Doing More with Dendrograms: The dendextend R Package

Tal Galili
Tel-Aviv University

Yoav Benjamini
Tel-Aviv University

Abstract

The **dendextend** package extends the dendrogram objects in R.

The paper gives a detailed exposition of both the internal structure of the package and the provided user interfaces.

Keywords: Dendrogram, hclust, heirarchical clustering, visualization, tanglegram, R.

1. Introduction

1.1. The dendrogram object

The `dendrogram` class provides general functions for handling tree-like structures in R (R Development Core Team 2013). It is intended as a replacement for similar functions in hierarchical clustering and classification/regression trees, such that all of these can use the same engine for plotting or cutting trees.

A dendrogram object represents a tree as a nested `list` object, with various attributes.

Dendrogram has several useful methods bundled with R:

```
methods(class = "dendrogram")

##  [1] [[.dendrogram*         as.hclust.dendrogram*  cophenetic.dendrogram*
##  [4] cut.dendrogram*        head.dendrogram*       labels.dendrogram*
##  [7] labels<-.dendrogram*   merge.dendrogram*      nleaves.dendrogram*
## [10] plot.dendrogram*       print.dendrogram*      reorder.dendrogram*
## [13] rev.dendrogram*        rotate.dendrogram*     sort.dendrogram*
## [16] str.dendrogram*        trim.dendrogram*       unroot.dendrogram*
##
##    Non-visible functions are asterisked
```

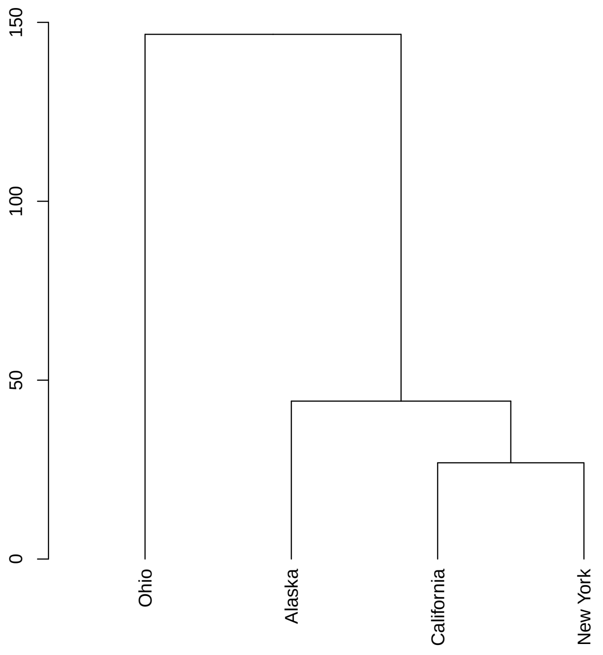
For example, let's create a dendrogram object based on an heirarchical clustering of 4 states in the U.S.:

```
# our data:
data(USArrests)
```

```
US_data <- USArrests[c(2, 5, 32, 35), ]
print(US_data)

##            Murder Assault UrbanPop Rape
## Alaska       10.0     263       48 44.5
## California    9.0     276       91 40.6
## New York     11.1     254       86 26.1
## Ohio          7.3     120       75 21.4


hc <- hclust(dist(US_data), "ave")  # create an heirarchical clustering object
dend <- as.dendrogram(hc)
```

Here are some examples for some dendrogram methods:

```
print(dend)

## 'dendrogram' with 2 branches and 4 members total, at height 146.7

labels(dend)

## [1] "Ohio"       "Alaska"     "California" "New York"

str(dend)

## --[dendrogram w/ 2 branches and 4 members at h = 147]
##   |--leaf "Ohio"
##   `--[dendrogram w/ 2 branches and 3 members at h = 44.1]
##      |--leaf "Alaska"
##      `--[dendrogram w/ 2 branches and 2 members at h = 26.9]
##         |--leaf "California"
##         `--leaf "New York"

str(dend[[2]])  # looking at one branch of the dendrogram

## --[dendrogram w/ 2 branches and 3 members at h = 44.1]
##   |--leaf "Alaska"
##   `--[dendrogram w/ 2 branches and 2 members at h = 26.9]
##      |--leaf "California"
##      `--leaf "New York"

plot(dend)
```



You might notice how the order of the items (leaves/terminal nodes) of the dendrogram is different than their order in the table. In order to re-order the rows in the data-table to have the same order as the items in the dendrogram, we can use the `order.dendrogram` function:

```
(new_order <- order.dendrogram(dend))

## [1] 4 1 2 3

# the order of the original items to have them be at the same order as
# they assume in the dendrogram
print(US_data[new_order, ])

##            Murder Assault UrbanPop Rape
## Ohio          7.3     120       75 21.4
## Alaska       10.0     263       48 44.5
## California    9.0     276       91 40.6
## New York     11.1     254       86 26.1
```

In order to see what our dendrogram (`list`) object includes, we need to use the `unclass` function, which will strip away the class atrribute and will allow us to print the list as is, without going through the `print.dendrorgam` method. We can see how each node in the dendrogram/list object has the following (self explaining) attributes:

```
str(unclass(dend))

## List of 2
##  $ : atomic [1:1] 4
##   ..- attr(*, "members")= int 1
##   ..- attr(*, "height")= num 0
##   ..- attr(*, "label")= chr "Ohio"
##   ..- attr(*, "leaf")= logi TRUE
##  $ :List of 2
##   ..$ : atomic [1:1] 1
##   .. ..- attr(*, "members")= int 1
##   .. ..- attr(*, "height")= num 0
##   .. ..- attr(*, "label")= chr "Alaska"
##   .. ..- attr(*, "leaf")= logi TRUE
##   ..$ :List of 2
##   .. ..$ : atomic [1:1] 2
##   .. .. ..- attr(*, "label")= chr "California"
##   .. .. ..- attr(*, "members")= int 1
##   .. .. ..- attr(*, "height")= num 0
##   .. .. ..- attr(*, "leaf")= logi TRUE
##   .. ..$ : atomic [1:1] 3
##   .. .. ..- attr(*, "label")= chr "New York"
##   .. .. ..- attr(*, "members")= int 1
##   .. .. ..- attr(*, "height")= num 0
##   .. .. ..- attr(*, "leaf")= logi TRUE
##   .. ..- attr(*, "members")= int 2
##   .. ..- attr(*, "midpoint")= num 0.5
##   .. ..- attr(*, "height")= num 26.9
##   ..- attr(*, "members")= int 3
##   ..- attr(*, "midpoint")= num 0.75
##   ..- attr(*, "height")= num 44.1
##  - attr(*, "members")= int 4
##  - attr(*, "midpoint")= num 0.875
##  - attr(*, "height")= num 147
```

Also, terminal nodes also has the ”leaf” attribute (set to TRUE).

```
names(attributes(dend)[-4])

## [1] "members"  "midpoint" "height"
```

A very important function is `dendrapply`. It applies some function recursively to each node of a dendrogram. It is often used for adjusting attributes of the object, or extracting something from it.

One current ”feature” with this function is that just sending a dendrogram through it will return it with each of its nodes becoming of class ”dendrogram”. Notice the use of the `unclass_dend` function. Example:

```
# dendrapply(dend, unclass) # in case the
itself <- function(x) x
dend_from_dendrapply <- dendrapply(dend, itself)

# here we must first use unclass since '[[]]' inherits its class to the
# output:
class(unclass(dend)[[2]])

## [1] "list"

class(unclass(dend_from_dendrapply)[[2]])

## [1] "dendrogram"

class(unclass_dend(dend_from_dendrapply)[[2]])  # the new uncless_dend solves it.

## [1] "list"
```

1.2. Motivation for creating `dendextend`

The `dendrogram` object has several **advantages**:

1. `dendrogram` objects are simply list R objects. This makes their structure very simple to understand by R users.
2. `dendrogram` objects has various methods and functions for using them in R.
3. `dendrogram` objects are relatively simple to manipulte and extend.
4. Other tree objects (such as *hclust*, and objects from the *ape* package) include an *as.dendrogram* method for converting their objects into a dendrogram.

However, even with all of its advantages, the `dendrogram` class in R still lacks various basic features.

The `dendextend` package aims at filling some gaps in base R, by extending the available functions for dendrogram manipulation, statistical analysis, and visualization.

This vignettes Provides a step-by-step description of the functionality provided by the `dendextend` package.

1.3. Installing `dendextend`

To install the stable version from CRAN use:

```
install.packages("dendextend")  # not yet available from CRAN
```

To install the GitHub version use:

```
if (!require("devtools")) install.packages("devtools")
require("devtools")
install_github("dendextend", "talgalili")
```

2. Tree labels (extraction, assignment, length)

2.1. labels in base R

In base R, the `labels` function is intended to find/extract a suitable set of labels from an object for use in printing or plotting, for example. By default, it uses the `names` and `dimnames` functions.

What base R `labels` function is mising is assignment. In the next few examples we will go through different examples of what the `dendextend` package offers for various objects.

Credits: These assignment functions were originally written by Gavin Simpson (in a post on (stackoverflow)), and adopted/adjusted to this package by Tal Galili. Some modifactions were inspired by Gregory Jefferis's code from the **dendroextras** package.

2.2. labels for vectors and matrixes

In base R, for vectors, labels gives the `names` of the object. And if these are missing, then `labels` will give the vector itself as a character vector:

```
x <- 1:3
names(x)  # this vector has no names

## NULL

labels(x)  # this vector has no labels

## [1] "1" "2" "3"
```

Assignment to names is available in base R and works as follows:

```
x <- 1:3
names(x) <- letters[1:3]  # assignment for names is in base R
# both names and labels will give the same result:
names(x)

## [1] "a" "b" "c"
```

```
labels(x)

## [1] "a" "b" "c"
```

The new labels assignment function will allow a user to change the labels of the vector just as if it was "names":

```
x <- 1:3
labels(x) <- letters[1:3]
names(x)

## [1] "a" "b" "c"

labels(x)

## [1] "a" "b" "c"
```

Labels assignment are also available for matrixes.

2.3. labels for dendrogram objects

We can get a dendrogram's labels using the `labels` function from base R. However, in order to assing new values to it, we'll need the assignment function from **dendextend**:

```
labels(dend)  # from base R

## [1] "Ohio"       "Alaska"     "California" "New York"

set.seed(2354235)
labels(dend) <- sample(labels(dend))  # labels assingment - thanks to dendextend
labels(dend)

## [1] "New York"   "Ohio"       "Alaska"     "California"
```

2.4. labels for hclust objects

dendextend offers a `labels` method for `hclust` objects. It take speciel care to have the order of the labels be the same as is with dendrogram object, which is the order of the labels in the plotted tree. This can be turned off when using the `order` parameter:

```
# All are from dendextend
labels(hc)

## [1] "Ohio"       "Alaska"     "California" "New York"
```

```
labels(hc, order = FALSE)  # this is the order of the rows of the original data.

## [1] "Alaska"     "California" "New York"   "Ohio"

set.seed(229835)
labels(hc) <- sample(labels(hc))  # labels assingment - thanks to dendextend
labels(hc)

## [1] "California" "New York"   "Alaska"     "Ohio"
```

2.5. labels assignment and recycling

When the assigned vector has a different length, the **dendextend** assignment functions will recycle the value but also give a warning:

```
x <- 1:3
hc <- hclust(dist(US_data), "ave")
dend <- as.dendrogram(hc)
y <- matrix(1:9, 3, 3)

labels(x) <- "bob"

## Warning:  The lengths of the new labels is shorter than the length of the object
- labels are recycled.

labels(x)

## [1] "bob" "bob" "bob"

labels(hc) <- "bob"

## Warning:  The lengths of the new labels is shorter than the number of leaves
in the hclust - labels are recycled.

labels(hc)

## [1] "bob" "bob" "bob" "bob"

labels(dend) <- "bob"

## Warning:  The lengths of the new labels is shorter than the number of leaves
in the dendrogram - labels are recycled.

labels(dend)
```

```
## [1] "bob" "bob" "bob" "bob"

labels(y) <- "bob"

## Warning:  The lengths of the new labels is shorter than the length of the object's
colnames - labels are recycled.

labels(y)

## [1] "bob" "bob" "bob"
```

2.6. Tree size

Getting the size of a tree (e.g: number of leaves/terminal-nodes) is good for validation of functions, and also when we wish to initiate a variable to later fill with data from the leaves. The `labels` function for dendrogram is expensive, since it uses recurssion to get all of the tree's elements. If we are only intrested in getting the tree size, it is better to use the `nleaves` function. It has an S3 method for hclust, dendrogram and phylo (from the **ape**):

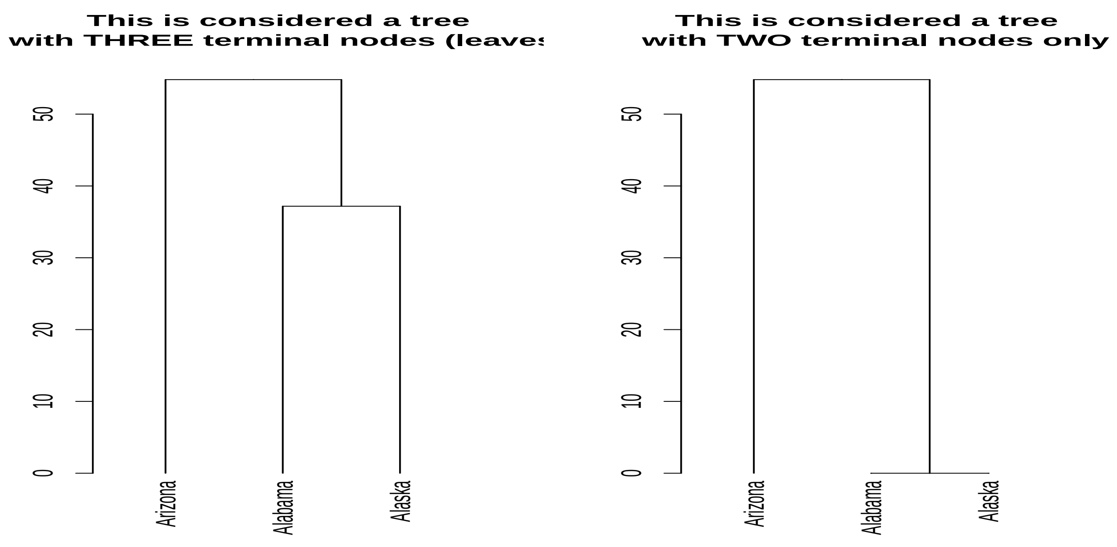
```
nleaves(hc)

## [1] 4

nleaves(dend)

## [1] 4
```

For dendrograms the speed imporvement is about 10 times using `labels`, whereas for hclust, there is not any gain made by using `nleaves`. Here is a quick benchmark:

```
library(microbenchmark)
microbenchmark(nleaves(dend), length(labels(dend)))

## Unit: microseconds
##                  expr    min     lq median     uq   max neval
##         nleaves(dend)  23.52  25.76  28.28  30.52 288.4   100
##  length(labels(dend)) 374.59 382.15 395.59 421.62 810.2   100

microbenchmark(nleaves(hc), length(labels(hc)))

## Unit: microseconds
##                expr   min    lq median    uq   max neval
##         nleaves(hc) 16.80 17.36  18.20 19.04  30.8   100
##  length(labels(hc)) 29.68 30.80  31.36 31.36 127.7   100
```

There are border-line cases where the node above some leaves is of height 0. In such a case, we would consider that node as a ”terminal node”, and in order to count the number of such terminal nodes we would use `count_terminal_nodes` function. For example:

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

par(mfrow = c(1, 2))

### Trivial case
count_terminal_nodes(dend)  # 3 terminal nodes

## [1] 3

length(labels(dend))  # 3 - the same number

## [1] 3

plot(dend, main = "This is considered a tree \n with THREE terminal nodes (leaves)")

### NON-Trivial case
str(dend)

## --[dendrogram w/ 2 branches and 3 members at h = 54.8]
##   |--leaf "Arizona"
##   `--[dendrogram w/ 2 branches and 2 members at h = 37.2]
##      |--leaf "Alabama"
##      `--leaf "Alaska"

attr(dend[[2]], "height") <- 0
count_terminal_nodes(dend)  # 2 terminal nodes, why? see this plot:

## [1] 2

# while we have 3 leaves, in practice we have only 2 terminal nodes (this
# is a feature, not a bug.)
plot(dend, main = "This is considered a tree \n with TWO terminal nodes only")
```



3. Summary

The **dendextend** package presented in this paper greatly extends the available functionality of the dendrogram objects in R.

References

R Development Core Team (2013). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Affiliation:

Tal Galili
Department of Statistics and Operations Research
Tel Aviv University, Israel
E-mail: tal.galili@math.tau.ac.il
URLs: http://www.r-statistics.com/, http://www.r-bloggers.com/

Yoav Benjamini
The Nathan and Lily Silver
Professor of Applied Statistics
Department of Statistics and Operations Research
Tel Aviv University, Israel
E-mail: ybenja@tau.ac.il
URL: http://www.tau.ac.il/~ybenja/