Package 'shipunov'

January 4, 2021

Type Package

Title Miscellaneous Functions from Alexey Shipunov

Version 1.13

Date 2020-12-20

Author Alexey Shipunov [aut, cre], Paul Murrell [ctb], Marcello D'Orazio [ctb], Stephen Turner [ctb], Eugeny Altshuler [ctb], Roland Rau [ctb], Marcus W Beck [ctb], Sebastian Gibb [ctb], Weiliang Qiu [ctb], Emmanuel Paradis [ctb], Roger Koenker [ctb], R Core Team [ctb]

Maintainer Alexey Shipunov <dactylorhiza@gmail.com>

Description A collection of functions for data manipulation, plotting and statistical computing, to use separately or with the book ``Visual Statistics. Use R!'':
Shipunov (2020) <http://ashipunov.info/shipunov/software/r/r-en.htm>.
Most useful functions:
Bclust(), Jclust() and BootA() which bootstrap hierarchical clustering;
Recode() which does multiple recoding in a fast, simple and flexible way;
Misclass() which outputs confusion matrix even if classes are not concerted;
Overlap() which measures group separation on any projection;
Biarrows() which converts any scatterplot into biplot;
and Pleiad() which is fast and flexible correlogram.

Imports PBSmapping

Suggests apcluster, ape, class, cluster, dbscan, e1071, effsize, grid, ips, kernlab, MASS, mclust, meanShiftR, nnet, phangorn, randomForest, rpart, smirnov, StatMatch, tapkee, tree, vegan

License GPL (>= 2)

LazyLoad yes

LazyData yes

NeedsCompilation no

Repository CRAN

Date/Publication 2021-01-04 12:00:05 UTC

R topics documented:

`Adj.Rand` *Adjusted Rand index*

Description

Adjusted Rand index to compare different clusterings

Usage

```
Adj.Rand(cl1, cl2, ...)
```

Arguments

`cl1`	First classification (character vector of group names)
`cl2`	Second classification
`...`	Further arguments to table()

Details

Use 'useNA="ifany"' or similar option to take NAs as a separate class (for more explanations, see help for table() command).

Note that in rare cases, Adjusted Rand Index might become negative, this might be some evidence that differences between two partitions are "worse than random", i.e., there is a pattern in differences.

Value

Similarity: numerical vector of length 1

Author(s)

Alexey Shipunov

References

Hubert L. and Arabie P. 1985. Comparing partitions. Journal of Classification. 2. 193–218.

See Also

Misclass

Examples

```
iris.dist <- dist(iris[, 1:4], method="manhattan")
iris.hclust <- hclust(iris.dist)
iris.3 <- cutree(iris.hclust, 3)
Adj.Rand(iris.3, iris[, 5])
```

Aggregate1 *Aggregates by one vector and uses it for row names*

Description

Aggregates by one vector and uses it for row names

Usage

```
Aggregate1(df, by, ...)
```

Arguments

df	Data frame to aggregate
by	Atomic object to use for aggregating
...	Further arguments for 'aggregate()'

Details

'Aggregate1()' is an 'aggregate()' helper: aggregates only by one atomic variable and uses it for row names.

Value

Same as of 'aggregate()'

Author(s)

Alexey Shipunov

See Also

aggregate

Examples

```
trees3 <- sample(letters[1:3], nrow(trees), replace=TRUE)
Aggregate1(trees, trees3, median, na.rm=TRUE)
```

Alldups *Finds all duplicates*

Description

Finds duplicates from both ends, optionally returns indexes of duplicate groups

Usage

```
Alldups(v, groups=FALSE)
```

Arguments

v	Vector, matrix or data frame
groups	If TRUE, returns group indexes (non-duplicated are 0)

Details

This is extension of duplicated() which _does not_ skip the first duplicate in each group. 'NA' consider for duplicates but do not count as duplicate group.

If the first argument is a matrix or data frame and 'groups=TRUE', Aldups() starts from converting them into character vector with paste0(..., collapse="").

If 'groups=TRUE', Alldups() uses as.numeric(as.character(v)) twice to index duplicated groups with natural numbers (and non-duplicated with 0).

Value

Logical vector of length equal to 'v', or numerical vector if 'groups=TRUE'

Author(s)

Alexey Shipunov

See Also

duplicated

Examples

```
aa <- c("one", "two", "", NA, "two", "three", "three", "three", NA, "", "four")
Alldups(aa)
data.frame(v=aa, dups=Alldups(aa), groups=Alldups(aa, groups=TRUE))

## clustering based on duplicates from rounding
(iris.dgr <- Alldups(round(iris[, 1:4]/10), groups=TRUE))
Misclass(iris.dgr, iris$Species, best=TRUE)
```

---

atmospheres *atmospheres*

Description

Atmospheres of Solar System.

Mercury might be easily taken out because it does not have atmosphere in strict sense.

All data in percentages.

Usage

```
atmospheres
```

Source

Data from the NASA Web site, once was available as 'planetatmoscomp.pdf' document.

Bclabels *Plot bootstrap values*

Description

Print (bootstrap) values on 'hclust' plot

Usage

```
Bclabels(hcl, values, coords=NULL, horiz=FALSE, method="text",
 threshold=NULL, top=NULL, percent=FALSE, ...)
```

Arguments

hcl	hclust object
values	numeric, (bootstrap) values to use
coords	If NULL (default), coordinates will be calculated with Hcoords(hcl)
horiz	Plot values for a horizontal tree?
method	If "text" (default), plot text values, if "points", plot points
threshold	If set, do not plot text or points for values < threshold; respects percents if set
top	If set as 'n', plot values only for 'n' highest clusters
percent	Plot values as percents?
...	If "text" (default), additional arguments to text(), if "points", to points()

Details

This low-level plot function plots text or points in accordance with bootstrap values to the corresponding node of the plotted 'hclust' object.

Value

List with components: 'coords' for coordinates, 'labels' for (selected) values.

See Also

Bclust

Examples

```
## 'atmospheres' data
(bb <- Bclust(t(atmospheres))) # specify 'mc.cores=4' or similar to speed up the process

## standard use
plot(bb$hclust)
Bclabels(bb$hclust, bb$values, col="blue", pos=3, offset=0.1, threshold=0.9)

## 'points' method
plot(bb$hclust)
Bclabels(bb$hclust, bb$values, method="points", threshold=0.9, pch=19, cex=2)

## 'points' which grow with support
plot(bb$hclust)
Bclabels(bb$hclust, bb$values, method="points", pch=19, cex=bb$values*3)

## pre-defined coordinates
coords1 <- Hcoords(bb$hclust)
plot(bb$hclust)
Bclabels(bb$hclust, bb$values, coords=coords1, method="points", pch=19,
 cex=bb$values*3)
```

```
## use with horizontal Ploth()
oldpar <- par(mar=c(2,1,0,4))
Ploth(bb$hclust, horiz=TRUE)
Bclabels(bb$hclust, bb$values, col="blue", pos=3, offset=0.1, horiz=TRUE)
par(oldpar)

## 'moldino' data
m.bb <- Bclust(t(moldino)) # specify 'mc.cores=4' or similar to speed up the process
plot(m.bb$hclust)
Bclabels(m.bb$hclust, m.bb$values, col="red", pos=3, offset=0.1, threshold=0.5)

## 'iris' data, with hyper-binding to make number of variables reliable
iris.bb <- Bclust(iris[, rep(1:4, 6)], iter=100) # remove iter=100 for better bootstrap
plot(iris.bb$hclust, labels=FALSE, main="", xlab="", sub="Bootstrap, 100 replicates")
## use 'percent' and 'top'
Bclabels(iris.bb$hclust, iris.bb$values, top=5, percent=TRUE, pos=3, offset=0.1)
Fence(iris.bb$hclust, iris$Species)
legend("topright", legend=levels(iris$Species), col=1:3, lwd=2.5, bty="n")
```

`Bclust` *Bootstrapped hclust*

Description

Bootstraps (or jacknifes) hierarchical clustering

Usage

```
Bclust(data, method.d="euclidean", method.c="ward.D", FUN=function(.x)
 hclust(dist(.x, method=method.d), method=method.c),iter=1000,
 mc.cores=1, monitor=TRUE, bootstrap=TRUE, relative=FALSE, hclist=NULL)

## S3 method for class 'Bclust'
plot(x, main="", xlab=NULL, ...)
```

Arguments

`data`	Data suitable for the chosen distance method
`method.d`	Method for dist()
`method.c`	Method for hclust()
`FUN`	Function to make 'hclust' objects
`iter`	Number of replicates
`mc.cores`	`integer`, number of processes to run in parallel
`monitor`	If TRUE (default), prints a dot for each replicate

bootstrap	If FALSE (not default), performs jacknife (and makes 'iter=ncol(data)')
relative	If TRUE (not default), use the relative matching of branches (see in Details)
hclist	Allows to supply the list of 'hclust' objects
x	Object of the class 'Bclust'
main	Plot title
xlab	Horizontal axis label
...	Additional arguments to the plot.hclust()

Details

This function provides bootstrapping for hierarchical clustering (hclust objects). Internally, it uses Hcl2mat() which converts 'hclust' objects into binary matrix of cluster memberships.

The default clustering method is the variance-minimizing "ward.D" (which works better with Euclidean distances); to make it coherent with hclust() default, specify 'method.c="complete"'. Also, it sometimes makes sense to transform non-Euclidean distances into Euclidean with 'dist(_non_euclidean_dist_)'.

Bclust() and companion functions were based on functions from the 'bootstrap' package of Sebastian Gibb.

Option 'hclist' presents the special case when list of 'hclust' objects is pre-build. In that case, other arguments (except 'mc.cores' and 'monitor') will be ignored, and the first component of 'hclist', that is 'hclist[[1]]', will be used as "original" clustering to compare with all other objects in the 'hclist'. Number of replicates is the length of 'hclist' minus one.

Option 'relative' changes the mechanism of how branches of reference clustering ("original") and bootstrapped clustering ("current") compared. If 'relative=FALSE' (default), only absolute matches (present or absent) are count, and vector of matches is binary (either 0 or 1). If 'relative=TRUE', branches of "original" which have no matches in "current", are checked additionally for the similarity with all branches of "current", and the minimal (asymmetric) binary dissimilarity value is used as a match. Therefore, the matching vector in this case is numeric instead of binary. This will typically result in the reliable raising of bootstrap values. The underlying methodology is similar to what is defined in Lemoine et al. (2018) as a "transfer bootstrap". As the asymmetric binary is the _proportion_ of items in which only one is "1" amongst those which have one or two "1", it is possible to rephrase Lemoine et al. (2018), and say that this distance is equal to the _proportion_ of items that must be _removed_ to make both branches identical. Please note that with 'relative=TRUE', the whole algorithm is several times slower then default.

Please note that Bclust() frequently underestimates the cluster stability when number of characters is relatively small. One of possible remedies is to use hyper-binding (like "cbind(data, data, data)") to reach the reliable number of characters.

plot.Bclust() designed for quick plotting and plots labels (bootstrap support values) with the following defaults: 'percent=TRUE, pos=3, offset=0.1'. To change how labels are plotted, use separate Bclabels() command.

Value

Returns object of class 'Bclust' which is a list with components: 'values' for bootstrapped frequencies of each node, 'hcl' for original 'hclust' object, 'consensus' which is a sum of all Hcl2mat() matrices, 'meth' (bootstrap or jacknife), and 'iter', for number of iterations.

References

Felsenstein J. 1985. Confidence limits on phylogenies: an approach using the bootstrap. Evolution. 39 (4): 783–791.

Efron B., Halloran E., Holmes S. 1996. Bootstrap confidence levels for phylogenetic trees. Proceedings of the National Academy of Sciences. 93 (23): 13429–13429.

Lemoine F. et al. 2018. Renewing Felsenstein's phylogenetic bootstrap in the era of big data. Nature, 556(7702): 452–456

See Also

Jclust, BootA, Hcl2mat, Bclabels, Hcoords

Examples

```
data <- t(atmospheres)

## standard use
(bb <- Bclust(data)) # specify 'mc.cores=4' or similar to speed up the process
plot(bb)

## more advanced plotting with Bclabels()
plot(bb$hclust)
Bclabels(bb$hclust, bb$values, threshold=0.5, col="grey", pos=1)

## how to use the consensus data
plot(hclust(dist(bb$consensus)), main="Net consensus tree") # net consensus
## majority rule is 'consensus >= 0.5', strict is like 'round(consensus) == 1'

## how to make user-defined function
bb1 <- Bclust(t(atmospheres), FUN=function(.x) hclust(Gower.dist(.x)))
plot(bb1)

## how to jacknife
bb2 <- Bclust(data, bootstrap=FALSE, monitor=FALSE)
plot(bb2)

## how to make (and use) the pre-build list of clusterings
hclist <- vector("list", length=0)
hclist[[1]] <- hclust(dist(data)) # "orig" is the first
for (n in 2:101) hclist[[n]] <- hclust(dist(data[, sample.int(ncol(data), replace=TRUE)]))
(bb3 <- Bclust(hclist=hclist))
plot(bb3)

## how to use the relative matching
bb4 <- Bclust(data, relative=TRUE)
plot(bb4)

## how to hyper-bind
bb5 <- Bclust(cbind(data, data, data)) # now data has 24 characters
plot(bb5)
```

```
## how to use hclust() defaults
bb6 <- Bclust(data, method.c="complete")
plot(bb6)
```

`BestOverlap` *Calculates the best overlap*

Description

Uses multiple datasets, measures overlaps between class-related convex hulls and reports the best dataset, the best overlap table and summary with confidence intervals. Can be used to assess bootstrap or jackknife results, to compare different dimension reduction and/or clustering methods, and to average results of stochastic methods.

Usage

```
BestOverlap(xylabels, ci="95%", round=4)
```

Arguments

`xylabels`	List of data frames, each with at least 3 columns named exactly as: "x" for x coordinates, "y" for y coordinates and "labels" for class labels
`ci`	Confidence interval (character string with percent sign)
`round`	How to round numbers in summary table

Details

'BestOverlap()' requires object, typically created after bootstrapping or similar procedure (see below for examples). This 'xylabels' object must contain at least three columns named exactly as c("x", "y", "labels"), in any order.

Please note that label types must be the same between data frames inside 'xylabels' list. For consistency, first data frame is used as a label standard. If any next data frame contain label types different from standard, it will be ignored.

Value

List with three components: 'best' data frame, 'best.overlap' table and 'summary' data frame.

Author(s)

Alexey Shipunov

Examples

```
## Bootstrap PCA
B <- 100
xylabels <- vector("list", length=0)
for (n in 1:B) {
ROWS <- sample(nrow(iris), replace=TRUE)
tmp <- prcomp(iris[ROWS, -5])$x[, 1:2]
xylabels[[n]] <- data.frame(x=tmp[, 1], y=tmp[, 2], labels=iris[ROWS, 5])
}
BestOverlap(xylabels)

## Jacknife PCA
B <- nrow(iris)
xylabels <- vector("list", length=0)
for (n in 1:B) {
ROWS <- (1:B)[-n]
tmp <- prcomp(iris[ROWS, -5])$x[, 1:2]
xylabels[[n]] <- data.frame(x=tmp[, 1], y=tmp[, 2], labels=iris[ROWS, 5])
}
BestOverlap(xylabels)

## Stochastic method: Stochastic Proximity Embedding
library(tapkee)
B <- 100
xylabels <- vector("list", length=0)
for (n in 1:B) {
tmp <- Tapkee(iris[, -5], method="spe")
xylabels[[n]] <- data.frame(x=tmp[, 1], y=tmp[, 2], labels=iris[, 5])
}
BestOverlap(xylabels)

## Diverse dimension reduction methods
library(tapkee)
B <- c("lle", "npe", "ltsa", "lltsa", "hlle", "la", "lpp", "dm", "isomap", "l-isomap")
xylabels <- vector("list", length=0)
for (n in B) {
tmp <- Tapkee(iris[, -5], method=n, add="-k 50")
xylabels[[n]] <- data.frame(x=tmp[, 1], y=tmp[, 2], labels=iris[, 5])
}
BestOverlap(xylabels)

## One dimension reduction but many clusterings
B <- 100
xylabels <- vector("list", length=0)
tmp1 <- prcomp(iris[, -5])$x[, 1:2]
for (n in 1:B) {
tmp2 <- kmeans(iris[, -5], centers=3)$cluster
xylabels[[n]] <- data.frame(x=tmp1[, 1], y=tmp1[, 2], labels=letters[tmp2])
}
BestOverlap(xylabels)
```

`Biarrows` *Adds correlation arrows to the scatterplot*

Description

Plots 'orig' variables as arrows on the 'deriv' variables 2D scatterplot

Usage

```
Biarrows(deriv, orig, coeffs=NULL, shrink=0.45, closer=0.9,
 pt.col="forestgreen", pt.cex=1, pt.pch=NA, tx=colnames(orig),
 tx.col="forestgreen", tx.cex=0.8, tx.font=1, tx.pos=NULL, tx.off=0.5, xpd=TRUE,
 ar.col="forestgreen", ar.len=0.05, shift="auto", ...)
```

Arguments

`deriv`	Data derived from, e.g., dimension reducion of 'orig'
`orig`	Original data
`coeffs`	(Optional) two-column matrix with proposed coordinates of arrow tips, row names must represent 'orig' variables
`shrink`	How to shrink arrows in relation to 'deriv' ranges, default is 45% (0.45)
`closer`	How closer to the center (in relation to the text label) is the arrow tip, default is 0.9
`pt.col`	Color of points, default is "forestgreen"
`pt.cex`	Size of points, default is 1
`pt.pch`	Type of points, default is NA (no points)
`tx`	Text labels, default are 'colnames(orig)'
`tx.col`	Color of text labels, default is "forestgreen"
`tx.cex`	Size of text, default is 0.8
`tx.font`	Font of text, default is 1 (plain)
`tx.pos`	Position of text, default is NULL (in the center)
`tx.off`	Offest for text labels, default 0.5 (works only if 'tx.pos' is not NULL)
`xpd`	Allow text to go outside of plotting region?
`ar.col`	Color of arrows, default is "forestgreen"
`ar.len`	Length of the edges of the arrow head (in inches)
`shift`	Shift from the center which is c(0, 0); default is "auto" which is colMeans(deriv)
`...`	Further arguments to arrows()

Details

Biarrows() calculates correlations between two sets of variables which generally belong to the same data: more then one 'orig' variables and exactly two 'deriv' variables. These correlations might be understood as importances of the 'orig' variables. Then Biarrows() scales correlations to the 'deriv' ranges and adds text labels and arrows (possibly also points) to the scatterplot of derived variables. These arrows represent the original variables in relation with derived variables. Resulted plot may be seen as a biplot which simultaneously shows two sets of variables. In fact, it is possible to show three and more sets of variables (see examples).

This approach might work for data derived from (almost) any kind of dimensional reduction. Biarrows() is also much more flexible than standard biplot(). Please note, however, that Biarrows() is only visualization, and numerical conclustions might not be justified.

If 'deriv' data contains more then 2 variables, the rest will be discarded. Both 'deriv' and 'orig' should be either data frames or matrices with column names and compatible dimensions, possibly with NAs.

Biarrows(dr, coeffs=...) allows to use pre-calculated coefficients. In that case, 'data' is ignored (except for column names, but they might be supplied separately as 'tx' value), and 'coeffs' will be scaled. See examples to understand better how it works.

To suppress arrows or text, use zero color. Points are suppressed by default.

Author(s)

Alexey Shipunov

See Also

biplot

Examples

```
iris.cmd <- cmdscale(dist(iris[, -5]))
plot(iris.cmd, xlab="Dim 1", ylab="Dim 2")
Biarrows(iris.cmd, iris[, -5])
title(main="MDS biplot with Biarrows()")

## ===

library(MASS)
iris.mds <- isoMDS(dist(unique(iris[, -5])))
plot(iris.mds$points, xlab="Dim 1", ylab="Dim 2")
Biarrows(iris.mds$points, unique(iris[, -5]))
title(main="Non-metric MDS biplot with Biarrows()")

## ===

library(MASS)
iris.smm <- sammon(dist(unique(iris[, -5])))
plot(iris.smm$points, xlab="Dim 1", ylab="Dim 2")
Biarrows(iris.smm$points, unique(iris[, -5]))
```

```
title(main="Sammon mapping biplot with Biarrows()")

## ===

iris.p <- prcomp(iris[, -5], scale=TRUE)
biplot(iris.p, xpd=TRUE, main="Original PCA biplot")
plot(iris.p$x)
Biarrows(iris.p$x, iris[, -5])
title(main="PCA biplot with Biarrows()")

## ===

plot(iris.p$x, xlab="PCA1", ylab="PCA2")
## how to use 'coeffs'
## they also useful as surrogates of variable importances
(coeffs <- cor(iris[, -5], iris.p$x, method="spearman"))
Biarrows(iris.p$x, tx=rownames(coeffs), coeffs=coeffs)

## ===

plot(iris[, c(1, 3)])
Biarrows(iris[, c(1, 3)], iris.p$x)
title(main="\"Reversed biplot\"")

## ===

plot(iris[, c(1, 3)])
Biarrows(iris[, c(1, 3)], iris[, c(2, 4)])
title(main="Iris flowers: lengths vs. widths")

## ===

plot(iris.p$x)
Biarrows(iris.p$x[, 1:2], iris.p$x[, 1:2])
title(main="\"Self-biplot\" on PCA")

## ===

library(MASS)
iris.ldap <- predict(lda(Species ~ ., data=iris), iris[, -5])
plot(iris.ldap$x)
Biarrows(iris.ldap$x, iris[, -5])
Biarrows(iris.ldap$x, iris.p$x[, 1:2], shift=c(9, 2.5),
 shrink=0.95, lty=2, ar.col="darkgrey", tx.col="darkgrey")
title(main="Triplot: LDA, original variables and PCA axes")

## ===

iris.cl <- Classproj(iris[, -5], iris$Species)
plot(iris.cl$proj, col=iris$Species)
Biarrows(iris.cl$proj, iris[, -5])
title(main="Classproj biplot")
```

Biokey *Convert diagnostic keys and classification lists*

Description

Convert the oldest biological data structures: diagnostic keys ("keys") and classification lists ("classifs")

Usage

```
Biokey(data, from="", to="", recalculate=TRUE, internal=FALSE, force=FALSE)
Numranks(nums=NULL, ranks=NULL, add=NULL, empty="Species")
```

Arguments

data	Diagnostic keys ("keys"), classification lists ("classifs") and tables, or Newick phylogeny trees
from	Data type to convert from
to	Data type to convert to
recalculate	Recalculate the numeric ids?
internal	(For debugging) Output internal 4-column 'key' objects instead?
force	(For debugging) Ignore list of allowable conversion pairs?
nums	Numbers to convert into ranks
ranks	Ranks to convert into numbers
add	Rank-number conversion rule to add (overrides embedded rules)
empty	What rank to use for empty number?

Details

Biokey() is a way to convert classification lists ("classifs") or diagnostic keys into each other. In addition, it handles species classification tables ("table") and Newick trees ("newick").

To know which conversions are allowed, simply type Biokey() without arguments (this will also induce the harmless error message).

Numranks() converts biological rank names into numbers and numbers into rank names (Shipunov, 2017). To see the embedded conversion table, type Numranks() without arguments.

To know more about keys and classifs, read help for "classifs" and "keys".

Bracket keys (see help for "keys") could have more than two conditions, other keys not, so problems might arise during conversion (see examples).

Backreferenced keys (see help for "keys") is just a variety of bracket keys so the only possible way to make them is from bracket keys.

Branched key (see help for "keys") is an indented key with omitted "indent" column, therefore it does not require the separate conversion way. See examples about how to convert indent column into actual indents.

Classification "table" is the data frame where each column represent some particular rank (see examples to understand better). Similarly to "classif", "table" should use numerical ranks. In this case, numerical ranks should be column names (see examples).

When Biokey() converts "classif" to "newick", it keeps higher group names as node labels. It does not do that in all other cases.

It is an open question if phylogeny tree (Newick) should be converted into "classif" (see help for "classifs") with all intermediate ranks propagated (thus frequently become monotypic, i.e. with just one subgroup), or with only main ranks (whole numbers) propagated, or terminals (by default, they always have "species" rank = 1) could follow much bigger ranks (i.e., "species" = 1 might follow "family" = 3, not "genus" = 2). At the moment, the last variant is implemented.

Comparably, "newick" to "classif" conversion does _not_ remove names of monotypic intermediate taxa, this might result in "crowding" of node labels (see the example). Also, this conversion automatically propagates intermediate ranks to make all ranks concerted, this might result in empty labels.

Value

Typically, the data frame or just a character string (in case of Newick output). Output may contain column names but this is only to facilitate understanding of the format and could be stripped without consequences. If 'internal=TRUE', outputs a standardized 4-column data frame in a form of branched key (columns 'id', 'description', 'terminal'), plus 'goto' column which might be just NAs.

Author(s)

Alexey Shipunov

References

Shipunov A. 2017. "Numerical ranks" to improve biological nomenclature of higher groups. See "https://arxiv.org/abs/1708.07260".

See Also

classifs, keys

Examples

```
## Biokey() # makes (harmless) error message but also shows which conversions are available
Numranks() # shows the conversion table

## ===

Numranks(nums=1:7)
Numranks(ranks="kingdom") # "kingdom", "order", "family" and "tribe" translate into Latin

## ===

## three branched keys
i1 <- c("1 A ", "2 B Name1", "2 BB Name2", "1 AA ", "3 C Name3", "3 CC Name4")
```

```
i2 <- c("1 A Name1", "2 B Name2", "2 BB ", "3 C Name3", "3 CC Name4")
i3 <- c("1 A Name1", "2 B Name2", "2 BB ", "3 C Name3",
 "3 CC Name4", "2 BBB ", "4 D Name5", "4 DD Name6", "4 DDD Name7")
k1 <- read.table(textConnection(i1), sep=" ", as.is=TRUE)
k2 <- read.table(textConnection(i2), sep=" ", as.is=TRUE)
k3 <- read.table(textConnection(i3), sep=" ", as.is=TRUE)

## convert them into phylogeny trees and plot
t1 <- Biokey(k1, from="branched", to="newick")
t2 <- Biokey(k2, from="branched", to="newick")
t3 <- Biokey(k3, from="branched", to="newick")
library(ape) # load 'ape' to plot Newick trees below
plot(read.tree(text=t1))
plot(read.tree(text=t2))
plot(read.tree(text=t3))

## ===

## Bracket keys
bracket1 <- keys[[1]]
bracket1
Biokey(bracket1, from="bracket", to="backreferenced")
(ii <- Biokey(bracket1, from="bracket", to="indented"))
## Remove third condition to avoid warnings:
Biokey(bracket1[bracket1[, 3] != "Horse", ], from="bracket", to="serial")
(nn <- Biokey(bracket1, from="bracket", to="newick"))
plot.phylo(read.tree(text=nn)) # plot newick as phylogeny trees

## Now convert indent column into actual indents:
for (i in 1:length(ii[, 1])) ii[i, 1] <- paste(rep(" ", ii[i, 1]), collapse="")
## and make also dot leaders
ifelse(!is.na(ii[, 3]), "...", "")
ii

## Branched keys
branched1 <- keys[[3]]
head(branched1)
Biokey(branched1, from="branched", to="bracket")[1:7, ]
Biokey(branched1, from="branched", to="indented")[1:7, ]
Biokey(branched1, from="branched", to="serial")[1:7, ]
(nn <- Biokey(branched1, from="branched", to="newick"))
plot.phylo(read.tree(text=nn))

## Indented keys (same as branched but with indent as first column)
indented0 <- c("0 1 Blue ", "1 2 Gas Sky", "1 2 Liquid ",
 "0 1 Yellow ", "2 3 Star Sun", "2 3 Buttecup Flower")
(indented1 <- read.table(textConnection(indented0), sep=" ", as.is=TRUE))
Biokey(indented1, from="indented", to="bracket")
Biokey(indented1, from="indented", to="serial")
(nn <- Biokey(indented1, from="indented", to="newick"))
plot.phylo(read.tree(text=nn))

## Serial keys
```

```
serial1 <- keys[[4]]
head(serial1)
Biokey(serial1, from="serial", to="bracket")[1:7, ]
Biokey(serial1, from="serial", to="indented")[1:7, ]
(nn <- Biokey(serial1, from="serial", to="newick"))
plot.phylo(read.tree(text=nn))

## Classifs
classif2 <- classifs[[2]]
classif2[, 1] <- Numranks(ranks=classif2[, 1], add=c(Series=1.1))
head(classif2)
Biokey(classif2, from="classif", to="table")[1:7, ]
(nn <- Biokey(classif2, from="classif", to="newick"))
tt <- read.tree(text=nn)
plot.phylo(tt, node.depth=2)
nodelabels(tt$node.label, frame="none", bg="transparent", adj=-0.05)

## Classification tables
table0 <- c("FAMILY SUBFAMILY TRIBE GENUS", "Hominidae Homininae Hominini Homo",
 "Hominidae Homininae Hominini Pan", "Hominidae Homininae Gorillini Gorilla",
 "Hominidae Ponginae Ponginini Pongo")
(table1 <- read.table(textConnection(table0), sep=" ", as.is=TRUE, h=TRUE))
names(table1) <- Numranks(ranks=names(table1))
table1
Biokey(table1, from="table", to="classif")

## Newick phylogeny trees
newick1 <- "((Coronopus,Plantago),(Bougueria,(Psyllium_s.str.,Albicans)),Littorella);"
plot.phylo(read.tree(text=newick1))
Biokey(newick1, from="newick", to="classif")
```

`BootA` *Bootstrap clustering*

Description

How to bootstrap clustering with 'ape'

Usage

```
BootA(dat, FUN=function(.x) ape::nj(dist(.x)), iter=1000, mc.cores=1, tresh=50,
 cons=TRUE, prop=0.5)
```

Arguments

`dat`	data
`FUN`	how to bootstrap (see examples)
`iter`	number of iterations, default 1000

mc.cores	how many cores to employ (system-dependent)
tresh	Threshold for printing bootstrap values
cons	Calculate consensus tree?
prop	0.5 is majority-rule consensus (default), 1 is strict consensus

Details

This is how to bootstrap clustering with 'ape::boot.phylo()'.

Author(s)

Alexey Shipunov

See Also

Bclust, BootA, ape::boot.phylo

Examples

```
dat <- iris[, -5]
row.names(dat) <- abbreviate(make.names(iris[, 5], unique=TRUE))
iris.BA1 <- BootA(dat, iter=100)
plot(iris.BA1$boot.tree, show.node.label=TRUE)
plot(iris.BA1$cons.tree)
iris.BA2 <- BootA(dat, FUN=function(.x) ape::as.phylo(hclust(dist(.x))), iter=100)
## Not run:
## change (or remove) 'mc.cores=...' in accordance with your system features
iris.BA3 <- BootA(dat, FUN=function(.x) phangorn::NJ(dist(.x)), iter=100,
 mc.cores=4)

## End(Not run)
```

BootKNN *Bootstrap with kNN*

Description

How to bootstrap with kNN (and DNN)

Usage

```
BootKNN(data, classes, sub="none", nsam=4, nboot=1000, misclass=TRUE, method="knn", ...)
```

Arguments

data	Data frame to classify
classes	Character vector of class names
sub	Subsample to use (see example)
nsam	Number of training items from each level of grouping factor, default 4
nboot	Number of iterations
misclass	Calculate misclassification table?
method	Either "knn" (class::knn()) or "dnn" (shipunov::Dnn())
...	Further arguments to method functions

Details

This function samples equal numbers ('nsam') of training items from *each level* of grouping factor.

It also allows to use *sub*set of data which will be used for sub-sampling of training data.

Value

Returns all predictions as character matrix, each boot is a column

Author(s)

Alexey Shipunov

See Also

class::knn, Dnn

Examples

```
iris.sub <- 1:nrow(iris) %in% seq(1, nrow(iris), 5)
iris.bootknn <- BootKNN(iris[, -5], iris[, 5], sub=iris.sub)
## calculate and plot stability
st <- apply(iris.bootknn, 1, function(.x) var(as.numeric(as.factor(.x))))
plot(prcomp(iris[, -5])$x, col=iris$Species, pch=ifelse(st == 0, 19, 1))
## boot Dnn
BootKNN(iris[, -5], iris[, 5], nboot=50, method="dnn",
 k=1, FUN=function(.x) Gower.dist(.x))
```

BootRF *Bootstrap with 'randomForest()'*

Description

How to bootstrap with 'randomForest()'

Usage

```
BootRF(data, classes, sub="none", nsam=4, nboot=1000, misclass=TRUE, ...)
```

Arguments

data	Data frame to classify
classes	Character vector of class names
sub	Subsample to use (see example)
nsam	Number of training items from each level of grouping factor, default 4
nboot	Number of iterations
misclass	Calculate misclassification table?
...	Further options to randomForest()

Details

Note that as randomForest::randomForest() is based on sampling, BootRF() is the kind of second-level bootstrap.

BootRF() is very simple and does not interact with Random Forest algorithms. It is stratified, i.e. samples equal numbers ('nsam') of training items from the *each level* of grouping factor.

Also, it allows to use the *sub*set of data which will be in turn used for sub-sampling of training data.

Value

Returns all predictions as character matrix, each boot is a column

Author(s)

Alexey Shipunov

See Also

randomForest::randomForest

Examples

```
iris.sub <- 1:nrow(iris) %in% seq(1, nrow(iris), 5)

## could be slow
iris.bootrf <- BootRF(iris[, -5], iris[, 5], sub=iris.sub)
iris.bootrf <- BootRF(iris[, -5], iris[, 5]) # naturally, lower
## calculate and plot stability
st <- apply(iris.bootrf, 1, function(.x) var(as.numeric(as.factor(.x))))
plot(prcomp(iris[, -5])$x, col=iris$Species, pch=ifelse(st == 0, 19, 1))
```

Boxplots *Grouped boxplots*

Description

Boxplots for every scaled variable grouped by factor

Usage

```
Boxplots(vars, groups, boxcols=Pastels, legpos="topleft", srt=45, adj=1,
 slty=3, yticks=FALSE, ymarks=FALSE, ...)
```

Arguments

`vars`	data frame consists of variables to plot
`groups`	grouping factor
`boxcols`	colors of character boxes, default is 'Pastels', i.e. c("white", "lightblue", "mistyrose", "lightcyan", "lavender", "cornsilk")
`legpos`	where to place automatic legend, default is 'topleft', for no legend use 'legpos=NA'
`slty`	line type to delimit groups of boxes
`srt, adj, yticks, ymarks`	regular 'plot()' arguments
`...`	additional arguments to 'boxplot()'

Details

There are many ways to represent groups in data. One is trellis plots. 'Boxplots()' make grouped plots which fit the plot box linearly and therefore easy to compare. So the main idea for grouped plots is to make comparison easier.

Please note that because characters within group are likely of different nature, they are scaled. Consequently, tick marks are removed as they have no sense.

Alternatives: trellis designs.

Value

For the efficiency reasons, the function does not return anything.

Author(s)

Alexey Shipunov

See Also

boxplot, Linechart, Dotchart3

Examples

```
Trees <- trees
Trees[, 4] <- sample(letters[1:3], nrow(Trees), replace=TRUE)
Boxplots(Trees[, 1:3], factor(Trees[, 4]), srt=0, adj=c(.5, 1)) # horizontal labels

sp <- Recode(eq_s$N.POP, eq_l$N.POP, eq_l$SPECIES)
eq <- cbind(sp=as.factor(sp), eq_s[, -1])
eq3 <- eq[eq$sp %in% levels(eq$sp)[1:3], ]
Boxplots(eq3[, 2:9], eq3[, 1], boxcols=grey(1:3/3), slty=0) # no border lines
```

Cdate *System date, time plus easy save history*

Description

System date in 'yyyymmdd' format, system time in 'yyyymmdd_hhmmss' format plus easy save history

Usage

```
Cdate()
Ctime()
Save.history()
```

Details

System date / time in compact formats. These formats are by experience, the most appropriate formats both for file systems and for spreadsheets.

There is also easy 'savehistory' (does not work under macOS R GUI – but works under macOS 'Terminal.app' R).

Author(s)

Alexey Shipunov

See Also

savehistory

Examples

```
Cdate()
Ctime()
## Not run:
## does not work under macOS GUI
Save.history()

## End(Not run)
```

---

chaetocnema *Chaetocnema flea beetles*

Description

Lubischew data (1962, pp. 465–468, tables 4–6): 74 Chaetocnema flea beetles specimens which belong to three cryptic species.

Sources of specimens:

Chaetocnema concinna Marsh:

1-6 Environs of Uljianovsk; 7 Khvalynsk, the Volga; 8-9 Perm; 10-14 Environs of Leningrad; 15-17 The Ukraine; 18 Ashkhabad, Turkmenistan; 19-21 France.

Ch. heikertintgeri Lubis.:

1-8 Environs of Uljianovsk; 9 Khvalynsk; 10-14 Perm; 15-17 Environs of Leningrad; 18-20 The Ukraine; 21 Ustj-Zilma; 22 Gagra, Abkhazia; 23-27 Ussuri district; 28-29 Yakutsk district; 30 Khabarovsk; 31 Germany.

Ch. heptapotamnica Lubis.:

1-18 Environs of Lake Issyk-Kul, Kirghizia; 19 Alma-ata, Kazachstan; 20-22 Environs of Frunze, Kirghizia.

Usage

```
chaetocnema
```

Format

These data frame contains the following columns:

`Species` Species epithet

`No` Number of sample (see below)

`x10` Width of the first joint of the first tarsus (the sum of measurements for both tarsi), in microns

`x12` The same for the second joint

`x14` The maximal width of the aedeagus in the fore-part, in microns

`x18` The front angle of the aedeagus, 1 unit = 7.5 degrees

`x40` The maximal width of the head between the external edges of the eyes, in 0.01 mm

`x48` The aedeagus width from the side, in microns

Source

Lubischew A.A. 1962. On the use of discriminant functions in taxonomy. Biometrics. 18:455–477.

`Cladd` *Adds confidence bands to the simple linear model plots*

Description

Adds confidence bands to the simple linear model plots

Usage

```
Cladd(model, data, level=.95, lty=2, ab.lty=0, col="black", ab.col="black")
```

Arguments

`model`	Simple linear model name
`data`	Original data
`level`	Confidence level
`lty`	Confidence bands line type
`ab.lty`	Regression line type
`col`	Confidence bands line color
`ab.col`	Regression line color

Details

'Cladd()' adds confidence bands to the simple linear model plots. Works only for simple lm(y ~ x) objects!

Author(s)

Alexey Shipunov

See Also

`lm`

Examples

```
hg.lm <- lm(Height ~ Girth, data=trees)
plot(Height ~ Girth, data=trees)
Cladd(hg.lm, data=trees, ab.lty=1)
```

`Class.sample` *Samples along the class labels*

Description

Stratified sampling: sample separately within each class

Usage

```
Class.sample(lbls, nsam=NULL, prop=NULL, uniform=FALSE)
```

Arguments

`lbls`	Vector of labels convertable into factor
`nsam`	Number of samples to take from each class
`prop`	Proportion of samples to take from each class
`uniform`	Uniform instead of random?

Details

'Class.sample()' splits labels into groups in accordance with classes, and samples each of them separately. If 'prop' is specified, then number of samples in each class calculated separately from this value. Of both 'nsam' and 'prop' specified, preference is given to 'prop'.

Uniform method samples each n-th member of the class to reach the desired sample size.

If sample size is bigger then class size, the whole class will be sampled.

Class.sample() uses the ave() internally, and can be easily extended, for example, to make k-fold sampling, like:

ave(seq_along(lbls), lbls, FUN=function(.x) cut(sample(length(.x)), breaks=k, labels=FALSE))

Value

Logical vector of length equal to 'vector'

Author(s)

Alexey Shipunov

Examples

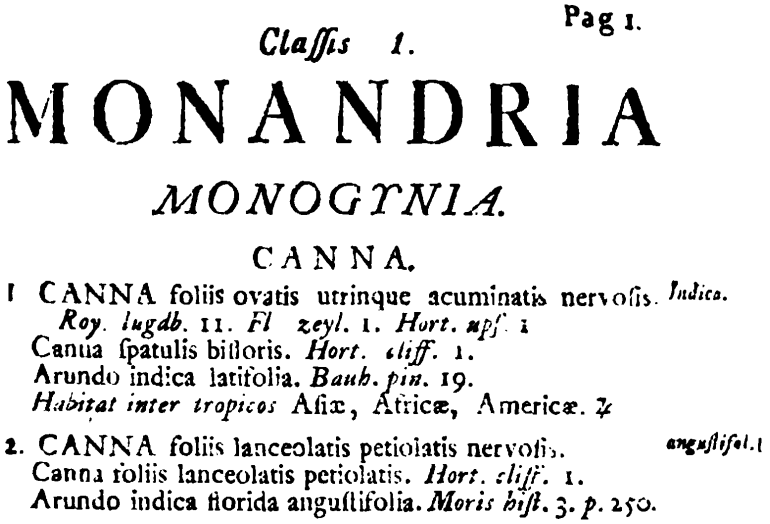
```
(sam <- Class.sample(iris$Species, nsam=5))
iris.trn <- iris[sam, ]
iris.tst <- iris[!sam, ]

(sample1 <- Class.sample(iris$Species, nsam=10))
table(iris$Species, sample1)
```

```
(sample2 <- Class.sample(iris$Species, prop=0.2))
table(iris$Species, sample2)
(sample3 <- Class.sample(iris$Species, nsam=10, uniform=TRUE))
table(iris$Species, sample3)
(sample4 <- Class.sample(iris$Species, prop=0.2, uniform=TRUE))
table(iris$Species, sample4)
```

`classifs` *Classification lists*

Description

Classification lists ('classifs') are probably one of the most ancient attempts to represent biological diversity, the ordered heterogeneity of living things. In biological systematics, they dated from 1753 when Linnaeus published his "Species Plantarum":



Pag 1.

Claſſis 1.

MONANDRIA

MONOGYNIA.

CANNA.

1 CANNA foliis ovatis utrinque acuminatis nervoſis. *Indica.*
Roy. lugdb. 11. *Fl zeyl.* 1. *Hort. upſ.* 1
Canna ſpatulis biflоris. *Hort. cliff.* 1.
Arundo indica latifolia. *Bauh. pin.* 19.
Habitat inter tropicos Aſiæ, Africæ, Americæ. 4

2. CANNA foliis lanceolatis petiolatis nervoſis. *anguſtifol.*
Canna foliis lanceolatis petiolatis. *Hort. cliff.* 1.
Arundo indica florida anguſtifolia. *Moris hiſt.* 3. *p.* 250.

(here on the first page of this book four ranks and five names are represented: class ("Monandria"), order ("Monogynia"), genus ("Canna") and species ("Canna indica" and "Canna angustilolia"))

In essence, classifs require only two columns: rank and name (in that order) so they are easy to standardize as two-column data frames. However, we need to know how to order the ranks. One way is to convert ranks into numbers (Shipunov, 2017). Numranks() implements this functionality.

It is possible to extend classifs with more columns: synonyms, name comments and taxonomic comments. Synonyms (the third column) are especially useful; each synonym will be then one row where second position is a valid name and third position is (one of) synonyms.

Please note that while 'classifs' as data frames are human-readable, they are not typographic. To make them better suited for publication, one might convert them into LaTeX where many packages could be used to typeset classifications (for example, my 'classif2' package).

Note also that in classif, species names must be given in full (in biology, species name consists of two words, (a) genus name and (b) species epithet). One of examples below shows how to replace abbreviations with full genus names.

Usage

```
classifs
```

Format

The list with two data frames representing 'classifs', classification lists. First is the classif with textual ranks, second with numerical ranks. Both based on some classifications of Plantago (ribworts, plantains), first (Shipunov, 2000) include species only from European Russia, the other is from the oldest Plantago monograph (Barneoud, 1845).

Source

Linnaeus C. 1753. Species Plantarum. Holmieae.

Barneoud F.M. 1845. Monographie generale de la familie des Plantaginaceae. Paris.

Shipunov A. 2019. classif2 – Biological classification tables. Version 2.2. See "https://ctan.org/pkg/classif2".

Shipunov A. 2000. The genera Plantago L. and Psyllium Mill. (Plantaginaceae Juss.) in the flora of East Europe. T. Novosti Systematiki Vysshikh Rastenij. 32: 139–152. [In Russian]

Shipunov A. 2017. "Numerical ranks" to improve biological nomenclature of higher groups. 2017. See "https://arxiv.org/abs/1708.07260".

See Also

Biokey, Numranks

Examples

```
## European Russian species classif
plevru <- classifs$plevru
## convert rank names into numbers
plevru[, 1] <- Numranks(ranks=plevru[, 1], add=c(Series=1.1))

## now convert into Newick tree and plot it
plevru.n <- Biokey(plevru, from="classif", to="newick")
library(ape) # to plot, load the 'ape' package
plot(read.tree(text=plevru.n))

## convert classif to taxonomic table
plevru.t <- Biokey(plevru, from="classif", to="table")
colnames(plevru.t) <- Numranks(nums=as.numeric(colnames(plevru.t)))
plevru.t

## two Newick trees
aa <- "(A,(B,C),(D,E));"
bb <- "((A,(B,C)),(D,E));"
## convert them to classif
aa.c <- Biokey(aa, from="newick", to="classif")
bb.c <- Biokey(bb, from="newick", to="classif")
## ... and back to Newick
```

```
aa.n <- Biokey(aa.c, from="classif", to="newick")
bb.n <- Biokey(bb.c, from="classif", to="newick")

## how to convert abbreviated species names
spp <- c ("Plantago afra", "P. arborescens", "P. arenaria")
stt <- do.call(rbind, strsplit(spp, " "))
stt[, 1] <- Fill(stt[, 1], "P.")
(res <- apply(stt, 1, paste, collapse=" "))
```

`Classproj` *Class projection*

Description

Class projection which preserves distances between class centers

Usage

```
Classproj(data, classes, method="DMS")
```

Arguments

`data`	Data: must be numeric and convertible into matrix
`classes`	Class labels (correspond to data rows), NAs are allowed (sic!)
`method`	Either "DMS" for Dhillon et al., 2002 or "QJ" for Qiu and Joe, 2006

Details

'Classproj' is the leveraged (supervised) or educated (semi-supervised) manifold learning (dimension reduction). See examples for the variety of its uses.

It uses classes to determine centers and then tries to preserve distances between centers; two methods are possible: "DMS" which is slightly faster, and "QJ" which frequently finds a better projection.

The code is based on the functions from 'clusterGeneration' package from Weiliang Qiu.

Value

Returns list with 'proj' coordinates of projected data points and 'centers' coordinates of class centers.

Author(s)

Alexey Shipunov

References

Dhillon I.S., Modha D.S., Spangler W.S. 2002. Class visualization of high-dimensional data with applications. Computational Statistics and Data Analysis. 41: 59–90.

Qiu W.-L., Joe H. 2006. Separation index and partial membership for clustering. Computational Statistics and Data Analysis. 50: 585–603.

Examples

```
## Leveraged approach (all classes are known)
iris.dms <- Classproj(iris[, -5], iris$Species, method="DMS")
plot(iris.dms$proj, col=iris$Species)
text(iris.dms$centers, levels(iris$Species), col=1:3)

iris.qj <- Classproj(iris[, -5], iris$Species, method="QJ")
plot(iris.qj$proj, col=iris$Species)
text(iris.qj$centers, levels(iris$Species), col=1:3)

## Educated approach (classes are known only for 10 data points per class)
sam <- Class.sample(iris$Species, 10)
newclasses <- iris$Species
newclasses[!sam] <- NA

iris.dms <- Classproj(iris[, -5], newclasses)
plot(iris.dms$proj, col=iris$Species, pch=ifelse(sam, 19, 1))
text(iris.dms$centers, levels(iris$Species), col=1:3)

iris.qj <- Classproj(iris[, -5], newclasses, method="QJ")
plot(iris.qj$proj, col=iris$Species, pch=ifelse(sam, 19, 1))
text(iris.qj$centers, levels(iris$Species), col=1:3)

## Automated approach (classes calculated automatically)
## Good to visualize _any_ clustering or learning
iris.km <- kmeans(iris[, -5], 3)

iris.dms <- Classproj(iris[, -5], iris.km$cluster)
plot(iris.dms$proj, col=iris.km$cluster)
text(iris.dms$centers, labels=1:3, col=1:3, cex=2)

iris.qj <- Classproj(iris[, -5], iris.km$cluster, method="QJ")
plot(iris.qj$proj, col=iris.km$cluster)
text(iris.qj$centers, labels=1:3, col=1:3, cex=2)
```

`Coeff.det` *Average coefficients of determination for each variable*

Description

Average coefficients of determination for each variable

Usage

```
Coeff.det(X, ...)
```

Arguments

X	Data frame or matrix with values
...	Arguments to 'cor()'

Details

Average coefficients of determination for each variable.

Allows to compare various correlation structures (Rostova, 1999; Rostova, 2002).

Value

Numerical vector of coefficients of determination

Author(s)

Alexey Shipunov

References

Rostova N.S. 1999. The variability of correlations systems between the morphological characters. Part 1. Natural populations of Leucanthemum vulgare (Asteraceae). Botanicheskij Zhurnal. 84(11): 50–66.

Rostova N.S. 2002. Correlations: Structure and Variability. Saint Petersburg, St. Petersburg University Publisher.

Examples

```
Coeff.det(trees, use="pairwise")
```

Coml *Compare checklists*

Description

Compare species checklists

Usage

```
Coml(df1, df2)
## S3 method for class 'Coml'
summary(object, ..., n=10)
```

Arguments

df1	First data frame with species presence/absence data, species as row names
df2	Second data frame
object	Object of the class 'Coml'
n	Number of indicator species
...	Additional arguments

Details

Compare two (groups of) checklists (Abramova et al., 2003).

Calculates difference (in %) between checklists with *common base*, i.e., species occurrence/abundance columns of data frame with species names as row names.

Finds names of "indicators" most characteristic to each group

Value

Object of the class 'Coml', or nothing

Author(s)

Alexey Shipunov

References

Abramova L. A., Rimskaya-Korsakova N. N., Shipunov A. B. 2003. The comparative study of the flora of Kiv Gulf, Chupa Gulf and Keret' Archipelago islands (Kandalaksha Bay of White Sea). Proceedings of the Pertsov White Sea Biological Station. Vol. 9. Moscow. P. 22–33. in Russian (English abstract)

Examples

```
y.Coml <- Coml(dolbli[1:45], dolbli[46:79])
summary(y.Coml, n=5)
```

Cor *Correlation matrix with p-values*

Description

Correlation matrix with p-values

Usage

```
Cor(X, stars=TRUE, dec=4, p.level=0.05, ...)
Cor2(X, dec=4, p.level=0.05)
```

Arguments

X	Matrix or data frame with values
stars	Replaces p-values with stars if it not greater than 'p.level'
dec	Decimal point
p.level	P-level
...	Arguments to 'cor.test()'

Details

'Cor()' calculates correlation matrix with p-values.

'Cor2()' is another (faster) variant of correlation matrix with p-values based on F-statistic. Shows significances in the upper triagle. Uses Pearson correlation only but much faster than 'Cor()'.

Author(s)

Alexey Shipunov

Examples

```
Cor(longley, dec=2)
Cor2(longley, dec=2)
```

Cor.vec *Calculates correlation and converts results into the named long vector*

Description

Calculates correlation and converts results into the named long vector

Usage

```
Cor.vec(X, ...)
```

Arguments

X	Data frame or matrix with values
...	Arguments to 'cor()'

Details

Calculates correlation and converts results into the named long vector (Rostova, 1999; Rostova, 2002).

Value

Named numerical vector of correlations.

Author(s)

Alexey Shipunov

References

Rostova N.S. 1999. The variability of correlations systems between the morphological characters. Part 1. Natural populations of Leucanthemum vulgare (Asteraceae). Botanicheskij Zhurnal. 84(11): 50–66.

Rostova N.S. 2002. Correlations: Structure and Variability. Saint Petersburg, St. Petersburg University Publisher.

See Also

Rostova.tbl

Examples

```
Cor.vec(trees, method="spearman")
```

CVs *Coefficients of variation*

Description

Coefficients of variation

Usage

```
CVs(sample, na.rm=TRUE)
```

Arguments

sample	Numerical vector
na.rm	Remove NAs?

Details

Coefficients of variation: different variants of the standardized range

Value

Named numerical vector

Author(s)

Alexey Shipunov

Examples

```
sapply(trees, CVs)
```

Ditto *Removes duplicated data values downstream*

Description

Replaces duplicated values with "ditto" string

Usage

```
Ditto(x, ditto="")
```

Arguments

x	Vector, possibly with missing values
ditto	String to replace with, typically empty string "" (default)

Details

If the first argument is not a character vector, Ditto() converts it to the character.

Value

Vector with replaced values

Author(s)

Alexey Shipunov

See Also

Fill

Examples

```
Ditto(c("a", "a", "", "b", "b"))
Ditto(c("a", "a", "", "b", NA, "b"))
Ditto(c("a", "a", "", "b", NA, "b"), ditto=NA)
Ditto(c("a", "a", "", "b", NA, "b"), ditto="--")
```

DNN *Distance matrix based kNN classification*

Description

DNN uses pre-cooked distance matrix to replace missing values in class labels.

Usage

```
DNN(dst, cl, k, d, details=FALSE, self=FALSE)
Dnn(trn, tst, classes, FUN=function(.x) dist(.x), ...)
```

Arguments

dst	Distance matrix (object of class 'dist').
cl	Factor of class labels, should contain NAs to designate testing sub-group.
k	How many neighbors to select, odd numbers preferable. If specified, do not use "d".
d	Distance to consider for neighborhood, in fractions of maximal distance. If specified, do not use "k".
details	If TRUE, function will return voting matrix. Default is FALSE.
self	Allow self-training? Default is FALSE.
trn	Data to train from, classes variable out.
tst	Data with unknown classes.
classes	Classes variable for training data.
FUN	Function to calculate distances, by default, just dist() (i.e., Euclidean distances).
...	Additional arguments from Dnn() to DNN(), note that either 'k' or 'd' must be specified.

Details

If classic kNN is a lazy classifier, DNN is super-lazy because it does not even calculate the distance matrix itself. Instead, you supply it with distance matrix (object of class 'dist') pre-computed with _any_ possible tool. This lifts many restrictions. For example, arbitrary distance could be used (like Gower distance which allows any type of variable). This is also much faster than typical kNN.

In addition to neighbor-based kNN classification, DNN implements _neighborhood_ classification when all neighbors within selected distance used for voting.

As usual in kNN, ties are broken at random. DNN also controls situations when no neighbors are within the given distance (and returns NA), and also when all neighbors are relevant (also returns NA).

By default, DNN() returns missing part of class labels, completely or partially filled with new (predicted) class labels. If 'cl' has no NAs and self=FALSE (default), DNN() returns it back with warning. It allows for combined and stepwise extensions (see examples). If 'details=TRUE', DNN()

will return matrix where each column represents the table used for voting. If self=TRUE, DNN() could be used to calculate the class proximity surrogate.

Dnn() is based on DNN() but has more class::knn()-like interface (see examples).

Value

Character vector with predicted class labels; or matrix if 'details=TRUE'.

Author(s)

Alexey Shipunov

See Also

class::knn

Examples

```
iris.d <- dist(iris[, -5])

cl1 <- iris$Species
sam <- c(rep(0, 4), 1) > 0
cl1[!sam] <- NA
table(cl1, useNA="ifany")

## based on neighbor number
iris.pred <- DNN(dst=iris.d, cl=cl1, k=5)
Misclass(iris$Species[is.na(cl1)], iris.pred)

## based on neighborhood size
iris.pred <- DNN(dst=iris.d, cl=cl1, d=0.05)
table(iris.pred, useNA="ifany")
Misclass(iris$Species[is.na(cl1)], iris.pred)

## protection against "all points relevant"
DNN(dst=iris.d, cl=cl1, d=1)[1:5]
## and all are ties:
DNN(dst=iris.d, cl=cl1, d=1, details=TRUE)[, 1:5]

## any distance works
iris.d2 <- Gower.dist(iris[, -5])
iris.pred <- DNN(dst=iris.d2, cl=cl1, k=5)
Misclass(iris$Species[is.na(cl1)], iris.pred)

## combined
cl2 <- cl1
iris.pred <- DNN(dst=iris.d, cl=cl2, d=0.05)
cl2[is.na(cl2)] <- iris.pred
table(cl2, useNA="ifany")
iris.pred2 <- DNN(dst=iris.d, cl=cl2, k=5)
cl2[is.na(cl2)] <- iris.pred2
table(cl2, useNA="ifany")
```

```
Misclass(iris$Species, cl2)

## self-training and class proximity surrogate
cl3 <- iris$Species
t(DNN(dst=iris.d, cl=cl3, k=5, details=TRUE, self=TRUE))/5

## Dnn() with more class::knn()-like interface
iris.trn <- iris[sam, ]
iris.tst <- iris[!sam, ]
Dnn(iris.trn[, -5], iris.tst[, -5], iris.trn[, 5], k=7)

## stepwise DNN, note the warning when no NAs left
cl4 <- cl1
for (d in (5:14)/100) {
iris.pred <- DNN(dst=iris.d, cl=cl4, d=d)
cl4[is.na(cl4)] <- iris.pred
}
table(cl4, useNA="ifany")
Misclass(iris$Species, cl4)
## rushing to d=14% gives much worse results
iris.pred <- DNN(dst=iris.d, cl=cl1, d=0.14)
table(iris.pred, useNA="ifany")
Misclass(iris$Species[is.na(cl1)], iris.pred)
```

`dolbli` *dolbli*

Description

Plants of two Arctic lakes.

Observations on the coastal flora of Arctic lakes. Flora was sampled on the 20 m coastline. 1999–2004.

Usage

```
dolbli
```

Format

`columns` Lake names with plot numbers, data is abundance of plant species, in 1543 scale (0 – absent; 1 – one individual plant; 2 – no more than 12 individual plants (rametes); 3 – number of individuals is more than 12 but no more than 5% of total number of plants on a plot; 4 – number of individuals is more than 5% but no more than 25% of total number of plants on a plot; 5 – number of individuals is more than 25% but no more than 50% of total number of plants on a plot; 6 – number of individuals is more than 50% but no more than 75% of total number of plants on a plot; 7 – number of individuals is more than 75% of total number of plants on a plot.)

`rows` Names of plant species, trees start with 0

Source

Shipunov, A. The creation of databases about flora of isles and lakes of North Karelia. Abstract. P. 97–98. Study of the flora of East Europe: Achievements and prospects. 23–28 May 2005, St.-Petersburg.

Shipunov A., Motyleva M. The comparative study of the flora of lakes in Tchupa gulf environs. III scientific session of Marine Biological Station of Saint-Petersburg State University. Abstracts. P. 27–29. Saint-Petersburg, 2002. [In Russian].

`Dotcharts` *Improved dotcharts*

Description

Dotcharts, improved and extended

Usage

```
Dotchart1(x, labels=NULL, groups=NULL, gdata=NULL, offset=1/8,
 ann=par("ann"), xaxt=par("xaxt"),  frame.plot=TRUE, log="",
 cex=par("cex"), pt.cex=cex, pch=21, gpch=21, bg=par("bg"),
 color=par("fg"),  gcolor=par("fg"), lcolor="gray", xlim=
 range(x[is.finite(x)]), main=NULL, xlab=NULL, ylab=NULL, ...)

Dotchart(x, ...)

Dotchart3(values, left, right, pch=21, bg="white", pt.cex=1.2,
 lty=1, lwd=2, gridcol="grey", ...)
```

Arguments

`x`	Either a vector or matrix of numeric values. Inputs are coerced by 'as.numeric()', with a message.
`labels`	A vector of labels for each point.
`groups`	An optional factor indicating how the elements of 'x' are grouped.
`gdata`	Data values for the groups. This is typically a summary such as the median or mean of each group.
`offset`	Offset in inches of 'ylab' and 'labels'.
`ann`	Logical value indicating whether title and x and y axis labels should appear on the plot.
`xaxt`	String indicating the x-axis style; use 'n' to suppress and see also par("xaxt").
`frame.plot`	Logical indicating whether a box should be drawn around the plot.

log	Character string indicating if one or the other axis should be logarithmic, see ?plot.default.
cex	The character size to be used.
pt.cex	The 'cex' to be applied to plotting symbols.
pch	The plotting character or symbol to be used.
gpch	The plotting character or symbol to be used for group values.
bg	The background color of plotting characters.
color	The color(s) to be used for points and labels.
gcolor	The single color to be used for group labels and values.
lcolor	The color(s) to be used for the horizontal lines.
xlim	Horizontal range for the plot.
main	Overall title for the plot, see 'title'.
xlab, ylab	Axis annotations as in 'title'.
values	Centers for 'Dotchart3()'
left	Left margins for 'Dotchart3()'
right	Right margins for 'Dotchart3()'
lty	Line type for 'Dotchart3()'
lwd	Line width for 'Dotchart3()'
gridcol	Grid color for 'Dotchart3()'
...	Additional arguments

Details

For better explanations of options, see 'help(dotchart)'.

Dotchart1() is a dotchart() corrected for use with 1-dimensional tables. If the argument is a 1-dimensional table, Dotchart() converts it into numeric vector first and instead of warning, outputs the message. This is helpful to the beginners with R, and especially on macOS GUI where warnings are in red. It also allows dotcharts to show 'ylab' (this was not available in R < 4.0.3 but corrected later).

Dotchart() is a prettified dotchart with the following defaults: 'lcolor="black", bg="white", pt.cex=1.2'.

Dotchart3() is the dotchart extension which shows values together with ranges. Somewhat similar to Linechart() but more general (and does not work with grouped data).

Author(s)

Alexey Shipunov

See Also

dotchart, Linechart

Examples

```
## Compare:
aa <- table(c(1, 1, 1, 2, 2, 3))
dotchart(aa, ylab="Ylab") # produces warning; does not show 'ylab' if R version < 4.0.3
Dotchart1(aa, ylab="Ylab") # outputs message instead of warning; always shows 'ylab'
Dotchart(aa, ylab="Ylab") # in addition to Dotchart1(), dots and grid are more visible

iris1 <- aggregate(iris[, 1], iris[5], function(.x) fivenum(.x)[c(3, 1, 5)])
iris1x <- iris1$x
row.names(iris1x) <- iris1$Species
Dotchart3(iris1x[, 1], iris1x[, 2], iris1x[, 3])
```

`drosera` *drosera*

Description

Observations on the round-leaf sundew, Drosera rotundifolia, White Sea coast, August 2000.

Usage

```
drosera
```

Format

This data frame contains the following columns:

`POP` Code of the population

`YOUNG.L` Number of young, not opened leaves

`MATURE.L` Number of mature, catching leaves

`OLD.L` Number of old, degrading leaves

`INSECTS` Total number of insects per plant

`INFL.L` Inflorescence length (0 if absent), mm

`STALK.L` Length of stalk (without flowers), mm

`N.FLOW` Number of flowers

`LEAF.L` Length of maximal leaf, mm

`LEAF.W` Width of maximal leaf, mm

`PET.L` Length of maximal leaf petiole, mm

`Ell` *Plot ellipse*

Description

Plot ellipse

Usage

```
Ell(x, y, width, height=width, theta=2*pi, npoints=100, plot=TRUE, ...)
```

Arguments

`x`	x coordinate of center
`y`	y coordinate of center
`width`	length of major axis
`height`	length of minor axis
`theta`	rotation
`npoints`	number of points to send to polygon
`plot`	if TRUE, add to current device, if FALSE, returns list of components
`...`	arguments to 'polygon()'

Details

Plots ellipse based on 'polygon()'.

Value

If plot=FALSE, returns list of components.

Author(s)

Alexey Shipunov

Examples

```
plot(1:8, type="n")
Ell(4, 5, 6)
```

`Ellipses` *Confidence ellipses*

Description

Calculates and plots group confidence ellipses

Usage

```
Ellipses(pts, groups, match.color=TRUE, usecolors=NULL,
 centers=FALSE, c.pch=0, c.cex=3,
 level=0.95, df=1000, prec=51,
 coords=NULL, plot=TRUE, ...)
```

Arguments

`pts`	Data points to plot
`groups`	Grouping variable
`match.color`	Match colors
`usecolors`	Use colors (palette)
`centers`	Show centers?
`c.pch`	Color of center points
`c.cex`	Scale of center points
`level`	Confidence level for F-distribution
`df`	Used in calculation of P-content according to F(2, df) distribution
`prec`	Precision of ellipse plotting (default is 51 points)
`coords`	Pre-calculated ellipses coordinates: list of two-column matrices named as groups (by default, not required)
`plot`	Plot?
`...`	Arguments to lines()

Details

Note that (at least at the moment), ellipses are plotted with line(), therefore shading is not straightforward (but possible, see examples).

Also, with a help from Pinhull() (see its help), it is possible to reveal "outliers", points outside of each ellipse borders.

See also package 'cluster' for ellipsoidhulls() function that allows to draw ellipse-like hulls.

Value

Invisibly returns the list in the form similar to Hulls(), to use as a list of polygons or with Overlap().

Author(s)

Alexey Shipunov

See Also

Hulls, Overlap, Pinhull

Examples

```
iris.p <- prcomp(iris[, -5])$x[, 1:2]
plot(iris.p, type="n", xlab="PC1", ylab="PC2")
text(iris.p, labels=abbreviate(iris[, 5], 1, method="both.sides"))
iris.e <- Ellipses(iris.p[, 1:2], iris[, 5], centers=TRUE)

## calculate overlap between ellipses
Overlap(iris.e)

## how to plot filled ellipses
plot(iris.p, type="n", xlab="PC1", ylab="PC2")
text(iris.p, labels=abbreviate(iris[, 5], 1, method="both.sides"))
for (i in seq_along(iris.e))
 polygon(iris.e[[i]], border=NA, col=adjustcolor(i, alpha.f=0.2))

## how to reveal (and label) "outliers", points outside of _all_ ellipses
iris.pie <- Pinhull(iris.p, iris.e)
outs <- which(apply(iris.pie, 1, sum) == 0)
points(iris.p[outs, ], cex=2, pch=4)

## embedded convex hulls
plot(iris.p, col=iris$Species)
for (i in seq_along(iris.e)) lines(iris.e[[i]], col=i, lty=2)
mi <- cbind(seq_len(nrow(iris)), as.numeric(iris$Species)) # indexing matrix
## remove "outliers" in broad sense, points which are outside of its "own" ellipse:
emb <- rowSums(iris.pie) == 1 & iris.pie[mi]
Hulls(iris.p[emb, ], iris$Species[emb])

## LDA ellipes
library(MASS)
ch.lda <- lda(Species ~ ., data=chaetocnema[, -2])
ch.lda.pred <- predict(ch.lda, chaetocnema[, -(1:2)])
## ellipses here are by default bigger then plot so use workaround:
ee <- Ellipses(ch.lda.pred$x, chaetocnema$Species, plot=FALSE)
xx <- range(c(do.call(rbind, ee)[, 1], ch.lda.pred$x[, 1]))
yy <- range(c(do.call(rbind, ee)[, 2], ch.lda.pred$x[, 2]))
plot(ch.lda.pred$x, col=chaetocnema$Species, xlim=xx, ylim=yy)
Ellipses(ch.lda.pred$x, chaetocnema$Species, coords=ee)

## search for the maximal level which gives zero overlap
plot(x5 ~ x17, data=haltica, pch=as.numeric(haltica$Species))
for (i in (99:59)/100) {
cat(i, "\n")
```

```
ee <- Ellipses(haltica[, c("x17", "x5")], haltica$Species, level=i, plot=FALSE)
print(mean(Overlap(ee), na.rm=TRUE))
cat("\n")
}
Ellipses(haltica[, c("x17", "x5")], haltica$Species, level=.62)
```

---

eq *eq*

Description

Horsetails (*Equisetum*) are the old, pre-dinosaur plant lineage. Only several dozen species survived, but despite a long evolution the borders between these species are still unclear for researchers.

In 2005–2006, morphometric analysis was performed of more than 1,000 horsetail plants belong to most widespread Eurasian species growing in Middle Russia. For the analysis, we used 8 morphological characters and also tried to identify species.

'eq_l' contains population locations and species determinations.

'eq_s' and 'eq' are actual morphometric data, but 'eq' contains only 2 species out of 8.

Usage

```
eq
eq_l
eq_s
```

Format

This data frame contains the following columns:

`DL.R` plant height, mm

`DIA.ST` maximal diameter of stem, mm

`N.REB` number of ridges on a stem

`N.ZUB` number of teeth (reduced leaves)

`DL.OSN.Z` length of tooth base

`DL.TR.V` length of sheath

`DL.BAZ` length of basal segment of branch

`DL.PER` length of first (after the basal) segment of branch

`SPECIES` preliminary species determination

`N.POP` population number

`WHERE` population location (region)

Ex.boxplot *Boxplot explanation*

Description

Boxplot explanation

Usage

```
Ex.boxplot(...)
```

Arguments

...	Arguments to 'boxplot()'

Details

The scheme which explains typical boxplot.

Author(s)

Alexey Shipunov

See Also

boxplot

Examples

```
Ex.boxplot()
```

Ex.col *Examples of colors*

Description

Examples of colors (current colors or all named colors)

Usage

```
Ex.col(all=FALSE)
Ex.cols(all=FALSE)
```

Arguments

all	Show all named colors?

Details

If 'all=FALSE' (default), plots current colors along with their names and numeric codes; "white" is added as the first color (with numeric code 0). This plot does not usually look nice if the current palette contains more than 40–45 colors.

If 'all=TRUE', plots all named colors plus (for completedness) "transparent", which also can be used as color specification in R. Large device is required to see all (almost 500) named colors.

For the palettes, run 'example(rainbow)' and other palette-related commands.

Author(s)

Alexey Shipunov

See Also

palette, rainbow, colors

Examples

```
Ex.cols()
Ex.cols(all=TRUE)
```

Ex.font *Examples of fonts*

Description

Examples of standard fonts

Usage

```
Ex.font()
```

Details

Examples of standard fonts

Author(s)

Alexey Shipunov

See Also

par

Examples

```
Ex.fonts()
```

Ex.lty *Examples of line types*

Description

Line type examples

Usage

```
Ex.lty(custom="431313")
Ex.lines(custom="431313")
```

Arguments

custom	character string to specify custom line type (see '?lines').

Details

Line type examples. To see other possible custom line types, try custom="F8" or similar.

Author(s)

Alexey Shipunov

See Also

lines

Examples

```
Ex.lines(custom="F8")
```

Ex.margins *Example of plot margins*

Description

Example of plot margins

Usage

```
Ex.margins()
```

Details

Example of plot margins. Modified from Paul Murrell (2006).

Author(s)

Alexey Shipunov

References

Murrell P. 2006. R Graphics.

See Also

par

Examples

```
Ex.margins()
```

Ex.pch *Point examples*

Description

Point ('pch') examples

Usage

```
Ex.pch(extras=c("*", ".", "+", "a"), cex=2, col="black", bg="gray",
 coltext="black", cextext=1.2, main="")
Ex.points(extras=c("*", ".", "+", "a"), cex=2, col="black", bg="gray",
 coltext="black", cextext=1.2, main="")
```

Arguments

extras	which extra symbols to show
cex	point scale, default 2
col	point color, default black
bg	point background (for symbols with a 'bg'-colored interior), default gray
coltext	text color, default black
cextext	text scale, default 1.2
main	plot title, no title by default

Details

Point ('pch') examples, modified from 'example(points)'.

Author(s)

Alexey Shipunov

See Also

points

Examples

```
Ex.points()
```

`Ex.plots` *Examples of plot types*

Description

Examples of plot types

Usage

```
Ex.plots()
Ex.types()
```

Details

Examples of nine standard plot types.

Author(s)

Alexey Shipunov

See Also

par

Examples

```
Ex.types()
```

Fence *Colorize tips of 'hclust' plot*

Description

Uses segments() and Tcoords() to colorize 'hclust' plot

Usage

```
Fence(hcl, fct, ex=0.05, lwd=2.5, horiz=FALSE, hang=0.1, ...)
```

Arguments

hcl	hclust object
fct	Variable to colorize labels, will be converted into factor
ex	The fraction of the plot height by which segments go up and down from the tips; by default, it is half of the 'hang'
lwd	Line width of segments
horiz	Plot on a horizontal tree?
hang	The fraction of the plot height by which labels should hang below the rest of the plot; by default, it is equal to the default 'hang' from hclust which is 0.1
...	Further arguments to segments()

See Also

Tcoords, hclust

Examples

```
iris.h <- hclust(dist(iris[, -5]))
plot(iris.h, labels=FALSE)
Fence(iris.h, iris$Species)
legend("topright", legend=levels(iris$Species), col=1:3, lwd=2.5, bty="n")
```

`Files` *Textual file system browser*

Description

Textual file system browser

Usage

```
Files(root=getwd(), multiple=FALSE, hidden=FALSE)
```

Arguments

`root`	Root directory to explore, default is the working directory
`multiple`	Allows multiple files to be selected
`hidden`	Show hidden files?

Details

Interactive text-based file chooser dialog, modified from code published by "mathematical.coffee" on Stack Overflow as "R command-line file dialog".

If 'multiple=TRUE', one can select files one by one (they will "disappear" from the displayed list), and typing "0" will output this list. If "multiple=FALSE", typing "0" will output the name of the current directory.

Files() uses normalizePath() so symbolic links will be resolved. Also, Files() is not very useful when number of files in the directory is large.

Alternatives for Linux: 'tcltk::tk_choose.files()' and 'tcltk::tk_choose.dir()'

Value

Returns character vector of selected files, or directory name (useful for 'setwd()'), or new user-defined file name with full path.

Author(s)

Alexey Shipunov

See Also

`setwd`, `getwd`, `dir`

Examples

```
## Not run:
## interactive commands
setwd <- Files() # then select directory to work in
Files("~", hidden=TRUE) # explore home directory with hidden files (Linux, macOS)

## End(Not run)
```

`Fill` *Fill data values downstream, like in spreadsheets*

Description

Replaces "ditto" values with preceding values

Usage

```
Fill(x, ditto="")
```

Arguments

`x`	Vector, possibly with missing values
`ditto`	What to fill, typically empty string "" (default) or NA

Value

Vector with replaced values

Author(s)

Alexey Shipunov

See Also

`Ditto`

Examples

```
aa <- c("a", "a", "", "b", "", "c", "d", "")
Fill(aa)
bb <- c("a", "a", NA, "b", NA, "c", "d", NA)
Fill(bb, ditto=NA)
dd <- c("", "a", "a", "", "", "b", NA, "", "c", "d", "")
Fill(dd)
```

`Gap.code` *Gap coding*

Description

Gap coding of DNA nucleotide alignments

Usage

```
Gap.code(seqs)
```

Arguments

`seqs`	Character vector of aligned (and preferably flank trimmed) DNA sequences.

Details

FastGap-like gap code nucleotide alignments ('ATGCN-' are allowed).

Encodes gap presence as 'A' and absence as 'C'.

Likely too straightforward, and only weakly optimized (really slow).

Value

Outputs character matrix where each column is a gapcoded position.

Author(s)

Alexey Shipunov

References

Borchsenius F. 2009. FastGap 1.2. Department of Biosciences, Aarhus University, Denmark. See "http://www.aubot.dk/FastGap_home.htm".

Examples

```
write(file=file.path(tempdir(), "tmp.fasta"),  c(
 ">1\nGAAC------ATGC",
 ">2\nGAAC------TTGC",
 ">3\nGAAC---CCTTTGC",
 ">4\nGAA---------GC"))
write(file=file.path(tempdir(), "tmp_expected.fasta"), c(
 ">1\nGAAC------ATGCCA-",
 ">2\nGAAC------TTGCCA-",
 ">3\nGAAC---CCTTTGCCCA",
 ">4\nGAA---------GCA--"))
tmp <- Read.fasta(file=file.path(tempdir(), "tmp.fasta"))
expected <- Read.fasta(file=file.path(tempdir(), "tmp_expected.fasta"))
```

```
seqs <- tmp$sequence
gc <- Gap.code(seqs)
tmp$sequence <- apply(cbind(seqs, gc), 1, paste, collapse="")
identical(tmp, expected) # TRUE, isn't it?
```

`Gen.cl.data` *Generates datasets for clustering*

Description

Imitation of the Python `sklearn.datasets` functions.

Usage

```
Gen.cl.data(type=c("blobs", "moons", "circles"), N=100, noise=NULL,
 shuffle=TRUE, bdim=2, bcenters=3, bnoise=1, bbox=c(-10, 10), cfactor=0.8)
```

Arguments

`type`	'blobs' are Gaussian blobs; 'moons' are two interleaving half-circles; 'circles' are two embedded circles
`N`	Number of data points
`shuffle`	Whether to randomize the output
`noise`	Standard deviation of Gaussian noise applied to point positions
`bdim`	Dimensionality of 'blobs' dataset
`bcenters`	Number of 'blobs' centers
`bnoise`	Standard deviation of 'blobs' Gaussian noise: vector of length one or length equal to the number of centers
`bbox`	The bounding box within which blobs centers will be created
`cfactor`	Scale factor between 'circles' (should be > 0 and < 1)

Details

Algorihms were taken partly from Python 'scikit-learn' and from Github 'elbamos/clusteringdatasets'.

Author(s)

Alexey Shipunov

Examples

```
scikit.palette <- c("#377EB8", "#FF7F00", "#4DAF4A", "#F781BF", "#A65628", "#984EA3",
"#999999", "#E41A1C", "#DEDE00", "#000000")
palette(scikit.palette)
n.samples <- 500

## data
set.seed(21)
no.structure <- list(samples=cbind(runif(n.samples), runif(n.samples)),
 labels=rep(1, n.samples))
noisy.circles <- Gen.cl.data(type="circles", N=n.samples, cfactor=0.5, noise=0.05)
noisy.moons <- Gen.cl.data(type="moons", N=n.samples, noise=0.05)
blobs <- Gen.cl.data(type="blobs", N=n.samples, noise=1)
## anisotropically distributed data
aniso <- Gen.cl.data(type="blobs", N=n.samples)
aniso$samples <- aniso$samples %*% rbind(c(0.6, -0.6), c(-0.4, 0.8))
## blobs with varied variances
varied <- Gen.cl.data(type="blobs", N=n.samples, bnoise=c(1, 2.5, 0.5))
set.seed(NULL)

## single example
plot(aniso$samples, col=aniso$labels, pch=19)

## all data objects example
## old.X11.options <- X11.options(width=6, height=6) # to make square cells
oldpar <- par(mfrow=c(2, 3), mar=c(1, 1, 3, 1))
for (n in c("noisy.circles", "noisy.moons", "no.structure",
 "blobs", "aniso", "varied")) {
 plot(get(n)$samples, col=get(n)$labels, pch=19, main=n, xlab="", ylab="",
 xaxt="n", yaxt="n")
}
par(oldpar)
## X11.options <- old.X11.options


## comparison of clustering techniques example
## old.X11.options <- X11.options(width=10, height=6)  # to make square cells
oldpar <- par(mfrow=c(6, 10), mar=rep(0, 4), xaxt="n", yaxt="n")
COUNT <- 1
for (n in c("noisy.circles", "noisy.moons", "no.structure", "blobs", "aniso", "varied")) {
 K <- 3
 if (n %in% c("noisy.circles", "noisy.moons")) K <- 2
 TITLE <- function(x) if (COUNT==1) { legend("topleft", legend=x, cex=1.25, bty="n") }
##
 newlabels <- cutree(hclust(dist(get(n)$samples), method="ward.D2"), k=K)
 plot(get(n)$samples, col=newlabels, pch=19)
 TITLE("Ward")
##
 newlabels <- cutree(hclust(dist(get(n)$samples), method="average"), k=K)
 plot(get(n)$samples, col=newlabels, pch=19)
 TITLE("UPGMA")
##
```

```
 newlabels <- kmeans(round(get(n)$samples, 5), centers=K)$cluster
 plot(get(n)$samples, col=newlabels, pch=19)
 TITLE("K-means")
##
 newlabels <- cutree(as.hclust(cluster::diana(dist(get(n)$samples))), k=K) # slow
 plot(get(n)$samples, col=newlabels, pch=19)
 TITLE("DIANA")
##
 nn <- cluster::fanny(get(n)$samples, k=K) # a bit slow
 dunn <- apply(nn$membership, 1, function(.x) (sum(.x^2) - 1/K) / (1 - 1/K))
 fuzzy <- dunn < 0.05
 plot(get(n)$samples[!fuzzy, ], col=nn$clustering[!fuzzy], pch=19)
 points(get(n)$samples[fuzzy, ], col="black", pch=1)
 TITLE("FANNY")
##
 newlabels <- kernlab::specc(get(n)$samples, centers=K)
 plot(get(n)$samples, col=newlabels, pch=19)
 TITLE("spectral")
##
 nn <- apcluster::apclusterK(apcluster::negDistMat(), get(n)$samples, K=K) # very slow
 newlabes <- apply(sapply(nn@clusters,
  function(.y) 1:nrow(get(n)$samples) %in% .y), 1, which)
 plot(get(n)$samples, col=newlabels, pch=19)
 TITLE("AP") # affinity propagation
##
 ## eps values taken out of scikit and 'dbscan::kNNdistplot() "knee"', 'minPts' default
 EPS <- c(noisy.circles=0.3, noisy.moons=0.3, no.structure=0.3, blobs=1,
  aniso=0.5, varied=1)
 nn <- dbscan::dbscan(get(n)$samples, eps=EPS[n])
 outliers <- nn$cluster == 0
 plot(get(n)$samples[!outliers, ], col=nn$cluster[!outliers], pch=19)
 points(get(n)$samples[outliers, ], col="black", pch=1)
 TITLE("DBSCAN")
##
 newlabels <- meanShiftR::meanShift(get(n)$samples, nNeighbors=10)$assignment
 plot(get(n)$samples, col=newlabels, pch=19)
 TITLE("mean-shift")
##
 library(mclust)
 newlabels <- Mclust(get(n)$samples)$classification
 plot(get(n)$samples, col=newlabels, pch=19)
 TITLE("Gaussian")
 COUNT <- COUNT + 1
}
par(oldpar)
## X11.options <- old.X11.options



## comparison of linkages example
## old.X11.options <- X11.options(width=8, height=6) # to make square cells
oldpar <- par(mfrow=c(6, 8), mar=rep(0, 4), xaxt="n", yaxt="n")
COUNT <- 1
```

```
for (n in c("noisy.circles", "noisy.moons", "no.structure", "blobs", "aniso", "varied")) {
 K <- 3 ; if (n %in% c("noisy.circles", "noisy.moons")) K <- 2
 TITLE <- function(x) if (COUNT==1) { legend("topleft", legend=x, cex=1.25, bty="n") }
 newlabels <- cutree(hclust(dist(get(n)$samples), method="ward.D2"), k=K)
 plot(get(n)$samples, col=newlabels, pch=19)
 TITLE("Ward orig")
 newlabels <- cutree(hclust(dist(get(n)$samples), method="ward.D"), k=K)
 plot(get(n)$samples, col=newlabels, pch=19)
 TITLE("Ward")
 newlabels <- cutree(hclust(dist(get(n)$samples), method="average"), k=K)
 plot(get(n)$samples, col=newlabels, pch=19)
 TITLE("UPGMA")
 newlabels <- cutree(hclust(dist(get(n)$samples), method="single"), k=K)
 plot(get(n)$samples, col=newlabels, pch=19)
 TITLE("single")
 newlabels <- cutree(hclust(dist(get(n)$samples), method="complete"), k=K)
 plot(get(n)$samples, col=newlabels, pch=19)
 TITLE("complete")
 newlabels <- cutree(hclust(dist(get(n)$samples), method="mcquitty"), k=K)
 plot(get(n)$samples, col=newlabels, pch=19)
 TITLE("WPGMA")
 newlabels <- cutree(hclust(dist(get(n)$samples), method="median"), k=K)
 plot(get(n)$samples, col=newlabels, pch=19)
 TITLE("WPGMC")
 newlabels <- cutree(hclust(dist(get(n)$samples), method="centroid"), k=K)
 plot(get(n)$samples, col=newlabels, pch=19)
 TITLE("UPGMC")
 COUNT <- COUNT + 1
}
par(oldpar)
## X11.options <- old.X11.options

palette("default")
```

`Gower.dist` *Gower distance*

Description

Calculates Gower distance

Usage

```
Gower.dist(data.x, data.y=data.x, rngs=NULL,
 KR.corr=TRUE, na.rm=FALSE)
```

Arguments

`data.x`	A matrix or a data frame containing variables that should be used in the computation of the distance.

data.y	A numeric matrix or data frame with the same variables, of the same type, as those in 'data.x'
rngs	A vector with the ranges to scale the variables. Its length must be equal to number of variables in 'data.x'
KR.corr	When TRUE (default) the extension of the Gower's dissimilarity measure proposed by Kaufman and Rousseeuw (1990) is used. Otherwise, the original Gower's (1971) formula is considered.
na.rm	Replace missing values with maximal distance?

Details

Gower.dist() code based on analogous function from 'StatMatch' package; please see this package for the original code and full documentation.

This function computes the Gower's distance (dissimilarity) among units in a dataset or among observations in two distinct datasets. Columns of mode numeric will be considered as interval scaled variables; columns of mode character or class factor will be considered as categorical nominal variables; columns of class ordered will be considered as categorical ordinal variables and, columns of mode logical will be considered as binary asymmetric variables. Missing values (NA) are allowed. If only data.x is supplied, the dissimilarities between _rows_ of data.x will be computed.

For 'rngs', in correspondence of non-numeric variables, just put 1 or NA. When rngs=NULL (default), the range of a numeric variable is estimated by jointly considering the values for the variable in 'data.x' and those in 'data.y'.

When 'na.rm=TRUE', all missing values (NAs and NaNs) in the result will be replaced with maximal distance. This is discussable but helps, e.g., to bootstrap hierarchical clustering in case if data is rich of NAs.

Value

A distance object with distances among rows of 'data.x' and those of 'data.y'.

Author(s)

Alexey Shipunov

References

Gower J.C. 1971. A general coefficient of similarity and some of its properties. Biometrics. 27: 623–637.

Kaufman L., Rousseeuw P.J. 1990. Finding Groups in Data: An Introduction to Cluster Analysis. Wiley, New York.

See Also

dist, cluster::daisy

Examples

```
x1 <- as.logical(rbinom(10, 1, 0.5))
x2 <- sample(letters, 10, replace=TRUE)
x3 <- rnorm(10)
x4 <- ordered(cut(x3, -4:4, include.lowest=TRUE))
xx <- data.frame(x1, x2, x3, x4, stringsAsFactors=FALSE)

## matrix of distances among first obs. in xx and the remaining ones
Gower.dist(data.x=xx[1:6, ], data.y=xx[7:10, ])

## matrix of distances among observations in xx
row.names(xx) <- LETTERS[1:nrow(xx)]
dx <- Gower.dist(xx)
plot(hclust(dx))
```

Gradd *Classification grid and decision boundaries*

Description

Adds to the 2D ordination either colored points to make classification grid, or lines to show decision boundaries

Usage

```
Gradd(model2var, data2var, spacing=75, what="points",
 trnsp=0.2, pch=16, cex=0.8, lwd=2, lty=2, lcol="grey", palette=NULL,
 type="ids", User.Predict=function(model2var, X) {}, ...)
```

Arguments

model2var	Model based on 'data2var' (see below).
data2var	Data with _exactly_ 2 variables, e.g., result of PCA.
spacing	Density of points to predict.
what	What to draw: either "points" for classification grid, or "lines" for decision boundaries
trnsp	Transparency of points.
pch	Type of points.
cex	Scale of points.
lwd	Width of lines.
lty	Type of lines.
lcol	Color of lines.
palette	Palette to use.

type	Type of the model: "ids", "lda", "tree", or "user" (see examples).
User.Predict	Function to define in case of 'type="user"'.
...	Additional arguments to points() or contour().

Details

Gradd() takes model and its 2D data, makes new data with the same range but made of dense equidistantly spaced (grid-like) points, then predicts class labels from this new data, probably also calculates decision boundaries, and finally plots either points colored by predition, or lines along boundaries.

Before you run Gradd(), make the model. This model should have 'predict' method, and use ids (to make colors) and exactly 2 variables with names same as 'data2var' column names, e.g:

model2var <- somemodel(ids ~ ., data=cbind(ids, data2var))

If the model type is "user", the Gradd() uses predefined 'User.Predict(model2var, X)' function which must return factor ids from testing X data (see examples).

To plot both lines and grid, use Gradd() twice.

Gradd() is mainly a teching demo. It is useful if the goal is to illustrate the general properties of the supervised method and/or underlying data. It uses the entire 2D dataset to learn new data but learning from training subset is also possible, see the Naive Bayes example below.

Author(s)

Alexey Shipunov

Examples

```
## SVM:
library(e1071)
iris.p <- prcomp(iris[, -5])$x[, 1:2]
iris.svm.pca <- svm(Species ~ ., data=cbind(iris[5], iris.p))
plot(iris.p, type="n", main="SVM")
Gradd(iris.svm.pca, iris.p) # type="ids" (default)
text(iris.p, col=as.numeric(iris[, 5]), labels=abbreviate(iris[, 5], 1,
 method="both.sides"))

## LDA:
library(MASS)
iris.p <- prcomp(iris[, -5])$x[, 1:2]
iris.lda.pca <- lda(Species ~ . , data=cbind(iris[5], iris.p))
plot(iris.p, type="n", main="LDA")
Gradd(iris.lda.pca, iris.p, type="lda")
text(iris.p, col=as.numeric(iris[, 5]), labels=abbreviate(iris[, 5], 1,
 method="both.sides"))

## tree::tree() (note how to draw decision boundaries):
library(tree)
iris.p <- prcomp(iris[, -5])$x[, 1:2]
iris.tree.pca <- tree(Species ~ . , data=cbind(iris[5], iris.p))
plot(iris.p, type="n", main="tree")
```

```
Gradd(iris.tree.pca, iris.p, type="tree", what="lines")
text(iris.p, col=as.numeric(iris[, 5]), labels=abbreviate(iris[, 5], 1,
 method="both.sides"))

## randomForest:
library(randomForest)
iris.p <- prcomp(iris[, -5])$x[, 1:2]
iris.rf.pca <- randomForest(Species ~ ., data=cbind(iris[5], iris.p))
plot(iris.p, type="n", main="randomForest")
Gradd(iris.rf.pca, iris.p) # type="ids" (default)
text(iris.p, col=as.numeric(iris[, 5]), labels=abbreviate(iris[, 5], 1,
 method="both.sides"))

## naiveBayes (note how to use training subsample):
library(e1071)
iris.p <- prcomp(iris[, -5])$x[, 1:2]
sel <- 1:nrow(iris)
plot(iris.p, col=iris$Species, pch=ifelse(sel, 19, 1), main="naiveBayes")
iris.nb2 <- naiveBayes(Species ~ ., data=cbind(iris[5], iris.p)[sel, ])
Gradd(iris.nb2, iris.p[sel, ], what="lines")

## rpart (note how to use MDS for the base plot):
iris.dist <- dist(iris[, -5], method="manhattan")
iris.dist[iris.dist == 0] <- abs(jitter(0))
library(MASS)
iris.m <- isoMDS(iris.dist)$points
colnames(iris.m) <- c("Dim1", "Dim2")
library(rpart)
iris.rpart.mds <- rpart(Species ~ . , data=cbind(iris[5], iris.m))
plot(iris.m, type="n", main="rpart + MDS")
Gradd(iris.rpart.mds, iris.m, type="tree")
text(iris.m, col=as.numeric(iris[, 5]), labels=abbreviate(iris[, 5], 1,
 method="both.sides"))

## QDA:
library(MASS)
iris.p <- prcomp(iris[, -5])$x[, 1:2]
iris.qda.pca <- qda(Species ~ . , data=cbind(iris[5], iris.p))
plot(iris.p, type="n", main="QDA")
Gradd(iris.qda.pca, iris.p, type="lda")
text(iris.p, col=as.numeric(iris[, 5]), labels=abbreviate(iris[, 5], 1,
 method="both.sides"))

## nnet:
library(nnet)
iris.p <- prcomp(iris[, -5])$x[, 1:2]
iris.nnet.pca <- nnet(Species ~ . , data=cbind(iris[5], iris.p), size=4)
plot(iris.p, type="n", main="nnet")
Gradd(iris.nnet.pca, iris.p, type="tree")
text(iris.p, col=as.numeric(iris[, 5]), labels=abbreviate(iris[, 5], 1,
 method="both.sides"))

## kNN (note how to employ User.Predict()):
```

```
library(class)
iris.p <- prcomp(iris[, -5])$x[, 1:2]
plot(iris.p, type="n", main="kNN")
Gradd(cbind(iris[5], iris.p), iris.p, type="user",
 User.Predict=function(model2var, X) knn(model2var[, 2:3], X, model2var[, 1], k=5))
text(iris.p, col=as.numeric(iris[, 5]), labels=abbreviate(iris[, 5], 1,
 method="both.sides"))
```

`Gridmoon` *Draw with 'R'*

Description

Sraw with 'R'

Usage

```
Gridmoon(Skyres=50, Nightsky=TRUE, Daysky="deepskyblue", Moon=TRUE,
 Moonsize=0.05, Stars=TRUE, Hillcol="black", Text=c("Once upon a time..."),
 Textsize=22, Textpos=c(.15, .51), Textcol="white")
```

Arguments

`Skyres`	Sky resoluiton
`Nightsky`	If TRUE, ther eis a night
`Daysky`	Color of day sky
`Moon`	If TRUE, there is a moon
`Moonsize`	Moon size
`Stars`	If TRUE, there are stars
`Hillcol`	Hill color
`Text`	Text to print
`Textsize`	Text size
`Textpos`	Text position
`Textcol`	Text color

Details

'Gridmoon()' is an example how to paint (draw) with 'R'. Just for fun. From Murrell (2006) "R Graphics", with modifications.

Author's comments:

An example of a one-off image drawn using the grid system.

The code is somewhat modular and general, with functions for producing different shapes, but the sizes and locations used in this particular image assume a 2:1 aspect ratio.

The gradient-fill background (dark at the top to lighter at the bottom) is achieved by filling multiple overlapping polygons with slowly changing shades of grey.

Author(s)

Alexey Shipunov

References

Murrell P. 2006. R Graphics.

Examples

```
## Examples best viewed with 2:1 aspect ratio, use something like
## dev.new(width=10, height=5)
Gridmoon(Skyres=75)
Gridmoon(Nightsky=FALSE, Moon=FALSE, Stars=FALSE, Hillcol="forestgreen",
 Text="Use R!", Textcol="yellow", Textpos=c(.25, .85), Textsize=96)
```

`haltica` *Haltica flea beetles*

Description

Lubischew data (1962, pp. 460–461, table 2): 39 Haltica flea beetles specimens which belong to two cryptic species.

Sources of specimens:

Haltica oleracea:

1, 2, 3, 4, 6: Western Europe (Germany, France, Italy); 5: Leningrad; 7, 8: Perm; 11, 12: Kiev; 15, 16: Middle Volga (Kuibyshev); 17: Orel district, Middle Russia; 9, 10: Northern Caucasus; 13, 14, 18, 19: Transcaucasia (Delizhan, Akstafa).

H. carduorum:

6: Northern Russia (Elabuga); 1, 5, 9, 10, 11, 12, 14, 16, 20: Middle Russia (Penza, Orel, Voronezh districts); 3, 4, 17; Northern Caucasus; 2, 15, 18, 19: Black Sea Coast of the Caucasus; 13: Transcaucasia (Armenia); 7, 8: Middle Asia (Schachriziabs).

Usage

```
haltica
```

Format

These data frame contains the following columns:

`Species` Species epithet

`No` Number of sample (see below)

`x5` The distance of the transverse groove from the posterior border of the prothorax, in microns

`x14` The length of the elytram, in 0.01 mm

`x17` The length of the second antennal joint, in microns

`x18` The length of the third antennal joint, in microns

Source

Lubischew A.A. 1962. On the use of discriminant functions in taxonomy. Biometrics. 18:455–477.

Examples

```
plot(prcomp(haltica[, -(1:2)])$x[, 1:2], col=haltica$Species)

haltica.qj <- Classproj(haltica[, -(1:2)], haltica$Species, method="QJ")
plot(haltica.qj$proj, col=haltica$Species)
text(haltica.qj$centers, levels(haltica$Species), col=1:2)
```

`Hcl2mat` *Clustering to matrix*

Description

Converts clustering to matrix

Usage

```
Hcl2mat(hcl)
```

Arguments

`hcl`	`hclust` object

Details

This function converts 'hclust' object into binary matrix in accordance with clusterings.

It has many uses: clustering bootstrap, clustering compare, and matrix representation of hierarchical clustering.

See Also

`Bclust`, `MRH`

Examples

```
head(Hcl2mat(hclust(dist(iris[, -5]))))
```

Hclust.match *Counts matches between two hierarchical clusterings*

Description

Counts matches between two hierarchical clusterings

Usage

```
Hclust.match(hc1, hc2, scale=FALSE)
```

Arguments

hc1	First hclust object
hc2	Second hclust object
scale	Scale by the sum size of trees?

Details

'Hclust.match()' counts matches between two hierarchical clusterings (based on 'cutree()').

Result is a sort of consensus distances. Useful, for example, for clustering heatmaps.

Author(s)

Alexey Shipunov

Examples

```
aa.d1 <- hclust(dist(t(atmospheres)))
aa.d2 <- hclust(as.dist(1 - abs(cor(atmospheres, method="spearman"))),
 method="ward.D")
aa12.match <- Hclust.match(aa.d1, aa.d2)
heatmap(aa12.match, scale="none")
```

Hcoords *Calculates coordinates of nodes from 'hclust' plot*

Description

Takes the 'hclust' plot and calculates coordinates of ann internal nodes

Usage

```
Hcoords(hcl)
```

Arguments

hcl	hclust object

Details

This function calculates coordinates for each 'hclust' node. Inspired by pvclust::hc2axes().

Hcoords() is useful in connection with Bclust() family (namely, Bclabels()) and also separately. Since Hcoords() allows to label separate nodes, it can be used to label selected clusters (see examples).

See Also

Bclust

Examples

```
head(Hcoords(hclust(dist(iris[, -5]))))

## simple example: number all nodes
hcl <- hclust(UScitiesD, "ward.D2")
plot(hcl)
hcoo <- Hcoords(hcl)
text(hcoo, labels=1:nrow(hcoo), pos=1)

## complex example:
## find MCCN (Most Close Common Node)
## and label it
plot(hcl)
mat <- Hcl2mat(hcl)
nodes <- 1:nrow(mat) # nodes are rows
colnames(mat) <- hcl$labels
## take two tips and select those rows (nodes) where both present
sel1 <- rowSums(mat[, colnames(mat) %in% c("Denver", "Chicago")]) > 1
## MCCN is the node with both our tips but with the minimum of other tips
MCCN1 <- nodes[sel1][which.min(rowSums(mat[sel1, ]))]
text(hcoo[MCCN1, , drop=FALSE], labels="Eastern + Central", pos=1)
sel2 <- rowSums(mat[, colnames(mat) %in% c("Miami", "Chicago")]) > 1
MCCN2 <- nodes[sel2][which.min(rowSums(mat[sel2, ]))]
text(hcoo[MCCN2, , drop=FALSE], labels="Eastern", pos=1)
```

Histr *Histogram with overlaid curve*

Description

Histogram with overlaid normal curve or density, optionally with rug

Usage

```
Histr(x, overlay="normal", rug=FALSE, col="gray80", ...)
```

Arguments

x	numerical vector
overlay	type of curve to overlay, accepted values are "normal" and "density"
rug	if TRUE, will add rug plot
col	curve color
...	arguments to 'hist()'

Details

Histr() plots histogram with overlaid normal curve or density, optionally with rug. Based on analogous function from Stephen Turner's 'Tmisc' package.

Author(s)

Alexey Shipunov

See Also

hist, density, rnorm

Examples

```
x <- rnorm(1000, mean=5, sd=2)
Histr(x)
Histr(x, overlay="density")
Histr(x^2, overlay="density", rug=TRUE, breaks=50, col="lightblue2")
```

---

hrahn *Angiosperm families: morphological characters*

Description

This data originated from the Hansen and Rahn (1969) "punched cards" publication, and subsequent additions and corrections (Hansen and Rahn, 1972; Hansen and Rahn, 1979). Idea was to use paper cards with holes to assist identification of flowering plants (angiosperm) families. These cards were digitized (Duncan and Meacham, 1986) and then used in several multi-entry identification systems (for example, Duncan and Meacham, 1986; Ray, 1995; Families..., 2008).

But what was a sizeable task in 1980–1990s, now is only few hours of R programming. It is therefore quite easy to make such system with R, please see the example. The core function is only a few lines of code, everything else is the interface "bells and whistles". This example system is also applicable to any data with similar structure.

The 'hrahn' data can also be used for the purposes other than identification, for example, to assist in the morphological analysis of angiosperm families.

Comparing with original printed sources, the version used here misses supporting illustrations and some comments to characters. Comparing with digital sources, it was slightly modified, mostly to correct the imperfect digitization, and add some comments from the printed version (they are in lowercase).

One of comments is large but important so it is placed below as "Note I".

===

Note I. [concerning naming of perianth]

A. Perianth segments in 1 cycle or 2 cycles uniform in colour, size and shape.

B. coloured and petal-like ... all _petals_

BB. green (colourless if the plant is without chlorophyll) or dry and hyaline, glumaceous or scarious ... all _sepals_

AA. Perianth segments in 2 cycles different in colour, size or shape.

C. outer cycle ... _sepals_

CC. inner cycle ... _petals_

AAA. Perianth segments spirally arranged with a gradual transition in colour, size and shape from inner to outer segments: in these cases we have guarded against misinterpretations by stating _all_ segments as _sepals_ _and_ as _petals_. If there is a tendency to differentiation into sepals and petals, then the numbers judged by us to be interpretable as sepals are stated as such and in the same way for the petals.

===

The data is based on the family concepts and characters used in Melchior (1964), Hutchinson (1967) and Cronquist (1981). Therefore, family concepts might be different from those which are in use now. In the data, families in are given in accordance with classifications above so outputted list of families is not sorted alphabetically.

Usage

```
hrahn
```

Format

This is a list which contains two components:

`data` Binary matrix, row names are families, columns with 'chars'

`chars` Character vector with descriptions of characters, posititons correspond with columns of 'data'

Source

Cronquist A. 1981. An integrated system of classification of flowering plants. Columbia University Press, New York.

Duncan T., Meacham C.A. 1986. Multiple-entry keys for the identification of angiosperm families using a microcomputer. Taxon. 35: 492–494.

Families of Angiosperms: Punched Cards by Hansen and Rahn. 2008. eFloras. URL: http://www.efloras.org/flora_page.aspx' Missouri Botanical Garden, St. Louis, MO and Harvard University Herbaria, Cambridge, MA.

Hansen B., Rahn K. 1969. Determination of angiosperm families by means of a punched-card system. Dansk Botanisk Arkiv. 26: 1–46 + 172 punched cards.

Hansen B., Rahn K. 1972. Determination of angiosperm families by means of a punched-card system. Additions and corrections. I. Botanisk tidsskrift. 67: 152–163.

Hansen B., Rahn K. 1979. Determination of angiosperm families by means of a punched-card system. Additions and corrections. II. Botanisk tidsskrift. 74: 177–178.

Hutchinson J. 1967. Key to the families of flowering plants of the world. Clarendon Press, Oxford.

Melchior H. 1964. A. Engler's Syllabus der Pflanzenfamilien 12. II Band. Angiospermen. Gerbrueder Borntbaeger, Berlin, Nikolassee.

Ray Ph. 1995. Flowering plant family identification. URL: http://www.colby.edu/info.tech/BI211/info.html

Examples

```
data <- hrahn$data
chars <- hrahn$chars

showcharlist <- function(selchar) {
 tmp <- tempfile()
 selected <- ifelse(seq_along(chars) %in% selchar, "[X]", "[ ]")
 useful <- makeuseful(selchar)
 selected[useful] <- "[O]"
 write.table(data.frame(selected, seq_along(chars), chars),
  file=tmp, quote=FALSE, col.names=FALSE, row.names=FALSE)
 file.show(tmp)
}

makeuseful <- function(selchar) { # numbers of potentially useful characters
 selrows <- rowSums(data[, selchar, drop=FALSE]) == length(selchar)
 sums <- colSums(data[selrows, , drop=FALSE])
 seq_len(ncol(data))[sums > 0 & sums < sum(selrows)]
}

makefam <- function(selchar) { # the core function
 selrows <- rowSums(data[, selchar, drop=FALSE]) == length(selchar)
 row.names(data)[selrows]
}

displayfam <- function(selfam, howmany=12) { # display first "howmany" families
 if (is.null(selfam) || length(selfam) == 0) return("None")
 lfam <- length(selfam)
```

```
 if (lfam > howmany) {
 dfam <- selfam[seq_len(howmany)]
 res <- paste(c(dfam, paste0("and ", lfam-12, " more")), collapse=", ")
 } else {
 res <- paste(selfam, collapse=", ")
 }
 res
}

updatechar <- function(old, new) { # add or remove characters
 positive <- new[new > 0 & new <= length(chars)]
 old <- union(na.omit(old), positive)
 negative <- abs(new[new < 0])
 setdiff(old, negative)
}

displaydn <- function(num, sym="-") { # display numbers with dashes
 if (!is.numeric(num)) stop("Argument must be numeric")
 if (length(num) == 1) return(as.character(num))
 num <- sort(unique(num))
 if (length(num) == 2) return(paste(num, collapse=", "))
 num[abs(num - c(num[length(num)], num[-length(num)])) == 1 &
  abs(num - c(num[-1], num[1])) == 1] <- "-"
 gsub(", (-, )+", sym, paste(num, collapse=", "))
## slightly longer (but concatenates with +1 number):
## cc <- paste0(num, c(ifelse(diff(num) == 1, "-", ""), ""), collapse=", ")
## gsub("-, ", "-", gsub("-, (-*[0-9]+-, )+", "-", cc))
}

displaychar <- function(selchar) {
 if (is.null(selchar) || length(selchar) == 0) return("None")
 displaydn(selchar)
}

run <- function(howmany=12, selfam=NULL, selchar=NULL) { # interface, recursive function
if (!interactive()) return(cat("Please run in interactive mode\n"))
cat("Results:", displayfam(selfam, howmany=howmany), "\n")
cat("Selected characters:", displaychar(selchar), "\n")
cat("Potentially useful characters:", displaychar(makeuseful(selchar)), "\n")
cat("===\n")
cat("Type (character) numbers, separate with comma, negative numbers remove from selection\n")
cat("Type 'c' to see the list of characters, [X] selected, [O] potentially useful\n")
cat("Type any other single letter to exit\n")
cat("===\n")
x <- readline(prompt="Your choice: ")
while (TRUE) {
 if (x == "c") showcharlist(selchar)
 if (x %in% c(letters[-3], LETTERS)) break
 new <- suppressWarnings(as.integer(strsplit(x, split=",")[[1]]))
 selchar <- updatechar(selchar, new)
 selfam <- makefam(selchar)
 run(howmany=howmany, selfam=selfam, selchar=selchar)
 break
```

```
 }
}

run()
```

`Hulls` *Convex hulls for multiple groups*

Description

Calculates and plots groups hulls and related information

Usage

```
Hulls(pts, groups, match.colors=TRUE, usecolors=NULL,
 plot=TRUE, centers=FALSE, c.pch=0, c.cex=3,
 outliers=TRUE, coef=1.5, ...)
```

Arguments

`pts`	Data points to plot, 2-dimensional
`groups`	Grouping variable, any type
`match.colors`	Match colors with groups
`usecolors`	Which group colors to use (does not rotate)
`plot`	Plot?
`centers`	Show centers?
`c.pch`	Type of center points
`c.cex`	Scale of center points
`outliers`	Include outliers?
`coef`	Determines how to detect outliers, see 'coef' from 'boxplot.stats()'
`...`	Arguments to 'lines()'

Details

If 'centers=TRUE', Hulls() calculates centroids of polygons corrsponding with convex hulls.

If 'outliers=FALSE', Hulls() uses boxplot.stats() to detect outliers (points which are most distant from centers). This option could be used for cluster sharpening. It also automatically switches to 'centers=TRUE' so if you want to plot smoothed hulls but do not want to plot their centers, use something like 'c.pch=NA' or 'c.cex=0' (see examples).

Please also check Ellipses() function which uses confidence ellipses based on F-distribution.

Note that (at least at the moment), polygons are plotted with line() function, therefore shading is not straightforward (but possible, see examples).

Value

Invisibly outputs list of hulls (polygons) with coordinates, and possibly also with 'centers' and 'outliers' attributes. Indices of margin points returned as row names of each polygon.

See also package 'cluster' for ellipsoidhulls() function that allows to draw ellipse-like hulls.

Author(s)

Alexey Shipunov

See Also

Ellipses, Overlap, boxplot.stats

Examples

```
iris.p <- prcomp(iris[, -5])$x[, 1:2]
plot(iris.p, type="n", xlab="PC1", ylab="PC2")
pal <- rainbow(3)
text(iris.p, labels=abbreviate(iris[, 5], 1, method="both.sides"),
 col=pal[as.numeric(iris[, 5])])
Hulls(iris.p, iris[, 5], centers=TRUE, usecolors=pal)

## smoothed hulls
plot(iris.p, col=iris$Species, xlab="PC1", ylab="PC2")
ppts <- Hulls(iris.p, iris[, 5], centers=TRUE, outliers=FALSE, c.pch=NA)
## reveal outliers:
(out <- attr(ppts, "outliers"))
points(iris.p[out, ], pch=4, cex=1.4)

## this might complement Overlap()
cnts <- attr(ppts, "centers")
dist(cnts)
## how to use centers for clustering groups
plot(hclust(dist(cnts)))

## this is how to plot shaded hulls
plot(iris.p, pch=as.numeric(iris$Species))
for (i in seq_along(ppts))
 polygon(ppts[[i]], border=NA, col=adjustcolor(i, alpha.f=0.2))
```

---

hwc *hwc*

Description

Artificial data for teaching purposes.

Usage

```
hwc
hwc2
hwc3
```

Format

This data frame contains the following columns:

`COLOR` hair color

`WEIGHT` weight, kg

`HEIGHT` height, cm

Examples

```
## 'hwc' was made like (commands repeated until sd was around 3):
sd(VES.BR <- round(rnorm(30, mean=mean(70:90), sd=3)))
sd(VES.BL <- round(rnorm(30, mean=mean(69:79), sd=3)))
sd(VES.SH <- round(rnorm(30, mean=mean(70:80), sd=3)))
sd(ROST.BR <- round(rnorm(30, mean=mean(160:180), sd=3)))
sd(ROST.BL <- round(rnorm(30, mean=mean(155:160), sd=3)))
sd(ROST.SH <- round(rnorm(30, mean=mean(160:170), sd=3)))
data.frame(COLOR=rep(c("black", "blond", "brown"), each=30),
 WEIGHT=c(VES.BR, VES.BL, VES.SH), HEIGHT=c(ROST.BR, ROST.BL, ROST.SH))

## 'hwc2' is similar but 'sd' was not controlled so it is usually not homogeneous

## 'hwc3' was made like:
set.seed(1683)
VES.BR <- sample(70:90, 30, replace=TRUE)
VES.BL <- sample(69:79, 30, replace=TRUE)
VES.SH <- sample(70:80, 30, replace=TRUE)
ROST.BR <- sample(160:180, 30, replace=TRUE)
ROST.BL <- sample(155:160, 30, replace=TRUE)
ROST.SH <- sample(160:170, 30, replace=TRUE)
data.frame(COLOR=rep(c("black", "blond", "brown"), each=30),
 WEIGHT=c(VES.BR, VES.BL, VES.SH), HEIGHT=c(ROST.BR, ROST.BL, ROST.SH))
```

Infill *Rarefaction curves*

Description

Rarefaction curves

Usage

```
Infill(x, n=10)
## S3 method for class 'Infill'
plot(x, ...)
## S3 method for class 'Infill'
summary(object, ...)
```

Arguments

x	Data frame where columns are species
object	Object of the class "Infill"
n	Number of permutations
...	Arguments to 'plot()' or 'summary()'

Details

'Infill()' returns matrix to draw accumulation curves (each column is one curve).

'Infill' uses checklists of biological organisms to build rarefaction curves. You can estimate how many taxa will appear in the next sample to plan your investigations (e.g. revealing flora or fauna of the certain area).

If cells contain taxa abundance it will be automatically replaced with 1 or 0. Permutation is a random shuffle of the samples to get more valid estimation of the taxa accumulation process. It does not matter which sample appeared first. The resulting plot gives information on the process of taxa revealing during the investigation. High number of permutations gives more precise results, but the calculations are more slow. Empirically, 100 permutations are enough. The plot indicates full taxa number which has been accumulated in this and all the previous samples.

Value

Object of the class "Infill", or nothing

Author(s)

Alexey Shipunov, Eugeny Altshuler

References

Diaz-Frances E., Soberon J. 2005. Statistical estimation and model selection of species accumulation curves. Conservation Biology. Vol. 19, N 2. P. 569-573.

Gotelli N.J., Colwell R.C. 2001. Quantifying biodiversity: procedures and pitfalls in the measurement and comparison of species richness. Ecology Letters. Vol. 4. P. 379-391.

Soberon J.M., Llorente J.B. 1993. The use of species accumulation functions for the prediction of species richness. Conservation Biology. Vol. 7. N 3. P. 480-488.

Examples

```
x <- t(dolbli)
data <- x[1:45, ] # one of two lakes selected
data.I <- Infill(data)
summary(data.I)
plot(data.I)
```

Jclust *Simple bootstrap and jackknife clustering*

Description

Simple bootstrap and jackknife clustering

Usage

```
Jclust(data, n.cl, iter=1000, method.d="euclidean", method.c="ward.D",
 bootstrap=TRUE, monitor=TRUE)

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

## S3 method for class 'Jclust'
plot(x, main="", xlab=NULL, rect.lty=3, rect.col=1,
 rect.xpd=TRUE, top=FALSE, lab.pos=3, lab.offset=0.5, lab.col=par("col"),
 lab.font=par("font"), ...)
```

Arguments

data	Data
n.cl	Number of desired clusters
iter	Number of iterations, default 1000
method.d	Distance method
method.c	Hierarchical clustering method
bootstrap	Bootstrap or jackknife?
monitor	If TRUE (default), prints a dot for each replicate
x	Object of the class 'Jclust'
main	Plot title
xlab	Horizontal axis label
rect.lty	Line type for the rectangles
rect.col	Color of rectangles

rect.xpd	Plot rectangle sides if they go outside the plotting region?
top	Plot values on top?
lab.pos	Position specifier for the values text labels
lab.offset	Distance of the text labels in fractions of a character width
lab.col	Color of the text labels
lab.font	Font of the text labels
...	Additional arguments to the print() or plot.hclust()

Details

Simple method to bootstrap and jackknife cluster memberships, and plot consensus membership tree. Requires the desired number of clusters.

The default clustering method is the variance-minimizing "ward.D" (which works better with Euclidean distances); to make it coherent with hclust() default, specify 'method.c="complete"'.

Note that Jclust() is fast indirect bootstrap, it boostrap the consensus (not the original) tree and narrows results with the desired number of clusters. Please consider also Bclust() which is the direct method, and phylogeny-based BootA().

Value

Returns 'Jclust' object which is a list with components "meth" (bootstrap or jacknife), "mat" (matrix of results, consensus matrix), "hclust" (consensus tree as 'hclust' object), "gr" (groups), "supp" (support values), "iter" (number of iterations) and "n.cl" (number of cluters used.)

Author(s)

Alexey Shipunov

See Also

Bclust,

BootA,

Fence

Examples

```
## 'moldino' data, 1000 iterations
(mo.j <- Jclust(t(moldino), n.cl=3, iter=1000))
plot(mo.j)

## adjust locations of value labels
data.jb <- Jclust(t(atmospheres), method.c="complete", n.cl=3)
plot(data.jb, top=TRUE, lab.pos=1, lab.offset=1, lab.col=2, lab.font=2)

## plot together with Fence()
iris.jb <- Jclust(iris[, -5], n.cl=3)
```

```
plot(iris.jb, labels=FALSE)
Fence(iris.jb$hclust, iris$Species)
legend("topright", legend=levels(iris$Species), col=1:3, lwd=2.5, bty="n")


## This is how one can bootstrap _all_ reliable cluster numbers:
for (i in 2:(nrow(t(moldino)) - 1)) print(Jclust(t(moldino), i, iter=1000, boot=TRUE))
```

K *Coefficient of divergence*

Description

Lubischew's coefficient of divergence (SSMD^2)

Usage

```
K(x, y=NULL, data=NULL, mad=FALSE, na.rm=TRUE)
## S3 method for class 'K'
print(x, ...)
## S3 method for class 'K'
summary(object, ..., num=2)
```

Arguments

x	Numeric vector, or formula, or object of the class 'K'
y	Second numeric vector, or nothing
data	Data with two columns (in case of formula)
mad	Non-parametric variant of K (not Lubischew's)
na.rm	Remove NAs?
object	Object of the class 'K'
num	Digits to round
...	Additional arguments

Details

One of the effect size measures, Lubischew's K, coefficient of divergence (Lubischew, 1959). Interestingly, the recently invented "striclty standardized mean difference" SSMD (see, for example, "https://en.wikipedia.org/wiki/Strictly_standardized_mean_difference") is just a square root of K.

Value

K() returns value of K, or nothing. summary.K() returns also magnitude and P, "probability of misclassification".

Author(s)

Alexey Shipunov

References

Lubischew A. A. 1959. How to apply biometry to systematics. Leningrad University Herald. N 9. P. 128–136. [In Russian, English abstract].

Examples

```
K(1:3, 2:100)
sapply(eq[, -1], function(.x) K(.x ~ eq[, 1]))
summary(K(x17 ~ Species, data=haltica), num=5)
```

---

keys *Diagnostic keys*

Description

Diagnostic keys are data structures which help to identify biological samples, i.e. give them (scientific) names. They are old but still very popular because they are simple and efficient, sometimes even for not very experienced user.

The second goal of these keys is the compact representation of biological diversity. Diagnostic keys are not very far from classification lists (see 'classifs'), phylogeny trees (like 'phylo' objects in 'ape' package), from core R 'dendrogram' and 'hclust' objects, and especially from recursive partitioning objects (e.g., from 'tree' or 'rpart' packages).

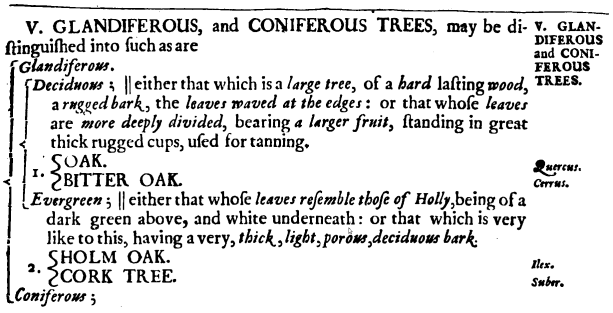
In biology, diagnostic keys exist in many flavors which are possible to reduce into two main types:

I. Branched keys, where alternatives are separated.

You compare your sample with the first description. Then, if the sample agrees with first description, you go to second description (these keys are usually fully dichotomous), then to the third, until you reach the temninal (name of the organism). If not, you find the alternative description of the _same level_ (same depth). The main difficulty here is how to find it.

To help user find descriptions of the same depths, branched keys are usually presented as _indented_ where each line starts with an indent. Bigger indent means bigger depth.

Branched or indented keys could be traced at least to 1668, to one of John Wilkins books:



V. GLANDIFEROUS, and CONIFEROUS TREES, may be distinguiſhed into ſuch as are

Glandiferous.

Deciduous; ‖ either that which is a *large tree*, of a *hard* laſting *wood*, a *rugged bark*, the *leaves waved at the edges*: or that whoſe *leaves* are *more deeply divided*, bearing *a larger fruit*, ſtanding in great thick rugged cups, uſed for tanning.

1. OAK. *Quercus.*
1. BITTER OAK. *Cerrus.*

Evergreen; ‖ either that whoſe *leaves reſemble thoſe of Holly*, being of a dark green above, and white underneath: or that which is very like to this, having a very, *thick*, *light*, *porous*, *deciduous bark*.

2. HOLM OAK. *Ilex.*
2. CORK TREE. *Suber.*

Coniferous;

V. GLANDIFEROUS and CONIFEROUS TREES.

(and maybe to much earlier scholastic works.)

Indented keys are widely used, especially in English-language publications.

Another modification could be traced to 1892 when A. Semenow-Tjan-Shanskij published his serial key:

1 (2). *Vertex tuberculo fere corniformi in utroque sexu praeditus. Mandibulae in utroque sexu simplices, i. e. absque appendicibus.*

sg. **Abrognathus** Jak.

2 (1). *Vertex absque tuberculo corniformi.*

3 (4). *Mandibularum appendices in ♂ saepius vix indicatae, dentiformes seu obtuse anguliformes, rarius magis evolutae, spiniformes, semper aequales. Corpus supra rude sculptum.*

sg. **Lethrulus** m.

4 (3). *Mandibularum appendices in ♂ plus minusve evolutae (saltem sinistra), modo aequales, modo inaequales.*

Serial keys are similar to all branched keys but numbering style is different. All steps are numbered sequentially but each has a back-reference to the alternative so user is not required to find the description of the same depth, they are already here. Serial keys are strictly dichotomous. They are probably the most space-saving keys, and still in use, especially in entomology.

II. Bracket keys, where alternatives are together, and user required to use 'goto' references to take the next step.

They can be traced to the famous "Flora Francoise" (1778) where J.-B. Lamarck likely used them the first time:

9. *Fleurs libres & non réunies dans un calice commun*	Corolle monopétale. .	10
	Corolle polypétale...	13
10. *Corolle monopétale..*	Corolle régulière....	11
	Corolle irrégulière...	12
11.	Corolle régulière.	
	Anagallis arvenſis.	
12.	Corolle irrégulière.	
	Salvia pratenſis.	

You compare your sample with first description, and if it agrees, go to where 'goto' reference says. If not, go to second (alternative) description, and then again use its 'goto'. On the last steps, 'goto'

is just the terminal, the name you want. Sometimes, bracket keys have more than one alternative (e.g., not fully dichotomous).

Bracket keys pose another difficulty: it is not easy to go back (up) if you by mistake went into the wrong direction. Williamson (1922) proposed backreferenced keys where each step supplied with back-reference:

1.	Tarsi spurred	2.
1′.	Tarsi not spurred	5.
2 (1).		*a.*
2′.		3.
3 (2′).		
		4.
3′.		
		b.
4 (3).		*c.*
4′.		*d.*

Sometimes, back-references exist only in case where the referenced step is not immediately before the current.

Bracket keys (backreferenced or not) are probably most popular in biology, and most international as well.

Here bracket, branched and serial keys are standardized as rectangular tables (data frames). Each feature (id, backreference, description, terminal, 'goto') is just one column. In bracket keys, terminal and 'goto' are combined. For example, if you need a bracket key without backreferences, use three columns: id, description and terminal+'goto'. Order of columns is important, column name is not. Please see examples to understand better.

Note that while this format is human-readable, it is not typographic. To make keys more typographic, user might want to convert them into LaTeX where several packages allow for typesetting diagnostic keys (for example, my 'biokey' package.)

Usage

```
keys
```

Format

The list which contains four data frames representing three different flavors of biological diagnostic keys: two simple bracket keys, one branched (indented variant) and one serial key. Last two keys are real-world keys, first to determine Plantago (ribworts, plantains) from European Russia (Shipunov, 2000), second – from North America (Shipunov, 2019).

Source

Lamarck J.-B., de. 1778. Flore Francoise. Paris.

Semenow-Tjan-Shanskij A.P. 1892. Note sur la subdivision du genre Lethrus Scop. et description de deux nouvelles. Trudy Russkago Entomologicheskago Obschestva. 26: 232–244.

Shipunov A. 2000. The genera Plantago L. and Psyllium Mill. (Plantaginaceae Juss.) in the flora of East Europe. Novosti Systematiki Vysshikh Rastenij. 32: 139–152. [In Russian]

Shipunov A. 2019. Plantago. In: Freeman, C. and Rabeler R. (eds.) Flora of North America. 2019. 17: 280–293. Oxford University Press, New York and Oxford.

Shipunov A. 2019. biokey – Flexible identification key tables in LaTeX. Version 3.1. See "https://ctan.org/pkg/biokey".

Sviridov A.V. 1994. Types of the biodiagnostic keys and their uses. Moscow. [In Russian]

Wilkins J. 1663. An essay towards the real character and philosophical language. London.

Williamson E. 1922. Keys in systematic work. Science. 55: 703.

See Also

Biokey

Examples

```
attach(keys)

head(bracket1)
head(bracket2)
head(branched)
head(serial)

## convert keys with Biokey()
sii <- Biokey(serial, from="serial", to="indented")
sbb <- Biokey(serial, from="serial", to="bracket")
bbr <- Biokey(branched, from="branched", to="bracket")

## convert keys and visualize them as trees
library(ape) # load 'ape' library to plot Newick trees
plot(read.tree(text=Biokey(bracket1, from="bracket", to="newick")))
plot(read.tree(text=Biokey(bracket2, from="bracket", to="newick")))
plot(read.tree(text=Biokey(branched, from="branched", to="newick")))
plot(read.tree(text=Biokey(serial, from="serial", to="newick")))

detach(keys)

## to make a new bracket key (without backreferences)
## supply three columns: id, description and 'goto'+terminal
bracket3 <- read.table(as.is=TRUE, text="
1 Small Ant
1 Big 2
2 Blue Sky
2 Green Grass
")
bracket3
Biokey(bracket3, from="bracket", to="newick")
cophenetic(ape::read.tree(text=Biokey(bracket3, from="bracket", to="newick")))
```

`Life` *Game of Life*

Description

Conway's Game of Life

Usage

```
Life(n.rows=40, n.cols=40, n.cycles=100, sleep.time=0.12,
 cols=c("#f0f0f0", "#2f81c1"), random=TRUE, rnd.threshold=0.3)
```

Arguments

`n.rows`	Number of rows
`n.cols`	Number of columns
`n.cycles`	Number of cycles
`sleep.time`	Time for pause after each cycle
`cols`	Main colors
`random`	If FALSE, runs in the interactive mode
`rnd.threshold`	0 empty board; 1 all squares are filled

Details

In the interactive mode (random=FALSE), left click to define or remove cells, then click on red square in the bottom left corner to start cycles. Click positions are rounded so they are not always precise.

To stop cycles, use Ctrl-C (Linux, macOS) or Esc (Windows) in the main R window.

The code was inspired by the Github gist (which is not available anymore) attributed to Vadim Vinichenko. Note that margins influence the behavior of cells, i.e., the field is not infinite as in the "classic" Game of Life.

Author(s)

Alexey Shipunov

References

Gardner M. 1970. The fantastic combinations of John Conway's new solitaire game "life". Scientific American. 223: 120–123.

Examples

```
Life(n.cols=10, n.rows=10, n.cycles=10, sleep.time=0.3)
```

`Linechart` *Dotchart-like plot sfor every scaled variable grouped by factor*

Description

Dotchart-like plot for every scaled variable grouped by factor

Usage

```
Linechart(vars, groups, xticks=TRUE, xmarks=TRUE, mad=FALSE, pch=19,
 se.lwd=1, se.col=1, ...)
```

Arguments

`vars`	Variables to draw (data frame)
`groups`	Grouing factor
`xticks`	Show xticks?
`xmarks`	Show xmarks?
`mad`	Show MAD instead of IQR?
`pch`	Points type
`se.lwd`	Lines width
`se.col`	Lines color
`...`	arguments to 'plot()'

Details

Linechart() is dotchart-based plot which shows medians and IQRs (or MADs) for every scaled variable grouped by 'groups' factor.

Alternatives: trellis designs.

Author(s)

Alexey Shipunov

See Also

Boxplots

Examples

```
Trees <- trees
Trees[, 4] <- sample(letters[1:3], nrow(Trees), replace=TRUE)
Linechart(Trees[, 1:3], factor(Trees[, 4]))

Linechart(iris[, 1:4], iris[, 5])
```

`Ls` *Ls*

Description

Advanced object browser

Usage

```
Ls (pos = 1, pattern, mode = "any", type = "any", exclude = "function", sort = "name")
```

Arguments

`mode`	which object mode to include, "any" to include all
`type`	which object type to include ("type" is typically, but not always an object's class attribute), "any" to include all
`exclude`	exclude functions (default), "none" to include all
`sort`	sort by name (default), "size" to sort by size
`pos`	specify environment, passed to ls()
`pattern`	optional regular expression, passed to ls()

Details

Based on 'ls()' but outputs data frame.

Value

Data frame with object features columns.

Author(s)

Alexey Shipunov

See Also

`ls`

Examples

```
data(trees)
Ls()
```

Mag *Interpreter for effect sizes*

Description

Interprets R^2-related effect sizes

Usage

```
Mag(x, squared=TRUE)
```

Arguments

x	Value
squared	Is value squared?

Details

Interpreter for R^2-related effect sizes (see example).

Author(s)

Alexey Shipunov

Examples

```
aa <- apply(cor(trees), 1:2, function(.x) Mag(.x, squared=FALSE))
aa[upper.tri(aa, diag=TRUE)] <- "-"
noquote(aa)
```

MDSv *MDS: dimension importance ("explained variance" surrogate)*

Description

Calculates R-squared coefficients of the linear relationships between each of derived variables and original data

Usage

```
MDSv(scores)
```

Arguments

scores	Data frame or matrix with values (e.g., result of 'isoMDS()'

Details

MDSv() converts each of the derived variables and original data into distance matrices, and then uses lm() to calculate adjusted R-squared coefficients. These coefficients may be used to understand the "importance" of each new dimension. They work for any dimension reduction techique including multidimensional scaling.

Value

Numeric vector, one values per column of scores

Author(s)

Alexey Shipunov

Examples

```
iris.dist <- dist(unique(iris[, -5]), method="manhattan")

library(MASS)
iris.m <- isoMDS(iris.dist)
(vv <- MDSv(iris.m$points)) # MDS explained variance surrogate
xlab <- paste0("Dim 1 (", round(vv[1], 2), "%)")
ylab <- paste0("Dim 2 (", round(vv[2], 2), "%)")
plot(iris.m$points, col=as.numeric(iris$Species), xlab=xlab, ylab=ylab)

iris.cmd <- cmdscale(iris.dist)
MDSv(iris.cmd)

iris.p <- prcomp(iris[, -5])
MDSv(iris.p$x)
100*summary(iris.p)$importance[2, ] # compare with MDSv() results
```

Miney *Miney game*

Description

Minesweeper game

Usage

```
Miney(n, ucol="#b8ff73", gcol="#f0f0f0", bcol="red", space=0.05, pbombs=0.15)
```

Arguments

n	Size of the field to play, i.e. n=9 (default) is 9 x 9 field.
ucol	Color of unknown cells, default is "law green"
gcol	Color of good cells, default is gray
bcol	Color of bad cells
space	Space between cells
pbombs	Proportion of cells with bombs

Details

Heavily modified from 'Miney::miney()' of Roland Rau. See also the fun::mine_sweeper() function.

Author(s)

Alexey Shipunov

Examples

```
## Not run:
## interactive command:
Miney(3)

## End(Not run)
```

Misclass *Misclassification (confusion) table*

Description

Misclassification (confusion) table

Usage

```
Misclass(pred, obs, best=FALSE, ignore=NULL, quiet=FALSE, force=FALSE, ...)
```

Arguments

pred	Predicted class labels
obs	Observed class labels
best	Perform a search for the classification table with minimal misclassification error?
ignore	Vector of class labels to ignore (convert into NAs)
quiet	Output summary?
force	Override the restriction of class number in 'best=TRUE' and speed up code?
...	Arguments to 'table'

Details

'Misclass()' produces misclassification (confusion) 2D table based on two classifications.

The simple variant ('best=FALSE') assumes that class labels are concerted (same number of corresponding classes).

Advanced variant ('best=TRUE') can search for the best classification table (with minimal misclassification rate), this is especially useful in case of unsupervised classifications which typically return numeric labels. It therefore assumes that the table is a result of some non-random process. However, internally it generates all permutations of factor levels and could be very slow if there are 8 and more class labels. Therefore, more than 8 classes are not allowed. It is possible nevertheless to override this restriction with 'force=TRUE'; this option also uses the experimental code which replaces internal table() with tabulate() and is much faster with many labels.

Variant with 'best=TRUE' might also add empty rows (filled with zeros) to the table in case if numbers of classes are not equal.

Additional arguments could be passed for table(), for example, 'useNA="ifany"'. If supplied data contains NAs, there will be also note in the end. Note that tabulate()-based code (activated with force="TRUE") does not take table()-specific arguments, so if this is a case, warning will be issued.

It is possible to ignore (convert into NAs) some class labels with 'ignore=...', this is useful for methods like DBSCAN which output special label for outliers. In that case, note about missing data is also issued.

Alternatives: confusion matrix from caret::confusionMatrix() which is more feature rich but much less flexible. See in examples how to implement some statistics used there.

Note that partial "Misclassification errors" are reverse sensitivities, and "Mean misclassification error" is a reverse accuracy.

If you want to plot misclassification table, Cohen-Friendly association plot, assocplot() is probably the best. On this plot, note rectangles which are big, tall and black (check help(assocplot) to know more). Diagonal which is black and other cells red indicate low misclassification rates.

Value

Invisibly returns the table of class comparison

Author(s)

Alexey Shipunov

See Also

Adj.Rand, link{assocplot}

Examples

```
iris.dist <- dist(iris[, -5], method="manhattan")
iris.hclust <- hclust(iris.dist)
iris.3 <- cutree(iris.hclust, 3)
Misclass(iris.3, iris[, 5])

set.seed(1)
```

```
iris.k <- kmeans(iris[, -5], centers=3)
Misclass(iris.k$cluster, iris[, 5])
Misclass(iris.k$cluster, iris[, 5], best=TRUE)

res <- Misclass(iris.k$cluster, iris[, 5], best=TRUE, quiet=TRUE)
## how to calculate statistics from caret::confusionMatrix()
binom.test(sum(diag(res)), sum(res))$conf.int
mcnemar.test(res) # to avoid NA's, add small number to 'res'
## how to plot misclassification table
assocplot(res)
## how to use Misclass() for Recode()
nn <- Recode(iris.k$cluster, from=dimnames(res)$pred, to=dimnames(res)$obs)
head(nn)

library(dbscan)
iris.db <- dbscan(iris[, -5], eps=0.3)
Misclass(iris.db$cluster, iris$Species, ignore=0, best=TRUE)

set.seed(NULL)
```

`Missing.map` *Textual plot of missing data*

Description

Textual plot of missing data

Usage

```
Missing.map(df)
```

Arguments

`df`	Data frame with any data

Details

'Missing.map()' makes textual plot of missing data, inspired by 'DescTools::PlotMiss()'.

Author(s)

Alexey Shipunov

Examples

```
Missing.map(salix_leaves)
```

moldino *moldino*

Description

Observations on island floras. Islands are located in the freshwater Moldino lake, Middle Russia. Data collected in 2013.

'moldino_l' contains squares and GPS locations.

'moldino' contains the actual abundance data.

Usage

```
moldino
```

Format

`columns` Island names, data is abundance of plant species, in 1543 scale (0 – absent; 1 – one individual plant; 2 – no more than 12 individual plants (rametes); 3 – number of individuals is more than 12 but no more than 5% of total number of plants on a plot; 4 – number of individuals is more than 5% but no more than 25% of total number of plants on a plot; 5 – number of individuals is more than 25% but no more than 50% of total number of plants on a plot; 6 – number of individuals is more than 50% but no more than 75% of total number of plants on a plot; 7 – number of individuals is more than 75% of total number of plants on a plot.)

`rows` Names of plant species

`NAME` Island name

`SQUARE` Island square, m2

`LAT` Latitude

`LON` Longitude

Source

Abramova L., Volkova P., Eliseeva E., Troshina A., Shipunov A. 2005–inward. The checklist of flora from environs of village Polukarpovo (Tver region). See "http://ashipunov.info/shipunov/moldino/nauka/molflora.pdf".

Shipunov A., Abramova L. 2014. Islands in lakes and the sea: how do they differ? European Journal of Environmental Sciences. 4: 112–115.

`MrBayes` *Calls MrBayes*

Description

A slight improvement of 'ips::mrbayes()'

Usage

```
MrBayes(x, file="", nst=6, rates="invgamma", ngammacat=4, nruns=2, ngen=1e+06,
 printfreq=100, samplefreq=10, nchains=4, savebrlens="yes", temp=0.2, burnin=10,
 contype="allcompat", run=FALSE, simple=TRUE, exec="mb-mpi", method="dna")
```

Arguments

`x`	The object to process (must be 'DNAbin' class)
`file`	A character string, giving the name of the MrBayes input file
`nst`	An integer giving the number of rates in the model of sequence evolution
`rates`	A character string; allowed are "equal", "gamma", "propinv", "invgamma", and "adgamma"; the default is "equal"
`ngammacat`	An integer; the number rate categories for the discretized Gamma distribution; the default is '4'
`nruns`	An integer; the number of runs
`ngen`	An integer; the number of states of the MCMC
`printfreq`	An integer; the interval between states of the MCMC to be printed on the screen
`samplefreq`	An integer; the interval between states of the MCMC to be sampled
`nchains`	An integer; number of Metropolis coupled MCMCs in each run
`savebrlens`	Logical; shall branch lengths be saved
`temp`	Heating parameter
`burnin`	An integer; the number of samples from the MCMC to be discarded prior to further analysis
`contype`	A character string; the type of consensus tree calculated from the posterior distribution of treeseither "halfcompat" (majority-rule consensus tree) or "allcombat" (strict consensus tree)
`run`	Logical; 'run = FALSE' will only print the NEXUS file, 'run = TRUE' will also start the MCMC runs, if the 'path' argument is correctly specified
`simple`	New option: if TRUE (default), then outputs tree in the format readable by functions from 'ape' package
`exec`	New option: name of UNIX executable (to allow multi-threaded version)
`method`	New option: either "dna", or "mixed" to handle mixed or purely morphologic data (see below)

Details

MrBayes() is an improvement of ips::mrbayes() and ips::mrbayes.mixed(). Please see its documentation for clarity and other options.

Comparing with 'ips' sources, MrBayes() has some code alterations and three more options. It also both views and saves output (works only on UNIX).

If 'method="mixed"', the function requires character matrix as input where missing data are labeled with "N", morphological columns encoded as 0/1 and placed after nucleotide columns (which might be absent).

Author(s)

Alexey Shipunov

See Also

ips::mrbayes

Examples

```
require(ips)
data(ips.cox1)
x <- ips.cox1[, 100:140]

## Not run:
## requires MrBayes program installation
MrBayes(x, file="cox1", ngen=100, run=TRUE)

str(plantago)
plantago[is.na(plantago)] <- "N"
row.names(plantago) <- gsub(" ", "_", row.names(plantago))
## requires MrBayes program installation
tr <- MrBayes(plantago, file="plantago", method="mixed", burnin=5000, run=TRUE) # makes many files
tr <- tr[[1]]
tr <- root(tr, outgroup="Plantago_maritima", resolve.root=TRUE)
tr$node.label <- suppressWarnings(round(as.numeric(tr$node.label)*100)) # warning is OK
tr$node.label[tr$node.label == "NA"] <- ""
plot(tr)
nodelabels(tr$node.label, frame="none", bg="transparent", adj=-0.1)
add.scale.bar()

## End(Not run)
```

MRH *Matrix Representation of Hierarchical Clustering*

Description

Matrix Representation of Hierarchical clustering (MRH)

Usage

```
MRH(hcl, dim=NULL, method="groups")
```

Arguments

hcl	'hclust' object
dim	Number of desired dimensions, if defaults are not suitable
method	Either "groups" (default), or "height", or "branches", or "cophenetic" (see below for explanations)

Details

This function calls cutree(), or Hcl2mat(), or cmdscale(cophenetic()) in order to output the Matrix Representation of Hierarchical clustering (MRH).

If method="groups" then clustering tree is cut by all possible numbers of clusters 'k' (excluding 'k=1' and 'k=n' which bring no information) so 'dim' is always 'n-2'.

If method="height" then clustering tree is cut by equally spaced agglomeration heights (excluding minimal and maximal heights which bring no information). Default 'dim' here is '2*n', but higher values might work even better.

If method="branches" then use Hcl2mat() to transform object into the binary matrix of memberships, always with 'n-1' dimensions (so user-specified 'dim' is not taken into account). Each column in this matrix represents the tree branch.

If method="cophenetic" then multidimensional scaling scores with maximum dimensionality on cophenetic distances are computed. Default 'dim' is 'n-1' but lesser numbers might work better.

The main feature of the resulted matrices is that they provide the "bridge" of conversion between original data, distance matrices and clustering (including phylogenetic trees) results. After conversion, many interesting applications become possible. For example, if converted trees represent the _same_ objects, it is possible to "hyper-bind", or "average" (Ashkenazy et al., 2018) them.

To work with 'phylo' objects, convert them first to 'hclust' with as.hclust(), and before that, possibly also apply compute.brlen(), multi2di() and collapse.singles().

Value

Matrix with default number of columns equal to number of objects (n) minus 1 (method="branches" or method="cophenetic") or 'n-2' (method="groups"), or '2*n' (method="height").

Rows are objects, values are either cluster numbers (method="groups" or method="height") so matrix consist of whole positive numbers, binary cluster memberships (method="branches") or decimal MDS scores (method="cophenetic").

References

Ashkenazy H., Sela I., Levy Karin E., Landan G., Pupko T. 2018. Multiple sequence alignment averaging improves phylogeny reconstruction. Systematic Biology. 68: 117–130.

See Also

cutree, link{cmdscale}, link{Hcl2mat}

Examples

```
aa.h <- hclust(dist(t(atmospheres)))
plot(aa.h)

(aa.mrh1 <- MRH(aa.h))
plot(hclust(dist(aa.mrh1)))

aa.mrh2 <- MRH(aa.h, method="height", dim=100) # here 'dim' should better be large
str(aa.mrh2)
plot(hclust(dist(aa.mrh2)))

plot(hclust(dist(cbind(aa.mrh1, aa.mrh2)))) # hyper-bind

(aa.mrh3 <- MRH(aa.h, method="branches"))
plot(hclust(dist(aa.mrh3)))

(aa.mrh4 <- MRH(aa.h, method="cophenetic"))
plot(hclust(dist(aa.mrh4)))

library(ape)
tree <- read.tree(text="((A:1,B:1):2,(C:3,D:4):2):3;")
(tree.mrh3 <- MRH(as.hclust(compute.brlen(tree)), method="branches"))
```

Normality *Check normality*

Description

Check normality through Shapiro-Wilks test

Usage

```
Normality(x, p=0.05)
```

Arguments

x	numerical vector
p	level of significance

Details

Normality via Shapiro-Wilks test. Kolmogorov-Smirnov is apparently too weak for small samples. The word of caution: this function only *helps* to decide if the data complains with parametric methods ("normal").

Value

Character vector of length one.

Author(s)

Alexey Shipunov

See Also

qqnorm, hist, rnorm

Examples

```
Normality(rnorm(100))
sapply(trees, Normality)
```

Overlap *Calculates overlap between polygons*

Description

Calculates overlaps between polygons (typically, convex hulls or confidence ellipses from some scatterplot). Requires 'PBSmapping' package.

Usage

```
Overlap(ppts, symmetric=FALSE, negative=FALSE)
## S3 method for class 'Overlap'
summary(object, ...)
```

Arguments

ppts	List with hulls information (e.g., output from Hulls())
symmetric	Make overlaps symmetric (like in distance matrix)?
negative	Calculate "negative overlaps" (relative distance between non-overlapped hulls)?
object	Object of the class 'Overlap'
...	Additional arguments

Details

The main idea of Overlap() is to provide the measurement of the separation between groups in 2D space.

Overlap() employs calculations of areas of polygons and their intersects provided by 'PBSmapping' package. Initially, it was based on the code provided by J. Oksanen for his "ordihulldist" function.

By default, overlaps are asymmetric, so overlap between a and b is not necessarily equal to the overlap between b and a. If 'symmetric=TRUE', then Overlap() will calculate symmetric overlaps, less precise but more suitable, e.g., for interpreting overlaps as distances.

When 'negative=TRUE', Overlap() calculates also negative polygon-based distances between non-overlapping polygons. They are symmetric and might be used as similarities too (please look on examples).

summary.Overlap() provides some general numbers, including mean and total overlaps for each hull. In these calculations, hulls without overlaps are ignored. Note that summary.Overlap() calculates the arithmetic, not geometric, mean (whereas symmetric Overlap() uses geometric mean). The average of all overlaps could serve as the reliable measure of the quality of dimension reduction.

Please also check out vegan::ordiareatest() function; this studies the one-side hypothesis that actuall hull areas are smaller than with randomized groups (i.e., that actuall hulls are better than random).

Value

Object (square matrix) of class 'Overlap', or nothing.

Author(s)

Alexey Shipunov

References

Serebryanaya A., Shipunov A. 2009. Morphological variation of plants on the uprising islands of northern Russia. Annales Botanici Fennici. 2009. 46: 81-89.

See Also

Hulls, Ellipses

Examples

```
iris.p <- prcomp(iris[, -5])$x[, 1:2]
iris.h <- Hulls(iris.p, iris$Species, plot=FALSE)

Overlap(iris.h)
Overlap(iris.h, negative=TRUE)
Overlap(iris.h, symmetric=TRUE)

(iris.o <- Overlap(iris.h, symmetric=TRUE, negative=TRUE))
as.dist(1 - iris.o) # how to convert overlaps into distance-like objects

summary(Overlap(iris.h))
summary(Overlap(iris.h, negative=TRUE))
summary(Overlap(iris.h, symmetric=TRUE))
summary(iris.o)

iris.e <- Ellipses(iris.p, iris$Species, plot=FALSE, centers=TRUE)
Overlap(iris.e, negative=TRUE)
```

`pairwise.Eff` *Pairwise table of effects with magnitudes*

Description

Pairwise table of effects with magnitudes

Usage

```
pairwise.Eff(vec, fac, eff="K", dec=2, mad=FALSE)
```

Arguments

`vec`	Values
`fac`	Groups
`eff`	Effect, either 'K' or 'cohen.d', or 'cliff.delta'
`dec`	Decimals to round
`mad`	Use MAD-based nonparametric modification of K?

Details

Pairwise table of effect sizes.

At the moment, classic Lyubischev's K (a.k.a. SSSMD), effsize::cliff.delta() and effsize::cohen.d() are supported.

Value

List with test outputs.

Author(s)

Alexey Shipunov

Examples

```
pairwise.Eff(hwc$WEIGHT, hwc$COLOR)
pairwise.Eff(hwc$WEIGHT, hwc$COLOR, mad=TRUE)
pairwise.Eff(hwc$WEIGHT, hwc$COLOR, eff="cohen.d")
pairwise.Eff(hwc$WEIGHT, hwc$COLOR, eff="cliff.delta")
```

`pairwise.Rro.test` *Robust rank order test post hoc derivative*

Description

Robust rank order test post hoc derivative

Usage

```
pairwise.Rro.test(x, g, p.adjust.method="BH")
```

Arguments

`x`	Values
`g`	Groups
`p.adjust.method`	See '?p.adjust'

Details

'pairwise.Rro.test()' is the Robust rank order test post hoc derivative.

Value

List with test outputs

Author(s)

Alexey Shipunov

See Also

`Rro.test`

Examples

```
pairwise.Rro.test(airquality$Ozone, airquality$Month)
```

`pairwise.Table2.test` *Pairwise Chi-squared or Fisher test for 2-dimensional tables*

Description

Pairwise Chi-squared or Fisher test for 2-dimensional tables

Usage

```
pairwise.Table2.test(tbl, names=rownames(tbl), p.adjust.method="BH", exact=FALSE, ...)
```

Arguments

`tbl`	Contingency table
`names`	Level names
`p.adjust.method`	See '?p.adjust'
`exact`	Run exact test?
`...`	Arguments to test function

Details

Pairwise Chi-squared or Fisher test for 2-dimensional tables.

Alternatives: NCStats::chisqPostHoc() and fifer::chisq.post.hoc(). Both of them are not CRAN packages.

Value

List with test outputs.

Author(s)

Alexey Shipunov

Examples

```
titanic <- margin.table(Titanic, c(1, 4))
chisq.test(titanic)
pairwise.Table2.test(titanic)
```

`Phyllotaxis` *Plant phyllotaxis*

Description

Outputs the plant phyllotaxis formula or angle of divergence

Usage

```
Phyllotaxis(n, angle=FALSE)
Fibonacci(x)
```

Arguments

`n`	non-negative integer
`angle`	if TRUE, output angle of divergence
`x`	non-negative integer

Details

'Fibonacci(x)' calculates the n's Fibonacci's number, it is the rare case that is not exercise but really used for work.

'Phyllotaxis(n)' uses 'Fibonacci(x)' to output the phyllotaxis formula (see examples) or (if 'angle=TRUE') the angle of divergence.

Value

Number or character vector of length one.

Author(s)

Alexey Shipunov

Examples

```
sapply(1:10, Fibonacci)
sapply(1:10, Phyllotaxis)
sapply(1:10, Phyllotaxis, angle=TRUE)
```

`Pinhull` *Point in hull*

Description

For each observation, returns if it is within a polygon

Usage

```
Pinhull(pts, ppts)
```

Arguments

`pts`	Data points, 2-dimensional
`ppts`	List with polygon information (e.g., output from Hulls() or Ellipses())

Details

For each 'pts' observation, Pinhull() uses PBSmapping::findPolys() to find if it is within (or on the border) of each polygon described in 'ppts'.

The output or Pinhull is easy to use to calculate the "observation overlap", it also allows to reveal "outliers" (points outside all polygons) and all polygon membership features (e.g., which points belong to more than one polygon).

Value

Logical matrix, each column is the hull (polygon) name, rows correspond with rows of data points.

Author(s)

Alexey Shipunov

See Also

`Hulls`, `Ellipses`, `Overlap`

Examples

```
iris.p <- prcomp(iris[, -5])$x[, 1:2]
iris.h <- Hulls(iris.p, iris$Species, plot=FALSE)
iris.e <- Ellipses(iris.p, iris$Species, plot=FALSE)

## convex hulls
iris.pih <- Pinhull(iris.p, iris.h)
```

```
## confidence ellipses
iris.pie <- Pinhull(iris.p, iris.e)
## membership overlap
dist(t(iris.pie), method="binary")
## how to find outliers (points outside of all ellipses)
which(apply(iris.pie, 1, sum) == 0) # outliers
## how to make membership table
iris.pie.g <- cbind(iris.pie, group=Alldups(iris.pie, groups=TRUE))
key <- iris.pie.g[!duplicated(iris.pie), ]
key <- key[order(key[, "group"]), ]
mem <- aggregate(1:nrow(iris.p), list(group=iris.pie.g[, "group"]), paste0, collapse=", ")
mem <- cbind(key, mem)
mem[, mem %-% "group"] # all memberships

## distance based on membership intersection, Overlap() analog
dist(t(iris.pie), method="binary") # asymmetric binary
SM.dist(t(iris.pie)) # symmetric binary

## uniqueness of species
lapply(1:3, function(.x) sum(rowSums(iris.pie[as.numeric(iris$Species) == .x,
 ]) > 1)/table(iris$Species)[.x]) ## versicolor is least unique
```

---

plantago *plantago*

Description

Plantago (ribworts, plantains) species from European Russia: morphological table (Shipunov, 1998).

All not applicable, unknown and "both" values are labeled as "NA".

Usage

```
plantago
```

Format

This data frame has species names as row names, and contains the following columns (all variables are binary):

`V01` 0 annuals or biennials, 1 perennials

`V02` 0 not taller than 20 cm, 1 taller than 20 cm

`V03` 0 aboveground stems herbaceous, 1 aboveground stems woody

`V04` 0 vegetative nodes shortened, 1 vegetative nodes elongated

`V05` 0 vegetative shoots do not branch, 1 vegetative shoots branch

`V06` 0 phyllotaxis opposite, 1 phyllotaxis alternate

`V07` 0 well developed green leaves <= 5, 1 more

V08 0 the base of main shoot covered with remains of withered leaves, 1 the base is not covered with remains of withered leaves

V09 0 rhizome > 1 cm diam, 1 rhizome thinner

V10 0 slanted or horizontal rhizome, 1 vertical rhizome

V11 0 main root fast degrading, 1 main root presents on adult plants

V12 0 adventitious and lateral roots >= 1 mm diam, 1 less than 1 mm diam

V13 0 heterophylly present, 1 leaves similar

V14 0 leaves thin, transparent, 1 leaves not transparent

V15 0 leaves darken when dry, 1 leaves do not darken, sometimes became yellow or brown

V16 0 leaves (almost) naked, 1 leaves pubescent

V17 0 leaves flat, 1 leaves section rounded or leaves with furrow

V18 0 leaves with large teeth or even lobes, 1 leaves margin whole or with small teeth

V19 0 leaves linear, 1 leaves more broad

V20 0 leaves lanceolate, 1 leaves more broad

V21 0 leaves obovate, 1 leaves elliptic or ovate

V22 0 leaf tip blunt, 1 leaf tip sharp

V23 0 leaf base broad, suddenly narrowing into petiole, 1 leaf base narrow, smoothly become a petiole

V24 0 leaf margin with teeth, 1 leaf margin whole

V25 0 leaf veins >=7, 1 < 7

V26 0 petioles present, 1 petioles absent

V27 0 petioles almost equal or a bit shorter than leaf blades, 1 petioles much shorter than blades

V28 0 petioles without wings at the lowest 1/3 of length, 1 petioles with wings at the lowest 1/3 of length

V29 0 stalks horizontal or arcuate, 1 stalks straight or curved

V30 0 stalks naked, 1 stalks pubescent

V31 0 stalks with ribs, 1 stalks without ribs

V32 0 spikes longer, equal or slightly shorter than stalks, 1 spikes much shorter than stalks

V33 0 spikes long cylindrical or tail-like, 1 spikes rounded or short cylindrical

V34 0 lower bracts are much much broader than others, 1 all bracts similar

V35 0 middle and upper bracts not longer than sepals, 1 longer than sepals

V36 0 bracts with sharp tip, 1 bracts with blunt tip

V37 0 bract width >= length, 1 bract length > width

V38 0 bracts pubescent, 1 bracts naked

V39 0 bracts awned, 1 bracts not awned

V40 0 flowers slanted, spike lax, 1 flowers appressed, spike dense

V41 0 outer and inner sepals significantly different, 1 all sepals more or less similar

V42 0 sepals narrow, 1 sepals broad

V43 0 sepals with sharp tip, 1 sepals with blunt tip

V44 0 sepals pubescent, 1 sepals naked

V45 0 outer sepals fused, 1 outer sepals separate

V46 0 corolla tube pubescent, 1 corolla tube naked

V47 0 corolla lobes broad, elliptic or rounded, 1 corolla lobes narrow, oblanceolate or lanceolate

V48 0 corolla lobes with sharp tip, 1 corolla lobes with blunt tip

V49 0 corolla lobes white or silver, 1 corolla lobes yellowish or brownish

V50 0 stamens not exserted, 1 stamens exserted

V51 0 filaments yellowish or brownish, 1 filaments white, pinkish or purplish

V52 0 pollen grains with thickened pore margin, 1 pollen grains without thickened pore margin

V53 0 pollen grains with >= 9 pores, 1 pollen grains with < 9 pores

V54 0 pyxidium ovate or broadly conical, 1 pyxidium narrowly conical or elongated

V55 0 one or two seeds are much smaller than others, 1 all seeds similar

V56 0 seeds =< 2, 1 seeds > 2

V57 0 seeds 3–5, 1 seeds >= 6

V58 0 seeds flattened, 1 seeds rounded or angled

V59 0 plane which passes through embryo cotyledons is perpendicular to placenta, 1 plane which passes through embryo cotyledons is parallel to placenta

V60 0 polyploids, 1 diploids

V61 0 x=5, 1 x=6

Source

Shipunov A. 1998. Plantains (genera Plantago L. and Psyllium Mill., Plantaginaceae) of European Russia and adjacent territories. Ph. D. Thesis. Moscow State University.

Examples

```
plot(hclust(dist(plantago, method="binary")))
```

Pleiad *Correlation circles (correlation pleiads)*

Description

Plot correlation circles (correlation pleiads, correlograms)

Usage

```
Pleiad(tbl, abs=FALSE, corr=FALSE, dist=FALSE, treshold=FALSE,
 circ=list(1, 1, 1), breaks=5, auto=TRUE, gr=6, lwd=NULL, lty=NULL,
 lcol=NULL,  abbr=-1, lbltext="internal", lblcex=1, off=1.09, hofft=0.07,
 hoff=1.02, legend=TRUE,  legtext=1, legpos="topright", leghoriz=FALSE,
 show.int=FALSE, dig.lab=1, neg.col=NULL, ...)
```

Arguments

tbl	Data: square, numeric, symmetric matrix with same row and column names
abs	If TRUE, uses absolute values instead of real
corr	If TRUE, uses absolute values instead of real and cuts from 0 to 1, this is good for correlation matrices
dist	If TRUE, converts distance matrix to the data frame – good for "dist" objects
treshold	If this is (saying) =.5, selects for plotting (with lty=1) only those values which are >.5
circ	Line type, width and color for the cirle; if first or third =0, no cicrle
breaks	How to cut() values, if "cramer", then =c(0, .1, .3, .5, 1)
auto	If FALSE, specify 'lwd', 'lty' and 'lcol'
gr	Grayscale scheme starts from 6 breaks
lwd	If auto=FALSE, specify here the vector concerted with breaks
lty	If auto=FALSE, specify here the vector concerted with breaks
lcol	If auto=FALSE, specify here the vector concerted with breaks; if length(lcol) == 1, all lines are of particular color
abbr	If =-1, no abbreviation; if =0, no labels; other values run abbreviate(..., abbr)
lbltext	If this is a vector starting from something else, will replace dimnames
lblcex	Magnification of labels
off	Radial offset of labels, be careful!
hofft	Treshold determining which labels are rigtmost/leftmost, 'hofft=0' put all labels into this group
hoff	Horizontal offset for rightmost/leftmost labels; 'hoff=1' removes offset
legend	If FALSE, no legend
legtext	If =1 then "weaker ... stronger"; if =2, shows cutting intervals; if =3, then 1:5; if >3, issues error
legpos	Position of the legend, see help(legend)
leghoriz	Make the legend horizontal?
show.int	Show intervals in (...] form
dig.lab	dig.lab for cut(), use to change notation
neg.col	If not NULL and 'abs' or 'corr' are TRUE, colorize negative correlations using specified color
...	Further arguments to points()

Details

Correlation circles (correlation pleiads, correlograms) based on the works of Petr Terentjev's (Saint-Petersburg) school.

Please be sure to use 'corr=TRUE' or 'dist=TRUE' when working with correlation matrices or 'dist' objects, respectively.

It is probably a good idea to order data entries with hierarchical clustering to optimize the resulted graph (see examples).

Note that: (1) 'lty', 'lwd' and 'lcol' are not recycling; (2) plot has no margins so consider second 'legend()' to add any text to the plot (see examples); (3) it works best when number of items is relatively low (< 30); (4) it can visualize (colorize) negative correlations (see examples) but this works bes with default (black) color; (5) 'dist=TRUE' will convert dissimilarities into similarities by substracting from maximum.

Alternatives: those graph plotting packages which are able to plot "correlogram".

Value

Data frame (invisibly) with position of points, might help in plot enchancing.

Author(s)

Alexey Shipunov

Examples

```
l.c <- cor(datasets::longley, method="spearman", use="pairwise")
Pleiad(l.c, corr=TRUE, legtext=2, pch=21, cex=2, bg="white", breaks=3,
 gr=3, hoff=1, show.int=TRUE)
legend("topleft", legend="Macroeconomic correlations", text.font=2,
bty="n")

## colorize negative correlations and use hclust() to re-order
reorder <- hclust(dist(t(longley)))$order
Pleiad(l.c[reorder, reorder], corr=TRUE, neg.col="red")

dr.c <- cor(drosera[, -1], method="spearman", use="pairwise")
## simple example with most defaults
Pleiad(dr.c, corr=TRUE)
## complex example with user-specified colors etc.
Pleiad(dr.c, corr=TRUE, legtext=2, pch=19, cex=1.2, hoff=1.06,
 auto=FALSE, lwd=c(1, 1, 1, 2.5, 4), lty=rep(1, 5),
 lcol=colorRampPalette(c("grey", "#D8284F"))(5),
 circ=c(2, 1, 1))

## visualize distances
Pleiad(dist(t(atmospheres)), dist=TRUE, breaks=3, legtext=2, dig.lab=3)
```

Plot.phylocl *Plot phylogenetic tree with clades collapsed*

Description

Plot phylogenetic tree with clades collapsed into triangles or rectangles

Usage

```
Plot.phylocl(tree, cl, strict=TRUE, keep.mono=FALSE, what="triangles",
 col.ed="black", col.td="black", col.etr="transparent", col.ttr="transparent",
 col.pfl="lightgrey", col.pbr="black", lty.p=1, lwd.p=1, col.ct="black",
 ct.off=0, ct.fnt=1, cex=par("cex"), longer="0%", ...)
```

Arguments

tree	phylo object
cl	two columns classification table
strict	default TRUE: do not join all descendants
keep.mono	default FALSE: do not keep monotypic clades
what	default "triangles", also possible to use "rectangles"
col.ed	default "black", default edge color
col.td	default black", default tips color
col.etr	default "transparent", color to suppress original edges
col.ttr	default "transparent", color to suppress original tips
col.pfl	default "lightgrey", fill color for polygons
col.pbr	default "black", border color of polygons
lty.p	default 1, line type of polygon borders
lwd.p	default 1, line width
col.ct	default "black", color of clade labels
ct.off	default 0, text offset of clade labels
ct.fnt	default 1, text font of clade labels
cex	default par("cex"), text font size of all labels
longer	default "0%", percent to increase xlim to fit longer clade labels
...	options to ape::plot.phylo()

Details

Plot.phylocl() plots phylogenetic tree with clades collapsed into triangles or rectangles.

Alternative is phytools::plot.backbonePhylo() which however requires more manual work.

Some tricks used (null plotting and transparent elements), the last one is actually useful in other ways.

Intersections and other deviated cases not controlled. However, they are really easy to spot.

All parameters of polygons should be either "scalars" or vectors of the same length as clade list (minus monotypic clades), clades are in alphabetical order. To help, list of clade names is invisibly returned in the end.

If keep.mono=TRUE, then monotypic clades must have names in the clade list, otherwise this option is useless.

Value

Returns list of clade names.

Author(s)

Alexey Shipunov

See Also

phytools::plot.backbonePhylo()

Examples

```
aa.d <- hclust(dist(t(atmospheres)))
tree <- ape::unroot(ape::as.phylo(aa.d))

cl <- data.frame(
planet=c(
 "Mercury",
 "Venus",
 "Earth",
 "Mars",
 "Jupiter",
 "Saturn",
 "Uranus",
 "Neptune"),
clade=c(
 "Mercury",
 "Mars group",
 "Earth",
 "Mars group",
 "Close giants",
 "Close giants",
 "Distant giants",
 "Distant giants"),
stringsAsFactors=FALSE)

Plot.phylocl(tree, cl, longer="5%", ct.off=0.1)
```

PlotBest.dist *Dotchart which reflects the "best" base distance method*

Description

Plots dotchart with shows correspondences between data and various base distances

Usage

```
PlotBest.dist(data, distances=c("euclidean", "maximum", "manhattan",
"canberra", "binary", "minkowski"))
```

Arguments

data	Data frame with values
distances	Distances to use

Details

Shows the "best" distance method. Please note that this is a mere visualization, and numbers are used only to understand the relative correspondence between raw data and distances.

Uses maximal correlations between multidimensional scaling of distance object (converted internally to Euclidean) and PCA of data. Both MDS and PCA use two dimensions.

Author(s)

Alexey Shipunov

Examples

```
PlotBest.dist(iris[, -5])

PlotBest.dist(t(moldino))
```

PlotBest.hclust *Plots dotchart with best clustering method*

Description

Plots dotchart with best clustering method

Usage

```
PlotBest.hclust(dist, clust=c("ward.D", "ward.D2", "single", "complete",
"average", "mcquitty", "median", "centroid"), plot=TRUE)
```

Arguments

dist	Distance matrix
clust	Clustering method
plot	Plot?

Details

Shows the "best" hierarchical clustering method. Uses cophenetic correlation.

Value

Numeric vector with correlation values (equal to the number of clusterings involved)

Author(s)

Alexey Shipunov

Examples

```
PlotBest.hclust(dist(iris[, -5]))

PlotBest.hclust(dist(t(moldino)))
```

PlotBest.mdist *Dotchart which reflects the "best" of non-base distances*

Description

Plots dotchart which shows correspondences between data and various non-base distances

Usage

```
PlotBest.mdist(data, distances=c("manhattan", "euclidean", "canberra",
"clark",  "bray", "kulczynski", "jaccard", "gower", "altGower",
"morisita", "horn", "binomial",  "chao", "cao", "mahalanobis",
"cor.pearson", "cor.spearman", "cor.kendall",  "gower_dist",
"simple_match_dist", "daisy.gower", "smirnov"),
binary.only=FALSE)
```

Arguments

data	Data frame with values
distances	Distances to use
binary.only	Use binary only distances?

Details

Shows the "best" distance method using many non-base distances from several packages (namely, "cluster", "smirnov" and "vegan" – but does not include "mountford" and "raup" as they are very special). Please note that this is a mere visualization, and numbers are used only to understand the relative correspondence between raw data and distances.

Uses maximal correlations between multidimensional scaling of distance object (converted internally to Euclidean) and PCA of data. Both MDS and PCA use two dimensions.

Author(s)

Alexey Shipunov

See Also

PlotBest.dist

Examples

```
PlotBest.mdist(iris[, -5])

m1 <- t((moldino > 0) * 1)
PlotBest.mdist(m1, binary.only=TRUE)
```

Ploth *Changes the appearance of cluster dendrogram*

Description

Modifies several aspects of the cluster dendrogram

Usage

```
Ploth(hclust, labels=hclust[["labels"]], lab.font=1, lab.col=1, col=1,
 pch.cex=1, pch=NA, bg=0, col.edges=FALSE, hang=-1, ...)
```

Arguments

hclust	Hclust object
labels	Labels
lab.font	Label font
lab.col	Label colors
col	Colors of edges and points

pch.cex	Scale of points
pch	Point types
bg	Points backgrounds
col.edges	Colorize edges?
hang	Makes leaves hang, see plot.hclust(); -1 is default here whereas 0.1 is default for 'hclust'
...	Further arguments to plot.dendrogram()

Details

Changes the appearance of cluster dendrogram. If labels are long, you might need to modify the plot margins.

Please take into account that supplied labels are meant to be in their _initial_ order, not in order of their appearance on the dendrogram.

Ploth() does not change the text size of labels, please use Tctext() and Tcoords() for this (and other) purposes.

Author(s)

Alexey Shipunov

See Also

Tcoords

Examples

```
iris.dist <- dist(iris[, 1:4], method="manhattan")
iris.hclust <- hclust(iris.dist)
Ploth(iris.hclust, col=as.numeric(iris[, 5]), pch=16, col.edges=TRUE, horiz=TRUE,
 leaflab="none")
legend("topleft", fill=1:nlevels(iris[, 5]), legend=levels(iris[, 5]))

Ploth(hclust(UScitiesD, "ward.D2"), labels=abbreviate(attr(UScitiesD, "Labels")),
 lab.col=c(1, rep(2, 9)), lab.font=c(2, rep(1, 9)), hang=0.1)
```

Points *Number of cases in each location reflected in the point size*

Description

Number of cases in each location reflected in the point size

Usage

```
Points(x, y, pch=1, centers=FALSE, scale=1, cex.min=1, col=1,
 na.omit=TRUE, plot=TRUE, ...)
PPoints(groups, x, y, cols=as.numeric(groups), pchs=as.numeric(groups),
 na.omit.all=TRUE, ...)
```

Arguments

x, y	Coordinates
pch	Point type
pchs	Types of point groups
centers	If TRUE, show centers of each location as a pixel-size dot (pch=".")
cex.min	Minimal point size
col	Color of points
cols	Color of point groups
na.omit	If TRUE (default), skip data points with NAs
plot	If FALSE, does not plot
na.omit.all	If TRUE (default), skip data points and corresponding factor values with NAs, then make 'na.omit' for internal Points() FALSE
scale	Scale factor for point size
groups	Factor defining groups
...	Points() passes other arguments to points(), PPoints() passes other arguments to Points()

Details

Frequently, more then one data point is located in one coordinate place (so called "overplotting"). How to show overplotting? One way is 'jitter()', these is also (really advanced) 'sunflowerplot()'. 'Points()' does it in its own way: number of cases in each point will be reflected in the point size. 'Points()' is a low-level graphic function, analogous to 'points()'.

'PPoints()' is the same as 'Points()' but for multiple subgroups.

To prettify plot, it is recommended to change 'scale' and optionally also 'cex.min'.

Alternative is the base R 'sunflowerplot()' but it is hard to read and there is no possibility to show multiple groups in data. Another alternative might be points with transparent color.

Value

Invisibly returns vector of "multiplication indexes", in case of PPoints() it is group-wise so overplotting between groups does not count. Please keep in mind that these indexes only indicate how many times the point is overplotted, but do not show groups of duplicates. Use Alldups() for groups.

Author(s)

Alexey Shipunov

See Also

jitter, sunflowerplot

Examples

```
## colors modified via palette()
plot(iris[, 1:2], type="n")
palette(rainbow(3))
PPoints(iris[, 5], iris[, 1], iris[, 2], pchs=0, scale=0.7)
palette("default")
## now with centers, colors default, pch by group, and one NA
iris[1, 1] <- NA
plot(iris[, 1:2], type="n")
PPoints(iris[, 5], iris[, 1], iris[, 2], scale=0.7, centers=TRUE)
data(iris) ## to restore default embedded object
```

Polyarea *Area of the polygon*

Description

Calculates area of polygon

Usage

```
Polyarea(x)
```

Arguments

x	Polygon vertices: two-column numerical matrix or data frame

Details

Based on vegan::summary.ordihulls().

Value

Numerical vector of length 1.

Author(s)

Alexey Shipunov

See Also

Squares, Hulls, Ellipses

Examples

```
x <- c(1:9, 8:1) # from ?polygon
y <- c(1, 2*(5:3), 2, -1, 17, 9, 8, 2:9)
Polyarea(cbind(x, y)) # numerical matrix
Polyarea(data.frame(x, y)) # numerical data frame
```

Polycenter *Center of the polygon*

Description

Finds polygon center

Usage

```
Polycenter(x)
```

Arguments

x	Polygon vertices: two-column numerical matrix or data frame

Details

Based on vegan::summary.ordihulls().

Value

Named numerical vector of length 2 (x and y coordinates of the center).

Author(s)

Alexey Shipunov

See Also

Squares, Hulls, Ellipses

Examples

```
x <- c(1:9, 8:1) # from ?polygon
y <- c(1, 2*(5:3), 2, -1, 17, 9, 8, 2:9)
Polycenter(cbind(x, y)) # numerical matrix
Polycenter(data.frame(x, y)) # numerical data frame

iris.p <- prcomp(iris[, -5])$x[, 1:2]
iris.h <- Hulls(iris.p, iris$Species, plot=FALSE)
```

```
sapply(iris.h, Polycenter)
```

`Pull` *Select rows from data frame*

Description

Selects rows from data frame basing on the evaluation of the second argument

Usage

```
Pull(df, ...)
```

Arguments

`df`	Data frame to select from
`...`	Arguments to with(df, ...)

Details

If the first argument is not a data frame, function will stop with an error.

Pull() is similar to subset() (but is much simpler and allows non-logical values) and to dplyr::filter() function.

Please avoid using Pull() in non-ineractive mode.

Value

Data frame

Author(s)

Alexey Shipunov

Examples

```
`[`(trees, 3, 1) # ... so square bracket is a command
## arguments of `[` are independent; this is why square bracket does not "catch" the context:
trees[trees$Girth < 11 & trees$Height == 65, ] # boring and long
trees[trees$Girth < 11 & sample(0:1, nrow(trees), replace=TRUE), ] # yes, boring, long but flexible
trees[with(trees, Girth < 11 & Height == 65), ] # less boring but still long
## it would be nice to avoid typing "trees" twice:
Pull(trees, Girth < 11 & Height == 65) # shorter
Pull(trees, Girth < 11 & sample(0:1, nrow(trees), replace=TRUE)) # flexibility is still here
Pull(trees, Girth < 11 & sample(0:1, nrow(trees),
 replace=TRUE))$Height # if you want also select columns
Pull(trees, grep(81, Height)) # select not only by TRUE/FALSE but also by row index
```

R.logo *Imitation (!) of the modern 'R' logo*

Description

Imitation (!) of the modern 'R' logo

Usage

```
R.logo(x, y, col.e="#B8BABF", col.l="#1E63B5", cex=12)
```

Arguments

x	x coordinate of the letter
y	y coordinate of the letter
col.e	ellipse color
col.l	letter color
cex	scale, default 12

Details

Imitation (sic!) of the modern (flat) 'R' logo. Font and proportions are not exactly the same, also there is no gradient.

Author(s)

Alexey Shipunov

See Also

Ell

Examples

```
plot(1, type="n", axes=FALSE, xlab="", ylab="")
R.logo(1.1, 0.9, cex=25)
##
plot(1:20, type="n")
for (i in 1:20) R.logo(i, i, cex=2)
```

Read.fasta *Read 'FASTA' files*

Description

Simple reading of 'FASTA' files

Usage

```
Read.fasta(file)
```

Arguments

file	File name

Details

Simple reading of 'FASTA' files.

Value

Data frame with two columns: 'name' and 'sequence'.

Author(s)

Alexey Shipunov

Examples

```
write(file=file.path(tempdir(), "tmp.fasta"), ">some_id\nATGC")
Read.fasta(file=file.path(tempdir(), "tmp.fasta"))
```

Read.tri.nts *Read 'NTSYSpc' files*

Description

Read a lower triangular matrix

Usage

```
Read.tri.nts(file, ...)
```

Arguments

file	File to read
...	Arguments to 'scan()'

Details

Reads a lower triangular matrix which at least in my practice, typically come from 'NTSYSpc' program.

Author(s)

Alexey Shipunov

Examples

```
write(file=file.path(tempdir(), "tmp.nts"), x=c(
'" Procrustes distances between all pairs:
2 12 12 0
0.000E+000
4.058E-002 0.000E+000
5.753E-002 6.489E-002 0.000E+000
6.445E-002 8.124E-002 9.509E-002 0.000E+000
2.610E-001 2.395E-001 2.317E-001 3.051E-001 0.000E+000
2.719E-001 2.508E-001 2.461E-001 3.132E-001 4.531E-002 0.000E+000
2.563E-001 2.357E-001 2.278E-001 3.008E-001 4.414E-002 6.510E-002 0.000E+000
8.003E-002 6.611E-002 7.738E-002 9.885E-002 2.206E-001 2.270E-001 2.161E-001
0.000E+000
6.838E-002 8.893E-002 6.691E-002 1.018E-001 2.585E-001 2.704E-001 2.497E-001
1.019E-001 0.000E+000
6.233E-002 6.756E-002 4.079E-002 8.329E-002 2.396E-001 2.507E-001 2.338E-001
5.519E-002 5.932E-002 0.000E+000
2.504E-001 2.313E-001 2.230E-001 2.967E-001 8.714E-002 1.080E-001 6.522E-002
2.205E-001 2.323E-001 2.281E-001 0.000E+000
2.590E-001 2.688E-001 2.424E-001 2.757E-001 3.698E-001 3.926E-001 3.689E-001
3.051E-001 2.280E-001 2.603E-001 3.312E-001 0.000E+000 '
))

## interactive
file.show(file=file.path(tempdir(), "tmp.nts"))

Read.tri.nts(file=file.path(tempdir(), "tmp.nts"), skip=2)
```

Recode *Basic multiple recoding*

Description

Basic multiple recoding (similar to the 'SQL' left join)

Usage

```
Recode(var, from, to, char=TRUE, recycle=FALSE)
Recode4(var, from, to, missed="", ...)
RecodeR(var, from, to, char=TRUE, recycle=FALSE)
Recode4R(var, from, to, missed="", ...)
```

Arguments

var	Variable to recode
from	'from' column of the recoding "table"
to	'to' column
char	If TRUE (default), do not treat 'to' character vectors as factors
recycle	If TRUE (not default), recycle 'to' along 'from'
missed	Replace missed (not recoded) with something, default is "" (empty charactrer string)
...	Further options to Recode() and RecodeR()

Details

Basic multiple recoding is similar to 'SQL' left join.

Inspired from Paul Johnston (Univ. of Kansas) recode() function.

Alternatives are car::recode(), lessR::Recode(), admisc::recode() and 'mgsub' package. First three are much more complicated, last is much slower and less flexible.

To understand the idea better, please look on the examples.

There are four functions:

1. Recode() – base function. If starting points ("from") are the same, only the *last* rule ("from-to" pair) has an effect. If rules are chained, they still work independently (i.e., chaining has no effect).

2. Recode4() – considers not recoded (missing). By default, this will replace non-Recode()'d entries with empty string ("").

3. RecodeR() – running recode. If starting points ("from") are the same, only the *first* rule ("from-to" pair) has an effect. Chaining is possible.

4. Recode4R() – running plus considers missing. By default, this will replace non-RecodeR()'ed entries with empty string ("").

Value

Recoded vector (note that mode will not necessarily be the same, e.g., when recoding numbers with characters).

Author(s)

Alexey Shipunov

Examples

```
## recoding a phrase
phrase <- "The quick brown fox jumps over 123 lazy dogs"
var <- unlist(strsplit(phrase, split=""))
from <- letters[1:20]
to <- rev(from)
Recode.result <- paste(Recode(var, from, to), collapse="")
Recode4.result <- paste(Recode4(var, from, to, missed="-"), collapse="")
```

```
RecodeR.result <- paste(RecodeR(var, from, to), collapse="")
Recode4R.result <- paste(Recode4R(var, from, to, missed="-"), collapse="")
from.rule <- paste(from, collapse=" ")
to.rule <- paste(to, collapse=" ")
rbind(from.rule, to.rule, phrase, Recode.result, Recode4.result, RecodeR.result, Recode4R.result)

## reverse complement of DNA sequence
dna <- "GAATTC" # EcoR1 palindromic sequence
paste(Recode(rev(strsplit(dna, NULL)[[1]]),
 c("A", "T", "G", "C"), c("T", "A", "C", "G")), collapse="") # = 'dna', as expected
dna <- "ATTCGGC" # something random
paste(Recode(rev(strsplit(dna, NULL)[[1]]),
 c("A", "T", "G", "C"), c("T", "A", "C", "G")), collapse="")

## Recode4() when value recoded to itself
Recode4(1:5, 1:4, c(2, 1, 3, 3), NA)
Recode4(1:5, 1:4, c(2, 1, 3, 3))

## this is how "char" option works
Recode(1, 1, factor(2), char=FALSE)
Recode(1, 1, factor(2))

## this is how "recycle" option works
Recode(1:3, 1:3, 4)
Recode(1:3, 1:3, 4, recycle=TRUE)
```

`Root1` *Roots phylogenetic trees even if outgroup is not monophyletic*

Description

`Root1` non-interactively reroots a phylogenetic tree with respect to the specified outgroup even if it is not monophyletic.

Usage

```
Root1(phy, outgroup, select=1, ...)
```

Arguments

`phy`	An object of class `"phylo"`.
`outgroup`	A vector of mode numeric or character specifying the new outgroup.
`select`	Which element of outgroup to select if it is not monophyletic.
`...`	Arguments passed to ape::root().

Details

This is a wrapper of ape::root() to use in non-interactive mode. If specified outgroup is not monophyletic, instead of error, it issues error _message_, and chooses the 'select' element as a new outgroup.

Value

An object of class "phylo"

Author(s)

Alexey Shipunov

See Also

ape::root

Examples

```
data(bird.orders, package="ape")
ape::root(bird.orders, 1:2)
## ape::root(bird.orders, 1:3) # gives error
Root1(bird.orders, 1:3) # only outputs error _message_
Root1(bird.orders, 1, resolve.root=TRUE)
```

Rostova.tbl *Calculates multiple correlation matrices (via 'factor1') and stacks them together*

Description

Calculates multiple correlation matrices (via 'factor1') and stacks them together

Usage

```
Rostova.tbl(X, GROUP, ...)
```

Arguments

X	Data frame or matrix with values
GROUP	Number of grouping variable
...	Arguments to 'Cor.vec()'

Details

Calculates multiple correlation matrices (via GROUP) and stacks them together.

Output is suitable for PCA, distance calculations and other multivariate methods (Rostova, 1999; Rostova, 2002).

Value

Data frame with correlation structure

Author(s)

Alexey Shipunov

References

Rostova N.S. 1999. The variability of correlations systems between the morphological characters. Part 1. Natural populations of Leucanthemum vulgare (Asteraceae). Botanicheskij Zhurnal. 84(11): 50–66.

Rostova N.S. 2002. Correlations: Structure and Variability. Saint Petersburg, St. Petersburg University Publisher.

See Also

Cor.vec

Examples

```
Trees <- trees
Trees[, 4] <- sample(letters[1:3], nrow(Trees), replace=TRUE)
(rr <- Rostova.tbl(Trees, 4))
plot(hclust(dist(rr)))
```

Rpart2newick *Converts 'rpart' object into Newick tree*

Description

Converts 'rpart' object into Newick tree

Usage

```
Rpart2newick(rpart.object)
```

Arguments

rpart.object	'rpart' object, output of rpart::rpart()

Details

Inspired by 'shaunpwilkinson/rpart2dendro.R' gist.

Value

Newick tree (text string).

Examples

```
library(rpart)
(fit <- rpart(Kyphosis ~ Age + Number + Start, data=kyphosis))
plot(fit); text(fit, all=TRUE, xpd=TRUE)
library(ape)
tree1 <- read.tree(text=Rpart2newick(fit))
plot(tree1)
nodelabels(tree1$node.label, frame="none", bg="transparent", adj=-0.1)

(fit2 <- rpart(Species ~ ., data=iris))
plot(fit2); text(fit2, all=TRUE, xpd=TRUE)
tree2 <- read.tree(text=Rpart2newick(fit2))
plot(tree2)
nodelabels(tree2$node.label, frame="none", bg="transparent", adj=-0.1)
```

Rresults *Rresults shell script*

Description

Rresults shell script

Details

'Rresults' is a bash shell script which allows to gather all R input and R textual output in one text file, and (unnamed) R graphical output in another (PDF) file (only if the 'pdftk' utility is installed). If graphical output has name(s), it will be saved in its own file(s).

Very good for the debugging and other non-interactive activities with R scripts as everything is one place.

The script has one option "-d" which adds the timestamp to the file name of text results.

Author(s)

Alexey Shipunov

Examples

```
## Not run:
## works only if the script is properly installed
cat("\"Hello, world!\"\n", "plot(1:20)\n", file="hello.r")
system("Rresults hello.r")
system("Rresults -d hello.r")
## interactive command
file.show("hello_rresults.txt")

## End(Not run)
```

Rro.test *Robust rank order test*

Description

Robust rank order test

Usage

```
Rro.test(x1, y1)
```

Arguments

x1	Fist numerical variable
y1	Second numerical variable

Details

Robust rank order test (modification of Wilcoxon test for samples with contrasting variation), a variant of Fligner-Policello test.

Alternatives: robustrank::mod.wmw.test() (probably more sophisticated); npsm::fp.test(); NSM3::pFligPoli() (very advanced, with possibilities of exact and Monte Carlo testing); RVAideMemoire::fp.test() (developed in the way similar to most base R tests, probably the best alternative).

Value

Returns z statistic and p-value.

Author(s)

Alexey Shipunov

Examples

```
## data from help(wilcox.test)
x <- c(0.80, 0.83, 1.89, 1.04, 1.45, 1.38, 1.91, 1.64, 0.73, 1.46)
y <- c(1.15, 0.88, 0.90, 0.74, 1.21)
Rro.test(x, y)
```

`S.value` *S-value*

Description

`S.value` returns S-values, Shannon information transforms of p-values.

Usage

```
S.value(x)
```

Arguments

`x`	Either numerical vector of p-values, or list where at least one element has the name similar to "p.value".

Details

Greenland (2019) proposes that researchers "think of p-values as measuring the _compatibility_ between hypotheses and datas." S-values should help to understand this concept better.

From Wasserstein et al. (2019): S-values supplement a focal p-value p with its Shannon information transform (s-value or surprisal) s = -log2(p). This measures the amount of information supplied by the test against the tested hypothesis (or model): rounded off, the s-value shows the number of heads in a row one would need to see when tossing a coin to get the same amount of information against the tosses being “fair” (independent with “heads” probability of 1/2) instead of being loaded for heads. For example, if p = 0.03, this represents -log2(0.03) = 5 bits of information against the hypothesis (like getting 5 heads in a trial of “fairness” with 5 coin tosses); and if p = 0.25, this represents only -log2(0.25) = 2 bits of information against the hypothesis (like getting 2 heads in a trial of “fairness” with only 2 coin tosses).

For the convenience, S.value() works directly with output of many statistical tests (see examples). If the output is a list which has more than one component with name similar to "pvalue", only first will be used.

Value

Numerical vector.

Author(s)

Alexey Shipunov

References

Wasserstein R.L., Schirm A.L., Lazar N.A. 2019. Moving to a World Beyond “p < 0.05”. The American Statistician. 73(S1): 1–19.

Greenland S. 2019. Valid P-Values Behave Exactly as They Should: Some Misleading Criticisms of P-Values and Their Resolution With S-Values. The American Statistician. 73(S1): 106–114.

Examples

```
S.value(0.05)

S.value(0.01)
S.value(0.1)
S.value(0.00000000001)

S.value(t.test(extra ~ group, data = sleep))
S.value(list(pvalues=c(0.01, 0.002)))
```

`salix_leaves` *salix_leaves*

Description

Morphometry on willows (Salix).

Three files (datasets): 'salix_pop' localities, 'salix_plants' measures on whole plants, 'salix_leaves' measures on leaves from from these plants.

Usage

```
salix_leaves
```

Format

These data frames contain the following columns:

`POP` Location ID

`WHERE` Geography

`SPECIES` Species

`PLN` Plant ID

`HEIGHT` Height, m

`SEX` Plant sex (willows are dioeciuos)

`PID` Shoot ID

`N.CIRCLES` Number of circles of the imaginary spiral between two leaves (below)

`N.LEAVES` Number of leaves between the chosen one and the next in the same position

`INTERNODE` Internode length, average, mm

`DIAM` Stem diameter in the middle of shoot, mm

`NL` Leaf ID

`LL` Maximal length of the leaf, mm (along midvein from blade basement to blade top)

`LW` Maximal width of the leaf, mm

`PW` Position of maximal width, mm (along midvein)

`PTL` Length of the petiole, mm (from the place of attachment to blade base)

STPL Stipules present?

SL Maximal width of maximal stipule, mm (0 if no stipule present)

SW Maximal width of maximal stipule, mm (0 if no stipule present)

TL Length of maximal marginal tooth, mm (0 in no teeth)

ADC Color of the adaxial (upper) leaf surface: 1 glaucous, 2 other shades of green

ABC Color of the abaxial (lower) leaf surface: 1 glaucous, 2 other shades of green

ADP Pubescence of the adaxial (upper) leaf surface under magnification: 1 absent, 2 rare (epidermis surface visible), 3 dense (epidermis surface is not visible or barely visible)

ABP Pubescence of the abaxial (lower) leaf surface under magnification: 1 absent, 2 rare (epidermis surface visible), 3 dense (epidermis surface is not visible or barely visible)

`Saynodynamite` *Say "no" to dynamite plots!*

Description

Say "no" to dynamite plots!

Usage

```
Saynodynamite()
```

Details

'Poster' plot to emphasize the harmfulness of so called dynamite plots.

See, for example, thorough analysis at "http://emdbolker.wikidot.com/blog%3Adynamite".

Author(s)

Alexey Shipunov

See Also

`boxplot`

Examples

```
Saynodynamite()
```

SM.dist *Simple Match distance*

Description

Calculates simple match distance

Usage

```
SM.dist(data, zeroes=TRUE, cut=FALSE)
```

Arguments

data	Matrix (or data frame) with variables that should be used in the computation of the distance between rows.
zeroes	If FALSE (not default), zeroes will be ignored, so if data is binary, result will be close to the asymmetric binary distance ('dist(..., method="binary")').
cut	If TRUE (not default), attempt will be made to discretize all numeric columns with number of breaks default to hist(); zeroes will be saved.

Details

If argument is the data frame, SM.dist() internally converts it into the matrix. If there are character values, they will be converted column-wise to factors and then to integers.

SM.dist() ignores NAs when computing the distance values, and treates zeroes the same way if 'zeroes=FALSE'.

Value

Distance object with distances among rows of 'data'

Author(s)

Alexey Shipunov

See Also

dist

Examples

```
(mm <- rbind(c(1, 0, 0), c(1, NA, 1), c(1, 1, 0)))
SM.dist(mm)
SM.dist(mm, zeroes=FALSE)
dist(mm, method="binary")

ii <- cluster::pam(SM.dist(sapply(iris[, -5], round)), k=3)
```

```
Misclass(ii$clustering, iris$Species, best=TRUE)

i2 <- cluster::pam(SM.dist(iris), k=3) # SM.dist() "consumes" all types of data
Misclass(i2$clustering, iris$Species, best=TRUE)
```

`Squares` *Areas of polygons*

Description

Calculates areas of polygons

Usage

```
Squares(ppts, relative=FALSE)
```

Arguments

`ppts`	Output from Hulls() or Ellipses(), or just a list of polygons
`relative`	Calculate relative squares?

Details

"List of polygons" must be the list which contains any number of two-column numerical matrices of data frames, each represents the vertices of one polygon.

Might be useful to understand the variability of groups.

Value

Numerical (possibly named) vector of polygon areas, one element per polygon.

Author(s)

Alexey Shipunov

See Also

`Hulls`, `Ellipses`

Examples

```
iris.pca <- prcomp(iris[, -5])
iris.p <- iris.pca$x[, 1:2]

iris.h <- Hulls(iris.p[, 1:2], as.numeric(iris[, 5]), plot=FALSE)
Squares(iris.h, relative=TRUE)

iris.e <- Ellipses(iris.p, iris$Species, plot=FALSE, centers=TRUE)
Squares(iris.e)
```

Str *'str()' enchanced for data frames*

Description

Enhanced 'str()': with variable numbers, row names, missing data indication and possibly more

Usage

```
Str(df, as.factor=FALSE)
```

Arguments

df	Data frame
as.factor	Convert character columns to factors?

Details

'Str()' is an enhanced 'str()'. 'Str()' (1) shows data frame structure with column indexes, (2) indicates presence of NA(s) with star (*) and (3) lists first five row names, if they are not default.

If the object is a data frame with atomic columns, this function captures output of internal 'str()', changes it and outputs the new one. If the object is not a data frame or is a data frame with non-atomic columns, then output is not changed.

If 'as.factor=TRUE', converts all character columns to factors before reporting the structure, thus mimicking pre-R4 behavior of many functions related with data frames (and also invisibly outputs the new data frame). Might be useful, for example, to understand the number of unique character values which will be shown as "factor levels", works well in conjunction with summary(), please see examples.

Alternative: DescTools::Str() which uses cycles (slower!), has less features, but works with non-atomic columns.

Value

If 'as.factor=TRUE', invisibly outputs the data frame with all character columns converted into factors.

Author(s)

Alexey Shipunov

See Also

str

Examples

```
trees1 <- trees
row.names(trees1)[1] <- "a"
trees1[1, 1] <- NA
Str(trees)
Str(trees1)

## Not run:
trees.crazy <- trees
trees.crazy[[2]] <- trees[, 2, drop=FALSE]
str(trees.crazy)
Str(trees.crazy) # columns non-atomic: output as from str()

## End(Not run)

abc <- data.frame(N=1:26, LETTERS, letters, stringsAsFactors=FALSE)
abc[3, 1] <- NA
Str(abc)
Str(abc, as.factor=TRUE)
summary(Str(abc, as.factor=TRUE))
```

Table2df *Convert table to data frame saving structure*

Description

Convert table to data frame saving structure

Usage

```
Table2df(table)
```

Arguments

table	'table' object

Details

Convert contingency table into data frame and keep structure.

Value

Data frame

Author(s)

Alexey Shipunov

Examples

```
Table2df(table(iris[, 5]))
```

`Tcoords` *Calculates coordinates of tips from 'hclust' plot*

Description

Takes the 'hclust' plot and calculates coordinates of all tips

Usage

```
Tcoords(hcl, hang=0.1, add=0, horiz=FALSE)
```

Arguments

`hcl`	`hclust` object
`hang`	The fraction of the plot height by which labels should hang below the rest of the plot; better to make it equal to the 'hang' from `hclust` (default is 0.1).
`add`	Add to 'hang' to make labels look better; the reliable value is about 0.03
`horiz`	Plot values for a horizontal tree?

Details

This function calculates coordinates for each tip of the plotted 'hclust' object. It is useful together with plot.hclust(..., labels=FALSE).

There are numerous applications of Tcoords(). Typical situation is when user wants to change default labels on plot.hclust() but it is possible only after conversion into the dendrogram. However, this conversion might alter the graphical representation, and, what is worse, is not suitable for advanced forms of plot.hclust(), for example, in 'pvclust' package or those from Bclust() or Jclust()-related commands.

Tcoords() allows to plot labels (or points, or any low lewel structures) separately. Therefore, it is possible to rotate them, colorize them, abbreviate them, change their font and so on. By default, labels will be plotted horizontally, not vertically as it is in plot.hclust().

Tcoords() treats labels in order of their appearance on the dendrogram and not in their initial order, so do not forget to apply the 'order' component of 'hclust' object (see below for examples).

Together with Hcoords(), Tcoords() in principle allows to plot dendrogram in the alternative way (for example, with aid of segments() and text()). That will allow, for example, to plot 'hclust' horizontally without conversion into dendrogram. This possibility, however, reqiures a further research.

Please also see Fence() and Tctext() which are convenient functions based on Tcoords() for adding segments and text labels, respectively.

See Also

hclust, Ploth, Fence, Tctext

Examples

```
hcl <- hclust(UScitiesD, "ward.D2")
newlabels <- abbreviate(hcl$labels, 3)
newlabels <- newlabels[hcl$order] # do not forget to reorder labels!
plot(hcl, labels=FALSE)
text(Tcoords(hcl, add=0.03), newlabels, col=ifelse(grepl("W.D", newlabels), 2, 1))

## points instead of text labels
plot(hcl, labels=FALSE)
points(Tcoords(hcl), pch=19)

## how to colorize tips, useful if dendrogram is dense
iris.h <- hclust(dist(iris[, -5]))
plot(iris.h, labels=FALSE)
points(Tcoords(iris.h, add=0.01), col=iris$Species[iris.h$order]) # reorder lalels!

## can be used with Ploth(), i.e., with dendrogram
Ploth(hcl, labels=NA, horiz=TRUE) # hang=-1 is default
tc <- Tcoords(hcl, hang=-1, horiz=TRUE)
text(tc, newlabels, pos=4, xpd=TRUE, cex=1.1) # Ploth() cannot change text size
```

Tctext *Easy way to add text labels to 'hclust' plot*

Description

Uses text() and Tcoords() to add text labels to 'hclust' plot

Usage

```
Tctext(hcl, labels=hcl[["labels"]], hang=0.1, add=0, horiz=FALSE, xpd=TRUE, ...)
```

Arguments

hcl	hclust object
labels	Character vector with the size of 'labels' component of 'hcl'; by default, these exact 'labels'
hang	The fraction of the plot height by which labels should hang below the rest of the plot; by default, it is equal to the default 'hang' from hclust which is 0.1
add	Add to 'hang' to make labels look better; the reliable value is about 0.03
horiz	Plot on a horizontal tree?
xpd	Plot text if it goes outside of the plotting region?
...	Further arguments to text()

Details

Please note that labels (similarly to Ploth()) are treated in their _initial_, pre-clustered order because Tctext() reorders everything internally. This is not similar to Tcoords() which treats them in order of their appearance on the dendrogram and therefore requires manual re-ordering.

Please feel free to use the simple enough code of this function to produce other convenient 'hclust'-labeling routines, for example, one can make 'Tcpoints' based on Tcoords() and points().

See Also

Tcoords, hclust, Ploth

Examples

```
hcl <- hclust(UScitiesD, "ward.D2")

## how to imitate the default plot.hclust() with Tctext()
old.par <- par(mfrow=c(1, 2))
plot(hcl, labels=gsub("[A-z.]", "  ", hcl$labels))
Tctext(hcl, srt=90, add=0.04, adj=c(1, 0.5))
plot(hcl)
par(old.par)

## how to use different properties of text()
plot(hcl, labels=FALSE)
Tctext(hcl, srt=45, add=0.02, adj=c(0.8, 1), font=2:1, col=1:5, cex=0.8)

## how to use Tctext() with Ploth()
old.par <- par(mar=c(3, 1, 0, 7))
Ploth(hcl, horiz=TRUE, labels=NA, col=c(1:5, 1:5), col.edges=TRUE)
Tctext(hcl, horiz=TRUE, hang=-1, col=1:5, pos=4, cex=1.1, font=3)
par(old.par)
```

Tobin *Binarize (make dummy variables)*

Description

Converts vector into matrix with binary columns

Usage

```
Tobin(var, convert.names=TRUE)
```

Arguments

var	character or numerical variable
convert.names	if TRUE (default), construct new variable names, otherwise, use unique variable values as variable names

Details

'Tobin()' transforms character or numeric vector into the matrix with 0/1 (absent/present) cells.

Two approaches are in use: through '==' operation and through the conversion into factor.

First approach also constructs new names of variables whereas the second ('convert.names=FALSE') makes variable names from names of factor levels (i.e., labels).

Alternatives: "*dumm*" packages (there are few in CRAN).

Value

Matrix with binary columns

Author(s)

Alexey Shipunov

Examples

```
(ee <- sample(letters[1:5], 10, replace=TRUE))
Tobin(ee, conv=FALSE)
Tobin(ee, conv=TRUE)
```

`Toclip` *Insert content to Linux X11 clipboard*

Description

Insert content to Linux X11 clipboard (uses 'xclip')

Usage

```
Toclip(x, sep="\t", row.names=FALSE, col.names=TRUE, ...)
```

Arguments

`x`	Data frame
`sep`	Separator, tab by default
`row.names`	FALSE by default
`col.names`	TRUE by default
`...`	Arguments to 'write.table()'

Details

Linux-specific. Inserts data frame to Linux X11 clipboard (not primary or secondary). Useful for interface with spreadsheets.

Works if 'xclip' utility is already installed.

Alternative with more flexibility: 'clipr' package.

Author(s)

Alexey Shipunov

Examples

```
## Not run:
aa <- data.frame(1:3) # Linux- (and X11-) specific
Toclip(aa) # then load the content into spreadsheet

## End(Not run)
```

Topm *Stacks correlation matrix*

Description

Stacks (correlation) matrix and selects values which are above the “level”

Usage

```
Topm(X, level=0.45, values=0, corr=TRUE, square=TRUE)
```

Arguments

X	Data frame or matrix with values
level	Threshold
values	If > 0, ignores "level" and outputs until reaches number, if "all", outputs all values
corr	If FALSE, does not show magnitude
square	If FALSE, does not use lower triangle, some rows could be redundant

Details

’Topm()’ stacks (correlation) matrix and selects (and sorts) values which are above the “level”.

Good for the analysis of correlation matrices.

Value

Data frame with correlation values

Author(s)

Alexey Shipunov

See Also

Cor

Examples

```
Topm(cor(trees), corr=TRUE)
```

Updist *Educated distances for semi-supervised clustering*

Description

Updates distance matrix to help link or unlink objects

Usage

```
Updist(dst, link=NULL, unlink=NULL, dmax=max(dst), dmin=min(dst))
```

Arguments

dst	dist object
link	1-level list with the arbitrary number of components, each component is a numeric vector of row numbers for objects which you prefer to be linked
unlink	1-level list with the arbitrary number of components, each component is a numeric vector of row numbers for objects which you prefer to be not linked
dmax	Distance to set for not linked objects
dmin	Distance to set for linked objects

Details

This function borrows the idea of MPCKM semi-supervised k-means (Bilenko et al., 2004) but instead of updating distances on the run, it simply updates the distances object beforehand in accordance with 'link' and 'unlink' constraints.

Amazingly, it works as expected :) Please see the examples below.

References

Bilenko M., Basu S., Mooney R.J. 2004. Integrating constraints and metric learning in semi-supervised clustering. In: Proceedings of the twenty-first international conference on Machine learning. P. 11. ACM.

See Also

dist

Examples

```
iris.d <- dist(iris[, -5])
iris.km <- kmeans(iris.d, 3)
iris.h <- cutree(hclust(iris.d, method="ward.D"), k=3)

Misclass(iris.km$cluster, iris$Species, best=TRUE)
Misclass(iris.h, iris$Species, best=TRUE)
```

```
i.vv <- cbind(which(iris$Species == "versicolor"), which(iris$Species == "virginica"))
i.link <- list(sample(i.vv[, 2], 25), sample(i.vv[, 1], 25))
i.unlink <- list(i.vv[1, ], i.vv[2, ])

iris.upd <- Updist(iris.d, link=i.link, unlink=i.unlink)

iris.ukm <- kmeans(iris.upd, 3)
iris.uh <- cutree(hclust(iris.upd, method="ward.D"), k=3)

Misclass(iris.ukm$cluster, iris$Species, best=TRUE)
Misclass(iris.uh, iris$Species, best=TRUE)

## ===

aad <- dist(t(atmospheres))
plot(hclust(aad))

aadu <- Updist(aad, unlink=list(c("Earth", "Mercury")))
plot(hclust(aadu))
```

`Vicinities` *Arrange observations by the distance from center*

Description

Uses group centers to order all observations within group

Usage

```
Vicinities(data, groups, num=NULL, centers=NULL, method.c="median", method.d="manhattan")
```

Arguments

`data`	Numeric data frame or matrix
`groups`	Grouping factor
`num`	How many indices per group to return, default is all
`centers`	Matrix or data frame with group centers, one row per 'groups' level
`method.c`	How to calculate group centers, name of function
`method.d`	How to calculate distances between centers and individual observations, dist() mehtod

Details

This function takes data (data frame or matrix), grouping factor and (optionally) matrix or data frame with group centers. Then it calculates proximities between all observations and corresponding center, group by group. Result is a list of proximity indices (row numbers). This list allows, for example, to find "central" ("typical", "nuclear") observations useful, e.g., as centroids or medoids, and also "peripheral" observations, "outliers".

Distances by default are calculated with dist(..., method="manhattan") but it is possible to specify any other dist() method via "method.d".

Pre-defined centers might be taken from many sources, e.g., Hulls(), Ellipses(), Classproj(), see examples.

If "centers" data is not supplied, then Vicinities() will perform a naive computation of group centers via univariate medians. It is also possible to use (via "method.c") mean or any similar function which works within sapply() and accepts "na.rm=TRUE" option.

If size of the group is less then "num", the resulted list will contain NAs. If this is not a desired behavior, use something like lapply(res, head, num).

Value

The list of nlevels(as.factor(groups)) size, components named from these levels and contained "num" numeric indices, corresponding with the row numbers of original data.

Author(s)

Alexey Shipunov

See Also

Hulls, Ellipses, Classproj

Examples

```
## use for MDS
iris.d <- dist(iris[, -5])
iris.c <- cmdscale(iris.d)
iris.sc <- as.data.frame(iris.c)
## naive calculation
first3n <- unlist(Vicinities(iris.sc, iris$Species, num=3))
last10n <- unlist(lapply(Vicinities(iris.sc, iris$Species), tail, 10))
##
plot(iris.sc, col=iris$Species)
points(iris.sc[first3n, ], pch=19, col=iris$Species[first3n])
points(iris.sc[last10n, ], pch=4, cex=2, col="black")

## use pre-defined centers from Hulls()
plot(iris.sc, col=iris$Species)
iris.h <- Hulls(iris.sc, groups=iris$Species, centers=TRUE)
first3h <- unlist(Vicinities(iris.sc, iris$Species, centers=iris.h$centers, num=3))
points(iris.sc[first3h, ], pch=19, col=iris$Species[first3h])
```

```
## use pre-defined centers from Ellipses()
plot(iris.sc, col=iris$Species)
iris.e <- Ellipses(iris.sc, groups=iris$Species, centers=TRUE)
first3e <- unlist(Vicinities(iris.sc, iris$Species, centers=iris.e$centers, num=3))
points(iris.sc[first3e, ], pch=19, col=iris$Species[first3e])

## ===

## plot and use pre-defined centers from Classproj()
iris.cl <- Classproj(iris[, -5], iris[, 5])
first3cc <- unlist(Vicinities(iris.cl$proj, iris[, 5], centers=iris.cl$centers, num=3))
plot(iris.cl$proj, col=iris$Species)
points(iris.cl$proj[first3cc, ], pch=19, col=iris$Species[first3cc])
## now calculate centers naively from projection data
first3cn <- unlist(Vicinities(iris.cl$proj, iris[, 5], num=3))
points(iris.cl$proj[first3cn, ], pch=1, cex=1.5, col=iris$Species[first3cn])

## ===

## use as medoids for PAM
library(cluster)
iris.p <- pam(iris.d, k=3)
Misclass(iris.p$clustering, iris[, 5], best=TRUE) # to compare
## naive method, raw data (4 columns)
first1nr <- unlist(Vicinities(iris[, -5], iris$Species, 1))
iris.pm <- pam(iris.d, k=3, medoids=first1nr)
Misclass(iris.pm$clustering, iris[, 5], best=TRUE) # slightly better!
## ... or as centers for k-means, for stability
first1h <- unlist(Vicinities(iris.sc, iris$Species, centers=iris.h$centers, num=1))
iris.km <- kmeans(iris[, -5], centers=iris[first1h, -5])
Misclass(iris.km$cluster, iris[, 5], best=TRUE)

## ===

## PCA and different vicinities methods
iris.p <- prcomp(iris[, -5])$x[, 1:2]
plot(iris.p, col=iris$Species)
first3p1 <- unlist(Vicinities(iris.p, iris[, 5], num=3))
first3p2 <- unlist(Vicinities(iris.p, iris[, 5], num=3,
 method.c="mean", method.d="euclidean")) # mean, Euclidean
points(iris.p[first3p1, ], pch=19, col=iris[first3p1, 5])
points(iris.p[first3p2, ], pch=1, cex=1.5, col=iris[first3p2, 5])

## ===

## use MDS vicinities to reduce dataset for hierarchical clustering with bootstrap
iris3 <- iris[first3n, ]
iris3b <- Bclust(iris3[, -5], method.d="euclidean", method.c="average", iter=100)
plot(iris3b$hclust, labels=iris3[, 5])
Bclabels(iris3b$hclust, iris3b$values)

iris3j <- Jclust(iris3[, -5], method.d="euclidean", method.c="average",
```

```
 n.cl=3, iter=100)
plot(iris3j, labels=iris3[, 5])

## ===

## use of external function to compute naive distances
Mode <- function(x, na.rm=TRUE) {
 if (length(x) <= 2) return(x[1])
 if (na.rm & anyNA(x)) x <- x[!is.na(x)]
 ux <- unique(x)
 ux[which.max(tabulate(match(x, ux)))]
}
Vicinities(iris[, -5], iris[, 5], method.c="Mode", 3)
```

VTcoeffs *Effect sizes of association between categorical variables*

Description

Effect sizes of association between categorical variables

Usage

```
VTcoeffs(table, correct=FALSE, ...)
```

Arguments

table	Contingency table
correct	Perform continuity correction in underlying chi-square test?
...	Additional arguments to underlying chisq.test()

Details

Association between categorical variables.

Calculates Cramer's V and Tschuprow's, original and corrected (Bergsma, 2013)

Alternative: vcd::assocstats()

Includes magnitude interpretation for original Cramer's V (for df < 6).

Value

Data frame with coefficients, values and tables.

Author(s)

Alexey Shipunov

Examples

```
x <- margin.table(Titanic, 1:2)
VTcoeffs(x)
VTcoeffs(x)[2, ] # most practical
```

`Write.fasta` *Write 'FASTA' files*

Description

Simple writing of 'FASTA' files

Usage

```
Write.fasta(df, file)
```

Arguments

`df`	Name of data frame
`file`	File name

Details

Simple writing of 'FASTA' files. If the data frame has more then two columns, only two first columns will be used (with warning).

Value

'FASTA' file on the disk.

Author(s)

Alexey Shipunov

Examples

```
ff <- data.frame(one="some_id", two="ATGC", three="something else")
Write.fasta(ff, file=file.path(tempdir(), "tmp.fasta")) # warning will be produced
file.show(file=file.path(tempdir(), "tmp.fasta")) # interactive
```

Xpager *Separate terminal pager for Linux*

Description

Separate terminal pager for Linux X11 (uses some terminal and 'less')

Usage

```
Xpager(pager="xterm")
```

Arguments

pager	name of the terminal application to use, or "old" for the default

Details

Linux pager in the new terminal window. 'xterm' is default, there is also setting for 'mate-terminal'; 'konsole' (KDE terminal) and 'gnome-terminal' are easy to add.

Run Xpager("old") to restore default behavior.

BTW, for some reason, 'editor()' does not work this way.

Author(s)

Alexey Shipunov

Examples

```
## Not run:
## Linux- (and X11-) specific
Xpager()
?help
Xpager("old")
?help

## End(Not run)
```

`%-%` *Minus names*

Description

Subtract names from names

Usage

```
x %-% y
```

Arguments

x	Character vector (likely named) to subtract from
y	Subtracting character vector

Details

Instead of 'x', the function uses 'names(x)'. If 'x' has no names, they will be assigned from values.

Value

Character vector

Author(s)

Alexey Shipunov

Examples

```
str(iris[, iris %-% "Species"])
str(iris[, !names(iris) %in% "Species"]) # this is how to make it without %-%
c("apples", "bananas") %-% "apples" # simple character string also works
```

Index