Package 'influential'

October 7, 2026

Type Package

Title Identification and Classification of the Most Influential Nodes

Language en-US

Version 2.3.3

Description Provides functions for the identification, classification, and ranking of influential nodes and candidate features from network and omics data. The package implements the Integrated Value of Influence (IVI) for integrative network centrality analysis, the SIR-based Influence Ranking (SIRIR) model for unsupervised influence ranking, and the Experimental data-based Integrative Ranking (ExIR) model for prioritizing candidate driver, biomarker, and mediator features from experimental omics data. Functions are provided for network reconstruction from adjacency matrices and data frames, topological analysis, centrality calculation, assessment of associations between centrality measures, and conditional probability analysis. ExIR supports bulk and single-cell omics data, including matrices, sparse matrices, data frames, tibbles, and Seurat objects.

Imports igraph, irlba, edgeR, cli, Matrix, SeuratObject, tibble, janitor, ranger, foreach, doParallel, ggplot2, Rcpp

LinkingTo Rcpp

Suggests Hmisc (>= 4.3-0), mgcv (>= 1.8-31), nortest (>= 1.0-4), NNS (>= 0.4.7.1), BiocManager, readr, shiny, shinythemes, shinyWidgets, shinyjs, shinycssloaders, colourpicker, magrittr, DT, knitr, rmarkdown

Depends R (>= 4.0.0)

URL https://github.com/asalavaty/influential, https://asalavaty.github.io/influential/

BugReports https://github.com/asalavaty/influential/issues

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

VignetteBuilder knitr

NeedsCompilation yes

Author Adrian Salavaty [aut, cre] (ORCID:
<https://orcid.org/0000-0001-9320-5889>),
Mirana Ramialison [ths],
Peter D. Currie [ths]

Maintainer Adrian Salavaty <abbas.salavaty@gmail.com>

Repository CRAN

Date/Publication 2026-10-06 23:20:02 UTC

Contents

betweenness . . . 3
centrality.measures . . . 4
cent_network.vis . . . 5
clusterRank . . . 8
coexpression.adjacency . . . 9
coexpression.data . . . 10
collective.influence . . . 10
comp_manipulate . . . 12
cond.prob.analysis . . . 14
diff_data.assembly . . . 15
double.cent.assess . . . 16
double.cent.assess.noRegression . . . 17
exir . . . 19
exir.vis . . . 24
fcor . . . 27
hubness.score . . . 29
h_index . . . 31
ivi . . . 32
ivi.from.indices . . . 34
lh_index . . . 35
neighborhood.connectivity . . . 36
runShinyApp . . . 38
sif2igraph . . . 38
sirir . . . 39
spreading.score . . . 41

Index **43**

`betweenness` *Vertex betweenness centrality*

Description

This function and all of its descriptions have been obtained from the igraph package.

Usage

```
betweenness(
  graph,
  v = V(graph),
  directed = TRUE,
  weights = NULL,
  normalized = FALSE,
  ...
)
```

Arguments

`graph`	The graph to analyze (an igraph graph).
`v`	The vertices for which the vertex betweenness will be calculated.
`directed`	Logical, whether directed paths should be considered while determining the shortest paths.
`weights`	Optional positive weight vector for calculating weighted betweenness. If the graph has a weight edge attribute, then this is used by default. Weights are used to calculate weighted shortest paths, so they are interpreted as distances.
`normalized`	Logical scalar, whether to normalize the betweenness scores. If TRUE, then the results are normalized.
`...`	Additional arguments according to the original `betweenness` function in the package igraph.

Value

A numeric vector with the betweenness score for each vertex in v.

See Also

`ivi`, `cent_network.vis`, and `betweenness` for a complete description on this function

Other centrality functions: `clusterRank()`, `collective.influence()`, `h_index()`, `lh_index()`, `neighborhood.connectivity()`, `sirir()`

Examples

```
## Not run:
MyData <- coexpression.data
My_graph <- graph_from_data_frame(MyData)
GraphVertices <- V(My_graph)
My_graph_betweenness <- betweenness(My_graph, v = GraphVertices,
                                    directed = FALSE,
                                    normalized = FALSE)

## End(Not run)
```

`centrality.measures` *Centrality measures dataset*

Description

The centrality measures of a co-expression network of lncRNAs and mRNAs in lung adenocarcinoma

Usage

```
centrality.measures
```

Format

A data frame with 794 rows and 6 variables:

\

DC Degree Centrality

CR ClusterRank

NC Neighborhood Connectivity

LH_index Local H-index

BC Betweenness Centrality

CI Collective Influence ...

Source

doi:10.1111/jcmm.14458

cent_network.vis *Centrality-based network visualization*

Description

This function has been developed for the visualization of a network based on applying a centrality measure to the size and color of network nodes. You are also able to adjust the directedness and weight of connections. Some of the documentations of the arguments of this function have been adapted from ggplot2 and igraph packages. A shiny app has also been developed for the calculation of IVI as well as IVI-based network visualization, which is accessible using the 'influential::runShinyApp("IVI")' command. You can also access the shiny app online at https://influential.erc.monash.edu/.

Usage

```
cent_network.vis(
  graph,
  cent.metric,
  layout = "kk",
  node.group = NULL,
  node.color = "viridis",
  node.size.min = 3,
  node.size.max = 15,
  dist.power = 1,
  node.shape = "circle",
  stroke.size = 1.5,
  stroke.color = "identical",
  stroke.alpha = 0.6,
  show.labels = TRUE,
  label.cex = 0.4,
  label.color = "black",
  directed = FALSE,
  arrow.width = 25,
  arrow.length = 0.07,
  edge.width = 0.5,
  weighted = FALSE,
  edge.width.min = 0.2,
  edge.width.max = 1,
  edge.color = "grey75",
  edge.linetype = "solid",
  legend.position = "right",
  legend.direction = "vertical",
  legend.title = "Centrality\nmeasure",
  boxed.legend = TRUE,
  show.plot.title = TRUE,
  plot.title = "Centrality Measure-based Network",
  title.position = "center",
```

```
  show.bottom.border = TRUE,
  show.left.border = TRUE,
  seed = 1234
)
```

Arguments

graph	A graph (network) of the igraph class.
cent.metric	A numeric vector of the desired centrality measure previously calculated by any means. For example, you may use the function ivi for the calculation of the Integrated Value of Influence (IVI) of network nodes. Please note that if the centrality measure has been calculated by any means other than the influential package, make sure that the order of the values in the cent.metric vector is consistent with the order of vertices in the network (V(graph)).
layout	The layout to be used for organizing network nodes. Current available layouts include "kk", "star", "tree", "components", "circle", "automatic", "grid","sphere", "random", "dh", "drl", "fr", "gem", "graphopt", "lgl", "mds", and "sugiyama" (default is set to "kk"). For a complete description of different layouts and their underlying algorithms please refer to the function layout_.
node.group	A vector of the same length as the number of network nodes defining the group each node of the network belongs to.
node.color	A character string indicating the colormap option to use. Five options are available: "magma" (or "A"), "inferno" (or "B"), "plasma" (or "C"), "viridis" (or "D", the default option) and "cividis" (or "E").
node.size.min	The size of nodes with the lowest value of the centrality measure (default is set to 3).
node.size.max	The size of nodes with the highest value of the centrality measure (default is set to 15).
dist.power	The power to be used to visualize more distinction between nodes with high and low centrality measure values. The higher the power, the smaller the nodes with lower values of the centrality measure will become. Default is set to 1, meaning the relative sizes of nodes are reflective of their actual centrality measure values.
node.shape	The shape of nodes. Current available shapes include "circle","square", "diamond", "triangle", and "inverted triangle" (default is set to "circle"). You can also set different shapes to different groups of nodes by providing a character vector of shapes of nodes with the same length and order of network vertices. This is useful when plotting a network that include different type of node (for example, up- and down-regulated features).
stroke.size	The size of stroke (border) around the nodes (default is set to 1.5).
stroke.color	The color of stroke (border) around the nodes (default is set to "identical" meaning that the stroke color of a node will be identical to its corresponding node color). You can also set different colors to different groups of nodes by providing a character vector of colors of nodes with the same length and order of network vertices. This is useful when plotting a network that include different type of node (for example, up- and down-regulated features).

`stroke.alpha` The transparency of the stroke (border) around the nodes which should be a number between 0 and 1 (default is set to 0.6).

`show.labels` Logical scalar, whether to show node labels or not (default is set to TRUE).

`label.cex` The amount by which node labels should be scaled relative to the node sizes (default is set to 0.4).

`label.color` The color of node labels (default is set to "black").

`directed` Logical scalar, whether to draw the network as directed or not (default is set to FALSE).

`arrow.width` The width of arrows in the case the network is directed (default is set to 25).

`arrow.length` The length of arrows in inch in the case the network is directed (default is set to 0.07).

`edge.width` The constant width of edges if the network is unweighted (default is set to 0.5).

`weighted` Logical scalar, whether the network is a weighted network or not (default is set to FALSE).

`edge.width.min` The width of edges with the lowest weight (default is set to 0.2). This parameter is ignored for unweighted networks.

`edge.width.max` The width of edges with the highest weight (default is set to 1). This parameter is ignored for unweighted networks.

`edge.color` The color of edges (default is set to "grey75").

`edge.linetype` The line type of edges. Current available linetypes include `"twodash"`, `"longdash"`, `"dotdash"`, `"dotted"`, `"dashed"`, and `"solid"` (default is set to "solid").

`legend.position`
The position of legends ("none", "left", "right", "bottom", "top", or two-element numeric vector). The default is set to "right".

`legend.direction`
layout of items in legends ("horizontal" or "vertical"). The default is set to "vertical".

`legend.title` The legend title in the string format (default is set to "Centrality measure").

`boxed.legend` Logical scalar, whether to draw a box around the legend or not (default is set to TRUE).

`show.plot.title`
Logical scalar, whether to show the plot title or not (default is set to TRUE).

`plot.title` The plot title in the string format (default is set to "Centrality Measure-based Network").

`title.position` The position of title ("left", "center", or "right"). The default is set to "center".

`show.bottom.border`
Logical scalar, whether to draw the bottom border line (default is set to TRUE).

`show.left.border`
Logical scalar, whether to draw the left border line (default is set to TRUE).

`seed` A single value, interpreted as an integer to be used for random number generation for preparing the network layout (default is set to 1234).

Value

A plot with the class ggplot.

See Also

ivi

Other visualization functions: exir.vis()

Examples

```
## Not run:
MyData <- coexpression.data
My_graph <- graph_from_data_frame(MyData)
Graph_IVI <- ivi(graph = My_graph, mode = "all")
Graph_IVI_plot <- cent_network.vis(graph = My_graph, cent.metric = Graph_IVI,
                                   legend.title = "IVI",
                                   plot.title = "IVI-based Network")

## End(Not run)
```

---

clusterRank *ClusterRank (CR)*

Description

This function calculates the ClusterRank of input vertices and works with both directed and undirected networks. This function and all of its descriptions have been adapted from the centiserve package with some minor modifications. ClusterRank is a local ranking algorithm which takes into account not only the number of neighbors and the neighbors' influences, but also the clustering coefficient.

Usage

```
clusterRank(
  graph,
  vids = V(graph),
  directed = FALSE,
  loops = TRUE,
  ncores = "default",
  verbose = FALSE
)
```

Arguments

graph	The input graph as igraph object
vids	Vertex sequence, the vertices for which the centrality values are returned. Default is all vertices.

directed	Logical scalar, whether to directed graph is analyzed. This argument is ignored for undirected graphs.
loops	Logical; whether the loop edges are also counted.
ncores	Integer; the number of cores to be used for parallel processing. If ncores == "default" (default), the number of cores to be used will be the max(number of available cores) - 1. We recommend leaving ncores argument as is (ncores = "default").
verbose	Logical; whether the accomplishment of different stages of the algorithm should be printed (default is FALSE).

Value

A numeric vector contaning the ClusterRank centrality scores for the selected vertices.

See Also

ivi, cent_network.vis

Other centrality functions: betweenness(), collective.influence(), h_index(), lh_index(), neighborhood.connectivity(), sirir()

Examples

```
## Not run:
MyData <- coexpression.data
My_graph <- graph_from_data_frame(MyData)
GraphVertices <- V(My_graph)
cr <- clusterRank(graph = My_graph, vids = GraphVertices,
directed = FALSE, loops = TRUE, ncores = 1)

## End(Not run)
```

---

coexpression.adjacency

Adjacency matrix

Description

The adjacency matrix of a co-expression network of lncRNAs and mRNAs in lung adenocarcinoma that was generated using igraph functions

Usage

```
coexpression.adjacency
```

Format

A data frame with 794 rows and 794 variables:

lncRNA lncRNA symbol

lncRNA lncRNA symbol ...

Source

doi:10.1111/jcmm.14458

---

coexpression.data *Co-expression dataset*

Description

A co-expression dataset of lncRNAs and mRNAs in lung adenocarcinoma

Usage

```
coexpression.data
```

Format

A data frame with 2410 rows and 2 variables:

lncRNA lncRNA symbol

Coexpressed.Gene Co-expressed gene symbol ...

Source

doi:10.1111/jcmm.14458

---

collective.influence *Collective Influence (CI)*

Description

This function calculates the collective influence of input vertices and works with both directed and undirected networks. This function and its descriptions are obtained from https://github.com/ronammar/collective_influence with minor modifications. Collective Influence as described by Morone & Makse (2015). In simple terms, it is the product of the reduced degree (degree - 1) of a node and the total (sum of) reduced degrees of all nodes at a distance d from the node.

Usage

```
collective.influence(
  graph,
  vertices = V(graph),
  mode = "all",
  d = 3,
  verbose = FALSE
)
```

Arguments

graph	A graph (network) of the igraph class.
vertices	A vector of desired vertices, which could be obtained by the V function.
mode	The mode of collective influence depending on the directedness of the graph. If the graph is undirected, the mode "all" should be specified. Otherwise, for the calculation of collective influence based on incoming connections select "in" and for the outgoing connections select "out". Also, if all of the connections are desired, specify the "all" mode. Default mode is set to "all".
d	The distance, expressed in number of steps from a given node (default=3). Distance must be > 0. According to Morone & Makse (https://doi.org/10.1038/nature14604), optimal results can be reached at d=3,4, but this depends on the size/"radius" of the network. NOTE: the distance d is not inclusive. This means that nodes at a distance of 3 from our node-of-interest do not include nodes at distances 1 and 2. Only 3.
verbose	Logical; whether the accomplishment of different stages of the algorithm should be printed (default is FALSE).

Value

A vector of collective influence for each vertex of the graph corresponding to the order of vertices output by V(graph).

See Also

ivi, cent_network.vis

Other centrality functions: betweenness(), clusterRank(), h_index(), lh_index(), neighborhood.connectivity(), sirir()

Examples

```
## Not run:
MyData <- coexpression.data
My_graph <- graph_from_data_frame(MyData)
GraphVertices <- V(My_graph)
ci <- collective.influence(graph = My_graph, vertices = GraphVertices, mode = "all", d=3)

## End(Not run)
```

comp_manipulate *Computational manipulation of cells*

Description

This function works based on the SIRIR (SIR-based Influence Ranking) model and could be applied on the output of the ExIR model or any other independent association network. For feature (gene/protein/etc.) knockout the SIRIR model is used to remove the feature from the network and assess its impact on the flow of information (signaling) within the network. On the other hand, in case of up-regulation a node similar to the desired node is added to the network with exactly the same connections (edges) as of the original node. Next, the SIRIR model is used to evaluate the difference in the flow of information/signaling after adding (up-regulating) the desired feature/node compared with the original network. In case you are applying this function on the output of ExIR model, you may note that as the gene/protein knockout would impact on the integrity of the under-investigation network as well as the networks of other overlapping biological processes/pathways, it is recommended to select those features that simultaneously have the highest (most significant) ExIR-based rank and lowest knockout rank. In contrast, as the up-regulation would not affect the integrity of the network, you may select the features with highest (most significant) ExIR-based and up-regulation-based ranks. A shiny app has also been developed for Running the ExIR model, visualization of its results as well as computational simulation of knockout and/or up-regulation of its top candidate outputs, which is accessible using the 'influential::runShinyApp("ExIR")' command. You can also access the shiny app online at https://influential.erc.monash.edu/.

Usage

```
comp_manipulate(
  exir_output = NULL,
  graph = NULL,
  ko_vertices = igraph::V(graph),
  upregulate_vertices = igraph::V(graph),
  beta = 0.5,
  gamma = 1,
  no.sim = 100,
  node_verbose = FALSE,
  loop_verbose = TRUE,
  ncores = "default",
  seed = 1234
)
```

Arguments

exir_output	The output of the ExIR model (optional).
graph	A graph (network) of the igraph class (not required if the exir_output is inputted).
ko_vertices	A vector of desired vertices/features to knockout. Default is set to V(graph) meaning to assess the knockout of all vertices/features.

upregulate_vertices
: A vector of desired vertices/features to up-regulate. Default is set to V(graph) meaning to assess the up-regulation of all vertices/features.

beta
: Non-negative scalar corresponding to the SIRIR model. The rate of infection of an individual that is susceptible and has a single infected neighbor. The infection rate of a susceptible individual with n infected neighbors is n times beta. Formally this is the rate parameter of an exponential distribution.

gamma
: Positive scalar corresponding to the SIRIR model. The rate of recovery of an infected individual. Formally, this is the rate parameter of an exponential distribution.

no.sim
: Integer scalar corresponding to the SIRIR model. The number of simulation runs to perform SIR model on for the original network as well perturbed networks generated by leave-one-out technique. You may choose a different no.sim based on the available memory on your system.

node_verbose
: Logical; whether the process of Parallel Socket Cluster creation should be printed (default is FALSE).

loop_verbose
: Logical; whether the accomplishment of the evaluation of network nodes in each loop should be printed (default is TRUE).

ncores
: Integer; the number of cores to be used for parallel processing. If ncores == "default" (default), the number of cores to be used will be the max(number of available cores) - 1. We recommend leaving ncores argument as is (ncores = "default").

seed
: A single value, interpreted as an integer to be used for random number generation.

Value

Depending on the input data, a list including one to three data frames of knockout/up-regulation rankings.

See Also

exir, sirir, and sir for a complete description on SIR model

Other integrative ranking functions: exir(), hubness.score(), ivi(), ivi.from.indices(), spreading.score()

Examples

```
## Not run:
set.seed(1234)
My_graph <- igraph::sample_gnp(n=50, p=0.05)
GraphVertices <- V(My_graph)
Computational_manipulation <- comp_manipulate(graph = My_graph, beta = 0.5,
                                              gamma = 1, no.sim = 10, seed = 1234)

## End(Not run)
```

`cond.prob.analysis` *Conditional probability of deviation from means*

Description

This function calculates the conditional probability of deviation of two centrality measures (or any two other continuous variables) from their corresponding means in opposite directions.

Usage

```
cond.prob.analysis(data, nodes.colname, Desired.colname, Condition.colname)
```

Arguments

`data`	A data frame containing the values of two continuous variables and the name of observations (nodes).
`nodes.colname`	The character format (quoted) name of the column containing the name of observations (nodes).
`Desired.colname`	The character format (quoted) name of the column containing the values of the desired variable.
`Condition.colname`	The character format (quoted) name of the column containing the values of the condition variable.

Value

A list of two objects including the conditional probability of deviation of two centrality measures (or any two other continuous variables) from their corresponding means in opposite directions based on both the entire network and the split-half random sample of network nodes.

See Also

Other centrality association assessment functions: `double.cent.assess()`, `double.cent.assess.noRegression()`

Examples

```
## Not run:
MyData <- centrality.measures
My.conditional.prob <- cond.prob.analysis(data = MyData,
                                          nodes.colname = rownames(MyData),
                                          Desired.colname = "BC",
                                          Condition.colname = "NC")

## End(Not run)
```

`diff_data.assembly` *Assembling the differential/regression data*

Description

This function assembles a dataframe required for running the `ExIR` model. You may provide as many differential/regression data as you wish. Also, the datasets should be filtered beforehand according to your desired thresholds and, consequently, should only include the significant data. Each dataset provided should be a dataframe with one or two columns. The first column should always include differential/regression values and the second one (if provided) the significance values. Please also note that the significance (adjusted P-value) column is mandatory for differential datasets.

Usage

```
diff_data.assembly(...)
```

Arguments

`...`	Desired datasets/dataframes.

Value

A dataframe including the collective list of features in rows and all of the differential/regression data and their statistical significance in columns with the same order provided by the user.

See Also

`exir`

Examples

```
## Not run:
my.Diff_data <- diff_data.assembly(Differential_data1,
                                   Differential_data2,
                                   Regression_data1)

## End(Not run)
```

`double.cent.assess` *Assessment of innate features and associations of two network centrality measures (dependent and independent)*

Description

This function assesses innate features and the association of two centrality measures (or any two other continuous variables) from the aspect of distribution mode, dependence, linearity, monotonicity, partial-moments based correlation, and conditional probability of deviating from corresponding means in opposite direction. This function assumes one variable as dependent and the other as independent for regression analyses. The non-linear nature of the association of two centrality measures is evaluated based on generalized additive models (GAM). The monotonicity of the association is evaluated based on comparing the squared coefficient of Spearman correlation and R-squared of rank regression analysis. Also, the correlation between two variables is assessed via non-linear non-parametric statistics (NNS). For the conditional probability assessment, the independent variable is considered as the condition variable.

Usage

```
double.cent.assess(
  data,
  nodes.colname,
  dependent.colname,
  independent.colname,
  plot = FALSE
)
```

Arguments

`data`	A data frame containing the values of two continuous variables and the name of observations (nodes).
`nodes.colname`	The character format (quoted) name of the column containing the name of observations (nodes).
`dependent.colname`	The character format (quoted) name of the column containing the values of the dependent variable.
`independent.colname`	The character format (quoted) name of the column containing the values of the independent variable.
`plot`	logical; FALSE (default) Plots quadrant means of NNS correlation analysis.

Value

A list of 11 objects including:

- Summary of the basic statistics of two centrality measures (or any two other continuous variables).

- The results of normality assessment of two variable (p-value > 0.05 imply that the variable is normally distributed).

- Description of the normality assessment of the dependent variable.

- Description of the normality assessment of the independent variable.

- Results of the generalized additive modeling (GAM) of the data.

- The association type based on simultaneous consideration of normality assessment, GAM Computation with smoothness estimation, Spearman correlation, and ranked regression analysis of splines.

- The Hoeffding's D Statistic of dependence (ranging from -0.5 to 1).

- Description of the dependence significance.

- Correlation between variables based on the NNS method.

- The last two objects are the conditional probability of deviation of two centrality measures from their corresponding means in opposite directions based on both the entire network and the split-half random sample of network nodes.

See Also

ad.test for Anderson-Darling test for normality, gam for Generalized additive models with integrated smoothness estimation, lm for Fitting Linear Models, hoeffd for Matrix of Hoeffding's D Statistics, and NNS.dep for NNS Dependence

Other centrality association assessment functions: cond.prob.analysis(), double.cent.assess.noRegression()

Examples

```
## Not run:
MyData <- centrality.measures
My.metrics.assessment <- double.cent.assess(data = MyData,
                                            nodes.colname = rownames(MyData),
                                            dependent.colname = "BC",
                                            independent.colname = "NC")

## End(Not run)
```

`double.cent.assess.noRegression`

Assessment of innate features and associations of two network centrality measures

---

Description

This function assesses innate features and the association of two centrality measures (or any two other continuous variables) from the aspect of distribution mode, dependence, linearity, partial-moments based correlation, and conditional probability of deviating from corresponding means in opposite direction (centrality2 is used as the condition variable). This function doesn't consider which variable is dependent and which one is independent and no regression analysis is done. Also, the correlation between two variables is assessed via non-linear non-parametric statistics (NNS). For the conditional probability assessment, the centrality2 variable is considered as the condition variable.

Usage

```
double.cent.assess.noRegression(
  data,
  nodes.colname,
  centrality1.colname,
  centrality2.colname
)
```

Arguments

`data`	A data frame containing the values of two continuous variables and the name of observations (nodes).
`nodes.colname`	The character format (quoted) name of the column containing the name of observations (nodes).
`centrality1.colname`	The character format (quoted) name of the column containing the values of the Centrality_1 variable.
`centrality2.colname`	The character format (quoted) name of the column containing the values of the Centrality_2 variable.

Value

A list of nine objects including:

- Summary of the basic statistics of two centrality measures (or any two other continuous variables).

- The results of normality assessment of two variable (p-value > 0.05 imply that the variable is normally distributed).

- Description of the normality assessment of the centrality1 (first variable).

- Description of the normality assessment of the centrality2 (second variable).

- The Hoeffding's D Statistic of dependence (ranging from -0.5 to 1).

- Description of the dependence significance.

- Correlation between variables based on the NNS method.

- The last two objects are the conditional probability of deviation of two centrality measures from their corresponding means in opposite directions based on both the entire network and the split-half random sample of network nodes.

See Also

`ad.test` for Anderson-Darling test for normality, `hoeffd` for Matrix of Hoeffding's D Statistics, and `NNS.dep` for NNS Dependence

Other centrality association assessment functions: `cond.prob.analysis()`, `double.cent.assess()`

Examples

```
## Not run:
MyData <- centrality.measures
My.metrics.assessment <- double.cent.assess.noRegression(data = MyData,
                                                         nodes.colname = rownames(MyData),
                                                         centrality1.colname = "BC",
                                                         centrality2.colname = "NC")

## End(Not run)
```

---

exir *Experimental data-based Integrated Ranking*

Description

This function runs the Experimental data-based Integrated Ranking (ExIR) model for the classification and ranking of top candidate features. The input data could come from any type of experiment such as transcriptomics and proteomics. A shiny app has also been developed for Running the ExIR model, visualization of its results as well as computational simulation of knockout and/or up-regulation of its top candidate outputs, which is accessible using the 'influential::runShinyApp("ExIR")' command. You can also access the shiny app online at https://influential.erc.monash.edu/.

Usage

```
exir(
  Desired_list = NULL,
  Diff_data,
  Diff_value,
  Regr_value = NULL,
  Sig_value,
  Exptl_data,
  Exptl_data_type = c("bulk", "sc"),
  condition = "condition",
  Exptl_data_orientation = c("features_rows", "samples_rows"),
  assay = "RNA",
  layer = "counts",
  normalize = FALSE,
  pseudo_sample = FALSE,
  pseudo_samples_per_group = 100,
  Exptl_data_size_check = TRUE,
  feature_filter = TRUE,
  min_feature_prevalence = NULL,
  min_feature_total = NULL,
  min_feature_variance = 1e-12,
  always_keep_diff_features = TRUE,
  cor_thresh_method = "mr",
  r = 0.5,
```

```
  mr = 20,
  max.connections = 50000,
  alpha = 0.05,
  num_trees = 500,
  mtry = NULL,
  num_permutations = 50,
  inf_const = 10^10,
  ncores = "default",
  seed = 1234,
  verbose = TRUE
)
```

Arguments

`Desired_list`	(Optional) A character vector of your desired features. This vector could be, for instance, a list of features obtained from cluster analysis, time-course analysis, or a list of dysregulated features with a specific sign.
`Diff_data`	A dataframe of all significant differential/regression data and their statistical significance values (p-value/adjusted p-value). Note that the differential data should be in the log fold-change (log2FC) format. You may have selected a proportion of the differential data as the significant ones according to your desired thresholds. A function, named `diff_data.assembly`, has also been provided for the convenient assembling of the Diff_data dataframe.
`Diff_value`	An integer vector containing the column number(s) of the differential data in the Diff_data dataframe. The differential data could result from any type of differential data analysis. One example could be the fold changes (FCs) obtained from differential expression analyses. The user may provide as many differential data as he/she wish.
`Regr_value`	(Optional) An integer vector containing the column number(s) of the regression data in the Diff_data dataframe. The regression data could result from any type of regression data analysis or other analyses such as time-course data analyses that are based on regression models.
`Sig_value`	An integer vector containing the column number(s) of the significance values (p-value/adjusted p-value) of both differential and regression data (if provided). Providing significance values for the regression data is optional.
`Exptl_data`	Experimental data used by the ExIR model. This can be a data frame, tibble, matrix, sparse matrix such as a `dgCMatrix`, or a Seurat object. For non-Seurat inputs, the expected orientation is controlled by `Exptl_data_orientation`. By default, features/genes are expected to be in rows and samples/cells in columns, which is the usual omics layout. Internally, ExIR converts the data to its required analysis format, with samples/cells in rows and features in columns.
`Exptl_data_type`	Character string specifying the experimental data type. One of `"bulk"` or `"sc"`. This is used for data-type checks, optional normalization, and pseudo-sampling. For `"bulk"`, the input expression data may be either already normalized/log-transformed or raw count-like data when `normalize = TRUE`. For `"sc"`, raw counts are recommended and raw counts are required when `pseudo_sample = TRUE`.

condition
: A character string or character/factor vector specifying the sample/cell conditions. If a single character string is supplied, it is interpreted as the name of the condition column/row in Exptl_data, or as the name of a metadata column when Exptl_data is a Seurat object. If a vector is supplied, it must have the same length and order as the samples/cells in Exptl_data. Default is "condition".

Exptl_data_orientation
: Character string specifying the orientation of non-Seurat Exptl_data. One of "features_rows" or "samples_rows". If "features_rows", features are rows and samples/cells are columns. If "samples_rows", samples/cells are rows and features are columns. Default is "features_rows".

assay
: Character string specifying the assay to use when Exptl_data is a Seurat object. Default is "RNA".

layer
: Character string specifying the assay layer to use when Exptl_data is a Seurat object. For pseudo-sampling of single-cell data, this should usually be a raw-count layer such as "counts". Default is "counts".

normalize
: Logical; whether to normalize count-like input data using TMM normalization followed by logCPM transformation with **edgeR**. Default is FALSE. For Exptl_data_type = "bulk", this can be used when raw bulk RNA-seq count-like data are supplied. This normalization strategy is appropriate for many bulk omics count datasets, especially bulk RNA-seq, but users should confirm that TMM/logCPM normalization is suitable for their specific data modality. If the data modality requires a different normalization strategy, users should pre-normalize their data and set normalize = FALSE. For Exptl_data_type = "sc", normalization is automatically applied after pseudo-bulking when pseudo_sample = TRUE. If Exptl_data_type = "sc" and pseudo_sample = FALSE, users should provide pre-normalized single-cell data and keep normalize = FALSE.

pseudo_sample
: Logical; whether to perform pseudo-sampling before running ExIR. Pseudo-sampling is recommended when the number of cells/samples is greater than 500 or when computational resources are limited. For bulk data, pseudo-sampling averages normalized log-expression values within non-overlapping condition-specific groups. For single-cell data, pseudo-sampling sums raw counts within non-overlapping condition-specific groups, followed by TMM normalization and logCPM transformation using **edgeR**. Default is FALSE.

pseudo_samples_per_group
: Integer specifying the target number of pseudo-samples to generate per condition group when pseudo_sample = TRUE. For example, if one condition contains 500 cells/samples and pseudo_samples_per_group = 100, each pseudo-sample will contain 5 cells/samples. If another condition contains 536 cells/samples, 64 pseudo-samples will contain 5 cells/samples and 36 pseudo-samples will contain 6 cells/samples. Default is 100.

Exptl_data_size_check
: Logical; whether to check the number of input samples/cells and, in interactive sessions, prompt the user to consider pseudo-sampling when more than 500 samples/cells are provided and pseudo_sample = FALSE. In non-interactive sessions, a message is shown and the function continues. Default is TRUE.

feature_filter Logical; whether to apply conservative feature filtering before running RF, PCA and correlation analysis. This filter is not a highly variable gene filter. It removes only features with insufficient expression/prevalence or essentially zero variance, which are unlikely to produce reliable correlations. Default is TRUE.

min_feature_prevalence
Integer or NULL; minimum number of samples/cells/ pseudo-samples in which a feature must be non-zero to be retained. If NULL, an adaptive conservative threshold is used based on Exptl_data_type, pseudo-sampling, and sample size.

min_feature_total
Numeric or NULL; minimum total abundance/count/expression support required for a feature to be retained. If NULL, this criterion is not used. For raw count-like data, values such as 10 or 20 may be useful. For normalized/log-scale data, users should usually leave this as NULL.

min_feature_variance
Numeric; minimum variance required for a feature to be retained. This is intended only to remove zero-variance or near-zero-variance features, not to perform HVG selection. Default is 1e-12.

always_keep_diff_features
Logical; whether to always retain features present in Diff_data and Desired_list, even if they fail the conservative expression/ prevalence filters. This helps preserve candidate differential features while still reducing uninformative non-DE background features. Default is TRUE.

cor_thresh_method
A character string indicating the method for filtering the correlation results, either "mr" (default; Mutual Rank) or "cor.coefficient".

r The threshold of Spearman correlation coefficient for the selection of correlated features (default is 0.5).

mr An integer determining the threshold of mutual rank for the selection of correlated features (default is 20). Note that higher mr values considerably increase the computation time.

max.connections
The maximum number of connections to be included in the association network. Higher max.connections might increase the computation time, cost, and accuracy of the results (default is 50,000).

alpha The threshold of the statistical significance (p-value) used throughout the entire model (default is 0.05)

num_trees Number of trees to be used for the random forests classification (supervised machine learning). Default is set to 500.

mtry Number of features to possibly split at in each node. Default is the (rounded down) square root of the number of variables. Alternatively, a single argument function returning an integer, given the number of independent variables.

num_permutations
Number of permutations to be used for computation of the statistical significance (p-values) of the importance scores resulted from random forests classification (default is 50).

`inf_const` The constant value to be multiplied by the maximum absolute value of differential (logFC) values for the substitution with infinite differential values. This results in noticeably high biomarker values for features with infinite differential values compared with other features. Having said that, the user can still use the biomarker rank to compare all of the features. This parameter is ignored if no infinite value is present within Diff_data. However, this is used in the case of sc-seq experiments where some genes are uniquely expressed in a specific cell-type and consequently get infinite differential values. Note that the sign of differential value is preserved (default is 10^10).

`ncores` Integer; the number of cores to be used for parallel processing. If ncores == "default" (default), the number of cores to be used will be the max(number of available cores) - 1. We recommend leaving ncores argument as is (ncores = "default").

`seed` The seed to be used for all of the random processes throughout the model (default is 1234).

`verbose` Logical; whether to display formatted progress messages and a progress bar using **cli**. If `TRUE`, ExIR reports the major analysis stages, selected warnings, and a final output summary. Default is `TRUE`.

Value

A list of one graph and one to four tables including:

- Driver table: Top candidate drivers

- DE-mediator table: Top candidate differentially expressed/abundant mediators

- nonDE-mediator table: Top candidate non-differentially expressed/abundant mediators

- Biomarker table: Top candidate biomarkers

The number of returned tables depends on the input data and specified arguments.

See Also

`exir.vis`, `diff_data.assembly`, `prcomp`, `ranger`, `importance_pvalues`

Other integrative ranking functions: `comp_manipulate()`, `hubness.score()`, `ivi()`, `ivi.from.indices()`, `spreading.score()`

Examples

```
## Not run:
MyDesired_list <- Desiredlist
MyDiff_data <- Diffdata
Diff_value <- c(1,3,5)
Regr_value <- 7
Sig_value <- c(2,4,6,8)
MyExptl_data <- Exptldata
condition <- "condition"
My.exir <- exir(Desired_list = MyDesired_list,
               Diff_data = MyDiff_data, Diff_value = Diff_value,
               Regr_value = Regr_value, Sig_value = Sig_value,
```

```
                Exptl_data = MyExptl_data, condition = condition)

## End(Not run)
```

`exir.vis` *Visualization of ExIR results*

Description

This function has been developed for the visualization of ExIR results. Some of the documentations of the arguments of this function have been adapted from ggplot2 package. A shiny app has also been developed for Running the ExIR model, visualization of its results as well as computational simulation of knockout and/or up-regulation of its top candidate outputs, which is accessible using the 'influential::runShinyApp("ExIR")' command. You can also access the shiny app online at https://influential.erc.monash.edu/.

Usage

```
exir.vis(
  exir.results,
  synonyms.table = NULL,
  n = 10,
  driver.type = "combined",
  biomarker.type = "combined",
  show.drivers = TRUE,
  show.biomarkers = TRUE,
  show.de.mediators = TRUE,
  show.nonDE.mediators = TRUE,
  basis = "Rank",
  label.position = "top",
  nrow = 1,
  dot.size.min = 2,
  dot.size.max = 5,
  type.color = "viridis",
  stroke.size = 1.5,
  stroke.alpha = 1,
  dot.color.low = "blue",
  dot.color.high = "red",
  legend.position = "bottom",
  legend.direction = "vertical",
  legends.layout = "horizontal",
  boxed.legend = TRUE,
  show.plot.title = TRUE,
  plot.title = "auto",
  title.position = "left",
  plot.title.size = 12,
  show.plot.subtitle = TRUE,
```

```
  plot.subtitle = "auto",
  subtitle.position = "left",
  y.axis.title = "Feature",
  show.y.axis.grid = TRUE
)
```

Arguments

exir.results	An object of class "ExIR_Result" which is the output of the function "exir".
synonyms.table	(Optional) A data frame or matrix with two columns including a column for the used feature names in the input data of the "exir" model and the other column their synonyms. Note, the original feature names should always come as the first column and the synonyms as the second one. For example, if the original feature names used for running the "exir" model are Ensembl gene symbols, you can use their HGNC synonyms in the second column to be used for the visualization of the ExIR results
n	An integer specifying the number of top candidates to be selected from each category of ExIR results (default is set to 10).
driver.type	A string specifying the type of drivers to be used for the selection of top N candidates. The possible types include "combined" (meaning both driver types), "accelerator" and "decelerator" (default is set to "combined").
biomarker.type	A string specifying the type of biomarkers to be used for the selection of top N candidates. Possible types include "combined" (meaning both biomarker types), "up-regulated" and "down-regulated" (default is set to "combined").
show.drivers	Logical scalar, whether to show Drivers or not (default is set to TRUE).
show.biomarkers	Logical scalar, whether to show Biomarkers or not (default is set to TRUE).
show.de.mediators	Logical scalar, whether to show DE-mediators or not (default is set to TRUE).
show.nonDE.mediators	Logical scalar, whether to show nonDE-mediators or not (default is set to TRUE).
basis	A string specifying the basis for the selection of top N candidates from each category of the results. Possible options include "Rank" and "Adjusted p-value" (default is set to "Rank").
label.position	By default, the labels are displayed on the top of the plot. Using label.position it is possible to place the labels on either of the four sides by setting label.position = c("top", "bottom", "left", "right").
nrow	Number of rows of the plot (default is set to 1).
dot.size.min	The size of dots with the lowest statistical significance (default is set to 2).
dot.size.max	The size of dots with the highest statistical significance (default is set to 5).
type.color	A character string or function indicating the color palette to be used for the visualization of different types of candidates. You may choose one of the Viridis palettes including "magma" (or "A"), "inferno" (or "B"), "plasma" (or "C"), "viridis" (or "D", the default option) and "cividis" (or "E"), use a function specifying your desired palette, or manually specify the vector of colors for different types.

`stroke.size` The size of stroke (border) around the dots (default is set to 1.5).

`stroke.alpha` The transparency of the stroke (border) around the dots which should be a number between 0 and 1 (default is set to 1).

`dot.color.low` The color to be used for the visualization of dots (features) with the lowest Z-score values (default is set to "blue").

`dot.color.high` The color to be used for the visualization of dots (features) with the highest Z-score values (default is set to "red").

`legend.position`
The position of legends ("none", "left", "right", "bottom", "top", or two-element numeric vector). The default is set to "bottom".

`legend.direction`
Layout of items in legends ("horizontal" or "vertical"). The default is set to "vertical".

`legends.layout` Layout of different legends of the plot ("horizontal" or "vertical"). The default is set to "horizontal".

`boxed.legend` Logical scalar, whether to draw a box around the legend or not (default is set to TRUE).

`show.plot.title`
Logical scalar, whether to show the plot title or not (default is set to TRUE).

`plot.title` The plot title in the string format (default is set to "auto" which automatically generates a title for the plot).

`title.position` The position of title ("left", "center", or "right"). The default is set to "left".

`plot.title.size`
The font size of the plot title (default is set to 12).

`show.plot.subtitle`
Logical scalar, whether to show the plot subtitle or not (default is set to TRUE).

`plot.subtitle` The plot subtitle in the string format (default is set to "auto" which automatically generates a subtitle for the plot).

`subtitle.position`
The position of subtitle ("left", "center", or "right"). The default is set to "left".

`y.axis.title` The title of the y axis (features title). Default is set to "Features".

`show.y.axis.grid`
Logical scalar, whether to draw y axis grid lines (default is set to TRUE).

Value

A plot with the class ggplot.

See Also

`exir`

Other visualization functions: `cent_network.vis()`

Examples

```
## Not run:
MyResults <- exir.results
ExIR.plot <- exir.vis(exir.results = MyResults, n = 5)

## End(Not run)
```

`fcor` *Fast correlation and mutual rank analysis*

Description

This function calculates Pearson/Spearman correlations between all pairs of features in a matrix/dataframe much faster than the base R cor function. It can also calculate correlations between all pairs of features from two input matrices/dataframes when ‘data2‘ is provided. It is also possible to simultaneously calculate mutual rank (MR) of correlations as well as their p-values and adjusted p-values. Additionally, this function can automatically combine and flatten the result matrices. Selecting correlated features using an MR-based threshold rather than based on their correlation coefficients or an arbitrary p-value is more efficient and accurate in inferring functional associations in systems, for example in gene regulatory networks.

Usage

```
fcor(
  data,
  data2 = NULL,
  na_to_zero = TRUE,
  method = "spearman",
  mutualRank = TRUE,
  mutualRank_mode = "unsigned",
  pvalue = FALSE,
  adjust = "BH",
  flat = TRUE,
  remove_self = TRUE,
  remove_duplicate_pairs = TRUE
)
```

Arguments

`data`	a numeric dataframe/matrix with features on columns and samples/observations on rows. If ‘data2‘ is not provided, correlations are calculated between all pairs of features in ‘data‘.
`data2`	an optional numeric dataframe/matrix with features on columns and samples/observations on rows. If provided, correlations are calculated between all features in ‘data‘ and all features in ‘data2‘. ‘data‘ and ‘data2‘ must have the same number of rows, and the rows must correspond to the same samples/observations in the same order. Default is ‘NULL‘.

na_to_zero	logical, whether to convert NAs to 0 in the output (default) or not.
method	a character string indicating which correlation coefficient is to be computed. One of ‘"pearson"‘ or ‘"spearman"‘ (default).
mutualRank	logical, whether to calculate mutual ranks of correlations or not.
mutualRank_mode	a character string indicating whether to rank based on ‘"signed"‘ or ‘"unsigned"‘ (default) correlation values. In the ‘"unsigned"‘ mode, only the level of a correlation value is important and not its sign; therefore, the function ranks the absolute values of correlations. Options are ‘"unsigned"‘ and ‘"signed"‘.
pvalue	logical, whether to calculate p-values of correlations or not.
adjust	p-value correction method when ‘pvalue = TRUE‘, a character string including any of ‘"BH"‘ (default), ‘"bonferroni"‘, ‘"holm"‘, ‘"hochberg"‘, ‘"hommel"‘, or ‘"none"‘.
flat	logical, whether to combine and flatten the result matrices or not.
remove_self	logical, whether to remove self-correlations from the flattened output when ‘data2‘ is provided. This is useful when ‘data2‘ contains some or all of the same features as ‘data‘. Default is ‘TRUE‘.
remove_duplicate_pairs	logical, whether to remove duplicate undirected feature pairs from the flattened output when ‘data2‘ is provided. This is useful when ‘data2‘ contains the same features as ‘data‘, because pairs such as ‘geneA-geneB‘ and ‘geneB-geneA‘ may otherwise both be returned. Default is ‘TRUE‘.

Details

When ‘data2 = NULL‘, the function performs the standard all-pairs correlation analysis among the features of ‘data‘. When ‘data2‘ is provided, the function performs a rectangular correlation analysis between the features of ‘data‘ and the features of ‘data2‘.

For Spearman correlation with ‘data2‘, the two input matrices are internally combined before rank transformation so that feature-wise ranks are calculated consistently across the same samples/observations.

When ‘mutualRank = TRUE‘ and ‘data2‘ is provided, the calculated MR values are based on the rectangular correlation space between ‘data‘ and ‘data2‘. Therefore, these MR values are not necessarily identical to MR values obtained from a full all-pairs correlation matrix followed by post hoc filtering.

Value

Depending on the input data and the value of ‘flat‘, a dataframe or list including ‘cor‘ correlation coefficients, ‘mr‘ mutual ranks of correlation coefficients, ‘p‘ p-values of correlation coefficients, and ‘p.adj‘ adjusted p-values. If ‘data2‘ is not provided and ‘flat = TRUE‘, the flattened output contains the upper triangle of the all-pairs correlation matrix. If ‘data2‘ is provided and ‘flat = TRUE‘, the flattened output contains feature pairs between ‘data‘ and ‘data2‘.

See Also

p.adjust and graph_from_data_frame

Examples

```
## Not run:
set.seed(1234)

# All-pairs correlation among features
data <- datasets::attitude
cor <- fcor(data = data)

# Correlation between two sets of features
data1 <- mtcars[, 1:4]
data2 <- mtcars[, 5:11]
cor_rect <- fcor(data = data1, data2 = data2)

# Correlation between selected features and all features
selected_data <- mtcars[, 1:4]
all_data <- mtcars
cor_selected_all <- fcor(data = selected_data, data2 = all_data)

## End(Not run)
```

`hubness.score` *Hubness score*

Description

This function calculates the Hubness score of the desired nodes from a graph. Hubness score reflects the power of each node in its surrounding environment and is one of the major components of the IVI.

Usage

```
hubness.score(
  graph,
  vertices = V(graph),
  directed = FALSE,
  mode = "all",
  loops = TRUE,
  scale = "range",
  verbose = FALSE
)
```

Arguments

`graph`	A graph (network) of the igraph class.
`vertices`	A vector of desired vertices, which could be obtained by the V function.
`directed`	Logical scalar, whether to directed graph is analyzed. This argument is ignored for undirected graphs.

mode	The mode of Hubness score depending on the directedness of the graph. If the graph is undirected, the mode "all" should be specified. Otherwise, for the calculation of Hubness score based on incoming connections select "in" and for the outgoing connections select "out". Also, if all of the connections are desired, specify the "all" mode. Default mode is set to "all".
loops	Logical; whether the loop edges are also counted.
scale	Character string; the method used for scaling/normalizing the results. Options include 'range' (normalization within a 1-100 range), 'z-scale' (standardization using the z-score), and 'none' (no data scaling). The default selection is 'range'. Opting for the 'range' method is suitable when exploring a single network, allowing you to observe the complete spectrum and distribution of node influences. In this case, there is no intention to establish a specific threshold for the outcomes. However, it is possible to identify and present the top hub nodes based on their rankings. Conversely, the 'z-scale' option proves advantageous if the aim is to compare node influences across multiple networks or if there is a desire to establish a threshold (usually z-score > 1.645) for generating a list of the most hub nodes without manual intervention.
verbose	Logical; whether the accomplishment of different stages of the algorithm should be printed (default is FALSE).

Value

A numeric vector with the Hubness scores.

See Also

cent_network.vis

Other integrative ranking functions: comp_manipulate(), exir(), ivi(), ivi.from.indices(), spreading.score()

Examples

```
## Not run:
MyData <- coexpression.data
My_graph <- graph_from_data_frame(MyData)
GraphVertices <- V(My_graph)
Hubness.score <- hubness.score(graph = My_graph, vertices = GraphVertices,
                               directed = FALSE, mode = "all",
                               loops = TRUE, scale = "range")

## End(Not run)
```

h_index *H-index*

Description

This function calculates the H-index of input vertices and works with both directed and undirected networks.

Usage

```
h_index(graph, vertices = V(graph), mode = "all", verbose = FALSE)
```

Arguments

graph	A graph (network) of the igraph class.
vertices	A vector of desired vertices, which could be obtained by the V function.
mode	The mode of H-index depending on the directedness of the graph. If the graph is undirected, the mode "all" should be specified. Otherwise, for the calculation of H-index based on incoming connections select "in" and for the outgoing connections select "out". Also, if all of the connections are desired, specify the "all" mode. Default mode is set to "all".
verbose	Logical; whether the accomplishment of different stages of the algorithm should be printed (default is FALSE).

Value

A vector including the H-index of each vertex inputted.

See Also

ivi, cent_network.vis

Other centrality functions: betweenness(), clusterRank(), collective.influence(), lh_index(), neighborhood.connectivity(), sirir()

Examples

```
## Not run:
MyData <- coexpression.data
My_graph <- graph_from_data_frame(MyData)
GraphVertices <- V(My_graph)
h.index <- h_index(graph = My_graph, vertices = GraphVertices, mode = "all")

## End(Not run)
```

ivi *Integrated Value of Influence (IVI)*

Description

This function calculates the IVI of the desired nodes from a graph. #' A shiny app has also been developed for the calculation of IVI as well as IVI-based network visualization, which is accessible using the 'influential::runShinyApp("IVI")' command. You can also access the shiny app online at https://influential.erc.monash.edu/.

Usage

```
ivi(
  graph,
  vertices = V(graph),
  weights = NULL,
  directed = FALSE,
  mode = "all",
  loops = TRUE,
  d = 3,
  scale = "range",
  ncores = "default",
  verbose = FALSE
)
```

Arguments

graph	A graph (network) of the igraph class.
vertices	A vector of desired vertices, which could be obtained by the V function.
weights	Optional positive weight vector for calculating weighted betweenness centrality of nodes as a requirement for calculation of IVI. If the graph has a weight edge attribute, then this is used by default. Weights are used to calculate weighted shortest paths, so they are interpreted as distances.
directed	Logical scalar, whether to directed graph is analyzed. This argument is ignored for undirected graphs.
mode	The mode of IVI depending on the directedness of the graph. If the graph is undirected, the mode "all" should be specified. Otherwise, for the calculation of IVI based on incoming connections select "in" and for the outgoing connections select "out". Also, if all of the connections are desired, specify the "all" mode. Default mode is set to "all".
loops	Logical; whether the loop edges are also counted.
d	The distance, expressed in number of steps from a given node (default=3). Distance must be > 0. According to Morone & Makse (https://doi.org/10.1038/nature14604), optimal results can be reached at d=3,4, but this depends on the size/"radius" of the network. NOTE: the distance d is not inclusive. This means that nodes at a

distance of 3 from our node-of-interest do not include nodes at distances 1 and 2. Only 3.

`scale` Character string; the method used for scaling/normalizing the results. Options include 'range' (normalization within a 1-100 range), 'z-scale' (standardization using the z-score), and 'none' (no data scaling). The default selection is 'range'. Opting for the 'range' method is suitable when exploring a single network, allowing you to observe the complete spectrum and distribution of node influences. In this case, there is no intention to establish a specific threshold for the outcomes. However, it is possible to identify and present the top influential nodes based on their rankings. Conversely, the 'z-scale' option proves advantageous if the aim is to compare node influences across multiple networks or if there is a desire to establish a threshold (usually z-score > 1.645) for generating a list of the most influential nodes without manual intervention.

`ncores` Integer; the number of cores to be used for parallel processing. If ncores == "default" (default), the number of cores to be used will be the max(number of available cores) - 1. We recommend leaving ncores argument as is (ncores = "default").

`verbose` Logical; whether the accomplishment of different stages of the algorithm should be printed (default is FALSE).

Value

A numeric vector with the IVI values based on the provided centrality measures.

See Also

`cent_network.vis`

Other integrative ranking functions: `comp_manipulate()`, `exir()`, `hubness.score()`, `ivi.from.indices()`, `spreading.score()`

Examples

```
## Not run:
MyData <- coexpression.data
My_graph <- graph_from_data_frame(MyData)
GraphVertices <- V(My_graph)
My.vertices.IVI <- ivi(graph = My_graph, vertices = GraphVertices,
                       weights = NULL, directed = FALSE, mode = "all",
                       loops = TRUE, d = 3, scale = "range")

## End(Not run)
```

`ivi.from.indices` *Integrated Value of Influence (IVI)*

Description

This function calculates the IVI of the desired nodes from previously calculated centrality measures. This function is not dependent to other packages and the required centrality measures, namely degree centrality, ClusterRank, betweenness centrality, Collective Influence, local H-index, and neighborhood connectivity could have been calculated by any means beforehand. A shiny app has also been developed for the calculation of IVI as well as IVI-based network visualization, which is accessible using the 'influential::runShinyApp("IVI")' command. You can also access the shiny app online at https://influential.erc.monash.edu/.

Usage

```
ivi.from.indices(
  DC,
  CR,
  LH_index,
  NC,
  BC,
  CI,
  scale = "range",
  verbose = FALSE
)
```

Arguments

DC	A vector containing the values of degree centrality of the desired vertices.
CR	A vector containing the values of ClusterRank of the desired vertices.
LH_index	A vector containing the values of local H-index of the desired vertices.
NC	A vector containing the values of neighborhood connectivity of the desired vertices.
BC	A vector containing the values of betweenness centrality of the desired vertices.
CI	A vector containing the values of Collective Influence of the desired vertices.
scale	Character string; the method used for scaling/normalizing the results. Options include 'range' (normalization within a 1-100 range), 'z-scale' (standardization using the z-score), and 'none' (no data scaling). The default selection is 'range'. Opting for the 'range' method is suitable when exploring a single network, allowing you to observe the complete spectrum and distribution of node influences. In this case, there is no intention to establish a specific threshold for the outcomes. However, it is possible to identify and present the top influential nodes based on their rankings. Conversely, the 'z-scale' option proves advantageous if the aim is to compare node influences across multiple networks or if there is a desire to establish a threshold (usually z-score > 1.645) for generating a list of the most influential nodes without manual intervention.

verbose — Logical; whether the accomplishment of different stages of the algorithm should be printed (default is FALSE).

Value

A numeric vector with the IVI values based on the provided centrality measures.

See Also

cent_network.vis

Other integrative ranking functions: comp_manipulate(), exir(), hubness.score(), ivi(), spreading.score()

Examples

```
## Not run:
MyData <- centrality.measures
My.vertices.IVI <- ivi.from.indices(DC = centrality.measures$DC,
                                    CR = centrality.measures$CR,
                                    NC = centrality.measures$NC,
                                    LH_index = centrality.measures$LH_index,
                                    BC = centrality.measures$BC,
                                    CI = centrality.measures$CI)

## End(Not run)
```

---

lh_index *local H-index (LH-index)*

Description

This function calculates the local H-index of input vertices and works with both directed and undirected networks.

Usage

```
lh_index(
  graph,
  vertices = V(graph),
  mode = "all",
  ncores = "default",
  verbose = FALSE
)
```

Arguments

graph	A graph (network) of the igraph class.
vertices	A vector of desired vertices, which could be obtained by the V function.
mode	The mode of local H-index depending on the directedness of the graph. If the graph is undirected, the mode "all" should be specified. Otherwise, for the calculation of local H-index based on incoming connections select "in" and for the outgoing connections select "out". Also, if all of the connections are desired, specify the "all" mode. Default mode is set to "all".
ncores	Integer; the number of cores to be used for parallel processing. If ncores == "default" (default), the number of cores to be used will be the max(number of available cores) - 1. We recommend leaving ncores argument as is (ncores = "default").
verbose	Logical; whether the accomplishment of different stages of the algorithm should be printed (default is FALSE).

Value

A vector including the local H-index of each vertex inputted.

See Also

ivi, cent_network.vis

Other centrality functions: betweenness(), clusterRank(), collective.influence(), h_index(), neighborhood.connectivity(), sirir()

Examples

```
## Not run:
MyData <- coexpression.data
My_graph <- graph_from_data_frame(MyData)
GraphVertices <- V(My_graph)
lh.index <- lh_index(graph = My_graph, vertices = GraphVertices, mode = "all", ncores = 1)

## End(Not run)
```

---

neighborhood.connectivity

Neighborhood connectivity

Description

This function calculates the neighborhood connectivity of input vertices and works with both directed and undirected networks.

Usage

```
neighborhood.connectivity(
  graph,
  vertices = V(graph),
  mode = "all",
  verbose = FALSE
)
```

Arguments

graph	A graph (network) of the igraph class.
vertices	A vector of desired vertices, which could be obtained by the V function.
mode	The mode of neighborhood connectivity depending on the directedness of the graph. If the graph is undirected, the mode "all" should be specified. Otherwise, for the calculation of neighborhood connectivity based on incoming connections select "in" and for the outgoing connections select "out". Also, if all of the connections are desired, specify the "all" mode. Default mode is set to "all".
verbose	Logical; whether the accomplishment of different stages of the algorithm should be printed (default is FALSE).

Value

A vector including the neighborhood connectivity score of each vertex inputted.

See Also

ivi, cent_network.vis

Other centrality functions: betweenness(), clusterRank(), collective.influence(), h_index(), lh_index(), sirir()

Examples

```
## Not run:
MyData <- coexpression.data
My_graph <- graph_from_data_frame(MyData)
GraphVertices <- V(My_graph)
neighrhood.co <- neighborhood.connectivity(graph = My_graph,
                                           vertices = GraphVertices,
                                           mode = "all")

## End(Not run)
```

runShinyApp *Run shiny app*

Description

Run shiny apps included in the influential R package. Also, a web-based Influential Software Package with a convenient user-interface (UI) has been developed for the comfort of all users including those without a coding background.

Usage

```
runShinyApp(shinyApp)
```

Arguments

shinyApp	The name of the shiny app you want to run. You can get the exact name of the available shiny apps via the following command. *list.files(system.file("ShinyApps", package = "influential"))*. Please also note this function is case-sensitive.

Value

A shiny app.

Examples

```
## Not run:
runShinyApp(shinyApp = "IVI")

## End(Not run)
```

sif2igraph *SIF to igraph*

Description

This function imports and converts a SIF file from your local hard drive, cloud space, or internet into a graph with an igraph class, which can then be used for the identification of most influential nodes via the ivi function, for instance.

Usage

```
sif2igraph(Path, directed = FALSE)
```

Arguments

Path	A string or character vector indicating the path to the desired SIF file. The SIF file could be on your local hard drive, cloud space, or on the internet.
directed	Logical scalar, whether or not to create a directed graph.

Value

An igraph graph object.

Examples

```
## Not run:
MyGraph <- sif2igraph(Path = "/Users/User1/Desktop/mygraph.sif", directed=FALSE)

## End(Not run)
```

---

sirir *SIR-based Influence Ranking*

Description

This function is achieved by the integration susceptible-infected-recovered (SIR) model with the leave-one-out cross validation technique and ranks network nodes based on their true universal influence. One of the applications of this function is the assessment of performance of a novel algorithm in identification of network influential nodes by considering the SIRIR ranks as the ground truth (gold standard).

Usage

```
sirir(
  graph,
  vertices = V(graph),
  beta = 0.5,
  gamma = 1,
  no.sim = 100,
  ncores = "default",
  seed = 1234,
  loop_verbose = TRUE,
  node_verbose = FALSE
)
```

Arguments

graph	A graph (network) of the igraph class.
vertices	A vector of desired vertices, which could be obtained by the V function.

beta	Non-negative scalar. The rate of infection of an individual that is susceptible and has a single infected neighbor. The infection rate of a susceptible individual with n infected neighbors is n times beta. Formally this is the rate parameter of an exponential distribution.
gamma	Positive scalar. The rate of recovery of an infected individual. Formally, this is the rate parameter of an exponential distribution.
no.sim	Integer scalar, the number of simulation runs to perform SIR model on the original network as well as perturbed networks generated by leave-one-out technique. You may choose a different no.sim based on the available memory on your system.
ncores	Integer; the number of cores to be used for parallel processing. If ncores == "default" (default), the number of cores to be used will be the max(number of available cores) - 1. We recommend leaving ncores argument as is (ncores = "default").
seed	A single value, interpreted as an integer to be used for random number generation.
loop_verbose	Logical; whether the accomplishment of the evaluation of network nodes in each loop should be printed (default is TRUE).
node_verbose	Logical; whether the process of Parallel Socket Cluster creation should be printed (default is FALSE).

Value

A two-column dataframe; a column containing the difference values of the original and perturbed networks and a column containing node influence rankings

See Also

cent_network.vis, and sir for a complete description on SIR model

Other centrality functions: betweenness(), clusterRank(), collective.influence(), h_index(), lh_index(), neighborhood.connectivity()

Examples

```
## Not run:
set.seed(1234)
My_graph <- igraph::sample_gnp(n=50, p=0.05)
GraphVertices <- V(My_graph)
Influence.Ranks <- sirir(graph = My_graph, vertices = GraphVertices,
                     beta = 0.5, gamma = 1, ncores = "default", no.sim = 10, seed = 1234)

## End(Not run)
```

spreading.score *Spreading score*

Description

This function calculates the Spreading score of the desired nodes from a graph. Spreading score reflects the spreading potential of each node within a network and is one of the major components of the IVI.

Usage

```
spreading.score(
  graph,
  vertices = V(graph),
  weights = NULL,
  directed = FALSE,
  mode = "all",
  loops = TRUE,
  d = 3,
  scale = "range",
  verbose = FALSE
)
```

Arguments

graph	A graph (network) of the igraph class.
vertices	A vector of desired vertices, which could be obtained by the V function.
weights	Optional positive weight vector for calculating weighted betweenness centrality of nodes as a requirement for calculation of spreading score. If the graph has a weight edge attribute, then this is used by default. Weights are used to calculate weighted shortest paths, so they are interpreted as distances.
directed	Logical scalar, whether to directed graph is analyzed. This argument is ignored for undirected graphs.
mode	The mode of Spreading score depending on the directedness of the graph. If the graph is undirected, the mode "all" should be specified. Otherwise, for the calculation of Spreading score based on incoming connections select "in" and for the outgoing connections select "out". Also, if all of the connections are desired, specify the "all" mode. Default mode is set to "all".
loops	Logical; whether the loop edges are also counted.
d	The distance, expressed in number of steps from a given node (default=3). Distance must be > 0. According to Morone & Makse (https://doi.org/10.1038/nature14604), optimal results can be reached at d=3,4, but this depends on the size/"radius" of the network. NOTE: the distance d is not inclusive. This means that nodes at a distance of 3 from our node-of-interest do not include nodes at distances 1 and 2. Only 3.

scale	Character string; the method used for scaling/normalizing the results. Options include 'range' (normalization within a 1-100 range), 'z-scale' (standardization using the z-score), and 'none' (no data scaling). The default selection is 'range'. Opting for the 'range' method is suitable when exploring a single network, allowing you to observe the complete spectrum and distribution of node influences. In this case, there is no intention to establish a specific threshold for the outcomes. However, it is possible to identify and present the top spreading nodes based on their rankings. Conversely, the 'z-scale' option proves advantageous if the aim is to compare node influences across multiple networks or if there is a desire to establish a threshold (usually z-score > 1.645) for generating a list of the most spreading nodes without manual intervention.
verbose	Logical; whether the accomplishment of different stages of the algorithm should be printed (default is FALSE).

Value

A numeric vector with Spreading scores.

See Also

cent_network.vis

Other integrative ranking functions: comp_manipulate(), exir(), hubness.score(), ivi(), ivi.from.indices()

Examples

```
## Not run:
MyData <- coexpression.data
My_graph <- graph_from_data_frame(MyData)
GraphVertices <- V(My_graph)
Spreading.score <- spreading.score(graph = My_graph, vertices = GraphVertices,
                                   weights = NULL, directed = FALSE, mode = "all",
                                   loops = TRUE, d = 3, scale = "range")

## End(Not run)
```

Index

∗ **IVI**
ivi, 32
∗ **SIF.to.igraph**
sif2igraph, 38
∗ **association_assessment**
cond.prob.analysis, 14
double.cent.assess, 16
double.cent.assess.noRegression, 17
∗ **betweenness_centrality**
betweenness, 3
∗ **cent_network.vis**
cent_network.vis, 5
∗ **centrality association assessment functions**
cond.prob.analysis, 14
double.cent.assess, 16
double.cent.assess.noRegression, 17
∗ **centrality functions**
betweenness, 3
clusterRank, 8
collective.influence, 10
h_index, 31
lh_index, 35
neighborhood.connectivity, 36
sirir, 39
∗ **clusterRank**
clusterRank, 8
∗ **collective.influence**
collective.influence, 10
∗ **comp_manipulate**
comp_manipulate, 12
∗ **conditional_probability**
cond.prob.analysis, 14
∗ **datasets**
centrality.measures, 4
coexpression.adjacency, 9
coexpression.data, 10
∗ **dependence_assessment**
double.cent.assess, 16
double.cent.assess.noRegression, 17
∗ **diff_data.assembly**
diff_data.assembly, 15
∗ **exir.vis**
exir.vis, 24
∗ **exir**
exir, 19
∗ **fcor**
fcor, 27
∗ **h_index**
h_index, 31
∗ **hubness.score**
hubness.score, 29
∗ **integrated_value_of_influence**
ivi, 32
∗ **integrative ranking functions**
comp_manipulate, 12
exir, 19
hubness.score, 29
ivi, 32
ivi.from.indices, 34
spreading.score, 41
∗ **ivi.from.indices**
ivi.from.indices, 34
∗ **lh_index**
lh_index, 35
∗ **neighborhood_connectivity**
neighborhood.connectivity, 36
∗ **network_reconstruction functions**
sif2igraph, 38
∗ **runShinyApp**
runShinyApp, 38
∗ **sirir**
sirir, 39
∗ **spreading.score**
spreading.score, 41
∗ **visualization functions**

cent_network.vis, 5
exir.vis, 24

ad.test, *17, 18*

BC *(*betweenness*)*, 3
betweenness, *3*, 3, *9*, *11*, *31*, *36, 37*, *40*

cent_network.vis, *3*, 5, *9*, *11*, *26*, *30, 31*, *33*, *35–37*, *40*, *42*
centrality.measures, 4
CI *(*collective.influence*)*, 10
clusterRank, *3*, 8, *11*, *31*, *36, 37*, *40*
coexpression.adjacency, 9
coexpression.data, 10
collective.influence, *3*, *9*, 10, *31*, *36, 37*, *40*
comp_manipulate, 12, *23*, *30*, *33*, *35*, *42*
cond.prob.analysis, 14, *17, 18*
CPA *(*cond.prob.analysis*)*, 14
CR *(*clusterRank*)*, 8

DCA *(*double.cent.assess*)*, 16
DCANR *(*double.cent.assess.noRegression*)*, 17
DDA *(*diff_data.assembly*)*, 15
diff_data.assembly, 15, *20*, *23*
double.cent.assess, *14*, 16, *18*
double.cent.assess.noRegression, *14*, *17*, 17

ExIR *(*exir*)*, 19
exir, *13*, *15*, 19, *26*, *30*, *33*, *35*, *42*
exir.vis, *8*, *23*, 24

fcor, 27

gam, *17*
graph_from_data_frame, *28*

h.index *(*h_index*)*, 31
h_index, *3*, *9*, *11*, 31, *36, 37*, *40*
hoeffd, *17, 18*
hubness.score, *13*, *23*, 29, *33*, *35*, *42*

importance_pvalues, *23*
IVI *(*ivi*)*, 32
ivi, *3*, *6*, *8, 9*, *11*, *13*, *23*, *30, 31*, 32, *35–37*, *42*
IVI.FI *(*ivi.from.indices*)*, 34
ivi.from.indices, *13*, *23*, *30*, *33*, 34, *42*

layout_, *6*
lh.index *(*lh_index*)*, 35
lh_index, *3*, *9*, *11*, *31*, 35, *37*, *40*
lm, *17*

NC *(*neighborhood.connectivity*)*, 36
neighborhood.connectivity, *3*, *9*, *11*, *31*, *36*, 36, *40*
NNS.dep, *17, 18*

p.adjust, *28*
prcomp, *23*

ranger, *23*
runShinyApp, 38

sif2igraph, 38
sir, *13*, *40*
SIRIR *(*sirir*)*, 39
sirir, *3*, *9*, *11*, *13*, *31*, *36, 37*, 39
spreading.score, *13*, *23*, *30*, *33*, *35*, 41