Package ‘PLEXI’

August 9, 2023

Type Package

Title Multiplex Network Analysis

Version 1.0.0

Description
Interactions between different biological entities are crucial for the function of biological systems. In such networks, nodes represent biological elements, such as genes, proteins and microbes, and their interactions can be defined by edges, which can be either binary or weighted. The dysregulation of these networks can be associated with different clinical conditions such as diseases and response to treatments.
However, such variations often occur locally and do not concern the whole network.
To capture local variations of such networks, we propose multiplex network differential analysis (MNDA).
MNDA allows to quantify the variations in the local neighborhood of each node (e.g. gene) between the two given clinical states, and to test for statistical significance of such variation.
Yousefi et al. (2023) <doi:10.1101/2023.01.22.525058>.

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.2.3

Imports assertthat, igraph, keras, ggraph, ggplot2, aggregation,
stats, graphics

NeedsCompilation no

Author Behnam Yousefi [aut, cre, cph],
Farzaneh Firoozbakht [aut]

Maintainer Behnam Yousefi <yousefi.bme@gmail.com>

Repository CRAN

Date/Publication 2023-08-09 15:50:03 UTC

R topics documented:

`as_igraph` *Convert plexi graph data to igraph*

Description

Convert plexi graph data to igraph

Usage

```
as_igraph(plexi.graph, edge.threshold = 0)
```

Arguments

`plexi.graph`	plexi graph data
`edge.threshold`	numeric

Value

igraph object

Examples

```
data = example_data()
graph = as_igraph(plexi.graph = data[["plexi_graph_example"]])
```

as_plexi_graph — *Convert adjacency matrix to plexi graph data*

Description

Convert adjacency matrix to plexi graph data

Usage

```
as_plexi_graph(adj.list, outcome = NULL)
```

Arguments

`adj.list`	list of adjacency matrices with matching nodes
`outcome`	graph outcomes or graph labels. If NULL, `outcome = 1:N_graphs`.

Value

plexi.graph data

Examples

```
data = example_data()
adj.list = list(data[["adj_mat_example"]], data[["adj_mat_example"]])
graph.data = as_plexi_graph(adj.list)
```

distance — *Function to calculate distance between two vectors*

Description

Function to calculate distance between two vectors

Usage

```
distance(x, y, method = "cosine")
```

Arguments

`x`	numeric vector
`y`	numeric vector
`method`	distance calculation method: cosine (default), dot.prod, euclidian, manhattan, chebyshev, coassociation

Value

the distance value

Examples

```
x = c(1,2,3)
y = c(6,4,6)
distance(x,y)
```

---

ednn *Encoder decoder neural network (EDNN) function*

Description

Encoder decoder neural network (EDNN) function

Usage

```
ednn(
  x,
  y,
  x.test,
  embedding.size = 2,
  epochs = 10,
  batch.size = 5,
  l2reg = 0,
  demo = TRUE,
  verbose = FALSE
)
```

Arguments

x	concatenated adjacency matrices for different layers containing the nodes in training phase
y	concatenated random walk probability matrices for different layers containing the nodes in training phase
x.test	concatenated adjacency matrices for different layers containing the nodes in test phase. Can be = *X* for transductive inference.
embedding.size	the dimension of embedding space, equal to the number of the bottleneck hidden nodes.
epochs	maximum number of pocks. An early stopping callback with a patience of 5 has been set inside the function (default = 10).
batch.size	batch size for learning (default = 5).
l2reg	the coefficient of L2 regularization for the input layer (default = 0).
demo	a boolean vector to indicate this is a demo example or not
verbose	if *TRUE* a progress bar is shown.

Value

The embedding space for x.test.

Examples

```
myNet = network_gen(n.nodes = 50)
graphData = myNet[["data_graph"]]
edge.list = graphData[,1:2]
edge.weight = graphData[,3:4]
XY = ednn_io_prepare(edge.list, edge.weight)
X = XY[["X"]]
Y = XY[["Y"]]
embeddingSpace = ednn(x = X, y = Y, x.test = X)
```

`ednn_io_prepare` *Preparing the input and output of the EDNN for a multiplex graph*

Description

Preparing the input and output of the EDNN for a multiplex graph

Usage

```
ednn_io_prepare(
  edge.list,
  edge.weight,
  outcome = NULL,
  indv.index = NULL,
  edge.threshold = 0,
  walk.rep = 10,
  n.steps = 5,
  random.walk = TRUE,
  verbose = TRUE
)
```

Arguments

`edge.list`	edge list as a dataframe with two columns.
`edge.weight`	edge weights as a dataframe. Each column corresponds to a graph. By default, the `colnames` are considered as outcomes unless indicated in `outcome` argument.
`outcome`	clinical outcomes for each graph. If not mentioned, the `colnames(edge.weight)` are considered by default.
`indv.index`	the index of individual networks.
`edge.threshold`	numeric value to set edge weights below the threshold to zero (default: 0). the greater edge weights do not change.

`walk.rep`	number of repeats for the random walk (default: 100).
`n.steps`	number of the random walk steps (default: 5).
`random.walk`	boolean value to enable the random walk algorithm (default: TRUE).
`verbose`	if *TRUE* a progress bar is shown.

Value

the input and output required to train the EDNN

Examples

```
myNet = network_gen(n.nodes = 50)
graphData = myNet[["data_graph"]]
edge.list = graphData[,1:2]
edge.weight = graphData[,3:4]
XY = ednn_io_prepare(edge.list, edge.weight)
X = XY[["X"]]
Y = XY[["Y"]]
```

`example_data` *Example Data*

Description

Example Data

Usage

```
example_data()
```

Value

example data as a list: "adj_mat_example", "igraph_example", "plexi_graph_example"

Examples

```
data = example_data()
```

network_gen — *Multiplex Network Generation*

Description

Multiplex Network Generation

Usage

```
network_gen(n.nodes = 100, n.var.nodes = 5, n.var.nei = 90, noise.sd = 0.1)
```

Arguments

`n.nodes`	number of nodes in the graph
`n.var.nodes`	number of nodes whose neighborhood should change from layer 1 to 2
`n.var.nei`	number of neighbors that should be changing from layer 1 to 2
`noise.sd`	the standard deviation of the noise added to the edge weights

Details

In this script we generate random pairs of gene co-expression networks, which are different only in a few (pre-set) number of nodes.

Value

No return value, called to plot subgraphs

Examples

```
myNet = network_gen(n.nodes = 100)
graphData = myNet[["data_graph"]]
varNodes = myNet[["var_node_set"]]
```

plexi_distance_test1_isn — *Test the extremeness of embedding distances of local neighbors.*

Description

Test the extremeness of embedding distances of local neighbors.

Usage

```
plexi_distance_test1_isn(distance, p.adjust.method = "none")
```

Arguments

distance a distance list obtained by the `plexi_node_distance()` function.

p.adjust.method
method for adjusting p-value (including methods on `p.adjust.methods`). If set to "none" (default), no adjustment will be performed.

Details

The adjusted p-values for each node is calculated based on their distance.

Value

The adjusted pvalues for each node.

Examples

```
ISN1 = network_gen(n.nodes = 50, n.var.nodes = 5, n.var.nei = 40, noise.sd = .01)
ISN2 = network_gen(n.nodes = 50, n.var.nodes = 5, n.var.nei = 40, noise.sd = .01)
graph_data = cbind(ISN1[["data_graph"]], ISN1[["data_graph"]][,3:4])
embeddingSpaceList = plexi_embedding(graph.data=graph_data, outcome=c(1,2,1,2),
indv.index=c(1,1,2,2), train.rep=2, random.walk=FALSE)
Dist = plexi_node_distance(embeddingSpaceList)
Result = plexi_distance_test1_isn(Dist)
```

---

plexi_distance_test_isn

Test the embedding distances of local neighbors change between the two conditions for ISNs.

Description

Test the embedding distances of local neighbors change between the two conditions for ISNs.

Usage

```
plexi_distance_test_isn(
  distance,
  y,
  stat.test = "wilcox.test",
  p.adjust.method = "none"
)
```

Arguments

`distance`	a distance list obtained by the `plexi_node_distance()` function.
`y`	vector with the length equal to the number of individuals.
`stat.test`	statistical test used to detect the nodes `c("t.test","wilcox.test")` (default: wilcox.test)
`p.adjust.method`	method for adjusting p-value (including methods on `p.adjust.methods`). If set to "none" (default), no adjustment will be performed.

Details

The adjusted p-values for each node is calculated based on their distance variation between the two conditions.

Value

The adjusted pvalues for each node.

Examples

```
ISN1 = network_gen(n.nodes = 50, n.var.nodes = 5, n.var.nei = 40, noise.sd = .01)
ISN2 = network_gen(n.nodes = 50, n.var.nodes = 5, n.var.nei = 40, noise.sd = .01)
graph_data = cbind(ISN1[["data_graph"]], ISN1[["data_graph"]][,3:4])
embeddingSpaceList = plexi_embedding(graph.data=graph_data, outcome=c(1,2,1,2),
indv.index=c(1,1,2,2), train.rep=2, random.walk=FALSE)
Dist = plexi_node_distance(embeddingSpaceList)
Result = plexi_distance_test_isn(Dist, y = c(1,2))
```

`plexi_embedding` *Calculate the embedding space for a multiplex network*

Description

Calculate the embedding space for a multiplex network

Usage

```
plexi_embedding(
  graph.data,
  outcome,
  indv.index = NULL,
  edge.threshold = 0,
  train.rep = 50,
  embedding.size = 2,
  epochs = 10,
  batch.size = 5,
```

```
  l2reg = 0,
  walk.rep = 100,
  n.steps = 5,
  random.walk = TRUE,
  demo = TRUE,
  verbose = FALSE
)
```

Arguments

`graph.data`	dataframe of the graph data containing edge list and edge weights. column 1 and 2 consisting of the edge list (undirected). column 3 and 4 consisting the edge weights corresponding to each graph, respectively.
`outcome`	a vector of outcomes for each network.
`indv.index`	the index of individual networks.
`edge.threshold`	numeric value to set edge weights below the threshold to zero (default: 0). the greater edge weights do not change.
`train.rep`	numeric value to set the number of EDNN training repeats (default: 50).
`embedding.size`	the dimension of embedding space, equal to the number of the bottleneck hidden nodes (default: 5).
`epochs`	maximum number of pocks. An early stopping callback with a patience of 5 has been set inside the function (default = 10).
`batch.size`	batch size for learning (default = 5).
`l2reg`	the coefficient of L2 regularization for the input layer (default = 0).
`walk.rep`	number of repeats for the random walk (default: 100).
`n.steps`	number of the random walk steps (default: 5).
`random.walk`	boolean value to enable the random walk algorithm (default: TRUE).
`demo`	a boolean vector to indicate this is a demo example or not
`verbose`	if *TRUE* a progress bar is shown.

Value

a list of embedding spaces for each node.

Examples

```
myNet = network_gen(n.nodes = 50, n.var.nodes = 5, n.var.nei = 40, noise.sd = .01)
graph_data = myNet[["data_graph"]]
embeddingSpaceList = plexi_embedding(graph.data=graph_data, outcome=c(1,2),
train.rep=2, random.walk=FALSE)
```

plexi_embedding_2layer — *Calculate the embedding space for a two layer multiplex network*

Description

Calculate the embedding space for a two layer multiplex network

Usage

```
plexi_embedding_2layer(
  graph.data,
  edge.threshold = 0,
  train.rep = 50,
  embedding.size = 2,
  epochs = 10,
  batch.size = 5,
  l2reg = 0,
  walk.rep = 100,
  n.steps = 5,
  random.walk = TRUE,
  null.perm = TRUE,
  demo = TRUE,
  verbose = FALSE
)
```

Arguments

`graph.data`	dataframe of the graph data containing edge list and edge weights. column 1 and 2 consisting of the edge list (undirected). column 3 and 4 consisting the edge weights corresponding to each graph, respectively.
`edge.threshold`	numeric value to set edge weights below the threshold to zero (default: 0). the greater edge weights do not change.
`train.rep`	numeric value to set the number of EDNN training repeats (default: 50).
`embedding.size`	the dimension of embedding space, equal to the number of the bottleneck hidden nodes (default: 5).
`epochs`	maximum number of pocks. An early stopping callback with a patience of 5 has been set inside the function (default = 10).
`batch.size`	batch size for learning (default = 5).
`l2reg`	the coefficient of L2 regularization for the input layer (default = 0).
`walk.rep`	number of repeats for the random walk (default: 100).
`n.steps`	number of the random walk steps (default: 5).
`random.walk`	boolean value to enable the random walk algorithm (default: TRUE).

null.perm	boolean to enable permuting two random graphs and embed them, along with the main two graphs, for the null distribution (default: TRUE).
demo	a boolean vector to indicate this is a demo example or not
verbose	if *TRUE* a progress bar is shown.

Value

a list of embedding spaces for each node.

Examples

```
myNet = network_gen(n.nodes = 50, n.var.nodes = 5, n.var.nei = 40, noise.sd = .01)
graph_data = myNet[["data_graph"]]
embeddingSpaceList = plexi_embedding_2layer(graph.data=graph_data, train.rep=5, walk.rep=5)
```

`plexi_node_detection_2layer` *Detecting the nodes whose local neighbors change bweteen the two conditions.*

Description

Detecting the nodes whose local neighbors change bweteen the two conditions.

Usage

```
plexi_node_detection_2layer(
  embeddingSpaceList,
  p.adjust.method = "none",
  alpha = 0.05,
  rank.prc = 0.1,
  volcano.plot = TRUE,
  ranksum.sort.plot = FALSE
)
```

Arguments

embeddingSpaceList	a list obtained by the `plexi_embedding_2layer()` function.
p.adjust.method	method for adjusting p-value (including methods on `p.adjust.methods`). If set to "none" (default), no adjustment will be performed.
alpha	numeric value of significance level (default: 0.05)
rank.prc	numeric value of the rank percentage threshold (default: 0.1)
volcano.plot	boolean value for generating the Volcano plot (default: TRUE)
ranksum.sort.plot	boolean value for generating the sorted rank sum plot (default: FALSE)

Details

Calculating the distance of node pairs in the embedding space and check their significance. To find the significantly varying nodes in the 2-layer-network, the distance between the corresponding nodes are calculated along with the null distribution. The null distribution is obtained based on the pairwise distances on null graphs. if in `plexi_embedding_2layer` function `null.perm=FALSE`, the multiplex network does not have the two randomly permuted graphs, thus the distances between all the nodes will be used for the null distribution.

Value

the highly variable nodes

Examples

```
myNet = network_gen(n.nodes = 50, n.var.nodes = 5, n.var.nei = 40, noise.sd = .01)
graph_data = myNet[["data_graph"]]
embeddingSpaceList = plexi_embedding_2layer(graph.data=graph_data, train.rep=5, walk.rep=5)
Nodes = plexi_node_detection_2layer(embeddingSpaceList)
```

---

plexi_node_distance — *Detecting the nodes whose local neighbors change between the two conditions for ISNs.*

Description

Detecting the nodes whose local neighbors change between the two conditions for ISNs.

Usage

```
plexi_node_distance(embedding.space.list)
```

Arguments

`embedding.space.list`
: a list obtained by the `plexi_embedding_2layer()` function.

Details

Calculating the distance of node pairs in the embedding space and check their significance. To find the significantly varying nodes in the 2-layer-network, the distance between the corresponding nodes are calculated along with the null distribution. The null distribution is obtained based on the pairwise distances on null graphs. if in `plexi_embedding_2layer` function `null.perm=FALSE`, the multiplex network does not have the two randomly permuted graphs, thus the distances between all the nodes will be used for the null distribution.

Value

the distances for each repeat

Examples

```
myNet = network_gen(n.nodes = 50, n.var.nodes = 5, n.var.nei = 40, noise.sd = .01)
graph_data = myNet[["data_graph"]]
embeddingSpaceList = plexi_embedding(graph.data=graph_data, outcome=c(1,2),
indv.index=c(1,1), train.rep=2, random.walk=FALSE)
Dist = plexi_node_distance(embeddingSpaceList)
```

---

p_val_count *Calculate p.value for x given set of null values using counts*

Description

Calculate p.value for x given set of null values using counts

Usage

```
p_val_count(x, null.values, alternative = "two.sided")
```

Arguments

x	numeric value
null.values	a numeric vector of null distribution samples
alternative	alterative test including: "two.sided" [default], "greater", and "less"

Value

p.value

Examples

```
p.val = p_val_count(1, 1:100)
```

---

p_val_norm *Calculate p.value for x given set of null values using a Gaussian null pdf*

Description

Calculate p.value for x given set of null values using a Gaussian null pdf

Usage

```
p_val_norm(x, null.values, alternative = "two.sided")
```

Arguments

`x`	numeric value
`null.values`	a numeric vector of null distribution samples
`alternative`	alterative test including: `"two.sided"` [default], `"greater"`, and `"less"`

Value

p.value

Examples

```
p.val = p_val_norm(1, rnorm(1000,0,1))
```

`p_val_rank` *Calculate p.value for x given a set of null values using ranks*

Description

Calculate p.value for x given a set of null values using ranks

Usage

```
p_val_rank(x, null.values, alternative = "two.sided")
```

Arguments

`x`	numeric value
`null.values`	a numeric vector of null distribution samples
`alternative`	alterative test including: `"two.sided"` [default], `"greater"`, and `"less"`

Value

p.value

Examples

```
p.val = p_val_rank(1, 1:100)
```

Rank *Ranking a vector*

Description

Ranking a vector

Usage

```
Rank(x, decreasing = FALSE)
```

Arguments

x	a numeric vector
decreasing	logical. Should the sort order be increasing or decreasing? (defualt: FALSE)

Details

hint: What is the difference between Order and Rank
Order: [the index of the greatest number, ..., the index of the smallest number]
Rank: [the rank of the 1st number, ..., the rank of the last number]
In Rank, the order of the numbers remains constant so can be used for ranksum.
ex)
> a = c(10, 20, 50, 30, 40)
> order(a)
[1] 1 2 4 5 3]]
> Rank(a)
[1] 1 2 5 3 4

Value

the rank of the vector elements

Examples

```
a = c(10, 20, 50, 30, 40)
Rank(a)
```

`rep_random_walk` *Repetitive Fixed-length (weighted) random walk algorithm*

Description

Repetitive Fixed-length (weighted) random walk algorithm

Usage

```
rep_random_walk(
  graph,
  Nrep = 100,
  Nstep = 5,
  weighted_walk = TRUE,
  verbose = TRUE
)
```

Arguments

`graph`	an igraph object
`Nrep`	number of repeats (default:100)
`Nstep`	maximum number steps (default:5)
`weighted_walk`	choose the *weighted walk* algorithm if *TRUE* and *simple random* walk if *FALSE*. (default: *TRUE*)
`verbose`	if *TRUE* a progress bar is shown.

Value

Steps (S): The total number of times a node is visited starting from the corresponding node in the row. Probabilities (P): The node visit probabilities starting from the corresponding node in the row.

Examples

```
data = example_data()
RW = rep_random_walk(graph = data[["igraph_example"]])
Steps = RW[["Steps"]]
Probabilities = RW[["Probabilities"]]
```

subgraph_difference_plot
Visualization of a difference subgroup using a circular graph

Description

Visualization of a difference subgroup using a circular graph

Usage

```
subgraph_difference_plot(
  plexi.graph,
  node.importance,
  n.var.nodes = 5,
  n.neigh = 10,
  diff.threshold = 0,
  edge.width = c(0.5, 4)
)
```

Arguments

`plexi.graph`	plexi.graph data
`node.importance`	named numeric vector of the node importance to sort the nodes clockwise.
`n.var.nodes`	number of variable nodes to show
`n.neigh`	number of neighboring nodes to show
`diff.threshold`	edge threshold
`edge.width`	numeric value to adjust the thickness of the edges in plot. Two modes are defined: [i] two numbers indicating the min and max (default: c(0.5,4)); or [ii] a single number that weights the min/max of original edge weights.

Value

nothing to return

Examples

```
myNet = network_gen(n.nodes = 100, n.var.nodes = 5, n.var.nei = 90, noise.sd = .01)
graph_data = myNet[["data_graph"]]
node_importance_dummy = 1:100
names(node_importance_dummy) = 1:100
subgraph_difference_plot(graph_data, node.importance = node_importance_dummy)
```

`subgraph_plot` *Visualization of a subgroup using a circular graph*

Description

Visualization of a subgroup using a circular graph

Usage

```
subgraph_plot(
  graph,
  node_set,
  labels = NULL,
  node.importance = NULL,
  n.nodes = NULL,
  node_size = 5,
  font_size = 4,
  edge_width = c(0.5, 4),
  margin = 2.5
)
```

Arguments

- `graph` an igraph object
- `node_set` the names or indices of the nodes around which the subgroup is plotted.
- `labels` the labels of the nodes to be indicated. Labels should be a named vector if the `node_set` consists of the node names.
- `node.importance` named numeric vector of the node importance to sort the nodes clockwise.
- `n.nodes` number of nodes to be displayed. If NULL, all the `node_set` and their neighbors are considered.
- `node_size` size of the nodes in plot (default: 5)
- `font_size` font size of labels if available (default: 4)
- `edge_width` numeric value to adjust the thickness of the edges in plot. Two modes are defined: [i] two numbers indicating the min and max (default: c(0.5,4)); or [ii] a single number that weights the min/max of original edge weights.
- `margin` the figure margin (default: 2.5)

Details

This function plots a sub-graph given by a set of nodes as circular plot. the main inputs to the function are: a graph (as an igraph object) and a set of nodes (e.g. highly variable nodes) around which the subgroup is calculated.

Value

nothing to return

Examples

```
data = example_data()
subgraph_plot(graph = data[["igraph_example"]], node_set = "a")
```

---

weightd_random_walk — *Weighted Random Walk algorithm*

Description

Weighted Random Walk algorithm

Usage

```
weightd_random_walk(graph, startNode, maxStep = 5, node_names = FALSE)
```

Arguments

`graph`	an igraph object
`startNode`	the starting node (i.e. a node name or a node index)
`maxStep`	maximum number steps (default:5)
`node_names`	a list of names for nodes

Value

The set of nodes passed by the random walker.

Examples

```
data = example_data()
nodePath = weightd_random_walk(graph = data[["igraph_example"]], startNode = 1)
```

Index