Package 'networktools'

October 6, 2021

Title Tools for Identifying Important Nodes in Networks

Version 1.4.0

Date 2021-10-6

Description Includes assorted tools for network analysis. Bridge centrality; gold-
bricker; MDS, PCA, & eigenmodel network plotting.

Depends R (>= 3.0.0)

License GPL-3

Encoding UTF-8

LazyData true

Imports
qgraph,igraph,IsingFit,reshape2,nnet,ggplot2,gridExtra,stats,graphics,utils,NetworkComparisonTest,cocor,RColorBrewer,

RoxygenNote 7.1.2

Suggests dplyr, testthat

URL `https://CRAN.R-project.org/package=networktools`

BugReports `https://github.com/paytonjjones/networktools/issues`

NeedsCompilation no

Author Payton Jones [aut, cre]

Maintainer Payton Jones <`paytonjjones@gmail.com`>

Repository CRAN

Date/Publication 2021-10-06 19:00:02 UTC

R topics documented:

`assumptionCheck` *Assumption Checking Function*

Description

Checks some basic assumptions about the suitability of network analysis on your data

Usage

```
assumptionCheck(
  data,
  type = c("network", "impact"),
  percent = 20,
  split = c("median", "mean", "forceEqual", "cutEqual", "quartiles"),
  plot = FALSE,
  binary.data = FALSE,
  na.rm = TRUE
)
```

Arguments

`data`	dataframe or matrix of observational data (rows: observations, columns: nodes)
`type`	which assumptions to check? "network" tests the suitability for network analysis in general. "impact" tests the suitability for analyzing impact
`percent`	percent difference from grand mean that is acceptable when comparing variances.
`split`	if type="impact", specifies the type of split to utilize
`plot`	logical. Should histograms each variable be plotted?

`binary.data`	logical. Defaults to FALSE
`na.rm`	logical. Should missing values be removed?

Details

This function is in BETA. Please report any errors.

Network analysis rests on several assumptions. Among these: - Variance of each node is (roughly) equal - Distributions are (roughly) normal

Comparing networks in impact rests on additional assumptions including: - Overall variances are (roughly) equal in each half

This function checks these assumptions and notifies any violations. This function is not intended as a substitute for careful data visualization and independent assumption checks.

See citations in the references section for further details.

References

Terluin, B., de Boer, M. R., & de Vet, H. C. W. (2016). Differences in Connection Strength between Mental Symptoms Might Be Explained by Differences in Variance: Reanalysis of Network Data Did Not Confirm Staging. PLOS ONE, 11(11), e0155205. Retrieved from https://doi.org/10.1371/journal.pone.0155205

`bridge` *Bridge Centrality*

Description

Calculates bridge centrality metrics (bridge strength, bridge betweenness, bridge closeness, and bridge expected influence) given a network and a prespecified set of communities.

Usage

```
bridge(
  network,
  communities = NULL,
  useCommunities = "all",
  directed = NULL,
  nodes = NULL,
  normalize = FALSE
)
```

Arguments

`network`	a network of class "igraph", "qgraph", or an adjacency matrix representing a network
`communities`	an object of class "communities" (igraph) OR a character vector of community assignments for each node (e.g., c("Comm1", "Comm1", "Comm2", "Comm2)). The ordering of this vector should correspond to the vector from argument "nodes". Can also be in list format (e.g., list("Comm1"=c(1:10), "Comm2"=c(11:20)))

`useCommunities`	character vector specifying which communities should be included. Default set to "all"
`directed`	logical. Directedness is automatically detected if set to "NULL" (the default). Symmetric adjacency matrices will be undirected, asymmetric matrices will be directed
`nodes`	a vector containing the names of the nodes. If set to "NULL", this vector will be automatically detected in the order extracted
`normalize`	logical. Bridge centralities are divided by their highest possible value (assuming max edge strength=1) in order to normalize by different community sizes

Details

To plot the results, first save as an object, and then use plot() (see ?plot.bridge)

Centrality metrics (strength, betweenness, etc.) illuminate how nodes are interconnected among the entire network. However, sometimes we are interested in the connectivity *between specific communities* in a larger network. Nodes that are important in communication between communities can be conceptualized as bridge nodes.

Bridge centrality statistics aim to identify bridge nodes. Bridge centralities can be calculated across all communities, or between a specific subset of communities (as identified by the `useCommunities` argument)

The bridge() function currently returns 5 centrality metrics: 1) bridge strength, 2) bridge betweenness, 3) bridge closeness, 4) bridge expected influence (1-step), and 5) bridge expected influence (2-step)

See ?plot.bridge for plotting details.

Bridge strength is defined as the sum of the absolute value of all edges that exist between a node A and all nodes that are not in the same community as node A. In a directed network, bridge strength can be separated into bridge in-degree and bridge out-degree.

Bridge betweenness is defined as the number of times a node B lies on the shortest path between nodes A and C, where nodes A and C come from different communities.

Bridge closeness is defined as the inverse of the average length of the path from a node A to all nodes that are not in the same community as node A.

Bridge expected influence (1-step) is defined as the sum of the value (+ or -) of all edges that exist between a node A and all nodes that are not in the same community as node A. In a directed network, expected influence only considers edges extending from the given node (e.g., out-degree)

Bridge expected influence (2-step) is similar to 1-step, but also considers the indirect effect that a node A may have on other communities through other nodes (e.g, an indirect effect on node C as in A -> B -> C). Indirect effects are weighted by the first edge weight (e.g., A -> B), and then added to the 1-step expected influence. Indirect effects back on node A's own community (A -> B -> A) are not counted.

If negative edges exist, bridge expected influence should be used. Bridge closeness and bridge betweenness are only defined for positive edge weights, thus negative edges, if present, are deleted in the calculation of these metrics. Bridge strength uses the absolute value of edge weights.

Value

`bridge` returns a list of class "`bridge`" which contains:

$'Bridge Strength'

$'Bridge Betweenness'

$'Bridge Closeness'

$'Bridge Expected Influence (1-step)'

$'Bridge Expected Influence (2-step)'

Each of these contains a vector of named centrality values

$'communities' is also returned, which returns the communities in vector format. If communities were supplied as a list or igraph object, it is advised that one check the accuracy of this vector.

Examples

```
graph1 <- qgraph::qgraph(cor(depression))

b <- bridge(graph1, communities=c('1','1','2','2','2','2','1','2','1'))
b
```

`coerce_to_adjacency` *Coerce to adjacency matrix*

Description

Takes an object of type "qgraph", "igraph", or an adjacency matrix (or data.frame) and outputs an adjacency matrix

Usage

```
coerce_to_adjacency(input, directed = NULL)
```

Arguments

`input`	a network of class "igraph", "qgraph", or an adjacency matrix representing a network
`directed`	logical. is the network directed? If set to NULL, auto-detection is used

depression — *Simulated Depression Profiles*

Description

This simulated dataset contains severity ratings for 9 symptoms of major depressive disorder in 1000 individuals. Symptom ratings are assumed to be self-reported on a 100 point sliding scale.

Usage

```
depression
```

Format

a dataframe. Columns represent symptoms and rows represent individuals

Examples

```
head(depression)
```

EIGENnet — *EIGENnet*

Description

Convenience function for converting a qgraph object to an eigenmodel layout

Usage

```
EIGENnet(
  qgraph_net,
  EIGENadj = NULL,
  S = 1000,
  burn = 200,
  seed = 1,
  repulse = F,
  repulsion = 1,
  eigenmodelArgs = list(),
  ...
)
```

Arguments

`qgraph_net`	an object of type `qgraph`
`EIGENadj`	to use a base matrix for the eigenmodel other than the adjacency matrix stored in `qgraph_net`, provide it in this argument
`S`	number of samples from the Markov chain
`burn`	number of initial scans of the Markov chain to be dropped
`seed`	a random seed
`repulse`	logical. Add a small repulsion force with wordcloud package to avoid node overlap?
`repulsion`	scalar for the repulsion force (if repulse=T). Larger values add more repulsion
`eigenmodelArgs`	additional arguments in list format passed to `eigenmodel::eigenmodel_mcmc`
`...`	additional arguments passed to `qgraph`

Details

An eigenmodel can be interpreted based on coordinate placement of each node. A node in the top right corner scored high on both the first and second latent components

References

Jones, P. J., Mair, P., & McNally, R. J. (2018). Visualizing psychological networks: A tutorial in R. Frontiers in Psychology, 9, 1742. https://doi.org/10.3389/fpsyg.2018.01742

`expectedInf` *Expected Influence*

Description

Calculates the one-step and two-step expected influence of each node.

Usage

```
expectedInf(network, step = c("both", 1, 2), directed = FALSE)
```

Arguments

`network`	an object of type `qgraph`, `igraph`, or an adjacency matrix representing a network. Adjacency matrices should be complete (e.g., not only upper or lower half)
`step`	compute 1-step expected influence, 2-step expected influence, or both
`directed`	logical. Specifies if edges are directed, defaults to FALSE

Details

When a network contains both positive and negative edges, traditional centrality measures such as strength centrality may not accurately predict node influence on the network. Robinaugh, Millner, & McNally (2016) showed that in these cases, expected influence is a more appropriate measure.

One-step expected influence is defined as the sum of all edges extending from a given node (where the sign of each edge is maintained).

Two-step expected influence, as the name implies, measures connectivity up to two edges away from the node. It is defined as the sum of the (weighted) expected influences of each node connected to the initial node plus the one-step expected influence of the initial node. Weights are determined by the edge strength between the initial node and each "second step" node.

See citations in the references section for further details.

References

Robinaugh, D. J., Millner, A. J., & McNally, R. J. (2016). Identifying highly influential nodes in the complicated grief network. *Journal of abnormal psychology*, 125, 747.

Examples

```
out1 <- expectedInf(cor(depression[,1:5]))

out1$step1
out1$step2
plot(out1)
plot(out1, order="value", zscore=TRUE)

igraph_obj <- igraph::graph_from_adjacency_matrix(cor(depression))
out_igraph <- expectedInf(igraph_obj)

qgraph_obj <- qgraph::qgraph(cor(depression), DoNotPlot=TRUE)
out_qgraph <- expectedInf(qgraph_obj)
```

---

goldbricker *Goldbricker - Identifying redundant nodes in networks using compared correlations*

Description

This function compares correlations in a psychometric network in order to identify nodes which most likely measure the same underlying construct (i.e., are colinear)

Usage

```
goldbricker(
  data,
  p = 0.05,
  method = "hittner2003",
  threshold = 0.25,
  corMin = 0.5,
  progressbar = TRUE
)
```

Arguments

`data`	a data frame consisting of n rows (participants) and j columns (variables)
`p`	a p-value threshold for determining if correlation pairs are "significantly different"
`method`	method for comparing correlations. See ?cocor.dep.groups.overlap for a full list
`threshold`	variable pairs which have less than the threshold proportion of significantly different correlations will be considered "bad pairs"
`corMin`	the minimum zero-order correlation between two items to be considered "bad pairs". Items that are uncorrelated are unlikely to represent the same underlying construct
`progressbar`	logical. prints a progress bar in the console

Details

In a given psychometric network, two nodes may be redundantly measuring the same underlying construct. If this is the case, the correlations between those two variables and all other variables should be highly similar. That is, they should correlate to the same degree with other variables.

The cocor package uses a p-value threshold to determine whether a pair of correlations to a third variable are significantly different from each other. Goldbricker wraps the cocor package to compare every possible combination of correlations in a psychometric network. It calculates the proportion of correlations which are significantly different for each different pair of nodes.

Using the threshold argument, one can set the proportion of correlations which is deemed "too low". All pairs of nodes which fall below this threshold are returned as defined "bad pairs".

Pairs can then be combined using the reduce_net function

Note: to quickly change the threshold, one may simply enter an object of class "goldbricker" in the data argument, and change the threshold. The p-value cannot be modified in the same fashion, as re-computation is necessary.

Value

`goldbricker` returns a list of class "`goldbricker`" which contains:

$proportion_matrix - a j x j matrix of proportions. Each proportion signifies the amount of significantly different correlations between the given node pair (j x j) $suggested_reductions - a vector of "bad pairs" (names) and their proportions (values) $p - p value from input $threshold - threshold from input

Examples

```
gb_depression <- goldbricker(depression, threshold=0.5)

reduced_depression <- net_reduce(data=depression, badpairs=gb_depression)

## Set a new threshold quickly
gb_depression_60 <- goldbricker(data=gb_depression, threshold=0.6)
```

`impact` *Network Impact (combined function)*

Description

Generates the global strength impact, network structure impact, and edge impact simultaneously for a given set of nodes.

Usage

```
impact(
  input,
  gamma,
  nodes = c("all"),
  binary.data = FALSE,
  weighted = TRUE,
  split = c("median", "mean", "forceEqual", "cutEqual", "quartiles")
)
```

Arguments

`input`	a matrix or data frame of observations (not a network/edgelist). See included example datasets `depression` and `social`.
`gamma`	the sparsity parameter used in generating networks. Defaults to 0.5 for interval data and 0.25 for binary data
`nodes`	indicates which nodes should be tested. Can be given as a character string of desired nodes (e.g., c("node1","node2")) or as a numeric vector of column numbers (e.g., c(1,2)).
`binary.data`	logical. Indicates whether the input data is binary
`weighted`	logical. Indicates whether resultant networks preserve edge weights or binarize edges.
`split`	method by which to split network given non-binary data. "median": median split (excluding the median), "mean": mean split, "forceEqual": creates equally sized groups by partitioning random median observations to the smaller group, "cutEqual": creates equally sized groups by deleting random values from the bigger group,"quartile": uses the top and bottom quartile as groups

Details

The structures of networks sometimes vary as a function of certain external variables. For instance, Pe et al. (2015) found that the structure of mood networks varied as a function of whether or not individuals had been diagnosed with major depression.

The structures of networks may also vary as a function of *internal* variables; that is to say, as a function of each node. ***Impact statistics measure the degree to which node levels impact network structure.*** Impact statistics are similar to centrality statistics in the sense that they are a property of each node in a network.

Three relevant impact statistics are included in the `networktools` package: global strength impact, network structure impact, and edge impact. To ease computational burden, all three statistics are calculated simultaneously in the `impact` function.

Impact statistics are calculated by temporarily regarding a node as an *external* variable to the network. The remaining data are then divided into two networks according to a median split (default) on the external node. Network invariance measures are then computed on the two networks. While median splits are not advisable when continuous analyses are possible, it is not possible to compute networks in a continuous fashion. The median split excludes observations that fall exactly on the median. In the case of binary data, data are split by level rather than by median.

Value

`impact` returns a list of class "`all.impact`" which contains:

1. A list of class "global.impact"
2. A list of class "structure.impact"
3. A list of class "edge.impact"

`impact.boot` *Bootstrapping convenience function for impact statistics*

Description

impact.boot is DEPRECATED. The function will be removed in the next update. Use impact.NCT instead.

Usage

```
impact.boot(
  input,
  boots,
  gamma,
  nodes = c("all"),
  binary.data = FALSE,
  weighted = TRUE,
  split = c("median", "mean", "forceEqual", "cutEqual", "quartiles"),
  progressbar = TRUE
)
```

Arguments

`input`	a matrix or data frame of observations (not a network/edgelist). See included example datasets `depression` and `social`.
`boots`	the number of times to bootstrap the impact function
`gamma`	the sparsity parameter used in generating networks. Defaults to 0.5 for interval data and 0.25 for binary data
`nodes`	indicates which nodes should be tested. Can be given as a character string of desired nodes (e.g., c("node1","node2")) or as a numeric vector of column numbers (e.g., c(1,2)).
`binary.data`	logical. Indicates whether the input data is binary
`weighted`	logical. Indicates whether resultant networks preserve edge weights or binarize edges.
`split`	method by which to split network given non-binary data. "median": median split (excluding the median), "mean": mean split, "forceEqual": creates equally sized groups by partitioning random median observations to the smaller group, "cutEqual": creates equally sized groups by deleting random values from the bigger group,"quartile": uses the top and bottom quartile as groups
`progressbar`	Logical. Should the pbar be plotted in order to see the progress of the estimation procedure? Defaults to TRUE.

Details

This function wraps the function `impact` and bootstraps to provide confidence intervals of node impacts.

This method is computationally intensive. It is recommended that users test a subset of nodes at a time using the `nodes` argument, rather than testing all nodes simultaneously.

`impact.boot` returns an object of class `impact.boot`, which includes confidence intervals.

Value

`impact.boot` returns a list of class "impact.boot"

`impact.NCT` *Network Comparison Test for Impact Statistics*

Description

This function wraps the function `NCT` from the NetworkComparisonTest package to provide an explicit test for the significance of node impacts.

Usage

```
impact.NCT(
  input,
  it,
  gamma,
  nodes = c("all"),
  binary.data = FALSE,
  weighted = TRUE,
  split = c("median", "mean", "forceEqual", "cutEqual", "quartiles"),
  paired = FALSE,
  AND = TRUE,
  test.edges = FALSE,
  edges,
  progressbar = TRUE
)
```

Arguments

`input`	a matrix or data frame of observations (not a network/edgelist). See included example datasets `depression` and `social`.
`it`	the number of iterations (permutations) in each network comparison test
`gamma`	the sparsity parameter used in generating networks. Defaults to 0.5 for interval data and 0.25 for binary data
`nodes`	indicates which nodes should be tested. Can be given as a character string of desired nodes (e.g., c("node1","node2")) or as a numeric vector of column numbers (e.g., c(1,2)).
`binary.data`	logical. Indicates whether the input data is binary
`weighted`	logical. Indicates whether resultant networks preserve edge weights or binarize edges.
`split`	method by which to split network given non-binary data. "median": median split (excluding the median), "mean": mean split, "forceEqual": creates equally sized groups by partitioning random median observations to the smaller group, "cutEqual": creates equally sized groups by deleting random values from the bigger group,"quartile": uses the top and bottom quartile as groups (paired, AND, test.edges, edges, progressbar)
`paired`	Logical. Can be TRUE of FALSE to indicate whether the samples are dependent or not. If paired is TRUE, relabeling is performed within each pair of observations. If paired is FALSE, relabeling is not restricted to pairs of observations. Note that, currently, dependent data is assumed to entail one group measured twice.
`AND`	Logical. Can be TRUE of FALSE to indicate whether the AND-rule or the OR-rule should be used to define the edges in the network. Defaults to TRUE. Only necessary for binary data.
`test.edges`	Logical. Can be TRUE of FALSE to indicate whether or not differences in individual edges should be tested.

`edges`	Character or list. When 'all', differences between all individual edges are tested. When provided a list with one or more pairs of indices referring to variables, the provided edges are tested. A Holm-Bonferroni correction is applied to control for multiple testing.
`progressbar`	Logical. Should the pbar be plotted in order to see the progress of the estimation procedure? Defaults to TRUE.

Details

The NCT method is computationally intensive. It is recommended that users test a subset of nodes at a time using the `nodes` argument, rather than testing all nodes simultaneously.

In order to be interpreted in a meaningful way, the significance of impact statistics should be explicitly tested.

The NCT function from the NetworkComparisonTest uses a permutation test to determine the significance of structure invariances between two networks. Because impact statistics are mathematically defined as structural invariance between two networks, `NCT` is an appropriate method to test the significance of impact statistics.

`impact.NCT` returns an object of class `NCT`, which includes p-values for invariances.

Value

`impact` returns a list where each element is an object of class `NCT`

`MDSnet` *MDSnet*

Description

Convenience function for converting a qgraph object to a layout determined by multidimensional scaling

Usage

```
MDSnet(
  qgraph_net,
  type = c("ordinal", "interval", "ratio", "mspline"),
  MDSadj = NULL,
  stressTxt = F,
  repulse = F,
  repulsion = 1,
  mdsArgs = list(),
  ...
)
```

Arguments

`qgraph_net`	an object of type `qgraph`
`type`	transformation function for MDS, defaults to "ordinal"
`MDSadj`	to use a proximities matrix other than the adjacency matrix stored in `qgraph_net`, provide it in this argument
`stressTxt`	logical. Print the stress value in the lower left corner of the plot?
`repulse`	logical. Add a small repulsion force with wordcloud package to avoid node overlap?
`repulsion`	scalar for the repulsion force. Larger values add more repulsion
`mdsArgs`	additional arguments in list format passed to `smacof::mds`
`...`	additional arguments passed to `qgraph`

Details

A network plotted with multidimensional scaling can be interpreted based on the distances between nodes. Nodes close together represent closely associated nodes, whereas nodes that are far apart represent unassociated or negatively associated nodes.

References

Jones, P. J., Mair, P., & McNally, R. J. (2018). Visualizing psychological networks: A tutorial in R. Frontiers in Psychology, 9, 1742. https://doi.org/10.3389/fpsyg.2018.01742

`networktools` *networktools.*

Description

Tools for Identifying Important Nodes in Networks

Details

Includes assorted tools for network analysis. Specifically, includes functions for calculating impact statistics, which aim to identify how each node impacts the overall network structure (global strength impact, network structure impact, edge impact), and for calculating and visualizing expected influence.

For a complete list of functions, use library(help = "networktools")

For a complete list of vignettes, use browseVignettes("networktools")

Author(s)

Payton J. Jones

net_reduce *net_reduce*

Description

This function takes predefined pairs of colinear variables in a dataset and a) combines them via PCA or b) picks the "better" variable and eliminates the other variable

Usage

```
net_reduce(data, badpairs, method = c("PCA", "best_goldbricker"))
```

Arguments

`data`	a data frame consisting of n rows (participants) and j columns (variables)
`badpairs`	pairs of variables to be combined. Input may consist of: -an object of class "goldbricker" (all bad pairs are combined) -a vector of item names, each consecutive pair will be considered a bad pair -a matrix with 2 columns where each bad pair takes up 1 row
`method`	method for combining variables. PCA takes the first principal component of the two variables and defines it as a new variable. best_goldbricker requires that the input of "badpairs" be an object of class "goldbricker" it selects the more unique variable, and eliminates the other variable in the pair.

Details

In a given psychometric network, two nodes may be redundantly measuring the same underlying construct. If this is the case, both variables should not appear in the same network, or network properties will be inaccurate. These variable pairs can be reduced by combining them, or by eliminating one of them. net_reduce automates this process when given a list of "bad pairs"

If the same variable appears in multiple "bad pairs" (e.g., "x" and "y" is a bad pair, and so is "x" and "z"), only the first of these pairs which appears in the badpairs argument will be reduced by the function.

Value

`goldbricker` returns a dataframe of n rows (participants) and j - x columns, where j is the number of variables in the original dataframe, and x is the number of bad pairs to reduce.

Examples

```
gb_depression <- goldbricker(depression, threshold=0.5)

reduced_depression_PCA <- net_reduce(data=depression, badpairs=gb_depression)
reduced_depression_best <- net_reduce(data=depression,
                           badpairs=gb_depression, method="best_goldbricker")
```

PCAnet *PCAnet*

Description

Convenience function for converting a qgraph object to a layout determined by principal components analysis

Usage

```
PCAnet(
  qgraph_net,
  cormat,
  varTxt = F,
  repulse = F,
  repulsion = 1,
  principalArgs = list(),
  ...
)
```

Arguments

`qgraph_net`	an object of type `qgraph`
`cormat`	the correlation matrix of the relevant data. If this argument is missing, the function will assume that the adjacency matrix from `qgraph_net` is a correlation matrix
`varTxt`	logical. Print the variance accounted for by the PCA in the lower left corner of the plot
`repulse`	logical. Add a small repulsion force with wordcloud package to avoid node overlap?
`repulsion`	scalar for the repulsion force (if repulse=T). Larger values add more repulsion
`principalArgs`	additional arguments in list format passed to `psych::principal`
`...`	additional arguments passed to `qgraph`

Details

A network plotted with PCA can be interpreted based on coordinate placement of each node. A node in the top right corner scored high on both the first and second principal components

References

Jones, P. J., Mair, P., & McNally, R. J. (2018). Visualizing psychological networks: A tutorial in R. Frontiers in Psychology, 9, 1742. https://doi.org/10.3389/fpsyg.2018.01742

plot.all.impact *Plot "all.impact" objects*

Description

Convenience function for generating impact plots

Usage

```
## S3 method for class 'all.impact'
plot(
  x,
  order = c("given", "value", "alphabetical"),
  zscore = FALSE,
  abs_val = FALSE,
  ...
)
```

Arguments

`x`	an output object from an impact function (class `all.impact`)
`order`	"alphabetical" orders nodes alphabetically, "value" orders nodes from highest to lowest impact value
`zscore`	logical. Converts raw impact statistics to z-scores for plotting
`abs_val`	logical. Plot absolute values of global strength impacts. If both abs_val=TRUE and zscore=TRUE, plots the absolute value of the z-scores.
`...`	other plotting specifications (ggplot2)

Details

Inputting an object of class `global.impact` or `structure.impact` will return a line plot that shows the relative impacts of each node. Inputting a `all.impact` object will return both of these plots simultaneously

plot.bridge *Plot "bridge" objects*

Description

Convenience function for plotting bridge centrality

Usage

```
## S3 method for class 'bridge'
plot(
  x,
  order = c("given", "alphabetical", "value"),
  zscore = FALSE,
  include,
  color = FALSE,
  colpalette = "Dark2",
  plotNA = FALSE,
  ...
)
```

Arguments

`x`	an output object from `bridge` (class `bridge`)
`order`	"alphabetical" orders nodes alphabetically, "value" orders nodes from highest to lowest centrality values
`zscore`	logical. Converts raw impact statistics to z-scores for plotting
`include`	a vector of centrality measures to include ("Bridge Strength", "Bridge Betweenness", "Bridge Closeness", "Bridge Expected Influence (1-step)", "Bridge Expected Influence (2-step)"), if missing all available measures will be plotted
`color`	logical. Color each community separately in the plot?
`colpalette`	A palette name from RColorBrewer, for coloring of axis labels
`plotNA`	should nodes with NA values be included on the y axis?
`...`	other plotting specifications in ggplot2 (aes)

Details

Inputting an object of class `bridge` will return a line plot that shows the bridge centrality values of each node

Examples

```
b <- bridge(cor(depression))
plot(b)
plot(b, order="value", zscore=TRUE,include=c("Bridge Strength", "Bridge Betweenness"))
```

plot.edge.impact *Plot "edge.impact" objects*

Description

Convenience function for generating edge impact plots

Usage

```
## S3 method for class 'edge.impact'
plot(
  x,
  nodes = c("first", "all"),
  type.edgeplot = c("contrast", "single"),
  title = NULL,
  maximum = "auto",
  ...
)
```

Arguments

`x`	an output object from an impact function (`edge.impact`)
`nodes`	specifies which impact graph(s) to be plotted. Can be given as a character string of desired node(s) (e.g., c("node1","node2")) or as a numeric vector of column numbers (e.g., c(1,2)).
`type.edgeplot`	"contrast" returns two separate networks: one for low values of the given node, and one for high values. "single" returns a network where edges represent the edge impact of the given node.
`title`	if not otherwise specified, title is automatically generated
`maximum`	sets a maximum for edge thickness (see maximum argument in ?qgraph). "auto" uses the maximum edge from the collective two networks.
`...`	other plotting specifications (qgraph)

Details

Inputting a `edge.impact` object will return network plots. Depending on the `type.edgeplot` argument, two types of networks are possible. Using "contrast" will return "true" estimated networks from the data, separated by a median split on the selected node. Using "single" will return a network where the edges represent the edge impacts for the selected node (e.g., thick positive edges represent a strong positive edge impact)

`plot.expectedInf` *Plot "expectedInf" objects*

Description

Convenience function for plotting expected influence

Usage

```
## S3 method for class 'expectedInf'
plot(x, order = c("given", "alphabetical", "value"), zscore = TRUE, ...)
```

Arguments

`x`	an output object from an `expectedInf` (class `expectedInf`)
`order`	"alphabetical" orders nodes alphabetically, "value" orders nodes from highest to lowest impact value
`zscore`	logical. Converts raw impact statistics to z-scores for plotting
`...`	other plotting specifications (ggplot2)

Details

Inputting an object of class `expectedInf` will return a line plot that shows the relative one-step and/or two-step expected influence of each node.

Examples

```
myNetwork <- cor(depression[,1:5])
out1 <- expectedInf(myNetwork)
plot(out1$step1)

plot(out1, order="value", zscore=TRUE)
```

`plot.global.impact` *Plot "global.impact" objects*

Description

Convenience function for generating global strength impact plots

Usage

```
## S3 method for class 'global.impact'
plot(
  x,
  order = c("given", "value", "alphabetical"),
  zscore = FALSE,
  abs_val = FALSE,
  ...
)
```

Arguments

`x`	an output object from an impact function (class `global.impact`)
`order`	"alphabetical" orders nodes alphabetically, "value" orders nodes from highest to lowest impact value
`zscore`	logical. Converts raw impact statistics to z-scores for plotting
`abs_val`	logical. Plot absolute values of global strength impacts. If both abs_val=TRUE and zscore=TRUE, plots the absolute value of the z-scores.
`...`	other plotting specifications (ggplot2)

Details

Inputting an object of class `global.impact` will return a line plot that shows the relative global impacts of each node.

`plot.structure.impact` *Plot "structure.impact" objects*

Description

Convenience function for generating network structure impact plots

Usage

```
## S3 method for class 'structure.impact'
plot(
  x,
  order = c("given", "alphabetical", "value"),
  zscore = FALSE,
  abs_val = FALSE,
  ...
)
```

Arguments

`x`	an output object from an impact function (class `structure.impact`)
`order`	"alphabetical" orders nodes alphabetically, "value" orders nodes from highest to lowest impact value
`zscore`	logical. Converts raw impact statistics to z-scores for plotting
`abs_val`	logical. Plot absolute values of network structure impacts. If both abs_val=TRUE and zscore=TRUE, plots the absolute value of the z-scores.
`...`	other plotting specifications (ggplot2)

Details

Inputting an object of class `network.impact` will return a line plot that shows the relative network impacts of each node.

`PROCRUSTESnet` *PROCRUSTESnet*

Description

Convenience function for simultaneously plotting two networks containing the same nodes.

Usage

```
PROCRUSTESnet(
  qgraph_net1,
  qgraph_net2,
  type1 = c("ordinal", "interval", "ratio", "mspline"),
  type2 = type1,
  MDSadj1 = NULL,
  MDSadj2 = NULL,
  stressTxt = F,
  congCoef = F,
  repulse = F,
  repulsion = 1,
  mdsArgs = list(),
  ...
)
```

Arguments

`qgraph_net1`	an object of type `qgraph`
`qgraph_net2`	an object of type `qgraph`. Contains the same nodes as `qgraph_net2`
`type1`	transformation function for first MDS, defaults to "ordinal"
`type2`	transformation function for second MDS, defaults to the same as `type1`

`MDSadj1`	to use a proximities matrix other than the adjacency matrix stored in `qgraph_net1`, provide it in this argument
`MDSadj2`	to use a proximities matrix other than the adjacency matrix stored in `qgraph_net2`, provide it in this argument
`stressTxt`	logical. Print the stress value in the lower left corner of the plots?
`congCoef`	logical. Print the congruence coefficient fo the two layouts?
`repulse`	logical. Add a small repulsion force with wordcloud package to avoid node overlap?
`repulsion`	scalar for the repulsion force. Larger values add more repulsion
`mdsArgs`	additional arguments in list format passed to `smacof::mds`
`...`	additional arguments passed to `qgraph`

Details

Each network's layout is determined by multidimensional scaling, and then the layouts are brought into a similar space by using the Procrustes algorithm.

A network plotted with multidimensional scaling can be interpreted based on the distances between nodes. Nodes close together represent closely associated nodes, whereas nodes that are far apart represent unassociated or negatively associated nodes.

The Procrustes algorithm brings the two layouts into a similar space through rotations and dilations that do not impact the fit of the MDS solutions. In this implementation, the second network is rotated and dilated to fit the first.

References

Jones, P. J., Mair, P., & McNally, R. J. (2018). Visualizing psychological networks: A tutorial in R. Frontiers in Psychology, 9, 1742. https://doi.org/10.3389/fpsyg.2018.01742

`social` *Simulated Social Engagement Data*

Description

This simulated dataset contains binary social engagement scores for 16 individuals. For 400 social media posts on a group forum, individuals were given a score of 1 if they engaged in group conversation regarding the post, and a score of 0 if they did not engage with the post.

Usage

```
social
```

Format

a dataframe. Columns represent individuals (nodes) and rows represent engagement in social media group conversations

Examples

```
head(social)
```

Index