Package ‘dendextend’

February 19, 2015

Type Package

Title Extending R’s Dendrogram Functionality

Version 0.18.3

Date 2015-01-31

Description Offers a set of functions for extending
dendrogram objects in R, letting you visualize and compare trees of
hierarchical clusterings. You can (1) Adjust a tree graphical parameters
- the color, size, type, etc of its branches, nodes and labels. (2)
Visually and statistically compare different dendrograms to one another.

Depends R (>= 3.0.0)

Imports utils, stats, magrittr (>= 1.0.1), whisker

Suggests dendextendRcpp (>= 0.5.0), knitr, testthat, colorspace,
ggplot2, plyr, ape, profdpm, microbenchmark, gplots, DendSer,
dynamicTreeCut, pvclust

Enhances ggdendro, labeltodendro, dendroextras, data.table

VignetteBuilder knitr

License GPL-2 | GPL-3

URL https://github.com/talgalili/dendextend/,
http://cran.r-project.org/web/packages/dendextend/,
http://www.r-statistics.com/tag/dendextend/

BugReports https://github.com/talgalili/dendextend/issues

Author Tal Galili [aut, cre, cph] (http://www.r-statistics.com),
Gavin Simpson [ctb],
Gregory Jefferis [ctb] (imported code from his dendroextras package),
Marco Gallotta [ctb] (a.k.a: marcog),
Johan Renaudie [ctb] (https://github.com/plannapus),
The R Core Team [ctb] (Thanks for the Infrastructure, and code in the
examples),
Kurt Hornik [ctb],
Uwe Ligges [ctb],
Andrej-Nikolai Spiess [ctb],
Steve Horvath [ctb],
Peter Langfelder [ctb],
skullkey [ctb],
Mark Van Der Loo [ctb] (https://github/markvanderloo d3dendrogram),
Andrie de Vries [ctb] (ggdendro author),
Yoav Benjamini [ths]

Maintainer Tal Galili <tal.galili@gmail.com>

NeedsCompilation no
Repository CRAN
Date/Publication 2015-01-31 21:57:37

R topics documented:

dendextend-package ........................................... 4
all.equal.dendrogram ......................................... 5
all_couple_rotations_at_k .................................. 6
all_unique ..................................................... 7
as.dendlist .................................................. 8
as.phylo.dendrogram ......................................... 9
assign_dendextend_options .................................. 10
assign_values_to_branches_edgePar .......................... 10
assign_values_to_leaves_edgePar ............................ 11
assign_values_to_leaves_nodePar ............................ 12
assign_values_to_nodes_nodePar ............................. 13
as_hclust_fixed ............................................. 15
bakers_gamma_for_2_k_matrix ............................... 15
Bk ......................................................... 16
Bk_permutations .............................................. 18
Bk_plot ...................................................... 20
branches_attr_by_clusters .................................. 23
branches_attr_by_labels .................................... 25
click_rotate ................................................. 26
collapse_branch .............................................. 28
colored_bars ................................................ 29
color_branches .............................................. 31
color_labels ................................................ 34
cor.dendlist ................................................ 36
cor_bakers_gamma .......................................... 37
cor_cophenetic ............................................. 39
count_terminal_nodes ...................................... 41
cutree ....................................................... 42
cutree_1h.dendrogram ...................................... 46
cutree_1k.dendrogram ...................................... 47
cut_lower_fun ............................................... 49
d3dendrogram ............................................... 50
d3dendro_options ........................................... 51
R topics documented:

dendextend_options .................................. 52
dendlist ............................................. 53
DendSer.dendrogram .................................. 54
dend_diff ........................................... 55
dist.dendlist ........................................ 56
distinct_edges ...................................... 57
dist_long ............................................ 58
duplicate_leaf ...................................... 58
entanglement ....................................... 59
fac2num ............................................. 62
fix_members_attr.dendrogram ....................... 62
flatten.dendrogram .................................. 63
flip_leaves ......................................... 64
FM_index ............................................ 65
FM_index_permutation ................................ 67
FM_index_profdpm ................................... 69
FM_index_R ......................................... 71
get_branches_heights ................................ 72
get_childrens_heights ................................ 73
get_leaves_attr ...................................... 74
get_leaves_nodePar ................................... 75
get_nodes_attr ....................................... 76
get_nodes_xy ......................................... 78
get_root_branches_attr ............................... 79
ggdend ............................................... 80
hang.dendrogram ..................................... 83
heights_per_k.dendrogram ............................. 84
highlight_distinct_edges .............................. 85
identify.dendrogram .................................. 87
intersect_trees ....................................... 88
is.dendlist ......................................... 89
is.dendrogram ....................................... 90
is.hclust .......................................... 90
is.natural.number ................................... 91
is.phylo ............................................ 92
is_null_list ....................................... 92
labels< ............................................ 93
labels_colors ....................................... 95
ladderize .......................................... 97
leaf_Colors ......................................... 98
lowest_common_branch ............................... 99
match_order_by_labels ................................ 99
match_order_dendrogram_by_old_order ............ 100
na_locf ........................................... 102
nleaves ............................................ 103
nnodes ............................................. 104
noded_with_condition ................................ 105
order.dendrogram< .................................. 106
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>order.hclust</td>
<td>108</td>
</tr>
<tr>
<td>partition_leaves</td>
<td>108</td>
</tr>
<tr>
<td>plot_horiz.dendrogram</td>
<td>109</td>
</tr>
<tr>
<td>prune</td>
<td>111</td>
</tr>
<tr>
<td>prune_leaf</td>
<td>112</td>
</tr>
<tr>
<td>pvclust_show_signif</td>
<td>113</td>
</tr>
<tr>
<td>pvclust_show_signif_gradient</td>
<td>114</td>
</tr>
<tr>
<td>raise.dendrogram</td>
<td>116</td>
</tr>
<tr>
<td>rank_branches</td>
<td>116</td>
</tr>
<tr>
<td>rank_order.dendrogram</td>
<td>117</td>
</tr>
<tr>
<td>rect.dendrogram</td>
<td>118</td>
</tr>
<tr>
<td>remove_branches_edgePar</td>
<td>120</td>
</tr>
<tr>
<td>remove_leaves_nodePar</td>
<td>121</td>
</tr>
<tr>
<td>remove_nodes_nodePar</td>
<td>122</td>
</tr>
<tr>
<td>rllply</td>
<td>122</td>
</tr>
<tr>
<td>rotate</td>
<td>123</td>
</tr>
<tr>
<td>rotate_DendSer</td>
<td>126</td>
</tr>
<tr>
<td>sample.dendrogram</td>
<td>127</td>
</tr>
<tr>
<td>set</td>
<td>128</td>
</tr>
<tr>
<td>shuffle</td>
<td>132</td>
</tr>
<tr>
<td>sort_2_clusters_vectors</td>
<td>134</td>
</tr>
<tr>
<td>sort_dist_mat</td>
<td>135</td>
</tr>
<tr>
<td>sort_levels_values</td>
<td>136</td>
</tr>
<tr>
<td>tanglegram</td>
<td>137</td>
</tr>
<tr>
<td>theme_dendrogram</td>
<td>141</td>
</tr>
<tr>
<td>unbranch</td>
<td>142</td>
</tr>
<tr>
<td>unclass_dend</td>
<td>143</td>
</tr>
<tr>
<td>untangle</td>
<td>144</td>
</tr>
<tr>
<td>untangle_DendSer</td>
<td>145</td>
</tr>
<tr>
<td>untangle_random_search</td>
<td>146</td>
</tr>
<tr>
<td>untangle_step_rotate_1side</td>
<td>148</td>
</tr>
<tr>
<td>untangle_step_rotate_2side</td>
<td>149</td>
</tr>
<tr>
<td>which_leaf</td>
<td>151</td>
</tr>
<tr>
<td>which_node</td>
<td>152</td>
</tr>
<tr>
<td>%&gt;%</td>
<td>153</td>
</tr>
</tbody>
</table>

**Index**

---

**dendextend-package**

*Functions for extending dendrogram objects*

**Description**

Functions for extending dendrogram objects

**See Also**

*dendrogram, hclust* in *stats* package.
all.equal.dendrogram  

Global Comparison of two (or more) dendrograms

Description

This function makes a global comparison of two or more dendrograms trees.

Usage

## S3 method for class 'equal.dendrogram'
all(target, current, use.edge.length = TRUE,
    use.tip.label.order = FALSE, use.tip.label = TRUE, use.topology = TRUE,
    tolerance = .Machine$double.eps^0.5, scale = NULL, ...)

Arguments

- **target**: an object of type dendrogram or dendlist
- **current**: an object of type dendrogram
- **use.edge.length**: logical (TRUE). If to check branches' heights.
- **use.tip.label.order**: logical (FALSE). If to check labels are in the same and in identical order
- **use.tip.label**: logical (TRUE). If to check that labels are the same (regardless of order)
- **use.topology**: logical (TRUE). If to check the existence of distinct edges
- **tolerance**: the numeric tolerance used to compare the branch lengths.
- **scale**: a positive number (NULL as default), comparison of branch height is made after scaling (i.e., dividing) them by this number.
- **...**: Ignored.

Value

Either TRUE (NULL for attr.all.equal) or a vector of mode "character" describing the differences between target and current.

See Also

all.equal, all.equal.phylo, identical

Examples

## Not run:

```r
set.seed(23235)
s <- sample(1:150, 10 )
dend1 <- iris[ss,-5]  #> dist  #> hclust("com")  #> as.dendrogram
dend2 <- iris[ss,-5]  #> dist  #> hclust("single")  #> as.dendrogram
```
all_couple_rotations_at_k

Rotate tree branches for k

Description
Given a tree and a k number of clusters, the tree is rotated so that the extra clusters added from k-1 to k clusters are flipped.
This is useful for finding good trees for a tanglegram.

Usage
all_couple_rotations_at_k(dend, k, dend_heights_per_k, ...)

Arguments
- dend: a dendrogram object
- k: integer scalar with the number of clusters the tree should be cut into.
- dend_heights_per_k: a named vector that resulted from running heights_per_k.dendrogram. When running the function many times, supplying this object will help improve the running time if using the cutree.dendrogram method.
- ...: not used

Value
A list with dendrogram objects with all the possible rotations for k clusters (beyond the k-1 clusters!).
all_unique

See Also
tanglegram, match_order_by_labels, entanglement, flip_leaves.

Examples

### Not run:
```r
dend1 <- USArrests[1:5,] %>% dist %>% hclust %>% as.dendrogram
dend2 <- all_couple_rotations_at_k(dend1, k=2)[[2]]
tanglegram(dend1,dend2)
entanglement(dend1,dend2, L = 2) # 0.5

dend2 <- all_couple_rotations_at_k(dend1, k=3)[[2]]
tanglegram(dend1,dend2)
entanglement(dend1,dend2, L = 2) # 0.4

dend2 <- all_couple_rotations_at_k(dend1, k=4)[[2]]
tanglegram(dend1,dend2)
entanglement(dend1,dend2, L = 2) # 0.05

### End(Not run)
```

---

**all_unique**

*Check if all the elements in a vector are unique*

**Description**

Check if all the elements in a vector are unique

**Usage**

```r
all_unique(x, ...)
```

**Arguments**

- `x` : a vector
- `...` : ignored.

**Value**

logical (are all the elements in the vector unique)

**Source**


**See Also**

unique
Examples

```r
call_unique(c(1,5, 1,1))
call_unique(c(1,1,2))
call_unique(c(1,1,2, 3,3,3))
call_unique(c(1,3,2))
call_unique(c(1:10))
```

---

**as.dendlist**

*Try to coerce something into a dendlist*

Description

It removes stuff that are not dendrogram/dendlist and turns what is left into a dendlist

Usage

```r
as.dendlist(x, ...)
```

Arguments

- **x**: a list with several dendrogram/hclust/phylo or dendlist objects and other junk that should be omitted.
- **...**: NOT USED

Value

A list of class dendlist where each item is a dendrogram

Examples

```r
## Not run:
dend <- iris[, -5] %>% dist %>% hclust %>% as.dendrogram
dend2 <- iris[, -5] %>% dist %>% hclust(method = "single") %>% as.dendrogram
x = list(dend, 1, dend2)
as.dendlist(x)
```

## End(Not run)
as.phylo.dendrogram

Convert a dendrogram into phylo

Description

Based on as.hclust.dendrogram with as.phylo.hclust
In the future I hope a more direct link will be made.

Usage

as.phylo.dendrogram(x, ...)

Arguments

x
  a dendrogram

... ignored.

Value

A phylo class object

See Also

as.dendrogram, as.hclust, as.phylo

Examples

## Not run:

library(dendextend)
library(ape)
dend <- iris[,3:5] %>% dist %>% hclust %>% as.dendrogram
dend2 <- as.phylo(dend)
plot(dend2, type = "fan")

# Also possible to with ggplot2 :)

library(dendextend)
library(ggplot2)
library(ggdendro)
dend <- iris[,3:5] %>% dist %>% hclust %>% as.dendrogram
# Rectangular lines
ddata <- dend %>% dendro_data(type = "rectangle")
p <- ggplot(segment(ddata)) +
  geom_segment(aes(x = x, y = y, xend = xend, yend = yend)) +
  coord_flip() +
  scale_y_reverse(expand = c(0.2, 0)) + coord_polar(theta="x")
p
assign_dendextend_options

*Populates dendextend functions into dendextend_options*

**Description**
Populates dendextend functions into dendextend_options

**Usage**
assign_dendextend_options()

assign_values_to_branches_edgepar

*Assigns values to edgePar of dendrogram's branches*

**Description**
Go through the dendrogram branches and updates the values inside its edgePar
If the value has Inf then the value in edgePar will not be changed.

**Usage**
assign_values_to_branches_edgepar(object, value, edgePar, skip_leaves = FALSE, warn = dendextend_options("warn"), ...)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>a dendrogram object</td>
</tr>
<tr>
<td>value</td>
<td>a new value scalar for the edgePar attribute.</td>
</tr>
<tr>
<td>edgePar</td>
<td>a character indicating the value inside edgePar to adjust. Can be either &quot;col&quot;, &quot;lty&quot;, or &quot;lwd&quot;.</td>
</tr>
<tr>
<td>skip_leaves</td>
<td>logical (FALSE) - should the leaves be skipped/ignored?</td>
</tr>
<tr>
<td>warn</td>
<td>logical (default from dendextend_options(&quot;warn&quot;) is FALSE). Set if warning are to be issued, it is safer to keep this at TRUE, but for keeping the noise down, the default is FALSE.</td>
</tr>
<tr>
<td>...</td>
<td>not used</td>
</tr>
</tbody>
</table>
**assign_values_to_leaves_edgePar**

Assign values to edgePar of dendrogram’s leaves

**Value**

A dendrogram, after adjusting the edgePar attribute in all of its branches.

**See Also**

get_root_branches_attr

**Examples**

```r
## Not run:

dend <- USArrests[1:5,] %>% dist %>% hclust %>% as.dendrogram
plot(dend)
dend <- assign_values_to_branches_edgePar(object=dend, value = 2, edgePar = "lwd")
plot(dend)
dend <- assign_values_to_branches_edgePar(object=dend, value = 2, edgePar = "col")
plot(dend)
dend <- assign_values_to_branches_edgePar(object=dend, value = 2, edgePar = "lty")
plot(dend)

## End(Not run)
```

**Description**

Go through the dendrogram leaves and updates the values inside its edgePar. If the value has Inf then the value in edgePar will not be changed.

**Usage**

```r
assign_values_to_leaves_edgePar(object, value, edgePar, 
  warn = dendextend_options("warn"), ...)
```

**Arguments**

- **object**: a dendrogram object
- **value**: a new value vector for the edgePar attribute. It should be the same length as the number of leaves in the tree. If not, it will recycle the value and issue a warning.
- **edgePar**: the value inside edgePar to adjust.
- **warn**: logical (default from dendextend_options("warn") is FALSE). Set if warning are to be issued, it is safer to keep this at TRUE, but for keeping the noise down, the default is FALSE.
- **...**: not used
**assign_values_to_leaves_nodePar**

**Value**

A dendrogram, after adjusting the edgePar attribute in all of its leaves.

**See Also**

get_leaves_attr, linkassign_values_to_leaves_nodePar

**Examples**

```r
## Not run:

dend <- USArrests[1:5,] %>% dist %>% hclust("ave") %>% as.dendrogram

plot(dend)
dend <- assign_values_to_leaves_edgePar(object=dend, value = c(3,2), edgePar = "col")
plot(dend)
dend <- assign_values_to_leaves_edgePar(object=dend, value = c(3,2), edgePar = "lwd")
plot(dend)
dend <- assign_values_to_leaves_edgePar(object=dend, value = c(3,2), edgePar = "lty")
plot(dend)

get_leaves_attr(dend, "edgePar", simplify=FALSE)

## End(Not run)
```

---

**assign_values_to_leaves_nodePar**

*Assign values to nodePar of dendrogram's leaves*

**Description**

Go through the dendrogram leaves and updates the values inside its nodePar

If the value has Inf then the value in edgePar will not be changed.

**Usage**

```r
assign_values_to_leaves_nodePar(object, value, nodePar,
    warn = dendextend_options("warn"), ...)
```

**Arguments**

- **object**
  - a dendrogram object
- **value**
  - a new value vector for the nodePar attribute. It should be the same length as the number of leaves in the tree. If not, it will recycle the value and issue a warning.
- **nodePar**
  - the value inside nodePar to adjust.
assign_values_to_nodes_nodePar

warn logical (default from dendextend_options("warn") is FALSE). Set if warning are to be issued, it is safer to keep this at TRUE, but for keeping the noise down, the default is FALSE.

Value

A dendrogram, after adjusting the nodePar attribute in all of its leaves,

See Also

generate_leaves_attr

Examples

## Not run:

dend <- USArrests[1:5,] > dist >%>% hclust("ave") >%>% as.dendrogram

# reproduces "labels_colors<-"
# although it does force us to run through the tree twice,
# hence "labels_colors<-" is better...
plot(dend)
dend <- assign_values_to_leaves_nodePar(object=dend, value = c(3,2), nodePar = "lab.col")
plot(dend)

plot(dend)
dend <- assign_values_to_leaves_nodePar(dend, 1, "pch")
plot(dend)
# fix the annoying pch=1:
plot(dend)
dend <- assign_values_to_leaves_nodePar(dend, NA, "pch")
plot(dend)
# adjust the cex:
plot(dend)
dend <- assign_values_to_leaves_nodePar(dend, 19, "pch")
dend <- assign_values_to_leaves_nodePar(dend, 2, "lab.cex")
plot(dend)

get_leaves_attr(dend, "nodePar", simplify=FALSE)

## End(Not run)

---

assign_values_to_nodes_nodePar

Assign values to nodePar of dendrogram's nodes

Description

Go through the dendrogram nodes and updates the values inside its nodePar
If the value has Inf then the value in edgePar will not be changed.
assign_values_to_nodes_nodePar

Usage

assign_values_to_nodes_nodePar(object, value, nodePar = c("pch", "cex", "col", "xpd", "bg"), warn = dendextend_options("warn"), ...)

Arguments

object a dendrogram object

value a new value vector for the nodePar attribute. It should be the same length as the number of nodes in the tree. If not, it will recycle the value and issue a warning.

nodePar the value inside nodePar to adjust. This may contain components named pch, cex, col, xpd, and/or bg.

warn logical (default from dendextend_options("warn") is FALSE). Set if warning are to be issued, it is safer to keep this at TRUE, but for keeping the noise down, the default is FALSE.

... not used

Value

A dendrogram, after adjusting the nodePar attribute in all of its nodes,

See Also

get_leaves_attr, assign_values_to_leaves_nodePar

Examples

## Not run:

dend <- USArrests[1:5,] %>% dist %>% hclust("ave") %>% as.dendrogram

# reproduces "labels_colors<-"
# although it does force us to run through the tree twice,
# hence "labels_colors<-" is better...
plot(dend)
dend2 <- dend %>%
  assign_values_to_nodes_nodePar(value = 19, nodePar = "pch") %>%
  assign_values_to_nodes_nodePar(value = c(1,2), nodePar = "cex") %>%
  assign_values_to_nodes_nodePar(value = c(2,1), nodePar = "col")
plot(dend2)

## End(Not run)
as_hclust_fixed  

**Convert dendrogram Objects to Class hclust**

**Description**

Convert dendrogram Objects to Class hclust while preserving the call/method/dist.method values of the original hclust object (hc)

**Usage**

```r
as_hclust_fixed(x, hc, ...)
```

**Arguments**

- `x`: any object which has an as.hclust method. (mostly used for dendrogram)
- `hc`: an old hclust object from which to re-use the call/method/dist.method values
- `...`: passed to as.hclust

**Value**

An hclust object (from a dendrogram) with the original hclust call/method/dist.method values

**See Also**

as.hclust

**Examples**

```r
hc <- hclust(dist(USArrests[,1:3]), "ave")
dend <- as.dendrogram(hc)

as.hclust(dend)
as_hclust_fixed(dend, hc)
```

---

bakers_gamma_for_2_k_matrix  

**Bakers Gamma for two k matrices**

**Description**

Bakers Gamma for two k matrices

**Usage**

```r
bakers_gamma_for_2_k_matrix(k_matrix_tree1, k_matrix_tree2, to_plot = FALSE)
```
Arguments

k_matrix_tree1  a matrix of k cluster groupings from a dendrogram
k_matrix_tree2  a (second) matrix of k cluster groupings from a dendrogram
to_plot        logical (FALSE). Should a scatterplot be plotted, showing the correlation between the lowest shared branch between two items in the two compared trees.

Value

Baker’s Gamma coefficient.

See Also

cor_bakers_gamma

Description

Bk is the calculation of Fowlkes-Mallows index for a series of k cuts for two dendrograms.

Usage

Bk(tree1, tree2, k, include_EV = TRUE, warn = dendextend_options("warn"),
   ...)

Arguments

tree1        a dendrogram/hclust/phylo object.
tree2        a dendrogram/hclust/phylo object.
k           an integer scalar or vector with the desired number of cluster groups. If missing - the Bk will be calculated for a default k range of 2:(nleaves-1). No point in checking k=1/k=n, since both will give Bk=1.
include_EV   logical (TRUE). Should we calculate expectancy and variance of the FM Index under null hypothesis of no relation between the clusterings? If TRUE (Default) - then the FM_index_R function, else (FALSE) we use the (faster) FM_index_profdpm function.
warn         logical (default from dendextend_options("warn") is FALSE). Set if warning are to be issued, it is safer to keep this at TRUE, but for keeping the noise down, the default is FALSE.

...  Ignored (passed to FM_index_R/FM_index_profdpm).
Details

From Wikipedia:

Fowlkes-Mallows index (see references) is an external evaluation method that is used to determine the similarity between two clusterings (clusters obtained after a clustering algorithm). This measure of similarity could be either between two hierarchical clusterings or a clustering and a benchmark classification. A higher the value for the Fowlkes-Mallows index indicates a greater similarity between the clusters and the benchmark classifications.

Value

A list (of k’s length) of Fowlkes-Mallows index between two dendrogram for a scalar/vector of k values. The names of the lists’ items is the k for which it was calculated.

References


http://en.wikipedia.org/wiki/Fowlkes-Mallows_index

See Also

FM_index, cor_bakers_gamma

Examples

```r
## Not run:

set.seed(23235)
ss <- TRUE # sample(1:150, 10 )
hc1 <- hclust(dist(iris[ss, -5]), "com")
hc2 <- hclust(dist(iris[ss, -5]), "single")
tree1 <- as.dendrogram(hc1)
tree2 <- as.dendrogram(hc2)
#  cutree(tree1)

Bk(hc1, hc2, k = 3)
Bk(hc1, hc2, k = 2:10)
Bk(hc1, hc2)

Bk(tree1, tree2, k = 3)
Bk(tree1, tree2, k = 2:5)

system.time(Bk(hc1, hc2, k = 2:5)) # 0.01
system.time(Bk(hc1, hc2)) # 1.28
system.time(Bk(tree1, tree2, k = 2:5)) # 0.24 # after fixes.
system.time(Bk(tree1, tree2, k = 2:10)) # 0.31 # after fixes.
system.time(Bk(tree1, tree2)) # 7.85
Bk(tree1, tree2, k= 99:101)

y <- Bk(hc1, hc2, k = 2:10)
```
Bk_permutations

Bk permutation - Calculating Fowlkes-Mallows Index for two dendrogram

Description

Bk is the calculation of Fowlkes-Mallows index for a series of k cuts for two dendrograms.

Bk permutation calculates the Bk under the null hypothesis of no similarity between the two trees by randomly shuffling the labels of the two trees and calculating their Bk.

Usage

Bk_permutations(tree1, tree2, k, R = 1000, warn = dendextend_options("warn"), ...)

Arguments

tree1 a dendrogram/hclust/phylo object.
tree2 a dendrogram/hclust/phylo object.
k an integer scalar or vector with the desired number of cluster groups. If missing - the Bk will be calculated for a default k range of 2:(nleaves-1). No point in checking k=1/k=n, since both will give Bk=1.
R integer (Default is 1000). The number of Bk permutation to perform for each k.
warn logical (default from dendextend_options("warn") is FALSE). Set if warning are to be issued, it is safer to keep this at TRUE, but for keeping the noise down, the default is FALSE. If set to TRUE, extra checks are made to verify that the two clusters have the same size and the same labels.
... Ignored (passed to FM_index_R/FM_index_profdpm).
Details

From Wikipedia:

Fowlkes-Mallows index (see references) is an external evaluation method that is used to determine the similarity between two clusterings (clusters obtained after a clustering algorithm). This measure of similarity could be either between two hierarchical clusterings or a clustering and a benchmark classification. A higher the value for the Fowlkes-Mallows index indicates a greater similarity between the clusters and the benchmark classifications.

Value

A list (of the length of k’s), where each element of the list has R (number of permutations) calculations of Fowlkes-Mallows index between two dendrogram after having their labels shuffled.

The names of the lists’ items is the k for which it was calculated.

References


http://en.wikipedia.org/wiki/Fowlkes-Mallows_index

See Also

FM_index, Bk

Examples

```r
## Not run:

set.seed(23235)
ss <- TRUE  # sample(1:150, 10 )
hc1 <- hclust(dist(iris[ss, -5]), "com")
hc2 <- hclust(dist(iris[ss, -5]), "single")
# tree1 <- as.treeogram(hc1)
# tree2 <- as.treeogram(hc2)
# cutree(tree1)

some_Bk <- Bk(hc1, hc2, k = 20)
some_Bk_permu <- Bk_permutations(hc1, hc2, k = 20)

# we can see that the Bk is much higher than the permutation Bks:
plot(x=rep(1,1000), y= some_Bk_permu[[1]],
main = "Bk distribution under H0",
ylim = c(0,1))
points(1, y= some_Bk, pch = 19, col = 2 )

## End(Not run)
```
Bk_plot

Bk plot - plotting the Fowlkes-Mallows Index of two dendrogram for various k's

Description

Bk is the calculation of Fowlkes-Mallows index for a series of k cuts for two dendrograms. A Bk plot is simply a scatter plot of Bk versus k. This plot helps in identifying the similarity between two dendrograms in different levels of k (number of clusters).

Usage

Bk_plot(tree1, tree2, k, add_E = TRUE, rejection_line_asymptotic = TRUE, rejection_line_permutation = FALSE, R = 1000, k_permutation, conf.level = 0.95, p.adjust.methods = c("none", "bonferroni"), col_line_Bk = 1, col_line_asymptotic = 2, col_line_permutation = 4, warn = dendextend_options("warn"), main = "Bk plot", xlab = "k (number of clusters)", ylab = "Bk (Fowlkes-Mallows Index)", xlim, ylim = c(0, 1), try_cutree_hclust = TRUE, ...)

Arguments

tree1 a dendrogram/hclust/phylo object.
tree2 a dendrogram/hclust/phylo object.
k an integer scalar or vector with the desired number of cluster groups. If missing - the Bk will be calculated for a default k range of 2:(nleaves-1). No point in checking k=1/k=n, since both will give Bk=1.
add_E logical (TRUE). Should we add a line of the Expected Bk value for each k, under the null hypothesis of no relation between the clusterings?
rejection_line_asymptotic logical (TRUE). Should we add a line of the one sided rejection region based on the asymptotic distribution of Bk values, for each k, under the null hypothesis of no relation between the clusterings?
rejection_line_permutation logical (FALSE). Should we add a line of the one sided rejection region based on the asymptotic distribution of Bk values, for each k, under the null hypothesis of no relation between the clusterings?
R integer (Default is 1000). The number of Bk permutation to perform for each k. Applicable only if rejection_line_permutation is TRUE.
k_permutation the k's to be used for permutation (sometimes we might be only interested in some k's and it is not important to run the simulation for all possible ks). If missing - k itself will be used.
conf.level the level of one sided confidence interval used for creation of the rejection lines.
Fowlkes-Mallows index (see references) is an external evaluation method that is used to determine the similarity between two clusterings (clusters obtained after a clustering algorithm). This measure of similarity could be either between two hierarchical clusterings or a clustering and a benchmark classification. A higher the value for the Fowlkes-Mallows index indicates a greater similarity between the clusters and the benchmark classifications.

The default Bk plot comes with a line with dots (type "b") of the Bk values. Also with a fragmented (lty=2) line (of the same color) of the expected Bk line under H0. And a solid red line of the upper critical Bk values for rejection.

Value

After plotting the Bk plot. Returns (invisible) the output of the elements used for constructing the plot: The Bk values, Bk permutations (if used), Bk theoretical values, etc.
References


http://en.wikipedia.org/wiki/Fowlkes-Mallows_index

See Also

FM_index, Bk, Bk_permutations

Examples

## Not run:

```r
set.seed(23235)
ss <- TRUE # sample(1:150, 10)
hc1 <- hclust(dist(iris[ss,-5]), "com")
hc2 <- hclust(dist(iris[ss,-5]), "single")
# tree1 <- as.teerogram(hc1)
# tree2 <- as.teerogram(hc2)
# cutree(tree1)

Bk_plot(hc1, hc2, k = 2:20, xlim = c(2,149))
Bk_plot(hc1, hc2)

Bk_plot(hc1, hc2, k = 3)
Bk_plot(hc1, hc2, k = 3:10)
Bk_plot(hc1, hc2)
Bk_plot(hc1, hc2, p.adjust.methods="bonferroni") # higher rejection lines

# this one can take a bit of time:
Bk_plot(hc1, hc2, rejection_line_permutation=TRUE,
k_permutation = c(2,4,6,8,10,20,30,40,50), R = 100)
# we can see that the permutation line is VERY close to the asymptotic line.
# This is great since it means one can often use the asymptotic results
# Without having to do many simulations.

# works just as well for dendrograms:
dend1 <- as.dendrogram(hc1)
dend2 <- as.dendrogram(hc2)
Bk_plot(dend1, dend2, k = 2:3, try_cutree_hclust = FALSE) # slower than hclust, but works...
Bk_plot(hc1, dend2, k = 2:3, try_cutree_hclust = FALSE) # slower than hclust, but works...
Bk_plot(dend1, dend1, k = 2:3, try_cutree_hclust = TRUE) # slower than hclust, but works...
Bk_plot(hc1, dend1, k = 2:3) # slower than hclust, but works...

# for some reason it can't turn dend2 back to hclust :
# a= Bk_plot(hc1, hc2, k = 2:3, try_cutree_hclust = TRUE) # slower than hclust, but works...

hc1_mixed <- as.hclust(sample(as.dendrogram(hc1)))
Bk_plot(tree1=hc1, tree2=hc1_mixed,
       add.E=FALSE,
       rejection_line_permutation=TRUE, k_permutation = c(2,4,6,8,10,20,30,40,50), R = 100)
```


## branches_attr_by_clusters

### Change col/lwd/lty of branches based on clusters

#### Description

The user supplies a dend, a vector of clusters, and what to modify (and how).

And the function returns a dendrogram with branches col/lwd/lty accordingly. (the function assumes unique labels)

#### Usage

```r
branches_attr_by_clusters(dend, clusters, values, attr = c("col", "lwd", "lty"), ...)
```

#### Arguments

- **dend**: a dendrogram
- **clusters**: an integer vector of clusters. This HAS to be of the size of the number of leaves. Items that belong to no cluster should get the value 0. The vector should be of the same order as that of the labels in the dendrogram. If you create the clusters from something like `cutree` you would first need to use `order.dendrogram` on it, before using it in the function.
- **values**: the attributes to use for non 0 values. This should be of the same length as the number of non-0 clusters. If it is not, it is recycled.
- **attr**: a character with one of the following values: `col/lwd/lty`
- **...**: ignored.

#### Details

This is probably NOT a very fast implementation of the function, but it works.

This function was designed to enable the manipulation (mainly coloring) of branches, based on the results from the `cutreeDynamic` function.

#### Value

A dendrogram with modified branches (col/lwd/lty).

#### See Also

- `branches_attr_by_labels`
- `get_leaves_attr`
- `nnodes`
- `nleaves`
- `cutreeDynamic`
- `plotDendroAndColors`
Examples

```r
## Not run:

### Getting the hc object
iris_dist <- iris[, -5] %>% dist
hc <- iris_dist %>% hclust
# This is how it looks without any colors:
dend <- as.dendrogram(hc)
plot(dend)

# Both functions give the same outcome
# options 1:
dend %>% set("branches_k_color", k = 4) %>% plot
# options 2:
clusters <- cutree(dend, 4)[order.dendrogram(dend)]
dend %>% branches_attr_by_clusters(clusters) %>% plot

# and the second option is much slower:
system.time(set(dend, "branches_k_color", k = 4)) # 0.26 sec
system.time(branches_attr_by_clusters(dend, clusters)) # 1.61 sec
# BUT, it also allows us to do more flexible things!

#------------------------
# Plotting dynamicTreeCut
#------------------------

# let's get the clusters
library(dynamicTreeCut)
clusters <- cutreeDynamic(hc, distM = as.matrix(iris_dist))
# we need to sort them to the order of the dendrogram:
clusters <- clusters[order.dendrogram(dend)]

# get some functions:
library(dendextendRcpp)
library(colorspace)
no0_unique <- function(x) {
  u_x <- unique(x)
  u_x[u_x != 0]
}

clusters_numbers <- no0_unique(clusters)
n_clusters <- length(clusters_numbers)
cols <- rainbow_hcl(n_clusters)
dend2 <- branches_attr_by_clusters(dend, clusters, values = cols)
# dend2 <- branches_attr_by_clusters(dend, clusters)
plot(dend2)
# add colored bars:
ord_cols <- rainbow_hcl(n_clusters)[order(clusters_numbers)]
tmp_cols <- rep(1, length(clusters))
tmp_cols[clusters != 0] <- ord_cols[clusters != 0][clusters]
colored_bars(tmp_cols, y_shift = -1.1, rowLabels = "")
# all of the ordering is to handle the fact that the cluster numbers are not ascending...
```
branches_attr_by_labels

Description

The user supplies a dend, labels, and type of condition (all/any), and TF_values And the function returns a dendrogram with branches col/lwd/lty accordingly

Usage

branches_attr_by_labels(dend, labels, TF_values = c(2, Inf), attr = c("col", "lwd", "lty"), type = c("all", "any"), ...)

Arguments

dend a dendrogram dend
labels a character vector of labels from the tree
TF_values a two dimensional vector with the TF_values to use in case a branch fulfills the condition (TRUE) and in the case that it does not (FALSE). Defaults are 2/Inf for col, lwd and lty. (so it will insert the first value, and will not change all the FALSE cases)
attr a character with one of the following values: col/lwd/lty
type a character vector of either "all" or "any", indicating which of the branches should be painted: ones that all of their labels belong to the supplied labels, or also ones that even some of their labels are included in the labels vector.
... ignored.

Value

A dendrogram with modified branches (col/lwd/lty).
See Also

`noded_with_condition`, `get_leaves_attr`, `nnodes`, `nleaves`

Examples

```r
## Not run:

library(dendextend)

set.seed(23235)
ss <- sample(1:150, 10 )

# Getting the dend dend
dend <- iris[ss,-5] %>% dist %>% hclust %>% as.dendrogram
  dend %>% plot

dend %>%
  branches_attr_by_labels(c("123", "126", "23", "29")) %>%
  plot

dend %>%
  branches_attr_by_labels(c("123", "126", "23", "29"), "all") %>%
  plot # the same as above

dend %>%
  branches_attr_by_labels(c("123", "126", "23", "29"), "any") %>%
  plot

dend %>%
  branches_attr_by_labels(c("123", "126", "23", "29"),
  "any", "col", c("blue", "red")) %>%
  plot

dend %>%
  branches_attr_by_labels(c("123", "126", "23", "29"),
  "any", "lwd", c(4,1)) %>%
  plot

dend %>%
  branches_attr_by_labels(c("123", "126", "23", "29"),
  "any", "lty", c(2,1)) %>%
  plot

## End(Not run)
```

---

`click_rotate`  
*Interactively rotate a tree object*

Description

Lets the user click a plot of dendrogram and rotates the tree based on the location of the click.
Usage

```r
click_rotate(x, ...)  

## S3 method for class 'dendrogram'
click_rotate(x, plot = TRUE, plot_after = plot, horiz = FALSE, continue = FALSE, ...)
```

Arguments

- `x` a tree object (either a dendrogram or hclust)
- `...` parameters passed to the plot
- `plot` (logical) should the dendrogram first be plotted.
- `plot_after` (logical) should the dendrogram be plotted after the rotation?
- `horiz` logical. Should the plot be normal or horizontal?
- `continue` logical. If TRUE, allows the user to keep clicking the plot until a click is made on the labels.

Value

A rotated tree object

Author(s)

Andrej-Nikolai Spiess, Tal Galili

See Also

`rotate.dendrogram`

Examples

```r
# create the dend:
dend <- USArrests %>% dist %>% hclust("ave") %>%  
as.dendrogram %>% color_labels

## Not run:
# play with the rotation once
dend <- click_rotate(dend)
dend <- click_rotate(dend, horiz = TRUE)
# keep playing with the rotation:
while(TRUE) dend <- click_rotate(dend)
# the same as
dend <- click_rotate(dend, continue = TRUE)

## End(Not run)
```
collapse_branch

Collapse branches under a tolerance level

Description

Collapse branches under a tolerance level

Usage

collapse_branch(dend, tol = 1e-08, lower = TRUE, ...)

Arguments

dend dendrogram object
tol a numeric value giving the tolerance to consider a branch length significantly
greater than zero
lower logical (TRUE). collapse branches which are lower than tol?
... passed on (not used)

Value

A dendrogram with both of the root’s branches of the same height

See Also

multi2di

Examples

# # ladderize is like sort(..., type = "node")
dend <- iris[1:5,-5] %>% dist %>% hclust %>% as.dendrogram
par(mfrow = c(1,3))
dend %>% ladderize %>% plot(horiz = TRUE); abline(v = .2, col = 2, lty = 2)
dend %>% collapse_branch(tol = 0.2) %>% ladderize %>% plot(horiz = TRUE)
dend %>% collapse_branch(tol = 0.2) %>% ladderize %>%
    hang.dendrogram(hang = 0) %>% plot(horiz = TRUE)

par(mfrow = c(1,2))
dend %>% collapse_branch(tol = 0.2, lower = FALSE) %>% plot(horiz = TRUE, main = "dendrogram")
library(ape)
dend %>% as.phylo %>% di2multi(tol = 0.2) %>% plot(main = "phylo")
colored_bars

Add colored bars to a dendrogram

Description

Add colored bars to a dendrogram, usually corresponding to either clusters or some outside categorization.

Usage

colored_bars(colors, dend, rowLabels = NULL, cex.rowLabels = 0.9,
    add = TRUE, y_scale, y_shift, text_shift = 1,
sort_by_labels_order = FALSE, ...)

Arguments

colors  Coloring of objects on the dendrogram. Either a vector (one color per object) or a matrix (can also be an array or a data frame) with each column giving one color per object. Each column will be plotted as a horizontal row of colors under the dendrogram.
dend  a dendrogram object. If missing, the colors are plotted without and re-ordering (this assumes that the colors are already ordered based on the dend’s labels) This is also important in order to get the correct height/location of the colored bars (i.e.: adjusting the y_scale and y_shift)
rowLabels  Labels for the colorings given in colors. The labels will be printed to the left of the color rows in the plot. If the argument is given, it must be a vector of length equal to the number of columns in colors. If not given, names(colors) will be used if available. If not, sequential numbers starting from 1 will be used.
cex.rowLabels  Font size scale factor for the row labels. See par.
add  logical(TRUE), should the colored bars be added to an existing dendrogram plot?
y_scale  how much should the bars be stretched on the y axis? If no dend is supplied - the default will be 1
y_shift  where should the bars be plotted underneath the x axis? By default it will try to locate the bars underneath the labels (it may miss, in which case you would need to enter a number manually) If no dend is supplied - the default will be 0
text_shift  a dendrogram object
sort_by_labels_order  logical(FALSE) - if TRUE, then the order of the colored bars will be sorted based on the order needed to change the original order of the observations to the current order of the labels in the dendrogram. If FALSE (default) the colored bars are plotted as-is, based on the order of the colors vector.
...

... ignored at this point.
Details

You will often need to adjust the y_scale, y_shift and the text_shift parameters, in order to get the
bars in the location you would want.

(this can probably be done automatically, but will require more work. since it has to do with the
current mar settings, the number of groups, and each computer’s specific graphic device. patches
for smarter defaults will be appreciated)

Value

An invisible vector/matrix with the ordered colors.

Author(s)

Steve Horvath <SHorvath@mednet.ucla.edu>, Peter Langfelder <Peter.Langfelder@gmail.com>,
Tal Galili <Tal.Galili@gmail.com>

Source

This function is based on the plotHclustColors from the moduleColor R package. It was modified so
that it would work with dendrograms (and not just hclust objects), as well allow to add the colored
bars on top of an existing plot (and not only as a separate plot).

See: http://cran.r-project.org/web/packages/moduleColor/ For more details.

See Also

branches_attr_by_clusters, plotDendroAndColors

Examples

```r
## Not run:
rows_picking <- c(1:5, 25:30)
dend <- (iris[rows_picking,-5]*10) >% dist >% hclust >% as.dendrogram
odd_numbers <- rows_picking %% 2
cols <- c("gold", "grey")[odd_numbers+1]
# scale is off
plot(dend)
colored_bars(dend, cols)
# move and scale a bit
plot(dend)
colored_bars(dend, cols, y_shift = -1,
  rowLabels = "Odd\n numbers")
# Now let's cut the tree and add that info to the plot:
k2 <- cutree(dend, k = 2)
cols2 <- c("#0082CE", "#CC476B")[k2]
plot(dend)
# sadly, the shift parameters need to be handled manually...
colored_bars(dend, cbind(cols2, cols), y_shift = -1,
  rowLabels = c("2 clusters", "Odd numbers"),
  text_shift = 1)
```
# let's add clusters color
# notice how we need to play with the colors a bit
# this is because color_branches places colors from
# left to right. Which means we need to give colored_bars
# the colors of the items so that after sorting they would be
# from left to right. Here is how it can be done:
the_k <- 3
library(colorspace)
cols3 <- rainbow_hcl(the_k, c=90, l=50)
dend %>% update("branches_k_color", k = the_k, with = cols3) %>% plot

kx <- cutree(dend, k = the_k)
ord <- order.dendrogram(dend)
kx <- sort_levels_values(kx[ord])
kx <- kx[match(seq_along(ord), ord)]
colored_bars(dend, cbind(cols3[kx], cols2, cols),
    y_shift = -1, y_scale = 1.4,
    rowLabels = c("3 clusters", "2 clusters", "Odd numbers"),
    text_shift = 1)

## End(Not run)

---

color_branches  

*Color tree’s branches according to sub-clusters*

description

This function is for dendrogram and hclust objects. This function colors both the terminal leaves of a tree’s cluster and the edges leading to those leaves. The edgePar attribute of nodes will be augmented by a new list item col. The groups will be defined by a call to `cutree` using the k or h parameters.

If col is a color vector with a different length than the number of clusters (k) - then a recycled color vector will be used.

Usage

color_branches(tree, k = NULL, h = NULL, col, groupLabels = NULL, warn = dendextend_options("warn"), ...)

Arguments

tree A dendrogram or hclust tree object
k number of groups (passed to `cutree`
height at which to cut tree (passed to cutree)

Function or vector of Colors. By default it tries to use rainbow_hcl from the colorspace package. (with parameters c=90 and l=50). If colorspace is not available, it will fall back on the rainbow function.

If TRUE add numeric group label - see Details for options

logical (default from dendextend_options("warn") is FALSE). Set if warning are to be issued, it is safer to keep this at TRUE, but for keeping the noise down, the default is FALSE.

... ignored.

Details

If groupLabels=TRUE then numeric group labels will be added to each cluster. If a vector is supplied then these entries will be used as the group labels. If a function is supplied then it will be passed a numeric vector of groups (e.g. 1:5) and must return the formatted group labels.

If the labels of the dendrogram are NOT character (but, for example integers) - they are coerced into character. This step is essential for the proper operation of the function. A dendrogram labels might happen to be integers if they are based on an hclust performed on a dist of an object without rownames.

Value

a tree object of class dendrogram.

Author(s)

Tal Galili, extensively based on code by Gregory Jefferis

Source

This function is a derived work from the color_clusters function, with some ideas from the slice function - both are from the dendroextras package by jefferis.

It extends it by using cutree.dendrogram - allowing the function to work for trees that hclust can not handle (unbranched and non-ultrametric trees). Also, it allows REPEATED cluster color assignments to branches on to the same tree. Something which the original function was not able to handle.

See Also

cutree,dendrogram, hclust, labels_colors

Examples

```r
# Not run:
dend <- USArrests %>% dist %>% hclust(method = "ave") %>% as.dendrogram
d1=color_branches(dend,5, col = c(3,1,1,4,1))
plot(d1) # selective coloring of branches :) d2=color_branches(d1,5)
```
plot(d2)

d2=color_branches(d1, 5, col = c(3,1,1,4,1), groupLabels=TRUE)
plot(d2)  # selective coloring of branches :)
d2=color_branches(d1, 5, groupLabels=TRUE)
plot(d2)

d5=color_branches(dend, 5)
plot(d5)
d5g=color_branches(dend, 5, groupLabels=TRUE)
plot(d5g)
d5gr=color_branches(dend, 5, groupLabels=as.roman)
plot(d5gr)

# messy - but interesting:
dend_override=color_branches(dend, 2, groupLabels=as.roman)
dend_override=color_branches(dend_override, 4, groupLabels=as.roman)
dend_override=color_branches(dend_override, 7, groupLabels=as.roman)
plot(dend_override)
d5=color_branches(tree=dend[[1]], k=5)

library(dendextend)
data(iris, envir = environment())
d_iris <- dist(iris[, -5])
hc_iris <- hclust(d_iris)
dend_iris <- as.dendrogram(hc_iris)
dend_iris=color_branches(dend_iris, k=3)

library(colorspace)
labels_colors(dend_iris) <-
  rainbow_hcl(3)[sort_levels_values(
    as.numeric(iris[, 5])[order.dendrogram(dend_iris)]
  )]

plot(dend_iris,
main = "Clustered Iris dataset",
sub = "labels are colored based on the true cluster")

# cutree(dend_iris, k=3, order_clusters_as_data=FALSE,
# sort_cluster_numbers = TRUE, try_cutree_hclust=FALSE)
# cutree(dend_iris, k=3, order_clusters_as_data=FALSE,
# sort_cluster_numbers = TRUE, try_cutree_hclust=TRUE)

library(colorspace)
data(iris, envir = environment())
d_iris <- dist(iris[, -5])
hc_iris <- hclust(d_iris)
labels(hc_iris)  # no labels, because "iris" has no row names
dend_iris <- as.dendrogram(hc_iris)
is.integer(labels(dend_iris)) # this could cause problems...

iris_species <- rev(levels(iris[,5]))
dend_iris <- color_branches(dend_iris,k=3, groupLabels=iris_species)
is.character(labels(dend_iris)) # labels are no longer "integer"

# have the labels match the real classification of the flowers:
labels_colors(dend_iris) <-
  rainbow_hcl()[sort_levels_values(
    as.numeric(iris[,5])[order.dendrogram(dend_iris)]
  )]

# We'll add the flower type
labels(dend_iris) <- paste(as.character(iris[,5])[order.dendrogram(dend_iris)],
  "(" , labels(dend_iris), ")",
  sep = "\"")

dend_iris <- hang.dendrogram(dend_iris,hang_height=0.1)

# reduce the size of the labels:
dend_iris <- assign_values_to_leaves_nodePar(dend_iris, 0.5, "lab.cex")

par(mar = c(3,3,3,7))
plot(dend_iris,
  main = "Clustered Iris dataset",
  horiz = TRUE, nodePar = list(cex = .007))
legend("topleft", legend = iris_species, fill = rainbow_hcl(3))
a= dend_iris[[1]]
dend_iris1 <- color_branches(a,k = 3)
plot(dend_iris1)

# str(dendrapply(d2, unclass))
# unclass(d1)

c(1:5) %>% # take some data
dist %>% # calculate a distance matrix,
hclust(method = "single") %>% # on it compute hierarchical clustering using the "average" method,
as.dendrogram %>% color_branches(k=3) %>% plot # nice, returns the tree as is...

## End(Not run)

color_labels

Color tree’s labels according to sub-clusters

Description

This function is for dendrogram and hclust objects. This function colors tree’s labels.
The groups will be defined by a call to `cutree` using the k or h parameters.
If col is a color vector with a different length than the number of clusters (k) - then a recycled color vector will be used.

**Usage**

```r
color_labels(tree, k = NULL, h = NULL, labels, col,
             warn = dendextend_options("warn"), ...)
```

**Arguments**

- `tree` A dendrogram or hclust tree object
- `k` number of groups (passed to `cutree`)
- `h` height at which to cut tree (passed to `cutree`)
- `labels` character vector. If not missing, it overrides k and h, and simply colors these labels in the tree based on "col" parameter.
- `col` Function or vector of Colors. By default it tries to use rainbow_hcl from the colorspace package. (with parameters c=90 and l=50). If colorspace is not available, It will fall back on the rainbow function.
- `warn` logical (default from dendextend_options("warn") is FALSE). Set if warning are to be issued, it is safer to keep this at TRUE, but for keeping the noise down, the default is FALSE. (in case h/k/labels are not supplied, or if col is too short)
- `...` ignored.

**Value**

a tree object of class dendrogram.

**Source**

This function is in the style of `color_branches`, and based on `labels_colors`.

**See Also**

`cutree`, `dendrogram`, `hclust`, `labels_colors`, `color_branches`

**Examples**

```r
# Not run:
hc <- hclust(dist(USArrests), "ave")
dend <- as.dendrogram(hc)
dend=color_labels(dend,5, col = c(3,1,1,4,1))
dend=color_branches(dend,5, col = c(3,1,1,4,1))
plot(dend) # selective coloring of branches AND labels :

# coloring some labels, based on label names:
dend=color_labels(dend,col = "red", labels = labels(dend)[c(4,16)])
plot(dend) # selective coloring of branches AND labels :
```
cor.dendlist

Correlation matrix between a list of trees.

Description

A correlation matrix between a list of trees.

Assumes the labels in the two trees fully match. If they do not please first use intersect_trees to have them matched.

Usage

cor.dendlist(x, method = c("cophenetic", "baker"), ...)

Arguments

- **x**: a dendlist of trees
- **method**: a character string indicating which correlation coefficient is to be computed. One of "cophenetic" (default), or "baker", can be abbreviated.
- **...**: Ignored.

Value

A correlation matrix between the different trees

See Also

cophenetic, cor_cophenetic, cor_bakers_gamma

Examples

```r
## Not run:

set.seed(23235)
s <- sample(1:150, 10)
dend1 <- iris[s, -5] %>% dist %>% hclust("com") %>% as.dendrogram
dend2 <- iris[s, -5] %>% dist %>% hclust("single") %>% as.dendrogram
dend3 <- iris[s, -5] %>% dist %>% hclust("ave") %>% as.dendrogram
dend4 <- iris[s, -5] %>% dist %>% hclust("centroid") %>% as.dendrogram
```
**cor_bakers_gamma**

Baker’s Gamma correlation coefficient

**Description**

Calculate Baker’s Gamma correlation coefficient for two trees.

Assumes the labels in the two trees fully match. If they do not please first use `intersect_trees` to have them matched.

**WARNING:** this can be quite slow for medium/large trees.

**Usage**

```r
cor_bakers_gamma(tree1, ...)  
```

```r  
## S3 method for class 'dendrogram'
cor_bakers_gamma(tree1, tree2, use_labels_not_values = TRUE, to_plot = FALSE, warn = dendextend_options("warn"), ...)  
## S3 method for class 'hclust'

cor_bakers_gamma(tree1, tree2, use_labels_not_values = TRUE, to_plot = FALSE, warn = dendextend_options("warn"), ...)  
## S3 method for class 'dendlist'

cor_bakers_gamma(tree1, which = c(1L, 2L), ...)  
```

**Arguments**

- `tree1`: a tree (dendrogram/hclust/phylo)
- `tree2`: a tree (dendrogram/hclust/phylo)
- `use_labels_not_values`: logical (TRUE). Should labels be used in the k matrix when using cutree? Set to FALSE will make the function a bit faster BUT, it assumes the two trees have the exact same leaves order values for each labels. This can be assured by using `match_order_by_labels`.
- `to_plot`: logical (FALSE). Passed to `bakers_gamma_for_2_k_matrix`
- `warn`: logical (default from dendextend_options("warn") is FALSE). Set if warning are to be issued, it is safer to keep this at TRUE, but for keeping the noise down, the default is FALSE. should a warning be issued when using `cutree`?
- `which`: an integer vector of length 2, indicating which of the trees in the dendlist object should be plotted (relevant for dendlist)
Details

Baker’s Gamma (see reference) is a measure of association (similarity) between two trees of hierarchical clustering (dendrograms).

It is calculated by taking two items, and see what is the highest possible level of k (number of cluster groups created when cutting the tree) for which the two item still belongs to the same tree. That k is returned, and the same is done for these two items for the second tree. There are n over 2 combinations of such pairs of items from the items in the tree, and all of these numbers are calculated for each of the two trees. Then, these two sets of numbers (a set for the items in each tree) are paired according to the pairs of items compared, and a Spearman correlation is calculated.

The value can range between -1 to 1. With near 0 values meaning that the two trees are not statistically similar. For exact p-value one should result to a permutation test. One such option will be to permute over the labels of one tree many times, and calculating the distribution under the null hypothesis (keeping the trees topologies constant).

Notice that this measure is not affected by the height of a branch but only of its relative position compared with other branches.

Value

Baker’s Gamma association Index between two trees (a number between -1 to 1)

References


See Also

cor_cophenetic

Examples

```R
## Not run:

set.seed(23235)
ss <- sample(1:150, 10)
hc1 <- hclust(dist(iris[ss,-5]), "com")
hc2 <- hclust(dist(iris[ss,-5]), "single")
dend1 <- as.dendrogram(hc1)
dend2 <- as.dendrogram(hc2)
# cutree(dend1)
cor_bakers_gamma(hc1, hc2)
cor_bakers_gamma(dend1, dend2)
dend1 <- match.order_by_labels(dend1, dend2) # if you are not sure
cor_bakers_gamma(dend1, dend2, use_labels_not_values = FALSE)

library(microbenchmark)
microbenchmark(
```
with_labels = cor_bakers_gamma(dend1, dend2, try_cutree_hclust=FALSE) ,
with_values = cor_bakers_gamma(dend1, dend2,
    use_labels_not_values = FALSE, try_cutree_hclust=FALSE) ,
times=10
)
cor_bakers_gamma(dend1, dend1, use_labels_not_values = FALSE)
cor_bakers_gamma(dend1, dend1, use_labels_not_values = TRUE)

## End(Not run)

cor_cophenetic = Cophenetic correlation between two trees

Description

Cophenetic correlation coefficient for two trees.

Assumes the labels in the two trees fully match. If they do not please first use intersect_trees to have them matched.

Usage

cor_cophenetic(tree1, ...)  

## Default S3 method:
cor_cophenetic(tree1, tree2, method = c("pearson", "kendall", "spearman"), ...)

## S3 method for class 'dendlist'
cor_cophenetic(tree1, which = c(1L,2L), method = c("pearson", "kendall", "spearman"), ...)

Arguments

tree1                   a tree (dendrogram/hclust/phylo, or dendlist)
...                     Ignored.
tree2                   a tree (dendrogram/hclust/phylo)
which                   an integer vector of length 2, indicating which of the trees in a dendlist object should have their cor_cophenetic calculated.
method                  a character string indicating which correlation coefficient is to be computed. One of "pearson" (default), "kendall", or "spearman", can be abbreviated. Passed to cor.
Details

From \texttt{cophenetic}: The cophenetic distance between two observations that have been clustered is defined to be the intergroup dissimilarity at which the two observations are first combined into a single cluster. Note that this distance has many ties and restrictions.

cor\_cophenetic calculates the correlation between two cophenetic distance matrices of the two trees. The value can range between -1 to 1. With near 0 values meaning that the two trees are not statistically similar. For exact p-value one should result to a permutation test. One such option will be to permute over the labels of one tree many times, and calculating the distribution under the null hypothesis (keeping the trees topologies constant).

Notice that this measure IS affected by the height of a branch.

Value

The correlation between cophenetic

References


http://en.wikipedia.org/wiki/Cophenetic_correlation

See Also

cophenetic, cor\_bakers\_gamma

Examples

## Not run:

```r
set.seed(23235)
ss <- sample(1:150, 10)
hc1 <- iris[ss,-5] %>%, dist %>%, hclust("com")
hc2 <- iris[ss,-5] %>%, dist %>%, hclust("single")
dend1 <- as.dendrogram(hc1)
dend2 <- as.dendrogram(hc2)
# cutree(dend1)
cophenetic(hc1)
cophenetic(hc2)
# notice how the dist matrix for the dendrograms have different orders:
cophenetic(dend1)
cophenetic(dend2)

cor(cophenetic(hc1), cophenetic(hc2)) # 0.874
cor(cophenetic(dend1), cophenetic(dend2)) # 0.16
# the difference is because the order of the distance table in the case of
# stats::cophenetic.dendrogram will change between dendrograms!
```
# however, this is consistant (since I force-sort the rows/columns):
cor_cophenetic(hc1, hc2)
cor_cophenetic(dend1, dend2)

cor_cophenetic(dendlist(dend1, dend2))

# we can also use different cor methods (almost the same result though):
cor_cophenetic(hc1, hc2, method = "spearman") # 0.8456014
cor_cophenetic(dend1, dend2, method = "spearman") #

# cophenetic correlation is about 10 times (!) faster than bakers_gamma cor:
library(microbenchmark)
microbenchmark(
    cor_bakers_gamma = cor_bakers_gamma(dend1, dend2, try_cutree_hclust=FALSE),
    cor_cophenetic = cor_cophenetic(dend1, dend2) ,
    times=10
)

# but only because of the cutree for dendrogram. When allowing hclust cutree
# it is only about twice as fast:
microbenchmark(
    cor_bakers_gamma = cor_bakers_gamma(dend1, dend2, try_cutree_hclust=TRUE),
    cor_cophenetic = cor_cophenetic(dend1, dend2) ,
    times=10
)

## End(Not run)

---

**countTerminalNodes**

Counts the number of terminal nodes (merging 0 nodes!)

### Description

This function counts the number of "practical" terminal nodes (nodes which are not leaves, but has 0 height to them are considered "terminal" nodes). If the tree is standard, that would simply be the number of leaves (only the leaves will have height 0). However, in cases where the tree has several nodes (before the leaves) with 0 height, the countTerminalNodes counts such nodes as terminal nodes.

The function is recursive in that it either returns 1 if it reached a terminal node (either a leaf or a 0 height node), else: it will count the number of terminal nodes in each of its sub-nodes, sum them up, and return them.

### Usage

```r
countTerminalNodes(dend_node, ...)
```
Arguments

dend_node   a dendrogram object for which to count its number of terminal nodes (leaves or 0 height nodes).

Value

The number of terminal nodes (excluding the leaves of nodes of height 0)

Examples

# define dendrogram object to play with:
hc <- hclust(dist(USArrests[1:3,]), "ave")
dend <- as.dendrogram(hc)

###
# Trivial case
count_terminal_nodes(dend) # 3 terminal nodes
length(labels(dend)) # 3 - the same number
plot(dend,
main = "This is considered a tree \n with THREE terminal nodes (leaves)"
)

###
# NON-Trivial case
str(dend)
attr(dend[[2]], "height") <- 0
count_terminal_nodes(dend) # 2 terminal nodes, why? see this plot:
plot(dend,
main = "This is considered a tree \n with TWO terminal nodes only"
)
# while we have 3 leaves, in practice we have only 2 terminal nodes
# (this is a feature, not a bug.)

---

**Cutree**: Cut a Tree (Dendrogram/hclust/phylo) into Groups of Data

Description

Cuts a dendrogram tree into several groups by specifying the desired number of clusters k(s), or cut height(s).

For hclust.dendrogram - In case there exists no such k for which exists a relevant split of the dendrogram, a warning is issued to the user, and NA is returned.

Usage

cutree(tree, k = NULL, h = NULL,...)

## S3 method for class 'hclust'
cutree(tree, k = NULL, h = NULL,
use_labels_not_values = TRUE,
order_clusters_as_data = TRUE,
sort_cluster_numbers = TRUE,
warn = dendextend_options("warn"),
NA_to_0L = TRUE,
...

## S3 method for class 'phylo'
cutree(tree, k = NULL, h = NULL,...)

## S3 method for class 'dendrogram'
cutree(tree, k = NULL, h = NULL,

dend_heights_per_k = NULL,
use_labels_not_values = TRUE,
order_clusters_as_data = TRUE,
sort_cluster_numbers = TRUE,
warn = dendextend_options("warn"),
try_cutree_hclust = TRUE,
NA_to_0L = TRUE,
...

Arguments

- **tree** a dendrogram object
- **k** numeric scalar (OR a vector) with the number of clusters the tree should be cut into.
- **h** numeric scalar (OR a vector) with a height where the tree should be cut.
- **...** (not currently in use)
- **dend_heights_per_k** a named vector that resulted from running. heights_per_k.dendrogram. When running the function many times, supplying this object will help improve the running time if using k!=NULL.
- **use_labels_not_values** logical, defaults to TRUE. If the actual labels of the clusters do not matter - and we want to gain speed (say, 10 times faster) - then use FALSE (gives the "leaves order" instead of their labels.). This is passed to cutree_1h.dendrogram.
- **order_clusters_as_data** logical, defaults to TRUE. There are two ways by which to order the clusters: 1) By the order of the original data. 2) by the order of the labels in the dendrogram. In order to be consistent with cutree, this is set to TRUE. This is passed to cutree_1h.dendrogram.
- **sort_cluster_numbers** logical (TRUE). Should the resulting cluster id numbers be sorted? (default is TRUE in order to make the function compatible with cutree ) from stats, but it allows for sensible color order when using color_branches.
- **warn** logical (default from dendextend_options("warn") is FALSE). Set if warning are to be issued, it is safer to keep this at TRUE, but for keeping the noise down, the
default is FALSE. Should the function send a warning in case the desired k is not available?

**try_cutree_hclust**

logical. default is TRUE. Since cutree for hclust is MUCH faster than for dendrogram - cutree.dendrogram will first try to change the dendrogram into an hclust object. If it will fail (for example, with unbranched trees), it will continue using the cutree.dendrogram function. If try_cutree_hclust=FALSE, it will force to use cutree.dendrogram and not cutree.hclust.

**NA_to_0L**

logical. default is TRUE. When no clusters are possible, Should the function return 0 (TRUE, default), or NA (when set to FALSE).

**Details**

At least one of k or h must be specified, k overrides h if both are given.

as opposed to cutree for hclust, cutree.dendrogram allows the cutting of trees at a given height also for non-ultrametric trees (ultrametric tree == a tree with monotone clustering heights). This sort_cluster_numbers parameter is based on the sort_levels_values function. Using that parameter with TRUE makes the clusters id’s from cutree to be ordered from left to right. e.g: the left most cluster in the tree will be numbered "1", the one after it will be "2" etc...).

**Value**

If k or h are scalar - cutree.dendrogram returns an integer vector with group memberships. Otherwise a matrix with group memberships is returned where each column corresponds to the elements of k or h, respectively (which are also used as column names).

In case there exists no such k for which exists a relevant split of the dendrogram, a warning is issued to the user, and NA is returned.

**Author(s)**

cutree.dendrogram was written by Tal Galili. cutree.hclust is redirecting the function to cutree from base R.

**See Also**

hclust, cutree, cutree_1h.dendrogram, cutree_1k.dendrogram,

**Examples**

```r
## Not run:
hc <- hclust(dist(USArrests[c(1,6,13,20, 23),]), "ave")
dend <- as.dendrogram(hc)
unbranch_dend <- unbranch(dend,2)

cutree(hc, k=2:4) # on hclust
cutree(dend, k=2:4) # on dendrogram

cutree(hc, k=2) # on hclust
cutree(dend, k=2) # on dendrogram
```
cutree
cutree(dend, h = c(20, 25.5, 50, 170))
cutree(hc, h = c(20, 25.5, 50, 170))

# the default (ordered by original data's order)
cutree(dend, k=2:3, order_clusters_as_data = FALSE)
labels(dend)

# as.hclust(unbranch_dend) # ERROR - can not do this...
cutree(unbranch_dend, k = 2) # all NA's
cutree(unbranch_dend, k = 1:4)
cutree(unbranch_dend, h = c(20, 25.5, 50, 170))
cutree(dend, h = c(20, 25.5, 50, 170))

library(microbenchmark)
## this shows how as.hclust is expensive - but still worth it if possible
microbenchmark(
  cutree(hc, k=2:4),
  cutree(as.hclust(dend), k=2:4),
  cutree(dend, k=2:4),
  cutree(dend, k=2:4, try_cutree_hclust=FALSE)
)

## the dendrogram is MUCH slower...

## Unit: microseconds
## expr      min       lq     median       uq      max neval
## cutree(hc, k = 2:4)  91.270  96.589  99.3885 107.5075 338.758  100
## tree(as.hclust(dend),
## k = 2:4)    1701.629 1767.700 1854.4895 2029.1875 8736.59  100
## cutree(dend, k = 2:4) 1887.456 1869.887 1963.3960 2125.2155 5579.705  100
## cutree(dend, k = 2:4, try_cutree_hclust = FALSE) 8393.914  8570.852  8755.3490 9686.7930 14194.790  100

## and trying to "hclust" is not expensive (which is nice...)
microbenchmark(
  cutree_unbranch_dend = cutree(unbranch_dend, k=2:4),
  cutree_unbranch_dend_not_trying_to_hclust =
  cutree(unbranch_dend, k=2:4, try_cutree_hclust=FALSE)
)

## Unit: milliseconds
## expr      min       lq     median       uq      max neval
## cutree_unbranch_dend 7.309329 7.428314 7.494107 7.752234 17.59581  100
## cutree_unbranch_dend_not_trying_to_hclust
## _trying_to_hclust   6.945375 7.079198 7.148629 7.577536 16.99780  100
## There were 50 or more warnings (use warnings() to see the first 50)

## notice that if cutree can't find clusters for the desired k/h, it will produce 0's instead!
## (It will produce a warning though...)
## This is a different behaviour than stats::cutree
## For example:
cutree(as.dendrogram(hclust(dist(c(1,1,1,2,2))))),
Description

Cuts a dendrogram tree into several groups by specifying the desired cut height (only a single height!).

Usage

```r
cutree_1h.dendrogram(tree, h, order_clusters_as_data = TRUE,
                     use_labels_not_values = TRUE, warn = dendextend_options("warn"), ...)
```

Arguments

- **tree**: a dendrogram object
- **h**: numeric scalar (NOT a vector) with a height where the tree should be cut.
- **order_clusters_as_data**: logical, defaults to TRUE. There are two ways by which to order the clusters: 1) By the order of the original data. 2) by the order of the labels in the dendrogram. In order to be consistent with `cutree`, this is set to TRUE.
- **use_labels_not_values**: logical, defaults to TRUE. If the actual labels of the clusters do not matter - and we want to gain speed (say, 10 times faster) - then use FALSE (gives the "leaves order" instead of their labels.).
- **warn**: logical (default from `dendextend_options("warn")`) is FALSE. Set if warning are to be issued, it is safer to keep this at TRUE, but for keeping the noise down, the default is FALSE.
- **...**: (not currently in use)

Value

cutree_1h.dendrogram returns an integer vector with group memberships

Author(s)

Tal Galili

See Also

`hclust, cutree`
Examples

```r
hc <- hclust(dist(USArrests[c(1,6,13,20,23),]), "ave")
dend <- as.dendrogram(hc)
cutree(hc, h=50) # on hclust
cutree_1h.dendrogram(dend, h=50) # on a dendrogram

labels(dend)

# the default (ordered by original data's order)
cutree_1h.dendrogram(dend, h=50, order_clusters_as_data = TRUE)

# A different order of labels - order by their order in the tree
cutree_1h.dendrogram(dend, h=50, order_clusters_as_data = FALSE)

# make it faster
## Not run:
library(microbenchmark)
microbenchmark(
  cutree_1h.dendrogram(dend, h=50),
  cutree_1h.dendrogram(dend, h=50,use_labels_not_values = FALSE)
)
# 0.8 vs 0.6 sec - for 100 runs

## End(Not run)
```

Description

Cuts a dendrogram tree into several groups by specifying the desired number of clusters k (only a single k value!).

In case there exists no such k for which exists a relevant split of the dendrogram, a warning is issued to the user, and NA is returned.

Usage

```r
cutree_1k.dendrogram(tree, k, dend_heights_per_k = NULL,
                     use_labels_not_values = TRUE, order_clusters_as_data = TRUE,
                     warn = dendextend_options("warn"), ...)
```

Arguments

tree          a dendrogram object
k             numeric scalar (not a vector!) with the number of clusters the tree should be cut into.
dend_heights_per_k

a named vector that resulted from running `heights_per_kNdendrogram`. When running the function many times, supplying this object will help improve the running time.

use_labels_not_values

logical, defaults to TRUE. If the actual labels of the clusters do not matter - and we want to gain speed (say, 10 times faster) - then use FALSE (gives the "leaves order" instead of their labels.). This is passed to `cutree_1h.dendrogram`.

order_clusters_as_data

logical, defaults to TRUE. There are two ways by which to order the clusters: 1) By the order of the original data. 2) by the order of the labels in the dendrogram. In order to be consistent with `cutree`, this is set to TRUE. This is passed to `cutree_1h.dendrogram`.

warn

logical (default from `dendextend_options("warn")` is FALSE). Set if warning are to be issued, it is safer to keep this at TRUE, but for keeping the noise down, the default is FALSE. Should the function send a warning in case the desired k is not available?

... (not currently in use)

Value

cutree_1k.dendrogram returns an integer vector with group memberships.

In case there exists no such k for which exists a relevant split of the dendrogram, a warning is issued to the user, and NA is returned.

Author(s)

Tal Galili

See Also

`hclust, cutree, cutree_1h.dendrogram`

Examples

```r
hc <- hclust(dist(USArrests[c(1,6,13,20, 23),]), "ave")
dend <- as.dendrogram(hc)
cutree(hc, k=3) # on hclust
cutree_1k.dendrogram(dend, k=3) # on a dendrogram

labels(dend)

# the default (ordered by original data's order)
cutree_1k.dendrogram(dend, k=3, order_clusters_as_data = TRUE)

# A different order of labels - order by their order in the tree
cutree_1k.dendrogram(dend, k=3, order_clusters_as_data = FALSE)
```
# make it faster
## Not run:
library(microbenchmark)
dend ks <- heights_per_k.dendrogram
microbenchmark(
cutree_1k.dendrogram = cutree_1k.dendrogram(dend, k=4),
cutree_1k.dendrogram_no_labels = cutree_1k.dendrogram(dend,
           k=4, use_labels_not_values = FALSE),
cutree_1k.dendrogram_no_labels_per_k = cutree_1k.dendrogram(dend,
           k=4, use_labels_not_values = FALSE,
           dend_heights_per_k = dend ks)
)

# the last one is the fastest...
## End(Not run)

---

**cut_lower_fun**

*Cut a dendrogram - and run a function on the output*

**Description**

Cuts the a tree at height \( h \) and returns a list with the FUN function implemented on all the sub trees created by cut at height \( h \). This is used for creating a `cutree.dendrogram` function, by using the labels function as FUN.

This is the Rcpp version of the function, offering a 10-60 times improvement in speed (depending on the tree size it is used on).

**Usage**

```r
cut_lower_fun(tree, h, FUN = labels, warn = dendextend_options("warn"), ...)
```

**Arguments**

- **tree**: a dendrogram object.
- **h**: a scalar of height to cut the tree by.
- **FUN**: a function to run. (default is "labels")
- **warn**: logical (default from dendextend_options("warn") is FALSE). Set if warning are to be issued, it is safer to keep this at TRUE, but for keeping the noise down, the default is FALSE. Should the user be warned if reverting to default?

**Value**

A list with the output of running FUN on each of the sub trees derived from cutting "tree"

**Author(s)**

Tal Galili
See Also

labels, dendrogram, cutree.dendrogram

Examples

dend = as.dendrogram(hclust(dist(iris[1:4,-5])))
# this is really cool!
cut_lower_fun(dend, .4, labels)
lapply(cut(dend, h = .4)$lower, labels)
cut_lower_fun(dend, .4, order.dendrogram)

## Not run:
# library(dendextend)
library(dendextendRcpp)
dend_big = as.dendrogram(hclust(dist(iris[1:150,-5])))
library(microbenchmark)
microbenchmark(old_cut_lower_fun(dend_big,.1),
               dendextendRcpp::dendextendRcpp_cut_lower_fun(dend_big,.1),
               times = 100)
# about 7-15 times faster. It is faster the larger the tree is, and the lower h is.

## End(Not run)

---

d3dendrogram  

Plot dendrogram to webpage and view.

Description

Plot dendrogram to webpage and view.

Usage

d3dendrogram(d, height = 500, width = 700, rightmargin = 200,
              open = TRUE, ...)

Arguments

d            a dendrogram object
height        pixels, height of the plot.
width         pixels, width of the plot
rightmargin   pixels to reserve on the right side for leaf labels.
open          open the graphic in a browser? (see details).
...           Graphical options, see d3dendro_options.
Details

If open=TRUE, the graphic is opened using the function defined by 
getOption("viewer"). If no viewer option is specified, 
utils::browseURL is opened. Specifically, in RStudio this means that 
the viewer is used.

Value

If open=TRUE, a character string containing the html is invisibly 
returned. If open=TRUE, a character 
string containing the html is returned.

Tips

Dendrogram plots with many end-nodes tend to get overlapping labels or (end)nodes. If the number 
of endnodes is not too large, there are several ways to overcome this.

• Decrease font size of the nodes (node_font, see d3dendro_options on how to specify)
• Decrease size of node radius (node_radius, idem)
• Increase the height of the plot. Use about (2 times font height) times count_terminal_nodes(d).
• Plot sub-dendrograms in stead of the whole dendrogram. There’s only so much detail a screen 
can display and your eye can observe. Consider plotting subdendrograms or other summaries 
when dendrograms have many, many nodes.

Arguments

... <option>=<value> pairs (see below). If no option/value pair is given, a list 
with current settings is returned. If the single input "reset" is given, options 
are restored to factory settings.

copy only when options are set: set global options or return a modified copy of the 
global option environment invisibly?
**Options**

Option values are exported as-is to the generated javascript/webpage.

- **node_fill (char,"#fff")**  
  Node color (fill)
- **node_stroke (char,"steelblue")**  
  Node color (circumference)
- **node_stroke_width (char,"1.5px")**  
  Width of node circumference
- **node_radius (num;4.5)**  
  Size of node (pixels)
- **node_font (char,"14px sans-serif")**  
  Font used to print labels. Size and type spec
- **link_fill (char,"none")**  
  Link fill color (you probably do not want to set this)
- **link_stroke (char,"#ccc")**  
  Link line color (that's the one you want)
- **link_stroke_width (char,"1.5px")**  
  Link line width (pixels)
- **axis_stroke (char,"black")**  
  Color of axis line
- **axis_width (char,"2px")**  
  Width of axis line
- **axis_ntick (num;5)**  
  Approximate no of tick marks
- **axis_ticklength_in (num;5)**  
  Length of tickmarks into the graph (pixels)
- **axis_ticklength_out (num;5)**  
  Length of tickmarks out of the graph (pixels)
- **axis_tickmark_offset (num;8)**  
  Distance between axis and tick labels (pixels)

---

dendextend_options  
Access to dendextend_options

**Description**

This is a function inside its own environment. This enables a bunch of functions to be manipulated outside the package, even when they are called from function within the dendextend package.

TODO: describe options.

A new "warn" dendextend_options parameter. logical (FALSE). Should warning be issued?

**Usage**

dendextend_options(option, value)

**Arguments**

- **option**  
  a character scalar of the value of the options we would like to access or update.
- **value**  
  any value that we would like to update into the "option" element in dendextend_options

**Value**

a list with functions

**Author(s)**

Kurt Hornik
dendlist

Examples

dendextend_options("a")
dendextend_options("a", 1)
dendextend_options("a")
dendextend_options("a", NULL)
dendextend_options("a")
dendextend_options()

---

Creating a dendlist object from several dendrograms

Description

It accepts several dendrograms and or dendlist objects and chain them all together. This function aim to help with the usability of comparing two or more dendrograms.

Usage

dendlist(...)  

## S3 method for class 'dendlist'
plot(x, which = c(1L, 2L), ...)

Arguments

... several dendrogram/hclust/phylo or dendlist objects If an object is hclust or phylo - it will be converted into a dendrogram.
x a dendlist object
which an integer vector of length 2, indicating which of the trees in the dendlist object should be plotted (relevant for dendlist)

Details

It there are list() in the ..., they are omitted. If ... is missing, it returns an empty dendlist.

Value

A list of class dendlist where each item is a dendrogram

Examples

## Not run:

dend <- iris[, -5]  
%>% dist  
%>% hclust  
%>% as.dendrogram
dend2 <- iris[, -5]  
%>% dist  
%>% hclust(method = "single")  
%>% as.dendrogram
dendlist(1:4, 5, a=dend)  
# Error

# dendlist <- function (...) list(...)
dendlist(dend)
dendlist(dend, dend)

dendlist(dend, dend, dendlist(dend))
  # notice how the order of
  dendlist(dend, dend2)
  dendlist(dend) %>% dendlist(dend2)
  dendlist(dend) %>% dendlist(dend2) %>% dendlist(dend)
  dendlist(dend, dend2) %>% tanglegram
  tanglegram(tree1 = dendlist(dend, dend2))

dend <- iris[1:20, -5] %>% dist %>% hclust %>% as.dendrogram
dend2 <- iris[1:20, -5] %>% dist %>% hclust(method = "single") %>% as.dendrogram

x <- dendlist(dend, dend2)
plot(x)

## End(Not run)

---

**DendSer.dendrogram**  
*Tries to run DendSer on a dendrogram*

---

**Description**

Implements dendrogram seriation. The function tries to turn the dend into hclust, on which it runs **DendSer**.

Also, if a distance matrix is missing, it will try to use the **cophenetic** distance.

**Usage**

```r
DendSer.dendrogram(dend, ser_weight, ...)
```

**Arguments**

- **dend**: An object of class dendrogram
- **ser_weight**: Used by cost function to evaluate ordering. For cost=costLS, this is a vector of object weights. Otherwise is a dist or symmetric matrix. passed to DendSer. If it is missing, the cophenetic distance is used instead.
- **...**: parameters passed to **DendSer**

**Value**

Numeric vector giving an optimal dendrogram order

**See Also**

DendSer, DendSer.dendrogram, untangle_DendSer, rotate_DendSer
**dend_diff**

**Examples**

```r
## Not run:
library(DendSer) # already used from within the function
hc <- hclust(dist(USArrests[,1:4]), "ave")
dend <- as.dendrogram(hc)
DendSer.dendrogram(dend)

## End(Not run)
```

**dend_diff**  
*Plots two trees side by side, highlighting edges unique to each tree in red.*

**Description**

Plots two trees side by side, highlighting edges unique to each tree in red.

**Usage**

```r
dend_diff(x, ...)
```

## S3 method for class 'dendrogram'
```r
dend_diff(x, y, horiz = TRUE, ...)
```

## S3 method for class 'dendlist'
```r
dend_diff(x, ..., which)
```

**Arguments**

- **x**  
a dendrogram or dendlist to compare with
- **...**  
passed to `plot.dendrogram`
- **y**  
a dendrogram to compare with
- **horiz**  
logical (TRUE) indicating if the dendrogram should be drawn horizontally or not.
- **which**  
an integer vector indicating, in the case "x" is a dendlist, on which of the trees should the modification be performed. If missing - the change will be performed on all of objects in the dendlist.

**Value**

Invisible dendlist of both trees.

**Source**

A dendrogram implementation for phylo.diff from the distory package
See Also

distinct_edges, highlight_distinct_edges, dist.dendlist, tanglegram assign_values_to_branches_edgePar,
distinct edges,

Examples

x <- 1:5 %>% dist %>% hclust %>% as.dendrogram
y <- set(x, "labels", 5:1)

dend_diff(x, y)
dend_diff(dendlist(x, y))
dend_diff(dendlist(y, x))

dend1 <- 1:10 %>% dist %>% hclust %>% as.dendrogram
dend2 <- dend1 %>% set("labels", c(1,3,2,4, 5:10) )
dend_diff(dend1, dend2)

dist.dendlist

Topological Distances Between Two dendrograms

Description

This function computes the Robinson-Foulds distance (also known as symmetric difference) between two dendrograms. This is the sum of edges in both trees with labels that exist in only one of the two trees (i.e.: the length of distinct_edges).

This function might implement other topological distances in the future.

Usage

dist.dendlist(x, method = c("edgeset"), ...)

Arguments

x a dendlist
method currently only 'edgeset' is implemented.
... Ignored.

Value

A dist object with topological distances between all trees

See Also

distinct_edges, dist.topo, dist.multiPhylo, treedist,
distinct_edges

Examples

```r
x <- 1:5 %>% dist %>% hclust %>% as.dendrogram
y <- set(x, "labels", 5:1)

dist.dendlist(dendlist(x1 = x, x2 = x, y1 = y))
dend_diff(x, y)
```

---

**distinct_edges**

Finds the edges present in the first tree but not in the second

Description

Finds the edges present in the first tree but not in the second

Usage

```r
distinct_edges(x, y, ...)```

Arguments

- `x`: a dendrogram to find unique edges in
- `y`: a dendrogram to compare with
- `...`: Ignored.

Value

A numeric vector of edge ids for the first tree (x) that are not present in the second tree (y).

Source

A dendrogram implementation for distinct.edges from the distory package

See Also

distinct_edges, highlight_distinct_edges, dist.dendlist, tanglegram distinct.edges

Examples

```r
x <- 1:5 %>% dist %>% hclust %>% as.dendrogram
y <- set(x, "labels", 5:1)
distinct_edges(x, y)
distinct_edges(y, x)
dend_diff(x, y)
```
**dist_long**

<table>
<thead>
<tr>
<th>Description</th>
<th>Turns a dist object to a &quot;long&quot; table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage</td>
<td>dist_long(d, ...)</td>
</tr>
<tr>
<td>Arguments</td>
<td>d: a distance object</td>
</tr>
<tr>
<td></td>
<td>...: not used</td>
</tr>
<tr>
<td>Value</td>
<td>A data.frame with two columns of rows and column names of the dist object and a third column (distance) with the distance between the two.</td>
</tr>
<tr>
<td>Examples</td>
<td>data(iris)</td>
</tr>
<tr>
<td></td>
<td>iris[2:6,-5] %&gt;% dist %&gt;% data.matrix</td>
</tr>
<tr>
<td></td>
<td>iris[2:6,-5] %&gt;% dist %&gt;% as.vector</td>
</tr>
<tr>
<td></td>
<td>iris[2:6,-5] %&gt;% dist %&gt;% dist_long</td>
</tr>
<tr>
<td></td>
<td># This can later be used to making a network plot based on the distances.</td>
</tr>
</tbody>
</table>

**duplicate_leaf**

<table>
<thead>
<tr>
<th>Description</th>
<th>Duplicate a leaf X times</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage</td>
<td>duplicate_leaf(dend, leaf_label, times, fix_members = TRUE, fix_order = TRUE, fix_midpoint = TRUE, ...)</td>
</tr>
</tbody>
</table>
entanglement

Arguments

dend a dendrogram object
leaf_label the label of the leaf to replicate.
times the number of times we will have this leaf after replication
fix_members logical (TRUE). Fix the number of members in attr using fix_members_attr.dendrogram
fix_order logical (TRUE). Fix the leaves order
fix_midpoint logical (TRUE). Fix the midpoint value. If TRUE, it overrides "fix_members" and turns it into TRUE (since it must have a correct number of members in order to work). values using rank_order.dendrogram

Value

A dendrogram, after duplicating one of its leaves.

Examples

## Not run:
# define dendrogram object to play with:
dend <- USArrests[,1:3,] %>% dist %>% hclust(method = "ave") %>% as.dendrogram
plot(dend)
duplicate_leaf(dend, "Alaska", 3)
duplicate_leaf(dend, "Arizona", 2, fix_members =FALSE, fix_order = FALSE)
plot(duplicate_leaf(dend, "Alaska", 2))
plot(duplicate_leaf(dend, "Alaska", 4))
plot(duplicate_leaf(dend, "Arizona", 2))
plot(duplicate_leaf(dend, "Arizona", 4))

## End(Not run)

entanglement Measures entanglement between two trees

Description

Measures the entanglement between two trees. Entanglement is a measure between 1 (full entanglement) and 0 (no entanglement). The exact behavior of the number depends on the L norm which is chosen.

Usage

entanglement(tree1, ...)

## S3 method for class 'dendrogram'
entanglement(tree1, tree2, L = 1.5, leaves_matching_method = c("labels", "order"), ...)
## S3 method for class 'dendlist'
entanglement(tree1, which = c(1L, 2L), ...)

## S3 method for class 'hclust'
entanglement(tree1, tree2, ...)

## S3 method for class 'phylo'
entanglement(tree1, tree2, ...)

### Arguments

- **tree1**: a tree object (of class dendrogram/hclust/phylo).
- **tree2**: a tree object (of class dendrogram/hclust/phylo).
- **which**: an integer vector of length 2, indicating which of the trees in a dendlist object should have their entanglement calculated.
- **L**: the distance norm to use for measuring the distance between the two trees. It can be any positive number, often one will want to use 0, 1, 1.5, 2 (see 'details' for more).
- **leaves_matching_method**: a character scalar, either "order" or "labels" (default). If using "labels", then we use the labels for matching the leaves order value (safer). And if "order" then we use the old leaves order value for matching the leaves order value. Using "order" is faster, but "labels" is safer. "order" will assume that the original two trees had their labels and order values MATCHED. Hence, it is best to make sure that the trees used here have the same labels and the SAME values matched to these values - and then use "order" (for fastest results).

### Details

Entanglement is measured by giving the left tree’s labels the values of 1 till tree size, and than match these numbers with the right tree. Now, entanglement is the L norm distance between these two vectors. That is, we take the sum of the absolute difference (each one in the power of L). e.g: `sum(abs(x-y)**L)`. And this is devided by the "worst case" entanglement level (e.g: when the right tree is the complete reverse of the left tree).

L tells us which panelty level we are at (L0, L1, L2, partial L’s etc). L>1 means that we give a big panelty for sharp angles. While L>0 means that any time something is not a streight horizontal line, it gets a large penalty If L=0.1 it means that we much prefer streight lines over non streight lines.

### Value

The number of leaves in the tree
See Also
tanglegram, match_order_by_labels.

Examples

```r
## Not run:
dend1 <- iris[, -5] %>% dist %>% hclust("com") %>% as.dendrogram
dend2 <- iris[, -5] %>% dist %>% hclust("sin") %>% as.dendrogram
dend12 <- dendlist(dend1, dend2)
tanglegram(dend12)

entanglement(dend12)
entanglement(dend12, L = 0)
entanglement(dend12, L = 0.25)
entanglement(dend1, dend2, L = 0) # 1
entanglement(dend1, dend2, L = 0.25) # 0.97
entanglement(dend1, dend2, L = 1) # 0.93
entanglement(dend1, dend2, L = 2) # 0.88

# a somewhat better tanglegram
tanglegram(sort(dend1), sort(dend2))
# and also a MUCH better entanglement
entanglement(sort(dend1), sort(dend2), L=1.5) # 0.0811
# but not that much, for L=0.25
entanglement(sort(dend1), sort(dend2), L=.25) # 0.579

# massing up the order of leaves is dangerous:
entanglement(dend1, dend2, 1.5, "order") # 0.91
order.dendrogram(dend2) <- seq_len(nleaves(dend2))
# this 0.95 number is NO LONGER correct!!
entanglement(dend1, dend2, 1.5, "order") # 0.95
# but if we use the "labels" method - we still get the correct number:
entanglement(dend1, dend2, 1.5, "labels") # 0.91

# however, we can fix our dend2, as follows:
dend2 <- match_order_by_labels(dend2, dend1)
# Now that labels and order are matched - entanglement is back at working fine:
entanglement(dend1, dend2, 1.5, "order") # 0.91

## End(Not run)
```
**fac2num**  
_Turns a factor into a number_

**Description**

Turning a factor into a number is not trivial. Using `as.numeric` would only return to us the indicator numbers and NOT the factor levels turned into a number. `fac2num` simply turns a factor into a number, as we often need.

**Usage**

```r
fac2num(x, force_integer = FALSE, keep_names = TRUE, ...)
```

**Arguments**

- `x` - an object.
- `force_integer` - logical (FALSE). Should the values returned be integers?
- `keep_names` - logical (TRUE). Should the values returned keep the names of the original vector?
- `...` - ignored.

**Value**

- if `x` is an object - it returns logical - is the object of class dendrogram.

**Examples**

```r
cx <- factor(3:5)
as.numeric(cx) # 1 2 3
fac2num(cx)  # 3 4 5
```

---

**fix_members_attr.dendrogram**  
_Fix members attr in a dendrogram_

**Description**

Fix members attr in a dendrogram after (for example), the tree was pruned or manipulated.

**Usage**

```r
fix_members_attr.dendrogram(dend, ...)
```
Arguments

dend      a dendrogram object

...      not used

Value

A dendrogram, after adjusting the members attr in all of its nodes.

Examples

# define dendrogram object to play with:
hc <- hclust(dist(USArrests[1:3,]), "ave")
dend <- as.dendrogram(hc)
# plot(dend)
# prune one leaf
dend[[2]] <- dend[[2]][[1]]
# plot(dend)
dend # but it is NO LONGER true that it has 3 members total!
fix_members_attr.dendrogram(dend) # it now knows it has only 2 members :)

hc <- hclust(dist(USArrests[1:3,]), "ave")
dend <- as.dendrogram(hc)

identical(prune_leaf(dend , "Alaska"), fix_members_attr.dendrogram(prune_leaf(dend , "Alaska")))
str(unclass(prune_leaf(dend , "Alaska")))
str(unclass(fix_members_attr.dendrogram(prune_leaf(dend , "Alaska"))))

---

flatten.dendrogram  Flatten the branches of a dendrogram's root

Description

The function makes sure the two branches of the root of a dendrogram will have the same height. The user can choose how to decide which height to use.

Usage

flatten.dendrogram(dend, FUN = max, new_height, ...)

Arguments

dend      dendrogram object
FUN       how to choose the new height of both branches (defaults to taking the max between the two)
new_height overrides FUN, and sets the new height of the two branches manually
...      passed on (not used)
Value

A dendrogram with both of the root’s branches of the same height

Examples

\[
\text{hc} \leftarrow \text{hclust}(\text{dist(USArrests}[2:9,]), \text{"com"})
\]
\[
\text{dend} \leftarrow \text{as.dendrogram(hc)}
\]
\[
\text{attr(dend}[1]], \text{"height"}) \leftarrow 150 \quad \text{# make the height un-equal}
\]

\[
\text{par(mfrow = c}(1,2))
\]
\[
\text{plot(dend, main = "original tree")}
\]
\[
\text{plot(flatten.dendrogram(dend), main = "Raised tree")}
\]

Description

Rotate a branch in a tree so that the locations of two bundles of leaves are flipped.

Usage

\[
\text{flip_leaves}(\text{dend, leaves1, leaves2, ...})
\]

Arguments

- **dend**: a dendrogram object
- **leaves1**: a vector of leaves order value to flip.
- **leaves2**: a (second) vector of leaves order value to flip.
- **...**: not used

Details

This function is based on a bunch of string manipulation functions. There may be a smarter/better way for doing it...

Value

A dendrogram object with flipped leaves.

See Also

tanglegram, match_order_by_labels, entanglement.
Calculating Fowlkes-Mallows Index

Description

Calculating Fowlkes-Mallows index.

As opposed to the `FM_index_profdpm` function, the `FM_index_R` function also calculates the expectancy and variance of the FM Index under the null hypothesis of no relation.

Usage

```r
FM_index(A1_clusters, A2_clusters, include_EV = TRUE,
          assume_sorted_vectors = FALSE, warn = dendextend_options("warn"), ...)  
```

Arguments

- **A1_clusters**: a numeric vector of cluster grouping (numeric) of items, with a name attribute of item name for each element from group A1. These are often obtained by using some k cut on a dendrogram.
- **A2_clusters**: a numeric vector of cluster grouping (numeric) of items, with a name attribute of item name for each element from group A2. These are often obtained by using some k cut on a dendrogram.
- **include_EV**: logical (TRUE). Should we calculate expectancy and variance of the FM Index under null hypothesis of no relation between the clusterings? If TRUE (Default) - then the `FM_index_R` function, else (FALSE) we use the (faster) `FM_index_profdpm` function.
- **assume_sorted_vectors**: logical (FALSE). Can we assume to two group vectors are sorter so that they have the same order of items? IF FALSE (default), then the vectors will be sorted based on their name attribute.
- **warn**: logical (default from dendextend_options("warn") is FALSE). Set if warning are to be issued, it is safer to keep this at TRUE, but for keeping the noise down, the default is FALSE.
- **...**: Ignored (passed to `FM_index_R/FM_index_profdpm`).

Examples

```r
## Not run:
dend1 <- USArrests[1:5,] %>% dist %>% hclust %>% as.dendrogram
dend2 <- flip_leaves(dend1, c(3,5), c(1,2))
tanglegram(dend1,dend2)
entanglement(dend1,dend2, L = 2) # 0.4

## End(Not run)
```
Details

From Wikipedia:

Fowlkes-Mallows index (see references) is an external evaluation method that is used to determine the similarity between two clusterings (clusters obtained after a clustering algorithm). This measure of similarity could be either between two hierarchical clusterings or a clustering and a benchmark classification. A higher the value for the Fowlkes-Mallows index indicates a greater similarity between the clusters and the benchmark classifications.

Value

The Fowlkes-Mallows index between two vectors of clustering groups.

Includes the attributes E_FM and V_FM for the relevant expectancy and variance under the null hypothesis of no-relation.

References


http://en.wikipedia.org/wiki/Fowlkes-Mallows_index

See Also

cor_bakers_gamma, FM_index_profdpm

Examples

```r
## Not run:

set.seed(23235)
s <- TRUE # sample(1:150, 10 )
hc1 <- hclust(dist(iris[ss,-5]), "com")
hc2 <- hclust(dist(iris[ss,-5]), "single")
# dend1 <- as.dendrogram(hc1)
# dend2 <- as.dendrogram(hc2)
# cutree(dend1)

FM_index(cutree(hc1, k=3), cutree(hc1, k=3)) # 1 with EV
FM_index(cutree(hc1, k=3), cutree(hc1, k=3), include_EV= FALSE) # 1

# checking speed gains
library(microbenchmark)
microbenchmark(FM_index(cutree(hc1, k=3), cutree(hc1, k=3)),
  FM_index(cutree(hc1, k=3), cutree(hc1, k=3),
    include_EV= FALSE),
  FM_index(cutree(hc1, k=3), cutree(hc1, k=3),
    include_EV= TRUE, assume_sorted_vectors=TRUE),
  FM_index(cutree(hc1, k=3), cutree(hc1, k=3),
    include_EV= FALSE, assume_sorted_vectors=TRUE)
)

# C code is 1.2-1.3 times faster.
```
set.seed(1341)
FM_index(cutree(hc1, k=3), sample(cutree(hc1, k=3)),
  assume_sorted_vectors =TRUE) # 0.38037
FM_index(cutree(hc1, k=3), sample(cutree(hc1, k=3)),
  assume_sorted_vectors =FALSE) # 0.8059
FM_index(cutree(hc1, k=3), cutree(hc2, k=3)) # 0.8059
FM_index(cutree(hc1, k=30), cutree(hc2, k=30)) # 0.4529

fo <- function(k) FM_index(cutree(hc1, k), cutree(hc2, k))
lapply(1:4, fo)
ks <- 1:150
plot(sapply(ks, fo), ks, type = "b", main = "Bk plot for the iris dataset")

## End(Not run)

---

**FM_index_permutation**

*Calculating Fowlkes-Mallows Index under H0*

**Description**

Calculating Fowlkes-Mallows index under the null hypothesis of no relation between the clusterings (random order of the items labels).

**Usage**

```r
FM_index_permutation(A1_clusters, A2_clusters,
  warn = dendextend_options("warn"), ...)
```

**Arguments**

- `A1_clusters` a numeric vector of cluster grouping (numeric) of items, with a name attribute of item name for each element from group A1. These are often obtained by using some k cut on a dendrogram.
- `A2_clusters` a numeric vector of cluster grouping (numeric) of items, with a name attribute of item name for each element from group A2. These are often obtained by using some k cut on a dendrogram.
- `warn` logical (default from dendextend_options("warn") is FALSE). Set if warning are to be issued, it is safer to keep this at TRUE, but for keeping the noise down, the default is FALSE.
- `...` Ignored (passed to FM_index_R/FM_index_profdpm).

**Value**

The Fowlkes-Mallows index between two vectors of clustering groups. Under H0. (a double without attr)
References


http://en.wikipedia.org/wiki/Fowlkes-Mallows_index

See Also

cor_bakers_gamma, FM_index_profdpm, FM_index_R, FM_index

Examples

## Not run:

```r
set.seed(23235)
ss <- TRUE # sample(1:150, 10)
hc1 <- hclust(dist(iris[ss,-5]), "com")
hc2 <- hclust(dist(iris[ss,-5]), "single")
# dend1 <- as.dendrogram(hc1)
# dend2 <- as.dendrogram(hc2)
# cutree(dend1)

# small k
A1_clusters <- cutree(hc1, k=3) # will give a right tailed distribution
# large k
A1_clusters <- cutree(hc1, k=50) # will give a discrete distribution
# "medium" k
A1_clusters <- cutree(hc1, k=25) # gives almost the normal distribution!
A2_clusters <- A1_clusters

R <- 10000
set.seed(414130)
FM_index_H0 <- replicate(R, FM_index_permutation(A1_clusters, A2_clusters)) # can take 10 sec
plot(density(FM_index_H0), main = "FM Index distribution under H0\n\n(10000 permutation)")
abline(v = mean(FM_index_H0), col = 1, lty = 2)
# The permutation distribution is with a heavy right tail:
library(psych)
skew(FM_index_H0) # 1.254
kurtosi(FM_index_H0) # 2.5427

mean(FM_index_H0); var(FM_index_H0)
the_FM_index <- FM_index(A1_clusters, A2_clusters)
the_FM_index
our_dnorm <- function(x) {
  dnorm(x, mean = attr(the_FM_index, "E_FM"),
        sd = sqrt(attr(the_FM_index, "V_FM")))
}
# our_dnorm(0.35)
curve(our_dnorm, from = -1, to=1, n=10000)
abline(v = attr(the_FM_index, "E_FM"), col = 4, lty = 2)
```
Calculating Fowlkes-Mallows index using the profdpm R package

Description

Calculating Fowlkes-Mallows index using the pci function from the profdpm R package. This function uses C code (thanks to Matthew Shotwell’s work) and is a bit faster than the R code (from my simple tests - it is about 1.2-1.3 times faster - which is not much but it might be useful). As opposed to the \texttt{FM\_index\_R} function, the \texttt{FM\_index\_profdpm} function does NOT calculate the expectancy or the variance of the FM Index under the null hypothesis of no relation. This function also allows us to compare our calculations with an independent writing of a function calculating the same statistic.

Usage

\begin{verbatim}
FM_index_profdpm(A1_clusters, A2_clusters, assume_sorted_vectors = FALSE, 
                 warn = dendextend_options("warn"), ...)
\end{verbatim}

Arguments

- \texttt{A1\_clusters} a numeric vector of cluster grouping (numeric) of items, with a name attribute of item name for each element from group A1. These are often obtained by using some \texttt{k cut} on a dendrogram.
- \texttt{A2\_clusters} a numeric vector of cluster grouping (numeric) of items, with a name attribute of item name for each element from group A2. These are often obtained by using some \texttt{k cut} on a dendrogram.
- \texttt{assume_sorted_vectors} logical (FALSE). Can we assume to two group vectors are sorter so that they have the same order of items? IF FALSE (default), then the vectors will be sorted based on their name attribute.
- \texttt{warn} logical (default from dendextend\_options("warn") is FALSE). Set if warning are to be issued, it is safer to keep this at TRUE, but for keeping the noise down, the default is FALSE.

Details

From Wikipedia:

Fowlkes-Mallows index (see references) is an external evaluation method that is used to determine the similarity between two clusterings (clusters obtained after a clustering algorithm). This measure of similarity could be either between two hierarchical clusterings or a clustering and a benchmark classification. A higher the value for the Fowlkes-Mallows index indicates a greater similarity between the clusters and the benchmark classifications.
Value

The Fowlkes-Mallows index between two vectors of clustering groups.

References


http://en.wikipedia.org/wiki/Fowlkes-Mallows_index

See Also

cor_bakers_gamma

Examples

```R
## Not run:

set.seed(23235)
ss <- TRUE # sample(1:150, 10)
hc1 <- hclust(dist(iris[ss,-5]), "com")
hc2 <- hclust(dist(iris[ss,-5]), "single")
# dend1 <- as.dendrogram(hc1)
# dend2 <- as.dendrogram(hc2)
# cutree(dend1)

FM_index_profdpm(cutree(hc1, k=3), cutree(hc1, k=3)) # 1
set.seed(1341)
FM_index_profdpm(cutree(hc1, k=3), sample(cutree(hc1, k=3)),
    assume_sorted_vectors =TRUE) # 0.38037
FM_index_profdpm(cutree(hc1, k=3), sample(cutree(hc1, k=3)),
    assume_sorted_vectors =FALSE) # 1 again :)
FM_index_profdpm(cutree(hc1, k=3), cutree(hc2, k=3)) # 0.8059
FM_index_profdpm(cutree(hc1, k=30), cutree(hc2, k=30)) # 0.4529

fo <- function(k) FM_index_profdpm(cutree(hc1, k), cutree(hc2, k))
lapply(1:4, fo)
ks <- 1:150
plot(sapply(ks, fo)~ ks, type = "b", main = "Bk plot for the iris dataset")
```

## End(Not run)
Calculating Fowlkes-Mallows index.

As opposed to the `FM_index_profdpm` function, the `FM_index_R` function also calculates the expectancy and variance of the FM Index under the null hypothesis of no relation.

**Usage**

```r
FM_index_R(A1_clusters, A2_clusters, assume_sorted_vectors = FALSE,
           warn = dendextend_options("warn"), ...)  
```

**Arguments**

- `A1_clusters`: a numeric vector of cluster grouping (numeric) of items, with a name attribute of item name for each element from group A1. These are often obtained by using some k cut on a dendrogram.
- `A2_clusters`: a numeric vector of cluster grouping (numeric) of items, with a name attribute of item name for each element from group A2. These are often obtained by using some k cut on a dendrogram.
- `assume_sorted_vectors`: logical (FALSE). Can we assume the two group vectors are sorter so that they have the same order of items? IF FALSE (default), then the vectors will be sorted based on their name attribute.
- `warn`: logical (default from dendextend_options("warn") is FALSE). Set if warning are to be issued, it is safer to keep this at TRUE, but for keeping the noise down, the default is FALSE.
- `...`: Ignored.

**Details**

From Wikipedia:

Fowlkes-Mallows index (see references) is an external evaluation method that is used to determine the similarity between two clusterings (clusters obtained after a clustering algorithm). This measure of similarity could be either between two hierarchical clusterings or a clustering and a benchmark classification. A higher the value for the Fowlkes-Mallows index indicates a greater similarity between the clusters and the benchmark classifications.

**Value**

The Fowlkes-Mallows index between two vectors of clustering groups.

Includes the attributes `E_FM` and `V_FM` for the relevant expectancy and variance under the null hypothesis of no-relations.
get_branches_heights

References


http://en.wikipedia.org/wiki/Fowlkes-Mallows_index

See Also

cor_bakers_gamma, FM_index_profdpm

Examples

## Not run:

```r
set.seed(23235)
ss <- TRUE # sample(1:150, 10 )
hc1 <- hclust(dist(iris[ss,-5]), "com")
hc2 <- hclust(dist(iris[ss,-5]), "single")
# dend1 <- as.dendrogram(hc1)
# dend2 <- as.dendrogram(hc2)
#    cutree(dend1)

FM_index_R(cutree(hc1, k=3), cutree(hc1, k=3)) # 1
set.seed(1341)
FM_index_R(cutree(hc1, k=3), sample(cutree(hc1, k=3)), assume_sorted_vectors =TRUE) # 0.38037
FM_index_R(cutree(hc1, k=3), sample(cutree(hc1, k=3)), assume_sorted_vectors =FALSE) # 1 again :)
FM_index_R(cutree(hc1, k=3), cutree(hc2, k=3)) # 0.8059
FM_index_R(cutree(hc1, k=30), cutree(hc2, k=30)) # 0.4529

fo <- function(k) FM_index_R(cutree(hc1, k), cutree(hc2, k))
lapply(1:4, fo)
ks <- 1:150
plot(sapply(ks, fo)- ks, type = "b", main = "Bk plot for the iris dataset")

clu_1 <- cutree(hc2, k = 100) # this is a lie - since this one is NOT well defined!
clu_2 <- cutree(as.dendrogram(hc2), k = 100) # We see that we get a vector of NAs for this...

FM_index_R(clu_1, clu_2) # NA
```

## End(Not run)

---

get_branches_heights  Get height attributes from a dendrogram

Description

Get height attributes from a dendrogram
get_childrens_heights

Usage

get_branches_heights(tree, sort = TRUE, decreasing = FALSE, ...)

Arguments

tree  a dendrogram.
sort  logical. Should the heights be sorted?
decreasing  logical. Should the sort be increasing or decreasing? Not available for partial sorting.
...  not used.

Value

a vector of the dendrogram’s nodes heights (excluding leaves).

Examples

hc <- hclust(dist(USArrests[1:4,]), "ave")
dend <- as.dendrogram(hc)
get_branches_heights(dend)

## Not run:
dat1 <- iris[,1:5]
dat1 <- rbind(dat1,dat1,dat1,dat1,dat1,dat1)
dend = as.dendrogram(hclust(dist(dat1)))
library(microbenchmark)
library(dendextendRcpp)
microbenchmark(
  dendextendRcpp::dendextendRcpp_get_branches_heights(dend),
  old_get_branches_heights(dend, sort=F)
)

## End(Not run)
Arguments

dend a dendrogram.
... not used.

Value

a vector of the heights of a dendrogram’s current node’s (first level) children.

See Also

get_branches_heights

Examples

hc <- hclust(dist(USArrests[1:4,]), "ave")
dend <- as.dendrogram(hc)
get_childrens_heights(dend)

get_leaves_attr Get/set attributes of dendrogram’s leaves

Description

Get/set attributes of dendrogram’s leaves

Usage

get_leaves_attr(object, attribute, simplify = TRUE, ...)

Arguments

object a dendrogram object
attribute character scalar of the attribute (attr) we wish to get/set from the leaves
simplify logical. If TRUE (default), then the return vector is after using unlist on it.
... not used

Value

A vector (or a list) with the dendrogram’s leaves attribute

Source

Heavily inspired by the code in the function labels.dendrogram, so credit should go to Martin Maechler.
get_leaves_nodePar

Description
Get nodePar of dendrogram’s leaves

Usage
get_leaves_nodePar(object, simplify = FALSE, ...)

Arguments

- **object**: a dendrogram object
- **simplify**: logical (default is FALSE). If TRUE, then the return vector is after using unlist on it.
- **...**: not used

Value
A list (or a vector) with the dendrogram’s leaves nodePar attribute

See Also
get_nodes_attr, nnodes, nleaves, assign_values_to_leaves_nodePar

Examples

```r
# define dendrogram object to play with:
hc <- hclust(dist(USArrests[1:3,]), "ave")
dend <- as.dendrogram(hc)

# get_leaves_attr(dend) # error :
get_leaves_attr(dend, "label")
labels(dend, "label")
get_leaves_attr(dend, "height") # should be 0's
get_nodes_attr(dend, "height")

get_leaves_attr(dend, "nodePar")

get_leaves_attr(dend, "leaf") # should be TRUE's
get_nodes_attr(dend, "leaf") # contains NA's

get_leaves_attr(dend, "members") # should be 1's
get_nodes_attr(dend, "members") #

get_leaves_attr(dend, "members", simplify = FALSE) # should be 1's
```

---

get_leaves_nodePar  Get nodePar of dendrogram’s leaves

---
get_nodes_attr

Description

Allows easy access to attributes of branches and/or leaves, with option of returning a vector with/without NA's (for marking the missing attr value)

Usage

get_nodes_attr(object, attribute, id, include_leaves = TRUE,
include_branches = TRUE, simplify = TRUE, na.rm = FALSE, ...)

Arguments

object  
a dendrogram object
attribute  
character scalar of the attribute (attr) we wish to get from the nodes
id  
integer vector. If given - only the attr of these nodes id will be returned (via depth first search)
include_leaves  
logical. Should leaves attributes be included as well?
include_branches  
logical. Should non-leaf (branch node) attributes be included as well?
simplify  
logical (default is TRUE). should the result be simplified to a vector (using simplify2array ) if possible? If it is not possible it will return a matrix. When FALSE, a list is returned.
na.rm  
logical. Should NA attributes be REMOVED from the resulting vector?
...  
not used

See Also

get_nodes_attr, assign_values_to_leaves_nodePar, labels_colors

Examples

# define dendrogram object to play with:
hc <- hclust(dist(USArrests[1:3,]), "ave")
dend <- as.dendrogram(hc)

# get_leaves_attr(dend) # error :)
get_leaves_nodePar(dend)
labels_colors(dend) <- 1:3
get_leaves_nodePar(dend)

dend <- assign_values_to_leaves_nodePar(dend, 2, "lab.cex")
get_leaves_nodePar(dend)

plot(dend)
Value

A vector with the dendrogram’s nodes attribute. If an attribute is missing from some nodes, it will return NA in that vector.

Source

Heavily inspired by the code in the function `labels.dendrogram`, so credit should go to Martin Maechler.

See Also

`get_leaves_attr`, `nnodes`, `nleaves`

Examples

```r
# define dendrogram object to play with:
hc <- hclust(dist(USArrests[,1:3]), "ave")
dend <- as.dendrogram(hc)

# get_leaves_attr(dend) # error :)
get_leaves_attr(dend, "label")
labels(dend, "label")
get_leaves_attr(dend, "height") # should be 0's
get_nodes_attr(dend, "height")

get_leaves_attr(dend, "leaf") # should be TRUE's
get_nodes_attr(dend, "leaf") # contains NA's

get_nodes_attr(dend, "members") # should be 1's
get_nodes_attr(dend, "members", include_branches = FALSE, na.rm = TRUE) #
get_nodes_attr(dend, "members") #
get_nodes_attr(dend, "members", simplify = FALSE)
get_nodes_attr(dend, "members", include_leaves = FALSE, na.rm = TRUE) #

get_nodes_attr(dend, "members", id = c(1,3), simplify = FALSE)
get_nodes_attr(dend, "members", id = c(1,3))#

hang_dend <- hang.dendrogram(dend)
get_leaves_attr(hang_dend, "height") # no longer 0!
get_nodes_attr(hang_dend, "height") # does not include any 0s!

# does not include leaves values:
get_nodes_attr(hang_dend, "height", include_leaves = FALSE)
# remove leaves values all together:
get_nodes_attr(hang_dend, "height", include_leaves = FALSE, na.rm = TRUE)

## Not run:
```
get_nodes_xy

Get the x-y coordinates of a dendrogram’s nodes

Description

Get the x-y coordinates of a dendrogram’s nodes

Usage

get_nodes_xy(x, type = c("rectangle", "triangle"), center = FALSE, horiz = FALSE, ...)

Arguments

x a dendrogram object
type type of plot.
center logical; if TRUE, nodes are plotted centered with respect to the leaves in the branch. Otherwise (default), plot them in the middle of all direct child nodes.
horiz logical indicating if the dendrogram should be drawn horizontally or not.
... not used

Value

A 2-dimensional matrix, with rows as the number of nodes, and the first column is the x location, while the second is the y location.

Source

This is a stripped down version of the function plot.dendrogram. It performs (almost) the same task, only it does not do any plotting but it does save the x-y coordinates of the nodes.

See Also

get_nodes_attr, nnodes, nleaves

library(microbenchmark)
# get_leaves_attr is twice faster than get_nodes_attr
microbenchmark(
  get_leaves_attr(dend, "members"), # should be 1's
  get_nodes_attr(dend, "members", include_branches = FALSE, na.rm = TRUE)
)

## End(Not run)
Examples

```r
## Not run:

# If we would like to see the numbers from plot:
# ?getOption("verbose")
# options(VERBOSE=TRUE)
# options(VERBOSE=FALSE)

# -----
# Draw a depth first search illustration
# -----

dend <- 1:5 %>% dist %>% hclust %>% as.dendrogram
get_nodes_xy(dend)

# polygon(get_nodes_xy(dend), col = 2)
plot(dend,
   leaflab = "none",
   main = "Depth-first search in a dendrogram")
xy <- get_nodes_xy(dend)
for(i in 1:(nrow(xy)-1)) {
  arrows(xy[i,1], xy[i,2], angle = 17,
         length = .5,
         xy[i+1,1], xy[i+1,2],
         lty = 1, col = 3, lwd = 1.5)
}
points(xy, pch = 19, cex = 4)
text(xy, labels = 1:nodes(dend), cex = 1.2, col = "white", adj = c(0.4,0.4))

## End(Not run)
```

get_root_branches_attr

get attributes from the dendrogram’s root(!) branches

Description

get attributes from the dendrogram’s root(!) branches

Usage

```r
get_root_branches_attr(dend, the_attr, warn = dendextend_options("warn"), ...)
```

Arguments

dend: dendrogram object

the_attr: the attribute to get from the branches (for example "height")
warn logical (default from dendextend_options("warn") is FALSE). Set if warning are to be issued, it is safer to keep this at TRUE, but for keeping the noise down, the default is FALSE. Should a warning be printed when the function is used on an object which is NOT a dendrogram.

... passed on to attr

Value

The attributes of the branches (often two) of the dendrogram's root

See Also

attr

Examples

hc <- hclust(dist(USArrests[2:9,]), "com")
dend <- as.dendrogram(hc)

get_root_branches_attr(dend, "height") # 0.00000 71.96247
# plot(dend)
str(dend, 2)

---

**ggdend**

**Creates dendrogram plot using ggplot.**

Description

Several functions for creating a dendrogram plot using ggplot2. The core process is to transform a dendrogram into a ggdend object using as.ggdend, and then plot it using ggplot. These two steps can be done in one command with either the function ggplot or ggdend.

The reason we want to have as.ggdend (and not only ggplot.dendrogram), is (1) so that you could create your own mapping of ggdend and, (2) since as.ggdend might be slow for large trees, it is probably better to be able to run it only once for such cases.

A ggdend class object is a list with 3 components: segments, labels, nodes. Each one contains the graphical parameters from the original dendrogram, but in a tabular form that can be used by ggplot2+geom_segment+geom_text to create a dendrogram plot.

Usage

as.ggdend(dend, ...)

prepare.ggdend(data, ...)

## S3 method for class 'dendrogram'
as.ggdend(dend,
        type = c("rectangle", "triangle"), edge.root = FALSE, ...)
ggplot.ggdend(data, segments = TRUE,  
    labels = TRUE, horiz = FALSE, theme = theme_dendro(), ...)

ggplot.dendrogram(data, ...)

Arguments

...            mostly ignored.
dend            a dendrogram tree (to be turned into a ggdend object)
type            The type of plot, indicating the shape of the dendrogram. "rectangle" will draw  
                rectangular lines, while "triangle" will draw triangular lines.
edge.root       currently ignored. One day it might do the following: logical; if true, draw an  
                edge to the root node.
data            a ggdend class object.
segments        a logical (TRUE) if to plot the segments (branches).
labels          a logical (TRUE) if to plot the labels.
horiz           a logical (TRUE) indicating if the dendrogram should be drawn horizontally or  
                not.
theme           the ggplot2 theme to use (default is theme_dendro, can also be NULL for the  
                default ggplot2 theme)

Details

prepare.ggdend is used by plot.ggdend to take the ggdend object and prepare it for plotting.  
This is because the defaults of various parameters in dendrogram's are not always stored in the  
object itself, but are built-in into the plot.dendrogram function. For example, the color of the labels  
is not (by default) specified in the dendrogram (only if we change it from black to something else).  
Hence, when taking the object into a different plotting engine (say ggplot2), we want to prepare the  
object by filling-in various defaults. This function is automatically invoked within the plot.ggdend  
function. You would probably use it only if you’d wish to build your own ggplot2 mapping.

Value

• as.ggdend - returns an object of class ggdend which is a list with 3 components: segments,  
  labels, nodes. Each one contains the graphical parameters from the original dendrogram, but in  
a tabular form that can be used by ggplot2+geom_segment+geom_text to create a dendrogram  
plot.
• prepare.ggdend - a ggdend object (after filling it with various default values)
• ggplot.ggdend - a ggplot object

Author(s)

Tal Galili, using code modified from Andrie de Vries
Source

These are extended versions of the functions `ggdendrogram`, `dendro_data` (and the hidden dendrogram_data) from Andrie de Vries's ggdendro package. The motivation for this fork is the need to add more graphical parameters to the plotted tree. This required a strong mixture of functions from ggdendro and dendextend (to the point that it seemed better to just fork the code into its current form)

See Also

dendrogram, get_nodes_attr, get_leaves_nodePar, ggplot, ggdendrogram, dendro_data,

Examples

```r
## Not run:

library(dendextend)
# library(ggdendro)
# Create a complex dend:
dend <- iris[1:30,-5] %>% dist %>% hclust %>% as.dendrogram %>%
  set("branches_k_color", k=3) %>% set("branches_lwd", c(1.5,1,1.5)) %>%
  set("branches_lty", c(1,1,3,1,1,2)) %>%
  set("labels_colors") %>% set("labels_cex", c(.9,1.2))
# plot the dend in usual "base" plotting engine:
plot(dend)
# Now let's do it in ggplot2 :)
ggd1 <- as.ggdend(dend)
library(ggplot2)
ggplot(ggd1) # reproducing the above plot in ggplot2 :
ggplot(ggd1, horiz = TRUE) # horiz plot in ggplot2
# Adding some extra spice to it...
# creating a radial plot:
  ggplot(ggd1) + scale_y_reverse(expand = c(0.2, 0)) + coord_polar(theta="x")
  # The text doesn't look so great, so let's remove it:
  ggplot(ggd1, labels = FALSE) + scale_y_reverse(expand = c(0.2, 0)) + coord_polar(theta="x")

  # This can now be sent to plot.ly - which adds zoom-in abilities, and more.
  # Here is how it might look like: https://plot.ly/~talgalili/6/y-vs-x/

## Quick guide:
# install.packages("devtools")
# library("devtools")
# devtools::install_github("ropensci/plotly")
# library(plotly)
# set_credentials_file(...)
# you'll need to get it from here: https://plot.ly/ggplot2/getting-started/

# ggplot(ggd1)
# py <- plotly()
# py$ggplotly()

# And you'll get something like this: https://plot.ly/~talgalili/6/y-vs-x/
```
hang.dendrogram

# Another example: https://plot.ly/ggplot2/

## End(Not run)

---

**Description**

Adjust the height attr in all of the dendrogram leaves so that the tree will hang. This is similar to `as.dendrogram(hclust, hang=0.1)` only that it now works on other object than `hclust` turned into a dendrogram. For example, this allows us to hang non-binary trees.

**Usage**

```
hang.dendrogram(dend, hang = 0.1, hang_height, ...)
```

**Arguments**

- `dend` a dendrogram object
- `hang` The fraction of the plot height by which labels should hang below the rest of the plot. A negative value will cause the labels to hang down from 0.
- `hang_height` is missing, then using "hang". If a number is given, it overrides "hang" (except if "hang" is negative)
- `...` not used

**Value**

A dendrogram, after adjusting the height attr in all of its leaves, so that the tree will hang.

**Source**

Noticing that `as.dendrogram` has a "hang" parameter was thanks to Enrique Ramos’s answer here: [http://stackoverflow.com/questions/17088136/plot-horizontal-dendrogram-with-hanging-leaves-r](http://stackoverflow.com/questions/17088136/plot-horizontal-dendrogram-with-hanging-leaves-r)

**Examples**

```
# define dendrogram object to play with:
hc <- hclust(dist(USArrests[1:5,]), "ave")
dend <- as.dendrogram(hc)

par(mfrow = c(1,2))
plot(hang.dendrogram(dend))
plot(hc)
# identical(as.dendrogram(hc, hang = 0.1), hang.dendrogram(dend, hang = 0.1))
# TRUE!!
```
par(mfrow = c(1,4))

plot(dend)
plot(hang.dendrogram(dend, hang = 0.1))
plot(hang.dendrogram(dend, hang = 0))
plot(hang.dendrogram(dend, hang = -0.1))

par(mfrow = c(1,1))
plot(hang.dendrogram(dend), horiz = TRUE)

heights_per_k.dendrogram

Description
Which height will result in which k for a dendrogram

Usage
heights_per_k.dendrogram(tree, ...)

Arguments

  tree               a dendrogram.

  ...               not used.

Value

  a vector of heights, with its names being the k clusters that will result for cutting the dendrogram at each height.

Examples

## Not run:
hc <- hclust(dist(USArrests[1:4,]), "ave")
dend <- as.dendrogram(hc)
heights_per_k.dendrogram(dend)
## 1 2 3 4
##86.47086 68.84745 45.98871 28.36531

cutree(hc, h = 68.8) # and indeed we get 2 clusters

unbranch_dend <- unbranch(dend,2)
plot(unbranch_dend)
heights_per_k.dendrogram(unbranch_dend)
##1 3 4
##97.90023 57.41808 16.93594
# we do NOT have a height for k=2 because of the tree's structure.

```r
library(microbenchmark)
dend = as.dendrogram(hclust(dist(iris[1:150,-5])))
dend = as.dendrogram(hclust(dist(iris[1:30,-5])))
dend = as.dendrogram(hclust(dist(iris[1:3,-5])))
microbenchmark(
  # dendextendRcpp::heights_per_k.dendrogram(dend),
  # dendextendRcpp::dendextendRcpp_heights_per_k.dendrogram(dend),
  # dendextendRcpp::old_heights_per_k.dendrogram(dend)
)
```

# improvement is 10 times faster (in Rcpp) for a tree of size 3
# 76 times faster for a tree of size 30
# And:
# 134 times faster for a tree of size 150!!

## End(Not run)

---

**highlight_distinct_edges**

_highlight distinct edges in a tree (compared to another one)_

### Description

Highlight distinct edges in a tree (compared to another one) by changing the branches’ color, line width, or line type.

This function enables this feature in `dend_diff` and `tanglegram`.

### Usage

```r
highlight_distinct_edges(x, ...)
```

### Arguments

- `x` a dendrogram or `dendlist` to find unique edges in (to highlight)
- `...` Ignored.
- `y` a dendrogram to compare with
value | a new value scalar for the edgePar attribute.
edgePar | a character indicating the value inside edgePar to adjust. Can be either "col", "lty", or "lwd".
which | an integer vector indicating, in the case "x" is a dendlist, on which of the trees should the modification be performed. If missing - the change will be performed on all of objects in the dendlist.

Value

A dendrogram with modified edges - the distinct ones are changed (color, line width, or line type)

See Also

distinct_edges, highlight_distinct_edges, dist.dendlist, tanglegram, assign_values_to_branches_edgePar,
distinct.edges,

Examples

```r
x <- 1:5 dist hclust as.dendrogram
y <- set(x, "labels", 5:1)
distinct_edges(x, y)
distinct_edges(y, x)

par(mfrow = c(1,2))
plot(highlight_distinct_edges(x, y))
plot(y)

# tanglegram(highlight_distinct_edges(x, y), y)
# dend_diff(x, y)
```

```
## Not run:

# using highlight_distinct_edges combined with dendlist and set
# to clearly highlight "stable" branches.
data(iris); ss <- c(1:5, 51:55, 101:105)
iris1 <- iris[ss,-5] dist hclust(method = "single") as.dendrogram
iris2 <- iris[ss,-5] dist hclust(method = "complete") as.dendrogram
iris12 <- dendlist(iris1, iris2) set("branches_k_color", k=3) set("branches_lwd", 3) highlight_distinct_edges(value = 1, edgePar = "lwd")
iris12 %>% untangle(method = "step2side") %>%
tanglegram(sub="Iris dataset", main_left = "'single' clustering", main_right = "'complete' clustering")
```

## End(Not run)
**identify.dendrogram**  Identify Clusters in a Dendrogram (not hclust)

**Description**
Just like *identify.hclust*: reads the position of the graphics pointer when the (first) mouse button is pressed. It then cuts the tree at the vertical position of the pointer and highlights the cluster containing the horizontal position of the pointer. Optionally a function is applied to the index of data points contained in the cluster.

**Usage**
```r
## S3 method for class 'dendrogram'
identify(x, FUN = NULL, N = 20, MAXCLUSTER, 
        DEV.FUN = NULL, horiz = FALSE, ...)  
```

**Arguments**
- **x**: a dendrogram object.
- **FUN**: (optional) function to be applied to the index numbers of the data points in a cluster (see 'Details' below).
- **N**: the maximum number of clusters to be identified.
- **MAXCLUSTER**: the maximum number of clusters that can be produced by a cut (limits the effective vertical range of the pointer).
- **DEV.FUN**: (optional) integer scalar. If specified, the corresponding graphics device is made active before FUN is applied.
- **horiz**: logical (FALSE), indicating if the rectangles should be drawn horizontally or not (for when using plot(dend, horiz = TRUE)).
- **...**: further arguments to FUN.

**Details**
By default clusters can be identified using the mouse and an invisible list of indices of the respective data points is returned. If FUN is not NULL, then the index vector of data points is passed to this function as first argument, see the examples below. The active graphics device for FUN can be specified using DEV.FUN. The identification process is terminated by pressing any mouse button other than the first, see also identify.

**Value**
(Invisibly) returns a list where each element contains a vector of data points contained in the respective cluster.

**Source**
This function is based on *identify.hclust*, with slight modifications to have it work with a dendrogram, as well as adding "horiz"
intersect_trees

Intersect trees

Description

Return two trees after pruning them so that the only leaves left are the intersection of their labels.

Usage

intersect_trees(x1, x2, warn = dendextend_options("warn"), ...)

Arguments

x1    tree object (dendrogram/hclust/phylo)
x2    tree object (dendrogram/hclust/phylo)
warn  logical (default from dendextend_options("warn") is FALSE). Set if warning are to be issued, it is safer to keep this at TRUE, but for keeping the noise down, the default is FALSE. Should a warning be issued if there was a need to perform intersection.
...   passed on

Value

A dendlist with two pruned trees

See Also

prune, intersect, labels
is.dendlist

Checks if an object is of class dendlist

Description

Checks if an object is of class dendlist

Usage

is.dendlist(x)

Arguments

x

Whatever object to check

Value

Logical

Examples

# TRUE:
is.dendlist(dendlist())
# FALSE
is.dendlist(1)
is.dendrogram  Is the object of class dendrogram

Description
Is the object of class dendrogram

Usage
is.dendrogram(x)

Arguments
x  an object.

Value
logical - is the object of class dendrogram.

is.hclust  Is the object of class hclust

Description
Is the object of class hclust

Usage
is.hclust(x)

Arguments
x  an object.

Value
logical - is the object of class hclust.
is.natural.number  

Check if numbers are natural

Description

Vectorized function for checking if numbers are natural or not. Helps in checking if a vector is of type "order".

Usage

```r
is.natural.number(x, tol = .Machine$double.eps^0.5, ...)
```

Arguments

- `x`  
  a vector of numbers
- `tol`  
  tolerance to floating point issues.
- `...`  
  (not currently in use)

Value

logical - is the entered number natural or not.

Author(s)

Marco Gallotta (a.k.a: marcog), Tal Galili

Source

This function was written by marcog, as an answer to my question here: [http://stackoverflow.com/questions/4562257/what-is-the-fastest-way-to-check-if-a-number-is-a-positive-natural-number-in](http://stackoverflow.com/questions/4562257/what-is-the-fastest-way-to-check-if-a-number-is-a-positive-natural-number-in)

See Also

`is.numeric`, `is.double`, `is.integer`

Examples

```r
is.natural.number(1) # is TRUE
(x <- seq(-1.5, by=0.5) )
is.natural.number( x )
# is.natural.number( "a" )
all(is.natural.number( x ))
```
is.phylo  

*Is the object of class phylo*

**Description**

Is the object of class phylo

**Usage**

`is.phylo(x)`

**Arguments**

- `x`: an object.

**Value**

logical - is the object of class phylo.

---

is_null_list  

*Checks if the value is an empty list()*

**Description**

Checks if the value is an empty list()

**Usage**

`is_null_list(x)`

**Arguments**

- `x`: whatever object to check

**Value**

logical
### Examples

```r
c # I can run this only if I'd make is_null_list exported
c # Not run:
c # TRUE:
c is_null_list(list())
c # FALSE
is_null_list(list(1))
is_null_list()

x <- list(1, list(), 123)
ss_list <- sapply(x, is_null_list)
x <- x[!ss_list]
x

x <- list(1, list(), 123)
ss_list <- sapply(x, is_null_list)
x <- list(list())
x

## End(Not run)

## Not run:
c # error
is_null_list()

## End(Not run)
```

---

### Description

"label" assignment operator for vectors, dendrogram, and hclust classes.

```
#'
```

### Usage

```r
labels(object, ...) <- value

## S3 method for class 'matrix'
labels(object, which = c("colnames", "rownames"), ...)

## S3 replacement method for class 'matrix'
labels(object, which = c("colnames", "rownames"), ...) <- value

## S3 replacement method for class 'dendrogram'
labels(object, ...) <- value
```
## S3 method for class 'hclust'
labels(object, order = TRUE, ...)

## S3 replacement method for class 'hclust'
labels(object, ...) <- value

## S3 method for class 'phylo'
labels(object, ...)

## S3 replacement method for class 'phylo'
labels(object, ...) <- value

## S3 method for class 'hclust'
labels(object, order = TRUE, ...)

## S3 replacement method for class 'hclust'
labels(object, ...) <- value

### Arguments

- **object**: a variable name (possibly quoted) whose label are to be updated
- **...**: parameters passed (not currently in use)
- **value**: a value to be assigned to object’s label
- **order**: default is FALSE. Only relevant for extracting labels from an `hclust` object (with `labels.hclust`). Setting `order=TRUE` will return labels in their order in the dendrogram, instead of the original labels order retained from `object$labels` - which usually corresponding to the row or column names of the `dist` object provided to the `hclust` function.
- **which**: “colnames” or “rownames”, to which of the two should labels refer to.

### Details

#### Value

The updated object

#### Author(s)

Gavin Simpson, Tal Galili (with some ideas from Gregory Jefferis’s dendroextras package)

#### Source

The functions here are based on code by Gavin and kohske from (adopted to dendrogram by Tal Galili): [http://stackoverflow.com/questions/4614223/how-to-have-the-following-work-labelsx-some-value](http://stackoverflow.com/questions/4614223/how-to-have-the-following-work-labelsx-some-value). Also with some ideas from Gregory Jefferis’s dendroextras package.
See Also

labels, labels.matrix

Examples

```r
x <- 1:3
labels(x)
labels(x) <- letters[1:3]
labels(x) # [1] "a" "b" "c"
```
```
x
# a b c
# 1 2 3

# get("labels<-")

# Example for matrix objects

x <- matrix(1:9, 3, 3)
labels(x)
x
labels(x) <- letters[1:3]
x
labels(x, which = "rownames") <- LETTERS[24:26]
x
# a b c
#X 1 4 7
#Y 2 5 8
#Z 3 6 9

# Example for using the assignment with dendrogram and hclust objects:
hc <- hclust(dist(USArrests[,1:3]), "ave")
dend <- as.dendrogram(hc)

labels(hc) # "Arizona" "Alabama" "Alaska"
labels(hc) <- letters[1:3]
labels(hc) # "a" "b" "c"
labels(dend) # "Arizona" "Alabama" "Alaska"
labels(dend) <- letters[1:3]
labels(dend) # "a" "b" "c"
labels(dend) <- LETTERS[1:2] # will produce a warning
labels(dend) # "A" "B" "A"
labels(dend) <- LETTERS[4:6] # will replace the labels correctly
# (the fact the tree had duplicate labels will not cause a problem)
labels(dend) # "D" "E" "F"
```

labels_colors

Retrieve/assign colors to the labels of a dendrogram
Description
Retrieve/assign colors to the labels of a dendrogram

Usage
labels_colors(object, labels = TRUE,...)
labels_colors(object, ...) <- value

Arguments
object a dendrogram object
labels Boolean (default is TRUE), should the returned vector of colors return with the leaves labels as names.
... not used
value a vector of colors to be used as new label’s colors for the dendrogram

Value
A vector with the dendrogram’s labels colors (or a colored dendrogram, in case assignment is used). The colors are labeled.

Source
Heavily inspired by the code in the example of dendrapply, so credit should go to Martin Maechler. I also implemented some ideas from Gregory Jefferis’s dendroextras package (having the "names" of the returned vector be the labels).

Examples
# define dendrogram object to play with:
hc <- hclust(dist(USArrests[1:3,]), "ave")
dend <- as.dendrogram(hc)

# Defaults:
labels_colors(dend)
plot(dend)

# let's add some color:
labels_colors(dend) <- 2:4
labels_colors(dend)
plot(dend)

# doesn't work...
# get_nodes_attr(dend, "nodePar", include_branches = FALSE)

# changing color to black
labels_colors(dend) <- 1
labels_colors(dend)
ladderize

plot(dend)

# removing color (and the nodePar completely - if it has no other attributed but lab.col)
labels_colors(dend) <- NULL
labels_colors(dend)
plot(dend)

ladderize Ladderize a Tree

Description

This function reorganizes the internal structure of the tree to get the ladderized effect when plotted.

Usage

ladderize(x, right = TRUE, ...)

## S3 method for class 'dendrogram'
ladderize(x, right = TRUE, ...)

## S3 method for class 'phylo'
ladderize(x, right = TRUE, phy, ...)

## S3 method for class 'dendlist'
ladderize(x, right = TRUE, ...)

Arguments

x a tree object (either a dendrogram, dendlist, or phylo)
right a logical (TRUE) specifying whether the smallest clade is on the right-hand side (when the tree is plotted upwards), or the opposite (if FALSE).
... Currently ignored.
phy a placeholder in case the user uses "phy ="

Value

A rotated tree object

See Also

ladderize, rev.dendrogram, rotate (ape)
Examples

dend <- USArrests[1:8,] %>% dist %>% hclust %>% as.dendrogram %>%
  set("labels_colors") %>% set("branches_k_color", k = 5)
set.seed(123)
dend <- shuffle(dend)

par(mfrow = c(1,3))
dend %>% plot(main = "Original")
dend %>% ladderize(TRUE) %>% plot(main = "Right (default)")
dend %>% ladderize(FALSE) %>% plot(main = "Left (rev of right)")

leaf_Colors

Return the leaf Colors of a dendrogram

Description

Return the leaf Colors of a dendrogram

Usage

leaf_Colors(d, col_to_return = c("edge", "node", "label"))

Arguments

d the dendrogram

col_to_return Character scalar - kind of Color attribute to return

Details

The returned Colors will be in dendrogram order.

Value

named character vector of Colors, NA_character_ where missing

Author(s)

jefferis

See Also

slice, color_branches

Examples

dend <- USArrests %>% dist %>% hclust(method = "ave") %>% as.dendrogram
d5=color_branches(dend,5)
leaf_Colors(d5)
**lowest_common_branch**

Find lowest common branch were the two items are shared

**Description**

Given two vectors, for two items, of cluster belonging - the function finds the lowest branch (e.g: largest number of k clusters) for which the two items are in the same cluster for the two trees.

**Usage**

`lowest_common_branch(item1, item2, ...)`

**Arguments**

- `item1`: a named numeric vector (of cluster group with names of k level)
- `item2`: a named numeric vector (of cluster group with names of k level)
- `...`: not used

**Value**

The first location (from left) where the two vectors have the same A dendrogram, after adjusting the members attr in all of its nodes.

**See Also**

- `cor_bakers_gamma`

**Examples**

```r
item1 <- structure(c(1L, 1L, 1L, 1L), .Names = c("1", "2", "3", "4"))
item2 <- structure(c(1L, 1L, 2L, 2L), .Names = c("1", "2", "3", "4"))
lowest_common_branch(item1, item2)
```

---

**match_order_by_labels**

Adjust the order of one dendrogram based on another (using labels)

**Description**

Takes one dendrogram and adjusts its order leaves values based on the order of another dendrogram. The values are matched based on the labels of the two dendrograms.

This allows for faster entanglement running time, since we can be sure that the leaves order is just as using their labels.
Usage

match_order_by_labels(dend_change, dend_template,
   check_that_labels_match = TRUE)

Arguments

dend_change      tree object (dendrogram)
dend_template    tree object (dendrogram)
check_that_labels_match
   logical (TRUE). If to check that the labels in the two dendrogram match. (if they
do not, the function aborts)

Value

Returns dend_change after adjusting its order values to be like dend_template.

See Also

entanglement, tanglegram

Examples

## Not run:

dend <- USArrests[1:4,] %>% dist %>% hclust %>% as.dendrogram
order.dendrogram(dend) # c(4L, 3L, 1L, 2L)

dend_changed <- dend
order.dendrogram(dend_changed) <- 1:4
order.dendrogram(dend_changed) # c(1:4)

# now let's fix the order of the new object to be as it was:
dend_changed <- match_order_by_labels(dend_changed, dend)
# these two are now the same:
order.dendrogram(dend_changed)
order.dendrogram(dend))

## End(Not run)
Description

Takes one dendrogram and adjusts its order leaves values based on the order of another dendrogram. The values are matched based on the order of the two dendrograms.

This allows for faster `entanglement` running time, since we can be sure that the leaves order is just as using their labels.

This is a function is FASTER than `match_order_by_labels`, but it assumes that the order and the labels of the two trees are matching!!

This will allow for a faster calculation of `entanglement`.

Usage

```r
match_order_dendrogram_by_old_order(dend_change, dend_template, dend_change_old_order, check_that_labels_match = FALSE, check_that_leaves_order_match = FALSE)
```

Arguments

- `dend_change`: tree object (dendrogram)
- `dend_template`: tree object (dendrogram)
- `dend_change_old_order`: a numeric vector with the order of leaves in dend_change (at least before it was changes for some reason). This is the vector based on which we adjust the new values of dend_change.
- `check_that_labels_match`: logical (FALSE). If to check that the labels in the two dendrogram match. (if they do not, the function aborts)
- `check_that_leaves_order_match`: logical (FALSE). If to check that the order in the two dendrogram match. (if they do not, the function aborts)

Value

Returns dend_change after adjusting its order values to be like dend_template.

See Also

- `entanglement`, `tanglegram`, `match_order_by_labels`

Examples

```r
## Not run:

dend <- USArrests[1:4,] %>% dist %>% hclust %>% as.dendrogram
order.dendrogram(dend) # c(4L, 3L, 1L, 2L)

# Watch this!
dend_changed <- dend
```
dend_changed <- rev(dend_changed)
expect_false(identical(order.dendrogram(dend_changed), order.dendrogram(dend)))
# we keep the order of dend_change, so that the leaves order are synced
# with their labels JUST LIKE dend:
old_dend_changed_order <- order.dendrogram(dend_changed)
# now we change dend_changed leaves order values:
order.dendrogram(dend_changed) <- 1:4
# and we can fix them again, based on their old kept leaves order:
dend_changed <- match_order_dendrogram_by_old_order(dend_changed, dend,
            old_dend_changed_order)
expect_identical(order.dendrogram(dend_changed), order.dendrogram(dend))

## End(Not run)

---

**na_locf**  
*Last Observation Carried Forward*

**Description**

A function for replacing each NA with the most recent non-NA prior to it.

**Usage**

na_locf(x, first_na_value = 0, recursive = TRUE, ...)

**Arguments**

- **x**  
  some vector
- **first_na_value**  
  If the first observation is NA, fill it with "first_na_value"
- **recursive**  
  logical (TRUE). Should na_locf be re-run until all NA values are filled?
- **...**  
  ignored.

**Value**

The original vector, but with all the missing values filled by the value before them.

**Source**


This could probably be solved MUCH faster using Rcpp.

**See Also**

na.locf
**Examples**

```r
na_locf(c(NA, NA))
na_locf(c(1, NA))
na_locf(c(1, NA, NA, NA))
na_locf(c(1, NA, NA, NA, 2, 2, NA, 3, NA, 4))
na_locf(c(1, NA, NA, NA, 2, 2, NA, 3, NA, 4), recursive = FALSE)
```

```r
# Not run:
library(microbenchmark)
library(zoo)

microbenchmark(
  na_locf = na_locf(c(1, NA, NA, NA, 2, 2, NA, 3, NA, 4)),
  na_locf = na_locf(c(1, NA, NA, NA, 2, 2, NA, 3, NA, 4))
) # my implementation is 6 times faster :)
```

```r
microbenchmark(
  na_locf = na_locf(rep(c(1, NA, NA, NA, 2, 2, NA, 3, NA, 4), 1000)),
  na.locf = na.locf(rep(c(1, NA, NA, NA, 2, 2, NA, 3, NA, 4), 1000))
) # my implementation is 3 times faster
```

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

---

**nleaves**

*Counts the number of leaves in a tree*

**Description**

Counts the number of leaves in a tree (dendrogram or hclust).

**Usage**

```r
nleaves(x, ...)
```

```r
# S3 method for class 'dendrogram'
nleaves(x, method = c("members", "order"), ...)
```

```r
# S3 method for class 'hclust'
nleaves(x, ...)
```

```r
# S3 method for class 'phylo'
nleaves(x, ...)
```

**Arguments**

- `x`: tree object (dendrogram/hclust/phylo)
nnodes

Counts the number of nodes (Vertices) in a tree

Description
Counts the number of nodes in a tree (dendrogram, hclust, phylo).

Usage

nnodes(x, ...)

## S3 method for class 'dendrogram'
nnodes(x, ...)

## S3 method for class 'hclust'
nnodes(x, ...)

## S3 method for class 'phylo'
nnodes(x, ...)
noded_with_condition  

Arguments

x  

...  

Details

The idea for the name is from functions like ncol, and nrow.

The phylo method is based on turning the phylo to hclust and than to dendrogram. It may not work for complex phylo trees.

Value

The number of leaves in the tree

See Also

ncol, count_terminal_nodes, nleaves

Examples

hc <- hclust(dist(USArrests[1:5,]), "ave")
dend <- as.dendrogram(hc)

nnodes(dend) # 9
nnodes(hc) # 9

noded_with_condition  

Find which nodes satisfies a condition

Description

Goes through a tree’s nodes in order to return a vector with whether (TRUE/FALSE) each node satisfies some condition (function)

Usage

noded_with_condition(dend, condition, include_leaves = TRUE, include_branches = TRUE, na.rm = FALSE, ...)

Arguments

dend  

a dendrogram dend

condition  

a function that gets a node and return TRUE or FALSE (based on whether or not that node/tree fulfills the "condition")

include_leaves  

logical. Should leaves attributes be included as well?

include_branches  

logical. Should non-leaf (branch node) attributes be included as well?
order.dendrogram <-

na.rm logical. Should NA attributes be REMOVED from the resulting vector?
...

passed to the condition function

Value

A logical vector with TRUE/FALSE, specifying for each of the dendrogram’s nodes if it fulfills the condition or not.

See Also

branches_attr_by_labels, get_leaves_attr, nnodes, nleaves

Examples

### Not run:

```r
library(dendextend)
set.seed(23235)
ss <- sample(1:150, 10 )
# Getting the dend dend
dend <- iris[ss,-5] %>% dist %>% hclust %>% as.dendrogram
dend %>% plot

this is the basis for branches_attr_by_labels
has_any_labels <- function(sub_dend, the_labels) any(labels(sub_dend) %in% the_labels)
cols <- noded_with_condition(dend, has_any_labels,
  the_labels = c("126", "109", "59") %>
ifelse(2,1)
set(dend, "branches_col", cols) %>% plot

# Similar to branches_attr_by_labels - but for heights!
high_enough <- function(sub_dend, height) attr(sub_dend, "height") > height
cols <- noded_with_condition(dend, high_enough, height = 1) %>% ifelse(2,1)
set(dend, "branches_col", cols) %>% plot

### End(Not run)
```

order.dendrogram <-

order.dendrogram <- assignment operator

Description

order.dendrogram <- assignment operator. This is useful in cases where some object is turned into a dendrogram but its leaves values (the order) are all mixed up.
**Usage**

```r
order.dendrogram(object, ...) <- value
```

**Arguments**

- `object`: a variable name (possibly quoted) whose label are to be updated
- `...`: parameters passed (not currently in use)
- `value`: a value to be assigned to object's leaves value (their "order")

**Value**

dendrogram with updated order leaves values

**See Also**

`order.dendrogram.labels<-`

**Examples**

```r
linik
#######
# Example for using the assignment with dendrogram and hclust objects:
> hc <- hclust(dist(USArrests[1:4,]), "ave")
> dend <- as.dendrogram(hc)

> str(dend)
> order.dendrogram(dend) # 4 3 1 2
> order.dendrogram(dend) <- 1:4
> order.dendrogram(dend) # 1 2 3 4
> str(dend) # the structure is still fine.

> # This function is very useful if we try playing with subtrees
> # For example:
> hc <- hclust(dist(USArrests[1:6,]), "ave")
> dend <- as.dendrogram(hc)
> sub_dend <- dend[[1]]
> order.dendrogram(sub_dend) # 4 6
> # now using as.hclust(sub_dend) will cause trouble:
> # labels(as.hclust(sub_dend)) # As of R 3.1.1-patched - this will produce an Error (as it should) :
> # let's fix it:

> order.dendrogram(sub_dend) <- rank(order.dendrogram(sub_dend), ties.method="first")
> labels(as.hclust(sub_dend)) # We now have labels :)
```
order.hclust  

Ordering of the Leaves in a hclust Dendrogram

Description

Ordering of the Leaves in a hclust Dendrogram

Usage

order.hclust(x, ...)

Arguments

x  
ab hclust object a distance matrix.

...  
Ignored.

Value

A vector with length equal to the number of leaves in the hclust dendrogram is returned. From 
r <- order.hclust(), each element is the index into the original data (from which the hclust was computed).

See Also

order.dendrogram

Examples

set.seed(23235)
ss <- sample(1:150, 10)
hc <- iris[ss,-5] %>% dist %>% hclust
# dend <- hc %>% as.dendrogram
order.hclust(hc)

partition_leaves  

A list with labels for each subtree (edge)

Description

Returns the set of all bipartitions from all edges, that is: a list with the labels for each of the nodes in the dendrogram.

Usage

partition_leaves(x, ...)

plot_horiz.dendrogram

Arguments

x          a dendrogram

...        Ignored.

Value

A list with the labels for each of the nodes in the dendrogram.

Source

A dendrogram implementation for partition.leaves from the distory package

See Also

distinct_edges, highlight_distinct_edges, dist.dendlist, tanglegram, partition.leaves

Examples

x <- 1:3 %>% dist %>% hclust %>% as.dendrogram
plot(x)
partition_leaves(x)

## Not run:
set.seed(23235)
ss <- sample(1:150, 10)
dend1 <- iris[ss,-5] %>% dist %>% hclust("com") %>% as.dendrogram
dend2 <- iris[ss,-5] %>% dist %>% hclust("single") %>% as.dendrogram

partition_leaves(dend1)
partition_leaves(dend2)

## End(Not run)

plot_horiz.dendrogram  Plotting a left-tip-adjusted horizontal dendrogram

Description

The default plot(dend, horiz = TRUE), gives us a dendrogram tree plot with the tips turned right. The current function enables the creation of the same tree, but with the tips turned left. The main challenge in doing this is finding the distance of the labels from the leaves tips - which is solved with this function.

Usage

plot_horiz.dendrogram(x, type = c("rectangle", "triangle"), center = FALSE,
edge.root = is.leaf(x) || !is.null(attr(x, "edgetext")), dLeaf = NULL,
horiz = TRUE, xaxt = "n", yaxt = "s", xlim = NULL, ylim = NULL,
nodePar = NULL, edgePar = list(), leaflab = c("perpendicular",
"textlike", "none"), side = TRUE, text_pos = 2, ...)


plot_horiz.dendrogram

Plotting a left-tip-adjusted horizontal dendrogram

Description

The default plot(dend, horiz = TRUE), gives us a dendrogram tree plot with the tips turned right. The current function enables the creation of the same tree, but with the tips turned left. The main challenge in doing this is finding the distance of the labels from the leaves tips - which is solved with this function.

Usage

plot_horiz.dendrogram(x, type = c("rectangle", "triangle"), center = FALSE,
edge.root = is.leaf(x) || !is.null(attr(x, "edgetext")), dLeaf = NULL,
horiz = TRUE, xaxt = "n", yaxt = "s", xlim = NULL, ylim = NULL,
nodePar = NULL, edgePar = list(), leaflab = c("perpendicular",
"textlike", "none"), side = TRUE, text_pos = 2, ...)

```r
x <- 1:3 %>% dist %>% hclust %>% as.dendrogram
plot(x)
partition_leaves(x)

## Not run:
set.seed(23235)
ss <- sample(1:150, 10)
dend1 <- iris[ss,-5] %>% dist %>% hclust("com") %>% as.dendrogram
dend2 <- iris[ss,-5] %>% dist %>% hclust("single") %>% as.dendrogram

partition_leaves(dend1)
partition_leaves(dend2)

## End(Not run)
```
Arguments

- **x**: tree object (dendrogram)
- **type**: a character vector with either "rectangle" or "triangle" (passed to `plot.dendrogram`)
- **center**: logical; if TRUE, nodes are plotted centered with respect to the leaves in the branch. Otherwise (default), plot them in the middle of all direct child nodes.
- **edge.root**: logical; if true, draw an edge to the root node.
- **dLeaf**: a number specifying the distance in user coordinates between the tip of a leaf and its label. If NULL as per default, 3/4 of a letter width is used.
- **horiz**: logical indicating if the dendrogram should be drawn horizontally or not. In this function it MUST be TRUE!
- **xaxt**: graphical parameters, or arguments for other methods.
- **yaxt**: graphical parameters, or arguments for other methods.
- **xlim**: (NULL) optional x- and y-limits of the plot, passed to `plot.default`. The defaults for these show the full dendrogram.
- **ylim**: (NULL) optional x- and y-limits of the plot, passed to `plot.default`. The defaults for these show the full dendrogram.
- **nodePar**: NULL.
- **edgePar**: list()
- **leaflab**: c("perpendicular", "textlike", "none")
- **side**: logical (TRUE). Should the tips of the drawn tree be facing the left side. This is the important feature of this function.
- **text_pos**: integer from either 1 to 4 (2). Two relevant values are 2 and 4. 2 (default) means that the labels are aligned to the tips of the tree leaves. 4 will have the labels align to the left, making them look like they were when the tree was on the left side (with leaves tips facing to the right).
- **...**: passed to `plot`.

Value

The invisible dLeaf value.

Source

This function is based on replicating `plot.dendrogram`. In fact, I'd be happy if in the future, some tweaks could be made to `plot.dendrogram`, so that it would replace the need for this function.

See Also

`plot.dendrogram`, `tanglegram`
prune

Examples

```r
## Not run:
dend <- USArrests[1:10,] #> dist #> hclust #> as.dendrogram

par(mfrow = c(1,2), mar = rep(6,4))
plot_horiz.dendrogram(dend, side=FALSE)
plot_horiz.dendrogram(dend, side=TRUE)
# plot_horiz.dendrogram(dend, side=TRUE, dLeaf= 0)
# plot_horiz.dendrogram(dend, side=TRUE, nodePar = list(pos = 1))
# sadly, lab.pos is not implemented yet,
## so the labels can not be right aligned...

plot_horiz.dendrogram(dend, side=FALSE)
plot_horiz.dendrogram(dend, side=TRUE, dLeaf= 0, xlim = c(100,-10)) # bad
plot_horiz.dendrogram(dend, side=TRUE, text_offset = 0)
plot_horiz.dendrogram(dend, side=TRUE, text_offset = 0, text_pos=4)

## End(Not run)
```

---

**prune**

Prunes a tree (using leaves' labels)

**Description**

Trims a tree (dendrogram, hclust) from a set of leaves based on their labels.

**Usage**

```r
prune(x, ...)
```

## S3 method for class 'dendrogram'
```r
prune(x, leaves,...)
```

## S3 method for class 'hclust'
```r
prune(x, leaves,...)
```

## S3 method for class 'phylo'
```r
prune(x, ...)
```

**Arguments**

- `x` tree object (dendrogram/hclust/phylo)
- ... passed on
- `leaves` a character vector of the label(S) of the tip(s) (leaves) we wish to prune off the tree.
prune_leaf

Details

I was not sure if to call this function drop.tip (from ape), snip/prune (from rpart) or just remove.leaves. I ended up deciding on prune.

Value

A pruned tree

See Also

prune_leaf, drop.tip ape

Examples

hc <- hclust(dist(USArrests[1:5,]), "ave")
dend <- as.dendrogram(hc)

par(mfrow = c(1,2))
plot(dend, main = "original tree")
plot(prune(dend, c("Alaska", "California")), main = "tree without Alaska and California")

---

prune_leaf 

Trimms one leaf from a dendrogram

Description

Trimms one leaf from a dendrogram

Usage

prune_leaf(x, leaf_name, ...)

Arguments

x dendrogram object
leaf_name a character string as the label of the tip we wish to prune
... passed on

Details

Used through prune

Value

A dendrogram with a leaf pruned
Examples

```r
hc <- hclust(dist(USArrests[1:5,]), "ave")
dend <- as.dendrogram(hc)

par(mfrow = c(1,2))
plot(dend, main = "original tree")
plot(prune_leaf(dend, "Alaska"), main = "tree without Alaska")
```

Description

Shows the significant branches in a dendrogram, based on a pvclust object.

Usage

```r
pvclust_show_signif(dend, pvclust_obj, signif_type = c("bp", "au"),
alpha = 0.05, signif_value = c(5, 1), show_type = c("lwd", "col"), ...)
```

Arguments

- `dend`: a dendrogram object
- `pvclust_obj`: a pvclust object
- `signif_type`: a character scalar (either "bp" or "au"), indicating which of the two should be used to update the dendrogram.
- `alpha`: a number between 0 to 1, default is .05. Indicates what is the cutoff from which branches will be updated.
- `signif_value`: a 2d vector (default: c(5,1)), with the first element tells us what the significant branches will get, and the second element which value the non-significant branches will get.
- `show_type`: a character scalar (either "lwd" or "col"), indicating which parameter of the branches should be updated based on significance.
- `...`: not used

Value

A dendrogram with updated branches

See Also

`pvclust_show_signif, pvclust_show_signif_gradient`
pvclust_show_signif_gradient

Shows the gradient of significance of branches in a dendrogram, based on a pvclust object

Description

Shows the gradient of significance of branches in a dendrogram, based on a pvclust object

Usage

`pvclust_show_signif_gradient(dend, pvclust_obj, signif_type = c("bp", "au"), signif_col_fun = colorRampPalette(c("black", "darkred", "red")), ...)`

Arguments

dend a dendrogram object

pvclust_obj a pvclust object
signif_type a character scalar (either "bp" or "au"), indicating which of the two should be used to update the dendrogram.

signif_col_fun a function to create colors for the significant gradient. Default is: colorRampPalette(c("black", "darkred", "red"))

... not used

Value

A dendrogram with updated branches

See Also

pvclust_show_signif, pvclust_show_signif_gradient

Examples

```r
## Not run:
library(pvclust)
data(lung) # 916 genes for 73 subjects
set.seed(13134)
result <- pvclust(lung[, 1:20], method.dist="cor", method.hclust="average", nboot=100)
dend <- as.dendrogram(result)
result %>% as.dendrogram %>%
  hang.dendrogram %>%
  plot(main = "Cluster dendrogram with AU/BP values (%)")
result %>% text
result %>% pvrect(alpha=0.95)
dend %>% pvclust_show_signif(result) %>% plot
dend %>% pvclust_show_signif(result, show_type = "1wd") %>% plot
result %>% text
result %>% pvrect(alpha=0.95)
dend %>% pvclust_show_signif_gradient(result) %>% plot

dend %>
  pvclust_show_signif_gradient(result) %>
  pvclust_show_signif(result) %>
  plot(main = "Cluster dendrogram with AU/BP values (%)
  bp values are highlighted by signif")
result %>% text
result %>% pvrect(alpha=0.95)

## End(Not run)
```
### rank_branches

**R**ank branches' heights

#### Description

Adjust the height attr in all of the dendrogram nodes so that the tree will have a distance of 1 unit between each parent/child nodes. It can be thought of as ranking the branches between themselves.

This is intended for easier comparison of the topology of two trees.

Notice that this function changes the height of all the leaves into 0, thus erasing the effect of `hang.dendrogram` (which should be run again, if that is the visualization you are interested in).

#### Usage

```r
rank_branches(dend, diff_height = 1, ...)  
```

#### Examples

```r
# Load data
hc <- hclust(dist(USArrests[2:9,]), "com")
dend <- as.dendrogram(hc)

# Plot dendrogram
par(mfrow = c(1,2))
plot(dend, main = "original tree")
plot(rank_branches(dend, 100), main = "Raised tree")
```
Arguments

dend       a dendrogram object
diff_height Numeric scalar (1). Affects the difference in height between two branches.
...       not used

Value

A dendrogram, after adjusting the height attr in all of its branches.

See Also

generic_branches_heights, get_childrens_heights, hang.dendrogram, tanglegram

Examples

# define dendrogram object to play with:
dend <- USArrests[1:5,] %>% dist %>% hclust %>% as.dendrogram

par(mfrow = c(1,3))

plot(dend)
plot(rank_branches(dend))
plot(hang.dendrogram(rank_branches(dend)))

rank_order.dendrogram  Fix rank of leaves order values in a dendrogram

Description

Generally, leaves order value should be a sequence of integer values. From 1 to nleaves(tree). This function fixes trees by using rank on existing leaves order values.

Usage

rank_order.dendrogram(dend, ...)

Arguments

dend       a dendrogram object
...       not used

Value

A dendrogram, after fixing its leaves order values.

See Also

prune
Examples

```r
# define dendrogram object to play with:
dend <- USArrests[1:4,]  # >% dist %>% hclust(method = "ave") %>% as.dendrogram
# plot(dend)
order.dendrogram(dend)
dend2 <- prune(dend, "Alaska")
order.dendrogram(dend2)
order.dendrogram(rank_order.dendrogram(dend2))
```

**rect.dendrogram**

*Draw Rectangles Around a Dendrogram's Clusters*

**Description**

Draws rectangles around the branches of a dendrogram highlighting the corresponding clusters. First the dendrogram is cut at a certain level, then a rectangle is drawn around selected branches.

**Usage**

```r
rect.dendrogram(tree, k = NULL, which = NULL, x = NULL, h = NULL,
border = 2, cluster = NULL, horiz = FALSE, density = NULL,
angle = 45, text = NULL, text_cex = 1, text_col = 1, xpd = TRUE,
lower_rect, ...)
```

**Arguments**

- **tree**: a *dendrogram* object.
- **k**: Scalar. Cut the dendrogram such that exactly k clusters (if possible) are produced.
- **which**: A vector selecting the clusters around which a rectangle should be drawn. which selects clusters by number (from left to right in the tree), Default is which = 1:k.
- **x**: A vector selecting the clusters around which a rectangle should be drawn. x selects clusters containing the respective horizontal coordinates.
- **h**: Scalar. Cut the dendrogram by cutting at height h. (k overrides h)
- **border**: Vector with border colors for the rectangles.
- **cluster**: Optional vector with cluster memberships as returned by cutree(dend_obj, k = k), can be specified for efficiency if already computed.
- **horiz**: logical (FALSE), indicating if the rectangles should be drawn horizontally or not (for when using plot(dend, horiz = TRUE) ).
- **density**: Passed to *rect*: the density of shading lines, in lines per inch. The default value of NULL means that no shading lines are drawn. A zero value of density means no shading lines whereas negative values (and NA) suppress shading (and so allow color filling). If border is a vector of colors, the color of density will default to 1.
angle  Passed to `rect`: angle (in degrees) of the shading lines. (default is 45)
text    a character vector of labels to plot underneath the clusters. When NULL (default), no text is displayed.
text_cex a numeric (scalar) value of the text’s cex value.
text_col a (scalar) value of the text’s col(or) value.
xpd     A logical value (or NA.), passed to `par`. Default is TRUE, in order to allow the rect to be below the labels. If FALSE, all plotting is clipped to the plot region, if TRUE, all plotting is clipped to the figure region, and if NA, all plotting is clipped to the device region. See also `clip`.
lower_rect a (scalar) value of how low should the lower part of the rect be. If missing, it will take the value of `par("usr")[3L]` (or `par("usr")[2L]`, depending if horiz = TRUE or not), with also the width of the labels. (notice that we would like to keep xpd = TRUE if we want the rect to be after the labels!) You can use a value such as 0, to get the rect above the labels.
Notice that for a plot with small margins, it would be better to set this parameter manually.

... parameters passed to `rect` (such as lwd, lty, etc.)

Value

(Invisibly) returns a list where each element contains a vector of data points contained in the respective cluster.

Source

This function is based on `rect.hclust`, with slight modifications to have it work with a dendrogram, as well as a few added features (e.g: ... to rect, and horiz)

The idea of adding text and shading lines under the clusters comes from skullkey from here: http://stackoverflow.com/questions/4720307/change-dendrogram-leaves

See Also

`rect.hclust`, `order.dendrogram`, `cutree.dendrogram`

Examples

```r
set.seed(23235)
ss <- sample(1:150, 10 )
hc <- iris[ss,-5] %>% dist %>% hclust
dend <- hc %>% as.dendrogram

plot(dend)
rect.dendrogram(dend,2, border = 2)

plot(dend)
rect.dendrogram(dend, 3 , border = 1:3,
    density = 2, text = c("1", "b", "miao"), text_cex = 3)
```
plot(dend)
rect.dendrogram(dend, 4, which = c(1, 3), border = c(2, 3))
rect.dendrogram(dend, 4, x = 5, border = c(4))
rect.dendrogram(dend, 3, border = 3, lwd = 2, lty = 2)
# now THIS, you can not do with the old rect.hclust
plot(dend, horiz = TRUE)
rect.dendrogram(dend, 2, border = 2, horiz = TRUE)
rect.dendrogram(dend, 4, border = 4, lty = 2, lwd = 3, horiz = TRUE)

remove_branches_edgePar

Remove all edgePar values from a dendrogram's branches

Description

Go through the dendrogram branches and remove its edgePar.

Usage

remove_branches_edgePar(object, ...)

Arguments

  object           a dendrogram object
  ...             not used

Value

A dendrogram, after removing the edgePar attribute in all of its branches.

See Also

  get_root_branches_attr, assign_values_to_branches_edgePar

Examples

## Not run:

dend <- USArrests[1:5, ] %>% dist %>% hclust %>% as.dendrogram
dend <- color_branches(dend, 3)
par(mfrow = c(1, 2))
plot(dend)
plot(dend)
plot(remove_branches_edgePar(dend))

## End(Not run)
remove_leaves_nodePar

Remove all nodePar values from a dendrogram’s leaves

Description

Go through the dendrogram leaves and remove its nodePar.

Usage

remove_leaves_nodePar(object, ...)

Arguments

object a dendrogram object
... not used

Value

A dendrogram, after removing the nodePar attribute in all of its leaves,

See Also

get_leaves_attr, assign_values_to_leaves_nodePar

Examples

## Not run:

dend <- USArrests[1:5,] %>% dist %>% hclust %>% as.dendrogram

dend <- color_labels(dend, 3)
par(mfrow = c(1,2))
plot(dend)
plot(remove_leaves_nodePar(dend))

get_leaves_attr(dend, "nodePar")
get_leaves_attr(remove_leaves_nodePar(dend), "nodePar")

## End(Not run)
remove_nodes_nodePar  Remove all nodePar values from a dendrogram’s nodes

Description

Go through the dendrogram nodes and remove its nodePar

Usage

remove_nodes_nodePar(object, ...)

Arguments

  object   a dendrogram object
  ...      not used

Value

A dendrogram, after removing the nodePar attribute in all of its nodes,

See Also

generate_branches_attr, assign_values_to_branches_edgePar

Examples

## Not run:

dend <- USArrests[1:5, ] %>% dist %>% hclust %>% as.dendrogram
dend <- color_branches(dend, 3)
par(mfrow = c(1, 2))
plot(dend)
plot(remove_branches_edgePar(dend))

## End(Not run)

rllply  recursively apply a function on a list

Description

recursively apply a function on a list - and returns the output as a list, following the naming convention in the plyr package the big difference between this and rapply is that this will also apply the function on EACH element of the list, even if it’s not a "terminal node" inside the list tree. An attribute is added to indicate if the value returned is from a branch or a leaf.
Usage

```r
rllply(x, FUN, add_notation = FALSE, ...)
```

Arguments

- **x**: a list.
- **FUN**: a function to apply on each element of the list.
- **add_notation**: logical. Should each node be added a "position_type" attribute, stating if it is a "Branch" or a "Leaf".
- **...**: not used.

Value

A list with ALL of the nodes (from the original "x" list), that FUN was applied on.

Examples

```r
## Not run:
x <- list()
x
rllply(x, function(x)(x), add_notation = TRUE)

x <- list(1, 2, list(3))
x
rllply(x, function(x)(x), add_notation = TRUE)
  # the first element is the entire tree
  # after FUN was applied to its root element.

hc <- hclust(dist(USArrests[1:4,]), "ave")
dend <- as.dendrogram(hc)
rllply(dend, function(x)(attr(x, "height"))
rllply(dend, function(x)(attr(x, "members")))
```

---

**rotate**

*Rotate a tree object*

**Description**

Rotates, rev and sort the branches of a tree object (dendrogram, hclust) based on a vector - either of labels order (numbers) or the labels in their new order (character).
Usage

rotate(x, ...)

## S3 method for class 'dendrogram'
rotate(x, order, ...)

## S3 method for class 'hclust'
rotate(x, order, ...)

## S3 method for class 'phylo'
rotate(x, ..., phy)

## S3 method for class 'hclust'
rev(x, ...)

## S3 method for class 'dendrogram'
sort(x, decreasing=FALSE, type = c("labels", "nodes"), ...)

## S3 method for class 'hclust'
sort(x, decreasing=FALSE, ...)

Arguments

x a tree object (either a dendrogram or hclust)

... parameters passed (for example, in case of sort)

order Either numeric or character vector. Is numeric: it is a numeric vector with the order of the value to be assigned to object’s label. The numbers say are just like when you use order: which of the items on the tree-plot should be "first" (e.g: most left), second etc. (this is relevant only to rotate) Is character: it must be a vector with the content of labels(x), in the order we’d like to have the new tree.

decreasing logical. Should the sort be increasing or decreasing? Not available for partial sorting. (relevant only to sort)

type a character indicating how to sort. If "labels" then by lexicographic order of the labels. If "nodes", then by using ladderize (order so that recursively, the leftmost branch will be the smallest)

phy a placeholder in case the user uses "phy ="

Details

The motivation for this function came from the function order.dendrogram NOT being very intuitive. What rotate aims to do is give a simple tree rotation function which is based on the order which the user would like to see the tree rotated by (just as order works for numeric vectors).

ev.dendrogram is part of base R, and returns the tree object after rotating it so that the order of the labels is reversed. Here we added an S3 method for hclust objects.

The sort methods sort the labels of the tree (using order) and then attempts to rotate the tree to fit that order.
The hclust method of "rotate" works by first changing the object into dendrogram, performing the rotation, and then changing it back to hclust. Special care is taken in preserving some of the properties of the hclust object.

The ape package has its own rotate(ape) function (Which is sadly not S3, so cannot be easily connected with the current implementation). Still, there is an S3 plug that makes sure people loading first ape and then dendextend will still be able to use rotate without a problem. Notice that if you will first load ape and only then dendextend, using "rotate" will fail with the error: "Error in rotate(dend, ____): object "phy" is not of class "phylo"" - this is because rotate in ape is not S3 and will fail to find the rotate.dendrogram function. In such a case simply run unloadNamespace(ape). Or, you can run: unloadNamespace("dendextend"); attachNamespace("dendextend") The solution for this is that if you have ape installed on your machine, It will be loaded when you load dendextend (but after it). This way, rotate will work fine for both dendrogram AND phylo objects.

Value
A rotated tree object

See Also
order.dendrogram, order, rev.dendrogram, rotate (ape), ladderize

Examples
hc <- hclust(dist(USArrests[,c(1,6,13,20,23),]), "ave")
dend <- as.dendrogram(hc)

# For dendrogram objects:
labels_colors(dend) <- rainbow(nleaves(dend))
# let's color the labels to make the followup of the rotation easier
par(mfrow = c(1,2))
plot(dend, main = "Original tree")
plot(rotate(dend, c(2:5,1)), main = "Rotates the left most leaf \n into the right side of the tree")
par(mfrow = c(1,2))
plot(dend, main = "Original tree")
plot(sort(dend), main = "Sorts the labels by alphabetical order \n and rotates the tree to give the best fit possible")
par(mfrow = c(1,2))
plot(dend, main = "Original tree")
plot(rev(dend), main = "Reverses the order of the tree labels")

# For hclust objects:
plot(hc)
plot(rotate(hc, c(2:5,1)), main = "Rotates the left most leaf \n into the right side of the tree")

par(mfrow = c(1,3))
dend %>% plot(main = "Original tree")
dend %>% sort %>% plot(main = "labels sort")
dend %>% sort(type = "nodes") %>% plot(main = "nodes (ladderize) sort")
rotate_DendSer

Rotates dend based on DendSer

Description

Rotates a dendrogram based on its seriation

The function tries to turn the dend into hclust using DendSer.dendrogram (based on DendSer)
Also, if a distance matrix is missing, it will try to use the cophenetic distance.

Usage

rotate_DendSer(dend, ser_weight, ...)

Arguments

dend
An object of class dendrogram

ser_weight
Used by cost function to evaluate ordering. For cost=costLS, this is a vector of object weights. Otherwise is a dist or symmetric matrix. passed to DendSer.dendrogram and from there to DendSer.
If it is missing, the cophenetic distance is used instead.

...
parameters passed to DendSer

Value

Numeric vector giving an optimal dendrogram order

See Also

DendSer, DendSer.dendrogram, untangle_DendSer, rotate_DendSer

Examples

## Not run:
library(DendSer) # already used from within the function
dend <- USArrests[1:4,] %>% dist %>% hclust("ave") %>% as.dendrogram
DendSer.dendrogram(dend)

tanglegram(dend, rotate_DendSer(dend))

## End(Not run)
Sample a tree

Description

Samples a tree, either by permuting the labels (which is useful for a permutation test), or by repeated sampling of the same labels (essential for bootstrapping when we don’t have access to the original data which produced the tree).

Duplicates a leaf in a tree. Useful for non-parametric bootstrapping trees since it emulates what would have happened if the tree was constructed based on a row-sample with replacements from the original data matrix.

Usage

```r
sample.dendrogram(dend, replace = FALSE, dend_labels, sampled_labels, 
  fix_members = TRUE, fix_order = TRUE, fix_midpoint = TRUE, ...)
```

Arguments

- **dend**: a dendrogram object
- **replace**: logical (FALSE). Should we shuffle the labels (if FALSE), or should we replicate the same leaf over and over, while omitting other leaves? (this is when set to TRUE).
- **dend_labels**: a character vector of the tree’s labels. This can save the time it takes for getting the tree labels (in case we run a simulating, computing this once might save some running time). If missing, it uses labels in order to get the labels.
- **sampled_labels**: a character vector of the tree’s sampled labels. This can help us if we wish to compare two trees. In such a case we’d like to be able to have the same sample of labels used on both trees. If missing, it uses sample in order to get the sampled labels.
  Only works when replace=TRUE!
- **fix_members**: logical (TRUE). Fix the number of members in attr using `fix_members_attr.dendrogram`
- **fix_order**: logical (TRUE). Fix the leaves order
- **fix_midpoint**: logical (TRUE). Fix the midpoint value. If TRUE, it overrides "fix_members" and turns it into TRUE (since it must have a correct number of members in order to work). values using `rank_order.dendrogram`
- **...**: not used

Value

A dendrogram, after "sampling" its leaves.

See Also

- `sample`
- `duplicate_leaf`
Examples

```r
## Not run:
# define dendrogram object to play with:
dend <- USArrests[,1:5] %>% dist %>% hclust(method = "ave") %>% as.dendrogram
plot(dend)

## same tree, with different order of labels
plot(sample.dendrogram(dend, replace = FALSE))

## A different tree (!), with some labels duplicated,
## while others are pruned
plot(sample.dendrogram(dend, replace = TRUE))

## End(Not run)
```

---

**set**

Set (/update) features to a dendrogram

**Description**

A master function for updating various attributes and features of dendrogram objects.

**Usage**

```r
set(object, ...)
```

## S3 method for class 'dendrogram'

```r
set(object,  
  what = c("labels",  
    "labels_colors",  
    "labels_cex",  
    "labels_to_character",  
    "leaves_pch",  
    "leaves_cex",  
    "leaves_col",  
    "nodes_pch",  
    "nodes_cex",  
    "nodes_col",  
    "hang_leaves",  
    "branches_k_color",  
    "branches_col",  
    "branches_lwd",  
    "branches_lty",  
    "by_labels_branches_col",  
    "by_labels_branches_lwd",  
    "by_labels_branches_lty",  
   "clear_branches",
  
```

"clear_leaves"
)

## S3 method for class 'dendlist'
set(object, ..., which)

## S3 method for class 'data.table'
set(...)

### Arguments

- **object**: a tree (dendrogram, or dendlist)
- **...**: passed to the specific function for more options.
- **what**: a character indicating what is the property of the tree that should be set/updated. (see the usage and the example section for the different options)
- **value**: an object with the value to set in the tree. (the type of the value depends on the "what")
- **which**: an integer vector indicating, in the case "object" is a dendlist, on which of the trees should the modification be performed. If missing - the change will be performed on all of objects in the dendlist.

### Details

This is a wrapper function for many of the main tasks we might wish to perform on a dendrogram before plotting.

The options of by_labels_branches_col, by_labels_branches_lwd, by_labels_branches_lty have extra parameters: type, attr, TF_value. You can read more about them here: branches_attr_by_labels

The "what" parameter” can accept the following options:

- labels - set the labels (labels<-.dendrogram)
- labels_colors - set the labels’ colors (color_labels)
- labels_cex - set the labels’ size (assign_values_to_leaves_nodePar)
- labels_to_character - set the labels’ to be characters
- leaves_pch - set the leaves’ point type (assign_values_to_leaves_nodePar)
- leaves_cex - set the leaves’ point size (assign_values_to_leaves_nodePar)
- leaves_col - set the leaves’ point color (assign_values_to_leaves_nodePar)
- nodes_pch - set the nodes’ point type (assign_values_to_nodes_nodePar)
- nodes_cex - set the nodes’ point size (assign_values_to_nodes_nodePar)
- nodes_col - set the nodes’ point color (assign_values_to_nodes_nodePar)
- hang_leaves - hang the leaves (hang.dendrogram)
- branches_k_color - color the branches (color_branches)
- branches_col - set the color of branches (assign_values_to_branches_edgePar)
set

- branches_lwd - set the line width of branches (assign_values_to_branches_edgePar)
- branches_lty - set the line type of branches (assign_values_to_branches_edgePar)
- by_labels_branches_col - set the color of branches with specific labels (branches_attr_by_labels)
- by_labels_branches_lwd - set the line width of branches with specific labels (branches_attr_by_labels)
- by_labels_branches_lty - set the line type of branches with specific labels (branches_attr_by_labels)
- clear_branches - clear branches’ attributes (remove_branches_edgePar)
- clear_leaves - clear leaves’ attributes (remove_branches_edgePar)

Value

An updated dendrogram (or dendlist), with some change to the parameters of it

See Also

labels<-.dendrogram, labels_colors<-, hang.dendrogram, color_branches, assign_values_to_leaves_nodePar, assign_values_to_branches_edgePar, remove_branches_edgePar, remove_leaves_nodePar, noded_with_condition, branches_attr_by_labels, dendrogram

Examples

## Not run:

set.seed(23235)
sample <- sample(1:150, 10)

# Getting the dend object
dend <- iris[sample, 5]

# set labels

dend %>% set("labels", 1:10) %>% plot

dend %>% set("labels", 1:10) %>% plot

dend %>% set("labels_color") %>% plot

dend %>% set("labels_col", c(1,2)) %>% plot

# set labels' color

dend %>% set("labels_cex", c(1, 1.2)) %>% plot

dend %>% set("leaves_pch", c(1, 2)) %>% plot

dend %>% set("leaves_pch", c(1, 5)) %>% plot

dend %>% set("leaves_pch", c(19, 19, NA)) %>%

# set hang

dend %>% set("hang") %>% plot

# set branches_k_col

dend %>% set("branches_k_col") %>% plot

dend %>% set("branches_k_col", c(1, 2)) %>% plot

dend %>% set("branches_k_col", c(1, 2, 3), k=3) %>% plot

dend %>% set("branches_k_col", k=3) %>% plot
dend %>% set("branches_col", c(1, 2, 1, 2, NA)) %>% plot
dend %>% set("branches_lwd", c(2, 1, 2)) %>% plot
dend %>% set("branches_lty", c(1, 2, 1)) %>% plot

# clears all of the things added to the leaves
dend %>%
  set("labels_color", c(19, 19, NA)) %>%
  set("leaves_pch", c(19, 19, NA)) %>% # plot
  set("clear_leaves") %>% # remove all of what was done until this point
plot

# Different order
dend %>%
  set("leaves_pch", c(19, 19, NA)) %>%
  set("labels_color", c(19, 19, NA)) %>%
  set("clear_leaves") %>% plot

# doing this without chaining (%>% will NOT be fun:
dend %>%
  set("labels", 1:10) %>%
  set("labels_color") %>%
  set("branches_col", c(1, 2, 1, 2, NA)) %>%
  set("branches_lwd", c(2, 1, 2)) %>%
  set("branches_lty", c(1, 2, 1)) %>%
  set("hang") %>%
plot

#------------------------
# Examples for: by_labels_branches_col, by_labels_branches_lwd, by_labels_branches_lty

old_labels <- labels(dend)
dend %>%
  set("labels", seq_len(nleaves(dend))) %>%
  set("by_labels_branches_col", c(1:4, 7)) %>%
  set("by_labels_branches_col", c(1:4, 7)) %>%
  set("by_labels_branches_lwd", c(1:4, 7)) %>%
  set("by_labels_branches_lty", c(1:4, 7)) %>%
  set("labels", old_labels) %>%
plot

dend %>%
  set("labels", seq_len(nleaves(dend))) %>%
  set("by_labels_branches_col", c(1:4, 7), type = "any", TF_values = c(4, 2)) %>%
  set("by_labels_branches_lwd", c(1:4, 7), type = "all", TF_values = c(4, 1)) %>%
  set("by_labels_branches_lty", c(1:4, 7), TF_values = c(4, 1)) %>%
plot

#------------------------
# A few dendlist examples:
dendlist(dend, dend) %>% set("hang") %>% plot
shuffle

Random rotation of trees

Description

'shuffle' randomly rotates ("shuffles") a tree, changing its presentation while preserving its topology. 'shuffle' is based on rotate and through its methods can work for any of the major tree objects in R (dendrogram/hclust/phylo).

This function is useful in combination with tanglegram and entanglement.
Usage

shuffle(object, ...)

## S3 method for class 'dendrogram'
shuffle(object, ...)

## S3 method for class 'dendlist'
shuffle(object, which, ...)

## S3 method for class 'hclust'
shuffle(object, ...)

## S3 method for class 'phylo'
shuffle(object, ...)

Arguments

object
  a tree object (dendrogram/hclust/phylo)

...  
  Ignored.

which
  an integer vector for indicating which of the trees in the dendlist object should be
  plotted default is missing, in which case all the dends in dendlist will be shuffled

Details

'shuffle' is a function that randomly rotates ("shuffles") a tree. a dendrogram leaves order (by
means of rotation)

Value

A randomly rotated tree object

See Also

tanglegram, entanglement, rotate

Examples

dend <- USArrests %>% dist %>% hclust %>% as.dendrogram
set.seed(234238)
dend2 <- shuffle(dend)
tanglegram(dend, dend2, margin_inner=7)
entanglement(dend, dend2) # 0.3983

# although these ARE the SAME tree:
tanglegram(sort(dend), sort(dend2), margin_inner=7)
sort_2_clusters_vectors

Sorts two clusters vector by their names

Description
Sorts two clusters vector by their names

Usage

sort_2_clusters_vectors(A1_clusters, A2_clusters,
assume_sorted_vectors = FALSE, warn = dendextend_options("warn"), ...)

Arguments

A1_clusters a numeric vector of cluster grouping (numeric) of items, with a name attribute of item name for each element from group A1. These are often obtained by using some k cut on a dendrogram.

A2_clusters a numeric vector of cluster grouping (numeric) of items, with a name attribute of item name for each element from group A2. These are often obtained by using some k cut on a dendrogram.

assume_sorted_vectors logical (FALSE). Can we assume to two group vectors are sorter so that they have the same order of items? IF FALSE (default), then the vectors will be sorted based on their name attribute.

warn logical (default from dendextend_options("warn") is FALSE). Set if warning are to be issued, it is safer to keep this at TRUE, but for keeping the noise down, the default is FALSE.

... Ignored.

Value
A list with two elements, correponding two the two clustering vectors.

See Also

FM_index_profdpm

Examples

## Not run:

set.seed(23235)
ss <- sample(1:150, 4)
hc1 <- hclust(dist(iris[ss,-5]), "com")
hc2 <- hclust(dist(iris[ss,-5]), "single")
# dend1 <- as.dendrogram(hc1)
sort_dist_mat

```
# dend2 <- as.dendrogram(hc2)
# cutree(dend1)

A1_clusters <- cutree(hc1, k=3)
A2_clusters <- sample(cutree(hc1, k=3))

sort_2_clusters_vectors(A1_clusters, A2_clusters, assume_sorted_vectors =TRUE) # no sorting
sort_2_clusters_vectors(A1_clusters, A2_clusters, assume_sorted_vectors =FALSE) # Sorted

## End(Not run)
```

---

**sort_dist_mat**

Sorts a distance matrix by rows and columns

**Description**

Sorts a distance matrix by rows and columns

**Usage**

```
sort_dist_mat(dist_mat, by_rows = TRUE, by_cols = TRUE, ...)
```

**Arguments**

- `dist_mat`: a distance matrix.
- `by_rows`: logical (TRUE). Sort the distance matrix by rows?
- `by_cols`: logical (TRUE). Sort the distance matrix by columns?
- `...`: Ignored.

**Value**

A distance matrix (after sorting)

**See Also**

- `dist`, `cor_cophenet`
sort_levels_values  

Sort the values level in a vector

Description

Takes a numeric vector and sort its values so that they would be increasing from left to right. It is different from `sort` in that the function will only "sort" the values levels, and not the vector itself.

This function is useful for `cutree` - making the sort_cluster_numbers parameter possible. Using that parameter with TRUE makes the clusters id’s from cutree to be ordered from left to right. e.g: the left most cluster in the tree will be numbered "1", the one after it will be "2" etc...).

Usage

```r
sort_levels_values(x, MARGIN = 2, decreasing = FALSE,
force_integer = FALSE, warn = dendextend_options("warn"), ...)
```

Arguments

- `x`  
a numeric vector.

- `MARGIN`  
passed to `apply`. It is a vector giving the subscripts which the function will be applied over. E.g., for a matrix 1 indicates rows, 2 indicates columns, c(1, 2) indicates rows and columns. Where X has named dimnames, it can be a character vector selecting dimension names.

- `decreasing`  
logical (FALSE). Should the sort be increasing or decreasing?

- `force_integer`  
logical (FALSE). Should the values returned be integers?

- `warn`  
logical (default from dendextend_options("warn") is FALSE). Set if warning are to be issued, it is safer to keep this at TRUE, but for keeping the noise down, the default is FALSE. (for example when x had NA values in it)

- `...`  
ignored.

Value

if `x` is an object - it returns logical - is the object of class dendrogram.

See Also

`sort`, `fac2num`, `cutree`

Examples

```r
x <- 1:4
sort_levels_values(x) # 1 2 3 4

x <- c(4:1)
names(x) <- letters[x]
attr(x, "keep_me") <- "a cat"
```
tanglegram

```r
sort_levels_values(x) # 1 2 3 4
x <- c(4:1, 4, 2)
sort_levels_values(x) # 1 2 3 4 1 3
x <- c(2, 2, 3, 2, 1)
sort_levels_values(x) # 1 1 2 1 3
x <- matrix(16:1, 4, 4)
rownames(x) <- letters[1:4]
x
apply(x, 2, sort_levels_values)
```

description

Plots a tanglegram plot of a side by side trees.

Usage

tanglegram(tree1, ...)

# S3 method for class 'dendrogram'
tanglegram(tree1, tree2, sort = FALSE, color_lines = "darkgrey", lwd = 3.5, edge.lwd = NULL, columns_width = c(5, 3, 5), margin_top = 3, margin_bottom = 2.5, margin_inner = 3, margin_outer = 0.5, left_dendo_mar = c(margin_bottom, margin_outer, margin_top, margin_inner), right_dendo_mar = c(margin_bottom, margin_inner, margin_top, margin_outer), intersecting = TRUE, dLeaf = NULL, dLeaf_left = dLeaf, dLeaf_right = dLeaf, axes = TRUE, type = "r", # can also be "t" lab.cex = NULL, remove_nodePar = FALSE, main = "", main_left = "", main_right = ",
```
sub = "", k_labels = NULL, k_branches = NULL, rank_branches = FALSE, hang = FALSE, match_order_by_labels = TRUE, cex_main = 2, cex_main_left = cex_main, cex_main_right = cex_main, cex_sub = cex_main, highlight_distinct_edges = TRUE, ...

## S3 method for class 'dendlist'
tanglegram(tree1, which = c(1L, 2L), ...)

## S3 method for class 'hclust'
tanglegram(tree1, ...)

## S3 method for class 'phylo'
tanglegram(tree1, ...)

**Arguments**

- **tree1**: tree object (dendrogram/dendlist/hclust/phylo), plotted on the left
- **tree2**: tree object (dendrogram/hclust/phylo), plotted on the right
- **which**: an integer vector of length 2, indicating which of the trees in the dendlist object should be plotted
- **sort**: logical (FALSE). Should the dendrogram’s labels be "sorted"? (might give a better tree in some cases).
- **color_lines**: a vector of colors for the lines connected the labels. If the colors are shorter than the number of labels, they are recycled (and a warning is issued).
- **lwd**: width of the lines connecting the labels. (default is 3.5)
- **edge.lwd**: width of the dendrograms lines.
- **columns_width**: a vector with three elements, giving the relative sizes of the the three plots (left dendrogram, connecting lines, right dendrogram). This is passed to `layout`. The default is: c(5,3,5)
- **margin_top**: the number of lines of margin to be specified on the top of the plots.
- **margin_bottom**: the number of lines of margin to be specified on the bottom of the plots.
- **margin_inner**: margin_bottom the number of lines of margin to be specified on the inner distance between the dendrograms and the connecting lines.
- **margin_outer**: margin_bottom the number of lines of margin to be specified on the outer distance between the dendrograms and the connecting lines.
- **left_dendo_mar**: mar parameters of the left dendrogram.
right_dendo_mar

mar parameters of the right dendrogram.

intersecting

logical (TRUE). Should the leaves of the two dendrograms be pruned so that the
two trees will have the same labels?

dLeaf

a number specifying the distance in user coordinates between the tip of a leaf
and its label. If NULL, as per default, 3/4 of a letter width or height is used.
Notice that if we are comparing two dendrograms with different heights, manu-
ally changing dLeaf will affect both trees differently. In such a case, it is recom-
mended to manually change dLeaf_left and dLeaf_right. This can be especially
important when changing the lab.cex of the dendrogram’s labels. Alternatively,
one could manually set the xlim parameter for both trees, which will force the
proportion of distances of the labels from the trees to remain the same.

dLeaf_left

dLeaf of the left dendrogram, by default it is equal to dLeaf (often negative).

dLeaf_right

dLeaf of the right dendrogram, by default it is equal to minus dLeaf (often pos-
itive).

axes

logical (TRUE). Should plot axes be plotted?

type

type of plot ("t"/"r" = triangle or rectangle)

lab.cex

numeric scalar, influencing the cex size of the labels.

remove_nodePar

logical (FALSE). Should the nodePar of the leaves be removed? (useful when
the trees’ leaves has too many parameters on them)

main

Character. Title above the connecting lines.

main_left

Character. Title of the left dendrogram.

main_right

Character. Title of the right dendrogram.

sub

Character. Title below the connecting lines.

k_labels

integer. Number of groups by which to color the leaves.

k_branches

integer. Number of groups by which to color the branches.

rank_branches

logical (FALSE). Should the branches heights be adjusted? (setting this to
TRUE - can make it easier for comparing topological differences)

hang

logical (FALSE). Should we hang the leaves of the trees?

match_order_by_labels

logical (TRUE). Should the leaves value order be matched between the two trees
based on labels? This is a MUST in order to have the lines connect the correct
labels. Set this to FALSE if you want to make the plotting a bit faster, and only
after you are sure the labels and orders are correctly aligned.

cex_main

A numerical value giving the amount by which plotting title should be magnified
relative to the default.

cex_main_left

see cex_main.

cex_main_right

see cex_main.

cex_sub

see cex_main.

highlight_distinct_edges

logical (default is TRUE). If to highlight distinct edges in each tree (by changing
their line types to 2).
Details

Notice that tanglegram does not "resize" well. In case you are resizing your window you would need to re-run the function.

Value

An invisible dendlist, with two trees after being modified during the creation of the tanglegram.

Author(s)

Tal Galili, Johan Renaudie

Source

The function is based on code from Johan Renaudie (plannapus), after major revisions. See: http://stackoverflow.com/questions/12456768/duelling-dendrograms-in-r-placing-dendrograms-back-to-back-2

As far as I could tell, this code was originally inspired by Dylan Beaudette function dueling.dendrograms from the sharpshootR package: http://cran.at.r-project.org/web/packages/sharpshootR/
tanglegram

See Also

remove_leaves_nodePar, plot_horiz.dendrogram, rank_branches, hang.dendrogram

Examples

```r
## Not run:
set.seed(23235)
s <- sample(1:150, 10)
dend1 <- iris[s, -5] %>% dist %>% hclust("com") %>% as.dendrogram
dend2 <- iris[s, -5] %>% dist %>% hclust("sin") %>% as.dendrogram
dend12 <- dendlist(dend1, dend2)
dend12 %>% tanglegram

  tanglegram(dend1, dend2)
  tanglegram(dend1, dend2, sort = TRUE)
  tanglegram(dend1, dend2, remove_nodePar = TRUE)
  tanglegram(dend1, dend2, k_labels = 6, k_branches = 4)

  tanglegram(dend1, dend2, lab.cex = 2, edge.lwd = 3,
             margin_inner= 5, type = "t", center = TRUE)

## works nicely:
  tanglegram(dend1, dend2, lab.cex = 2, edge.lwd = 3,
             margin_inner= 3.5, type = "t", center = TRUE,
             dLeaf = -0.1, xlim = c(7,0),
             k_branches=3)
```
# using rank_branches can make the comparison even easier
tanglegram(rank_branches(dend1), rank_branches(dend2), lab.cex = 2, edge.lwd = 3,
margin.inner = 3.5, type = "t", center = TRUE,
dLeaf = -0.1, xlim = c(5.1,0), columns.width = c(5,1,5),
k_branches=3)

## End(Not run)

---

**theme_dendro**

*Creates completely blank theme in ggplot*

**Description**

Sets most of the `ggplot` options to blank, by returning blank theme elements for the panel grid, panel background, axis title, axis text, axis line and axis ticks.

**Usage**

```r
theme_dendro()
```

**Author(s)**

Andrie de Vries

**Source**

This function is from Andrie de Vries’s `ggdendro` package.

The motivation for this fork is the need to add more graphical parameters to the plotted tree. This required a strong mixer of functions from `ggdendro` and `dendextend` (to the point that it seemed better to just fork the code into its current form)

**See Also**

`ggdend`
Description

unbranch trees

Usage

unbranch(x, ...)

## S3 method for class 'dendrogram'
unbranch(x, branch_becoming_root = 1, new_root_height, ...)

## S3 method for class 'hclust'
unbranch(x, branch_becoming_root = 1, new_root_height, ...)

## S3 method for class 'phylo'
unbranch(x, ...)

Arguments

x tree (dendrogram/hclust) object

... passed on

branch_becoming_root a numeric choosing the branch of the root which will become the new root (from left to right)

new_root_height the new height of the branch which will become the new root. If the parameter is not given - the height of the original root is used.

Value

An unbranched dendrogram

See Also

unroot ape

Examples

hc <- hclust(dist(USArrests[2:9,]), "com")
dend <- as.dendrogram(hc)

par(mfrow = c(1,3))
plot(dend, main = "original tree")
plot(unbranch(dend, 1), main = "unbranched tree (left branch)")
plot(unbranch(dend, 2), main = "tree without (right branch)"
**unclass_dend**

unclass_dend

**Description**

unclass an entire dendrogram tree

**Usage**

unclass_dend(dend, ...)

**Arguments**

dend  a dendrogram object

...  not used

**Value**

The list which was the dendrogram (but without a class)

**See Also**

nleaves

**Examples**

```r
# define dendrogram object to play with:
hc <- hclust(dist(USArrests[1:3,]), "ave")
dend <- as.dendrogram(hc)

itself <- function(x) x
dend <- dendrapply(dend, itself)
unclass(dend) # this only returns a list with
# two dendrogram objects inside it.
str(dend) # this is a great way to show a dendrogram,
# but it doesn't help us understand how the R object is built.
str(unclass(dend)) # this is a great way to show a dendrogram,
# but it doesn't help us understand how the R object is built.
unclass_dend(dend) # this only returns a list
# with two dendrogram objects inside it.
str(unclass_dend(dend)) # NOW we can more easily understand
# how the dendrogram object is structured...
```
Description

One untangle function to rule them all.

This function untangles dendrogram lists (dendlist), using various heuristics.

Usage

untangle(dend1, ...)  

## S3 method for class 'dendrogram'
untangle(dend1, dend2,  
    method = c("random", "step1side", "step2side", "DendSer"),  
    ...)  

## S3 method for class 'dendlist'
untangle(dend1,  
    method = c("random", "step1side", "step2side", "DendSer"),  
    which = c(1L, 2L), ...)  

Arguments

dend1 a dendrogram or a dendlist object  
... passed to the relevant untangle function  
dend2 A second dendrogram (to untangle against)  
which an integer vector of length 2, indicating which of the trees in the dendlist object should be plotted  
method a character indicating the type of untangle heuristic to use.

Details

This function wraps all of the untangle functions, in order to make it easier to find out about (and use) them.

Value

A dendlist, with two trees after they have been untangled.

Author(s)

Tal Galili
untangle_DendSer

See Also
tanglegram, untangle_random_search, untangle_step_rotate_1side, untangle_step_rotate_2side, entanglement

Examples

```r
## Not run:
set.seed(23235)
ss <- sample(1:150, 10 )
dend1 <- iris[ss,-5] %>% dist %>% hclust("com") %>% as.dendrogram
dend2 <- iris[ss,-5] %>% dist %>% hclust("sin") %>% as.dendrogram
dend12 <- dendlist(dend1, dend2)

untangle(dend1, dend2, method="random", R = 5) %>% tanglegram

# it works, and we get something different:
set.seed(1234)
dend12 %>% untangle(method="random", R = 5) %>% tanglegram

set.seed(1234)
# fixes it completely:
dend12 %>% untangle(method="random", R = 5) %>% untangle(method="step1") %>% tanglegram
# not good enough
# dend12 %>% untangle(method="step1") %>% tanglegram
# not good enough
# dend12 %>% untangle(method="step2") %>% tanglegram
# How we might wish to use it:
set.seed(12777)
dend12 %>%
  untangle(method="random", R = 1) %>%
  untangle(method="step2") %>%
tanglegram

## End(Not run)
```

untangle_DendSer 

Tries to run DendSer on a dendrogram

Description

The function tries to turn the dend into hclust. It then uses the cophenetic distance matrix for optimizing the tree’s rotation.

This is a good (and fast) starting point for linkuntangle_step_rotate_2side

Usage

untangle_DendSer(dend, ...)

untangle_random_search

Untangle - random search

Description

Searches for two untangled dendrogram by randomly shuffling them and each time checking if their entanglement was improved.

Usage

untangle_random_search(tree1, tree2, R = 100L, L = 1,
leaves_matching_method = c("labels", "order"), ...)

Arguments

dend
An object of class dendlist

Value

A dendlist object with ordered dends

See Also

DendSer, DendSer.dendrogram, untangle_DendSer, rotate_DendSer

Examples

## Not run:
set.seed(232)
ss <- sample(1:150, 20)
dend1 <- iris[ss,-5] %>% dist %>% hclust("com") %>% as.dendrogram
dend2 <- iris[ss,-5] %>% dist %>% hclust("sin") %>% as.dendrogram
dend12 <- dendlist(dend1, dend2)

# bad solutions
dend12 %>% tanglegram
dend12 %>% untangle("step2") %>% tanglegram
dend12 %>% untangle_DendSer %>% tanglegram

# but the combination is quite awesome:
dend12 %>% untangle_DendSer %>% untangle("step2") %>% tanglegram

## End(Not run)

---

untangle_random_search

Untangle - random search

Description

Searches for two untangled dendrogram by randomly shuffling them and each time checking if their entanglement was improved.

Usage

untangle_random_search(tree1, tree2, R = 100L, L = 1,
leaves_matching_method = c("labels", "order"), ...)

Arguments

dend
An object of class dendlist

Value

A dendlist object with ordered dends

See Also

DendSer, DendSer.dendrogram, untangle_DendSer, rotate_DendSer

Examples

## Not run:
set.seed(232)
ss <- sample(1:150, 20)
dend1 <- iris[ss,-5] %>% dist %>% hclust("com") %>% as.dendrogram
dend2 <- iris[ss,-5] %>% dist %>% hclust("sin") %>% as.dendrogram
dend12 <- dendlist(dend1, dend2)

# bad solutions
dend12 %>% tanglegram
dend12 %>% untangle("step2") %>% tanglegram
dend12 %>% untangle_DendSer %>% tanglegram

# but the combination is quite awesome:
dend12 %>% untangle_DendSer %>% untangle("step2") %>% tanglegram

## End(Not run)
untangle_random_search

Arguments

- **tree1**
  - a tree object (of class dendrogram/hclust/phylo).
- **tree2**
  - a tree object (of class dendrogram/hclust/phylo).
- **R**
  - numeric (default is 100). The number of shuffles to perform.
- **L**
  - the distance norm to use for measuring the distance between the two trees. It can be any positive number, often one will want to use 0, 1, 1.5, 2 (see 'details' for more). It is passed to `entanglement`.
- **leaves_matching_method**
  - a character scalar passed to `entanglement`. It can be either "order" or "labels" (default). If using "labels", then we use the labels for matching the leaves order value. And if "order" then we use the old leaves order value for matching the leaves order value.
  - Using "order" is faster, but "labels" is safer. "order" will assume that the original two trees had their labels and order values MATCHED.
  - Hence, it is best to make sure that the trees used here have the same labels and the SAME values matched to these values - and then use "order" (for fastest results).
  - If "order" is used, the function first calls `match_order_by_labels` in order to make sure that the two trees have their labels synced with their leaves order values.
- **NNN**
  - not used

Details

Untangaling two trees is a hard combinatorical problem without a closed form solution. One way for doing it is to run through a random spectrom of options and look for the "best" two trees. This is what this function offers.

Value

A dendlist with two trees with the best entanglement that was found.

See Also

- `tanglegram`, `match_order_by_labels`, `entanglement`.

Examples

```r
## Not run:
dend1 <- iris[, -5] %>% dist %>% hclust("com") %>% as.dendrogram
dend2 <- iris[, -5] %>% dist %>% hclust("sin") %>% as.dendrogram
tanglegram(dend1, dend2)

set.seed(65168)
dend12 <- untangle_random_search(dend1, dend2, R = 10)
tanglegram(dend12[[1]], dend12[[2]])
tanglegram(dend12)

entanglement(dend1, dend2, L = 2) # 0.8894
```
### untangle_step_rotate_1side

**Stepwise untangle one tree compared to another**

**Description**

Given a fixed tree and a tree we wish to rotate, this function goes through all of the k number of clusters (from 2 onward), and each time rotates the branch which was introduced in the new k'th cluster. This rotated tree is compared with the fixed tree, and if it has a better entanglement, it will be used for the following iterations.

This is a greedy forward selection algorithm for rotating the tree and looking for a better match. This is useful for finding good trees for a tanglegram.

**Usage**

```r
untangle_step_rotate_1side(dend1, dend2_fixed, L = 1.5,
  direction = c("forward", "backward"), k_seq = NULL, dend_heights_per_k,
  leaves_matching_method = c("labels", "order"), ...)
```

**Arguments**

- **dend1**
  a dendrogram object. The one we will rotate to best fit dend2_fixed.

- **dend2_fixed**
  a dendrogram object. This one is kept fixed.

- **L**
  the distance norm to use for measuring the distance between the two trees. It can be any positive number, often one will want to use 0, 1, 1.5, 2 (see 'details' in entanglement).

- **direction**
  a character scalar, either "forward" (default) or "backward". Impacts the direction of clustering that are tried. Either from 2 and up (in case of "forward"), or from nleaves to down (in case of "backward")

- **k_seq**
  a sequence of k clusters to go through for improving dend1. If NULL (default), then we use the "direction" parameter.

- **dend_heights_per_k**
  a numeric vector of values which indicate which height will produce which number of clusters (k)
untangle_step_rotate_2side

leaves_matching_method

a character scalar passed to entanglement. It can be either "order" or "labels" (default). If using "labels", then we use the labels for matching the leaves order value. And if "order" then we use the old leaves order value for matching the leaves order value.

Using "order" is faster, but "labels" is safer. "order" will assume that the original two trees had their labels and order values MATCHED.

Hence, it is best to make sure that the trees used here have the same labels and the SAME values matched to these values - and then use "order" (for fastest results).

If "order" is used, the function first calls match_order_by_labels in order to make sure that the two trees have their labels synced with their leaves order values.

Value

A dendlist with 1) dend1 after it was rotated to best fit dend2_fixed. 2) dend2_fixed.

See Also

tanglegram, match_order_by_labels, entanglement, flip_leaves, all_couple_rotations_at_k, untangle_step_rotate_2side.

Examples

```r
## Not run:
dend1 <- USArrests[1:10,] %>% dist %>% hclust %>% as.dendrogram
set.seed(3525)
dend2 <- shuffle(dend1)
tanglegram(dend1,dend2)
entanglement(dend1,dend2, L = 2) # 0.4727

dend2_corrected <- untangle_step_rotate_1side(dend2, dend1)[[1]]
tanglegram(dend1,dend2_corrected) # FIXED.
entanglement(dend1,dend2_corrected, L = 2) # 0

## End(Not run)
```

untangle_step_rotate_2side

Stepwise untangle two trees one at a time

Description

This is a greedy forward selection algorithm for rotating the tree and looking for a better match.
This is useful for finding good trees for a tanglegram.
It goes through rotating dend1, then dend2, and so on - until a locally optimal solution is found.
Usage
untangle_step_rotate_2side(dend1, dend2, L = 1.5, direction = c("forward", 
"backward"), max_n_iterations = 10L, 
print_times = dendextend_options("warn"), k_seq = NULL, ...)

Arguments
dend1 a dendrogram object. The one we will rotate to best fit dend2.
dend2 a dendrogram object. The one we will rotate to best fit dend1.
L the distance norm to use for measuring the distance between the two trees. It can be any positive number, often one will want to use 0, 1, 1.5, 2 (see 'details' in entanglement).
direction a character scalar, either "forward" (default) or "backward". Impacts the direction of clustering that are tried. Either from 2 and up (in case of "forward"), or from nleaves to down (in case of "backward")
If k_seq is not NULL, then it overrides "direction".
max_n_iterations integer. The maximal number of times to switch between optimizing one tree with another.
print_times logical (TRUE), should we print how many times we switched between rotating the two trees?
k_seq a sequence of k clusters to go through for improving dend1. If NULL (default), then we use the "direction" parameter.
... not used

Value
A list with two dendrograms (dend1/dend2), after they are rotated to best fit one another.

See Also
tanglegram, match_order_by_labels, entanglement, flip_leaves, all_couple_rotations_at_k. untangle_step_rotate_1side.

Examples
## Not run:
dend1 <- USArrests[1:20,] %>% dist %>% hclust %>% as.dendrogram
dend2 <- USArrests[1:20,] %>% dist %>% hclust(method = "single") %>% as.dendrogram
set.seed(3525)
dend2 <- shuffle(dend2)
tanglegram(dend1,dend2, margin_inner=6.5)
entanglement(dend1,dend2, L = 2) # 0.79

dend2_corrected <- untangle_step_rotate_1side(dend2, dend1)
tanglegram(dend1,dend2_corrected, margin_inner=6.5) # Good.
entanglement(dend1,dend2_corrected, L = 2) # 0.0067
# it is better, but not perfect. Can we improve it?
which_leaf

Gives a vector as the number of nodes (nnodes), which gives a TRUE when a node is a leaf.

Usage

which_leaf(dend, ...)

Arguments

dend a dendrogram dend
... ignored.

Value

A logical vector with the length of nnodes, which gives a TRUE when a node is a leaf.

See Also

noded_with_condition, is.leaf, nnodes

Examples

## Not run:

library(dendextend)

# Getting the dend dend
set.seed(23235)
ss <- sample(1:150, 10 )
dend <- iris[ss,-5] %>% dist %>% hclust %>% as.dendrogram
which_node

which_node(dend, labels, max_id = TRUE, 

Arguments

dend a dendrogram
labels a character vector of labels from the tree
max_id logical (TRUE) - if to return only the max id

Value

An integer with the id(s) of the nodes which includes all of the labels.

See Also

noded_with_condition, branches_attr_by_clusters, nnodes, branches_attr_by_labels, get_nodes_attr
which.edge

Examples

dend <- iris[1:10, -5] %>% dist %>% hclust %>% as.dendrogram %>% set("labels", 1:10)
dend %>% plot

which_node(dend, c(1,2), max_id = FALSE)
which_node(dend, c(2,3), max_id = FALSE)
which_node(dend, c(2,3))

dend %>% plot
the_h <- get_nodes_attr(dend, "height", which_node(dend, c(4,6)))
the_h
abline(h = the_h, lty = 2, col = 2)
get_nodes_attr(dend, "height", which_node(dend, c(4,6)))
get_nodes_attr(dend, "members", which_node(dend, c(4,6)))
Pipe an object forward into a function call/expression.

Usage

`lhs %>% rhs`

Arguments

- `lhs`: The value to be piped
- `rhs`: A function or expression

Source

This is just the imported function from the magrittr package. The documentation you should read for the `%>%` function can be found here: pipe

Adding the function in the package while importing it is a trick taken from the dplyr package (in the file chain.r)

The package could have also been placed in "Depends", But I wanted to keep it robust if at some point some other package will try to take over the symbol (not likely, but still).

See Also

pipe
Index

*Topic **dendrogram**
  - dendextend-package, 4
*Topic **package**,
  - dendextend-package, 4
%>%, 153
  - all.equal, 5
  - all.equal.dendlist
    - (all.equal.dendrogram), 5
  - all.equal.dendrogram, 5
  - all.equal.phylo, 5
  - all_couple_rotations_at_k, 6, 149, 150
  - all_unique, 7
  - apply, 136
  - as.dendlist, 8
  - as.dendrogram, 9
  - as.gg dend (gg dend), 80
  - as.hclust, 9, 15
  - as.hclust.dendrogram, 9
  - as.phylo, 9
  - as.phylo.dendrogram, 9
  - as.phylo.hclust, 9
  - as.hclust_fixed, 15
  - assign_dendextend_options, 10
  - assign_values_to_branches_edgePar, 10, 56, 86, 120, 122, 129, 130
  - assign_values_to_leaves_edgePar, 11
  - assign_values_to_leaves_nodePar, 12, 14, 75, 76, 121, 129, 130
  - assign_values_to_nodes_nodePar, 13, 129
  - attr, 80, 104
  - bakers.gamma_for_2_k_matrix, 15, 37
  - Bk, 16, 19, 22
  - Bk_permutations, 18, 22
  - Bk_plot, 20
  - branches_attr_by_clusters, 23, 30, 152
  - branches_attr_by_labels, 23, 25, 106, 129, 130, 152
  - branches_color (color_branches), 31
  - click_rotate, 26
  - clip, 119
  - collapse_branch, 28
  - color_branches, 31, 35, 43, 98, 129, 130
  - color_clusters, 32
  - color_labels, 34, 129
  - colored_bars, 29
  - colour_branches (color_branches), 31
  - colour_labels (color_labels), 34
  - cophenetic, 36, 40, 54, 126, 145
  - cor, 39
  - cor.dendlist, 36
  - cor_bakers.gamma, 16, 17, 36, 37, 40, 66, 68, 70, 72, 99
  - cor_cophenetic, 36, 38, 39, 135
  - count_terminal_nodes, 41, 104, 105
  - cut_lower_fun, 49
  - cutree, 23, 31, 32, 35, 37, 42, 43, 44, 46, 48, 136
  - cutree.dendrogram, 6, 32, 49, 50, 88, 119
  - cutree_lh.dendrogram, 44, 46, 48
  - cutree_lk.dendrogram, 44, 47
  - cutreeDynamic, 23
  - d3dendro_options, 50, 51, 51
  - d3dendrogram, 50
  - dend_diff, 55, 85
  - dendbackback (tanglegram), 137
  - dendextend (dendextend-package), 4
  - dendextend-package, 4
  - dendextend_cut_lower_fun
    - (cut_lower_fun), 49
  - dendextend_get_branches_heights
    - (get_branches_heights), 72
  - dendextend_heights_per_k.dendrogram
    - (heights_per_k.dendrogram), 84
  - dendextend_options, 52
  - dendlist, 5, 36, 53, 55, 56, 85, 88, 97, 129, 140, 144, 146
  - dendrapply, 96

154
INDEX

identify.dendrogram, 87
identify.hclust, 87, 88
intersect, 88
intersect_trees, 36, 37, 39, 88
is.dendlist, 89
is.dendrogram, 90
is.double, 91
is.hclust, 90
is.integer, 91
is.leaf, 151
is.natural_number, 91
is.numeric, 91
is.phylo, 92
is_null_list, 92
labels, 32, 50, 88, 95, 127
labels.hclust(labels<-), 93
labels.matrix, 95
labels.matrix(labels<-), 93
labels.phylo(labels<-), 93
labels<-, 93
labels<- .dendrogram, 129, 130
labels_colors, 32, 35, 76, 95
labels_colors<-, 130
labels_colors<- (labels_colors), 95
ladderize, 97, 97, 124, 125
layout, 138
leaf_Colors, 98
leaf_colors(leaf_Colors), 98
lowest_common_branch, 99
match_order_by_labels, 7, 37, 61, 64, 99,
101, 147, 149, 150
match_order_dendrogram_by_old_order, 100
multi2di, 28

na.locf, 102
na_locf, 102
names, 62
nleaves, 23, 26, 75, 77, 78, 103, 105, 106, 143
nrows, 23, 26, 75, 77, 78, 104, 106, 151, 152
noded_with_condition, 26, 105, 130, 151,
152
nrow, 104, 105
order, 124, 125
order.dendrogram, 23, 88, 104, 107, 108,
119, 124, 125

identical, 5

dendrogram, 4, 5, 32, 35, 50, 55, 57, 81, 82,
87, 97, 109, 118, 129, 130, 132, 133
DendSer, 54, 126, 146
DendSer.dendrogram, 54, 54, 126, 146
dist, 32, 56, 94, 135
dist.dendlist, 56, 56, 57, 86, 109
dist.multiphyl, 56
dist.topo, 56
dist_long, 58
distinct.edges, 56, 57, 86
distinct_edges, 56, 57, 86, 109
drop.tip, 112
duplicate_leaf, 58, 127
entanglement, 7, 59, 64, 99–101, 132, 133,
145, 147–150
fac2num, 62, 136
fix_members_attr.dendrogram, 59, 62, 127
flatten.dendrogram, 63
flip_leaves, 7, 64, 149, 150
FM_index, 17, 19, 22, 65, 68
FM_index_permutation, 67
FM_index_profdpm, 16, 65, 66, 68, 69, 71, 72,
134
FM_index_R, 16, 65, 68, 69, 71
get_branches_heights, 72, 74, 117
get_childrens_heights, 73, 117
get_leaves_attr, 12–14, 23, 26, 74, 77, 106,
121
get_leaves_nodePar, 75, 82
get_nodes_attr, 75, 76, 76, 78, 82, 152
get_nodes_xy, 78
get_root_branches_attr, 11, 79, 120, 122
ggdend, 80, 141
ggdendrogram, 82
ggplot, 81, 82
ggplot.dendrogram(ggdend), 80
ggplot.ggdend(ggdend), 80
hang.dendrogram, 83, 116, 117, 129, 130, 140
hclust, 4, 32, 35, 44, 46, 48, 94, 132, 133
heights_per_k.dendrogram, 6, 84
highlight_distinct_edges, 56, 57, 85, 86,
109
order.dendrogram<-, 106
order.hclust, 108
par, 29, 119
partition.leaves, 109
partition_leaves, 108
phylo, 97, 132, 133
phylo.diff, 55
pipe, 153
plot, 21, 110
plot.dendlist (dendlist), 53
plot.dendrogram, 55, 78, 81, 110
plot_horiz.dendrogram, 109, 140
plotDendroAndColors, 23, 30
plotHclustColors, 30
prepare.ggdend (ggdend), 80
prune, 88, 111, 112, 117
prune_leaf, 112
pvclust_show_signif, 113, 113, 115
pvclust_show_signif_gradient, 113, 114, 115
rainbow, 32, 35
rainbow_hcl, 32, 35
raise.dendrogram, 116
rank, 117
rank_branches, 116, 140
rank_order.dendrogram, 59, 117, 127
rect, 118, 119
rect.dendrogram, 118
rect.hclust, 88, 119
remove_branches_edgePar, 120, 130
remove_leaves_nodePar, 121, 130, 140
remove_nodes_nodePar, 122
rev.dendrogram, 97, 124, 125
rev.hclust (rotate), 123
rllply, 122
rotate, 97, 123, 125, 132, 133
rotate.dendrogram, 27
rotate_DendSer, 54, 126, 126, 146
rownames, 32
sample, 127
sample.dendrogram, 127
set, 128
shuffle, 132
simplifyRarray, 76
slice, 32, 98
sort, 136
sort.dendrogram (rotate), 123
sort.hclust (rotate), 123
sort_2_clusters_vectors, 134
sort_dist_mat, 135
sort_levels_values, 44, 136
stats, 4
tanglegram, 6, 7, 56, 57, 61, 64, 85, 86, 100,
101, 109, 110, 117, 132, 133, 137, 145, 147–150
theme_dendro, 81, 141
treedist, 56
unbranch, 142
unclass_dend, 143
unique, 7
unroot, 142
untangle, 144
untangle_DendSer, 54, 126, 145, 146
untangle_random_search, 145, 146
untangle_step_rotate_1side, 145, 148, 150
untangle_step_rotate_2side, 145, 149, 149
which.edge, 152
which_leaf, 151
which_node, 152