Package ‘dendextend’

March 25, 2023

Type Package

Title Extending ‘dendrogram’ Functionality in R

Version 1.17.1

Date 2023-03-24

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’s 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, datasets, magrittr (>= 1.0.1), ggplot2, viridis

Suggests knitr, rmarkdown, testthat, seriation, colorspace, ape, microbenchmark, gplots, heatmaply, dynamicTreeCut, pvclust, corrplot, DendSer, MASS, cluster, fpc, circlize (>= 0.2.5), covr

Enhances ggdendro, dendroextras, Hmisc, data.table, rpart, WGCNA, moduleColor, distory, phangorn, zoo

VignetteBuilder knitr

LazyData true

License GPL-2 | GPL-3


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

RoxygenNote 7.2.3

Encoding UTF-8


R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>assign_values_to_leaves_nodePar</td>
<td>15</td>
</tr>
<tr>
<td>assign_values_to_nodes_nodePar</td>
<td>17</td>
</tr>
<tr>
<td>as_hclust_fixed</td>
<td>18</td>
</tr>
<tr>
<td>bakers_gamma_for_2_k_matrix</td>
<td>19</td>
</tr>
<tr>
<td>Bk</td>
<td>20</td>
</tr>
<tr>
<td>Bk_permutations</td>
<td>21</td>
</tr>
<tr>
<td>Bk_plot</td>
<td>23</td>
</tr>
<tr>
<td>branches_attr_by_clusters</td>
<td>26</td>
</tr>
<tr>
<td>branches_attr_by_labels</td>
<td>29</td>
</tr>
<tr>
<td>branches_attr_by_lists</td>
<td>31</td>
</tr>
<tr>
<td>circlize_dendrogram</td>
<td>32</td>
</tr>
<tr>
<td>click_rotate</td>
<td>34</td>
</tr>
<tr>
<td>collapse_branch</td>
<td>36</td>
</tr>
<tr>
<td>collapse_labels</td>
<td>37</td>
</tr>
<tr>
<td>colored_bars</td>
<td>38</td>
</tr>
<tr>
<td>colored_dots</td>
<td>42</td>
</tr>
<tr>
<td>color_branches</td>
<td>46</td>
</tr>
<tr>
<td>color_labels</td>
<td>50</td>
</tr>
<tr>
<td>color_unique_labels</td>
<td>52</td>
</tr>
<tr>
<td>common_subtrees_clusters</td>
<td>53</td>
</tr>
<tr>
<td>cor.dendlist</td>
<td>54</td>
</tr>
<tr>
<td>cor_bakers_gamma</td>
<td>55</td>
</tr>
<tr>
<td>cor_common_nodes</td>
<td>58</td>
</tr>
<tr>
<td>cor_cophenetic</td>
<td>59</td>
</tr>
<tr>
<td>cor_FM_index</td>
<td>61</td>
</tr>
<tr>
<td>count_terminal_nodes</td>
<td>62</td>
</tr>
<tr>
<td>cutree</td>
<td>63</td>
</tr>
<tr>
<td>cutree_1h.dendrogram</td>
<td>67</td>
</tr>
<tr>
<td>cutree_1k.dendrogram</td>
<td>69</td>
</tr>
<tr>
<td>cut_lower_fun</td>
<td>71</td>
</tr>
<tr>
<td>dendextend_options</td>
<td>72</td>
</tr>
<tr>
<td>dendlist</td>
<td>73</td>
</tr>
<tr>
<td>DendSer.dendrogram</td>
<td>74</td>
</tr>
<tr>
<td>dend_diff</td>
<td>75</td>
</tr>
<tr>
<td>dend_expand</td>
<td>76</td>
</tr>
<tr>
<td>dist.dendlist</td>
<td>77</td>
</tr>
<tr>
<td>distinct_edges</td>
<td>79</td>
</tr>
<tr>
<td>dist_long</td>
<td>80</td>
</tr>
<tr>
<td>duplicate_leaf</td>
<td>80</td>
</tr>
<tr>
<td>entanglement</td>
<td>82</td>
</tr>
<tr>
<td>fac2num</td>
<td>84</td>
</tr>
<tr>
<td>find_dendrogram</td>
<td>85</td>
</tr>
<tr>
<td>find_k</td>
<td>86</td>
</tr>
<tr>
<td>fix_members_attr.dendrogram</td>
<td>87</td>
</tr>
<tr>
<td>flatten.dendrogram</td>
<td>88</td>
</tr>
<tr>
<td>flip_leaves</td>
<td>89</td>
</tr>
<tr>
<td>FM_index</td>
<td>90</td>
</tr>
<tr>
<td>FM_index_permutation</td>
<td>92</td>
</tr>
</tbody>
</table>
R topics documented:

FM_index_R ................................. 94
get_branches_heights .................. 96
get_childrens_heights ................. 97
get_leaves_attr ....................... 98
get_leaves_branches_attr ............. 99
get_leaves_branches_col ............... 100
get_leaves_edgePar .................... 101
get_leaves_nodePar ..................... 102
get_nodes_attr ....................... 103
get_nodes_xy ......................... 105
get_root_branches_attr ................ 106
get_subdendrograms .................... 107
ggdend ................................ 109
hang.dendrogram ...................... 112
has_component_in_attribute .......... 113
heights_per_k.dendrogram .............. 115
highlight_branches_col .............. 116
highlight_distinct_edges ............. 118
identify.dendrogram .................. 120
intersect_trees ....................... 122
is.natural.number ..................... 123
is_null_list ........................... 124
is_some_class ......................... 125
khan ................................ 125
labels<- ................................ 127
labels_cex ................................ 129
labels_colors .......................... 130
ladderize ................................ 131
Leaf_Colors ............................. 132
lowest_common_branch .................. 133
match_order_by_labels ............... 134
match_order_dendrogram_by_old_order . 135
min_depth ................................ 137
na_locf ................................ 137
nleaves ................................ 139
nnodes ................................ 140
noded_with_condition .................. 141
order.dendrogram<- ................... 142
order.hclust ............................ 143
partition_leaves ...................... 144
plot_horiz.dendrogram .................. 145
prune ................................ 147
prune_common_subtrees.dendlist ........ 149
prune_leaf .............................. 150
pvclust_edges ......................... 150
pvclust_show_signif ................... 151
pvclust_show_signif_gradient .......... 153
pvrect2 ................................ 154
Functions for extending dendrogram objects

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's graphical parameters - the color, size, type, etc of its branches, nodes and labels. (2) Visually and statistically compare different 'dendrograms' to one another.

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.
The function can get two dendlist objects and compare them using all.equal.list. If a dendlist is in only "target" (and not "current"), it will go through the dendlist and compare all of the dendrograms within it to one another.

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.
all_couple_rotations_at_k

See Also

all.equal, all.equal.phylo, identical

Examples

## Not run:

```r
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()
dend3 <- iris[ss, -5] %>%
  dist() %>%
  hclust("ave") %>%
  as.dendrogram()
dend4 <- iris[ss, -5] %>%
  dist() %>%
  hclust("centroid") %>%
  as.dendrogram()
# cutree(dend1)

all.equal(dend1, dend1)
all.equal(dend1, dend2)
all.equal(dend1, dend2, use.edge.length = FALSE)
all.equal(dend1, dend2, use.edge.length = FALSE, use.topology = FALSE)

all.equal(dend2, dend4, use.edge.length = TRUE)
all.equal(dend2, dend4, use.edge.length = FALSE)

all.equal(dendlist(dend1, dend2, dend3, dend4))
all.equal(dendlist(dend1, dend2, dend3, dend4), use.edge.length = FALSE)
all.equal(dendlist(dend1, dend1, dend1))
## End(Not run)
```

---

**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.
all_couple_rotations_at_k

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!).

See Also

tanglegram, match_order_by_labels, entanglement, flip_leaves.

Examples

## Not run:
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

Description

Checks if all the elements in a vector are unique

Usage

all_unique(x, ...)

Arguments

x a vector

... ignored.

Value

logical (are all the elements in the vector unique)

Source


See Also

unique

Examples

all_unique(c(1:5, 1, 1))
all_unique(c(1, 1, 2))
all_unique(c(1, 1, 2, 3, 3, 3, 3))
all_unique(c(1, 3, 2))
all_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

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

## 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[1:30, -5] %>%
   dist() %>%
   hclust() %>%
   as.dendrogram()
dend2 <- as.phylo(dend)
plot(dend2, type = "fan")

library(dendextend)
library(ggplot2)
# no longer needed: library(ggdendro)
dend <- iris[1:30, -5] %>%
   dist() %>%
   hclust() %>%
   as.dendrogram()
# there is a bug in the location of the labels
# If you want to solve it - please send a Pull Request to:
# https://github.com/talgalili/dendextend/
ggplot(dend) +
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

*Assign 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(
dend,
value,
edgePar,
skip_leaves = FALSE,
warn = dendextend_options("warn"),
...
)

```r
scale_y_reverse(expand = c(0.2, 0)) + coord_polar(start = 1, theta="x")

## End(Not run)

# see: https://github.com/klutometis/roxygen/issues/796
#
assign_values_to_branches_edgePar

Arguments

- **dend**: A dendrogram object
- **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".
- **skip_leaves**: Logical (FALSE) - should the leaves be skipped/ignored?
- **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 edgePar attribute in all of its branches,

See Also

- get_root_branches_attr

Examples

```r
# This failed before - now it works fine. (thanks to Martin Maechler)
dend <- 1:2 %>%
dist() %>%
hclust() %>%
as.dendrogram()
dend %>%
  set("branches_lty", 1:2) %>%
  set("branches_col", c("topbranch_never_plots", "black", "orange")) %>%
  plot()
## Not run:
dend <- USArrests[1:5, ] %>%
dist() %>%
hclust() %>%
as.dendrogram()
plot(dend)
dend <- assign_values_to_branches_edgePar(dend = dend, value = 2, edgePar = "lwd")
plot(dend)
dend <- assign_values_to_branches_edgePar(dend = dend, value = 2, edgePar = "col")
plot(dend)
dend <- assign_values_to_branches_edgePar(dend = dend, value = "orange", edgePar = "col")
plot(dend)
dend2 <- assign_values_to_branches_edgePar(dend = dend, value = 2, edgePar = "lty")
plot(dend2)
dend2 %>%
  unclass() %>
```
assign_values_to_leaves_edgePar

Assign values to edgePar of dendrogram's leaves

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

assign_values_to_leaves_edgePar(
  dend,
  value,
  edgePar,
  warn = dendextend_options("warn"),
  ...
)

Arguments

dend 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

Value

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

See Also

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

generate.n(3)  #>>

generate.n(3)  #>>

## 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(
  dend,  
  value,  
  nodePar,  
  warn = dendextend_options("warn"),  
  ...  
)
```

**Arguments**

- **dend**: 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.
warning 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

get_leaves_attr

Examples

## Not run:

```r
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(dend = dend, value = c(3, 2), nodePar = "lab.col")
plot(dend)

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

str(unclass(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.

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

Arguments
  
dend          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)

### Making sure this works for NA with character.
dend %>%
  assign_values_to_nodes_nodePar(value = 19, nodePar = "pch") %>%
  assign_values_to_nodes_nodePar(value = c("red", NA), nodePar = "col") -> dend2
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
bakers_gamma_for_2_k_matrix

Examples

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

bakers_gamma_for_2_k_matrix(
  k_matrix_dend1,
  k_matrix_dend2,
  to_plot = FALSE,
  ...
)

Arguments

k_matrix_dend1  a matrix of k cluster groupings from a dendrogram
k_matrix_dend2  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

\[ \text{Bk}(\text{tree1, tree2, k, warn = dendextend_options("warn"), ...}) \]

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.
- **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).

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


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

See Also

FM_index, cor_bakers_gamma, Bk_plot
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)
plot(unlist(y) ~ c(2:10), type = "b", ylim = c(0, 1))

# can take a few seconds
y <- Bk(hc1, hc2)
plot(unlist(y) ~ as.numeric(names(y)),
     main = "Bk plot", pch = 20,
     xlab = "k", ylab = "FM Index",
     type = "b", ylim = c(0, 1)
)

# we are still missing some hypothesis testing here.
# for this we'll have the Bk_plot function.
```

## End(Not run)

Bk_permutations

Bk_permutations - 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).

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


https://en.wikipedia.org/wiki/Fowlkes-Mallows_index
Bk_plot

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.treerogram(hc1)
# tree2 <- as.treerogram(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

```r
Bk_plot(
  tree1,
  tree2,
  k,
  add_E = TRUE,
  rejection_line_asymptotic = TRUE,
  rejection_line_permutation = FALSE,
  R = 1000,
)```
Bk_plot

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.
p.adjust.methods a character scalar of either "none" (default), or "bonferroni". This controls the multiple correction method to use for the critical rejection values. Currently only the Bonferroni method is implemented (based on the number of different k values).
col_line_Bk the color of the Bk line.
Bk_plot

col_line_asymptotic
the color of the rejection asymptotic Bk line.

col_line_permutation
the color of the rejection asymptotic Bk line.

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.

main
passed to plot.
xlab
passed to plot.
ylab
passed to plot.
xlim
passed to plot. If missign, xlim is from 2 to nleaves-1
ylim
passed to plot.

try_cutree_hclust
logical (TRUE). Since cutree for hclust is MUCH faster than for dendrogram - Bk_plot 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 functions. If try_cutree_hclust=FALSE, it will force to use cutree.dendrogram and not cutree.hclust.

... 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.
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 theoratical values, etc.

References


See Also

FM_index, Bk, Bk_permutations
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.treerogram(hc1)
# tree2 <- as.treerogram(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...
# 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...
hc1mixed <- as.hclust(sample(as.dendrogram(hc1)))
Bk_plot(
  tree1 = hc1, tree2 = hc1mixed,
  add_E = FALSE,
  rejection_line_permutation = TRUE, k_permutation = c(2, 4, 6, 8, 10, 20, 30, 40, 50), R = 100
)

## End(Not run)
```

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

branches_attr_by_clusters(
  dend, 
  clusters, 
  values, 
  attr = c("col", "lwd", "lty"), 
  branches_changed_have_which_labels = c("any", "all"), 
  ... 
)

Arguments

- `dend`: a dendrogram dend
- `clusters`: an integer vector of clusters. This HAS to be of the same length as 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 unique non-0 clusters. If it is shorter, it is recycled.
  OR. this can also be of the same length as the number of leaves in the tree In which case, the values will be aggregated (i.e.: tapply), to match the number of clusters. The first value of each cluster will be used as the main value.
  TODO: So far, the function doesn’t deal well with NA values. (this might be changed in the future)
- `attr`: a character with one of the following values: col/lwd/lty
- `branches_changed_have_which_labels`: character with either "any" (default) or "all". Inidicates how the branches should be updated.
- `...`: 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 flaxible 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(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)
```
branches_attr_by_labels

Change col/lwd/lty of branches matching labels condition

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
branches_attr_by_labels

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_lists

Change col/lwd/lty of branches from the root down to clusters defined by list of labels of respective members

Description

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

Usage

```r
branches_attr_by_lists(
  dend,
  lists,
  TF_values = c(2, 1),
  attr = c("col", "lwd", "lty"),
  ...
)
```

Arguments

dend a dendrogram dend

lists a list where each element contains the labels of members in selected nodes down to which the branches shall be adapted

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 2/1 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

... ignored.
circlize_dendrogram

Value

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

See Also

branches_attr_by_labels

Examples

## 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()

# define a list of nodes
L <- list(c("109", "123", "126", "145"), "29", c("59", "67", "97"))
dend %>%
  branches_attr_by_lists(L) %>%
  plot()

# choose different color, and also change lwd and lty
dend %>%
  branches_attr_by_lists(L, TF_value = "blue") %>%
  branches_attr_by_lists(L, attr = "lwd", TF_value = 4) %>%
  branches_attr_by_lists(L, attr = "lty", TF_value = 3) %>%
  plot()

## End(Not run)

circlize_dendrogram  Plot a circlized dendrograms

Description

Plot a circlized dendrograms using the circlize package (must be installed for the function to work).

This type of plot is also sometimes called fan tree plot (although the name fan-plot is also used for a different plot in time series analysis), radial tree plot, polar tree plot, circular tree plot, and probably other names as well.

An advantage for using the circlize package directly is for plotting a circular dendrogram so that you can add more graphics for the elements in the tree just by adding more tracks using circos.track.
**Usage**

```r
circlize_dendrogram(
    dend,
    facing = c("outside", "inside"),
    labels = TRUE,
    labels_track_height = 0.1,
    dend_track_height = 0.5,
    ...
)
```

**Arguments**

- `dend`: a `dendrogram` object
- `facing`: Is the dendrograms facing inside to the circle or outside.
- `labels`: logical (TRUE) - should the labels be plotted as well.
- `labels_track_height`: a value for adjusting the room for the labels. It is 0.2 by default, but if NULL or NA, it will adjust automatically based on the max width of the labels. However, if this is too long, the plot will give an error: Error in check.track.position(track.index, track.start, track.height): not enough space for cells at track index '2'.
- `dend_track_height`: a value for adjusting the room for the dendrogram.
- `...`: Ignored.

**Value**

The dend that was used for plotting.

**Author(s)**

Zuguang Gu, Tal Galili

**Source**

This code is based on the work of Zuguang Gu. If you use the function, please cite both `dendextend` (see: `citation("dendextend")`), as well as the circlize package (see: `citation("circlize")`).

**See Also**

`circos.dendrogram`

**Examples**

```r
## Not run:
dend <- iris[1:40, -5]
  %>%
  dist()
```
```r
hclust() %>%
  as.dendrogram() %>%
  set("branches_k_color", k = 3) %>%
  set("branches_lwd", c(5, 2, 1.5)) %>%
  set("branches_lty", c(1, 1, 3, 1, 1, 2)) %>%
  set("labels_colors") %>%
  set("labels_cex", c(.9, 1.2)) %>%
  set("nodes_pch", 19) %>%
  set("nodes_col", c("orange", "black", "plum", NA))

circlize_dendrogram(dend)
circlize_dendrogram(dend, labels = FALSE)
circlize_dendrogram(dend, facing = "inside", labels = FALSE)

# In the following we get the dendrogram but can also get extra information on top of it
circos.initialize("foo", xlim = c(0, 40))
circos.track(ylim = c(0, 1), panel.fun = function(x, y) {
  circos.rect(1:40 - 0.8, rep(0, 40), 1:40 - 0.2, runif(40), col = rand_color(40), border = NA)
}, bg.border = NA)
circos.track(ylim = c(0, 1), panel.fun = function(x, y) {
  circos.text(1:40 - 0.5, rep(0, 40), labels(dend),
  col = labels_colors(dend),
  facing = "clockwise", niceFacing = TRUE, adj = c(0, 0.5)
}
), bg.border = NA, track.height = 0.1)
max_height <- attr(dend, "height")
circos.track(ylim = c(0, max_height), panel.fun = function(x, y) {
  circos.dendrogram(dend, max_height = max_height)
}, track.height = 0.5, bg.border = NA)
circos.clear()
```

---

### click_rotate

**Interactively rotate a tree object**

**Description**

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

Code for mouse selection of (sub-)cluster to be rotated

**Usage**

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

## Default S3 method:
click_rotate(x, ...)

```r
```
## 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)
- **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)
```
collapse_branch

Collapsing 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]
dend %>%
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_labels

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")

collapse_labels  
Collapse a sub dendrogram of adjacent labels within a dend

Description
Given a dendrogram object, and a set of labels that are in the same sub-dendrogram, the function performs a recursive DFS algorithm to determine the sub-dendrogram which is composed of (exactly) all `selected_labels`. It then squashes this sub-dendrogram, and returns the original dendrogram with the squashed dendrogram with it.

Usage
collapse_labels(dend, selected_labels, ...)

Arguments
  dend  a dendrogram object
  selected_labels  A character vector with the labels we expect to have in the sub-dendrogram. This doesn’t have to be in the same order as in the dendrogram.
  ...  elipsis (passed to squash_dendrogram)

Value
Either the original dend. Or, if the labels properly are in the dend by each other, a dend with a squashed sub-dendrogram inside it.

Examples
library("dendextend")
set.seed(23235)
ss <- sample(1:150, 5)
# Getting the dend object
dend25 <- iris[ss, -5] %>%
  dist() %>%
  hclust() %>%
  as.dendrogram() %>%
  set("labels", letters[1:5])

par(mfrow = c(1,4))
plot(dend25)
plot(collapse_labels(dend25, c("d", "e")))
plot(collapse_labels(dend25, c("c", "d", "e")))
plot(collapse_labels(dend25, c("c", "d", "e"), squashed_original_height=TRUE))

---

**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**

```r
colored_bars(
  colors,
  dend,
  rowLabels = NULL,
  cex.rowLabels = 0.9,
  add = TRUE,
  y_scale,
  y_shift,
  text_shift = 1,
  sort_by_labels_order = TRUE,
  horiz = 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 group with color per object. Each column will be plotted as a horizontal row of colors (when horiz = FALSE) under the dendrogram. As long as the sort_by_labels_order parameter is TRUE (default), the colors vector/matrix should be provided in the order of the original data order (and it will be reordered automatically to the order of the dendrogram)
colored_bars

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(TRUE) - if TRUE (default), 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 the colored bars
are plotted as-is, based on the order of the colors vector.

horiz
logical (FALSE by default). Set to TRUE when using plot(dend, horiz = TRUE)

... ignored at this point.

Details
You will often needs 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: https://cran.r-project.org/package=moduleColor For more details.
See Also

branches_attr_by_clusters, plotDendroAndColors

Examples

```r
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(cols, dend)
# move and scale a bit
plot(dend)
colored_bars(cols, dend,
  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)
colored_bars(cbind(cols2, cols), dend,
  rowLabels = c("2 clusters", "Odd numbers")
)

# The same, but with an horizontal plot!
par(mar = c(6, 2, 2, 4))
plot(dend, horiz = TRUE)
colored_bars(cbind(cols2, cols), dend,
  rowLabels = c("2 clusters", "Odd numbers"),
  horiz = TRUE
)

# 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 %>%
  set("branches_k_color", k = the_k, with = cols3) %>%
```
colored_bars

plot()

kx <- cutree(dend, k = the_k)
ord <- order.dendrogram(dend)
kx <- sort_levels_values(kx[ord])
kx <- kx[match(seq_along(ord), ord)]

par(mar = c(5, 5, 2, 2))
plot(dend)
colored_bars(cbind(cols3[kx], cols2, cols), dend,
  rowLabels = c("3 clusters", "2 clusters", "Odd numbers")
)

## mtcars example

# Create the dend:
dend <- as.dendrogram(hclust(dist(mtcars)))

# Create a vector giving a color for each car to which company it belongs to
car_type <- rep("Other", length(rownames(mtcars)))
is_x <- grepl("Merc", rownames(mtcars))
car_type[is_x] <- "Mercedes"
is_x <- grepl("Mazda", rownames(mtcars))
car_type[is_x] <- "Mazda"
is_x <- grepl("Toyota", rownames(mtcars))
car_type[is_x] <- "Toyota"
car_type <- factor(car_type)
n_car_types <- length(unique(car_type))
col_car_type <- colorspace::rainbow_hcl(n_car_types, c = 70, l = 50)[car_type]

# extra: showing the various clusters cuts
k234 <- cutree(dend, k = 2:4)

# color labels by car company:
labels_colors(dend) <- col_car_type[order.dendrogram(dend)]
# color branches based on cutting the tree into 4 clusters:
dend <- color_branches(dend, k = 4)

### plots

par(mar = c(12, 4, 1, 12))
plot(dend)
colored_bars(cbind(k234[, 3:1], col_car_type), dend,
  rowLabels = c(paste0("k = ", 4:2), "Car Type")
)

# horiz version:
par(mar = c(4, 1, 1, 12))
plot(dend, horiz = TRUE)
colored_bars(cbind(k234[, 3:1], col_car_type), dend,
  rowLabels = c(paste0("k = ", 4:2), "Car Type"), horiz = TRUE
)
colored_dots

Add colored dots beside a dendrogram

Description

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

Usage

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

Arguments

colors  Coloring of the dots beside 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 group with color per object. Each column will be plotted as a colored point (when horiz = FALSE) under the dendrogram. As long as the sort_by_labels_order parameter is TRUE (default), the colors vector/matrix should be provided in the order of the original data order (and it will be re-ordered automatically to the order of 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 dots (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 dots be added to an existing dendrogram plot?
colored_dots

- **y_scale**: how much should the dots be stretched on the y axis? If no dend is supplied - the default will be 1
- **y_shift**: where should the dots be plotted underneath the x axis? By default it will try to locate the dots 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(TRUE) - if TRUE (default), then the order of the colored dots 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 the colored dots are plotted as-is, based on the order of the colors vector.
- **horiz**: logical (FALSE by default). Set to TRUE when using plot(dend, horiz = TRUE)
- **dot_size**: numeric (1 by default). Passed to cex argument in points
- **...**: ignored at this point.

**Details**

The reason you might choose colored_dots over colored_bars is when you have a lot of group types and/or a really large dendrogram. Hint: Make a group for each categorical factor and color it one color when true, and assign a fully transparent color when false.

You will often need to adjust the y_scale, y_shift and the text_shift parameters, in order to get the dots 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>, Tal Galili <Tal.Galili@gmail.com>, Peter Langfelder <Peter.Langfelder@gmail.com>, Chase Clark <chasec288@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 dots on top of an existing plot (and not only as a separate plot).

See: https://cran.r-project.org/package=moduleColor For more details.

**See Also**

branches_attr_by_clusters, plotDendroAndColors
Examples

```r
rows_picking <- c(1:5, 25:30)
dend <- (iris[rows_picking, -5] * 10) %>%
  dist() %>%
  hclust() %>%
  as.dendrogram()
odd_numbers <- rows_picking %% 2
cols <- c("red", "white")[odd_numbers + 1]
plot(dend)
colored_dots(cols, dend)
# Example of adjusting position of dots
plot(dend)
colored_dots(cols, dend,
y_shift = -1,
  rowLabels = "Odd\n numbers"
)
```

```r
rows_picking <- c(1:5, 25:30)
dend <- (iris[rows_picking, -5] * 10) %>%
  dist() %>%
  hclust() %>%
  as.dendrogram()
odd_numbers <- rows_picking %% 2
# For leaves that shouldn't have dots, make them the same color as the background,
# or set the alpha value to fully transparent
cols <- c("black", "white")[odd_numbers + 1]
# scale is off
plot(dend)
colored_dots(cols, dend)
# move and scale a bit
plot(dend)
colored_dots(cols, dend,
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("#1b9e77", 
  "#d95f02")[k2]
par(mar = c(5, 6, 1, 1))
plot(dend)
colored_dots(cbind(cols2, cols), dend,
  rowLabels = c("2 clusters", "Even numbers")
)
```

```r
# The same, but with an horizontal plot!
par(mar = c(6, 2, 2, 4))
plot(dend, horiz = TRUE)
colored_dots(cbind(cols2, cols), dend,
  rowLabels = c("2 clusters", "Even numbers")
)```

## mtcars example

# Create the dend:
dend <- as.dendrogram(hclust(dist(mtcars)))

# Get all company names
comp_names <- unlist(lapply(rownames(mtcars), function(x) strsplit(x, " ")[[1]][[1]]))

# Get the top three occurring companies
  top_three <- sort(table(comp_names), decreasing = TRUE)[1:3]

# Match the top three companies to where they are found in the dendrogram labels
  top_three <- sapply(names(top_three), function(x) grepl(x, labels(dend)))

# "top_three" is now a data frame of the top three companies as columns.
# Each column represents a vector (rows) which is the length of labels(dend).
# The vector has values TRUE and FALSE, for whether the company name matched
# labels(dend)[i]

# Colorblind friendly vector of HEX colors
colorblind_friendly <- c("#1b9e77", "#d95f02", "#7570b3")

# If we run the for-loop on "top_three" we will turn the vectors into a character-type too early,
# so make a copy to "colored_dataframe" which we will work on
  colored_dataframe <- top_three

for (i in 1:3) {
  # This replaces TRUE values with a color from our vector of colors
  colored_dataframe[top_three[, i], i] <- colorblind_friendly[i]

  # This replaces FALSE values with black HEX, but fully transparent (invisible on plot)
  colored_dataframe[!top_three[, i], i] <- "#00000000"
}

# Color branches and labels by "cutting" the dendrogram at an arbitrary height
  dend <- color_branches(dend, h = 170)
  dend <- color_labels(dend, h = 170)

### plots
  par(mar = c(12, 4, 1, 1))
  plot(dend)
  colored_dots(colored_dataframe, dend,
    rowLabels = colnames(colored_dataframe), horiz = FALSE, sort_by_labels_order = FALSE)

  abline(h = 170, lty = 3)

  # horiz version:
  par(mar = c(4, 1, 1, 12))
plot(dend, horiz = TRUE)
colored_dots(colored_dataframe, dend, 
  rowLabels = colnames(colored_dataframe), horiz = TRUE, sort_by_labels_order = FALSE
)
# Show a dotted line where the tree was "cut"
abline(v = 170, lty = 3)

---
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 dend’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**

```r
color_branches(
  dend,
  k = NULL,
  h = NULL,
  col,
  groupLabels = NULL,
  clusters,
  warn = dendextend_options("warn"),
  ...
)
```

**Arguments**

- `dend` A dendrogram or hclust tree object
- `k` number of groups (passed to `cutree`)
- `h` height at which to cut tree (passed to `cutree`)
- `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.
- `groupLabels` If TRUE add numeric group label - see Details for options
- `clusters` an integer vector of clusters. This is passed to `branches_attr_by_clusters`. This HAS to be of the same length as 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.
**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.

... 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, branches_attr_by_clusters, get_leaves_branches_col, color_labels

Examples

```r
# Not run:
par(mfrow = c(1, 2))
dend <- USArrests %>%
dist() %>%
hclust(method = "ave") %>%
as.dendrogram()
d1 <- color_branches(dend, k = 5, col = c(3, 1, 1, 4, 1))
plot(d1) # selective coloring of branches :
d2 <- color_branches(dend, 5)
plot(d2)
```
par(mfrow = c(1, 2))
d1 <- color_branches(dend, 5, col = c(3, 1, 1, 4, 1), groupLabels = TRUE)
plot(d1) # selective coloring of branches :)
d2 <- color_branches(dend, 5, groupLabels = TRUE)
plot(d2)

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

par(mfrow = c(1, 1))

# 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(dend = 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,
# try_cutree_hclust=FALSE)
# cutree(dend_iris,k=3, order_clusters_as_data=FALSE)
library(colorspace)
data(iris, envir = environment())
```r
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(3)[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 (the labels give the true flower species)",
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,
# on it compute hierarchical clustering using the "average" method,
hclust(method = "single") #
as.dendrogram() #
color_branches(k = 3) #
plot() # nice, returns the tree as is...

# Example of the "clusters" parameter
par(mfrow = c(1, 2))
dend <- c(1:5) #
```

color_labels

Color dend'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

color_labels(
  dend,
  k = NULL,
)
Arguments

dend 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, assign_values_to_leaves_edgePar

Examples

## 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 :)

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)

## End(Not run)

color_unique_labels  Color unique labels in a dendrogram

Description

Color unique labels in a dendrogram

Usage

color_unique_labels(dend, ...)

Arguments

dend a dend object

... NOT USED

Value

A dendrogram after the colors of its labels have been updated (a different color for each unique label).

Examples

names(x) <- x
dend <- as.dendrogram(hclust(dist(x)))

par(mfrow = c(1, 2))
plot(dend)
dend2 <- color_unique_labels(dend)
plot(dend2)
common_subtrees_clusters

Find clusters of common subtrees

Description

Gets a dend and the output from "nodes_with_shared_labels" and returns a vector (length of labels), indicating the clusters forming shared subtrees

Usage

common_subtrees_clusters(dend1, dend2, leaves_get_0_cluster = TRUE, ...)

Arguments

dend1 a dendrogram.
dend2 a dendrogram.
leaves_get_0_cluster logical (TRUE). Should the leaves which are not part of a larger common subtree get a unique cluster number, or the value 0.
... not used.

Value

An integer vector, with values indicating which leaves in dend1 form a common subtree cluster, with ones available in dend2

See Also

color_branches, tanglegram

Examples

library(dendextend)
dend1 <- 1:6 %>%
dist() %>%
hclust() %>%
as.dendrogram()
dend2 <- dend1 %>% set("labels", c(1:4, 6:5))
tanglegram(dend1, dend2)

clusters1 <- common_subtrees_clusters(dend1, dend2)
dend1_2 <- color_branches(dend1, clusters = clusters1)
plot(dend1_2)
plot(dend1_2, horiz = TRUE)
tanglegram(dend1_2, dend2, highlight_distinct_edges = FALSE)
tanglegram(dend1_2, dend2)
**cor.dendlist**

**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**

```r
cor.dendlist(
  dend,
  method = c("cophenetic", "baker", "common_nodes", "FM_index"),
  ...)
```

**Arguments**

- `dend` a dendlist of trees
- `method` a character string indicating which correlation coefficient is to be computed. One of "cophenetic" (default), "baker", "common_nodes", or "FM_index". It can be abbreviated.
- `...` passed to cor functions.

**Value**

A correlation matrix between the different trees

**See Also**

cophenetic, cor_cophenetic, cor_bakers_gamma, cor_common_nodes, cor_FM_index

**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("single") %>%
  as.dendrogram()
```
cor_bakers_gamma

Baker’s Gamma correlation coefficient

Description

Calculate Baker’s Gamma correlation coefficient for two trees (also known as Goodman-Kruskal-gamma index).

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(dend1, ...)
```

## Default S3 method:
cor_bakers_gamma(dend1, dend2, ...)

## S3 method for class 'dendrogram'
cor_bakers_gamma(
dend1,
dend2,
use_labels_not_values = TRUE,
to_plot = FALSE,
warn = dendextend_options("warn"),
...
)

## S3 method for class 'hclust'
```
cor_bakers_gamma(
    dend1,
    dend2,
    use_labels_not_values = TRUE,
    to_plot = FALSE,
    warn = dendextend_options("warn"),
    ...
)

## S3 method for class 'dendlist'
cor_bakers_gamma(dend1, which = c(1L, 2L), ...)

Arguments

dend1  a tree (dendrogram/hclust/phylo)
...     Passed to cutree.
dend2  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_common_nodes

Proportion of common nodes between two trees

Description

Calculates the number of nodes, in each tree, that are common (i.e.: that have the same exact list of labels). The correlation is between 0 (actually, \(2*(\text{n} - 1)/(2*\text{n})\), for two trees with the same list of labels - since the top node will always be identical for them). Where 1 means that every node in the one tree, has a node in the other tree with the exact same list of labels. Notice this measure is non-parametric (it ignores the heights and relative position of the nodes).

Usage

cor_common_nodes(dend1, dend2, ...)

Arguments

dend1 a dendrogram.
dend2 a dendrogram.
... not used.

Value

A correlation value between 0 to 1 (almost identical trees)

See Also

distinct_edges, cor.dendlist

Examples

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)

cor_cophenetic(dend1, dend2)
cor_common_nodes(dend1, dend2)
tanglegram(dend1, dend2)
# we can see we have only two nodes which are different...
**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**

```r
cor_cophenetic(dend1, ...)
```

### Default S3 method:

```r
cor_cophenetic(
  dend1,
  dend2,
  method_coef = c("pearson", "kendall", "spearman"),
  ...
)
```

### S3 method for class 'dendlist'

```r
cor_cophenetic(
  dend1,
  which = c(1L, 2L),
  method_coef = c("pearson", "kendall", "spearman"),
  ...
)
```

**Arguments**

- `dend1` a tree (dendrogram/hclust/phylo, or dendlist)
- `...` Ignored.
- `dend2` Either a tree (dendrogram/hclust/phylo), or a `dist` object (for example, from the original data matrix).
- `method_coef` a character string indicating which correlation coefficient is to be computed. One of "pearson" (default), "kendall", or "spearman", can be abbreviated. Passed to `cor`.
- `which` an integer vector of length 2, indicating which of the trees in a dendlist object should have their `cor_cophenetic` calculated.

**Details**

From `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

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

See Also

cophenetic, cor_bakers_gamma

Examples

```r
## Not run:
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)

---

**cor_FM_index**

*Correlation of FM_index for some k*

**Description**

Calculates the FM_index Correlation for some k.

**Usage**

```r
cor_FM_index(dend1, dend2, k, ...)
```

**Arguments**

- `dend1` a dendrogram.
- `dend2` a dendrogram.
- `k` an integer (number of clusters to cut the tree)
- `...` not used.
**count_terminal_nodes**

A correlation value between 0 to 1 (almost identical clusters for some k)

**See Also**

FM_index, cor.dendlist, Bk

**Examples**

```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)

cor_FM_index(dend1, dend2, k = 2)
cor_FM_index(dend1, dend2, k = 3)
cor_FM_index(dend1, dend2, k = 4)
```

---

**count_terminal_nodes**  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 count_terminal_nodes 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**

count_terminal_nodes(dend_node, ...)

**Arguments**

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

...  not used
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, ...)

## Default S3 method:
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,
warn = dendextend_options("warn"),
NA_to_0L = TRUE,
...
)

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

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

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

## S3 method for class 'diana'
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,
  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.
- **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?

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).

`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 $k!=NULL$.

`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.

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).

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(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, 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
## as.hclust(dend), k = 2:4) 1701.629 1767.700 1854.4895 2029.1875 8736.591 100
## cutree(dend, k = 2:4) 1807.456 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
```r
# cutree_unbranch_dend_not
# _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 behaviout than stats::cutree
# For example:
cutree(as.dendrogram(hclust(dist(c(1, 1, 1, 2, 2)))),
   k = 5
)

## End(Not run)
```

---

**cutree_1h.dendrogram**  cutree for dendrogram (by 1 height only!)

**Description**

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

**Usage**

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

**Arguments**

- **dend**  a dendrogram object
- **h**  numeric scalar (NOT a vector) with a height where the dend 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.).
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.

... (not currently in use)

Value

cutree_1h.dendrogram returns an integer vector with group memberships

Author(s)

Tal Galili

See Also

hclust, cutree

Examples

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

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

Arguments

dend 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_k.dendrogram. 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)
cutree_1k.dendrogram

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

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)
Description
Cuts the dend 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 \texttt{cutree.dendrogram} function, by using the \texttt{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
\[
\text{cut_lower_fun}(\text{dend}, h, \text{FUN} = \text{labels}, \text{warn} = \text{dendextend_options}("warn"), \ldots)
\]

Arguments
- \texttt{dend} a dendrogram object.
- \texttt{h} a scalar of height to cut the dend by.
- \texttt{FUN} a function to run. (default is "labels")
- \texttt{warn} logical (default from \texttt{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?
- \ldots passed to FUN.

Value
A list with the output of running FUN on each of the sub dends derived from cutting "dend"

Author(s)
Tal Galili

See Also
\texttt{labels, dendrogram, cutree.dendrogram}

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

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

```r
dendlist(..., which)
```

### S3 method for class 'dendlist'

```r
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.
- `which` an integer vector of length 2, indicating which of the trees in the dendlist object should be plotted (relevant for dendlist)
  - When used inside dendlist, which is still an integer, but it can be of any length, and it can be used to create a smaller dendlist.
- `x` a dendlist object

**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**

```r
## 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**

`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`
**dend_diff**

**Value**

Numeric vector giving an optimal dendrogram order

**See Also**

DendSer, DendSer.dendrogram, untangle_DendSer, rotate_DendSer

**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**

dend_diff(dend, ...)

# S3 method for class 'dendrogram'
dend_diff(dend, dend2, horiz = TRUE, ...)

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

**Arguments**

dend a dendrogram or dendlist to compare with

... passed to plot.dendrogram
dend2 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 "dend" 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

```r
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)
```

dend_expend

Finds a "good" dendrogram for a dist

Description

There are many options for choosing distance and linkage functions for `hclust`. This function goes through various combinations of the two and helps find the one that is most "similar" to the original distance matrix.

Usage

dend_expend(
find_dend(x, ...)  

Arguments  
  x: A matrix or a data.frame. Can also be a dist object.  
  dist_methods: A vector of possible dist methods.  
  hclust_methods: A vector of possible hclust methods.  
  hclust_fun: By default hclust.  
  optim_fun: A function that accepts a dend and a dist and returns how the two are in agreement. Default is cor_cophenetic.  
  ... options passed from find_dend to dend_expend.  

Value  
  dend_expend: A list with three items. The first item is called "dends" and includes a dendlist with all the possible dendrogram combinations. The second is "dists" and includes a list with all the possible distance matrix combination. The third, "performance", is data.frame with three columns: dist_methods, hclust_methods, and optim. optim is calculated (by default) as the cophenetic correlation (see: cor_cophenetic) between the distance matrix and the cophenetic distance of the hclust object.  
  find_dend: A dendrogram which is "optimal" based on the output from dend_expend.  

Examples  
  x <- datasets::mtcars  
  out <- dend_expend(x, dist_methods = c("euclidean", "manhattan"))  
  out$performance  
  dend_expend(dist(x))$performance  
  best_dend <- find_dend(x, dist_methods = c("euclidean", "manhattan"))  
  plot(best_dend)
Description

This function seems to bring different results than ape - checking this out is still an open issue:

[github issue]

This function computes the **Robinson-Foulds distance** (also known as symmetric difference) between two dendrograms. This is the number of edges (branches) in tree_1 with a combination of labels that exist in it but not in any subtree of tree2, plus the same calculation of tree2 when compared to tree1. This is the sum of length of `distinct_edges(x,y)` with `distinct_edges(y,x)`.

This function might implement other topological distances in the future.

Usage

\[
dist.dendlist(dend, method = c("edgeset"), \ldots)
\]

Arguments

- `dend`: a `dendlist`
- `method`: currently only 'edgeset' is implemented.
- `\ldots`: Ignored.

Value

A `dist` object with topological distances between all trees

See Also

```
dist.topo, dist.multiPhylo, treedist,
```

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)

# Larger trees
x <- 1:6 %>%
dist() %>%
hclust() %>%
as.dendrogram()
y <- set(x, "labels", c(1:3, 6, 4, 5))

dend_diff(x, y)
dist.dendlist(dendlist(x, y))
distinct_edges(x, y)
distinct_edges(y, x)
```
**distinct_edges**

\[ \text{length(distinct_edges}(x, y)) + \text{length(distinct_edges}(y, x)) \] # dist.dendlist

---

**distinct_edges**

Finds distinct edges in one tree compared to another

**Description**

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

**Usage**

```r
distinct_edges(dend, dend2, ...)
```

**Arguments**

- `dend`: a dendrogram to find unique edges in
- `dend2`: a dendrogram to compare with
- `...`: Ignored.

**Value**

A numeric vector of edge ids for the first tree (dend) that are not present in the second tree (dend2).

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

**Description**

Turns a dist object from a "wide" to a "long" table

**Usage**

```r
dist_long(d, ...)
```

**Arguments**

- `d`: a distance object
- `...`: not used

**Value**

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.

**Examples**

```r
data(iris)
iris[2:6, -5] %>%
dist() %>%
data.matrix()
iris[2:6, -5] %>%
dist() %>%
as.vector()
iris[2:6, -5] %>%
dist() %>%
dist_long()
# This can later be used to making a network plot based on the distances.
```

---

### duplicate_leaf

**Description**

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

Usage

duplicate_leaf(
  dend, 
  leaf_label, 
  times, 
  fix_members = TRUE, 
  fix_order = TRUE, 
  fix_midpoint = TRUE, 
  ...
)

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
... not used

Value

A dendrogram, after duplicating one of its leaves.

Examples

```r
## 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)
```
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(dend1, ...)

## S3 method for class 'hclust'
entanglement(dend1, dend2, ...)

## S3 method for class 'phylo'
entanglement(dend1, dend2, ...)

## S3 method for class 'dendlist'
entanglement(dend1, which = c(1L, 2L), ...)

## S3 method for class 'dendrogram'
entanglement(
  dend1,
  dend2,
  L = 1.5,
  leaves_matching_method = c("labels", "order"),
  ...
)

Arguments

dend1 a tree object (of class dendrogram/hclust/phylo).

... not used

dend2 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 then 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: \( \text{sum}(\text{abs}(x-y)^L) \). And this is divided by the "worst case" entanglement level (e.g. when the right tree is the complete reverse of the left tree).

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

Examples

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

Description

Given a dendrogram object, the function performs a recursive DFS algorithm to determine the sub-
dendrogram which is composed of (exactly) all `selected_labels`.

Usage

find_dendrogram(dend, selected_labels)

Arguments

dend a dendrogram object

selected_labels

A character vector with the labels we expect to have in the sub-dendrogram. This doesn’t have to be in
the same order as in the dendrogram.

Value

Either a sub-dendrogram composed of only members of selected_labels. If such a sub-dendrogram
doesn’t exist, the function returns NULL.

Examples

## Not run:
# define dendrogram object to play with:
dend <- iris[, -5] %>%
dist() %>%
hclust() %>%
as.dendrogram() %>%
set("labels_to_character") %>%
color_branches(k = 5)
first.subdend.only <- names(cutree(dend, 4)[cutree(dend, 4) == 1])
sub.dend <- find_dendrogram(dend, first.subdend.only)
# Plotting the result
par(mfrow = c(1, 2))
plot(dend, main = "Original dendrogram")
plot(sub.dend, main = "First subdendrogram")

dend <- 1:10 %>%
find_k

Find the (estimated) number of clusters for a dendrogram using average silhouette width

Description

This function estimates the number of clusters based on the maximal average silhouette width derived from running pam on the cophenetic distance matrix of the dendrogram. The output is based on the pamk output.

Usage

```r
find_k(dend, krange = 2:min(10, (nleaves(dend) - 1)), ...)
```

## S3 method for class 'find_k'

```r
plot(
  x,
  xlab = "Number of clusters (k)",
  ylab = "Average silhouette width",
  main = "Estimating the number of clusters using
 average silhouette width",
  ...
)
```

Arguments

- **dend**
  A dendrogram (or hclust) tree object

- **krange**
  integer vector. Numbers of clusters which are to be compared by the average silhouette width criterion. Note: average silhouette width and Calinski-Harabasz can’t estimate number of clusters nc=1. If 1 is included, a Duda-Hart test is applied and 1 is estimated if this is not significant.

- ... passed to pamk (the current defaults criterion="asw" and usepam=TRUE can not be changes).
x An object of class "find_k" (has its own S3 plot method).
xlab, ylab, main parameters passed to plot.

Value

A pamk output. This is a list with the following components: 1) pamobject - The output of the optimal run of the pam-function. 2) nc - the optimal number of clusters. 3) crit - vector of criterion values for numbers of clusters. crit[1] is the p-value of the Duda-Hart test if 1 is in krange and diss=FALSE. 4) k - a copy of nc (just to make it easier to extract - since k is often used in other functions)

See Also

pamk, pam, silhouette.

Examples

dend <- iris[, -5] %>%
  dist() %>%
  hclust() %>%
  as.dendrogram()
dend_k <- find_k(dend)
plot(dend_k)
plot(color_branches(dend, k = dend_k$nc))

library(cluster)
sil <- silhouette(dend_k$pamobject)
plot(sil)

dend <- USArrests %>%
  dist() %>%
  hclust(method = "ave") %>%
  as.dendrogram()
dend_k <- find_k(dend)
plot(dend_k)
plot(color_branches(dend, k = dend_k$nc))
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

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

par(mfrow = c(1, 2))
plot(dend, main = "original tree")
plot(flatten.dendrogram(dend), main = "Raised tree")

flip_leaves

Flip leaves

Description

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

Usage

flip_leaves(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.
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)
```

---

**FM_index**

*Calculating Fowlkes-Mallows Index*

**Description**

Calculating Fowlkes-Mallows index.

The `FM_index` function calculates the expectancy and variance of the FM Index under the null hypothesis of no relation.

**Usage**

```r
FM_index(
  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
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


https://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(cutree(hc1, k = 3), cutree(hc1, k = 3)) # 1 with EV

# 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),
    assume_sorted_vectors = TRUE
  ),
  FM_index(cutree(hc1, k = 3), cutree(hc1, k = 3),
    assume_sorted_vectors = TRUE
)
)```
# C code is 1.2-1.3 times faster.

```r
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
     ) # 1 again :)
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
Value

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

References


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

See Also

cor_bakers_gamma, 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(10000 permutation)")
abline(v = mean(FM_index_H0), col = 1, lty = 2)

# The permutation distribution is with a heavy right tail:
# Source of the skew functions is based on: library(psy)

skew <- function (x, na.rm = TRUE) {
  x <- na.omit(x)
  sum((x - mean(x))^3)/((length(x) * sd(x))^3)
}

skew(FM_index_H0) # 1.254
```
Calculating Fowlkes-Mallows index in R

Description

Calculating Fowlkes-Mallows index.

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 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.

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


https://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_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 of a dendrogram's branches

**Usage**

```r
get_branches_heights(
    dend,
    sort = TRUE,
    decreasing = FALSE,
    include_leaves = FALSE,
    ...)
```

**Arguments**

- **dend**: a dendrogram.
- **sort**: logical. Should the heights be sorted?
- **decreasing**: logical. Should the sort be increasing or decreasing? Not available for partial sorting.
- **include_leaves**: logical (FALSE). Should the output include the leaves value (0's).
- **...**: not used.
get_childrens_heights

Value

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

Examples

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

get_childrens_heights  Get height attributes from a dendrogram’s children

Description

Get height attributes from a dendrogram’s children nodes

Usage

```r
get_childrens_heights(dend, ...)
```

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

```r
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(dend, attribute, simplify = TRUE, ...)

Arguments
dend 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.

See Also
get_nodes_attr, nnodes, nleaves, assign_values_to_leaves_nodePar

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_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_branches_attr

get_leaves_branches_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_branches_attr

Get an attribute of the branches of a dendrogram's leaves

Description

This is helpful to get the attributes of branches of the leaves. For example, after we use color_branches, to get the colors of the labels to match (since getting the colors of branches to match those of the labels can be tricky). This is based on get_leaves_edgePar.

Usage

get_leaves_branches_attr(dend, attr = c("col", "lwd", "lty"), ...)

Arguments

dend a dendrogram object
attr character, the attr to get. Can be either "col", "lwd", or "lty".
... not used

Value

A vector with the dendrogram's leaves nodePar attribute

See Also

generate_nodes_attr, assign_values_to_leaves_nodePar, labels_colors get_nodes_attr, get_leaves_nodePar, get_leaves_edgePar

Examples

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

dend <- dend %>%
color_branches(k = 3) %>%
set("branches_lwd", c(2, 1, 2)) %>%
set("branches_lty", c(1, 2, 1))

plot(dend)
get_leaves_branches_attr(dend, "col")
get_leaves_branches_attr(dend, "lwd")
get_leaves_branches_attr(dend, "lty")

labels_colors(dend) <- get_leaves_branches_attr(dend, "col")
plot(dend)

---

get_leaves_branches_col

Get the colors of the branches of a dendrogram's leaves

Description

It is useful to get the colors of branches of the leaves, after we use `color_branches`, so to then match the colors of the labels to that of the branches (since getting the colors of branches to match those of the labels can be tricky). This is based on `get_leaves_branches_attr` which is based on `get_leaves_edgePar`.

TODO: The function `get_leaves_branches_col` may behave oddly when extracting colors with missing col attributes when the lwd attribute is available. This may result in a vector with the wrong length (with omitted NA values). This might need to be fixed in the future, and attention should be given to this case.

Usage

get_leaves_branches_col(dend, ...)

Arguments

dend a dendrogram object

Value

A vector with the dendrogram’s leaves’ branches’ colors

See Also

given_nodes_attr, assign_values_to_leaves_nodePar, labels_colors get_leaves_nodePar, get_leaves_edgePar, get_leaves_branches_attr

Examples

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

par(mfrow = c(1, 2), mar = c(5, 2, 1, 0))
dend <- dend %>%
**get_leaves_edgePar**

Get edgePar of dendrogram's leaves

**Description**

This is helpful to get the attributes of branches of the leaves. For example, after we use `color_branches`, to get the colors of the labels to match (since getting the colors of branches to match those of the labels can be tricky).

**Usage**

```r
get_leaves_edgePar(dend, simplify = FALSE, ...)```

**Arguments**

- `dend`: 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 edgePar attribute

**See Also**

- `get_nodes_attr`, `assign_values_to_leaves_nodePar`, `labels_colors` get_leaves_nodePar

**Examples**

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

# get_leaves_edgePar(dend) # error ;)
get_leaves_edgePar(dend)
dend <- color_branches(dend, k = 3)
get_leaves_edgePar(dend)
get_leaves_edgePar(dend, TRUE)
```
get_leaves_nodePar

Get nodePar of dendrogram’s leaves

Description

Get the nodePar attributes of dendrogram’s leaves (includes pch, color, and cex)

Usage

get_leaves_nodePar(dend, simplify = FALSE, ...)

Arguments

dend 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, assign_values_to_leaves_nodePar, labels_colors get_leaves_edgePar

Examples

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

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

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

plot(dend)
Get attributes of dendrogram's nodes

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

Arguments

dend 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

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

# 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_leaves_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:
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)
get_nodes_xy

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

Description

Get the x-y coordinates of a dendrogram’s nodes. Can be used to add text or images on the tree.

Usage

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

Arguments

dend       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 striped 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

Examples

### Not run:

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

Extract a list of \(k\) subdendrograms from a given dendrogram object

Description

Extracts a list (dendlist) of subdendrogram structures based on the cutree `cutree.dendrogram` function from a given dendrogram object. It can be useful in case we're interested in a visual investigation of specific clustering results.

Usage

```r
get_subdendrograms(dend, k, order_clusters_as_data = FALSE, ...)```

Arguments

- `dend`: a dendrogram object
- `k`: the number of subdendrograms that should be extracted
- `order_clusters_as_data`: passed to `cutree`, default is FALSE (while the cutree default is TRUE). The reason is since it's easier to look at the dendrogram plot and then get subtrees that are in the same order is in the plot/dendrogram object. This is in contrast to more traditional use of cutree, where it is used with the original order or rows from the data.
- `...`: parameters that should be passed to the cutree `cutree.dendrogram`
Value

A list of $k$ subdendrograms, based on the cutree function clustering clusters.

Examples

```r
# needed packages:
# install.packages(gplots)
# install.packages(viridis)
# install.packages(devtools)
# devtools::install_github('talgalili/dendextend') # dendextend from github

# define dendrogram object to play with:
dend <- iris[1:20, -5] %>%
  dist() %>%
  hclust() %>%
  as.dendrogram() %>%
  # set("labels_to_character") %>%
  color_branches(k = 5)
labels(dend) <- letters[1:20]
plot(dend)
dend_list <- get_subdendrograms(dend, 5)
lapply(dend_list, labels)
# [[1]]
# [1] "a" "b"
#
# [[2]]
# [1] "c" "d" "e" "f" "g"
#
# [[3]]
# [1] "h" "i"
#
# [[4]]
# [1] "j" "k" "l" "m"
#
# [[5]]
# [1] "n" "o" "p" "q" "r" "s" "t"

# define dendrogram object to play with:
dend <- iris[, -5] %>%
  dist() %>%
  hclust() %>%
  as.dendrogram() %>%
  set("labels_to_character") %>%
  color_branches(k = 5)
dend_list <- get_subdendrograms(dend, 5)

# Plotting the result
par(mfrow = c(2, 3))
plot(dend, main = "Original dendrogram")
sapply(dend_list, plot)
```
# plot a heatmap of only one of the sub dendrograms
par(mfrow = c(1, 1))
library(gplots)
sub_dend <- dend_list[[1]]  # get the sub dendrogram
# make sure of the size of the dend
nleaves(sub_dend)
length(order.dendrogram(sub_dend))
# get the subset of the data
subset_iris <- as.matrix(iris[order.dendrogram(sub_dend), -5])
# update the dendrogram's internal order so to not cause an error in heatmap.2
order.dendrogram(sub_dend) <- as.integer(rank(order.dendrogram(sub_dend)))
heatmap.2(subset_iris, Rowv = sub_dend, trace = "none", col = viridis::viridis(100))
horiz = FALSE,
theme = theme_dendro(),
offset_labels = 0,
na.rm = TRUE,
environment = parent.frame()
)
## S3 method for class 'dendrogram'
ggplot(data, ...)

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

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, x a ggdend class object (passed to ggplot.dendrogram or print.ggdend).
mapping (passed in ggplot.ggdend) Default list of aesthetic mappings to use for plot. If not specified, must be supplied in each layer added to the plot.
segments a logical (TRUE) if to plot the segments (branches).
labels a logical (TRUE) if to plot the labels.
nodes a logical (TRUE) if to plot the nodes (points).
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)
offset_labels a numeric value to offset the labels from the leaves
na.rm A logical (TRUE) to control removal of missing values. Passed to geom_line and geom_point
environment (passed in ggplot.ggdend) deprecated/ignored.

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.
GGDEND

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

Examples

## 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, 3, 1, 1, 2)) %>%
  set("labels_colors") %>%
  set("labels_cex", c(0.9, 1.2))
# plot the dend in usual "base" plotting engine:
plot(dend)
# Now let's do it in ggplot2 :) ggdl <- as.ggdend(dend)
library(ggplot2)
ggplot(ggdl) # reproducing the above plot in ggplot2 :)

# Triangle version:
plot(dend, type = "triangle")
hang.dendrogram

Hang dendrogram leaves

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: [https://stackoverflow.com/questions/17088136/plot-horizontal-dendrogram-with-hanging-leaves-r](https://stackoverflow.com/questions/17088136/plot-horizontal-dendrogram-with-hanging-leaves-r)

### Examples

```r
# 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)
```

---

### Description

Does a dendrogram has an edgePar/nodePar component?
has_component_in_attribute

Usage

has_component_in_attribute(dend, component, the_attrib = "edgePar", ...)

Arguments

dend a dendrogram object.
component a character value to be checked if exists in the tree. For edgePar the list: "col", "lty" and "lwd" (for the segments), "p.col", "p.lwd", and "p.lty" (for the polygon around the text) and "t.col" for the text color. For edgePar "pch", "cex", "col", "xpd", and/or "bg".
the_attrib A character of the attribute for which to check the existence of the component. Often either "edgePar" or "nodePar".
... ignored

Value

Logical. TRUE if such a component is defined somewhere in the tree, FALSE otherwise. If dend is not a dendrogram, the function will return FALSE.

See Also

generate_nodes_attr, set

Examples

dat <- iris[1:20, -5]
hca <- hclust(dist(dat))
hca2 <- hclust(dist(dat), method = "single")
dend <- as.dendrogram(hca)
dend2 <- as.dendrogram(hca2)

dend %>%
  set("branches_lwd", 2) %>%
  set("branches_lty", 2) %>%
  plot()
dend %>%
  set("branches_lwd", 2) %>%
  set("branches_lty", 2) %>%
  has_edgePar("lty")
dend %>%
  set("branches_lwd", 2) %>%
  has_edgePar("lty")
dend %>%
  set("branches_lwd", 2) %>%
  has_edgePar("lwd")

dend %>%
  set("branches_lwd", 2) %>%
  set("clear_branches") %>%
heights_per_k.dendrogram

Description

Which height will result in which k for a dendrogram. This helps with speeding up the cutree.dendrogram function.

Usage

heights_per_k.dendrogram(dend, ...)

Arguments

dend an 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 2 3 4
# 97.90023 57.41808 16.93594
# we do NOT have a height for k=2 because of the tree's structure.

## End(Not run)
highlight_branches_col

*Highlight a dendrogram’s branches heights via color and line-width*

**Description**

Highlights (update) the color (col) and/or line width (lwd) of each branch in a dendrogram based on it’s node’s height. This is a powerful pre-processing for a tanglegram plot of two dendrograms, as it emphasizes the topological structure of each tree (and hence, their similarity and differences). The colors are based on the viridis palate, and the line width is on the range of 1 to 10. These can be manually changed when using highlight_branches_col and highlight_branches_lwd respectively.

**Usage**

```r
highlight_branches_col(dend, values = rev(viridis(1000, end = 0.9)), ...)  
highlight_branches_lwd(dend, values = seq(1, 10, length.out = 1000), ...)  
highlight_branches(dend, type = c("col", "lwd"), ...)```

**Arguments**

- **dend**: a dendrogram tree (to be turned into a ggdend object)
- **values**: the gradient of values to be used for each branch. The colors are based on the viridis palate, and the line width is on the range of 1 to 10. These can be manually changed when using highlight_branches_col and highlight_branches_lwd respectively.
- **...**: Currently ignored.
- **type**: a character vector. Either "col", "lwd", or both. Based on whichever is chosen the dendrogram’s branches will be updated.

**Value**

A modified dendrogram, with colors/line-width in the branches that are proportional to each branch’s height (measured by its lower tip).

**See Also**

set, color_branches, get_branches_heights, viridis

**Examples**

```r
dat <- iris[1:20, -5]  
hca <- hclust(dist(dat))  
hca2 <- hclust(dist(dat), method = "single")```
```r
# highlight_branches_col

dend <- as.dendrogram(hca)
dend2 <- as.dendrogram(hca2)

par(mfrow = c(1, 3))
dend %>%
  highlight_branches_col() %>%
  plot(main = "Coloring branches")
dend %>%
  highlight_branches_lwd() %>%
  plot(main = "Emphasizing line-width")
dend %>%
  highlight_branches() %>%
  plot(main = "Emphasizing color and line-width")

library(viridis)
par(mfrow = c(1, 3))
dend %>%
  highlight_branches_col() %>%
  plot(main = "Coloring branches (default is reversed viridis)")
dend %>%
  highlight_branches_col(viridis(100)) %>%
  plot(main = "It is better to use lighter colors in the leaves")
dend %>%
  highlight_branches_col(rev(magma(1000))) %>%
  plot(main = "The magma color palette is also good")

dl <- dendlist(dend, dend2)
tanglegram(dl, sort = TRUE, common_subtrees_color_lines = FALSE, highlight_distinct_edges = FALSE, highlight_branches_lwd = FALSE)
tanglegram(dl)
tanglegram(dl, fast = TRUE)

dl <- dendlist(highlight_branches(dend), highlight_branches(dend2))
tanglegram(dl, sort = TRUE, common_subtrees_color_lines = FALSE, highlight_distinct_edges = FALSE)

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

dl <- dendlist(dend, dend2) %>% set("highlight_branches_col")
tanglegram(dl, sort = TRUE, common_subtrees_color_lines = FALSE, highlight_distinct_edges = FALSE)

# This is also useful for heatmaps
# --------------------------
# library(dendextend)

x <- as.matrix(datasets::mtcars)
Rowv <- x %>%
  dist() %>%
```
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(dend, ...)
```

```r
## S3 method for class 'dendrogram'
highlight_distinct_edges(
  dend,
  dend2,
  value = 2,
  edgePar = c("col", "lty", "lwd"),
  ...
)
```

```r
## S3 method for class 'dendlist'
highlight_distinct_edges(dend, ..., which = c(1L, 2L))
```
**highlight_distinct_edges**

**Arguments**

- **dend**: a dendrogram or **dendlist** to find unique edges in (to highlight)
- **dend2**: 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 "dend" 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, 100:105)
iris1 <- iris[ss, -5] %>%
  dist() %>%
  hclust(method = "single") %>%
  as.dendrogram()
iris2 <- iris[ss, -5] %>%
  dist() %>%
```
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

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

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)).

stop_if_out logical (default is FALSE). This default makes the function NOT stop if k of the locator is outside the range (this default is different than the behavior of the identify.hclust function - but it is nicer for the user.).

... 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"

**See Also**

identify.hclust, rect.hclust, order.dendrogram, cutree.dendrogram

**Examples**

```r
## Not run:
set.seed(23235)
ss <- sample(1:150, 10)
hc <- iris[ss, -5] %>%
  dist() %>%
  hclust()
dend <- hc %>% as.dendrogram()

plot(dend)
identify(dend)

plot(dend, horiz = TRUE)
identify(dend, horiz = TRUE)

## End(Not run)
```
**intersect_trees**

**Intersect trees**

**Description**

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

**Usage**

```r
intersect_trees(dend1, dend2, warn = dendextend_options("warn"), ...)
```

**Arguments**

- `dend1`: tree object (dendrogram/hclust/phylo)
- `dend2`: 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`

**Examples**

```r
hc <- hclust(dist(USArrests[1:5, ]), "ave")
dend <- as.dendrogram(hc)
labels(dend) <- 1:5
dend1 <- prune(dend, 1)
dend2 <- prune(dend, 5)
intersect_dend <- intersect_trees(dend1, dend2)

layout(matrix(c(1, 1, 2, 3, 4, 5), 3, 2, byrow = TRUE))
plot(dend, main = "Original tree")
plot(dend1, main = "Tree 1: \noriginal with label 1 pruned")
plot(dend2, main = "Tree 2: \noriginal with label 2 pruned")
plot(intersect_dend[[1]],
     main = "Tree 1 pruned
with the labels that intersected with those of Tree 2")
plot(intersect_dend[[2]],
     main = "Tree 2 pruned"
```
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

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: https://stackoverflow.com/questions/4562257/what-is-the-fastest-way-to-check-if-a-number-is-a-positive-natural-number-in-r

See Also

is.numeric, is.double, is.integer

Examples

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_null_list  Checks if the value is an empty list

Description
Checks if the value is an empty list(). Can be useful.

Usage
is_null_list(x)

Arguments
x  whatever object to check

Value
logical

Examples
# I can run this only if I'd make is_null_list exported
## Not run:
# TRUE:
is_null_list(list())  # FALSE
is_null_list(list(1))
is_null_list(1)
x <- list(1, list(), 123)
ss_list <- sapply(x, is_null_list)
x <- x[!ss_list]
x <- list(1, list(), 123)
ss_list <- sapply(x, is_null_list)
x <- list(list())
x

## End(Not run)

## Not run:
# error
is_null_list()

## End(Not run)
is_some_class

Is the object of some class

Description

Returns TRUE if some class (based on the name of the function).

Usage

is.hclust(x)

is.dendrogram(x)

is.phylo(x)

is.phylo(x)

is.dendlist(x)

is.dist(x)

Arguments

x an object.

Value

Returns TRUE if some class (based on the name of the function).

Examples

# TRUE:
is.dendlist(dendlist())
# FALSE
is.dendlist(1)
# TRUE:
is.dist(dist(mtcars))
# FALSE
is.dist(mtcars)

khan


Description

Usage

khan

Format

Khan is dataset containing the following:

- train: data.frame of 306 rows and 64 columns. The training dataset of 64 arrays and 306 gene expression values
- test: data.frame, of 306 rows and 25 columns. The test dataset of 25 arrays and 306 genes expression values
- gene.labels.imagesID: vector of 306 Image clone identifiers corresponding to the rownames of train and test.
- train.classes: factor with 4 levels "EWS", "BL-NHL", "NB" and "RMS", which correspond to the four groups in the train dataset
- test.classes: factor with 5 levels "EWS", "BL-NHL", "NB", "RMS" and "Norm" which correspond to the five groups in the test dataset
- annotation: data.frame of 306 rows and 8 columns. This table contains further gene annotation retrieved from SOURCE http://SOURCE.stanford.edu in May 2004. For each of the 306 genes, it contains:
  - CloneID: Image Clone ID
  - UGCluster: The Unigene cluster to which the gene is assigned
  - Symbol: The HUGO gene symbol
  - LLID: The locus ID
  - UGRepAcc: Nucleotide sequence accession number
  - LLRepProtAcc: Protein sequence accession number
  - Chromosome: Chromosome location
  - Cytoband: Cytoband location

Details

Khan et al., 2001 used cDNA microarrays containing 6567 clones of which 3789 were known genes and 2778 were ESTs to study the expression of genes in of four types of small round blue cell tumours of childhood (SRBCT). These were neuroblastoma (NB), rhabdomyosarcoma (RMS), Burkitt lymphoma, a subset of non-Hodgkin lymphoma (BL), and the Ewing family of tumours (EWS). Gene expression profiles from both tumour biopsy and cell line samples were obtained and are contained in this dataset. The dataset downloaded from the website contained the filtered dataset of 2308 gene expression profiles as described by Khan et al., 2001. This dataset is available from the http://bioinf.ucd.ie/people/aedin/R/.

In order to reduce the size of the MADE4 package, and produce small example datasets, the top 50 genes from the ends of 3 axes following bga were selected. This produced a reduced datasets of 306 genes.
Source

khan contains a filtered data of 2308 gene expression profiles as published and provided by Khan et al. (2001) on the supplementary web site to their publication OLD (site no longer found): https://research.nhgri.nih.gov/microarray/

The data was copied from the made4 package (https://www.bioconductor.org/packages/release/bioc/html/made4.html)

References


Examples

data(khan)
summary(khan)

labels<- "label" assignment operator

Description

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

Usage

labels(object, ...) <- value

## Default S3 replacement method:
labels(object, ...) <- 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

**Arguments**

- **object**: a variable name (possibly quoted) who’s 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.

**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): [https://stackoverflow.com/questions/4614223/how-to-have-the-following-work-labelsx-some-value](https://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

**Examples**

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

# get("labels<-")
### labels_cex

Get/assign `cex` to the labels of a dendrogram

#### Description

Retrieve/assign `cex` to the labels of a dendrogram

#### Usage

```r
labels_cex(dend, ...) <- value
```

#### Arguments

- **dend**: a dendrogram object
- **...**: not used
- **value**: a vector of `cex` to be used as new label's size for the dendrogram

#### Value

A vector with the dendrogram’s labels sizes (NULL if none are supplied).

#### Examples

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

# Defaults:
labels_cex(dend)
plot(dend)
```
# let's add some color:
labels_cex(dend) <- 1:3
labels_cex(dend)
plot(dend)

labels_cex(dend) <- 1
labels_cex(dend)
plot(dend)

---

labels_colors

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

Description

Retrieve/assign colors to the labels of a dendrogram. Note that usually dend objects come without any color assignment (and the output will be NULL, until colors are assigned).

Usage

labels_colors(dend, labels = TRUE, ...)

labels_col(dend, labels = TRUE, ...)

labels_colors(dend, ...) <- value

Arguments

dend 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).

See Also

cutree,dendrogram,hclust,color_labels,color_branches,assign_values_to_leaves_edgePar,get_leaves_branches_col
### Examples

```r
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)
plot(dend)

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

---

**Description**

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

**Usage**

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

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

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

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

Return the leaf Colors of a dendrogram

Description

The returned Colors will be in dendrogram order.

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 ="
which an integer (can have any number of elements). It indicates the elements in the dendlist to ladderize. If missing, it will ladderize all the dendrograms in the dendlist.

Value

A rotated tree object

See Also

ladderize, rev.dendrogram, rotate (dendextend), rotate (ape)

Examples

def <- USArrests[1:8, ] %>%
dist() %>%
hclust() %>%
as.dendrogram() %>%
set("labels_colors") %>%
set("branches_k_color", k = 5)
set.seed(123)
def <- shuffle(dend)
par(mfrow = c(1, 3))
def %>% plot(main = "Original")
def %>%
ladderize(TRUE) %>%
plot(main = "Right (default)")
def %>%
ladderize(FALSE) %>%
plot(main = "Left (rev of right)")
*lowest_common_branch*

**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

**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

```r
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

etanglement , tanglegram
Examples

```r
## 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, dend_changed)
# 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 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,
  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)
```
**min_depth**

*Find minimum/maximum depth of a dendrogram*

**Description**

As the name implies. This can also work for non-dendrogram nested lists.

**Usage**

```r
min_depth(dend, ...)
max_depth(dend, ...)
```

**Arguments**

- `dend` Any nested list object (including `dendrogram`).
- `...` unused at the moment.

**Value**

Integer, the (min/max) number of nodes from the root to the leaves

**Examples**

```r
hc <- hclust(dist(USArrests), "ave")
(dend1 <- as.dendrogram(hc)) # "print()" method
is.list(dend1)
is.list(dend1[[1]][[1]][[1]][[1]])
dend1[[1]][[1]][[1]][[1]]
plot(dend1)
min_depth(dend1)
max_depth(dend1)
```

---

**na_locf**

*Last Observation Carried Forward*

**Description**

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

**Usage**

```r
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)

## 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 :)

#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

## End(Not run)
Description
Counts the number of leaves in a tree (dendrogram or hclust).

Usage
nleaves(x, ...)
## Default S3 method:
nleaves(x, ...)

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

## S3 method for class 'dendlist'
nleaves(x, ...)

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

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

Arguments
x tree object (dendrogram/hclust/phylo,dendlist)
...
method a character scalar (default is "members"). If "order" than nleaves is based on length of order.dendrogram. If "members", than length is trusting what is written in the dendrogram’s root attr. "members" is about 4 times faster than "order".

Details
The idea for the name is from functions like ncol, and nrow.
Also, it is worth noting that the nleaves.dendrogram is based on order.dendrogram instead of labels.dendrogram since the first is MUCH faster than the later.
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

nrow, count_terminal_nodes

Examples

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

nleaves(dend) # 5
nleaves(hc) # 5

Description

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

Usage

nnodes(x, ...)

## Default S3 method:
nnodes(x, ...)

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

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

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

Arguments

x tree object (dendrogram or hclust)

... not used

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
noded_with_condition

See Also

nrow, 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?
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:

library(dendextend)

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

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

# 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)
Arguments

object  a variable name (possibly quoted) who's 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

#########################################################
# 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 :)
partition_leaves

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(dend, ...)

Arguments

dend 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

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

## 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()

partition_leaves(dend1)
partition_leaves(dend2)

## End(Not run)
```

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

```r
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,
...
)

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 make to plot.dendrogram, so that it would replace the need for this function.

See Also

plot.dendrogram, tanglegram

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 = F)
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

```
prune(dend, ...)
```

# Default S3 method:
prune(dend, ...)
## S3 method for class 'dendrogram'
prune(dend, leaves, reindex_dend = TRUE, ...)

## S3 method for class 'hclust'
prune(dend, leaves, ...)

## S3 method for class 'phylo'
prune(dend, ...)

## S3 method for class 'rpart'
prune(dend, ...)

### Arguments

- **dend**: 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.
- **reindex_dend**: logical (default is TRUE). If TRUE, the leaves of the new dendrograms include the rank of the old order.dendrogram. This insures that their values are just like the number of leaves. When FALSE, the values in the leaves is that of the original dendrogram. This is useful if pruning a dendrogram but then wanting to use order.dendrogram with the original values. When using prune.hclust, then reindex_dend is used by default since otherwise the as.hclust function would return an error.

### 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

```r
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_common_subtrees.dendlist

Prune trees to their common subtrees

Description

Prune trees to their common subtrees

Usage

prune_common_subtrees.dendlist(dend, ...)

Arguments

dend a dendlist of length two
... ignored

Value

A dendlist after pruning the labels to only include those that are part of common subtrees in both dendrograms.

See Also

common_subtrees_clusters

Examples

# NULL
prune_leaf  Trims one leaf from a dendrogram

**Description**

Trims (prunes) one leaf from a dendrogram.

**Usage**

`prune_leaf(dend, leaf_name, ...)`

**Arguments**

- `dend`  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")
```

---

pvclust_edges  Get Pvclust Edges Information

**Description**

Get pvclust edges information such as au and bp and return dataframe with proper sample labels. This function is useful when there are a lot of samples involved.

**Usage**

`pvclust_edges(pvclust_obj)`
The significant branches in a dendrogram, based on a pvclust object

pvclust_show_signif

The significant branches in a dendrogram, based on a pvclust object

Description

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

Usage

```
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`

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 = "lwd") %>%
  plot()
result %>% text()
result %>% pvrect(alpha = 0.95)
dend %>%
  pvclust_show_signif_gradient(result) %>%
```
pvclust_show_signif_gradient

Significance gradient of branches in a dendrogram (via pvclust)

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 = "lwd") %>%
  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)
```

---

**pvrect2**

*Draw Rectangles Around a Dendrogram’s Clusters with High/Low P-values*

**Description**

Draws rectangles around the branches of a dendrogram highlighting the corresponding clusters with low p-values. This is based on `pvrect`, allowing to draw the rects till the bottom of the labels.

**Usage**

`pvrect2()`
\begin{verbatim}
x, alpha = 0.95, pv = "au",
type = "geq",
max.only = TRUE,
border = 2,
xpd = TRUE,
lower_rect,
... 
)

Arguments

x          object of class pvclust.
alpha      threshold value for p-values., Default: 0.95
pv          character string which specifies the p-value to be used. It should be either of "au" or "bp", corresponding to AU p-value or BP value, respectively. See plot.pvclust for details., Default: 'au'
type       one of "geq", "leq", "gt" or "lt". If "geq" is specified, clusters with p-value greater than or equals the threshold given by "alpha" are returned or displayed. Likewise "leq" stands for lower than or equals, "gt" for greater than and "lt" for lower than the threshold value. The default is "geq"., Default: 'geq'
max.only   logical. If some of clusters with high/low p-values have inclusion relation, only the largest cluster is returned (or displayed) when max.only=TRUE., Default: TRUE
border     numeric value which specifies the color of borders of rectangles., Default: 2
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., Default: TRUE
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.
...
... passed to rect

See Also

pvrect, pvclust_show_signif

Examples

## Not run:

library(dendextend)
\end{verbatim}
library(pvclust)
data(lung) # 916 genes for 73 subjects
set.seed(13134)
result <- pvclust(lung[, 1:20], method.dist = "cor", method.hclust = "average", nboot = 10)

par(mar = c(9, 2.5, 2, 0))
dend <- as.dendrogram(result)
dend %>%
  pvclust_show_signif(result, signif_value = c(3, .5)) %>%
  pvclust_show_signif(result, signif_value = c("black", "grey"), show_type = "col") %>%
  plot(main = "Cluster dendrogram with AU/BP values (%)")

pvrect2(result, alpha = 0.95)
# getting the rects to the tips / above the labels
pvrect2(result, lower_rect = .15, border = 4, alpha = 0.95, lty = 2)
# Original function
# pvrect(result, alpha=0.95)
text(result, alpha = 0.95)

## End(Not run)

---

**raise.dendrogram**

*Raise the height of a dendrogram tree*

**Description**

Raise the height of nodes in a dendrogram tree.

**Usage**

```r
raise.dendrogram(dend, heiget_to_add, ...)
```

**Arguments**

- **dend**: dendrogram object
- **heiget_to_add**: how much height to add to all the branches (not leaves) in the dendrogram
- **...**: passed on (not used)

**Value**

A raised dendrogram

**Examples**

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

par(mfrow = c(1, 2))
plot(dend, main = "original tree")
plot(raise.dendrogram(dend, 100), main = "Raised tree")
```
**rank_branches**

**Rank 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, ...)
```

**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**

`get_branches_heights`, `get_childrens_heights`, `hang.dendrogram`, `tanglegram`

**Examples**

```r
d # 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(dend). 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

# 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))
**rank_values_with_clusters**

*Rank a vector based on clusters*

**Description**

Rank a vector based on clusters

**Usage**

```r
corr
```

**Arguments**

- `x` numeric vector
- `ignore0` logical (FALSE). If TRUE, will ignore the 0’s in the vector
- `...` not used

**Value**

an integer vector with the number of unique values as the number of uniques in the original vector. And the values are ranked from 1 (in the beginning of the vector) to the number of unique clusters.

**Examples**

```r
rank_values_with_clusters(c(1, 2, 3))
rank_values_with_clusters(c(1, 1, 3))
rank_values_with_clusters(c(0.1, 0.1, 3000))
rank_values_with_clusters(c(3, 1, 2))
rank_values_with_clusters(c(1, 3, 3, 3, 3, 3, 3, 4, 2, 2))
rank_values_with_clusters(c(3, 1, 2), ignore0 = TRUE)
rank_values_with_clusters(c(3, 1, 2), ignore0 = FALSE)
rank_values_with_clusters(c(3, 1, 0, 2), ignore0 = TRUE)
rank_values_with_clusters(c(3, 1, 0, 2), ignore0 = FALSE)
```

**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

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,
  upper_rect = 0,
  prop_k_height = 0.5,
  stop_if_out = FALSE,
  ...
)

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.
**rect.dendrogram**

- **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.
- **upper_rect**: a (scalar) value to add (default is 0) to how high should the upper part of the rect be.
- **prop_k_height**: a (scalar) value (should be between 0 to 1), indicating what proportion of the height our rect will be between the height needed for k and k+1 clustering.
- **stop_if_out**: logical (default is TRUE). This makes the function stop if k of the locator is outside the range (this default reproduces the behavior of the `rect.hclust` function).
- **...**: 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: [https://stackoverflow.com/questions/4720307/change-dendrogram-leaves](https://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)
rect.dendrogram(dend, 3, border = 4)
Vectorize(rect.dendrogram, "k")(dend, 4:5, border = 6)

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, 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)

# This had previously failed since it worked with a wrong k.

dend15 <- c(1:5) %>%
  dist() %>%
  hclust(method = "average") %>%
  as.dendrogram()
# dend15 <- c(1:25) %>% dist %>% hclust(method = "average") %>% as.dendrogram
dend15 %>%
  set("branches_k_color") %>%
  plot()
dend15 %>% rect.dendrogram(
  k = 3,
  border = 8, lty = 5, lwd = 2
)

---

**reindex_dend**

**Reindexing a pruned dendrogram**

### Description

*prune_leaf* does not update leaf indices as it prune leaves. As a result, some leaves of the pruned dendrogram may have leaf indices larger than the number of leaves in the pruned dendrogram, which may cause errors in downstream functions such as *as.hclust*.

This function re-indexes the leaves such that the leaf indices are no larger than the total number of leaves.

### Usage

reindex_dend(dend)
remove_branches_edgePar

Arguments

dend          dendrogram object

Value

A dendrogram object with the leaf reindexed

Examples

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

dend_pruned <- prune(dend, c("Alaska", "California"), reindex_dend = FALSE)

## A leave have an index larger than the number of leaves:
unlist(dend_pruned)
# [1] 4 3 1
#
#
dend_pruned_reindexed <- reindex_dend(dend_pruned)

## All leaf indices are no larger than the number of leaves:
unlist(dend_pruned_reindexed)
# [1] 3 2 1

## The dendrograms are equal:
all.equal(dend_pruned, dend_pruned_reindexed)
# 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(dend, ...)

Arguments

dend          a dendrogram object
...           not used

Value

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

Arguments

- `dend`: 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

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

```r
## 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

```r
remove_nodes_nodePar(dend, ...)
```

#### Arguments

- `dend`: a dendrogram object
- `...`: not used

#### Value

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

#### See Also

- `get_root_branches_attr`
- `assign_values_to_branches_edgePar`
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

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(1)
x
rlply(x, function(x) {
  x
}, add_notation = TRUE)

x <- list(1, 2, list(31))
x
rlply(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)
rlply(dend, function(x) {
  attr(x, "height")
})
rlply(dend, function(x) {
  attr(x, "members")
})

## End(Not run)
```

---

**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**

```r
rotate(x, ...)
```

**## Default S3 method:**

```r
rotate(x, order, ...)
```

**## S3 method for class 'dendrogram'**

```r
rotate(x, order, ...)
```

**## S3 method for class 'hclust'**

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

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

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

## S3 method for class 'dendlist'
sort(x, ...)

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

### 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.
- **phy**: a placeholder in case the user uses "phy ="
- **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)

### 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).

`rev.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**

```r
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**

```r
## 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.dendrogram

Sample a tree

Description

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

Duplicates a leaf in a tree. Useful for non-parametric bootstraping trees since it emulates what would have happened if the tree was constructed based on a row-sample with replacments 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

```
## 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)
```

---

seriate_dendrogram  

Rotates a dendrogram based on a seriation of a distance matrix

Description

Rotates a dendrogram so it confirms to an order of a provided distance object. The seriation algorithm is based on seriate, which tries to find a linear order for objects using data in form of a dissimilarity matrix (one mode data).

This is useful for heatmap visualization.

Usage

```
seriate_dendrogram(dend, x, method = c("OLO", "GW"), …)
```

Arguments

- **dend**: An object of class dendrogram or hclust
- **x**: a dist object.
method

- A character vector of either "OLO" or "GW": "OLO" - Optimal leaf ordering, optimizes the Hamiltonian path length that is restricted by the dendrogram structure - works in \(O(n^4)\) "GW" - Gruvaeus and Wainer heuristic to optimize the Hamiltonian path length that is restricted by the dendrogram structure

... parameters passed to \texttt{seriate}

Value

A dendrogram that is rotated based on the optimal ordering of the distance matrix

See Also

\texttt{rotate, seriate}

Examples

```r
## Not run:
# library(dendextend)
d <- dist(USArrests)
hc <- hclust(d, "ave")
dend <- as.dendrogram(hc)

heatmap(as.matrix(USArrests))

dend2 <- seriate_dendrogram(dend, d)
heatmap(as.matrix(USArrests), Rowv = dend)
## End(Not run)
```
"by_lists_branches_lwd", "by_lists_branches_lty", "highlight_branches_col",
"highlight_branches_lwd", "clear_branches",
"clear_leaves"),
value,
order_value = FALSE,
...
)

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

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

Arguments

- **dend**: 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 dendrogram tree. (the type of the value depends on the "what")
- **order_value**: logical. Default is FALSE. If TRUE, it means the order of the value is in the order of the data which produced the hclust or dendrogram - and will reorder the value to conform with the order of the labels in the dendrogram.
- **which**: an integer vector indicating, in the case "dend" is a dendlist, on which of the trees should the modification be performed. If missing - the change will be performed on all of dends 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, and by_lists_branches_col, by_lists_branches_lwd, by_lists_branches_lty have extra parameters: attr, TF_value. You can read more about them here: branches_attr_by_labels and branches_attr_by_lists

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). A leave is the terminal node of the tree.
• leaves_cex - set the leaves’ point size (assign_values_to_leaves_nodePar). For using this you
MUST also set leaves_pch, a good value to use is 19.
• leaves_col - set the leaves’ point color (assign_values_to_leaves_nodePar). For using this you
MUST also set leaves_pch, a good value to use is 19.
• leaves_bg - set the leaves’ point fill color (assign_values_to_leaves_nodePar). For using this
you MUST also set leaves_pch with values from 21-25.
• 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)
• nodes_bg - set the nodes’ point fill color (assign_values_to_nodes_nodePar). For using this
you MUST also set leaves_pch with values from 21-25.
• hang_leaves - hang the leaves (hang.dendrogram)
• branches_k_color - color the branches (color_branches), a k parameter needs to be supplied.
• branches_k_lty - updates the lwd of the branches (similar to branches_k_color), a k parameter
needs to be supplied.
• branches_col - set the color of branches (assign_values_to_branches_edgePar)
• 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)
• by_lists_branches_col - set the color of branches from the root of the tree down to (possibly
inner) nodes with specified members (branches_attr_by_lists)
• by_lists_branches_lwd - set the line width of branches from the root of the tree down to (pos-
sibly inner) nodes with specified members (branches_attr_by_lists)
• by_lists_branches_lty - set the line type of branches from the root of the tree down to (pos-
sibly inner) nodes with specified members (branches_attr_by_lists)
• highlight_branches_col - highlight branches color based on branches’ heights (highlight_branches_col)
• highlight_branches_lwd - highlight branches line-width based on branches’ heights (high-
light_branches_lwd)
• 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, branches_attr_by_lists, dendrogram
Examples

## Not run:

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

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

dend %>% labels()
dend %>%
  set("labels", 1:10) %>%
  labels()
dend %>%
  set("labels", 1:10) %>%
  plot()
dend %>%
  set("labels_color") %>%
  plot()
dend %>%
  set("labels_col", c(1, 2)) %>%
  plot() # Works also with partial matching :)
dend %>%
  set("labels_cex", c(1, 1.2)) %>%
  plot()
dend %>%
  set("leaves_pch", NA) %>%
  plot()
dend %>%
  set("leaves_pch", c(1:5)) %>%
  plot()
dend %>%
  set("leaves_pch", c(19, 19, NA)) %>%
  set("leaves_cex", c(1, 2)) %>%
  plot()
dend %>%
  set("leaves_pch", c(19, 19, NA)) %>%
  set("leaves_cex", c(1, 2)) %>%
  set("leaves_col", c(1, 1, 2, 2)) %>%
  plot()
dend %>%
  set("hang") %>%
  plot()

# using bg for leaves and nodes
set.seed(23235)
```
ss <- sample(1:150, 25)

# Getting the dend object
dend25 <- iris[ss, -5] %>%
dist() %>%
hclust() %>%
as.dendrogram()

dend25 %>%
  set("labels", 1:25) %>%
  set("nodes_pch", 21) %>% # set all nodes to be pch 21
  set("nodes_col", "darkred") %>%
  set("nodes_bg", "gold") %>%
  set("leaves_pch", 1:25) %>% # Change the leaves pch to move from 1 to 25
  set("leaves_col", "darkred") %>%
  set("leaves_bg", "gold") %>%
  plot(main = "pch 21 to 25 supports the\nnodes_bg and leaves_bg parameters")

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_k_lty", k = 3) %>%
  plot()
dend %>%
  set("branches_k_col", k = 3) %>%
  set("branches_k_lty", 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()
par(mfrow = c(1, 3))
dend %>%
  set("highlight_branches_col") %>%
plot()
par(mfrow = c(1, 1))
# 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_lwd", c(1:4, 7)) %>%
  set("by_labels_branches_lty", c(1:4, 7)) %>%
  set("labels", old_labels) %>%
plot()
#
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()
#---- using order_value
# This is probably not what you want, since cutree
# returns clusters in the order of the original data:
dend %>%
  set("labels_colors", cutree(dend, k = 3)) %>
  plot()
# The way to fix it, is to use order_value = TRUE
# so that value is assumed to be in the order of the data:
dend %>
  set("labels_colors", cutree(dend, k = 3), order_value = TRUE) %>
  plot()

#------------------
# Example for: by_lists_branches_col, by_lists_branches_lwd, by_lists_branches_lty
L <- list(c("109", "123", "126", "145"), "29", c("59", "67", "97"))
dend %>%
  set("by_lists_branches_col", L, TF_value = "blue") %>
  set("by_lists_branches_lwd", L, TF_value = 4) %>
  set("by_lists_branches_lty", L, TF_value = 3) %>
  plot()

#------------------
# A few dendlist examples:
dendlist(dend, dend) %>%
  set("hang") %>
  plot()
dendlist(dend, dend) %>%
  set("branches_k_col", k = 3) %>
  plot()
dendlist(dend, dend) %>%
  set("labels_col", c(1, 2)) %>
  plot()
dendlist(dend, dend) %>%
  set("hang") %>
  set("labels_col", c(1, 2), which = 1) %>
  set("branches_k_col", k = 3, which = 2) %>
  set("labels_cex", 1.2) %>
  plot()

#------------------
# example of modifying the dendrogram in a heatmap:
library(gplots)
data(mtcars)
x <- as.matrix(mtcars)
rc <- rainbow(nrow(x), start = 0, end = .3)
cc <- rainbow(ncol(x), start = 0, end = .3)
R code:

```r
## demonstrate the effect of row and column dendrogram options
#
Rowv_dend <- x %>%
  dist() %>%
  hclust() %>%
  as.dendrogram() %>%
  set("branches_k", k = 3) %>%
  set("branches_lwd", 2) %>%
  ladderize() # rotate_DendSer
Colv_dend <- t(x) %>%
  dist() %>%
  hclust() %>%
  as.dendrogram() %>%
  set("branches_k", k = 3) %>%
  set("branches_lwd", 2) %>%
  ladderize() # rotate_DendSer
heatmap.2(x, Rowv = Rowv_dend, Colv = Colv_dend)
## End(Not run)
```

`set_labels` documentation:

**Description**

Convenience functions for updating the labels of a dendrogram. `set_labels` and `place_labels` differs in their assumption about the order of the labels. * `set_labels` assumes the labels are in the same order as that of the labels in the dendrogram. * `place_labels` assumes the labels has the same order as that of the items in the original data matrix. This is useful for renaming labels based on some other columns in the data matrix.

**Usage**

```r
set_labels(dend, labels, ...)
```

**Arguments**

- `dend`: a `dendrogram` object
- `labels`: A vector of values to insert in the labels of a dendrogram.
- `...`: Currently ignored.

**Value**

The updated `dendrogram` object

**Author(s)**

Tal Galili, Garrett Grolemund
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(dend, ...)

## Default S3 method:
shuffle(dend, ...)

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

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

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

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

Arguments

dend 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
sort_2_clusters_vectors

Sorts two clusters vector by their names

Description

Sorts two clusters vector by their names and returns a list with the sorted vectors.

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, corresponding to the two clustering vectors.
sort_dist_mat

Sorts a distance matrix by rows and columns names

Description

Sorts a distance matrix by the names of the 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_cophenetic
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

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"
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)
```

tanglegram  Tanglegram plot

Description

Plots a tanglegram plot of a side by side trees.

Usage

tanglegram(dend1, ...)

## Default S3 method:
tanglegram(dend1, ...)

## S3 method for class 'hclust'
tanglegram(dend1, ...)

## S3 method for class 'phylo'
tanglegram(dend1, ...)

## S3 method for class 'dendlist'
tanglegram(
dend1,
which = c(1L, 2L),
main_left,
main_right,
just_one = TRUE,
```
## S3 method for class 'dendrogram'

tanglegram(
  dend1,
  dend2,
  sort = FALSE,
  color_lines,
  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",
  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,
  common_subtrees_color_lines = TRUE,
  common_subtrees_color_lines_default_single_leaf_color = "grey",
  common_subtrees_color_branches = FALSE,
  highlight_branches_col = FALSE,
  highlight_branches_lwd = TRUE,
  faster = FALSE,
  just_one = TRUE,
...
dendbackback(
  dend1,
  dend2,
  sort = FALSE,
  color_lines,
  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",
  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,
  common_subtrees_color_lines = TRUE,
  common_subtrees_color_lines_default_single_leaf_color = "grey",
  common_subtrees_color_branches = FALSE,
  highlight_branches_col = FALSE,
  highlight_branches_lwd = TRUE,
  faster = FALSE,
  just_one = TRUE,
  ...
)
Arguments

dend1 tree object (dendrogram/dendlist/hclust/phylo), plotted on the left
... not used.
which an integer vector of length 2, indicating which of the trees in the dendlist object
should be plotted
main_left Character. Title of the left dendrogram.
main_right Character. Title of the right dendrogram.
just_one logical (TRUE). If FALSE, it means at least two tanglegrams will be plotted on
the same page and so layout is not passed. See: https://stackoverflow.com/
q/39784746/4137985
dend2 tree object (dendrogram/hclust/phylo), plotted on the right
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). The colors in
the vector are applied on the lines from the bottom up.
lwd width of the lines connecting the labels. (default is 3.5)
edge.lwd width of the dendrograms lines. Default is NULL. If set, then it switches ‘highlight_branches_lwd’ to FALSE. If you want thicker lines which reflect the height,
please use highlight_branches_lwd on the dendrograms/dendlist.
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 if
parameter just_one is TRUE. 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 dis-
tance between the dendrograms and the connecting lines.
margin_outer margin_bottom the number of lines of margin to be specified on the outer dis-
tance 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 positive).
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.
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). (notice that this can be slow on large trees)
   This parameter will automatically be turned off if the tree already comes with a "lty" edgePar (this is checked using has_edgePar). A "lty" can be removed by using set("clear_branches"), by removing all of the edgePar parameters of the dendrogram.
common_subtrees_color_lines  logical (default is TRUE). color the connecting line based on the common subtrees of both dends. This only works if (notice that this can be slow on large trees)
common_subtrees_color_lines_default_single_leaf_color  When representing edges between common subtrees (i.e. common_subtrees_color_branches = TRUE), this parameter sets the color of edges for subtrees that are NOT common. Default is "grey"
common_subtrees_color_branches  logical (default is FALSE). Color the branches of both dends based on the common subtrees. (notice that this can be slow on large trees) This is FALSE by default since it will override the colors of the existing tree.
highlight_branches_col
logical (default is FALSE). Should highlight_branches_col be used on the dendrograms.

This parameter will automatically be turned off if the tree already comes with a "col" edgePar (this is checked using has_edgePar). A "lty" can be removed by using set("clear_branches"), by removing all of the edgePar parameters of the dendrogram.

highlight_branches_lwd
logical (default is TRUE). Should highlight_branches_lwd be used on the dendrograms.

This parameter will automatically be turned off if the tree already comes with a "lwd" edgePar (this is checked using has_edgePar). A "lty" can be removed by using set("clear_branches"), by removing all of the edgePar parameters of the dendrogram.

closer
logical (FALSE). If TRUE, it overrides some other parameters to have them turned off so that the plotting will go a tiny bit faster.

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: https://stackoverflow.com/questions/12456768/duelling-dendrograms-in-r-placing-dendrograms-back-to-back-

As far as I could tell, this code was originally inspired by Dylan Beaudette function dueling.dendrograms from the sharpshootR package: https://CRAN.R-project.org/package=sharpshootR tanglegram

See Also
remove_leaves_nodePar, plot_horiz.dendrogram, rank_branches, hang.dendrogram

Examples
## Not run:
set.seed(23235)
ss <- sample(1:150, 10)
dend1 <- iris[ss, -5] %>%
dist() %>%

```r
hclust("com") %>%
  as.dendrogram()
dend2 <- iris[ss, -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)

########
## Nice example of some colored trees

## see the coloring of common sub trees:
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)
# dend12 %>% untangle %>% tanglegram
dend12 %>% tanglegram(common_subtrees_color_branches = TRUE)

set.seed(22133513)
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)
# dend12 %>% untangle %>% tanglegram
dend12 %>% tanglegram(common_subtrees_color_branches = TRUE)
dend12 %>% tanglegram()

## End(Not run)

---

**theme_dendro**  
Creating 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**

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 mixter 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
unbranch

**Description**

unbranch trees and merges the subtree to the parent node.

**Usage**

```r
unbranch(dend, ...)
```

## Default S3 method:

```r
unbranch(dend, ...)
```

## S3 method for class 'dendrogram'

```r
unbranch(dend, branch_becoming_root = 1, new_root_height, ...)
```

## S3 method for class 'hclust'

```r
unbranch(dend, branch_becoming_root = 1, new_root_height, ...)
```

## S3 method for class 'phylo'

```r
unbranch(dend, ...)
```

**Arguments**

- `dend`: a dendrogram (or 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**

```r
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 an entire dendrogram tree**

**Description**

unclass all the nodes in a dendrogram tree. (Helps in cases when a dendrapply function was used wrongly)

**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...
```
untangle

untangle dendrograms

Description

One untangle function to rule them all.
This function untangles dendrogram lists (dendlist), using various heuristics.

Usage

untangle(dend1, ...)

## Default S3 method:
untangle(dend1, ...)

untangle_labels(dend1, dend2, ...)

## S3 method for class 'dendrogram'
untangle(
dend1,
dend2,
method = c("labels", "ladderize", "random", "step1side", "step2side", "DendSer"),
...
)

## S3 method for class 'dendlist'
untangle(
dend1,
method = c("labels", "ladderize", "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)
method a character indicating the type of untangle heuristic to use.
which an integer vector of length 2, indicating which of the trees in the dendlist object should be plotted

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.
If the dendlist was originally larger than 2, it will return the original dendlist but with the relevant trees properly rotate.

Author(s)
Tal Galili

See Also
tanglegram, untangle_random_search, untangle_step_rotate_1side, untangle_step_rotate_2side, untangle_DendSer, 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)
dend12 %>% tanglegram()

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, ...)

Arguments

dend An object of class dendlist

... NOT USED

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)
```
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.

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
entanglement(dend12[[1]], dend12[[2]], L = 2) # 0.0998

## End(Not run)
```
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

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")
  If k_seq is not NULL, then it overrides "direction".
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.

... not used

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
untangle_step_rotate_2side

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.

Similar to "step1side", one tree is held fixed and the other tree is rotated. 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. Once finished the rotated tree is held fixed, and the fixed tree is now rotated. This continues until a local optimal solution is reached.

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.
which_leaf

Which node is a leaf?

Description

Gives a vector as the number of nodes (nnodes), which gives a TRUE when a node is a leaf.

Usage

which_leaf(dend, ...)

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?
dend12_corrected <- untangle_step_rotate_2side(dend1, dend2)
tanglegram(dend12_corrected[[1]], dend12_corrected[[2]], margin_inner = 6.5) # Better...
entanglement(dend12_corrected[[1]], dend12_corrected[[2]], L = 2) # 0.0045

# best combination:
dend12_corrected_1 <- untangle_random_search(dend1, dend2)
dend12_corrected_2 <- untangle_step_rotate_2side(dend12_corrected_1[[1]], dend12_corrected_1[[2]])
tanglegram(dend12_corrected_2[[1]], dend12_corrected_2[[2]], margin_inner = 6.5) # Better...
entanglement(dend12_corrected_2[[1]], dend12_corrected_2[[2]], L = 2) # 0 - PERFECT.

## End(Not run)
which_node

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()
  dend %>% plot()
  which_leaf(dend)
  
  ## End(Not run)

which_node  Which node id is common to a group of labels

Description
  This function identifies which edge(s) in a tree has group of labels ("tips") in common. By default it only returns the edge (node) with the highest id.

Usage
  which_node(dend, labels, max_id = TRUE, ...)

Arguments
  dend         a dendrogram dend
  labels       a character vector of labels from the tree
  max_id       logical (TRUE) - if to return only the max id
  ...          ignored.
which_node

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)))
Index

* clustering
dendextend-package, 5

* datasets
khan, 125

* dendrogram
dendextend-package, 5

* package
dendextend-package, 5

* visualization
dendextend-package, 5

all.equal, 7
all.equal.dendlist
(all.equal.dendrogram), 6
all.equal.dendrogram, 6
all.equal.list, 6
all.equal.phylo, 7
all_couple_rotations_at_k, 7, 202, 204
all_unique, 9
apply, 185
as.dendlist, 10
as.dendrogram, 11
as.ggdend (ggdend), 109
as.hclust, 11, 18, 148
as.hclust.dendrogram, 11
as.phylo, 11
as.phylo.dendrogram, 11
as.phylo.hclust, 11
as_hclust_fixed, 18
assign_dendextend_options, 12
assign_values_to_branches_edgePar, 12, 76, 119, 164, 165, 175
assign_values_to_leaves_edgePar, 14, 51, 130
assign_values_to_leaves_nodePar, 15, 17, 98–102, 164, 174, 175
assign_values_to_nodes_nodePar, 17, 175
attr, 107, 139
bakers_gamma_for_2_k_matrix, 19, 56

Bk, 20, 23, 25, 62
Bk_permutations, 21, 25
Bk_plot, 20, 23
branches_attr_by_clusters, 26, 40, 43, 46, 47, 206
branches_attr_by_labels, 28, 29, 32, 142, 174, 175, 206
branches_attr_by_lists, 31, 174, 175
branches_color (color_branches), 46
circlize_dendrogram, 32
circos.dendrogram, 33
circos.track, 32
clip, 161
collapse_branch, 36
collapse_labels, 37
color_branches, 46, 51, 53, 99–101, 116, 130, 133, 175
color_clusters, 47
color_labels, 47, 50, 130, 174
color_unique_labels, 52
colored_bars, 38
colored_dots, 42
colour_branches (color_branches), 46
colour_labels (color_labels), 50
common_subtrees_clusters, 53, 149
cophenetic, 54, 59, 60, 74, 77, 86, 170, 198
cor, 59
cor.dendlist, 54, 58, 62
cor_bakers_gamma, 19, 20, 54, 55, 60, 91, 93, 95, 134
cor_common_nodes, 54, 58
cor_cophenetic, 54, 57, 59, 77, 184
cor_FM_index, 54, 61
count_terminal_nodes, 62, 140, 141
cut_lower_fun, 71
cutree, 27, 46, 47, 50, 51, 56, 63, 65, 67–70, 107, 130, 185
cutree.dendrogram, 8, 47, 71, 107, 108, 115, 121, 161

cutree_1h.dendrogram, 65, 67, 70
cutree_1k.dendrogram, 65, 69
cutreeDynamic, 27, 28
data.frame, 126
dend_diff, 75, 118
dend_expend, 76
dendbackback (tanglegram), 186
dendextend (dendextend-package), 5
dendextend-package, 5
dendextend_cut_lower_fun
  (cut_lower_fun), 71
dendextend_heights_per_k.dendrogram
  (heights_per_k.dendrogram), 115
dendextend_options, 72
dendlist, 6, 54, 73, 75, 76, 78, 107, 119, 122,
  132, 139, 149, 174, 191, 197, 198
dendrapply, 130
dendro_data, 111
dendrogram, 5, 6, 33, 47, 51, 53, 71, 76, 79,
  86, 110, 111, 114, 116, 120, 130,
  132, 137, 145, 160, 172, 174, 175,
  180–182
DendSer, 74, 75, 170, 198
DendSer. dendrogram, 74, 75, 170, 198
dist, 47, 59, 77, 78, 128, 172, 184
dist.dendlist, 76, 77, 79, 119, 145
dist.multiPhylo, 78
dist.topo, 78
dist_long, 80
distinct.edges, 76, 79, 119
distinct_edges, 58, 76, 78, 79, 79, 119, 145
drop.tip, 148
duplicate_leaf, 80, 172
entanglement, 8, 82, 89, 134–136, 181, 182,
  197, 199–204
fac2num, 84, 185
factor, 126
find_dend (dend expend), 76
find_dendrogram, 85
find_k, 86
fix_members_attr.dendrogram, 81, 87, 171
flatten.dendrogram, 88
flip_leaves, 8, 89, 202, 204
FM_index, 20, 23, 25, 62, 90, 93
FM_index_permutation, 92
FM_indexR, 93, 94
geom_line, 110
geom_point, 110
get_branches_heights, 96, 97, 116, 157
get_childrens_heights, 97, 157
get_leaves_attr, 14, 16, 17, 28, 30, 98, 104,
  142, 164
get_leaves_branches_attr, 99, 100
get_leaves_branches_col, 47, 100, 130
get_leaves_edgePar, 99, 100, 101, 102
get_leaves_nodePar, 99–101, 102, 111
get_nodes_attr, 98–102, 103, 105, 111, 114,
  206
get_nodes_xy, 105
get_root_branches_attr, 13, 106, 164, 165
get_subdendrograms, 107
ggdend, 109, 193
ggdendrogram, 111
ggplot, 111
ggplot.dendrogram (ggdend), 109

ggplot.ggdendrogram (ggdend), 109

hang.dendrogram, 112, 157, 175, 191
has_component_in_attribute, 113
has_edgePar, 190, 191
has_edgePar
  (has component_in_attribute), 113
has_nodePar
  (has component_in_attribute), 113
hclust, 5, 47, 51, 65, 68, 70, 77, 128, 130,
  172, 174, 181, 182
heights_per_k.dendrogram, 8, 115
highlight_branches
  (highlight branches_col), 116
highlight_branches_col, 116, 175, 191
highlight_branches_lwd, 175, 189, 191
highlight_branches_lwd
  (highlight branches_col), 116
highlight_distinct_edges, 76, 79, 118,
  119, 145
identical, 7
identify.dendrogram, 120
identify.hclust, 120, 121
intersect, 122
intersect_trees, 54, 55, 59, 122
is.dendlist(is_some_class), 125
is.dendrogram(is_some_class), 125
is.dist(is_some_class), 125
is.double, 123
is.hclust(is_some_class), 125
is.integer, 123
is.leaf, 205
is.natural.number, 123
is.numeric, 123
is.phylo(is_some_class), 125
is.null_list, 124
is_some_class, 125
khan, 125
labels, 47, 71, 122, 128, 171, 181
labels.hclust(labels<->), 127
labels.phylo(labels<->), 127
labels<->.dendrogram, 174, 175
labels_cex, 129
labels_col(labels_colors), 130
labels_colors, 47, 51, 99–102, 130
labels_colors<->.175
labels_colors<-(labels_colors), 130
ladderize, 131, 132, 168, 169
layout, 189
leaf_Colors, 132
leaf_colors(leaf_Colors), 132
lowest_common_branch, 133
match_order_by_labels, 8, 56, 83, 89, 134, 135, 136, 200, 202, 204
match_order.dendrogram_by_old_order, 135
max_depth(min_depth), 137
min_depth, 137
multi2di, 36
na.locf, 138
na_locf, 137
names, 84
nleaves, 28, 30, 98, 104, 105, 139, 141, 142, 195
nnodes, 28, 30, 98, 104, 105, 140, 142, 204–206
noded_with_condition, 30, 141, 175, 205, 206
nrow, 140, 141
order, 168, 169
order.dendrogram, 27, 46, 121, 139, 143, 144, 148, 161, 168, 169
order.dendrogram<-, 142
order.hclust, 143
pam, 86, 87
pamk, 86, 87
par, 39, 42, 161
partition.leaves, 145
partition_leaves, 144
phylo, 132, 181, 182
phylo.diff, 76
place_labels(set_labels), 180
plot, 25, 146
plot.dendlist(dendlist), 73
plot.dendrogram, 75, 105, 110, 146, 147
plot.find_k(find_k), 86
plot_horiz.dendrogram, 145, 191
plotDendroAndColors, 28, 40, 43
plotHclustColors, 39, 43
prepare.ggdend(ggdend), 109
print.ggdend(ggdend), 109
prune, 122, 147, 150, 158
prune_common_subtrees.dendlist, 149
prune_leaf, 148, 150
pvclust_edges, 150
pvclust_show_signif, 151, 152, 153, 155
pvclust_show_signif_gradient, 152, 153, 155
pvrect, 154, 155
pvrect2, 154
rainbow, 46, 51
rainbow_hcl, 46, 51
raise.dendrogram, 156
rank, 158
rank_branches, 157, 191
rank_order.dendrogram, 81, 158, 171
rank_values_with_clusters, 159
rect, 155, 160
rect.dendrogram, 159
rect.hclust, 121, 161
reindex_dend, 162
remove_branches_edgePar, 163, 175
remove_leaves_nodePar, 164, 175, 191
remove_nodes_nodePar, 165
rev.dendrogram, 132, 168, 169
rev.hclust (rotate), 167
rlply, 166
rotate, 132, 167, 168, 169, 173, 181, 182
rotate.dendrogram, 35
rotate_DendSer, 75, 170, 170, 198
rownames, 47
sample, 171, 172
sample.dendrogram, 171
seriate, 172, 173
seriate_dendrogram, 172
set, 114, 116, 173, 181
set_labels, 180
shuffle, 181
silhouette, 86, 87
simplify2array, 103
slice, 47, 133
sort, 185
sort.dendlist (rotate), 167
sort.dendrogram (rotate), 167
sort.hclust (rotate), 167
sort_2_clusters_vectors, 183
sort_dist_mat, 184
sort_levels_values, 185
stats, 5
tanglegram, 8, 53, 76, 79, 83, 89, 116, 118,
  119, 134, 136, 145, 147, 157, 181,
  182, 186, 197, 200–204
tapply, 27
theme_dendro, 110, 193
treedist, 78
unbranch, 194
unclass_dend, 195
unique, 9
unroot, 194
untangle, 196
untangle_DendSer, 75, 170, 197, 198, 198
untangle_labels (untangle), 196
untangle_random_search, 197, 199
untangle_step_rotate_1side, 197, 201,
  204
untangle_step_rotate_2side, 197, 202,
  202
viridis, 116
which.edge, 206