Package 'influential'

June 26, 2020

Type Package

Title Identification and Classification of the Most Influential Nodes

Version 1.1.2

Author Adrian (Abbas) Salavaty [aut, cre], Mirana Ramialison [ths], Peter D. Currie [ths]

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

Description Contains functions for the classification and ranking of top candidate features, reconstruction of networks from
adjacency matrices and data frames, analysis of the topology of the network
and calculation of centrality measures, and identification of the most
influential nodes. Also, a function is provided for running SIRIR model, which
is the combination of leave-one-out cross validation technique and the conventional SIR model, on a network to unsupervisedly rank the true influence of vertices. Additionally, some functions have been provided for the assessment
of dependence and correlation of two network centrality measures as well as
the conditional probability of deviation from their corresponding means in opposite direction.
Fred Viole and David Nawrocki (2013, ISBN:1490523995).
Csardi G, Nepusz T (2006). ``The igraph software package for complex network research.'' InterJournal, Complex Systems, 1695.
Adopted algorithms and sources are referenced in function document.

Imports igraph, ranger, coop, reshape2

Suggests Hmisc (>= 4.3-0), mgcv (>= 1.8-31), nortest (>= 1.0-4), NNS
(>= 0.4.7.1), parallel, knitr, rmarkdown

Depends R (>= 2.10)

URL http://github.com/asalavaty/influential

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

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.1.0

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2020-06-26 14:50:06 UTC

R topics documented:

`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,
  nobigint = TRUE,
  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.
`nobigint`	Logical scalar, whether to use big integers during the calculation. This is only required for lattice-like graphs that have very many shortest paths between a pair of vertices. If TRUE (the default), then big integers are not used.
`normalized`	Logical scalar, whether to normalize the betweenness scores. If TRUE, then the results are normalized.

Value

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

See Also

`betweenness` for a complete description on this function

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

Examples

```
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)
```

`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

https://www.ncbi.nlm.nih.gov/pubmed/31211495/

`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)
```

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.

Value

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

See Also

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

Examples

```
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)
```

`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

https://www.ncbi.nlm.nih.gov/pubmed/31211495/

`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

https://www.ncbi.nlm.nih.gov/pubmed/31211495/

`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 reduced degree of all nodes at a distance d from the node.

Usage

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

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.

Value

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

See Also

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

Examples

```
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)
```

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.noRegression()`, `double.cent.assess()`

Examples

```
MyData <- centrality.measures
My.conditional.prob <- cond.prob.analysis(data = MyData,
                                          nodes.colname = rownames(MyData),
                                          Desired.colname = "BC",
                                          Condition.colname = "NC")
```

`degree` *Degree of the vertices*

Description

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

Usage

```
degree(
  graph,
  v = V(graph),
  mode = c("all", "out", "in", "total"),
  loops = TRUE,
  normalized = FALSE
)
```

Arguments

`graph`	The graph to analyze (an igraph graph).
`v`	The ids of vertices of which the degree will be calculated.
`mode`	Character string, “out” for out-degree, “in” for in-degree or “total” for the sum of the two. For undirected graphs this argument is ignored. “all” is a synonym of “total”.
`loops`	Logical; whether the loop edges are also counted. 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 degree. If TRUE then the result is divided by n-1, where n is the number of vertices in the graph.

Value

A numeric vector of the same length as argument v.

See Also

`degree` for a complete description on this function

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

Examples

```
MyData <- coexpression.data
My_graph <- graph_from_data_frame(MyData)
GraphVertices <- V(My_graph)
My_graph_degree <- degree(My_graph, v = GraphVertices, normalized = FALSE)
```

`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.

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.

Usage

```
exir(
  Desired_list = NULL,
  Diff_data,
  Diff_value,
  Regr_value = NULL,
  Sig_value,
  Exptl_data,
  Condition_colname,
  Normalize = FALSE,
  r = 0,
  alpha = 0.05,
  num_trees = 10000,
  num_permutations = 100,
  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). 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`	A numeric 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) A numeric 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`	A numeric 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`	A dataframe containing all of the experimental including a column for specifying the conditions. The features/variables of the dataframe should be as the columns and the samples should come in the rows. The condition column should be of the character class. For example, if the study includes several replicates of cancer and normal samples, the condition column should include "cancer" and "normal" as the conditions of different samples. Also, the prior normalization of the experimental data is highly recommended. Otherwise, the user may set the Normalize argument to TRUE for a simple log2 transformation of the data. The experimental data could come from a variety sources such as transcriptomics and proteomics assays.
`Condition_colname`	A string or character vector specifying the name of the condition column of the Exptl_data dataframe.
`Normalize`	Logical; whether the experimental data should be normalized or not (default is FALSE). If TRUE, the experimental data will be log2 transformed.
`r`	The threshold of Pearson correlation coefficient for the selection of correlated features (default is 0).
`alpha`	The threshold of the statistical significance (p-value) used throughout the entir model (default is 0.05)
`num_trees`	Number of trees to be used for the random forest classification (supervised machine learning) Default is set to 10000.
`num_permutations`	Number of permutations to be used for computation of the statistical significances (p-values) of the importance scores resulted from random forest classification (default is 100).

`seed`	The seed to be used for all of the random processes throughout the model (default is 1234).
`verbose`	Logical; whether the accomplishment of different stages of the model should be printed (default is TRUE).

Value

A list of one to four tables including:

- Driver table: Top candidate drivers

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

- nonDE-mediators 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

`diff_data.assembly`, `ivi`, `pcor`, `prcomp`, `ranger`, `importance_pvalues`

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

`graph_from_adjacency_matrix`

Creating igraph graphs from adjacency matrices

Description

This function and all of its descriptions have been obtained from the igraph package. For a complete description if the function and its arguments try this: ?igraph::graph_from_adjacency_matrix

Usage

```
graph_from_adjacency_matrix(
  adjmatrix,
  mode = c("directed", "undirected", "max", "min", "upper", "lower", "plus"),
  weighted = NULL,
  diag = TRUE,
  add.colnames = NULL,
  add.rownames = NA
)
```

Arguments

`adjmatrix`	A square adjacency matrix. From igraph version 0.5.1 this can be a sparse matrix created with the Matrix package.
`mode`	Character scalar, specifies how igraph should interpret the supplied matrix. See also the weighted argument, the interpretation depends on that too. Possible values are: directed, undirected, upper, lower, max, min, plus.
`weighted`	This argument specifies whether to create a weighted graph from an adjacency matrix. If it is NULL then an unweighted graph is created and the elements of the adjacency matrix gives the number of edges between the vertices. If it is a character constant then for every non-zero matrix entry an edge is created and the value of the entry is added as an edge attribute named by the weighted argument. If it is TRUE then a weighted graph is created and the name of the edge attribute will be weight.
`diag`	Logical scalar, whether to include the diagonal of the matrix in the calculation. If this is FALSE then the diagonal is zerod out first.
`add.colnames`	Character scalar, whether to add the column names as vertex attributes. If it is 'NULL' (the default) then, if present, column names are added as vertex attribute 'name'. If 'NA' then they will not be added. If a character constant, then it gives the name of the vertex attribute to add.
`add.rownames`	Character scalar, whether to add the row names as vertex attributes. Possible values the same as the previous argument. By default row names are not added. If 'add.rownames' and 'add.colnames' specify the same vertex attribute, then the former is ignored.

Value

An igraph graph object.

See Also

`graph_from_adjacency_matrix` for a complete description on this function

Other network_reconstruction functions: `graph_from_data_frame()`

Examples

```
MyData <- coexpression.adjacency
My_graph <- graph_from_adjacency_matrix(MyData)
```

`graph_from_data_frame` *Creating igraph graphs from data frames*

Description

This function and all of its descriptions have been obtained from the igraph package. For a complete description if the function and its arguments try this: ?igraph::graph_from_data_frame

Usage

```
graph_from_data_frame(d, directed = TRUE, vertices = NULL)
```

Arguments

`d`	A data frame containing a symbolic edge list in the first two columns. Additional columns are considered as edge attributes. Since version 0.7 this argument is coerced to a data frame with as.data.frame.
`directed`	Logical scalar, whether or not to create a directed graph.
`vertices`	A data frame with vertex metadata, or NULL. Since version 0.7 of igraph this argument is coerced to a data frame with as.data.frame, if not NULL.

Value

An igraph graph object.

See Also

`graph_from_adjacency_matrix` for a complete description on this function

Other network_reconstruction functions: `graph_from_adjacency_matrix()`

Examples

```
MyData <- coexpression.data
My_graph <- graph_from_data_frame(d=MyData)
```

`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,
  scaled = TRUE
)
```

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.
`scaled`	Logical; whether the end result should be 1-100 range normalized or not (default is TRUE).

Value

A numeric vector with the Hubness scores.

See Also

`spreading.score`

Other integrative ranking functions: `exir()`, `ivi.from.indices()`, `ivi()`, `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, scaled = TRUE)

## 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")
```

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".

Value

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

See Also

`lh_index`

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

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.

Usage

```
ivi(
  graph,
  vertices = V(graph),
  weights = NULL,
  directed = FALSE,
  mode = "all",
  loops = TRUE,
  d = 3,
  scaled = TRUE
)
```

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.
`scaled`	Logical; whether the end result should be 1-100 range normalized or not (default is TRUE).

Value

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

See Also

`ivi.from.indices`, `exir`

Other integrative ranking functions: `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, scaled = TRUE)

## 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.

Usage

```
ivi.from.indices(DC, CR, LH_index, NC, BC, CI, scaled = TRUE)
```

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.
`scaled`	Logical; whether the end result should be 1-100 range normalized or not (default is TRUE).

Value

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

See Also

`ivi`, `exir`

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

Examples

```
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)
```

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")
```

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".

Value

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

See Also

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

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")

## 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")
```

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".

Value

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

See Also

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

Examples

```
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")
```

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.

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.

See Also

`graph_from_data_frame`

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.

Usage

```
sirir(
  graph,
  vertices = V(graph),
  beta = 0.5,
  gamma = 1,
  no.sim = igraph::vcount(graph) * 100,
  seed = 1234
)
```

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

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

`sir` for a complete description on SIR model.

Examples

```
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, no.sim = 10, seed = 1234)
```

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,
  scaled = TRUE
)
```

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 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.
`scaled`	Logical; whether the end result should be 1-100 range normalized or not (default is TRUE).

Value

A numeric vector with Spreading scores.

See Also

`hubness.score`

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

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, scaled = TRUE)

## End(Not run)
```

`V` *Vertices of an igraph graph*

Description

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

Usage

```
V(graph)
```

Arguments

`graph`	The graph (an igraph graph)

Value

A vertex sequence containing all vertices, in the order of their numeric vertex ids.

See Also

`V` for a complete description on this function

Examples

```
MyData <- coexpression.data
My_graph <- graph_from_data_frame(MyData)
My_graph_vertices <- V(My_graph)
```

Index