Package ‘networkD3’

July 11, 2015

Type Package

Title D3 JavaScript Network Graphs from R

Description Creates 'D3' 'JavaScript' network, tree, dendrogram, and Sankey graphs from 'R'.

Version 0.1.8

Date 2015-07-10

URL `http://cran.r-project.org/package=networkD3`

BugReports `https://github.com/christophergandrud/networkD3/issues`

License GPL (>= 3)

Depends R (>= 3.0.0)

Imports htmlwidgets (>= 0.3.2), plyr, rjson

Suggests htmltools (>= 0.2.6), RCurl

Enhances knitr, shiny

NeedsCompilation no

Author Christopher Gandrud [aut, cre],
J.J. Allaire [aut],
Kent Russel [aut],
B.W. Lewis [ctb],
Kevin Kuo [ctb],
Charles Sese [ctb],
Peter Ellis [ctb]

Maintainer Christopher Gandrud <`christopher.gandrud@gmail.com`>

Repository CRAN

Date/Publication 2015-07-11 11:02:21

R topics documented:

`networkD3-package` *Tools for Creating D3 Network Graphs from R*

Description

Creates D3 JavaScript network, tree, dendrogram, and Sankey graphs from R.

`as.treeNetwork` *Convert an R hclust or dendrogram object into a treeNetwork list.*

Description

`as.treeNetwork` converts an R hclust or dendrogram object into a list suitable for use by the `treeNetwork` function.

Usage

```
as.treeNetwork(d, root)
```

Arguments

`d`	An object of R class `hclust` or `dendrogram`.
`root`	An optional name for the root node. If missing, use the first argument variable name.

Details

`as.treeNetwork` coverts R objects of class `hclust` or `dendrogram` into a list suitable for use with the `treeNetwork` function.

Examples

```
# Create a hierarchical cluster object and display with treeNetwork
## dontrun
hc <- hclust(dist(USArrests), "ave")
treeNetwork(as.treeNetwork(hc))
```

`energy` *JSON data file of a projection of UK energy production and consumption in 2050.*

Description

JSON data file of a projection of UK energy production and consumption in 2050.

Format

A JSON file with two arrays `nodes` and `links`.

Source

See Mike Bostock `http://bost.ocks.org/mike/sankey/`.

`flare` *JSON data file of the Flare class hierarchy.*

Description

JSON data file of the Flare class hierarchy.

Format

A JSON file with two arrays `name` and `children`.

Source

See Mike Bostock `http://bl.ocks.org/mbostock/4063550`.

forceNetwork *Create a D3 JavaScript force directed network graph.*

Description

Create a D3 JavaScript force directed network graph.

Usage

```
forceNetwork(Links, Nodes, Source, Target, Value, NodeID, Nodesize, Group,
  height = NULL, width = NULL, colourScale = JS("d3.scale.category20()"),
  fontSize = 7, fontFamily = "serif", linkDistance = 50,
  linkWidth = JS("function(d) { return Math.sqrt(d.value); }"),
  radiusCalculation = JS(" Math.sqrt(d.nodesize)+6"), charge = -120,
  linkColour = "#666", opacity = 0.6, zoom = FALSE, legend = FALSE,
  bounded = FALSE, opacityNoHover = 0, clickAction = NULL)
```

Arguments

`Links`	a data frame object with the links between the nodes. It should include the `Source` and `Target` for each link. These should be numbered starting from 0. An optional `Value` variable can be included to specify how close the nodes are to one another.
`Nodes`	a data frame containing the node id and properties of the nodes. If no ID is specified then the nodes must be in the same order as the Source variable column in the `Links` data frame. Currently only a grouping variable is allowed.
`Source`	character string naming the network source variable in the `Links` data frame.
`Target`	character string naming the network target variable in the `Links` data frame.
`Value`	character string naming the variable in the `Links` data frame for how wide the links are.
`NodeID`	character string specifying the node IDs in the `Nodes` data frame.
`Nodesize`	character string specifying the a column in the `Nodes` data frame with some value to vary the node radius's with. See also `radiusCalculation`.
`Group`	character string specifying the group of each node in the `Nodes` data frame.
`height`	numeric height for the network graph's frame area in pixels.
`width`	numeric width for the network graph's frame area in pixels.
`colourScale`	character string specifying the categorical colour scale for the nodes. See `https://github.com/mbostock/d3/wiki/Ordinal-Scales`.
`fontSize`	numeric font size in pixels for the node text labels.
`fontFamily`	font family for the node text labels.
`linkDistance`	numeric or character string. Either numberic fixed distance between the links in pixels (actually arbitrary relative to the diagram's size). Or a JavaScript function, possibly to weight by `Value`. For example: `linkDistance = JS("function(d){return d.value * 10`

`linkWidth`	numeric or character string. Can be a numeric fixed width in pixels (arbitrary relative to the diagram's size). Or a JavaScript function, possibly to weight by `Value`. The default is `linkWidth = JS("function(d) { return Math.sqrt(d.value); }")`.
`radiusCalculation`	character string. A javascript mathematical expression, to weight the radius by `Nodesize`. The default value is `radiusCalculation = JS("Math.sqrt(d.nodesize)+6")`.
`charge`	numeric value indicating either the strength of the node repulsion (negative value) or attraction (positive value).
`linkColour`	character string specifying the colour you want the link lines to be. Multiple formats supported (e.g. hexadecimal).
`opacity`	numeric value of the proportion opaque you would like the graph elements to be.
`zoom`	logical value to enable (`TRUE`) or disable (`FALSE`) zooming.
`legend`	logical value to enable node colour legends.
`bounded`	logical value to enable (`TRUE`) or disable (`FALSE`) the bounding box limiting the graph's extent. See `http://bl.ocks.org/mbostock/1129492`.
`opacityNoHover`	numeric value of the opacity proportion for node labels text when the mouse is not hovering over them.
`clickAction`	character string with a JavaScript expression to evaluate when a node is clicked.

Source

D3.js was created by Michael Bostock. See `http://d3js.org/` and, more specifically for force directed networks `https://github.com/mbostock/d3/wiki/Force-Layout`.

See Also

`JS`.

Examples

```
#### Tabular data example.
# Load data
data(MisLinks)
data(MisNodes)
# Create graph
forceNetwork(Links = MisLinks, Nodes = MisNodes, Source = "source",
             Target = "target", Value = "value", NodeID = "name",
             Group = "group", opacity = 0.4, zoom = TRUE)

# Create graph with legend and varying node radius
forceNetwork(Links = MisLinks, Nodes = MisNodes, Source = "source",
             Target = "target", Value = "value", NodeID = "name",
             Nodesize = "size",
             radiusCalculation = "Math.sqrt(d.nodesize)+6",
             Group = "group", opacity = 0.4, legend = TRUE)

## Not run:
```

```
#### JSON Data Example
# Load data JSON formated data into two R data frames
library(RCurl)
# Create URL. paste0 used purely to keep within line width.
URL <- paste0("https://raw.githubusercontent.com/christophergandrud/",
              "networkD3/master/JSONdata/miserables.json")
MisJson <- getURL(URL)

MisLinks <- JSONtoDF(jsonStr = MisJson, array = "links")
MisNodes <- JSONtoDF(jsonStr = MisJson, array = "nodes")

# Create graph
forceNetwork(Links = MisLinks, Nodes = MisNodes, Source = "source",
             Target = "target", Value = "value", NodeID = "name",
             Group = "group", opacity = 0.4)

# Create graph with zooming
forceNetwork(Links = MisLinks, Nodes = MisNodes, Source = "source",
             Target = "target", Value = "value", NodeID = "name",
             Group = "group", opacity = 0.4, zoom = TRUE)


# Create a bounded graph
forceNetwork(Links = MisLinks, Nodes = MisNodes, Source = "source",
             Target = "target", Value = "value", NodeID = "name",
             Group = "group", opacity = 0.4, bounded = TRUE)

# Create graph with node text faintly visible when no hovering
forceNetwork(Links = MisLinks, Nodes = MisNodes, Source = "source",
             Target = "target", Value = "value", NodeID = "name",
             Group = "group", opacity = 0.4, bounded = TRUE,
             opacityNoHover = TRUE)


# Create graph with alert pop-up when a node is clicked.  You're
# unlikely to want to do exactly this, but you might use
# Shiny.onInputChange() to allocate d.XXX to an element of input
# for use in a Shiny app.
MyClickScript <- 'alert("You clicked " + d.name + " which is in row " +
       (d.index + 1) +  " of your original R data frame");'
forceNetwork(Links = MisLinks, Nodes = MisNodes, Source = "source",
             Target = "target", Value = "value", NodeID = "name",
             Group = "group", opacity = 1, zoom = F, bounded = T,
             clickAction = MyClickScript)

## End(Not run)
```

`forceNetworkOutput` *Shiny bindings for networkD3 widgets*

Description

Output and render functions for using networkD3 widgets within Shiny applications and interactive Rmd documents.

Usage

```
forceNetworkOutput(outputId, width = "100%", height = "500px")

renderForceNetwork(expr, env = parent.frame(), quoted = FALSE)

sankeyNetworkOutput(outputId, width = "100%", height = "500px")

renderSankeyNetwork(expr, env = parent.frame(), quoted = FALSE)

simpleNetworkOutput(outputId, width = "100%", height = "500px")

renderSimpleNetwork(expr, env = parent.frame(), quoted = FALSE)

treeNetworkOutput(outputId, width = "100%", height = "800px")

renderTreeNetwork(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

`outputId`	output variable to read from
`width,height`	Must be a valid CSS unit (like `"100%"`, `"400px"`, `"auto"`) or a number, which will be coerced to a string and have `"px"` appended.
`expr`	An expression that generates a networkD3 graph
`env`	The environment in which to evaluate `expr`.
`quoted`	Is `expr` a quoted expression (with `quote()`)? This is useful if you want to save an expression in a variable.

`JS` *Create character strings that will be evaluated as JavaScript*

Description

Create character strings that will be evaluated as JavaScript

Usage

```
JS(...)
```

Arguments

`...`	character string to evaluate

Source

A direct import of `JS` from Ramnath Vaidyanathan, Yihui Xie, JJ Allaire, Joe Cheng and Kenton Russell (2015). htmlwidgets: HTML Widgets for R. R package version 0.4.

`JSONtoDF` *Read a link-node structured JSON file into R as two data frames.*

Description

`JSONtoDF` reads a JSON data file into R and converts part of it to a data frame.

Usage

```
JSONtoDF(jsonStr = NULL, file = NULL, array)
```

Arguments

`jsonStr`	a JSON object to convert. Note if `jsonStr` is specified, then `file` must be `NULL`.
`file`	character string of the JSON file name. Note if `file` is specified, then `jsonStr` must be `NULL`.
`array`	character string specifying the name of the JSON array to extract. (JSON arrays are delimited by square brackets).

Details

`JSONtoDF` is intended to load JSON files into R and convert them to data frames that can be used to create network graphs. The command converts the files into R lists and then extracts the JSON array the user would like to make into a data frame.

Source

Part of the idea for the command comes from mropa's comment on StackExchange: `http://stackoverflow.com/questions/4227223/r-list-to-data-frame`.

`MisLinks` *A data file of links from Knuth's Les Miserables characters data base.*

Description

A data file of links from Knuth's Les Miserables characters data base.

Format

A data set with 254 observations of 3 variables.

Source

See Mike Bostock `http://bl.ocks.org/mbostock/4062045`.

`MisNodes` *A data file of nodes from Knuth's Les Miserables characters data base.*

Description

A data file of nodes from Knuth's Les Miserables characters data base.

Format

A data set with 77 observations of 2 variables, plus made up node size variable.

Source

See Mike Bostock `http://bl.ocks.org/mbostock/4062045`.

`sankeyNetwork` *Create a D3 JavaScript Sankey diagram*

Description

Create a D3 JavaScript Sankey diagram

Usage

```
sankeyNetwork(Links, Nodes, Source, Target, Value, NodeID, height = NULL,
  width = NULL, colourScale = JS("d3.scale.category20()"), fontSize = 7,
  fontFamily = "serif", nodeWidth = 15, nodePadding = 10)
```

Arguments

`Links`	a data frame object with the links between the nodes. It should have include the `Source` and `Target` for each link. An optional `Value` variable can be included to specify how close the nodes are to one another.
`Nodes`	a data frame containing the node id and properties of the nodes. If no ID is specified then the nodes must be in the same order as the `Source` variable column in the `Links` data frame. Currently only grouping variable is allowed.
`Source`	character string naming the network source variable in the `Links` data frame.
`Target`	character string naming the network target variable in the `Links` data frame.
`Value`	character string naming the variable in the `Links` data frame for how far away the nodes are from one another.
`NodeID`	character string specifying the node IDs in the `Nodes` data frame.
`height`	numeric height for the network graph's frame area in pixels.
`width`	numeric width for the network graph's frame area in pixels.

`colourScale`	character string specifying the categorical colour scale for the nodes. See `https://github.com/mbostock/d3/wiki/Ordinal-Scales`.
`fontSize`	numeric font size in pixels for the node text labels.
`fontFamily`	font family for the node text labels.
`nodeWidth`	numeric width of each node.
`nodePadding`	numeric essentially influences the width height.

Source

D3.js was created by Michael Bostock. See `http://d3js.org/` and, more specifically for Sankey diagrams `http://bost.ocks.org/mike/sankey/`.

See Also

`JS`

Examples

```
## Not run:
# Recreate Bostock Sankey diagram: http://bost.ocks.org/mike/sankey/
# Load energy projection data
library(RCurl)
# Create URL. paste0 used purely to keep within line width.
URL <- paste0("https://raw.githubusercontent.com/christophergandrud/",
              "networkD3/master/JSONdata/energy.json")
Energy <- getURL(URL, ssl.verifypeer = FALSE)

# Convert to data frame
EngLinks <- JSONtoDF(jsonStr = Energy, array = "links")
EngNodes <- JSONtoDF(jsonStr = Energy, array = "nodes")

# Plot
sankeyNetwork(Links = EngLinks, Nodes = EngNodes, Source = "source",
             Target = "target", Value = "value", NodeID = "name",
              fontSize = 12, nodeWidth = 30)

## End(Not run)
```

`saveNetwork` *Save a network graph to an HTML file*

Description

Save a networkD3 graph to an HTML file for sharing with others. The HTML can include it's dependencies in an adjacent directory or can bundle all dependencies into the HTML file (via base64 encoding).

Usage

```
saveNetwork(network, file, selfcontained = TRUE)
```

Arguments

`network`	Network to save (e.g. result of calling the function `simpleNetwork`).
`file`	File to save HTML into
`selfcontained`	Whether to save the HTML as a single self-contained file (with external resources base64 encoded) or a file with external resources placed in an adjacent directory.

`simpleNetwork` *Function for creating simple D3 JavaScript force directed network graphs.*

Description

`simpleNetwork` creates simple D3 JavaScript force directed network graphs.

Usage

```
simpleNetwork(Data, Source = NULL, Target = NULL, height = NULL,
  width = NULL, linkDistance = 50, charge = -200, fontSize = 7,
  fontFamily = "serif", linkColour = "#666", nodeColour = "#3182bd",
  nodeClickColour = "#E34A33", textColour = "#3182bd", opacity = 0.6,
  zoom = F)
```

Arguments

`Data`	a data frame object with three columns. The first two are the names of the linked units. The third records an edge value. (Currently the third column doesn't affect the graph.)
`Source`	character string naming the network source variable in the data frame. If `Source = NULL` then the first column of the data frame is treated as the source.
`Target`	character string naming the network target variable in the data frame. If `Target = NULL` then the second column of the data frame is treated as the target.
`height`	height for the network graph's frame area in pixels (if `NULL` then height is automatically determined based on context)
`width`	numeric width for the network graph's frame area in pixels (if `NULL` then width is automatically determined based on context)
`linkDistance`	numeric distance between the links in pixels (actually arbitrary relative to the diagram's size).
`charge`	numeric value indicating either the strength of the node repulsion (negative value) or attraction (positive value).

`fontSize`	numeric font size in pixels for the node text labels.
`fontFamily`	font family for the node text labels.
`linkColour`	character string specifying the colour you want the link lines to be. Multiple formats supported (e.g. hexadecimal).
`nodeColour`	character string specifying the colour you want the node circles to be. Multiple formats supported (e.g. hexadecimal).
`nodeClickColour`	character string specifying the colour you want the node circles to be when they are clicked. Also changes the colour of the text. Multiple formats supported (e.g. hexadecimal).
`textColour`	character string specifying the colour you want the text to be before they are clicked. Multiple formats supported (e.g. hexadecimal).
`opacity`	numeric value of the proportion opaque you would like the graph elements to be.
`zoom`	logical value to enable (`TRUE`) or disable (`FALSE`) zooming

Source

D3.js was created by Michael Bostock. See `http://d3js.org/` and, more specifically for directed networks `https://github.com/mbostock/d3/wiki/Force-Layout`

Examples

```
# Fake data
Source <- c("A", "A", "A", "A", "B", "B", "C", "C", "D")
Target <- c("B", "C", "D", "J", "E", "F", "G", "H", "I")
NetworkData <- data.frame(Source, Target)

# Create graph
simpleNetwork(NetworkData)
simpleNetwork(NetworkData, fontFamily = "sans-serif")
```

`treeNetwork` *Create Reingold-Tilford Tree network diagrams.*

Description

Create Reingold-Tilford Tree network diagrams.

Usage

```
treeNetwork(List, height = NULL, width = NULL, fontSize = 10,
  fontFamily = "serif", linkColour = "#ccc", nodeColour = "#fff",
  nodeStroke = "steelblue", textColour = "#111", opacity = 0.9,
  margin = 0)
```

Arguments

`List`	a hierarchical list object with a root node and children.
`height`	height for the network graph's frame area in pixels (if `NULL` then height is automatically determined based on context)
`width`	numeric width for the network graph's frame area in pixels (if `NULL` then width is automatically determined based on context)
`fontSize`	numeric font size in pixels for the node text labels.
`fontFamily`	font family for the node text labels.
`linkColour`	character string specifying the colour you want the link lines to be. Multiple formats supported (e.g. hexadecimal).
`nodeColour`	character string specifying the colour you want the node circles to be. Multiple formats supported (e.g. hexadecimal).
`nodeStroke`	character string specifying the colour you want the node perimeter to be. Multiple formats supported (e.g. hexadecimal).
`textColour`	character string specifying the colour you want the text to be before they are clicked. Multiple formats supported (e.g. hexadecimal).
`opacity`	numeric value of the proportion opaque you would like the graph elements to be.
`margin`	integer value of the plot margin. Set the margin appropriately to accomodate long text labels.

Source

Reingold. E. M., and Tilford, J. S. (1981). Tidier Drawings of Trees. IEEE Transactions on Software Engineering, SE-7(2), 223-228.

Mike Bostock: `http://bl.ocks.org/mbostock/4063550`.

Examples

```
## Not run:
#### Create tree from JSON formatted data
## Download JSON data
library(RCurl)
# Create URL. paste0 used purely to keep within line width.
URL <- paste0("https://raw.githubusercontent.com/christophergandrud/",
              "networkD3/master/JSONdata/flare.json")
Flare <- getURL(URL)

## Convert to list format
Flare <- rjson::fromJSON(Flare)

## Recreate Bostock example from http://bl.ocks.org/mbostock/4063550
treeNetwork(List = Flare, fontSize = 10, opacity = 0.9)

#### Create a tree dendrogram from an R hclust object
hc <- hclust(dist(USArrests), "ave")
```

```
treeNetwork(as.treeNetwork(hc))
treeNetwork(as.treeNetwork(hc), fontFamily = "cursive")

#### Create tree from a hierarchical R list
For an alternative structure see: http://stackoverflow.com/a/30747323/1705044
CanadaPC <- list(name = "Canada", children = list(list(name = "Newfoundland",
                    children = list(list(name = "St. John's"))),
               list(name = "PEI",
                    children = list(list(name = "Charlottetown"))),
               list(name = "Nova Scotia",
                    children = list(list(name = "Halifax"))),
               list(name = "New Brunswick",
                    children = list(list(name = "Fredericton"))),
               list(name = "Quebec",
                    children = list(list(name = "Montreal"),
                                    list(name = "Quebec City"))),
               list(name = "Ontario",
                    children = list(list(name = "Toronto"),
                                    list(name = "Ottawa"))),
               list(name = "Manitoba",
                    children = list(list(name = "Winnipeg"))),
               list(name = "Saskatchewan",
                    children = list(list(name = "Regina"))),
               list(name = "Nunavuet",
                    children = list(list(name = "Iqaluit"))),
               list(name = "NWT",
                    children = list(list(name = "Yellowknife"))),
               list(name = "Alberta",
                    children = list(list(name = "Edmonton"))),
               list(name = "British Columbia",
                    children = list(list(name = "Victoria"),
                                    list(name = "Vancouver"))),
               list(name = "Yukon",
                    children = list(list(name = "Whitehorse")))
))

treeNetwork(List = CanadaPC, fontSize = 10)

## End(Not run)
```

Index