**Examples**

```r
## Not run:
data(olives)
boxplot2g(olives$palmitoleic, olives$oleic, olives$Region)
boxplot2g(olives$palmitoleic, olives$oleic, olives$Area, alpha=0.5, smooth.args=list(span=0.3), colv = rainbow(9))

## End(Not run)
```

---

**Description**

The Burt matrix is a quadratic matrix where each row and column corresponds to a category in one of the variables. The entries of the matrix are the frequencies of the corresponding combination of categories.

**Usage**

`Burt(x)`

**Arguments**

- `x`: A dataframe with factor variables or a contingency table.

**Value**

A matrix.
Author(s)

Alexander Pilhoefer

See Also

idat, imat

Examples

```r
require(MASS)
Burt(housing)
th <- xtabs(Freq~Sat+Infl+Type, data = housing)
Burt(th)
```

### Description

The car customers dataset from 1983

This dataset is taken from the website of the Department of Statistics, University of Munich. The data are based upon a poll from a german car-company. In 1983 questionnaires were sent to 2000 customers, who had purchased a new car approximately three months earlier. The point of interest was the degree of satisfaction, reasons for the particular choice, consumer profile, etc. Participation was of course voluntary. Only 1182 persons answered the questions and after removing forms with "missing values" only 793 questionnaires remained. Each form contained 46 questions, which resulted in a dataset of 46 covariates with 793 observations each. Due to the abundance of ordinal and categorical covariates the dataset is particularly suited for generalized linear models.

### Usage

```r
data(carcustomers)
```

### Format

A data frame with 774 observations on the following 47 variables.

- `model`: a factor with levels A B C D
- `gear`: a factor with levels 4-gear 5-gear (overdrive) 5-gear (sport) Automatic
- `lease`: a factor with levels bought leased
- `usage`: a factor with levels business private private and business
- `premod`: a factor with levels Audi BMW 3er BMW 5er BMW 7er Ford Mercedes Benz Opel other origin Volkswagen
- `other`: a factor with levels No Yes, both Yes, other manufact Yes, same manufact.
- `testdrv`: influence on buying decision: testdrive
- `promotion`: influence on buying decision: promotion
exp  influence on buying decision: experience
recom  influence on buying decision: recommendation
clear  influence on buying decision: clearness
eco  influence on buying decision: economical aspects
drvchar  influence on buying decision: driving character
service  influence on buying decision: service
interior  influence on buying decision: interior
quality  influence on buying decision: overall quality
tech  influence on buying decision: technical aspects
evo  influence on buying decision: evolution
comfort  influence on buying decision: comfort
reliab  influence on buying decision: reliability
handling  influence on buying decision: handling
prestige  influence on buying decision: prestige
concept  influence on buying decision: overall concept
char  influence on buying decision: character
power  influence on buying decision: engine power
valdecr  influence on buying decision:value decrease
styling  influence on buying decision: styling
safety  influence on buying decision:safety
sport  influence on buying decision: sportive
fcons  influence on buying decision: fuel consumption
space  influence on buying decision: available space
sat  overall satisfaction with the car: 1(very satisfied) to 5(not satisfied)
adv1  satisfaction with concept and styling: a factor with levels does not suit neither nor suits
adv2  satisfaction with body/bare essentials: a factor with levels does not suit neither nor suits
adv3  satisfaction with chassis/drive/gearshift: a factor with levels does not suit neither nor suits
adv4  satisfaction with engine/power: a factor with levels does not suit neither nor suits
adv5  satisfaction with electronics: a factor with levels does not suit neither nor suits
adv6  satisfaction with financial aspects: a factor with levels does not suit neither nor suits
adv7  satisfaction with equipment: a factor with levels does not suit neither nor suits
spoco  balance variables: a factor with levels comfort could be better handling could be better well balanced
favor  usual driving style: a factor with levels economical extreme normal powerful
speed  usual speed (Autobahn): a factor with levels >110 mph 60–80 mph 81–90 mph 91–110 mph
satisfaction with fuel consumption: a factor with levels Appropriate Definitely too high Just okay Pleasingly low

sex customer’s gender: a factor with levels Female Male

prof customer’s profession: a factor with levels Employee/Workman Free lanced Self employed

family customers’s family type: a factor with levels >3 persons 1-2 persons

Freq the weighting variable

Source

http://www.stat.uni-muenchen.de/service/datenarchiv/auto/auto_e.html

Examples

data(Autos)
### maybe str(Autos) ; plot(Autos) ...

---

**CBCI**

*The Conditional Independence Bertin Classification Index*

**Description**

Computes the Conditional Independence Bertin Classification Index which uses conditional independence as a reference for normalization. High values indicate that the BCC is not far from the expectation if we know the two marginal 2D BBC values.

**Usage**

`CBCI(x, r = 1, joint.order = FALSE)`

**Arguments**

- `x` The 3D table with non-negative entries.
- `r` The index of the conditioning variable, e.g. `r = 1` uses the table with variables 2 and 3 conditionally independent given 1 for normalization.
- `joint.order` Whether or not to use a joint ordering for all variables. Otherwise the pairwise values are computed using separate reorderings.

**Details**

The BCI of a 3D table but instead of the total independence case the conditional independence case is used for normalization.

**Value**

Numeric value in $[0,1]$. 

cfcl

Author(s)

Alexander Pilhoefer

See Also

BCI, JBCI, WBCI

Examples

```r
## Not run:
A <- optile(arsim(10000, c(11,12,13), 4, 0.1))

BCI(A)

CBCI(A, 1, TRUE)
CBCI(A, 1, FALSE)

## End(Not run)
```

cfcl

Extract clusters from cfuctile

Description

Extract clusters from cfuctile

Usage

```r
cfcl(x, y = NULL, ll)
```

Arguments

- **x**: vector or dataframe.
- **y**: if `x` is a vector, `y` needs to be specified.
- **ll**: The list with the names of the levels which are combined.

Value

A 2-column dataframe with the cluster factors.

See Also

cfuctile
Examples

```r
a <- arsim(2000, c(12,17),5, noise=0.2,shuffle = FALSE)
cfa <- cfluctile(a)

da <- as.data.frame(a)
clusters <- cfcl( da, ll = cfa)

dev.new()
fluctile(xtabs(da$freq-clusters[,1] + clusters[,2]))
table(combcl(clusters))
```

---

**cfluctile**

*Pseudo-Diagonal Partitioning for two-way tables*

**Description**

Identifies a diagonal of block-clusters in a two-way table using a top-down-partitioning algorithm then plots the table and adds the clusters as rectangles.

**Usage**

```r
cfluctile(x, tau0 = NULL, method = "Kendall", nsplit = NULL,
          maxsplit = NULL, trafo = I, gap.prop = 0.2, floor = 0,
          rev.y = FALSE, add = FALSE, shape = "r", just = "c",
          dir = "b", plot = TRUE, rect.opt = list(), border =
          NULL, label = TRUE, lab.opt = list(), tile.col =
          hsv(0.1, 0.1, 0.1, alpha = 0.6),
          tile.border = NA, bg.col = "lightgrey", ...)
```

**Arguments**

- **x**  
  A 2-way table or matrix.

- **tau0**  
  The minimum acceptable value of Kendall’s tau, Cohen’s Kappa or WBCI. Defaults to the criterion of the input matrix `x`.

- **method**  
  Either "Kendall" for Kendall’s Tau, "Cohen" for Cohen’s Kappa, "WBCI" for the Weighted Bertin Classification Criterion and "s" for the minimum residual method.

- **nsplit**  
  The number of splits to make. `tau0` is ignored.

- **maxsplit**  
  The maximum number of splits.

- **trafo**  
  A transformation of the table entries for the plot, but not for the computation of the splits. E.g. `trafo = function(z) log(1+z)`.

- **gap.prop**  
  Proportion of the gaps between rows/columns.

- **floor**  
  Floor censored zooming: all cases will be plotted but only those with a frequency of at least `floor` will be considered for the clustering.
rev.y revert the y axis.
add Whether to make a new plot or to add to an existing one.
shape The shape of the objects. See fluctile.
just See fluctile.
dir See fluctile.
plot Whether or not to create a plot via fluctile.
rect.opt A list with optional parameters for the rectangles. Possible parameters are:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col</td>
<td>The rectangle color.</td>
</tr>
<tr>
<td>lwd</td>
<td>The line width. Default is &quot;red&quot;.</td>
</tr>
<tr>
<td>lty</td>
<td>The line type. Default is 1 (solid).</td>
</tr>
<tr>
<td>fill</td>
<td>The color to fill the rectangles. Defaults is NULL. A sensible choice is for instance alpha(col, 0.1).</td>
</tr>
<tr>
<td>border</td>
<td>The white margins around the plot which are also used for the labels. Must be a vector of length 1, 2 or 4 with values in [0, 1]. Default is border = 0.05.</td>
</tr>
<tr>
<td>label</td>
<td>Whether or not to draw labels.</td>
</tr>
<tr>
<td>lab.opt</td>
<td>Label options, see fluctile.</td>
</tr>
<tr>
<td>tile.col</td>
<td>Color(s) for the tiles, see fluctile.</td>
</tr>
<tr>
<td>tile.border</td>
<td>Border color for the tiles. Can also be a matrix.</td>
</tr>
<tr>
<td>bg.col</td>
<td>Color for the background of the cells, see fluctile.</td>
</tr>
<tr>
<td>...</td>
<td>dots</td>
</tr>
</tbody>
</table>

Details

This function calls fluctile to create a 2-way fluctuation diagram and then adds cluster rectangles to it. The cluster rectangles are computed in the following way:

The algorithm cuts the data matrix once horizontally and once vertically and computes a criterion for the 2x2 table consisting of the sums of the four parts that resulted from the cuts. This is done for all possible horizontal and vertical cuts and the best combination is chosen. Then the same procedure is applied to the bottom right submatrix and the top left submatrix. The algorithms stops if no cut yields a better criterion value than tau0.

Value

invisible(TRUE)

Note

This was part of the Google Summer of Code 2011.
Author(s)

Alexander Pilhoefer
Department for Computer Oriented Statistics and Data Analysis
University of Augsburg
Germany

See Also

optile, sortandcut, tfluctile

Examples

M <- arsim(10000, c(30,40), 8, noise = 0.4)
cfluctile( M2 <- optile(M, iter=20) )

cfluctile( M3 <- sortandcut(M) )
cfluctile( M3, nsplit = 4 )
cfluctile( M3, maxsplit = 12 )
cfluctile( M3, tau0 = 0.8 )

---

cmat pairwise association matrix

Description

Computes pairwise BCI values via qBCI.

Usage

cmat(x, sort = TRUE, crit = BCI, k = 5, iter = 20,
    p = NULL, jitter = TRUE, freqvar = NULL, diag = NULL, 
    fun = "BCC", foreign = NULL)

Arguments

- x: A data.frame with factor variables or numeric variables which will be transformed to ordinal interval variables via cut. The breakpoints are quantiles of the variables such that for each pair of numeric variables the expected number of observations in each combination of intervals is at least k.
- sort: Whether or not to sort the pairwise tables via optile.
- crit: The criterion function, e.g. kendalls, BCI, WBCI or wcor.
The minimum expected number for each cell after quantile binning. See also `qBCI`.

- `iter`: An optile parameter.
- `p`: The quantile distance. See `qBCI`.
- `jitter`: Whether or not to use jittering in order to avoid ties. This is equivalent to a random assignment of ranks to observations with the same value.
- `freqvar`: Optional weights, e.g. a frequency variable.
- `diag`: An optional value for the diagonal. Avoids unnecessary function calls for the diagonal elements. E.g. `diag = 0` for `crit = BCI` or `diag = 1` for `crit = kendalls` makes sense.
- `fun`: See `optile`.
- `foreign`: See `optile`.

Details

Uses pairwise complete cases only!

Value

A symmetric matrix.

Author(s)

Alexander Pilhoefer

See Also

- `qBCI`, See `wdcor`.

Examples

```r
## Not run:
m1 <- cmat(olives)
fluctile(1 - m1, shape="o")

## End(Not run)
```

**Description**

Cohen’s Kappa for quadratic and non-quadratic matrices using L1-weights.

**Usage**

`cohen(x)`
**combcl**

**Description**

Combines variables obtained via `cfcl` and `cfclustile` to a single factor variable with one level per block-cluster and one level for the rest.

**Usage**

`combcl(x)`

**Arguments**

`x` A matrix, table or data.frame. All variables should have the same number of categories.

**Value**

A factor variable with 1 level per diagonal element and 1 level for the rest.

**Examples**

```r
a <- arsim(2000, c(12,17), 5, noise=0.2, shuffle = FALSE)
cfa <- cfclustile(a)
da <- as.data.frame(a)
clusters <- cfcl(da, ll = cfa)

dev.new()
fluctile(xtabs(da$Freq-clusters[,1] + clusters[,2])))
```
Description

Different hierarchical clusterings and k-means clusterings as well as a model-based clustering have been applied to several financial variables for a random sample of ten thousand observations.

Usage

data(CPScluster)

Format

A data frame with 10000 observations on the following 39 variables.

Age   a numeric vector
Sex   a factor with levels female male
Race  a factor with levels Black White
Ethnic a factor
Marital.Status a factor
Kind.of.Family a factor
Classical a factor with levels All other Classical Husband-Wife family
Family.Type a factor
Number.of.Persons.in.Family a numeric vector
Number.of.Kids a numeric vector
Education.of.Head a factor
Labor.Status a factor
Class.of.Worker a factor
Working.Hours a numeric vector
Income.of.Head a numeric vector
Family.Income a numeric vector
Taxable.Income a numeric vector
Federal.tax a numeric vector
Family.sequence.number a numeric vector
State a factor
Division a factor
Region  a factor with levels Midwest North East South West
hc4  a numeric vector
hc6  a numeric vector
hc8  a numeric vector
hc12 a numeric vector
hcs4 a numeric vector
hcs6 a numeric vector
hcs8 a numeric vector
hcs12 a numeric vector
hcw4 a numeric vector
hcw6 a numeric vector
hcw8 a numeric vector
hcw12 a numeric vector
km4  a numeric vector
km6  a numeric vector
km8  a numeric vector
km12 a numeric vector
mc12 a numeric vector

Examples

```r
data(CPScluster)
## maybe str(CPScluster) ; plot(CPScluster) . . .
```

---

**cutbw**

*Active binning*

**Description**

Uses `cut` with breakpoints derived by `getbw`.

**Usage**

```r
cutbw(x, k = NULL, min_n = NULL, warn = FALSE)
```

**Arguments**

- **x**  
  A numeric variable.
- **k**  
  The desired number of active bins. A bin is active if it contains at least `min_n` observations. The default is `k <- 1 + 2*ceiling(log(N)/log(2))`.
- **min_n**  
  The minimum number of observations necessary for a bin to count as an active bin. Defaults to `min_n = max(log(N/10)/log(10), 1)`.
- **warn**  
  Whether or not to print a warning if the desired number of bins is not possible.
**dcormvdata**

**Value**

An ordinal factor variable.

**Note**

Experimental.

**Author(s)**

Alexander Pilhoefer

**See Also**

getbw, ahist

**Examples**

```r
y <- cutbw(c(rnorm(200), rnorm(100, mean = 0)), k = 30, min_n = 1)
barplot(table(y))
```

---

**Description**

Computes the distances within two sets of variables and the corresponding distance correlation.

**Usage**

```r
dcormvdata(x, ind = 1, method = "euclidean", approx = FALSE)
```

**Arguments**

- `x`: The `data.frame` which should only contain non-factor variables. For factor variables use `xtabs` in combination with `dcormVtable`.
- `ind`: The indices for the first set of variables. The second set consists of all remaining variables.
- `method`: The method for `dist`.
- `approx`: FALSE for no approximation via binning or an integer value for the number of bins.

**Value**

The distance correlation between 0 and 1 for the distances from the two sets of variables.

**Note**

This code has not been tested thoroughly and may still contain errors.
Author(s)
Alexander Pilhoefer

See Also
dcorMVtable, wdcor, approx.dcor

Examples

```r
## Not run:
sol <- scale(olives[,3:8])
dcorMVdata(sol,ind=1)

p1 <- princomp(sol)
sol1 <- cbind(sol,p1$scores[,1])
sol2 <- cbind(sol,p1$scores[,2])
sol3 <- cbind(sol,p1$scores[,1:2])

dcorMVdata(sol1,ind=7)
dcorMVdata(sol2,ind=7)
dcorMVdata(sol3,ind=7:8)
# how about principal curves?

## End(Not run)
```

---

**dcorMVtable**

Multivariate Distance Correlation for two sets of variables

Description

Computes the distances within two sets of variables and the corresponding distance correlation.

Usage

dcorMVtable(x, ind = 1, method = "euclidean")

Arguments

- **x** A contingency table of class `table`.
- **ind** The indices for the first set of variables. The second set consists of all remaining variables.
- **method** The method for `dist`

Value

The distance correlation between 0 and 1 for the distances from the two sets of variables.
Note

This code has not been tested thoroughly and may still contain errors.

Author(s)

Alexander Pilhoefer

See Also

dcorMVdata, wdcor, approx.dcor

Examples

```r
## Not run:
A2 <- arsim(2000,c(8,9),5,0.1)
A2 <- optile(A2, iter=100)
BCI(A2)
wdcor(A2)

p1 <- runif(11)+0.1
p1 <- p1/sum(p1)
A2b <- apply(A2,1:2,function(z) rmultinom(1,z,p1))

# now the first variable is roughly independent from the other two:
dcorMVtable(as.table(A2b),ind = 1)

# here the third variable is NOT independent from the others:
A3 <- arsim(2000,c(8,9,11),5,0.1)
A3 <- optile(A3, iter=100)
BCI(A3)
dcorMVtable(A3,ind = 3)
```

## End(Not run)

---

dendro  Waterfall Dendrogram

Description

Draws an alternative to the dendrogram using cpcp coordinates. Colors according to a specific number of clusters make the interpretation easier. Also splits which follow each other within a margin of min.gap (proportion of maximum height) can be displayed by boxes.

Usage

dendro(x, k = 30, color.id = k - 2, label = FALSE, opts = list(),
min.gap = 0.01, spline = FALSE, ...)
Arguments

- **x**: A hierarchical clustering object.
- **k**: The maximum number of clusters to plot. Possible are values up to $N-1$ but should usually be smaller.
- **color.id**: The number of clusters for the coloring.
- **label**: Whether or not to draw observation labels. Makes sense for small datasets.
- **opts**: Graphics and colour parameters such as lwd, ps or alpha.
- **min.gap**: Joins which are closer than min.gap from each other will be packed and displayed as a box.
- **spline**: Whether or not to use spline curves instead of straight line connections between the points.
- **...**: dots

Value

**TRUE**

Examples

```r
## Not run:
library(amap)
hc <- hcluster(USArrests)
# the full plot:
dendro(hc, k = 24, min.gap = 0.00)

# aggregation splits within 0.02 maximum height
dendro(hc, k = 24, min.gap = 0.02)

# the same graphic with spline curves instead of straight lines.
dendro(hc, k = 24, min.gap = 0.02, spline = TRUE)

# olive oil data
sx <- scale(olives[, -c(1,2,11)])
hc <- hcluster(sx)
plot(hc)
dendro(hc, 120, color.id = 6, min.gap=0.005)
dendro(hc, 120, color.id = 6, min.gap=0.1)
dendro(hc, 120, color.id = 6, min.gap=0.1, spline = TRUE)

## End(Not run)
```
Description
five insurance variables from the dmc 2009 dataset, which have a ordinal structure which has been lost somehow. Can we find it again?

Usage
data(dmc)

Format
A data frame with 693 observations on the following 6 variables.

- eiw_scr a factor with levels 5 6 4 3 2 1
- eih_scr a factor with levels 6 3 5 1 4 2
- ifi_scr a factor with levels 4 3 5 2 1 6
- tec_scr a factor with levels 5 1 3 2 4 6
- klv_scr a factor with levels 2 5 6 1 3 4
- Freq a numeric vector

Details
The Data Mining Cup (dmc) is a competition for students.

Examples
data(dmc)

extracat

Description
This package offers a variety of functions that can be used for categorical data analysis or at least have to do with categorical data.

Among the most interesting features are

- **rmb** RMB plots visualize contingency tables.
  The function offers different visualizations of conditional distributions and their corresponding weights (frequencies) including multiple barcharts, spineplots and piecharts. There are different ways of displaying the residuals from statistical models.

- **scpcp** A static version of CPCP plots with labeling and color options.
The OPTILE interface was developed for the Google Summer of Code 2011. It offers a variety of reordering techniques for contingency tables. The reordering of the categories not only improves visualizations. A matrix of RMB plots not unlike a Scatterplot matrix (SPLOM) is produced. It uses binning for continuous data. After a hexagonal binning of x and y a third categorical target variable is displayed via piecharts (or embedded hexagons) within the hexagons. This avoids problems with overplotting. A very fast implementation of the fechnerian scaling technique, which computes a fechnerian distance matrix from a (dis.)similarity matrix. The technique is equivalent to a shortest path algorithm. Offers two- or multidimensional fluctuation diagrams and multiple barcharts. Simulates categorical data arrays of any dimension. The number of observations, the number of block clusters and the noise level and type can be chosen.

Details

Package: extracat
Type: Package
Version: 1.6-4
Date: 2013-12-11
License: -
LazyLoad: yes

Author(s)

Alexander Pilhoefer
Department for Computer Oriented Statistics and Data Analysis
University of Augsburg
Germany

Maintainer: Alexander Pilhoefer <alexander.pilhoefer@math.uni-augsburg.de>

References

**facetshade**

**Description**

This function makes it possible to create ggplots using `facet_grid` with a plot of the complete data in the background of each facet. There are two options: If `geom` is specified then the background data is put into a separate layer. The original data is stored in the main object. The other option is to not specify a `geom`. In this case the modified data is stored in the main body. See examples.

**Usage**

```r
facetshade(data, mapping, f, geom, geom.mapping, bg.all = TRUE, keep.orig = FALSE, ...)
```

**Arguments**

- `data` The dataframe used for the background plots in the first layer.
- `mapping` The aesthetic mapping constructed via `aes`.
- `f` The formula specifying the grid for `facet_grid` or a facet/wrap.
- `geom` The geom used for the shade.
- `geom.mapping` Aesthetics for the shade.
- `bg.all` Whether or not to use all data points for each background plot. If `FALSE` then the data for the background is the complement of the data in the facet.
- `keep.orig` Logical. Whether to keep the original faceting variables defined by `f`. Those are renamed by adding `orig.` as a prefix. For example `f = .~variable` will work fine with `group = orig.variable`. See FinCal example.
- `...` Further arguments for the background layer or the main `ggplot` object.

**Value**

A `ggplot` object.

**Author(s)**

Alexander Pilhoefer

**See Also**

- `facet_grid`

**Examples**

```r
# produces a modified data.frame mdata and returns:
# ggplot(data = mdata, mapping, ... ) + facet_grid(f)
require(scales)
require(ggplot2)

# facetshade object:
fs1 <- facetshade(data = olives, aes(x = palmitoleic, y = oleic),
                 f = .~Region )
```
# only the background-data
fs1 + geom_point( colour = alpha(1, 0.2) )

# the actual data added in a second layer:
fs1 + geom_point( colour = alpha(1, 0.2) ) +
geom_point( data = olives )

# now again with colours:
fs1 + geom_point( colour = alpha(1, 0.2) ) +
geom_point( data = olives, aes(colour = Area) )

# a different geom for the background-plot:
fs1 + geom_density2d(colour=alpha(1,0.1)) +
geom_point( data = olives, aes(colour = Area) )
## Not run:
## OPTION 2: specify geom in facetshade call:
fs1b <- facetshade( data = olives, aes(x = palmitoleic, y = oleic),
f = .~Region , geom = geom_point)
fs1b + geom_point(aes(colour = Area))

## End(Not run)

## compare with complement:
fs2 <- facetshade( data = olives, aes(x = palmitoleic, y = oleic),
f = .~Region , bg.all = FALSE)

fs2 + geom_density2d(colour=alpha(1,0.1)) +
geom_point( data = olives, aes(colour = Area) )
## Not run:
## OPTION 2: specify geom in facetshade call:
fs2b <- facetshade( data = olives, aes(x = palmitoleic, y = oleic),
f = .~Region , geom = geom_density2d, bg.all = FALSE)
fs2b + geom_point(aes(colour = Area))

## End(Not run)

# a second dataset:
## Not run:
data(EURO4PlayerSkillsSep11, package="SportsAnalytics")
e4 <- subset(EURO4PlayerSkillsSep11,Attack > 0 & Defence > 0)

fs3 <- facetshade( data = e4, aes(x = Attack, y = Defence),
f = .~Position , compare.all = TRUE)

fs3 + geom_point( colour = alpha(1, 0.1) ) +
geom_point( data = e4, aes(colour = Position) ,alpha=0.3)

fs3 + geom_bin2d( colour = alpha(1, 0.1) ) +
geom_point( data = e4, aes(colour = Position) ,alpha=0.3)
# now with two facet variables
fs4 <- facetshade(data = e4, aes(x = Attack, y = Defence),
      f = Position~Side, compare.all = TRUE)

fs4 + geom_point( colour = alpha(1, 0.1) ) +
geom_point( data = e4, aes(colour = Position))

## End(Not run)

## Not run:
library(FinCal)
sh13 <- get.ohlc.google(symbols=c("AAPL","GOOG","IBM", "MSFT"),
                        start="2013-01-01",end="2013-12-31")

# OPTION 1
require(reshape2)
SH13 <- data.frame(date = as.Date(sh13$AAPL$date),
                  sapply(sh13, c("","close",USE.NAMES=TRUE)))
names(SH13) <- c("date",names(sh13))
SH13[,-1] <- apply(SH13[,-1], 2, function(x) 100*x/x[1])
SH13m <- melt(SH13, id="date")

# OPTION 2
SH13am <- do.call(rbind,
                mapply(function(z,y){
                     data.frame(
                     date = as.Date(z$date),
                     value = 100*z$close/z$close[1],
                     variable = y)
                }, z = sh13, y = names(sh13), SIMPLIFY = FALSE))

# original plot from GDAR:
ggplot(SH13am, aes(date, y=value, colour=variable,group=variable)) +
geom_line() + xlab("") + ylab("") +
theme(legend.position="bottom") +
theme(legend.title=element_blank())

# facetshade:
# compare to "average" of others:
facetshade(SH13am, aes(x=date, y=value), f = .~variable, bg.all = FALSE) +
geom_smooth(aes(x=date, y=value),method="loess",span = 1/20) +
geom_line(data=SH13am, aes(colour=variable), show_guide=FALSE) +
xlab("") + ylab("")

# compare to all others
facetshade(SH13am,aes(x=date, y=value),
          f = .~variable, bg.all = FALSE,keep.orig = TRUE) +
geom_line(aes(x=date, y=value, group=orig.variable), colour = alpha(1, 0.3)) +
geom_line(data=SH13am, aes(colour=variable), show_guide=FALSE, size = 1.2) +
xlab("") + ylab"

# --- parallel coordinates --- #
sc <- scale(olives[,3:10])

# OPT: order by var
ord <- order(apply(sc,2,sd))
sc <- sc[,ord]

require(scales)
# OPT: align at median
sc <- apply(sc,2,function(z) rescale_mid(z, mid = median(z, na.rm=TRUE)))

require(reshape2)
require(ggplot2)

msc <- melt(sc)
msc$Area <- olives$Area

f1 <- facetshade(msc,aes(x=Var2,y=value,group=Var1),f=~Area, bg.all = FALSE)
f1+geom_line(alpha=0.05)+
geom_line(data=msc,aes(colour=Area),alpha=0.2)+
facet_wrap(f=~Area,nrow=3)

## End(Not run)
## Not run:
# TESTCODE: instead of creating a new object
# a shade layer is added to an existing ggplot
# NOTE: function CHANGES the object!

# highlighting + alpha
pp0 <- ggplot()+geom_point(data = olives,
aes(x = palmitoleic, y = palmitic), colour = 2) +
facet_wrap(~Area, ncol = 3)
extracat:::facettshade2(pp0, alpha = 0.1, colour = 1)

# colours for both, alpha for shade
pp1 <- ggplot()+geom_point(data = olives,
aes(x = palmitoleic, y = oleic, colour = Area)) + facet_grid(~Region)
extracat:::facettshade2(pp1, alpha = 0.1)

# different geom and colour for shade
pp2 <- ggplot()+geom_point(data = olives,
aes(x = palmitoleic, y = oleic, colour = Area)) + facet_grid(~Region)
extracat:::facettshade2(pp2, geom = geom_density2d,
 mapping = aes(colour = NULL), colour = 7)
fluctile

fluctuation diagrams

Description

Create a fluctuation diagram from a multidimensional table.

Usage

```r
fluctile(tab, dir = "b", just = "c", hsplit = FALSE, shape = "r", gap.prop = 0.1, border = NULL, label = TRUE, lab.opt = list(), add = FALSE, maxv = NULL, tile.col = hsv(0.1,0.1,0.1, alpha=0.6), bg.col = ifelse(add,NA,"lightgrey"), tile.border = NA, vp = NULL, ... )
```

Arguments

- **tab**: The table which is to be plotted.
- **dir**: The bar/rectangle direction: "v" and "h" stand for vertical or horizontal bars. "b" stands for "both" and leads to standard fluctuation diagrams with quadratic rectangles. Use "n" for a same-binsize-plot.
- **just**: A shortcut version of the argument used in grid for the anchorpoint of the rectangles: "rb" is equivalent to c("right", "bottom"), "t" is equivalent to "ct" or c("centre", "top") and so on. See examples.
- **hsplit**: A logical for alternating columns and rows or a vector of logics with TRUE for each variable on the x-axis.
- **shape**: Instead of rectangles ("r") it is possible to use circles ("c"), diamonds ("d") or octagons ("o"). The arguments dir and just work for rectangular shapes only.
- **gap.prop**: proportion of the gaps between the rows/columns within each block.
- **border**: The proportion of the space used for the labels.
- **label**: Whether or not to plot labels.
- **lab.opt**: A list with options for the labels. Currently lab.cex and abbrev work.
- **add**: Whether to create a new plot or add it to an existing one.
- **maxv**: The maximum value for the scale. Default is equivalent to maxv = max(x).
tile.col The color of the tiles.
bg.col The background color in each cell.
tile.border The color for the tile border.
vp An optional viewport to plot in. vp = c(i, j) can be used as a shortcut to
go viewport(layout.pos.row = i, layout.pos.col = j)
... dots

Value

The viewport tree behind the graphic.

Note

This was part of the Google Summer of Code 2011.

Author(s)

Alexander Pilhoefer
Department for Computer Oriented Statistics and Data Analysis
University of Augsburg
Germany

See Also

mosaicplot

Examples

M <- arsim(1000, c(12,12), 3)
fluctile(M)

M2 <- optile(M)

# the standard fluctuation diagram with centralized rectangles
fluctile(M2)

# the standard fluctuation diagram with centralized octagons
fluctile(M2, shape = "o")

# another option such as it is used in iplots or MONDRIAN
# is to plot the rectangles in the bottom left corner
fluctile(M2, just = "lb")

# a multiple barchart
fluctile(M2, just = "b", dir = "h")

# or with vertical bars
fluctile(M2, just = "v", dir = "v")

# a same-binsize-plot
fluctile(M2, dir = "n")

require(MASS)
fluctile(xtabs(Freq~Type+Inf1+Cont+Sat, data=housing), dir="h", just="b",
lab.opt=list(lab.cex=1))

A <- arsim(2000, c(6,6,4,4), 3, shuffle = FALSE, noise = 0.05)
fluctile(A)

## Not run:
# airport footprints: Unique Carrier vs. Destination
require(grid)
iata <- c("ATL","BOS","CLT", "DEN", "DFW", "DTW",
"EWR", "IAH", "LAS", "LAX", "MCO", "MSP", "ORD", "PHX", "SFO", "SLC")

mat.layout <- grid.layout(nrow = 4, ncol = 4, widths = 1/4, heights=1/4)
grid.newpage()
vp.mat <- viewport(layout = mat.layout)
pushViewport(vp.mat)
for(i in seq_along(iata)){
ap <- assign(iata[i],read.table(
paste("http://rosuda.org/lehre/SS09-f/datasets/air07s_",
iata[i],".txt",sep=""),sep="\t",quote="",header=T) )

tt <- with(ap, table(UniqueCarrier, Dest))
jj <- ceiling(i/4)
ii <- i - (jj-1)*4
fluctile(optile(tt,iter=100),vp=c(ii,jj),
lab.opt=list(rot=0,lab.cex=0.5),
border=c(0.1,0.02,0.02,0.15),gap.prop=0.2)
pushViewport(viewport(layout.pos.row = ii, layout.pos.col = jj))
grid.text(iata[i],0.5,0.8, gp=gpar(col=2))
}
popViewport()
rm(ap)

## End(Not run)
Description

Gene expression dataset.

Usage

data(GeneEx)

Format

A data frame with 7705 observations (genes) on the following 52 variables (samples).

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLID</td>
<td>clinical ID</td>
</tr>
<tr>
<td>NAME</td>
<td>a numeric vector</td>
</tr>
<tr>
<td>shee177</td>
<td>a numeric vector</td>
</tr>
<tr>
<td>shfa047</td>
<td>a numeric vector</td>
</tr>
<tr>
<td>shfs151</td>
<td>a numeric vector</td>
</tr>
<tr>
<td>shfa044</td>
<td>a numeric vector</td>
</tr>
<tr>
<td>shee146</td>
<td>a numeric vector</td>
</tr>
<tr>
<td>shee129</td>
<td>a numeric vector</td>
</tr>
<tr>
<td>shee118</td>
<td>a numeric vector</td>
</tr>
<tr>
<td>shee109</td>
<td>a numeric vector</td>
</tr>
<tr>
<td>shee100</td>
<td>a numeric vector</td>
</tr>
<tr>
<td>shet058</td>
<td>a numeric vector</td>
</tr>
<tr>
<td>shet057</td>
<td>a numeric vector</td>
</tr>
<tr>
<td>shet033</td>
<td>a numeric vector</td>
</tr>
<tr>
<td>shee045</td>
<td>a numeric vector</td>
</tr>
<tr>
<td>shco045</td>
<td>a numeric vector</td>
</tr>
<tr>
<td>shco044</td>
<td>a numeric vector</td>
</tr>
<tr>
<td>shco039</td>
<td>a numeric vector</td>
</tr>
<tr>
<td>shco038</td>
<td>a numeric vector</td>
</tr>
<tr>
<td>shco031</td>
<td>a numeric vector</td>
</tr>
<tr>
<td>shco030</td>
<td>a numeric vector</td>
</tr>
<tr>
<td>shco029</td>
<td>a numeric vector</td>
</tr>
<tr>
<td>shfe113</td>
<td>a numeric vector</td>
</tr>
<tr>
<td>shfe085</td>
<td>a numeric vector</td>
</tr>
<tr>
<td>shfe084</td>
<td>a numeric vector</td>
</tr>
<tr>
<td>shfe081</td>
<td>a numeric vector</td>
</tr>
<tr>
<td>shfe052</td>
<td>a numeric vector</td>
</tr>
<tr>
<td>shcz090</td>
<td>a numeric vector</td>
</tr>
<tr>
<td>shcz089</td>
<td>a numeric vector</td>
</tr>
</tbody>
</table>
shcz087 a numeric vector
shcz086 a numeric vector
shcz085 a numeric vector
shcz064 a numeric vector
shcz061 a numeric vector
shdj104 a numeric vector
shdj102 a numeric vector
shdj094 a numeric vector
shdj089 a numeric vector
shdj067 a numeric vector
shfs240 a numeric vector
shfs229 a numeric vector
shfs227 a numeric vector
shfs226 a numeric vector
shfa109 a numeric vector
shfa107 a numeric vector
shee216 a numeric vector
shet172 a numeric vector
shee198 a numeric vector
shfa067 a numeric vector
shfa066 a numeric vector
shee188 a numeric vector
shfs159 a numeric vector

Details
See source.

Source
http://seurat.r-forge.r-project.org/

Examples
data(GeneEx)
## maybe str(GeneEx) ; plot(GeneEx) ...
getbw

Active binning

Description

Computes binwidth and breakpoints for a numeric or integer variable. The binwidth is a multiple of the minimal non-zero distance between two neighboring observations. The binwidth is chosen such that the number of active bins is as close as possible to a desired number $k$. An active bin is a bin which contains at least $\text{min}_n$ observations.

Usage

getbw(x, k = NULL, min_n = NULL, warn = FALSE)

Arguments

- **x**: A numeric variable.
- **k**: The desired number of active bins. A bin is active if it contains at least $\text{min}_n$ observations. The default is $k \leftarrow 1 + 2 \cdot \text{ceiling}(\log(N)/\log(2))$.
- **min_n**: The minimum number of observations necessary for a bin to count as an active bin. Defaults to $\text{min}_n = \max(\log(N/10)/\log(10), 1)$.
- **warn**: I don’t want to know about problems.

Value

A vector of breakpoints with attributes "bw", "k" and "outlier". The first one is the binwidth, the second one is the number of active bins which is as close as possible to the specified parameter $k$. "outlier" is a logical vector indicating which bins are not active.

Note

Experimental.

Author(s)

Alexander Pilhoefer

See Also

nclass.scott, cutbw, ahist
getbw

Examples

```r
require(scales)

hist(x <- rexp(200,1/10), breaks = gb <- getbw(x,24,min_n = 5, warn=TRUE),
    col = alpha(attr(gb, "outlier") + 1,.3))

hist(x <- rexp(2000,1/10), breaks = gb <- getbw(x,24,min_n = 5,warn=TRUE),
    col = alpha(attr(gb, "outlier") + 1,.3))

x <- rlnorm(1000,log(10),log(4))
x <- c(x, rnorm(500,400,30))

hist(x , breaks = gb <- getbw(x,24,min_n = 5,warn=TRUE),
    col = alpha(attr(gb, "outlier") + 1,.3))

x <- rlnorm(1000,log(10),log(4))
x <- c(x, rnorm(500,800,30))

hist(x , breaks = gb <- getbw(x,24,min_n = 5,warn=TRUE),
    col = alpha(attr(gb, "outlier") + 1,.3))

## Not run:

bws1 <- replicate(1000,
    x <- rexp(200,1/10)
    gb <- getbw(x,20)
    attr(gb, "bw")
)

hist(bws1,breaks=getbw(bws1,30))

bws2 <- replicate(1000,
    x <- rnorm(200)
    x <- x/rnorm(200)
    gb <- getbw(x,20)
    attr(gb, "bw")
)

hist(bws2,breaks=getbw(bws2,30))

mov <- read.table("http://www.rosuda.org/lehre/WS1213-f/MovieLens.txt",
    header=T, sep="\t")

require(extracat)
```
getcolors

Create a color vector

Description

Creates a color vector using different palettes, e.g. from the colorspace package.

Usage

getcolors(N, palette, col.opt = list(), revert = FALSE)

Arguments

N Number of colors.

palette Palette shortcut:

- "rgb", "hsv" RGB rainbow colors. See rainbow.
- "hcl" HCL rainbow colors. See rainbow_hcl.
- "s", "seq", "sqt", "sqn", "sequential" See sequential_hcl
- "d", "div", "diverging", "diverge" See diverge_hcl
- "h", "heat", "heatcolors" See heat_hcl
- "t", "ter", "terrain" See terrain_hcl
Description

Basically this is an auxiliary function used by `heattile`: It extracts the cluster indices from a biclust object and optimizes the order of the rows and columns in the data matrix with respect to these clusters. Uses the Measure of Effectiveness as an optimization criterion. See `ME` and `optME`.

Usage

```r
getIs(biclust, dim, nstart = 20, solver = "nn", adjust.dist = TRUE)
```

Arguments

- **biclust**: The biclust object.
- **dim**: The dimension of the matrix.
- **nstart**: Number of starting points for the TSP solver in `optME`.
- **solver**: The TSP solver to use with `optME`: See `solve_TSP`.
- **adjust.dist**: If TRUE the ME values used as a distance matrix for the TSP are slightly adjusted by adding a the hamming distance divided by a constant. This keeps identical cases together and also preserves the orders within such groups.
Details

The algorithm first computes an indicator matrix for each cluster and then combines these matrices
to a 3D table. Then for the first and the second dimension the category orders are optimized with
respect to ME. The optimization is done via optME which uses a TSP solver.

Value

The cluster indices with respect to the optimized row and column orders in form of a list. The
optimized orders for the data matrix are attached as an attribute attr(x, "orders").

Author(s)

Alexander Pilhoefer

See Also

heattile, getIs2

Description

Basically this is an auxiliary function used by heattile: It extracts the cluster indices from a biclust
object and optimizes the order of the rows and columns in the data matrix with respect to these
clusters. Uses the Measure of Effectiveness as an optimization criterion. See ME and optME.

Usage

getIs2(bic, dim, nstart = 20, solver = "nn", cpr = FALSE,
cpc = TRUE, adjust.dist = FALSE)

Arguments

bic The biclust object.
dim The dimension of the matrix.
nstart Number of starting points for the TSP solver in optME.
solver The TSP solver to use with optME: See solve_TSP.
cpr Whether or not to combine identical rows.
cpc Whether or not to combine identical columns.
adjust.dist If TRUE the ME values used as a distance matrix for the TSP are slightly adjusted
by adding a the hamming distance divided by a constant. This keeps identical
cases together (which is only necessary if cpr = FALSE or cpc = FALSE) and
also preserves the orders within such groups.
Details

The algorithm first computes an indicator matrix for each cluster and then combines these matrices to a 3D table. Then for the first and the second dimension the category orders are optimized with respect to ME. The optimization is done via optME which uses a TSP solver.

The difference to getIs is that rows and columns which are identical with respect to the biclusters are combined before the optimization. This keeps identical categories together and also speeds up the algorithm considerably (depending on the TSP solver).

The TSP solver solver = "nearest_insertion" for instance is inefficient in this case since it has to add (identical) cases one by one.

Value

The cluster indices with respect to the optimized row and column orders in form of a list. The optimized orders for the data matrix are attached as an attribute attr(x, "orders").

Author(s)

Alexander Pilhoefer

See Also

heattile, getIs2

getpath

Path extraction from quickfechner objects

Description

Uses the path.matrix to obtain the shortest paths of the quickfechner object.

Usage

getpath(fm, pm = NULL, from = 1, to = nrow(fm))

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fm</td>
<td>A Fechner matrix computed via quickfechner</td>
</tr>
<tr>
<td>pm</td>
<td>The path.matrix. Only necessary if the corresponding attribute in fm is missing.</td>
</tr>
<tr>
<td>from</td>
<td>Start index.</td>
</tr>
<tr>
<td>to</td>
<td>Final index.</td>
</tr>
</tbody>
</table>

Details

The path.matrix is defined as follows: The entry of the i-th row and j-th column is the index of the last node visited before j on the shortest path from i to j.
Value

A vector of indices defining the shortest path from i to j in the original matrix passed to the quick-fechner function.

Author(s)

Alexander Pilhoefer

Examples

# not a distance matrix, but a similarity matrix in some sense
cx <- 1-abs(cor(olives[-c(1,2,11)]))

cx2 <- quickfechner(cx)
getpath(cx2,from=1,to=5)

gsac | GSAC

Description

The generalized sort-and-cut algorithm. Reordering via optile and partitioning via cfluctile are iteratively combined to a clustering algorithm.

Usage

gsac(x, nc = Inf, maxiter = 40, zero = TRUE, r0 = 0,
force.cs = FALSE, force.rs = FALSE, resort = "complete",
method = "Kendall", tau0 = 0.5, stack = "max", clean = TRUE,
clean.Is = TRUE, cutoff = -20, ... )

Arguments

x The data matrix.
nc The desired maximum number of clusters. Useful to restrict the algorithm further.
maxiter Maximum number of iterations. Depending on the partitioning method/threshold and the size of the matrix the number of steps can sometimes become very large.
zero Each partition identifies a sparse part of the table. If zero = TRUE this part is set to 0 or r0 times the expectation from the marginals.
r0 A parameter controlling how sparse areas are handled. Usually left at zero, which means that sparse areas are zeroed.
force.cs Logical. If TRUE clusters may not share columns.
force.rs Logical. If TRUE clusters may not share rows.
resort

The reordering variation: "none" for no subtable reordering, "c" for restricted common reordering (the subtables share the orders), "s" for independent/unrestricted reordering.

method

Method used for the partitioning via cfluctile.

tau0

Threshold used for the partitioning via cfluctile.

stack

The rule which element (cluster) from the list to use next. "last" is the FILO and "first" uses FIFO. "max" and "min" mean that the largest / the smallest elements will be handled first.

clean

Whether or not to clean the results using setcover.

clean.Is

Whether or not to check the current list of non-finished clusters for redundancies.

cutoff

Clusters are pruned by removing rows and columns with an average residual below this value.

... Further arguments passed to optile.

Details

The clusters are returned as a 2D list. The first element lists the clusters by their row indices vectors, the second element lists the corresponding column indices. This can be used with getIs and heattile.

Value

A 2D list: row and column indices of the clusters.

Author(s)

Alexander Pilhoefer

See Also

sortandcut, cfluctile

Examples

```r
## Not run:
s <- sample(1:nrow(plants),500)
M <- t(as.matrix(plants[s,,-1]));

gs <- gsac(M, fun="IBCC", foreign=".Call")

heattile(M, Is = getIs2(gs, dim(M)), fluct = TRUE, hm.palette = 1)

## End(Not run)

# simulated example:
A <- arsim(3000,c(8,5),1)
B <- arsim(2000,c(7,6),1)
C <- arsim(4000,c(9,9),1)
```
M <- matrix(0,16,16)
M[1:8,1:5] <- A
M[4:10,6:11] <- B
M[8:16,8:16] <- C

M <- as.table(optile(M, iter=20))
t0 <- 0.6

# no subtable reordering
test1 <- gsac(M,resort="none", method= "BCI", tau0=t0)
require(scales)
heattile(M,ls=test1,hm.palette=alpha(1,0.8),shape="r",
fluct = TRUE, label = c(TRUE,TRUE),bg.col=NA, lab.opt = list(rot=c(0,90)))

## Not run:
# unrestricted subtable reordering
test2 <- gsac(M,resort="s", method= "BCI", tau0=t0)
#common reordering
test3 <- gsac(M,resort="c", method= "BCI", tau0=t0)
# clusters do not share rows, columns, both
test4 <- gsac(M,resort="s",force.cs=TRUE,method = "BCI", tau0=t0)
test5 <- gsac(M,resort="s",force.rs=TRUE,method = "BCI", tau0=t0)
test6 <- gsac(M,resort="s",force.rs=TRUE,force.cs=TRUE, tau0=t0)
## End(Not run)

## Not run:
heattile(M,Is=test2,hm.palette=alpha(1,0.8),shape="r",
fluct = TRUE, label = c(TRUE,TRUE),bg.col=NA, lab.opt = list(rot=c(0,90)))
heattile(M,Is=test3,hm.palette=alpha(1,0.8),shape="r",
fluct = TRUE, label = c(TRUE,TRUE),bg.col=NA, lab.opt = list(rot=c(0,90)))
heattile(M,Is=test4,hm.palette=alpha(1,0.8),shape="r",
fluct = TRUE, label = c(TRUE,TRUE),bg.col=NA, lab.opt = list(rot=c(0,90)))
heattile(M,Is=test5,hm.palette=alpha(1,0.8),shape="r",
fluct = TRUE, label = c(TRUE,TRUE),bg.col=NA, lab.opt = list(rot=c(0,90)))
heattile(M,Is=test6,hm.palette=alpha(1,0.8),shape="r",
fluct = TRUE, label = c(TRUE,TRUE),bg.col=NA, lab.opt = list(rot=c(0,90)))
## End(Not run)
Description

Draws a heatmap using fluctile as the workhorse and offers the possibility to add rectangles which visualize the biclusters.

Usage

heattile(x, biclust = NULL, Is = NULL, shape = "r", fluct = FALSE, gap.prop = 0, border = c(0.05, 0.03, 0.03, 0.05), label = c(TRUE,FALSE), lab.opt = list(abbrev = 24, lab.cex = 1, rot = 0), bg.col = "lightgrey", sym = FALSE, breaks = 20+10*sym, clust.col = NULL, clust.palette = "rgb", hm.palette = "div", clust.col.opt = list(), hm.col.opt = list(revert = TRUE))

Arguments

x
A two-was data matrix.
biclust
A biclustering object. The matrix is displayed in its original order.
Is
Instead of biclust one can define the indices of the clusters as a list where each element represents a cluster and is itself a list of length 2 containing the row indices and the column indices respectively. getIs or getIs2 return such lists and the row and column orders are changed with respect to the clusters.
shape
Shape of the tiles, see fluctile.
fluct
Plots polygons whose sizes are proportional to their corresponding values, see fluctile. If FALSE (default) a colored same-binsize plot is produced.
gap.prop
gaps between the tiles, see fluctile.
border
plot margins, see fluctile.
label
Whether or not to draw labels, see fluctile.
lab.opt
Label options, see fluctile.
bg.col
A background color, see fluctile.
sym
Whether or not the colors should be on a symmetric scale around zero.
breaks
The matrix entries are cut into intervals via fluctile. see fluctile.
clust.col
A color vector for the cluster rectangles.
clust.palette
If no colors are specified a palette is used to obtain them: Usually a quantitative palette is a reasonable choice, e.g. "rgb" for rainbow and "hcl" for rainbow_hcl. "seq" and "div" stand for sequential_hcl and diverge_hcl.
hm.palette
The color vector for the heatmap or a color palette. Usually "seq" and "div" which stand for sequential_hcl and diverge_hcl respectively will make sense. "rgb" for rainbow and "hcl" for rainbow_hcl are useful if the matrix entries are categorical. "terrain" and "heat" are also available.
clust.col.opt
Options for the cluster color palette. See col.opt for rmb.
hm.col.opt
Options for the heatmap color palette. See col.opt for rmb.
Value

TRUE

Author(s)

Alexander Pilhoefer

See Also

fluctile

Examples

```r
## Not run:

ss <- sample(1:nrow(plants), 500)
M <- t(as.matrix(plants[ ss, -1]))
M <- optME(M)
heattile(M, hm.palette = "seq")

require(biclust)

GE <- t(na.omit(GeneEx[,3:52]))

# draw a sample of 1000 genes
ss <- sample(1:ncol(GE),1000)

EY <- GE[,ss]
SEY <- scale(EY)

# compute sensible initial row and column orders:
require(seriation)
s1 <- seriate(dist(SEY),method="GW")
s2 <- seriate(dist(t(SEY)),method="GW")

o1 <- get_order(s1,1)
o2 <- get_order(s2,1)

SEY <- SEY[o1,o2]

# A plaid model with row effects
b1 <- biclust(SEY,method=BCPlaid(),row.release=0.4,
             col.release=0.4, fit.model = y ~ m + a )

# index sets from b1
Is2 <- getIs(b1,dim(SEY), nstart = 1)
```
hexpie

Description

This function bins two continuous variables into a hexagonal grid and represents a third variable (which is usually a factor) via piecharts or nested hexagons within the bins. The main idea is to avoid overplotting and unfortunate effects that emerge from mixing up colors, e.g. with alpha-blending.

Usage

```r
hexpie(x, y = NULL, z = NULL, n = 24, shape = "hex", p.rule = "radial",
       decr.by.rank = NULL, freq.trans = I, alpha.freq = FALSE, col = "hcl",
       col.opt = list(), show.hex = TRUE, random = NULL, xlim = range(x),
       ylim = range(y), label.opt = list(), vp = NULL)
```

Arguments

- `x`: The variable for the horizontal axis. Should be integer or numeric.
- `y`: The variable for the vertical axis. Should be integer or numeric.
- `z`: The target variable for the colors which is handled as a factor.
- `n`: The number of bins into which x is divided. See `hexbin`.
- `shape`: There are two possibilities: "hex", "hexagonal", and "h" lead to hexagonal representations and "pie", "piechart", "circular" and "c" lead to circular representations.
- `p.rule`: This controls the rules for the representation of the relative frequencies of the target categories. For shape = "hex" this should be one of "rad", "radius", "radial" meaning that the probabilities are represented by the radii. For shape = "circular" it is also possible to create piecharts via "angular", "angles" or "ang".
- `decr.by.rank`: Whether or not to sort the categories within each hexagon individually by their frequencies in decreasing order. Defaults to NULL for no reordering but may be either TRUE (decreasing order) or FALSE (increasing order).
- `freq.trans`: A function which is used to rescale the total counts of the cells. `sqrt` is a common choice.
The frequencies may additionally be reflected in terms of the alpha values of the colors.

The choice of a color palette. See `rmb` for further explanations.

Additional color options to replace the defaults. See `rmb` for further explanations.

Whether or not to draw the hexagons. Setting `col.opt = list(line.col.hex = NA)` leaves the lines out and draws the background only.

If this is not NULL in each bin a random sample of \( n = \text{random} \) observations will be drawn (with replacement) from the corresponding data points. The resulting frequencies are then used to draw the piechart or hexagon. The main idea is to use `random = 1` with larger numbers of bins such as \( n = 120 \) and `show.hex = FALSE`.

A vector of length 2 defining the x-limits e.g. computed via `innerval`.

A vector of length 2 defining the y-limits e.g. computed via `innerval`.

Additional labeling options to replace the defaults. Not yet implemented.

A viewport to plot in, e.g. for conditional plots.

Value

`invisible(TRUE)`

Author(s)

Alexander Pilhoefer

See Also

`stat_binhex`, `hexbin`

Examples

data(olives)
x <- olives$oleic
y <- olives$linoleic
z <- olives$Region

# the default
hexpie(x,y,z)

## Not run:
# zooming in (transformation of the total number of obs in each bin)
hexpie(x,y,z, freq.trans=sqrt)

# circular shapes
hexpie(x,y,z, freq.trans=sqrt, shape="pie")

# classical piecharts
hexpie(x,y,z, freq.trans=sqrt, shape="pie", p.rule="angles")
# the total numbers of obs are reflected via alpha-blending,
# the grid is not shown and RGB colors are used
hexpie(x,y,z, freq.trans=sqrt, shape="hex", p.rule ="radial",
alpha.freq=TRUE, col ="rgb",show.hex=F)

hexpie(x,y,z, freq.trans=NULL, shape="hex", p.rule ="radial",
alpha.freq=TRUE, col ="rgb",show.hex=T)

require(ggplot2)
data(diamonds)
x2 <- diamonds$carat
y2 <- diamonds$price
z2 <- diamonds$color

# a standard plot with colors via ggplot2
qplot(x2,y2,colour=z2)

# the hexpie version
hexpie(x2,y2,z2,n=36)

# due to the few bins with the majority of observations
# it is sensible to zoom in
hexpie(x2,y2,z2,n=36,freq.trans=function(s) log(1+s))

# the same, but this time the central color is the most frequent one
hexpie(x2,y2,z2,n=36,freq.trans=function(s) log(1+s), decr.by.rank = TRUE)

# this way the difference is more obvious
# (although the color palette is better suited for ordinal target variables)

mat.layout <- grid.layout(nrow = 1 , ncol = 2 , widths = c(1/2,1/2), heights=1)
grid.newpage()
vp.mat <- viewport(layout = mat.layout)
pushViewport(vp.mat)

vp1 <- viewport(layout.pos.row = 1, layout.pos.col = 1)
pushViewport(vp1)

hexpie(x2,y2,z2,n=18,freq.trans=NULL,
decr.by.rank=NULL,col="div", vp = vp1)

vp2 <- viewport(layout.pos.row = 1, layout.pos.col = 2)
pushViewport(vp2)

hexpie(x2,y2,z2,n=18,freq.trans=NULL,
decr.by.rank=T,col="div", vp = vp1)
popViewport()

# random samples from the data (within bins) with many bins
# (takes some time)
require(scales)
grid.newpage()
idat

**indicator dataframe**

**Description**

Converts all factor variables in a dataframe to a set of binary variables.

**Usage**

```r
idat(x, allcat = FALSE, keep = "Freq")
```

**Arguments**

- `x` : dataframe
- `allcat` : Whether or not to keep all categories or leave the last one out.
- `keep` : Variables which are kept unchanged such as a frequency variable.

**Value**

dataframe

**See Also**

Burt, imat

**Examples**

```r
require(MASS)
idat(housing)
```
**imat**

Indicator variables

**Description**

converts a single categorical variable into indicator variables

**Usage**

`imat(x, allcat = TRUE)`

**Arguments**

- `x` A factor variable.
- `allcat` Whether or not to keep all categories or leave the last one out.

**Value**

matrix

**Examples**

```r
require(MASS)
imat(housing$Type)
```

---

**innerval**

Interval boundaries

**Description**

This function computes the boundaries of an interval which is symmetric around the median and includes a given percentage of the data. If that's impossible due to ties the interval is chosen to minimize the squared difference between the desired percentage and the actual percentage of the observations included.

**Usage**

`innerval(x, p = 0.95, data.points = TRUE)`

**Arguments**

- `x` A data vector.
- `p` The percentage of observations inside the interval.
- `data.points` Whether to return the most extreme data points within the interval or the interval boundaries.
itab

Value
A vector with the lower and upper boundaries of the interval.

Author(s)
Alexander Pilhoefer

See Also
quantile

Examples
x1 <- rnorm(200)
innerval(x1)
quantile(x1, c(0.025, 0.975))

x2 <- rexp(200)
innerval(x2, data.points = FALSE)
innerval(x2)
quantile(x2, c(0.025, 0.975))

---

itab Independence Table

Description
Computes the independence table for a data table with non-negative entries. The entries of the independence table are defined by the multiplication of all corresponding marginal relative frequencies.

Usage
itab(x)

Arguments
x A data table of any dimension but with non-negative entries.

Value
A data table of the same dimension as the input table.

Author(s)
Alexander Pilhoefer
The Joint Bertin Classification Index

**Description**

Computes the Joint Bertin Classification Criterion which uses joint independence as a reference for normalization.

**Usage**

\[ JBCI(x, r = 1) \]

**Arguments**

- **x** The 3D table with non-negative entries.
- **r** The index of the variable which is tested for joint independence of the other two.

**Details**

The BCI of a 3D table but instead of the total independence case the joint independence case is used for normalization. With an optimal reordering we have \( JBCI(x) \geq BCI(x) \).

**Value**

Numeric value in [0,1].

**Author(s)**

Alexander Pilhoefer

**See Also**

BCI, CBCI, WBCI
Examples

```r
## Not run:
A <- optile( arsim(144*5*20,c(12,12),6,0.1) , iter = 1000)

p1 <- 0.1 + runif(5)
p1 <- p1/sum(p1)

A2 <- apply(A,1:2,function(z) rmultinom(1,z,p1))
A2 <- optile(A2, iter = 1000, return.type="table")

BCI(A)
BCI(A2)

DA2 <- subtable(A2,1:3)
names(DA2) <- c("X","Y","Z","Freq")

rmb(~Y+Z+X, data=DA2)

JBCI(A2,3)
## End(Not run)
```

---

**kendalls**

*Kendalls Tau for a matrix*

### Description

Computes Kendalls Tau for a two-way table or matrix.

### Usage

```r
kendalls(x)
```

### Arguments

- `x` A two-way table or matrix.

### Details

Kendalls tau is a rank-correlation coefficient.

### Value

numeric between -1 and +1.

### Author(s)

Alexander Pilhoefer
**Examples**

```r
M <- arsim(300, c(8, 8), 3)
kendalls(M)
kendalls(optile(M))
```

**Description**

Computes the measure of effectiveness for a table, a matrix or an array.

**Usage**

```r
ME(x)
```

**Arguments**

- `x` A matrix, table or array.

**Value**

The ME value.

**See Also**

- `optME`

**Examples**

```r
a <- arsim(2000, c(8, 9, 10), 3, 0.2)
ME(a)
a2 <- optME(a)
ME(a2)
```
olives  

Description

Various fatty acid measurements.

Usage

data(olives)

Format

A data frame with 572 observations on the following 11 variables.

- **area**: a factor with levels Calabria Coast-Sardinia East-Liguria Inland-Sardinia North-Apulia Sicily South-Apulia Umbria West-Liguria
- **region**: a factor with levels North Sardinia South
- **palmitic**: a numeric vector
- **palmitoleic**: a numeric vector
- **stearic**: a numeric vector
- **oleic**: a numeric vector
- **linoleic**: a numeric vector
- **linolenic**: a numeric vector
- **arachidic**: a numeric vector
- **eicosenoic**: a numeric vector
- **testntraining**: a factor with levels Test Training

Examples

data(olives)

---

optile  

Reordering Categorical Data

Description

This function will take a categorical data object (data.frame, table, ftable, matrix, array) and optimize its category orders. Most of the implemented techniques aim for a (pseudo-) diagonalization of the data matrix or table. This improves graphical representations (e.g. by minimizing crossings in scpcp plots) and can also be useful to compute clusters (e.g. via cfluctile).

The function offers an interface which will by default return the same type of object that has been passed to the function such that it is possible to write myplot( optile(x) ) for an optimized version of myplot(x). It is possible to use custom reordering functions (as long as they meet the requirements, see details).
Usage

optile(x, fun = "BCC", foreign = NULL,
       args = list(), perm.cat = TRUE, method = NULL, iter = 1,
       freqvar = NULL, return.data = TRUE,
       return.type = "data.frame", vs = 0, tree = NULL, sym = FALSE, ...)

## S3 method for class 'list'
optile(x, fun = "BCC", foreign = NULL,
       args = list(), perm.cat = TRUE, method = NULL, iter = 1,
       freqvar = NULL, return.data = TRUE,
       return.type = "table", vs = 0, tree = NULL,
       sym = FALSE, k = NULL, h = NULL, ...)

Arguments

x The categorical data of one of the following classes:
    data.frame, table, ftable, matrix, array

fun The optimization function. Currently available are:
    BCC, WBCC, CA, csvd, rmca, symtile, barysort and IBCC. For more
    information see details.

foreign Where to find the optimization function fun. NULL for an R
    function or for instance "Call" and .C for an external function.
    E.g. barysort needs foreign = "Call".

args further arguments which will be passed to fun.

perm.cat A logical vector indicating which variables are reordered and which will
    remain untouched. For example perm.cat = c(FALSE, TRUE) means that only the
    second variable is reordered. Has no effect if fun = "casort".

method Either NULL, "joint" or "stepwise". method = NULL means that the whole
    data table is passed to fun. method = "joint" uses the Burt matrix instead of
    the whole table which only reflects two-way associations not unlike a covariance
    matrix. method = "stepwise" will repeatedly call fun for 2, 3, 4, and so on
    variables.

iter Some optimizations depend on the initial category orders (e.g. "BCC" and "IBCC").
    If iter > 1 the optimization is repeated for iter random initial category orders
    and the best result is returned. In this case fun must return comparable values.

freqvar The name of the frequency variable, if any.

return.data Whether to return the data or just the new orders.

return.type The class of the object which will be returned. Defaults to the input type.

vs An optional version number. "WBCC" is currently equivalent to "BCC" and vs = 1

tree A list whose entries are either tree objects (e.g. from hclust) or the string "hc".
    If the i-th entry is a tree object, the i-th variable is the result from cutting the tree
    into dim(x)[i] clusters via subtree. "hc" will compute a hierarchical clustering
    for the rows and columns with arguments specified in args.

sym If fun is BCC or IBCC it is possible to run a symmetric version of the algorithm by setting sym = TRUE.
k A vector of integers specifying the numbers of clusters into which the tree objects shall be cut. See subtree.

h Instead of a number of clusters \( k \) the height at which the dendrogram shall be cut can be specified. See subtree.

Details

The optile interface makes it possible to resort the categories in different representations of categorical data.

The most important points to know are

The function by default returns the same type of object as was passed in the function call.
It is possible to specify custom optimization functions via \( \text{fun} \) and \( \text{foreign} \).
The function is able to handle tree objects which specify a hierarchical tree graph on the categories.
The function can pass either multidimensional tables, the corresponding Burt matrix (\texttt{method = "joint"}) or a hierarchical series of tables (\texttt{method = "stepwise"}) to the optimization functions.

How to add a custom optimization function:

It is possible to use custom functions for the optile interface as long as they meet the following requirements:

The function should have the form
\[
\text{fun}( \text{data}, \text{dims}, \text{perm.cat}, \ldots ) \text{ or }
\text{foreign}( \text{"fun"}, \text{data}, \text{dims}, \text{perm.cat}, \ldots )
\]

where \( \text{fun} \) is the name of the function and \( \text{foreign} \) is \"\text{Call}\", \"\text{C}\", \ldots

The function returns a vector of the new category orders (minus 1) and the resulting criterion, e.g.
\[
c( 0, 2, 4, 1, 3, 4, 3, 2, 1, 0, 5, 6, 0.7612 )
\]

dims is a vector with the number of categories for each variable and perm.cat is a 0/1 vector which indicates whether or not to change the category order of a variable.

There are three possible types for the data argument of \( \text{fun} \) which can be set via \texttt{method}:

The argument \texttt{method} can be one of \texttt{NULL}, \"stepwise\" or \"joint\". The default \texttt{method = NULL} indicates that \( \text{fun} \) accepts a multidimensional table as for instance can be produced via \texttt{xtabs}.

If \texttt{method = "joint"} a Burt matrix is computed and passed to \( \text{fun} \) (c.f. \texttt{Burt}). For instance \"fun=casort\" uses this data representation.

\texttt{method = "stepwise" or method = "sw"} passes \texttt{fun}, \texttt{data}, \texttt{foreign} as well as any \texttt{args} to a function called \texttt{steptile} which initially builds a 2-way table of the first pair of variables, passes it to \texttt{fun} and stores the computed category orders. Afterwards the other variables are added one by one. i.e. in a step for the k-th variable the function passes a k-way table to \texttt{fun} and a new category order for this variable is computed given the (already fixed) category orders of the variables 1 to k-1. This version is well suited for hierarchical visualizations like classical mosaicplots. A slightly different implementation which is not embedded in the optile framework but uses optile as its workhorse is \texttt{steptile}.

CURRENTLY AVAILABLE REORDERING FUNCTIONS:
"BCC" and "WBCC": minimize the Bertin Classification Criterion and the Weighted Bertin Classification Criterion. BCC is the number of observation pairs which are not fully concordant among all relevant observation pairs (pairs which differ in all variables). A pair of observations a and b is fully concordant if all entries in a are smaller than those in b or vice versa. Full concordance results in a so-called pseudo-diagonal. WBCC uses the Hamming distance between the observations as weights for the contradictions to such a diagonal form and also takes pairs within the same row or column into account.

"casort": computes a correspondence analysis (SVD) and sorts by the first coordinate vector of each dimension. For more than two dimensions Multiple CA based on the Burt matrix is used.

"rmca": Adopts the idea of CA for k > 2 dimensions without dropping information: For each dimension d = 1..k with categories d1...dr compute the scaled average k-1 dimensional profile sdd and perform an SVD of (sd1...sd-r)-sdd. Like in correspondence analysis the first coordinate vector is used for the reordering.

"csvd": For each variable d in 1..k (iteratively) compute the cumulative sums over the multidimensional table for each variable except d. Transform this multidimensional table to an r x s matrix with r being the number of categories of variable d and s being the product of these numbers for all other variables. Resort the categories of variable d by the first coordinate vector of an SVD of that matrix. Repeat this procedure for all variables in turn until a stopping criterion is met. Idea: for any variable h != d we have h1 < h2 < ... < hx due to the cumulative sums. Hence the current order of the categories will (tend to) be the same as in the coordinates of the SVD which means that the svd computes coordinates for variable d with respect to the current category orders of the other variables. The algorithm uses casort for an initial solution to start from.

"distcor": Two-way tables or matrices can also be optimized by means of the distance correlation. See wdcor.

"IBCC": Iteratively sorts the categories of one variable at a time. Therefore it computes the average over the remaining dimensions and scales the profiles of each category as well as the average profile. It then computes the classification criterion between each category profile and the average profile which results in one value per category. The categories are then sorted by this criterion. The procedure is very quick and yields good results in most cases. It strongly depends on the initial category orders as do the BCC or WBCC algorithms. This function is written in C which means that foreign = ".Call" must be set. Alternatively it can be used to presort the data via the shortcut presort = TRUE but this is deprecated and not recommended.

"barysort": Uses the barycenter heuristic to minimize the number of crossings heuristically. The heuristic is fast and yields good results but only works for two dimensions. For multiple dimensions use either "IBCC" or steptile. In optile the barysort is implemented in C and therefore requires foreign = ".Call").

Value

The function returns the reordered data. The return type is by default the same as the input type but can be redefined via return.type.

Note

Some parts of the code have been developed for the Google Summer of Code 2011.
Examples

# simple simulated example
A <- arsim(2000, c(11,13),3,0.3)

fluctile(A)
fluctile(optile(A))
fluctile(optile(A, iter = 100))
fluctile(optile(A, fun = "CA"))
fluctile(optile(A, fun = "barysort", foreign = ".Call"))

# simulated mv example
A2 <- arsim(20000, c(6,7,8,9),3,0.1)

scpcp(A2,sel="data[,1]")
scpcp(A3 <- optile(A2,iter=20),sel="data[,1]")

dev.new()
fluctile(A3)

## Not run:  
----------------- EXAMPLE I -----------------  
# ------ Cluster results from the Current Population Survey ------ #
data(CPScluster)
cpsX = subtable(CPScluster,c(5, 26, 34, 38, 39), allfactor=TRUE)

# joint and stepwise optimization of BCC
ss <- optile(cpsX,presort=TRUE, return.data=TRUE, method="joint")
ss2 <- optile(cpsX,presort=TRUE, return.data=TRUE, method="sw")

# original cpcp plot
cpcp(cpsX)

# cpcp for joint algorithm
cpcp(ss)

# cpcp and fluctuation for the stepwise algorithm
# (should be good for pcp plots and hierarchical plots)
fluctile(xtabs(Freq~.,data=ss2[,4]))
cpcp(ss2)

# The multivariate algorithm
ss3 <- optile(cpsX,presort=TRUE, return.data=TRUE, method=NULL)
cpcp(ss3)
# cpcp for casort algorithm
ssca <- optile(cpsX,presort=FALSE, fun = "casort", return.data=TRUE, method="joint")
cpcp(ssca)

# cpcp for rmca algorithm results, works better for the dmc data
ssR <- optile(cpsX,presort=FALSE, fun = "rmca", return.data=TRUE, method=NULL)
cpcp(ssR)

# cpcp for csvd algorithm
ssC <- optile(cpsX,presort=FALSE, fun = "csvd", return.data=TRUE, method=NULL)
fluctile(xtabs(Freq~.,data=ssC[,,-4]))
cpcp(ssC)

# cpcp for presort algorithm with 20 iterations
ssP <- optile(cpsX,presort=FALSE, fun = "PBCC", return.data=TRUE, method=NULL, foreign = ".Call",iter=20)
cpcp(ssP)

library(MASS)
data(wine)
swine <- scale(wine[,1:13])
kmd <- data.frame(wine$class, replicate(9, kmeans(swine, centers = 6)$cluster) )
kmd <- subtable(kmd, 1:10, allfactor = TRUE)
cpcp(kmd)

# there is a good joint order and hence the joint result is better than the stepwise
kmd2 <- optile(kmd, method = "sw")
kmd3 <- optile(kmd, method = "joint")
cpcp(kmd2)
cpcp(kmd3)

library(biclust)
data(BicatYeast)
Dby <- dist(BicatYeast)

hc1 <- hclust(Dby, method = "ward")
hc2 <- hclust(Dby, method = "average")
hc3 <- hclust(Dby, method = "complete")
hcc1 <- cutree(hc1, k = 6)
hcc2 <- cutree(hc2, k = 6)
hcc3 <- cutree(hc3, k = 6)

kml <- kmeans(BicatYeast, centers = 6, nstart = 100, iter.max = 30)$cluster

library(mclust)
mcl <- Mclust(BicatYeast, G = 6)$class

clusterings <- data.frame(hcc1,hcc2,hcc3,kml,mcl)
clusterings <- subtable(clusterings, 1:5, allfactor = TRUE)

clusterings2 <- optile(clusterings, method = "joint")
clusterings3 <- optile(clusterings, fun = "casort")

cpcp(clusterings2)

# a fluctuation diagram of all but the avg. clustering
fluctile(xtabs(Freq ~ ., data=clusterings2[,2]))

# compute agreement via Fleiss kappa in irr:
require(irr)
rawdata <-  untabelSet(clusterings2)
for(i in 1:5) levels(rawdata[,i]) <- 1:6
(kappam.fleiss(rawdata))
(kappam.fleiss(rawdata[,2]))

## Let's have a look at kmeans with 2:12 clusters
library(biclust)
data(BicatYeast)

cs <- NULL
for(i in 2:12) cs <- cbind(cs, kmeans(BicatYeast, centers=i,nstart=100)$cluster)

cs <- as.data.frame(cs)
names(cs) <- paste("v",2:12,sep="")
ocs <- optile(cs,method="joint")
cpcp(ocs,sort.individual=TRUE)
# and set alpha-blending to about 0.6, show.dots = TRUE

# and with hierarchical clusterings

cs2 <- NULL
library(amap)
hc <- hcluster(BicatYeast)
for(i in 2:20) cs2 <- cbind(cs2, subtree(hc,k=i)$data)

cs2 <- as.data.frame(cs2)
names(cs2) <- paste("v",2:20,sep="")
cpcp(cs2,sort.individual=TRUE)
# and set alpha-blending to about 0.6, show.dots = TRUE, then
ss <- iset()
ibar(ss$V6)
# and VIEW >> Set color (rainbow)
# Ideally the axes would be at a distance according to the heights of the cuts.
# e.g. for the first 12 clusters (after that there are some cuts at about the same height)

# the complete dendrogram doesn't look too attractive:
plot(hc)

# and plotting the top cuts only omits the information
# on how many cases are in each node or leaf

xcoords <- rev(tail(hc$height,11))
xcoords <- xcoords/max(hc$height)
ycoords <- data.matrix(ss[,20:30])
ycoords <- apply(ycoords,2,function(s){
y <- s - min(s)
y <- y/max(y)
return(y)
})
ycoords <- cbind(ycoords, as.integer(as.matrix(ss[,5])))
colv <- rainbow_hcl(6)
dev.new()
par(mfrow=c(1,2))
plot(1,pch='', ylim=min(xcoords)-0.007, ylim=(min(xcoords)+0.007))

apply(ycoords,1,function(s){
points(x=s[-12], y=xcoords,
points(x=s[-12],y=xcoords,pch=19, col = colv[s[-12]])
lines(x=s[-12], y=xcoords, col = colv[s[-12]])
})

hc$height <- hc$height/max(hc$height)
plclust(subtree(hc,12),hang=0.02)

################################### EXAMPLE IV ####################
# The Eisen Yeast data
#
library(biclust)
data(EisenYeast)
SEY <- scale(EisenYeast)

Dby2 <- dist(SEY)

hc1 <- hclust(Dby2, method = "ward")
hc2 <- hclust(Dby2, method = "complete")

hcc1 <- cutree(hc1, k = 16)
km1 <- kmeans(scale(EisenYeast), centers = 16, nstart = 20, iter.max = 30)$cluster
optile(table(hcc1, km1))

################################### EXAMPLE V ####################
# The Bicat Yeast data
#
# how many clusters are a good choice for kmeans?
# one possible way to find out:
# compute kmeans for 100 random initial settings, sort the results (clusters)
# and compute their agreement
# e.g. via Fleiss' Kappa (available in package irr)

require(biclust)
data(BicatYeast)
require(irr)

st <- Sys.time()
fk <- NULL
for(k in 3:8){
  test <- subtable(replicate(100,kmeans(BicatYeast, centers = k)$cluster),1:100)
  test <- optile(test, fun = "csort")
  test <- optile(test, method="joint")
  test <- untableSet(test)
  for(i in 1:100) levels(test[,i]) <- 1:k
  fk <- c(fk,kappam.fleiss(test)$value)
}
Sys.time()-st
plot(x = 3:8, y = fk, type="l", lwd=2)

# A list with hierarchical clustering objects:
require(ape)

hc1 <- hcluster(t(plants[-1]), method="manhattan", link = "ward")
hc2 <- hcluster(t(plants[-1]), method="manhattan", link = "complete")

hclist <- list(hc1, hc2)
tflu(tcube( optile(hclist, k= c(8,8) ) )

# or a table with corresponding tree objects:
n <- table( subtree(hc1, 12)$data, subtree(hc2, 8)$data )
tflu(tcube( optile(n, tree = list(hc1, hc2)) )

# only one tree object, the other variable is free:
n <- table( subtree(hc1, 8)$data, kk <- kmeans(t(plants[-1]),centers=8)$cluster )
tflu(tcube( optile(n, tree = list(hc1, NA)) )

## End(Not run)
**optME**

### Optimizing ME

**Description**

Computes optimal category orders for each dimension separately. Uses a TSP solver to achieve the best ME value.

**Usage**

```r
optME(x, dims = NULL, nstart = 1, 
solver = "nearest_insertion", 
return.table = TRUE, adjust.dist = FALSE)
```

**Arguments**

- **x**: A matrix, table or array.
- **dims**: Which dimensions to reorder. The dimensions are reordered independently.
- **nstart**: The number of different starting points for the TSP solver. If `nstart` is greater or equal to the number of cities in a dimension, the solver uses each city once.
- **solver**: Should be one of "nn", "nearest_insertion", "cheapest_insertion", "farthest_insertion". See `solve_TSP`
- **return.table**: Whether or not to return the optimized table. If `FALSE` only the new category orders are returned. If `TRUE` the new orders are attached to the table as an attribute "orders".
- **adjust.dist**: If `TRUE` a small proportion of the euclidean distances between the category profiles (e.g. rows) is added to the ME distance value. The idea is to keep identical profiles together which is otherwise not guaranteed, since the ME values can be identical even if the profiles aren’t.

**Details**

Each dimension is optimized separately via a TSP solver.

**Value**

The passed object as a table with optimized category orders.

**See Also**

- [ME](#)

**Examples**

```r
a <- arsim(2000,c(8,9,10),3,0.2)
ME(a)
a2<-optME(a)
ME(a2)
```
Description

Binary state variables indicating which of more than 30000 plants grow in that state.

Usage

data(plants)

Format

A data frame with 34781 observations on the following 70 variables.

V1 name
ab a numeric vector
ak a numeric vector
ar a numeric vector
az a numeric vector
ca a numeric vector
co a numeric vector
ct a numeric vector
dc a numeric vector
de a numeric vector
dc a numeric vector
fl a numeric vector
ga a numeric vector
hi a numeric vector
id a numeric vector
il a numeric vector
in a numeric vector
ia a numeric vector
ks a numeric vector
ky a numeric vector
la a numeric vector
me a numeric vector
md a numeric vector
ma a numeric vector
mi a numeric vector
mn a numeric vector
plants

ms a numeric vector
mo a numeric vector
mt a numeric vector
ne a numeric vector
nv a numeric vector
nh a numeric vector
nj a numeric vector
nm a numeric vector
ny a numeric vector
nc a numeric vector
nd a numeric vector
oh a numeric vector
ok a numeric vector
or a numeric vector
pa a numeric vector
pr a numeric vector
ri a numeric vector
sc a numeric vector
sd a numeric vector
tn a numeric vector
tx a numeric vector
ut a numeric vector
vt a numeric vector
va a numeric vector
vi a numeric vector
wa a numeric vector
wv a numeric vector
wi a numeric vector
wy a numeric vector
al a numeric vector
bc a numeric vector
mb a numeric vector
nb a numeric vector
lb a numeric vector
nf a numeric vector
nt a numeric vector
ns a numeric vector
nu a numeric vector
on a numeric vector
pe a numeric vector
qc a numeric vector
sk a numeric vector
yt a numeric vector
dengl a numeric vector
fraspm a numeric vector

Source
http://archive.ics.uci.edu/ml/datasets/Plants

---

qBCI | **Quantile BCI**
--- | ---

Description
Bins numeric variables according to their quantiles and computes the Bertin Classification Index BCI. The data.frame method computes the multivariate qBCI and not the pairwise values (c.f. cmat).

Usage
qBCI(x, ...)  
## Default S3 method:
qBCI(x, y, p = NULL, k = 5, iter=20, ...)  
## S3 method for class 'data.frame'
qBICI(x,y, p = NULL, k = 5, sort = TRUE, iter=20, ...)

Arguments

x A numeric vector (in this case y needs to be specified) or a data.frame with numeric or factor variables.

y A numeric vector.

p A percentage to use for the quantiles sequence. See details.

k A minimum expected number of observations in each cell after the binning.

sort Whether or not to compute the BCI for the optimized tables or not. If not, kendalls is usually a better alternative.

iter An optile parameter.

... dots
Details

The breakpoints for the binning are the data quantiles according to equidistant probabilities \( \text{seq}(0, 1, p) \) where \( p \) is minimal under the condition that each cell has an expected number of observations of at least \( k \).

Value

A value between 0 and 1.

Author(s)

Alexander Pilhoefer

See Also

BCI, kendalls, wdcor, cmat

Examples

```r
# Not run:
qBCI(rnorm(100), runif(100))

# non-functional relationship:
x1 <- runif(500, 0, 10)
x2 <- runif(500, 0, 10)
y1 <- x1 + rnorm(500, sd=1)
y2 <- 10 - x2 + rnorm(500, sd=1)
x <- c(x1, x2)
y <- c(y1, y2)
plot(x, y, pch = 19)
wdcor(x, y)
1 - qBCI(x, y)

y1 <- x1 + rnorm(500, sd=0.1)
y2 <- 10 - x2 + rnorm(500, sd=0.1)
x <- c(x1, x2)
y <- c(y1, y2)
plot(x, y, pch = 19)
wdcor(x, y)
1 - qBCI(x, y)

# or a quadratic curve:
test <- sapply(seq(0, 4, 0.2), function(s){
x <- runif(200, -1, 1)

```
```r
y <- 5+12*x^2+rnorm(200, sd=s)
return(c(cor(x,y),
wdcor(x,y),
1 - qBCI(x,y)))
}

plot(test[3,], type="l", ylim=c(-0.2,1))
lines(test[1,], col = 2, lwd = 2)
lines(test[2,], col = 3, lwd = 2)

## End(Not run)
```

---

**Description**

This function computes a fechnerian distance matrix from either a similarity matrix or a dissimilarity matrix. In addition to the basic procedure which looks for the shortest paths between the objects in the dissimilarity matrix a second approach is offered which connects similarities in a multiplicative manner.

**Usage**

```r
quickfechner(x, x.type = "diss", scale ="-", path.op = "+", sym.op = "+", 
rescale = FALSE, exclude.zero = FALSE, check = TRUE)
```

**Arguments**

- **x**: A similarity or dissimilarity matrix.
- **x.type**: The type of the matrix ("sim" or "diss").
- **scale**: Either divide the similarities by the diagonal entries ("div", "/", "+", "exp", "expected", "mult", "multiplicative") or subtract the diagonal entries in the dissimilarity matrix ("-", "+", "add", "additive")
- **path.op**: Whether to use the similarities to find multiplicative paths ("++++", "exp", "expected", "mult", "multiplicative") or to use the dissimilarities and find additive paths ("++++", "add", "additive", "max", "maximum")

Note that similarity matrices are simply converted to dissimilarity matrices by subtraction from 1. Other transitions such as 2M/(1+M) are not yet implemented and have to be done by hand.

- **sym.op**: This sets the function which is used to ensure symmetry. "min" uses the minimum value, "+"”, "sum" or "mean" use the sum. "none", NA or FALSE stand for no operation and hence the resulting matrix will not necessarily be symmetric.
- **rescale**: Whether or not the original diagonal will be used for a correction of the results.
- **exclude.zero**: If TRUE zero-entries are not considered in the updating algorithm.
- **check**: Whether or not to check for regular minimality or maximality.
Details

The algorithm first computes a dissimilarity matrix with a zero-diagonal. Then it iteratively tries to find shorter paths between the items.

Value

The Fechnerian distance matrix.

Author(s)

Alexander Pilhoefer

Examples

data(olives)
# not a distance matrix, but a similarity matrix in some sense
cx <- 1-abs(cor(olives[-c(1,2,11)]))

cx2 <- quickfechner(cx)

rownames(cx2) <- names(olives)[-c(1,2,11)]
plot(hclust(as.dist(cx2)))

dm <- matrix(runif(100), 10, 10)
dm <- dm+t(dm)
diag(dm) <- 0
dm2 <- quickfechner(dm)

dmS <- 1-dm/max(dm)
dmS2 <- quickfechner(dmS, x.type="sim", path.op = ")

## Not run:
# check triangular inequality:
extracat::trinq(dm)
extracat::trinq(dm2)
extracat::trinq(dmS2)

## End(Not run)

regmax(x)  regmin(x)

Description

Checks whether or not a matrix fulfills the regular maximality or minimality condition.

Usage

regmax(x)
regmin(x)
Arguments

x A symmetric data matrix.

Value

boolean

Author(s)

Alexander Pilhoefer

Examples

x <- replicate(20, rnorm(20))
cx <- abs(cor(x))
regmax(x)
regmin(x)
diag(cx) = runif(20)
regmax(x)
regmin(x)

---

rmb Multiple Barchart for relative frequencies and generalized Spineplots

Description

The `rmb` function basically produces a Multiple Barchart for the relative frequencies of some target categories within each combination of the explanatory variables. The weights of those combinations (that is the absolute frequencies) are represented in the total width of the corresponding barchart. The result is a graphic which allows to read the conditional target distributions exactly from the graphic without losing the information about the importance (in the sense of the number of observations) of the different combinations.

Additionally the `rmb` function allows to draw spineplots instead of the barcharts within each explanatory combination. On that score it can be seen as a generalization of Spineplots.

Usage

```r
## S3 method for class 'formula'
rmb(formula, data, col.vars = NULL, spine = FALSE,
circular = FALSE, eqwidth = FALSE, cat.ord = NULL, cut = NULL,
innerval = 1, freq.trans = NULL, num.mode = FALSE, max.scale = 1,
use.na = FALSE, expected = NULL, residuals = NULL, model.opt = list(),
gap.prop = 0.2, gap.mult = 1.5, col = "hcl", col.opt = list(), label = TRUE,
label.opt = list(), vp = NULL, ...)```
```r
## S3 method for class 'ftable'
```
rmb(x, col.vars = NULL, spine = FALSE, circular = FALSE,
    eqwidth = FALSE, cat.ord = NULL, freq.trans = NULL, max.scale = 1,
    use.na = FALSE, expected = NULL, residuals = NULL, model.opt = list(),
    gap.prop = 0.2, gap.mult = 1.5, col = "hcl", col.opt = list(), label = TRUE,
    label.opt = list(), vp = NULL, ...)

Arguments

x
Either a table or a model of class "glm" and family "poisson" or "binomial". A table must be either of class table or of class ftable. The latter also implicitly defines the order in which the variables will be added to the plot. The arguments formula and data will be omitted. Please note that the model based version is still beta and will be improved in a future release.

formula
The formula specifying the variables in their given order with the last variable being the target variable. The left hand side defines a weighting variable. If the weights are frequencies in a variable called "Freq" this is detected automatically if no other variable is defined.

data
The dataset as a data.frame or ftable.

col.vars
Logical vector with split directions where TRUE stands for horizontal splitting. The last (target) variable is always arranged on the x-axis.

spine
If TRUE a spineplot will be drawn instead of each barchart. This is recommended for binary target variables.

circular
If TRUE a piechart will be drawn instead of each barchart. spine is set to FALSE.

eqwidth
If TRUE the bar length of the multiple barchart in the background no longer restricts the width of the barcharts/spineplots for the relative frequencies of the target variable.

cat.ord
A vector specifying the categories of the target variable which will be visualized in the specified order. The default is to use all categories.

cut
Numeric variables will be cut into this number of intervals. May also be a vector with specifications for each variable.

innerval
The function innerval is used to reduce numeric variables to an interval which is symmetric around the median contains the specified proportion of observations (or as close to this as possible).

freq.trans
This parameter allows to transform the absolute frequencies used for the underlying multiple barchart. Possible values are "log", "sqrt" or list("sqrt",k). The latter stands for the k-th root transformation.

num.mode
In the numeric mode the gaps are removed and axes typical for numeric variables are drawn. Ignored for factor variables.

max.scale
The maximum value of the probability (y-axis) scale for each combination. Unsurprisingly the default is 1. The axis will be drawn if yaxis is TRUE.

use.na
If TRUE missing values will be changed to a level "N/A" and else (which is the default) the function na.omit will be called to reduce the dataset to complete cases only.
There are three possibilities how to specify this parameter:

1. A list of integer vectors denoting the interaction terms in the poisson or proportional odds model, e.g. `list(c(1,2,3), c(1,4))` for all interactions between variables 1, 2 and 3 as well as between 1 and 4.
2. A logical indicating whether or not to use a model (logit independence model).
3. A vector with expected values, e.g. from a model. If `residuals` remains undefined the response residuals will be plotted.

If undefined or set to `FALSE` only the observed values will be plotted.

If expected is a vector with expected values it is also possible to specify residuals. This is used internally by `rmb.glm`.

A list with optional parameters for model specifications. Possible parameters are:

- `use.expected.values`: A logical specifying whether or not to use the frequencies predicted by the model instead of the observed frequencies.
- `mod.type`: Either "poisson" or "polr". See `glm` and `polr`.
- `resid.type`: "pearson", "deviance", working, partial or "response". For polr models only the latter is available.
- `resid.display`: One of "values", "color" or "both". "values" will result in bars or wedges for both expected and observed frequencies. Hence the raw residuals are shown in the graph. "color" will set the `col` argument aside and use colors on a red-blue-scale to represent (pearson) residuals. "both" does both.
- `max.rat`: If a model is specified and `resid.display = "both"` the x-scales will not be reduced to less than `1/max.rat`.
  - The x-scales are reduced whenever an observed frequency exceeds the maximal scale.

The maximum proportion of the total plot width which is used for the gaps.

The incremental multiplier for the gaps of different dimensions. The gaps corresponding to any one variable are `gap.mult` times larger than those corresponding to the next variable on the same axis.

Either a vector defining the colors of the bars or a name specifying a palette: "hsv" and "rgb" for hsv-based rainbow colors, "hcl" for hcl-based rainbow colors (default), "div" or "diverge" for hcl-based diverging colors and finally "seq" or "sequential" for hcl-based sequential colors. Additional arguments can be specified via the `col.opt` argument according to the underlying functions in the colorspace package, e.g. `rainbow_hcl`. For the hsv-based colors see `rainbow`. Specifying a color or palette has no effect if an expected model is defined.

Further options for the color palettes. See e.g. `rainbow_hcl` or `rainbow`. Other parameters are:
col2 for the color of the background/weight bars,
line.col for the color of all lines (bars, rectangles),
bg for the background color of the whole graphic,
bgs for the background color of each tile

label Either a logical specifying whether or not to draw labels or a numeric vector defining which variables shall be labelled.

label.opt A list with optional parameters for label specifications. Possible parameters are:

  yaxis If TRUE a vertical axis will be drawn at both sides of the plot.
  This is recommended when changing the max.scale argument.
  boxes Should the labels be surrounded by boxes?
  lab.tv Should the target variable be included in the labeling?
  varnames Should the variable names be shown as labels?
  abbrev An single integer value or a vector of integer values specifying the number
           of characters to which the labels will automatically be abbreviated.
  lab.cex The fontsize multiplier.

vp An optional viewport to plot in. vp = c(i, j) can be used as a shortcut to
    viewport(layout.pos.row = i, layout.pos.col = j)

... further arguments. Usually not necessary.

Details

A similar way to regard the graphic is the following: A Multiple Barchart of the explanatory variables is drawn with bars in horizontal direction. Then within each of the resulting bars a barchart of the conditional distribution of the target variable is drawn with bars in vertical direction.

Value

invisible(TRUE)

Author(s)

Alexander Pilhoefer
Department for Computer Oriented Statistics and Data Analysis
University of Augsburg
Germany

References

See Also

mosaicplot

Examples

```r
require(MASS)

# simple example
rmb(formula = ~Type+Infl+Cont+Sat, data = housing, gap.mult = 2,
col.vars = c(FALSE,TRUE,TRUE,FALSE), label.opt = list(abbrev = 3))

# with sqrt-transformation and horizontal splits only
rmb(formula = ~Type+Infl+Cont+Sat, data = housing, gap.mult = 2,
col.vars = c(TRUE,TRUE,TRUE,TRUE), freq.trans = "sqrt",
label.opt = list(abbrev = 3) )

# a generalized spineplot with the first category highlighted
rmb(formula = ~Type+Infl+Cont+Sat, data = housing, spine = TRUE,
cat.ord = 1, mult = 2, col.vars = c(1,3,4),
freq.trans = list("sqrt",3), label.opt = list(abbrev = 2))

## Not run:
# a generalized spineplot with all categories highlighted
# in a changed order
rmb(formula = ~Type+Infl+Cont+Sat, data = housing, spine = TRUE,
cat.ord = c(3,1,2), gap.mult = 2, col.vars = c(TRUE,FALSE,TRUE,TRUE),
freq.trans = "sqrt", label.opt = list(abbrev = 3))

# the barchart version only for categories 1 and 3
rmb(formula = ~Type+Infl+Cont+Sat, data = housing,
cat.ord = c(1,3), gap.mult = 2, col.vars = c(TRUE,FALSE,TRUE,TRUE),
freq.trans = "sqrt", label.opt = list(abbrev = c(1,1,1)))

# with equal widths
rmb(formula = ~Type+Infl+Cont+Sat, data = housing, eqwidth = TRUE,
gap.mult = 2, col.vars = c(TRUE,FALSE,TRUE,TRUE),
label.opt = list(abbrev = 2, lab.tv = TRUE))

# ----- models and residuals ----- #
# using the logistic model: Sat by Type only

# residual shadings and expected values
rmb(formula = ~Type+Infl+Cont+Sat, data = housing,
gap.mult = 2, col.vars = c(TRUE,FALSE,TRUE,TRUE),
label.opt = list(abbrev = 3), expected = list(c(1,2,3),c(1,4)),
model.opt = list(use.expected.values = TRUE, resid.display = "color") )

# residual values without shadings
rmb(formula = ~Type+Infl+Cont+Sat, data = housing,
gap.mult = 2, col.vars = c(TRUE,FALSE,TRUE,TRUE),
label.opt = list(abbrev = 3), expected = list(c(1,2,3),c(1,4)),
model.opt = list( resid.display = "values" )
```
# residual shadings and expected values
rmb(formula = ~Type+Infl+Cont+Sat, data = housing,  
gap.mult = 2, col.vars = c(TRUE, FALSE, TRUE, TRUE),  
label.opt = list(abbrev = 3), expected = list(c(1,2,3),c(1,4)),  
model.opt = list(use.expected.values = TRUE, resid.display = "color") )

# barcharts with residual shadings and values
rmb(formula = ~Type+Infl+Cont+Sat, data = housing,  
gap.mult = 2, col.vars = c(TRUE, FALSE, TRUE, TRUE),  
label.opt = list(abbrev = 3), expected = list(c(1,2,3),c(1,4)) )

# spineplots with residual shadings and values
rmb(formula = ~Type+Infl+Cont+Sat, data = housing, spine = TRUE,  
gap.mult = 2, col.vars = c(TRUE, FALSE, TRUE, TRUE),  
label.opt = list(abbrev = 3), expected = list(c(1,2,3),c(1,4)) )

# piecharts with residual shadings and values
rmb(formula = ~Type+Infl+Cont+Sat, data = housing, circular = TRUE,  
gap.mult = 2, col.vars = c(TRUE, FALSE, TRUE, TRUE),  
label.opt = list(abbrev = 3), expected = list(c(1,2,3),c(1,4)) )

# ----- using an ftable to create the plot ----- #
tt = xtabs(Freq~Type+Cont+Infl+Sat, data = housing)  
ft = ftable(tt, col.vars= c(1,4))  
rmb(tt)  
rmb(ft)

# ----- using a glm model ----- #
mod1 <- glm(Freq ~ Type+Infl+Cont + Type+Sat, data = housing, family = poisson)  
rmb(mod1, circular = TRUE,  
gap.mult = 2, col.vars = c(TRUE, FALSE, TRUE, TRUE),  
label.opt = list(abbrev = 3), model.opt = list(use.expected.values = TRUE) )

# ----- the numeric mode and cuts ----- #
data(olives)  
# only three cuts to show how it works  
rmb(~palmitoleic+stearic+region, data = olives, cut = c(3,3,0))

require(ggplot2)  
data(diamonds)  
diamonds$price <- log(diamonds$price)  
# a minority of extreme observations mess the display up:  
rmb(~depth+table+price, data = diamonds, eqwidth = TRUE, spine = TRUE,  
cut = c(36,36,5), col = "seq", num.mode = TRUE)

# we can zoom in via innerval:  
rmb(~depth+table+price, data = diamonds, circular = TRUE,  
cut = c(36,36,5), col = "div", innerval = 0.95,  
num.mode = TRUE, freq.trans = "log")

# price, carat and color  
diamonds$price <- log(diamonds$price)
rmbmat

Pairwise RMB-Plots

Description

This function generates a matrix with RMB-plots of all pairs of variables with a specified target variable. Both categorical and numerical variables are accepted and the latter will be binned. This makes the graphic useful for a mixture of variable types and the binning avoids overplotting and color mash as it occurs in (colored) scatterplots of large datasets.

Usage

```
rmbmat(x, tv, cut = 20, freqvar = NULL, plot.tv = FALSE, num.mode = TRUE, mode = "circular", eqwidth = FALSE, freq.trans = "sqrt", innerval = 1, allocation = 1, max.scale = 1, use.na = FALSE, expected = FALSE, model.opt = list(), gap.prop = 0.2, gap.mult = 1.5, col = "hcl", col.opt = list(), label = FALSE, label.opt = list(), diag.opt = list(), lower.opt = list(), upper.opt = list(), rc.opt = list(), factor.opt = list(), ...)```

Arguments

- `x`: Anything that can be converted to a `data.frame` via `as.data.frame`.
- `tv`: The index of the target variable. The target variable will not be plotted unless `plot.tv` is `TRUE`.
- `cut`: The number of intervals into which numeric variables will be cut.
- `freqvar`: An optional frequency variable. "Freq" is handled automatically.
- `plot.tv`: Whether or not to include the target variable(s) in the plot.
- `num.mode`: Whether or not to use the numeric mode (no gaps and a numeric axis) for numeric variables.
- `mode`: One of "circular", "pie", "piechart", "p" or "c" for piecharts, "spine" or "s" for spineplots, "bars", "bar" or "b" for barcharts.

**NOT YET IMPLEMENTED:** "rect" or "r" for nested rectangles. "nested.circles" are abbreviated by "nc" or "ncircles".
- `eqwidth`: See `rmb`.
- `freq.trans`: See `rmb`.
- `innerval`: See `rmb`.
- `allocation`: The widths and heights for the plots are proportional to `allocation(nlevels(x))`.
max.scale See rmb.
use.na See rmb.
expected See rmb.
model.opt See rmb.
gap.prop See rmb.
gap.mut See rmb.
col See rmb.
col.opt See rmb.
label See rmb.
label.opt See rmb and details.
diag.opt A list with rmb parameters. These overwrite the general parameters for all plots on the diagonal.
lower.opt The same as diag.opt but for the lower triangular matrix. Additionally it is possible to define a second target variable, e.g. lower.opt = list(tv2 = 3, ... ).
upper.opt The same as diag.opt but for the upper triangular matrix. Additionally it is possible to define a second target variable, e.g. upper.opt = list(tv2 = 3, ... ).
rc.opt A list with which it is possible to define parameters for single matrix cells (plots), columns or rows. This will overwrite all other parameters for the specified plots. It works like this:
rc.opt = list( r2c12 = list(spine = FALSE), r1 = list(col="rgb"), c4 = list(col="seq"))
where the plot in row 2 and column 14 is a spineplot, the first row uses RGB colors and the fourth column a sequential color palette. Later arguments overwrite the preceding ones. For instance in the example the plot in row 1 and column 4 will use the sequential color palette.
factor.opt The same as diag.opt, lower.opt, upper.opt but for all pairs of two categorical variables. This overwrites the other option lists.
...
Further parameters.

Details

Creates a matrix of all pairwise rmb-plots using all possible rmb parameters except cat.ord, expected =list() and residuals. The parameters are applied to all plots and afterwards possibly overwritten by one of the parameter lists.

Value

An environment with the parameter lists and matrices. This can be used to update (parts of) the plot without a complete new construction. The update.rmbmat function is under development.

Author(s)

Alexander Pilhoefer
See Also

rmb, pairs

Examples

data(olives)

## Not run:

# mode = "c" piecharts are currently slow
rmbmat(olives, tv=2, mode = "s")
rmbmat(olives[,1:5], tv=2, col ="div", plot.tv = TRUE, lower.opt = list(tv2 = 1, col ="rgb")
rmbmat(olives[,c(1:5,11)], tv=2, upper.opt=list(mode="s", eqwidth = TRUE),
rc.opt = list( c5 = list(eqwidth=FALSE,mode="s"),
r5 = list(eqwidth=TRUE, mode="s")),allocation=NULL)

## End(Not run)

---

scpcp

**Static Categorical Parallel Coordinates Plot**

Description

This function creates a static CPCP plot using base R graphics. The function offers color brush / highlighting and several options for the labels and colors. Efficiency is improved by replacing sets of parallel lines by polygons.

A ggplot version is under construction. A deprecated interactive version based on iplots (without labeling) is still available as extracat::cpcp.

For reordering of category orders in CPCP plots see steptile.

Usage

scpcp(data, freqvar = "Freq", max.N = 1e6, gap = 0.2, sort.individual = TRUE, level.width = 0.2, polygon = TRUE, base.colour = alpha("black", 0.7), label = TRUE, lab.opt = list(rot = 0, col = 1, bg = TRUE, abbr = FALSE, abbr.var=12,hide.sel=TRUE, var.labels = TRUE), sel = NULL, sel.hide = TRUE, sel.palette = NULL, col.opt = list(), plot = TRUE, return.coords = !plot)
Arguments

data The data.frame which can contain a variable called "Freq".
freqvar Optional specification of a frequency variable.
max.N The plot handles each case as a separate polyline, similar to conventional pcps. This option limits the number of observations.
gap The size of the gaps between the categories as a total proportion.
sort.individual Whether or not the cases (lines) are additionally rearranged according to the neighboring variable. This minimizes crossings.
level.width The width of the rectangles representing the variables/categories.
polygon Whether or not to replace parallel lines by polygons. This improves both efficiency and undesirable color effects.
base.colour The standard color used for the cases which are not highlighted via sel.
label Whether or not to draw category labels.
lab.opt A list with options for the labels. See e.g. rmb
sel A selection defining colors. This can either be an integer vector, a factor or an expression which returns such a vector. For instance sel="data[,4]" colors by the fourth variable, sel="sample(1:6,nrow(data),T)" leads to random colors and sel="Sex=='Male' & Survived=='Yes'" selects survivors among the men (in the titanic data, see examples). The objects (observations/lines) are additionally sorted by their color which brings colors together and makes it possible to see proportions of the selections.

sel.hide Whether or not to hide the sel variable or plot it as the first variable.

sel.palette The color palette for the selection. See getcolors.
col.opt A list of options for the color palette. See getcolors.
plot Whether or not to plot.
return.coords Whether or not to return the coordinates per observation. plot = FALSE together with return.coords = FALSE is therefore rather stupid. The polygon coordinates are currently not returned.

Value

Either a logical value or the coordinates defining the polylines per observation.

Note

A grid-based version is still under construction. In polygon mode single cases are still shown as a line rather than a ribbon of width 1/n.

Author(s)

Karin Maria Gehweiler and Alexander Pilhoefer.
References


See Also

steptile

Examples

data(Titanic)
titanic <- as.data.frame(Titanic)

scpcp(titanic)

#scpcp(titanic, level.width=0)
#scpcp(titanic, gap=0)
#default with highlighting
scpcp(titanic, sel="data[,4]")

# random colors like for instance from a clustering
scpcp(titanic, sel="sample(1:6,nrow(data),T)")

# another one with some formal changes
require(scales)
scpcp(data=titanic, sel="Sex=='Male' & Survived=='Yes'", sel.palette = "w",
col.opt=list(alpha=0.7,border=alpha(1,0.3)), gap = 0.5, level.width= 0.3)

## Not run:

# mushroom data from the UCI machine learning repository
MR <- read.table("http://rosuda.org/mitarbeiter/pilhoefer/agaricus.dat",
sep="\t",quote="",header=TRUE)
levels(MR$stalk_root) <- c(levels(MR$stalk_root),"N/A")
MR$stalk_root[which(is.na(MR$stalk_root))] <- "N/A"

op <- optile(MR[,1:12], method="joint")

scpcp(op, sel = "odor",sel.palette="w",
col.opt = list(border = alpha(1,0.1)), lab.opt=list(rot=45))

# ADAC ecotest data with four clusterings (k-means, mclust, hc Ward, hc complete)
eco <- read.table("http://rosuda.org/mitarbeiter/pilhoefer/eco2plus.dat",
sep="\t",quote="",header=TRUE)
# Illustrate reordering success using coloring
scpccp(eco[,13:16], sel = "data[,1]", sel.paletaue="d")

cpccp(opitile(eco[,13:16]), sel = "data[,1]", sel.paletaue="d", 
col.opt = list(border=alpha(1,0.1)))

# Car classes (lower to upper class)
eco$Klasse <- factor(eco$Klasse, levels = levels(eco$Klasse)[c(3,1,2,7,4,5,6)])

scpccp(eco[,17:20], sel = eco$Klasse, sel.paletaue="s", col.opt = list(h=140))

# The color variable included
scpccp(eco[,c(3,17:20)], sel = eco$Klasse, sel.paletaue="s", 
col.opt = list(h=140), lab.opt = list(abbr=5))

## End(Not run)

---

### setcover

**greedy setcover optimisation**

### Description

This function takes an indicator matrix with rows representing objects and columns representing sets and computes a minimal redundancy free set using the greedy setcover optimization algorithm. The aim is to find a minimal set of clusters which covers all objects (or a minimum proportion rat).

Alternatively the number of clusters k can be specified. Then the problem becomes a maximum coverage problem. Both versions also permit weights such as frequencies (weighted setcover/maximum coverage).

### Usage

```r
setcover(x, k = NULL, rat = 1, s = NULL, w = NULL, check = TRUE)
```

### Arguments

- **x**
  - The indicator matrix.

- **k**
  - An optional number of clusters.

- **rat**
  - The minimum proportion of objects that is to be covered by the cluster set. If weights are specified in w then those are respected.

- **s**
  - If weights are specified but not all objects are covered by one of the sets it can be necessary to specify the total weight in order to compute a sensible ratio.

- **w**
  - Optional weights per object.

- **check**
  - Whether or not to check for redundancies.

### Value

The indices of the clusters in the minimal redundancy-free set. The result is not always the globally optimal solution since the algorithm is greedy.
sortandcut

sortandcut

Sort-and-Cut Reordering

Description

An implementation of the sort-and-cut algorithm which is a mixture of the top-down-partitioning algorithm used by cfluctile and the reordering techniques available through optile. See details.

Usage

sortandcut(x, iter=20, tau0 = NULL, fun = "BCC", method = "WBCI")
Arguments

- **x**: A matrix or 2D table.
- **iter**: The number of random initial orderings for `optile`.
- **tau0**: The minimum criterion value for a new cut. See `cfluctile`.
- **fun**: The reordering function used by `optile`. Currently "BCC", "barysort" and "preclass" are available.
- **method**: The method argument for `cfluctile` which defines the criterion used to find an optimal partition.

Details

The algorithm sorts a matrix using `optile` and cuts the reordered matrix once using `cfluctile` with `nsplit = 1`. Then the same procedure is applied to the resulting submatrices at the top left and the bottom right. The partitioning stops when the best cut leads to a criterion below `tau0`.

Value

The reordered matrix. The row and column order vectors are attached as attributes `attr(x, "orders")`.

See Also

- `cfluctile`

Examples

```r
M <- arsim(12000, c(30, 40), 7, noise = 0.3)
c1 <- cfluctile(M1 <- optile(M, iter = 20))
c2 <- cfluctile(M2 <- sortandcut(M))
```

---

**steptile**

*stepwise reordering*

Description

Starts with the first k+1 variables and applies `optile` to the corresponding subtable. Then one additional variable at a time is reordered using the subtable defined by this variable and the last k variables. Only the current variable is reordered and the others are fixed since they have been reordered in the previous steps.

Usage

```r
steptile(x, k = 1, cpcp = FALSE, ...)
```
Arguments

x  The data.frame (which is better for high-dimensional data) or data table.
k  The number of preceding variables used for the reordering. E.g. if \( k = 3 \) then variable 6 is reordered using the variables 3, 4, 5, 6.
cp cp  If TRUE a special version of the algorithm which minimizes crossings in CPCP plots (e.g. scpcp) is used. This modification works with aggregations of the last \( k \) variables and is much faster than the standard procedure if \( k > 1 \).

Details

The optile function also offers stepwise reordering via the argument method = "sw" but always starts with the first pair of variables and then considers the complete past: for the reordering of variable i all variables \( 1 \ldots (i-1) \) are considered. The stepwise algorithms are applicable to high-dimensional problems with a large number of variables where the multivariate techniques fail. Even if \( k \) is high (i.e. the subtables are also high-dimensional) the procedure is very fast since it can use the following trick: instead of applying optile to the multidimensional table it is applied to a 2D-table with one dimension defined by the variable that is reordered and the other dimension defined by the (ordered) combinations of all other variables. This way only combinations which appear at least once in the dataset matter and all empty entries (the majority in high-dimensional tables) can be left aside. The maximum possible size of such a table is therefore \( N \times \max(n_i) \) when \( N \) is the number of observations and \( n_i \) is the number of categories in dimension i.

Value

The reordered data either as a table or data.frame depending on the input type.

Author(s)

Alexander Pilhoefer

See Also

optile

Examples

```r
## Not run:

# scaled numeric variables from the olives data
# and 20 k-means solutions
so <- scale(olives[,3:10])
rr <- replicate(20,
    kmeans(so,8)$cluster
)

# par(mfrow=c(3,1))

# initial cluster orders
```
x <- as.data.frame(cbind(olives[,1:2],rr))
require(scales)
scpcp(x, sel = "data[,1,]",
   sel.palette="rgb", col.opt=list(alpha=0.5))

# reordering using steptile.
# optile does not work for the complete table since it has 9*3*2^60 > 3E19 entries
# colors by the first unordered example:

x2 <- steptile(x, k = 4)
scpcp(x2, sel = "match(data[,1],levels(.GlobalEnv$x[,1]))",
   sel.palette="rgb", col.opt=list(alpha=0.5))

# additionally reordering the variables ... cmat takes about 20-30 seconds
CM <- cmat(x[,3:22])
require(seriation)
SM <- get_order(seriate(1-CM))
x3 <- steptile(x2[, c(1,2,2+SM,23)], k = 4)
scpcp(x3, sel = "match(data[,1],levels(.GlobalEnv$x[,1]))",
   sel.palette="rgb", col.opt=list(alpha=0.5))

## End(Not run)

---

**subtable**

**data.frame reduction**

**Description**

Reduces a data.frame into a frequency table with pre-specified entries. Uses a modified version of the count function which also accepts weights. Zero-entries can be included or excluded and the variables can be coerced into factors if necessary.

**Usage**

```r
subtable(data, cols, freqvar = "Freq",
   keep.zero = FALSE, allfactor = FALSE, return.type = class(data))
```

**Arguments**

- **data**
  - The data.frame to reduce.
- **cols**
  - An ordered integer vector containing the indices of the columns to keep.
- **freqvar**
  - Optional name of a frequency variable in `V`.
- **keep.zero**
  - A logical indicating whether to include zero-cases in the output.
allfactor  A logical indicating whether to convert all variables into factor variables. Integer variables will be applied a fitting (non-lexicographic) level order.
return.type  The function is able to convert the output to a data.frame or table. The default is to use the same type as the input had.

Details
This function uses count as a workhorse and offers additional arguments keep.zero and allfactor. Both raw datasets, datasets with a frequency variable and tables can be handled.

Value
A data.frame including a "Freq" variable.

Author(s)
Alexander Pilhoefer
Department for Computer Oriented Statistics and Data Analysis
University of Augsburg
Germany

Examples
```r
require(MASS)
hs2 = subtable(housing,c(3,1))
summary(hs2)

A <- arsim(33333,c(11,11,11,11),3)
subtable(A, c(1,4))
```

Description
Takes a subtree of a dendrogram object such as generated by hclust according to a prespecified number of clusters or a prespecified height.

Usage
```r
subtree(tree, k = NULL, h = NULL)
```

Arguments
- **tree**  The tree object which contains the attributes merge and height in the same way as an hclust object.
- **k**  The number of clusters at which to cut.
- **h**  The height at which to cut.
Details

Does the same as `cutree` with two differences: Firstly it gives back an entire tree object, i.e. an object with attributes `merge`, `height`, `labels` and `order`, as well as `data`, which contains the cluster ids. Secondly the cluster ids are chosen by the heights at which the clusters were built.

Value

An `hclust` object.

Author(s)

Alexander Pilhoefer

See Also

`cutree`

Examples

```r
hc <- hclust(dist(USArrests), "ave")

hcs <- subtree(hc, k = 7)
hs <- subtree(hc, h = 30)

attributes(hcs)
fluctile(table(hcs$data, cutree(hc, k=7)))

par(mfrow=c(1,3))
plot(hc)
plot(hcs)
plot(hcs)
```
Arguments

x  The two-way table or matrix with the data.
tree  A list with tree objects. This may be NULL or will be disregarded if x has an attribute attr(x, "tree") which should also be a list. The latter way is the standard for objects returned by optile.list or optile.
dims  If x has more than two dimensions this vector of length 2 indicates which variables to plot.
tw  The proportion of the total space to the left and at the top which is used for the dendrogram.
border  How much space is left white around the dendrogram.
shape  Instead of rectangles ("r") it is possible to use circles ("c"), diamonds ("d") or octagons ("o"). The arguments dir and just work for rectangular shapes only.
dir  The bar/rectangle direction: "v" and "h" stand for vertical or horizontal bars. "b" stands for "both" and leads to standard fluctuation diagrams with quadratic rectangles. Use "n" for a same-binsize-plot.
just  A shortcut version of the argument used in grid for the anchorpoint of the rectangles: "rb" is equivalent to c("right", "bottom"), "t" is equivalent to "ct" or c("centre", "top") and so on. See examples.
tile.col  The tile color.
bg.col  A background color for the cells.
vp  A viewport to which the plot should be added or NULL.
lab.opt  A list with options for the labels. Currently lab.cex and abbrev work. Also lwd and line.col are the line width and the line color for the dendrogram.
...  further args

Value

invisible(TRUE)

Author(s)

Alexander Pilhoefer

See Also

fluctile, cfluctile

Examples

## Not run:
library(ape)
hc1 <- hclust(t(plants[,1:3]), method="manhattan", link = "ward")
hc2 <- hclust(t(plants[,1:3]), method="manhattan", link = "complete")
hclist <- list(hc1, hc2)
tfluctile( tt<-optile(hclist, k= c(8,8) ) )
untab|seSet

s1 <- subtree(hc1, k = 12)
s2 <- subtree(hc2, k = 10)

tfluctile( table(s1$data, s2$data), tree = list(s1, s2))

## End(Not run)

---

**untab|seSet**

*data frame conversion*

**Description**

Converts a frequency table into a raw data.frame.

**Usage**

`untab|seSet(data, freqvar = "Freq")`

**Arguments**

- `data`: The `data.frame` including a frequency variable "Freq".
- `freqvar`: The name of the frequency/weights variable which is used to expand V.

**Value**

A `data.frame`.

**Author(s)**

Alexander Pilhoefer
Department for Computer Oriented Statistics and Data Analysis
University of Augsburg
Germany

**References**

Alexander Pilhoefer *New approaches in visualization of categorical data: R-package extracat*  
Journal of Statistical Software, submitted March 2010

**Examples**

```r
require(MASS)
hs2 = untab|seSet(housing)
summary(hs2)
```
**MovieLens USER data**

**Description**

The user data from the MovieLens 1M data.

**Usage**

```r
data(USR)
```

**Format**

A data frame with 6040 observations on the following 25 variables.

- UserID  a numeric vector
- Gender  a factor with levels F M
- Age     a numeric vector
- Occupation  a numeric vector
- Zip.code a factor
- occupation a factor
- zip1    a numeric vector
- zip2    a numeric vector
- zip12   a numeric vector
- UserVotes  a numeric vector
- meanUserRat  a numeric vector
- medianUserRat  a numeric vector
- sdUserRat    a numeric vector
- minTs     a numeric vector
- AvgRat    a numeric vector
- meanDiffRat  a numeric vector
- meanAbsDiffRat  a numeric vector
- Pct.Action  a numeric vector
- Pct.Adventure a numeric vector
- Pct.Animation a numeric vector
- Pct.Children.s a numeric vector
- Pct.Fantasy a numeric vector
- Pct.Horror  a numeric vector
- Pct.Sci.Fi  a numeric vector
- Pct.Comedy  a numeric vector
Source
MovieLens.org

Examples
data(USR)
## maybe str(USR) ; plot(USR) ...

visid Visualizing Indicator Matrices and Missing Values

Description
This function aggregates a binary dataframe or matrix using `subtable` and visualizes the combinations along with the marginal distributions using `fluctile`. Options include reordering of rows and columns, filtering the most important rows and columns as well as ceiling censored zooming for the marginals.

Missing values can be visualized using `is.na` and datasets with categorical variables can be transformed via `idat`. `visna(x)` is a shortcut for `visid(is.na(x) + 0)`. `visdf(x, freqvar)` is a shortcut for `visid(idat(x, allcat = TRUE, keep = freqvar))`.

Usage
```
visid(x, freqvar = "Freq", tp = FALSE, fr = 1, fc = 1, sort = "n",
      sort.method = "count", col = "w",
      mar.col = c(alpha("black", 0.7), alpha("darkred", 0.8), "red", "green"),
      s = Inf, pmax = 1, opts = list(), plot = TRUE, return.data = !plot, ...)
visna(x, freqvar = "Freq", tp = FALSE, fr = 1, fc = 1, sort = "n",
      sort.method = "count", col = "w",
      mar.col = c(alpha("black", 0.7), alpha("darkred", 0.8), "red", "green"),
      s = Inf, pmax = 1, opts = list(), plot = TRUE, return.data = !plot, ...)
visdf(x, freqvar = "Freq", tp = FALSE, fr = 1, fc = 1, sort = "n",
      sort.method = "count", col = "w",
      mar.col = c(alpha("black", 0.7), alpha("darkred", 0.8), "red", "green"),
      s = Inf, pmax = 1, opts = list(), plot = TRUE, return.data = !plot, ...)
```

Arguments
- **x** A binary dataframe or matrix. `is.na` and `idat` can be used to bring in missing values and categorical variables.
- **freqvar** An optional frequency variable. If this is not found the data is aggregated using `subtable`.
- **tp** Logical. Whether or not to transpose the indicator matrix for the visualization.
- **fr** This controls the row filtering: Only the fr most frequent rows are kept. Values below 1 are interpreted as proportions and only the minimum number of rows covering at least fr percent of the observations are kept.
fc
See fr.

sort
One of "n" (no reordering), "r" (row reordering), "c" (column reordering) or "b" (row and column reordering).

sort.method
The default is reordering by frequency (rows) and average (columns). Other options include "mE" for reordering by the measure of effectiveness (see optME) and "optile" which enables all reorderings offered by optile.

col
The color palette. For the basic indicator case only a single color is meaningful. Transformations via idat feature coloring by variable.

mar.col
Colors for the marginals: rows, columns, row markers, column markers (see s and pmax).

s
Ceiling censored zooming for the rows. The maximum of the scale is s times the second largest value.

pmax
The maximum for the average/percentage scale used for the column marginal plot.

opts
A list of options. Currently:

gap.prop
  gaps proportion used in fluctile).
  The default is to use no gaps if more than 40 rows or columns are involved.
mar
  Vector controlling the size of the marginal plots.
border
  Borders around the plot.
shape
  Shape of the symbols. Default is rectangles.
bg.col
  Background color center/right/bottom
abbrev
  Label abbreviation.

plot
Whether or not to draw the plot.

return.data
Whether or not to return the data after filtering and reordering took place.

... Further arguments passed to either optME or optile. Can for instance be used to choose the reordering method.

Value
The filtered and reordered data or invisible(TRUE) (default).

Author(s)
Alexander Pilhoefer

See Also
fluctile, optile, idat

Examples
## Not run:
require(reshape2)
require(scales)

MJ <- read.table(
  "http://www.rosuda.org/~pilhoefer/MJnew.txt",
  header=T, sep="\t", quote=""
)

MJS <- MJ[,13:105]

visid(MJS)

# sort by count/percentage
visid(MJS, sort="b")

# sort via ME
visid(MJS, sort = "b", sort.method="ME")

# only rows, only columns
visid(MJS, sort = "r", sort.method="ME")
visid(MJS, sort = "c", sort.method="ME")

# sort via optile
visid(MJS, sort = "b", sort.method="optile")
visid(MJS, sort = "b", sort.method="optile", iter=10)
visid(MJS, sort = "b", sort.method="optile", fun="ca")

# 24 rows
visid(MJS, sort = "r", sort.method="optile", fr=24)

# 24 rows, >= 40
visid(MJS, sort = "r", sort.method="optile", fr=24, fc = 0.4)

# zoom y marginal
visid(MJS, sort = "r", sort.method="optile", fr=24, s=1)

# zoom x marginal
visid(MJS, sort = "r", sort.method="optile", fr=24, pmax=0.1, s =0.5)

## End(Not run)

# NA-example: GeneEx
visna(GeneEx, sort = "b", sort.method="optile", fr=50, pmax=0.05, s = 2)
require(MASS)
The Weighted Bertin Classification Index

Description

The weighted Bertin Classification Criterion using weights according to the Hamming distance is
normalized by means of the independence case.

Usage

WBCI(x)

Arguments

x

A data matrix.

Value

The criterion value.

Author(s)

Alexander Pilhoefer

See Also

kendalls

Examples

M <- arsim(1000, c(12,12), 3)
BCI(M)
WBCI(M)

M2 <- optile(M, iter = 100)
BCI(M2)
WBCI(M2)
**wdcor**

Weighted Distance Correlation

### Description

An efficient implementation of the distance correlation for two variables with the additional option to weight the observations. The main application for the weights is to use frequencies according to ordinal variables which can be represented by a contingency table (wdcor.table). Another idea is to make the distance correlation more robust by assigning small weights to observations which are far from the rest of the data.

For large datasets the distance correlation is often said to be too inefficient to be of any great use. The function approx.dcor offers a pretty good approximation of the distance correlation via binning and wdcor.table).

wdcor.data.frame computes a distance correlation matrix. Factor variables are transformed to integer via data.matrix.

### Usage

```r
wdcor(x,...)
```

#### S3 method:

```r
wdcor(x,y,w = NULL,ep = 1, approx = FALSE, n = 50,na.rm = TRUE, ...)
```

#### S3 method for class 'table'

```r
wdcor(x,ep = 1,...)
```

#### S3 method for class 'data.frame'

```r
wdcor(x, w = NULL, ep = 1, approx = FALSE, n = 50, ...)
```

### Arguments

- **x**: A data crosstable or a numeric vector.
- **y**: A numeric vector.
- **w**: Weights, typically frequencies. The default weights all cases the same, which leads to the standard distance correlation.
- **ep**: The euclidean (absolute) distances can be taken to the power of ep.
- **approx**: Whether or not to use approx.dcor instead of wdcor. This is automatically chosen for vectors with more than 20000 entries.
- **n**: The number of bins used by approx.dcor.
- **na.rm**: Whether or not to remove missing values.
- **...**: dots.

### Value

The correlation value which is between 0 and 1.
Note
 Automatically uses an approximation for vectors larger than 20000 entries!

Author(s)
 Alexander Pilhoefer

References

See Also
 approx.dcor

Examples

# repeat and change N for different results and computation times.
N <- 2000
x1 <- rnorm(N,mean=10,sd=3)
x2 <- runif(N,0,40)
x3 <- rnorm(N,mean=30,sd=4)
x <- sample(c(x1,x2,x3),N)
y <- rnorm(1,sd=0.0001)*(x-mean(x))^4+ rnorm(1,sd=0.01)*(x-mean(x))^3
y <- y+ rnorm(1,sd=0.1)*(x-mean(x))^2
y <- y+ rnorm(1)*(x-mean(x))+rnorm(N,sd=runif(N,3,10))
y <- y+ runif(N,0,20)*sin(abs(scale(x)))*2*pi

require(scales)
plot(x,y,pch=19,col=alpha("black",0.2))
system.time(dd<-wdcor(x,y))

y2 <- runif(2000)
system.time(dde<-wdcor(x,y2))

dd
dde

## Not run:
y <- diamonds$price
x <- diamonds$carat

length(x) # 53940

# auto approximation via approx.dcor
wdcor(x,y)

# the weighted distance correlation is also applicable to
# discrete data:
A <- arsim(2000,c(12,12),4,0.1)
wdcor(A)
wdcor(optile(A))
wdcor(optile(A, fun = "distcor"))

# kernel density weights:
kd <- kde2d(x,y,n=50)

xy <- expand.grid(kd$x,kd$y)
wdcor(xy[,1],xy[,2], w = kd$z)
# this is the approximate distance correlation for the 2D density estimate

## End(Not run)

# a pairwise matrix:
D <- wdcor(olives[,3:10])
fluctile(D*2, shape="c")
Index

*Topic \textasciitilde kwd1
steptile, 87
*Topic \textasciitilde kwd2
steptile, 87
*Topic categorical
eextracat, 27
*Topic datasets
carcustomers, 12
cPScluster, 21
dmc, 27
geneex, 35
olives, 58
plants, 68
USR, 94
*Topic frequency table
untableSet, 93
*Topic interactive
eextracat, 27
*Topic mosaicplots
eextracat, 27
*Topic multiple barcharts
eextracat, 27
*Topic parallel coordinates
eextracat, 27

aes, 29
ahist, 3, 23, 38
approx.dcor, 4, 24, 25, 99, 100
arsim, 6, 28

barysort, 7
BCC, 8
BCI, 9, 14, 15, 20, 55, 70, 71
boxplot2g, 10
Burt, 11, 52, 59, 60
carcustomers, 12
CBCI, 14, 55
cfcl, 15, 20
cfluctile, 15, 16, 20, 44, 45, 58, 86, 87, 92
cmat, 18, 70, 71
cohen, 19
combcl, 20
count, 11
count, 89, 90
cPScluster, 21
cut, 18, 22
cutbw, 3, 22, 38
cutree, 91
dcorMVdata, 23, 25
dcorMVtable, 23, 24, 24
dendro, 25
dist, 23, 24
diverge_hcl, 40, 47
dmc, 27
extracat, 27

facet_grid, 29
facetshade, 28
fluctile, 17, 28, 33, 47, 48, 91, 92, 95, 96
formula, 75
ftable, 75
GeneEx, 35
getbw, 3, 22, 23, 38
getcolors, 40, 83
getIs, 41, 43, 45, 47
getIs2, 42, 42, 43, 47
gpath, 43
ggplot, 29
glm, 76
gsac, 44, 86
hclust, 59, 90, 91
heat_hcl, 40
heat_tile, 41–43, 45, 47
hexbin, 49, 50
hexpie, 28, 49

102
idat, 12, 52, 95, 96
imat, 12, 52, 53
innerval, 50, 53, 75
is.na, 95
itab, 54
JBCI, 15, 55
kendalls, 9, 20, 56, 70, 71, 98
ME, 41–43, 57, 67
mosaicplot, 34, 78

na.omit, 75
nclass.scott, 38

olives, 58
optile, 18, 19, 28, 44, 45, 58, 70, 86–88, 96
optME, 41–43, 57, 67, 96

pairs, 82
plants, 68
polr, 76

qBCI, 18, 19, 70
quantile, 54
quickfechner, 28, 43, 72

rainbow, 40, 47, 76
rainbow_hcl, 40, 47, 76
regmax, 73
regmin (regmax), 73
rmb, 27, 47, 50, 74, 80–83
rmbmat, 28, 80

scpcp, 27, 58, 82, 88
sequential_hcl, 40, 47
setcover, 45, 85
solve_TSP, 41, 42, 67
sortandcut, 18, 45, 86
stat_binhex, 50
steptile, 60, 61, 82, 84, 87
subtable, 89, 95
subtree, 59, 60, 90

terrain_hcl, 40
tfluctile, 18, 91

untatableSet, 93
USR, 94