Package ‘shipunov’

February 26, 2022

Type Package

Title Miscellaneous Functions from Alexey Shipunov

Version 1.17

Date 2022-02-26

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>


- 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, methods

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 2022-02-26 13:30:02 UTC
### R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adj.Rand</td>
<td>4</td>
</tr>
<tr>
<td>Aggregate1</td>
<td>5</td>
</tr>
<tr>
<td>Alldups</td>
<td>6</td>
</tr>
<tr>
<td>atmospheres</td>
<td>7</td>
</tr>
<tr>
<td>Bclabels</td>
<td>8</td>
</tr>
<tr>
<td>Bclust</td>
<td>9</td>
</tr>
<tr>
<td>BestOverlap</td>
<td>12</td>
</tr>
<tr>
<td>Biarrows</td>
<td>14</td>
</tr>
<tr>
<td>Biokey</td>
<td>17</td>
</tr>
<tr>
<td>BootA</td>
<td>20</td>
</tr>
<tr>
<td>BootKNN</td>
<td>22</td>
</tr>
<tr>
<td>BootRF</td>
<td>23</td>
</tr>
<tr>
<td>Boxplots</td>
<td>24</td>
</tr>
<tr>
<td>Cdate</td>
<td>25</td>
</tr>
<tr>
<td>chaetocnema</td>
<td>26</td>
</tr>
<tr>
<td>Cladd</td>
<td>27</td>
</tr>
<tr>
<td>Class.sample</td>
<td>28</td>
</tr>
<tr>
<td>classifs</td>
<td>29</td>
</tr>
<tr>
<td>Classproj</td>
<td>31</td>
</tr>
<tr>
<td>Clustergram</td>
<td>32</td>
</tr>
<tr>
<td>Coeff.det</td>
<td>34</td>
</tr>
<tr>
<td>Coml</td>
<td>35</td>
</tr>
<tr>
<td>Cor</td>
<td>36</td>
</tr>
<tr>
<td>Cor.vec</td>
<td>37</td>
</tr>
<tr>
<td>CVs</td>
<td>38</td>
</tr>
<tr>
<td>Ditto</td>
<td>39</td>
</tr>
<tr>
<td>DNN</td>
<td>40</td>
</tr>
<tr>
<td>dolbli</td>
<td>42</td>
</tr>
<tr>
<td>Dotcharts</td>
<td>43</td>
</tr>
<tr>
<td>drosera</td>
<td>45</td>
</tr>
<tr>
<td>Ell</td>
<td>46</td>
</tr>
<tr>
<td>Ellipses</td>
<td>47</td>
</tr>
<tr>
<td>eq</td>
<td>49</td>
</tr>
<tr>
<td>Ex.boxplot</td>
<td>50</td>
</tr>
<tr>
<td>Ex.col</td>
<td>50</td>
</tr>
<tr>
<td>Ex.font</td>
<td>51</td>
</tr>
<tr>
<td>Ex.lty</td>
<td>52</td>
</tr>
<tr>
<td>Ex.margins</td>
<td>52</td>
</tr>
<tr>
<td>Ex.pch</td>
<td>53</td>
</tr>
<tr>
<td>Ex.plots</td>
<td>54</td>
</tr>
<tr>
<td>Fence</td>
<td>55</td>
</tr>
<tr>
<td>Files</td>
<td>56</td>
</tr>
<tr>
<td>Fill</td>
<td>57</td>
</tr>
<tr>
<td>Gap.code</td>
<td>58</td>
</tr>
<tr>
<td>Gen.cl.data</td>
<td>59</td>
</tr>
<tr>
<td>Gini</td>
<td>62</td>
</tr>
<tr>
<td>R topics documented:</td>
<td></td>
</tr>
<tr>
<td>-------------------------------------------------------------------------------------</td>
<td>---</td>
</tr>
<tr>
<td>Gower.dist</td>
<td>63</td>
</tr>
<tr>
<td>Gradd</td>
<td>65</td>
</tr>
<tr>
<td>Gridmoon</td>
<td>68</td>
</tr>
<tr>
<td>haltica</td>
<td>69</td>
</tr>
<tr>
<td>Hc12mat</td>
<td>70</td>
</tr>
<tr>
<td>Hclust.match</td>
<td>71</td>
</tr>
<tr>
<td>Hcoords</td>
<td>72</td>
</tr>
<tr>
<td>Histr</td>
<td>73</td>
</tr>
<tr>
<td>hrahn</td>
<td>74</td>
</tr>
<tr>
<td>Hulls</td>
<td>77</td>
</tr>
<tr>
<td>hwc</td>
<td>79</td>
</tr>
<tr>
<td>Infill</td>
<td>80</td>
</tr>
<tr>
<td>Jclust</td>
<td>81</td>
</tr>
<tr>
<td>K</td>
<td>83</td>
</tr>
<tr>
<td>keys</td>
<td>84</td>
</tr>
<tr>
<td>Life</td>
<td>88</td>
</tr>
<tr>
<td>Linechart</td>
<td>89</td>
</tr>
<tr>
<td>Ls</td>
<td>90</td>
</tr>
<tr>
<td>Mag</td>
<td>91</td>
</tr>
<tr>
<td>MDSv</td>
<td>92</td>
</tr>
<tr>
<td>Miney</td>
<td>93</td>
</tr>
<tr>
<td>Misclass</td>
<td>94</td>
</tr>
<tr>
<td>Missing.map</td>
<td>96</td>
</tr>
<tr>
<td>moldino</td>
<td>96</td>
</tr>
<tr>
<td>MrBayes</td>
<td>97</td>
</tr>
<tr>
<td>MRH</td>
<td>99</td>
</tr>
<tr>
<td>NC.dist</td>
<td>101</td>
</tr>
<tr>
<td>Normality</td>
<td>102</td>
</tr>
<tr>
<td>Overlap</td>
<td>103</td>
</tr>
<tr>
<td>pairwise.Eff</td>
<td>105</td>
</tr>
<tr>
<td>pairwise.Rro.test</td>
<td>106</td>
</tr>
<tr>
<td>pairwise.Table2.test</td>
<td>107</td>
</tr>
<tr>
<td>Phyllotaxis</td>
<td>108</td>
</tr>
<tr>
<td>Pinhull</td>
<td>109</td>
</tr>
<tr>
<td>plantago</td>
<td>110</td>
</tr>
<tr>
<td>Pleiad</td>
<td>112</td>
</tr>
<tr>
<td>Plot.phylocl</td>
<td>114</td>
</tr>
<tr>
<td>PlotBest.dist</td>
<td>116</td>
</tr>
<tr>
<td>PlotBest.hclus</td>
<td>117</td>
</tr>
<tr>
<td>PlotBest.mdist</td>
<td>118</td>
</tr>
<tr>
<td>Plot.h</td>
<td>119</td>
</tr>
<tr>
<td>Points</td>
<td>120</td>
</tr>
<tr>
<td>Polyarea</td>
<td>122</td>
</tr>
<tr>
<td>Polycenter</td>
<td>123</td>
</tr>
<tr>
<td>Pull</td>
<td>124</td>
</tr>
<tr>
<td>R.logo</td>
<td>125</td>
</tr>
<tr>
<td>Read.fasta</td>
<td>126</td>
</tr>
<tr>
<td>Read.tri.nts</td>
<td>126</td>
</tr>
</tbody>
</table>
Adjusted Rand index to compare different clusterings

Usage

\[ \text{Adj.Rand}(c1, c2, \ldots) \]

Arguments

- \text{c1} \quad \text{First classification (character vector of group names)}
- \text{c2} \quad \text{Second classification}
- \ldots \quad \text{Further arguments to \text{table()}}
**Aggregate1**

**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**


**See Also**

`Misclass`

**Examples**

```r
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()`
Alldups

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'. Alldups() 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

```r
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)
```

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

```r
## '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$hclust, labels=FALSE, main="", xlab="", sub="Bootstrap, 100 replicates")
Bclabels(iris.bb$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
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


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
BestOverlap

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

```r
## 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
```
Biarrow library(tapkee)
B <- c("lle", "npe", "lltsa", "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 reduction 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
**Biarrrows**

- `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`: Offset 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**

Biarrrows() 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 Biarrrows() 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. Biarrrows() is also much more flexible than standard biplot(). Please note, however, that Biarrrows() is only visualization, and numerical conclusions might not be justified.

If 'deriv' data contains more than 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.

Biarrrows(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

```r
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")

## ===
```
## Code

```r
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

```r
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

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 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
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="")

### 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")
(indented0 <- read.table(textConnection(indented0), sep=" ", as.is=TRUE))
Biokey(indented0, from="indented", to="bracket")
Biokey(indented0, from="indented", to="serial")
(nn <- Biokey(indented0, 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")
(table0 <- read.table(textConnection(table0), sep=" ", as.is=TRUE, h=TRUE))
names(table0) <- Numranks(ranks=names(table0))
table0
Biokey(table0, 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 *subset* 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**

```r
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**

```r
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 subset 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
Boxplots

See Also

randomForest::randomForest

Examples

```r
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

Description

Boxplots for every scaled variable grouped by factor

Usage

```r
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

```r
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
```

Description

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

Usage

```r
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**


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. heikertingeri 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 Khabarovsky; 31 Germany.

Ch. heptapotamnica Lubis.:


**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
Cladd

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

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lbls</td>
<td>Vector of labels convertible into factor</td>
</tr>
<tr>
<td>nsam</td>
<td>Number of samples to take from each class</td>
</tr>
<tr>
<td>prop</td>
<td>Proportion of samples to take from each class</td>
</tr>
<tr>
<td>uniform</td>
<td>Uniform instead of random?</td>
</tr>
</tbody>
</table>

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 than 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

```r
(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)
```
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":

![Page 1. MONANDRIA MONOGYNIA. CANNA.](image)

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

See Also
Biokey, Numranks

Examples

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

References


Examples

```r
## 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)
```

---

Clustergram

Clustergram: visualize the cluster structure

Description

Plot which shows cluster memberships and distances when clusters numbers increases
Clustergram

Usage

Clustergram(data, maxcl=ncol(data)*2, nboot=FALSE, method="kmeans", m.dist="euclidean", m.hclust="complete", plot=TRUE, broom=4e-3, col="gray", ...)

Arguments

data: Data, typically data frame
maxcl: Maximal number of clusters, default is number of columns times 2; minimal number of clusters is 2
nboot: Either ‘FALSE’ (no bootstrap, default) or number of bootstrap runs
method: Either ‘kmeans’ or ‘hclust’
m.dist: If method='hclust', method to calculate distances, see ?dist
m.hclust: If method='hclust', method to clusterize, see ?hclust
plot: Plot?
broom: Extent to which spread lines, default is 4e-3
col: Color of lines
...: Further arguments to plot()

Details

Clustergram shows how cluster members are assigned to clusters as the number of clusters increases. This graph is useful in exploratory analysis for non-hierarchical clustering algorithms like k-means and for hierarchical cluster algorithms when the number of observations is large enough to make dendrograms impractical (from Schonlau, 2004; see also www.schonlau.net).

One application is to use clustergram to determine the optimal number of clusters. Basic idea is that you look for the point (number of clusters) where more clusters do not significantly change the picture (i.e., do not add more information) The best number of clusters is _near_ that point (see examples).

See also Martin Fleischmann (martinfleischmann.net) for practical explanation and scikit-learn 'clustergram' Python package.

Clustergram() code based on simplified and optimized Tal Galili’s github 'clustergram’ code.

Author(s)

Alexey Shipunov

References

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

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


Examples

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

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


Examples

```r
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

```r
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 triangle. Uses Pearson correlation only but much faster than 'Cor()'.
Cor.vec

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
See Also

Rostova.tbl

Examples

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

<table>
<thead>
<tr>
<th>CVs</th>
<th>Coefficients of variation</th>
</tr>
</thead>
</table>

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()
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

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

c1 <- iris$Species
sam <- c(rep(0, 4), 1) > 0
c1[!sam] <- NA
table(c1, useNA="ifany")

## based on neighbor number
iris.pred <- DNN(dst=iris.d, cl=c1, k=5)
Misclass(iris$Species[is.na(c1)], iris.pred)

## based on neighborhood size
iris.pred <- DNN(dst=iris.d, cl=c1, d=0.05)
table(iris.pred, useNA="ifany")
Misclass(iris$Species[is.na(c1)], iris.pred)

## protection against "all points relevant"
DNN(dst=iris.d, cl=c1, d=1)[1:5]
## and all are ties:
DNN(dst=iris.d, cl=c1, d=1, details=TRUE)[, 1:5]

## any distance works
iris.d2 <- Gower.dist(iris[, -5])
iris.pred <- DNN(dst=iris.d2, cl=c1, k=5)
Misclass(iris$Species[is.na(c1)], iris.pred)

## combined
c2 <- c1
iris.pred <- DNN(dst=iris.d, cl=c2, d=0.05)
c2[is.na(c2)] <- iris.pred
table(c2, useNA="ifany")
iris.pred2 <- DNN(dst=iris.d, cl=c2, k=5)
c2[is.na(c2)] <- iris.pred2
table(c2, useNA="ifany")
```
## self-training and class proximity surrogate
c23 <- iris$Species
t(DNN(dst=iris.d, cl=c23, k=5, details=TRUE, self=TRUE))/5

## Dnn() with more class::knn()-like interface
c2r <- iris[sam, ]
c2t <- iris[!sam, ]
Dnn(iris.trn[, -5], iris.tst[, -5], iris.trn[, 5], k=7)

## stepwise DNN, note the warning when no NAs left
c24 <- c21
for (d in (5:14)/100) {
  iris.pred <- DNN(dst=iris.d, cl=c24, d=d)
c24[is.na(c24)] <- iris.pred
}
table(c24, useNA="ifany")
Misclass(iris$Species, c24)

## rushing to d=14% gives much worse results
iris.pred <- DNN(dst=iris.d, cl=c21, d=0.14)
table(iris.pred, useNA="ifany")
Misclass(iris$Species[is.na(c21)], iris.pred)

---

### 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
doibli

### 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
Dotcharts

Source


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

```r
## 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


### Usage

```r
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
**Description**

Plot ellipse

**Usage**

\[
\text{Ell}(x, y, \text{width}, \text{height}=\text{width}, \text{theta}=2\pi, \text{npoints}=100, \text{plot}=\text{TRUE}, \ldots)
\]

**Arguments**

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

**Details**

Plots ellipse based on 'polygon()'.

**Value**

If \text{plot}=\text{FALSE}, returns list of components.

**Author(s)**

Alexey Shipunov

**Examples**

\[
\text{plot}(1:8, \text{type}="n")
\]
\[
\text{Ell}(4, 5, 6)
\]
**Ellipses**  

**Confidence ellipses**

---

**Description**

Calculates and plots group confidence ellipses

**Usage**

```r
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()`.
Ellipses

Author(s)
Alexey Shipunov

See Also
Hulls, Overlap, Pinhull

Examples

```r
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 ellipses
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, xlab=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")
}
```
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 belonging to the 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)
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()

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

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

**Ex.pch**

**Author(s)**
Alexey Shipunov

**References**

**See Also**
par

**Examples**
Ex.margins()

---

**Description**
Point ('pch') examples

**Usage**

```r
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")
**Description**

Textual file system browser

**Usage**

```r
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`
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)
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


Examples

write(file=file.path(tempdir(), "tmp.fasta"), c(">1
GAAC------ATGC",
  ">2
GAAC------TTGC",
  ">3
GAAC---CCTTTGC",
  ">4
GAA---------GC"))
write(file=file.path(tempdir(), "tmp_expected.fasta"), c(">1
GAAC------ATGCCA-",
  ">2
GAAC------TTGCCA-",
  ">3
GAAC---CCTTTGCCCA",
  ">4
GAA---------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**

```r
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**

Algorithms 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(nnn$membership, 1, function(.x) (sum(.x^2) - 1/K) / (1 - 1/K))
fuzzy <- dunn < 0.05
plot(get(n)$samples[, fuzzy, ], col=nn$clustering[, fuzzy], pch=19)
points(get(n)$samples[fuzzy, ], col="black", pch=1)
TITLE("FANNY")
#
nnewlabels <- 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)$samples, col=newlabels, pch=19)
TITLE("mean-shift")
#
library(mclust)
nnewlabels <- Mclust(get(n)$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("median")
    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")

---

**Gini**

*Compute the simple Gini coefficient*

---

**Description**

Computes the simple Gini coefficient of inequality

**Usage**

```r
Gini(x)
```

**Arguments**

- `x` a numeric vector with non-negative elements
Details

Gini coefficient is a common measure of inequality. Here it presents only for the convenience to have this calculation "outside" of social science R packages (where it commonly presents). Please read elsewhere of its meaning and uses.

Code is based on the 'reldist' package from Mark S. Handcock but simplified to remove the using of weights (as a sideway result, it should be slightly faster).

Value

The Gini coefficient (number between 0 and 1).

Author(s)

Alexey Shipunov

References


Examples

```r
salary <- c(21, 19, 27, 11, 102, 25, 21)
Gini(salary)

new.1000 <- sample((median(salary) - IQR(salary)) : (median(salary) + IQR(salary)), 1000, replace=TRUE)
salary2 <- c(salary, new.1000)
Gini(salary2)

salary3 <- salary[-which.max(salary)]
salary3
Gini(salary3)

salary4 <- c(salary3, 1010)
salary4
Gini(salary4)
```

---

**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 compu-
tation 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 
to number of variables in ‘data.x’
KR.corr  When TRUE (default) the extension of the Gower’s dissimilarity measure pro-
posed 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 ob-
servations 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 vari-
ables; 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 (de-
fault), 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.

Wiley, New York.
See Also
dist, cluster::daisy

Examples

```r
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

```r
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.
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 prediction, 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:

```r
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 teaching 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

```r
## 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"))
```

```r
## 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"))
```

```r
## 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])$x[, 1:2]
iris.nnet.pca <- nnet(Species ~ ., data=cbind(iris[5], iris.p), size=4)
plot(iris.p, type="n", main="nnet")
Gradd(iris.nnet.pca, iris.p, type="tree")
```r
library(class)
iris.p <- prcomp(iris[, -5])$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**

```r
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 resolution
- **Nightsky**: If TRUE, there is 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


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)

Description


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

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  

Description
Converts clustering to matrix

Usage
Hcl2mat(hc1)

Arguments
hc1  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.
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")
aal2.match <- Hclust.match(aa.d1, aa.d2)
heatmap(aal2.match, scale="none")
**Hcoords**

Calculates coordinates of nodes from `hclust` plot

**Description**

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

**Usage**

```r
Hcoords(hcl)
```

**Arguments**

- `hcl` hclust object

**Details**

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

**Examples**

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

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

```r
## 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
sell1 <- 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[sell1][which.min(rowSums(mat[sell1, ]))]text(hcoo[MCCN1, , drop=FALSE], labels="Eastern + Central", pos=1)
```
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")
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_ _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.
**hrahn**

**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, positions correspond with columns of 'data'

**Source**


**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("-", "-", cc))
}

displaychar <- function(selchar) { # add or remove characters
  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")
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 corresponding 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

```r
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))
```
**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**

```r
## '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


Examples

```r
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

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

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

```r
# 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
Jclust

Names: bootstrap, monitor, x, main, xlab, rect.lty, rect.col, rect.xpd, top, lab.pos, lab.offset, lab.col, lab.font

Other: rect.xpd

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 bootstrap 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 jackknife), "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 clusters used.)

Author(s)

Alexey Shipunov

See Also

Bclust, BootA, Fence
Examples

```r
## '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

```r
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 "strictly 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


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 terminal (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:

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 praedictus.* Mandibulae in utroque sexu simplices, i. e. absque appendicibus.

sg. *Abrognathus* Jak.

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

3 (4). *Mandibularum appendices in 5 saepius vix indicatiae, dentiformes saepe obtuse anguliformes, rarius magis evolutae, spiniformes, semper aequales. Corpus supra rude sculptum.*

sg. *Lethrulus* m.

4 (3). *Mandibularum appendices in 5 plus minusve evolutae (saltim 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:
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

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.

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
")

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


Examples

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

| Linechart | Dotchart-like plot for 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

```r
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])
```

Description

Advanced object browser

Usage

```r
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)] <- "-"
nomorequote(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 technique including multidimensional scaling.

**Value**

Numeric vector, one values per column of scores

**Author(s)**

Alexey Shipunov

**Examples**

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

```r
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**

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

---

**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**

```r
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

<table>
<thead>
<tr>
<th>NAME</th>
<th>Island name</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQUARE</td>
<td>Island square, m2</td>
</tr>
<tr>
<td>LAT</td>
<td>Latitude</td>
</tr>
<tr>
<td>LON</td>
<td>Longitude</td>
</tr>
</tbody>
</table>

**Source**


---

**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 trees either "halfcompat" (majority-rule consensus tree) or "allcompat" (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

```r
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)
node.labels(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**


**See Also**

cutree, link{cmdscale}, link{Hcl2mat}

**Examples**

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

library(ape)
(tree.mrh3 <- MRH(as.hclust(compute.brlen(tree)), method="branches"))

---

NC.dist

Normalized Compression Distance

Description
Calculates the normalized compression distance

Usage
NC.dist(data, method="gzip", character=TRUE)

Arguments
- data: Matrix (or data frame) with variables that should be used in the computation of the distance between rows.
- method: Taken from memCompress(): either "gzip", or "bzip2", or "xz"; the last is very slow
- character: Convert to character mode (default), or use as raw?

Details
NC.dist() computes the distance based on the sizes of the compressed vectors. It is calculated as
dissimilarity(x, y) = B(x, y) - max(B(x), B(y)) / min(B(x), B(y))
where B(x) and B(y) are the bytesizes of the compressed 'x' and 'y', and B(x, y) is the compressed bytesize of concatenated 'x' and 'y'. The algorithm uses basic memCompress() function.

If argument is the data frame, NC.dist() internally converts it into the matrix. All columns by default will be converted into character mode (and if 'character=FALSE', into raw). This default behavior allows NC.dist() to be the universal distance which also does not mind NAs and zeroes.

Value
Distance object with distances among rows of 'data'

Author(s)
Alexey Shipunov

References
Normality

See Also

memCompress

Examples

```r
## converts variables into character, universal method
iris.nc <- NC.dist(iris[, -5])
iris.hnc <- hclust(iris.nc, method="ward.D2")
## amazingly, it works even for vectors with length=4 (iris data rows)
plot(prcomp(iris[, -5])$x, col=cutree(iris.hnc, 3))

## using variables as raw, it is good when they are uniform
iris.nc2 <- NC.dist(iris[, -5], character=FALSE)
iris.hnc2 <- hclust(iris.nc2, method="ward.D2")
plot(prcomp(iris[, -5])$x, col=cutree(iris.hnc2, 3))

## bzip2 uses Burrows-Wheeler transform
NC.dist(matrix(runif(100), ncol=10), method="bzip2")
```

Description

Check normality through Shapiro-Wilks test

Usage

```r
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.
Overlap

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 actual hull areas are smaller than with randomized groups (i.e., that actual hulls are better than random).

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

Author(s)
Alexey Shipunov

References

See Also
Hulls, Ellipses

Examples

```r
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)
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

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)
```
```r
## 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
```

### 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

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, lbtex="internal", lbtex=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

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

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, 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
Plot.phylocl

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.
PlotBest.dist

Value

Returns list of clade names.

Author(s)

Alexey Shipunov

See Also

phytools::plot.backbonePhylo()

Examples

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

cl <- data.frame(
  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

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

---

### PlotBest.mdist

**Description**

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

**Usage**

```r
```

**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
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 Ttext() 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)

<table>
<thead>
<tr>
<th>Points</th>
<th>Number of cases in each location reflected in the point size</th>
</tr>
</thead>
</table>

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, ...) 
PPPoints(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 (default), 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(), PPPoints() 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()'.
'PPPoints()' 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 PPPoints() it is group-wise so over-
plotting 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
Examples

```r
## 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
```

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

```r
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**

```r
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**

```r
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)
```
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

` 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

sapply(iris.h, Polycenter)
R.logo

**Description**

Imitation (!) of the modern 'R' logo

**Usage**

```r
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**

```r
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**

```r
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

```r
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 6.498E-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 2.451E-001 3.081E-001 4.531E-002 0.000E+000
  2.563E-001 2.357E-001 2.278E-001 3.008E-001 4.141E-002 6.510E-002 0.000E+000
  8.009E-002 6.611E-002 7.738E-002 9.885E-002 2.260E-001 2.278E-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 5.192E-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 2.295E-001 2.323E-001 2.281E-001 0.000E+000
  2.504E-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 ' "\n
## interactive
file.show(file=file.path(tempdir(), "tmp.nts"))

Read.tri.nts(file=file.path(tempdir(), "tmp.nts"), skip=2)
```

Description

Basic multiple recoding (similar to the 'SQL' left join)

Usage

```r
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 character string)

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

```r
## 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]]),
dna <- "ATTCGGC" # something random
paste(Recode(rev(strsplit(dna, NULL)[[1]]),

## 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)

---

### 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

```r
data(bird.orders, package="ape")
ap::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

```r
Rostova.tbl(X, GROUP, ...)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Data frame or matrix with values</td>
</tr>
<tr>
<td>GROUP</td>
<td>Number of grouping variable</td>
</tr>
<tr>
<td>...</td>
<td>Arguments to 'Cor.vec()'</td>
</tr>
</tbody>
</table>
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


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

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

```r
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)
```

```r
(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)
```

Description

Rresults shell script

Details

'Rresults' is a bash shell script which allows to gather all R input and R textual output into one text file, and (unnamed) R graphical output into 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 useful for the debugging and other non-interactive activities with R scripts as everything is in one place.

The script has one option "-d" which adds the timestamp to the file name of text results. This option also switches R to add sessionInfo() to the end of output.

Author(s)

Alexey Shipunov
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() (de-
veloped 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)
**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) \( s = -\log_2(p) \). 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 \(-\log_2(0.03) = 5\) 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 \(-\log_2(0.25) = 2\) 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**


Examples

S.value(0.05)
S.value(0.01)
S.value(0.1)
S.value(0.0000000001)
S.value(t.test(extra ~ group, data = sleep))
S.value(list(pvalues=c(0.01, 0.002)))

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 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 dioecious)
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 treats 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)
```r
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

<table>
<thead>
<tr>
<th>Squares</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Calculates areas of polygons</td>
</tr>
</tbody>
</table>

#### Usage

```r
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

'\texttt{str()}’ enhanced for data frames

Description

Enhanced ‘\texttt{str()}’: with variable numbers, row names, missing data indication and possibly more

Usage

\texttt{Str(df, as.factor=FALSE)}

Arguments

\begin{itemize}
  \item \texttt{df} Data frame
  \item \texttt{as.factor} Convert character columns to factors?
\end{itemize}

Details

‘\texttt{Str()}’ is an enhanced ‘\texttt{str()}’. ‘\texttt{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 ‘\texttt{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 ‘\texttt{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: \texttt{DescTools::Str()} which uses cycles (slower!), has less features, but works with non-atomic columns.

Value

If ‘\texttt{as.factor=TRUE}’, invisibly outputs the data frame with all character columns converted into factors.
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.
**Tcoords**

*Calculates coordinates of tips from ‘hclust’ plot*

**Value**

Data frame

**Author(s)**

Alexey Shipunov

**Examples**

```r
table2df(table(iris[, 5]))
```

**Description**

Takes the ‘hclust’ plot and calculates coordinates of all tips

**Usage**

```r
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 level 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, requires 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 labels!

## 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
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

```r
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**

```r
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**

```r
(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

```r
## 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)
**Ttcols**

**Text-to-columns**

**Description**

Splits character vector into columns of the matrix based on specified separator

**Usage**

```
Ttcols(text, missed=NA, ...)
```

**Arguments**

- `text` Character vector
- `missed` How to fill empty cells of the result, default is NA
- `...` Arguments to `split()` function

**Details**

Text-to-columns operation is common in spreadsheets. In R, demands for this functionality are likely also high because there are numerous solutions. Below are some most simple, flexible and extendable:

- `do.call(rbind, strsplit(..., split))` – fast and easy but it recycles short rows
- `read.table(text=..., sep=split, fill=TRUE, colClasses="character", header=FALSE)` – does not know how many columns you want; it uses only 5 first lines and requires to specify column names otherwise
- `strcapture()` – needs both rows of equal length _and_ number of future columns
- `stringr::str_split_fixed()` – needs number of columns to make
- `reshape2::colsplit()` – needs column names beforehand
- `tidyr::separate()` – cannot take vectors and also wants explicit column names

`Ttcols()` is fast, simple and flexible solution which does not have problems from above. It handles only one vector at time but it is easy to overcome because it is simple to extend and combine.

**Value**

Matrix of splitted strings without separators, empty cells filled with ‘missed’.

**Author(s)**

Alexey Shipunov

**See Also**

`Fill`
Examples

```r
aa <- c("one,I,i", "two,II", "three", NA, ",", Inf, "2,3,4,-5", 15, ",a,,b")
Ttcols(aa, split=" *, *")

bb <- c("one,I,i", "two,II", "three")
Ttcols(bb, split="," )

Ttcols(row.names(mtcars), split=" ", missed="")
```

---

**Updist**

*Educated distances for semi-supervised clustering*

**Description**

Updates distance matrix to help link or unlink objects

**Usage**

```r
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**

Vicinities

See Also

dist

Examples

```r
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("Earth", "Mercury"))
plot(hclust(aadu))
```

Vicinities

Arrange observations by the distance from center

Description

Uses group centers to order all observations within group

Usage

```r
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()
method

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

```r
## 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
VTcoeffs <- iris[1:150, ,]
VTcoeffs <- Bclust(iris3[, -5], method.d="euclidean", method.c="average", iter=100)
plot(ir3b$hclust, labels=iris3[, 5])
Vclabels(ir3b$hclust, ir3b$vvalues)

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?

Value

Data frame with coefficients, values and tables.
Write.fasta

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

* aplot
  Biarrows, 14
  Cladd, 27
  Ell, 46
  Ellipses, 47
  Gradd, 65
  Hulls, 77
  Missing.map, 96
  Points, 120
  R.logo, 125
* classif
  Biokey, 17
* cluster
  Adj.Rand, 4
  Bclabels, 8
  Bclust, 9
  BootA, 21
  Fence, 55
  Hcl2mat, 70
  Hclust.match, 71
  Hcoords, 72
  Jclust, 81
  MRH, 99
  Tcoords, 141
  Tctext, 142
  Updist, 148
* correlation
  Coeff.det, 34
  Cor, 36
  Cor.vec, 37
  Rostova.tbl, 130
  Topm, 146
* datagen
  Gen.cl.data, 59
* datasets
  atmospheres, 7
  chaetocnema, 26
  classifs, 29
  dolbli, 42
  drosera, 45
  eq, 49
  haltica, 69
  hrahn, 74
  hwc, 79
  keys, 84
  moldino, 96
  plantago, 110
  salix_leaves, 135
* environment
  Ls, 90
  Str, 139
* hplot
  Boxplots, 24
  Clustergram, 32
  Dotcharts, 43
  Ex.boxplot, 50
  Ex.col, 50
  Ex.font, 51
  Ex.lty, 52
  Ex.margins, 52
  Ex.pch, 53
  Ex.plots, 54
  Gridmoon, 68
  Histr, 73
  Linechart, 89
  Pleiad, 112
  Plot.phyloc1, 114
  PlotBest.dist, 116
  PlotBest.hclust, 117
  PlotBest.mdist, 118
  Ploth, 119
  Saynodynamite, 136
* htest
  Normality, 102
  pairwise.Eff, 105
  pairwise.Rro.test, 106
  pairwise.Table2.test, 107
  Rro.test, 133
INDEX

S. value, 134
* interface
  MrBayes, 97
* manip
  %-%, 155
  Aggregate1, 5
  Alldups, 6
  Class. sample, 28
  Coml, 35
  Ditto, 39
  Fill, 57
  Gap.code, 58
  Pull, 124
  Read.fasta, 126
  Read.tri.nts, 126
  Recode, 127
  Root1, 129
  Table2df, 140
  Tobin, 144
  Ttcols, 147
  Write.fasta, 153
* math
  Phyllotaxis, 108
* misc
  Life, 88
  Miney, 93
* multivariate
  BestOverlap, 12
  BootKNN, 22
  BootRF, 23
  Classproj, 31
  DNN, 40
  Gower.dist, 63
  Infill, 80
  MDSv, 92
  Misclass, 94
  Overlap, 103
  Pinhull, 109
  Polyarea, 122
  Polyceter, 123
  Rpart2newick, 131
  SM.dist, 137
  Squares, 138
  Vicinities, 149
* univar
  CVs, 38
  Gini, 62
  K, 83

Mag, 91
VTcoeffs, 152
* utilities
  Cdate, 25
  Files, 56
  Rresults, 132
  Toclip, 145
  Xpager, 154
  %-%, 155

Adj.Rand, 4, 95
aggregate, 6
Aggregate1, 5
Alldups, 6
atmospheres, 7
Bclabels, 8, 11
Bclust, 8, 9, 21, 71, 72, 82
BestOverlap, 12
Biarrows, 14
Biokey, 17, 30, 87
biplot, 15
boot.phylo, 21
BootA, 11, 20, 21, 82
BootKNN, 22
BootRF, 23
boxplot, 25, 50, 136
boxplot.stats, 78
Boxplots, 24, 90
Cdate, 25
chaetocnema, 26
Cladd, 27
Class. sample, 28
classifs, 19, 29
Classproj, 31, 150
Clustergram, 32
Coeff.det, 34
colors, 51
Coml, 35
Cor, 36, 146
Cor.vec, 37, 131
Cor2 (Cor), 36
Ctime (Cdate), 25
cutree, 100
CVs, 38
daisy, 65
density, 73
dir, 56

Dist, 34, 65, 137, 149

Ditto, 39, 57

DNN, 40

Dnn, 22

Dnn (DNN), 40
dobl1i, 42

Dotchart (Dotcharts), 43
dotchart, 44

Dotchart1 (Dotcharts), 43

Dotchart3, 25

Dotchart3 (Dotcharts), 43

Dotcharts, 43
drosera, 45
duplicated, 7

Ell, 46, 125

Ellipses, 47, 78, 104, 109, 122, 123, 138, 150

eq, 49
eq.1 (eq), 49

eq.s (eq), 49

Ex.boxplot, 50

Ex.col, 50

Ex.cols (Ex.col), 50

Ex.font, 51

Ex.fonts (Ex.font), 51

Ex.lines (Ex.lty), 52

Ex.lty, 52

Ex.margins, 52

Ex.pch, 53

Ex.plots, 54

Ex.points (Ex.pch), 53

Ex.types (Ex.plots), 54

Fence, 55, 82, 142

Fibonacci (Phyllotaxis), 108

Files, 56

Fill, 39, 57, 147

Gap.code, 58

Gen.cl.data, 59

getwd, 56

Gini, 62

Gower.dist, 63

Gradd, 65

Gridmoon, 68

haltica, 69

Hcl2mat, 11, 70

hclust, 10, 34, 55, 142, 143

Hclust.match, 71

Hcoords, 11, 72

hist, 73, 103

Histr, 73

hrahn, 74

Hulls, 48, 77, 104, 109, 122, 123, 138, 150

hwc, 79

hwc2 (hwc), 79

hwc3 (hwc), 79

Infill, 80

Jclust, 11, 81

cjitter, 122

K, 83

ekkeys, 19, 84

kmeans, 34

knn, 22, 41

Life, 88

Linechart, 25, 44, 89

lines, 52

lm, 27

ls, 90

ls, 91

Mag, 91

MDSv, 92

memCompress, 102

Miney, 93

Misclass, 5, 94

Missing.map, 96

moldino, 96

moldino1 (moldino), 96

MrBayes, 97

mrbayes, 98

MRH, 71, 99

NC.dist, 101

Normality, 102

Numranks, 40

Numranks (Biokey), 17

Overlap, 48, 78, 103, 109

pairwise.Eff, 105

pairwise.Rro.test, 106

pairwise.Table2.test, 107
palette, 51
par, 51, 53, 54
Phyllotaxis, 108
Pinhull, 48, 109
plantago, 110
Pleiad, 112
plot.Bclust (Bclust), 9
plot.Infill (Infill), 80
plot.Jclust (Jclust), 81
Plot.phylocl, 114
PlotBest.dist, 116, 119
PlotBest.hclust, 117
PlotBest.mdist, 118
Plotth, 119, 142, 143
Points, 120
points, 54
Polyarea, 122
Polycenter, 123
PPoints (Points), 120
print.Jclust (Jclust), 81
print.K (K), 83
Pull, 124

qqnorm, 103

R.logo, 125
rainbow, 51
randomForest, 24
Read.fasta, 126
Read.tri.nts, 126
Recode, 127
Recode4 (Recode), 127
Recode4R (Recode), 127
RecodeR (Recode), 127
rnorm, 73, 103
root, 130
Root1, 129
Rostova.tbl, 38, 130
Rpart2newick, 131
Rresults, 132
Rro.test, 106, 133
S.value, 134
salix_leaves, 135
salix_plants (salix_leaves), 135
salix_pop (salix_leaves), 135
Save.history (Cdate), 25
savehistory, 26
Saynodynamite, 136

setwd, 56
SM.dist, 137
Squares, 122, 123, 138
Str, 139
str, 140
summary.Coml (Coml), 35
summary.Infill (Infill), 80
summary.K (K), 83
summary.Overlap (Overlap), 103
sunflowerplot, 122
Table2df, 140
Tcoords, 55, 120, 141, 143
Tctext, 142, 142
Tobin, 144
Toclip, 145
Topm, 146
Ttcols, 147

Updist, 148
Vicinities, 149
VTcoeffs, 152

Write.fasta, 153

Xpager, 154