Package ‘TopoRNet’

October 13, 2020

Type Package

Title Analysis and Visualization of Topology Representing Networks

Version 1.0

Date 2020-04-25

Author Josh Taylor

Maintainer Josh Taylor <josh@somdisco.com>

Description TopoRNet provides access to the CONNvis visualization, Topology Preservation metrics, and pruning for Topology Representing Networks.

License GPL (>= 2)

Imports Rcpp (>= 1.0.1),
igraph,
dplyr,
Rdpack,
ggplot2,
changepoint,
ggrepel

LinkingTo Rcpp, RcppArmadillo, RcppParallel

RoxygenNote 7.1.1

Suggests knitr,
rmarkdown,
SOMDisco

VignetteBuilder knitr

RdMacros Rdpack

LazyData true

R topics documented:

as_list *Convert an TRN object to a list*

Description

This method extracts all fields of an TRN object and places their stored values into the fields of an R list, with list field names matching the TRN object field names.

Usage

```
TRNobj$as_list()
```

Value

A list

CADJ_active_verts *Return the CADJ active vertices*

Description

Active CADJ vertices participate in active CADJ edges; Thus if there are vertices in the graph with no active CADJ edges connecting them they are deemed inactive.

Usage

```
TRNobj$CADJ_active_verts()
```

Value

a list of vertex IDs that are connected by any `CADJ_active` edge.

calc_DMPrune_LambdaPath *Calculate the DM-Prune Lambda Path*

Description

The DM-Prune Lambda path, at each prune step t, is the (normalized) Dirichlet-Multinomial likelihood of CADJ edges after pruning edges whose `DMP_mu_rank` is **strictly less than** t.

Usage

```
TRNobj$calc_DMPrune_LambdaPath(priorADJ)
```

Arguments

`priorADJ`	an adjacency matrix (nrows = ncols = `nV`) whose (i,j) entries contain the Dirichlet prior CADJ values.

calc_TopoMeasures *Calculate Topology Preserving Measures of the TRN*

Description

Sets the Topology Preserving Measures of both the CADJ and CONN graphs (Topographic Product and several Topographic Functions). See `?Topographic_Product` and `?Topographic_Functions` for more details.

Usage

```
TRNobj$calc_TopoMeasures

Bauer H, Pawelzik K, Geisel T (1992).
A Topographic Product for the Optimization of Self-Organizing Feature Maps.
In Moody JE, Hanson SJ, Lippmann RP (eds.), Advances in Neural Information Processing Systems 4, 1141
Morgan-Kaufmann.
\url{http://papers.nips.cc/paper/508-a-topographic-product-for-the-optimization-of-self-organizi
Villmann T, Der R, Herrmann M, Martinetz TM (1997).
Topology preservation in self-organizing feature maps: exact definition and measurement.
IEEE Transactions on Neural Networks, 8(2), 256-266.
Zhang L, Merényi E (2006).
Weighted differential topographic function: Arefinement of the topographic function.
In in Proc. 14th European Symposium on Artificial Neural Networks (ESANN'2006, 13--18.
```

CONN_active_verts *Return the CONN active vertices*

Description

Active CONN vertices participate in active CONN edges; Thus if there are vertices in the graph with no active CONN edges connecting them they are deemed inactive.

Usage

```
TRNobj$CONN_active_verts()
```

Value

a list of vertex IDs that are connected by any `CONN_active` edge.

`dDirMult` *Density of Dirichlet-Multinomial Distribution*

Description

Computes the density of a Dirichlet-Multinomial (DM) distribution, at a given count vector `x`, with a user-specified prior count vector `prior`.

Usage

```
dDirMult(x, prior, log_form = TRUE, normalize = TRUE)
```

Arguments

`x`	the count vector at which to evaluate the DM density
`prior`	the prior count vector, must have `length` = `length(x)`
`log_form`	whether to return the log-density, default = TRUE
`normalize`	whether to normalize the calculated density by `sum(x)`. Default = TRUE.

Value

The evaluated DM density (a number)

`DMPrune_CADJ_step` *Prune CADJ at a DM-Prune step*

Description

Prune the CADJ graph by removing edges in `CADJ_EL` whose `DMP_mu_rank` is **strictly less than** a given minimum prune step.

Usage

```
TRNobj$DMPrune_CADJ_step(min_step)
```

Arguments

`min_step`

get_CADJ *Return the CADJ adjacency matrix*

Description

This method returns the CADJ adjacency matrix of the TRN (as set during `initialize_TRN`) whose `CADJ_active` flag = 1.

Usage

```
TRNobj$get_CADJ()
```

Value

an adjacency matrix (nrows = ncols = `nV`) whose (i,j) entries are the `CADJ` edge weights.

get_CADJvis_colors *Get the CADJvis edge colors*

Description

The edge colorings (red, blue, green, yellow, grayscale) are assigned in increasing order of an edge's `CADJ_lrank`.

Usage

```
TRNobj$get_CADJvis_colors()
```

Value

A vector containing the CADJvis (hex) color of each edge in `CADJ_EL`.

References

Taşdemir K, Merényi E (2009). "Exploiting Data Topology in Visualization and Clustering of Self-Organizing Maps." *IEEE Transactions on Neural Networks*, **20**(4), 549-562. ISSN 1045-9227, doi: 10.1109/TNN.2008.2005409.

get_CADJvis_stats *Get the CADJvis summary statistics.*

Description

The CADJvis summary statistics are computed for each unique grouping found in CADJ_lrank, CADJ_grank, and CADJ_fwdflen.

Usage

```
TRNobj$get_CADJvis_stats(group)
```

Arguments

group	a string identifying which summary statistics to return. Can be one of 'lrank', 'grank', or 'length'.

Value

A data frame with summary statistics for each group in its rows, and columns:

- lrank (or grank or length as applicable), identifying the grouping for each set of summary stats
- count the number of CADJ edges in the group
- pct the proportion of the total number of CADJ edges in each group
- cumpct the cumulative proportion of CADH edges in each group (as ordered by the grouping variable)
- mean the average of CADJ edge weights in each group
- sd the standard deviation of CADJ edge weights in each group
- q0 the 0.00 quantile (minimum) of CADJ edge weights in each group
- q25 the 0.25 quantile (Q1) of CADJ edge weights in each group
- q50 the 0.50 quantile (median) of CADJ edge weights in each group
- q75 the 0.75 quantile (Q3) of CADJ edge weights in each group
- q100 the 1.00 quantile (maximum) of CADJ edge weights in each group

References

Taşdemir K, Merényi E (2009). "Exploiting Data Topology in Visualization and Clustering of Self-Organizing Maps." *IEEE Transactions on Neural Networks*, **20**(4), 549-562. ISSN 1045-9227, doi: 10.1109/TNN.2008.2005409.

get_CADJvis_widths *Get the CADJvis edge widths*

Description

The edge widths are assigned in reverse of each edge's CADJ_grank. That is, if a CADJ graph has unique global ranks of 1,2,3,4,5, then edges with those global ranks will have widths = 5,4,3,2,1.

Usage

```
TRNobj$get_CADJvis_widths()
```

Value

A vector containing the CADJvis widths of each edge in CADJ_EL.

References

Taşdemir K, Merényi E (2009). "Exploiting Data Topology in Visualization and Clustering of Self-Organizing Maps." *IEEE Transactions on Neural Networks*, **20**(4), 549-562. ISSN 1045-9227, doi: 10.1109/TNN.2008.2005409.

get_CONN *Return the CONN adjacency matrix*

Description

This method returns the CONN adjacency matrix of the TRN (as set during initialize_TRN) whose CONN_active flag = 1.

Usage

```
TRNobj$get_CONN()
```

Value

an adjacency matrix (nrows = ncols = nV) whose (i,j) entries are the CONN edge weights.

`get_CONNvis_colors` *Get the CONNvis edge colors*

Description

The edge colorings (red, blue, green, yellow, grayscale) are assigned in increasing order of an edge's `CONN_lrank`.

Usage

```
TRNobj$get_CONNvis_colors()
```

Value

A vector containing the CONNvis (hex) color of each edge in `CONN_EL`.

References

Taşdemir K, Merényi E (2009). "Exploiting Data Topology in Visualization and Clustering of Self-Organizing Maps." *IEEE Transactions on Neural Networks*, **20**(4), 549-562. ISSN 1045-9227, doi: 10.1109/TNN.2008.2005409.

`get_CONNvis_stats` *Get the CONNvis summary statistics.*

Description

The CONNvis summary statistics are computed for each unique grouping found in `CONN_lrank`, `CONN_grank`, and `CONN_fwdflen`.

Usage

```
TRNobj$get_CONNvis_stats(group)
```

Arguments

`group`	a string identifying which summary statistics to return. Can be one of 'lrank', 'grank', or 'length'.

Value

A data frame with summary statistics for each group in its rows, and columns:

- `lrank` (or `grank` or `length` as applicable), identifying the grouping for each set of summary stats
- `count` the number of CONN edges in the group
- `pct` the proportion of the total number of CONN edges in each group
- `cumpct` the cumulative proportion of CADH edges in each group (as ordered by the grouping variable)

- `mean` the average of CONN edge weights in each group
- `sd` the standard deviation of CONN edge weights in each group
- `q0` the 0.00 quantile (minimum) of CONN edge weights in each group
- `q25` the 0.25 quantile (Q1) of CONN edge weights in each group
- `q50` the 0.50 quantile (median) of CONN edge weights in each group
- `q75` the 0.75 quantile (Q3) of CONN edge weights in each group
- `q100` the 1.00 quantile (maximum) of CONN edge weights in each group

References

Taşdemir K, Merényi E (2009). "Exploiting Data Topology in Visualization and Clustering of Self-Organizing Maps." *IEEE Transactions on Neural Networks*, **20**(4), 549-562. ISSN 1045-9227, doi: 10.1109/TNN.2008.2005409.

`get_CONNvis_widths` *Get the CONNvis edge widths*

Description

The edge widths are assigned in reverse of each edge's `CONN_grank`. That is, if a CONN graph has unique global ranks of 1,2,3,4,5, then edges with those global ranks will have widths = 5,4,3,2,1.

Usage

```
TRNobj$get_CONNvis_widths()
```

Value

A vector containing the CONNvis widths of each edge in `CONN_EL`.

References

Taşdemir K, Merényi E (2009). "Exploiting Data Topology in Visualization and Clustering of Self-Organizing Maps." *IEEE Transactions on Neural Networks*, **20**(4), 549-562. ISSN 1045-9227, doi: 10.1109/TNN.2008.2005409.

`get_OTADJ` *Return the TRN's output space adjacency matrix*

Description

This method returns the adjacency matrix of the output space topology of the TRN (as set during `initialize_TRN`),

Usage

```
TRNobj$get_OTADJ()
```

Value

an adjacency matrix (nrows = ncols = `nV`)

`initialize_TRN` *Initialize a TRN object*

Description

Sets the CADJ, CONN, and output topology graphs.

Usage

```
TRNobj$initialize_TRN(CADJ, OTADJ)
Martinetz T, Schulten K (1994).
Topology representing networks.
Neural Networks, 7(3), 507--522.
```

Arguments

`CADJ`	The CADJ adjacency matrix of the TRN.
`OTADJ`	The adjacency matrix of the TRN's output topology (e.g., the SOM neuron lattice adjacency). Only binary output topology adjacency matrices are supported at this time. ($OTADJ_{ij} = 1$ indicate an edge between vertices `i` and `j`).

Details

The following fields are computed and stored internally:

- `nV`, set = nrow(CADJ).
- `CADJ_EL` and `CONN_EL`, the CADJ and CONN edge lists.
- `CADJ` and `CONN`, the CADJ and CONN edge weights.
- `CADJ_nE` and `CONN_nE`, the number of edges in the CADJ and CONN graphs.
- `CADJ_deg` and `CONN_deg`, the degree of each TRN vertex in the CADJ and CONN graphs.
- `CADJ_wdeg` and `CONN_wdeg`, the weighted degree of each TRN vertex in the CADJ and CONN graphs.
- `OT_EL` and `OT_nE` and `OT_deg` the edge list, number of edges, and vertex degrees in the TRN's output topology.
- `OT_max_dist`, the maximum geodesic distance of any two vertices in the TRN's output topology.
- `CADJ_fwdflen` and `CONN_fwdflen`, the forward folding lengths of each CADJ and CONN edge.
- `CADJ_bwdflen` and `CONN_bwdflen`, the backward folding lengths of each output topology edge on the CADJ and CONN graphs.
- `CADJ_TP_radius` and `CONN_TP_radius`, the Topology Preserving Radius of the CADJ and CONN graphs.
- `CADJ_lrank` and `CONN_lrank` The local ranks of each CADJ and CONN edge.
- `CADJ_grank` and `CONN_grank` The global ranks of each CADJ and CONN edge.

Both `CADJ` and `OTADJ` should be square, and have the same dimensions. In addition to the above list of internally computed quantities, `initialize_TRN` also computes the CADJvis and CONNvis summary statistics (retrievable via get_CADJvis_stats and get_CONNvis_stats, respectively), and sets all `CADJ_active` and `CONN_active` flags = 1.

Value

None

`load` *Load an existing TRN object*

Description

TRN objects previously written to disk via the save method can be re-loaded into a new R environment with this function. All fields of the internal C++ class will be populated, and all methods can be called on the loaded TRN object.

Usage

```
TRNobj$load(TRNfile)
```

Arguments

`trnfile`	a string indicating the file path and name of the saved TRN object.

Details

Because the .trn file is in .rds format it can, technically, be loaded directly into an R environment as a list via readRDS. This can be useful for spot checking the contents of a saved TRN object, but does not allow use of any of its methods (or visualizations). The `load` method allows for proper restoration of a previously saved TRN

Value

None, the TRN object is loaded

`load_list` *Populate a TRN object from a list*

Description

This method populates all fields of a TRN object from the fields of an R list object. The list must have field names which exactly match the TRN field names.

Usage

```
TRNobj$load_list(TRNList)
```

Arguments

`TRNList`	a TRN object converted to a list, e.g., with `as_list`

Value

None

`new` *Create an empty TRN object*

Description

Create an empty TRN object

Usage

```
TRN$new()
```

Value

an empty TRN object templated for initialization (via initialize_TRN).

`prune_CADJ_edge_id` *Prune CADJ by edge ids*

Description

Prune the CADJ graph given a list of edge IDs to remove.

Usage

```
TRNobj$prune_CADJ_edge_id(edge_id)
```

Arguments

`edge_id`	a vector containing the rows indices of `CADJ_EL` which contain the edges desired to prune.

Value

None

`prune_CADJ_edge_list` *Prune CADJ by an edge list*

Description

Prune the CADJ graph given a list of edges to remove.

Usage

```
TRNobj$prune_CADJ_edge_list(edge_list)
```

Arguments

`edge_list`	a 2-column matrix rows define the vertices connected by each CADJ edge to prune

Details

The vertex indices in `edge_list` should be 1-based.

Value

None

`prune_CADJ_edge_weight` *Prune CADJ by a minimum edge weight*

Description

Prune the CADJ graph by removing edges whose weights are **strictly less than** a given minimum edge weight.

Usage

```
TRNobj$prune_CADJ_edge_weight(min_weight)
```

Arguments

`min_weight`	minimum weight allowed in the pruned graph. Any edges in `CADJ_EL` whose `CADJ` value is < `min_weight` will be pruned.

Value

None

`prune_CADJ_grank` *Prune CADJ by a maximum global rank*

Description

Prune the CADJ graph by removing edges whose `CADJ_grank` is **strictly greater than** a given maximum grank.

Usage

```
TRNobj$prune_CADJ_grank(max_grank)
```

Arguments

`max_grank`	maximum `CADJ_grank` allowed in the pruned graph. Any edges whose `CADJ_grank` is > `max_grank` will be pruned.

Value

None

prune_CADJ_length *Prune CADJ by a maximum output topology length*

Description

Prune the CADJ graph by removing edges whose `CADJ_fwdflen` is **stricly greater than** a given maximim forward folding length.

Usage

```
TRNobj$prune_CADJ_length(max_fwdflen)
```

Arguments

`max_fwdflen`	maximum `CADJ_fwdflen` allowed in the pruned graph. Any edges whose `CADJ_fwdflen` is > `max_fwdflen` will be pruned.

Value

None

prune_CADJ_lrank *Prune CADJ by a maximum local rank*

Description

Prune the CADJ graph by removing edges whose `CADJ_lrank` is **strictly greater than** a given maximum lrank.

Usage

```
TRNobj$prune_CADJ_lrank(max_lrank)
```

Arguments

`max_lrank`	maximum `CADJ_lrank` allowed in the pruned graph. Any edges whose `CADJ_lrank` is > `max_lrank` will be pruned.

Value

None

`prune_CADJ_vertex_id` *Prune CADJ by a set of vertex ids*

Description

Prune the CADJ graph by removing edges connecting vertices in a list of given vertex IDs.

Usage

```
TRNobj$prune_CADJ_vertex_id(vertex_id)
```

Arguments

`vertex_id` a vector containing the vertices to prune. All connections in `CADJ_EL` from or to the vertices in `vertex_id` will be pruned.

Value

None

`prune_CADJ_wdeg` *Prune CADJ by a minimum vertex weight*

Description

Prune the CADJ graph by removing edges incident to vertices whose `CADJ_wdeg` is **stricly less than** a given minimum weighted degree.

Usage

```
TRNobj$prune_CADJ_wdeg(min_weight)
```

Arguments

`min_weight` minimum vertex weight allowed in the pruned graph. Any vertices whose `CADJ_wdeg` is < `min_weight` will be pruned.

Value

None

`prune_CONN_CADJ` *Prune CONN by CADJ*

Description

Prune the CONN graph by removing CADJ inactive edges.

Usage

```
TRNobj$prune_CONN_CADJ()
```

Value

None

`prune_CONN_edge_id` *Prune CONN by edge ids*

Description

Prune the CONN graph given a list of edge IDs to remove.

Usage

```
TRNobj$prune_CONN_edge_id(edge_id)
```

Arguments

`edge_id`	a vector containing the rows indices of `CONN_EL` which contain the edges desired to prune.

Value

None

`prune_CONN_edge_list` *Prune CONN by an edge list*

Description

Prune the CONN graph given a list of edges to remove.

Usage

```
TRNobj$prune_CONN_edge_list(edge_list)
```

Arguments

`edge_list`	a 2-column matrix rows define the vertices connected by each CONN edge to prune

Details

The vertex indices in `edge_list` should be 1-based.

Value

None

`prune_CONN_edge_weight` *Prune CONN by a minimum edge weight*

Description

Prune the CONN graph by removing edges whose weights are **strictly less than** a given minimum edge weight.

Usage

```
TRNobj$prune_CONN_edge_weight(min_weight)
```

Arguments

`min_weight`	minimum weight allowed in the pruned graph. Any edges in `CONN_EL` whose `CONN` value is < `min_weight` will be pruned.

Value

None

`prune_CONN_grank` *Prune CONN by a maximum global rank*

Description

Prune the CONN graph by removing edges whose `CONN_grank` is **strictly greater than** a given maximum grank.

Usage

```
TRNobj$prune_CONN_grank(max_grank)
```

Arguments

`max_grank`	maximum `CONN_grank` allowed in the pruned graph. Any edges whose `CONN_grank` is > `max_grank` will be pruned.

Value

None

`prune_CONN_length` *Prune CONN by a maximum output topology length*

Description

Prune the CONN graph by removing edges whose `CONN_fwdflen` is **stricly greater than** a given maximim forward folding length.

Usage

```
TRNobj$prune_CONN_length(max_fwdflen)
```

Arguments

`max_fwdflen`	maximum `CONN_fwdflen` allowed in the pruned graph. Any edges whose `CONN_fwdflen` is > `max_fwdflen` will be pruned.

Value

None

`prune_CONN_lrank` *Prune CONN by a maximum local rank*

Description

Prune the CONN graph by removing edges whose `CONN_lrank` is **strictly greater than** a given maximum lrank.

Usage

```
TRNobj$prune_CONN_lrank(max_lrank)
```

Arguments

`max_lrank`	maximum `CONN_lrank` allowed in the pruned graph. Any edges whose `CONN_lrank` is > `max_lrank` will be pruned.

Value

None

prune_CONN_vertex_id *Prune CONN by a set of vertex ids*

Description

Prune the CONN graph by removing edges connecting vertices in a list of given vertex IDs.

Usage

```
TRNobj$prune_CONN_vertex_id(vertex_id)
```

Arguments

`vertex_id`	a vector containing the vertices to prune. All connections in `CONN_EL` from or to the vertices in `vertex_id` will be pruned.

Value

None

prune_CONN_wdeg *Prune CONN by a minimum vertex weight*

Description

Prune the CONN graph by removing edges incident to vertices whose `CONN_wdeg` is **stricly less than** a given minimum weighted degree.

Usage

```
TRNobj$prune_CONN_wdeg(min_weight)
```

Arguments

`min_weight`	minimum vertex weight allowed in the pruned graph. Any vertices whose `CONN_wdeg` is < `min_weight` will be pruned.

Value

None

restore_CADJ_edges *Restore all CADJ edges*

Description

When pruning of CADJ edges occurs (via any of the prune_CADJ_* methods), the CADJ_active flag corresponding to the pruned edges is changed from 1 to 0. This method restores all CADJ_active flags to 1 (reversing any existing pruning).

Usage

```
TRNobj$restore_CADJ_edges()
```

Value

None

restore_CONN_edges *Restore all CONN edges*

Description

When pruning of CONN edges occurs (via any of the prune_CONN_* methods), the CONN_active flag corresponding to the pruned edges is changed from 1 to 0. This method restores all CONN_active flags to 1 (reversing any existing pruning).

Usage

```
TRNobj$restore_CONN_edges()
```

Value

None

save *Save a TRN object*

Description

All fields in a TRN object can be saved to disk with this function, which allows them to be re-loaded into a new R environment at a later time.

Usage

```
TRNobj$save(trnfile)
```

Arguments

`trnfile`	a string indicating the file path and name in which to save the TRN object. This must end in extension ".trn", otherwise an error is returned.

Details

The TRN object is saved to disk as an R list, with each field occupying a corresponding field of the list. The file is saved in .rds format (can check its details with infoRDS).

Saved TRNs can be re-loaded with load

Value

None, the TRN object is saved to disk

`set_BootSig` *Set the CADJ significances*

Description

Set the Bootstrap significance values of each CADJ and CONN edge.

Usage

```
TRNobj$set_BootSig(BSADJ)
```

Arguments

`BSADJ`	an adjacency matrix (nrows = ncols = `nV`) whose (i,j) entries contain significance values of each CADJ edge.

`set_parallel` *Setter function for parallel computation.*

Description

Sets an internal flag controlling whether computations are performed in parallel.

Usage

```
TRNobj$set_parallel(parallel)
```

Arguments

`parallel`	either TRUE or FALSE, as desired

Details

Parallel computation is supported via RcppParallel, see setThreadOptions for details to control threading. By default, `parallel` = `TRUE` at TRNobj instantiation.

Value

None, `parallel` field is set internally

`Topographic_Functions` *Topographic Functions of a TRN*

Description

The Topograph Functions of a TRN measure topology preservation across a range of "folding" lengths, which are defined as the length (geodesic distance) by which edges in one space (input / output) must "fold" in order to be represented in the other.

The functions computed here are the TF (Topographic Function) of Villmann et al. and the DTF (Differential Topographic Function) & WDTF (Weighted DTF) of Zhang et al.

Usage

```
Topographic_Functions(inputADJ, outputADJ, parallel = TRUE)
```

Arguments

`inputADJ`	the adjacency matrix of vertices in the Input space of the TRN
`outputADJ`	the adjacency matrix of vertices in the Output sapce of the TRN
`parallel,`	whether to compute in parallel. Default = T.

Value

a data frame with columns:

- `k` list of folding lengths
- `TF` the TF computed at each folding length
- `DTF` the DTF computed at each folding length
- `WDTF` the WDTF computed at each folding length

Villmann T, Der R, Herrmann M, Martinetz TM (1997). “Topology preservation in self-organizing feature maps: exact definition and measurement.” *IEEE Transactions on Neural Networks*, **8**(2), 256-266. Zhang L, Merényi E (2006). “Weighted differential topographic function: Arefinement of the topographic function.” In *in Proc. 14th European Symposium on Artificial Neural Networks (ESANN'2006*, 13–18.

`Topographic_Product` *Topographic Product of a TRN*

Description

The Topographic Product of a TRN which is imbued with an output topology.

Usage

```
Topographic_Product(input_coords, output_coords, parallel = TRUE)
```

Arguments

`input_coords`	the coordinates of the vertices in the Input space (i.e, R^d) of the TRN
`output_coords`	the coordinates of the vertices in the Output space of the TRN
`parallel,`	whether to compute in parallel. Default = T.

Value

The Topographic Product (a number)

References

Bauer H, Pawelzik K, Geisel T (1992). "A Topographic Product for the Optimization of Self-Organizing Feature Maps." In Moody JE, Hanson SJ, Lippmann RP (eds.), *Advances in Neural Information Processing Systems 4*, 1141–1147. Morgan-Kaufmann. http://papers.nips.cc/paper/508-a-topographic-product-for-the-optimization-of-self-organizing-feature-maps.pdf.

`TRN` *The TRN object class*

Description

The TRN object class

Fields

- `parallel` Flag indicating whether computations should be performed in parallel, using the RcppParallel package.
- `nV` Number of vertices in the TRN.
- `CADJ_nE` Number of edges in the CADJ graph.
- `CONN_nE` Number of edges in the CONN graph.
- `OT_nE` Number of edges in the output space topology of the TRN (i.e., the SOM lattice).
- `CADJ_EL` List of CADJ edges. This is a 2-column matrix whose rows define the vertices connected by each CADJ edge.
- `CONN_EL` List of CONN edges. This is a 2-column matrix whose rows define the vertices connected by each CONN edge.

`OT_EL` List of edges in the output space topology of the TRN. This is a 2-column matrix whose rows define the vertices connected by each output space edge.

`CADJ` Weights of each CADJ edge defined in `CADJ_EL`. This is a vector of `length` = `nrow(CADJ_EL)`.

`CONN` Weights of each CONN edge defined in `CONN_EL`. This is a vector of `length` = `nrow(CONN_EL)`.

`CADJ_deg` The degree (number of incident edges) of each TRN vertex, according to the CADJ graph.

`CONN_deg` The degree (number of incident edges) of each TRN vertex, according to the CONN graph.

`OT_deg` The degree (number of incident edges) of each TRN vertex, according to the output space topology.

`CADJ_wdeg` The weighted degree of each TRN vertex (sum of edge weights emanating from each vertex), according to the CADJ graph.

`CONN_wdeg` The weighted degree of each TRN vertex (sum of edge weights emanating from each vertex), according to the CONN graph.

`CADJ_fwdflen` The forward folding length of each CADJ edge listed in `CADJ_EL`. This is the geodesic distance of each edge, measured on the TRN's output topology.

`CONN_fwdflen` The forward folding length of each CONN edge listed in `CONN_EL`. This is the geodesic distance of each edge, measured on the TRN's output topology.

`CADJ_bwdflen` The backward folding length (geodesic distance) of each edge in the TRN's output topology, measured on the CADJ graph. `length` = `nrow(OT_EL)`.

`CONN_bwdflen` The backward folding length (geodesic distance) of each edge in the TRN's output topology, measured on the CONN graph. `length` = `nrow(OT_EL)`.

`CADJ_TP_radius` The neighborhood size on the TRN's output topology which would preserve all of the topological adjacencies found in the CADJ graph, if all of these adjacencies were packed into a neighborhood of this radius.

`CONN_TP_radius` The neighborhood size on the TRN's output topology which would preserve all of the topological adjacencies found in the CONN graph, if all of these adjacencies were packed into a neighborhood of this radius.

`OT_geodesic_dist` The geodesic distance between TRN vertices, measured according to its output topology.

`OT_max_dist` The maximum geodesic distance between any two TRN vertices, measured according to its output topology.

`OT_nhb_sizes` The cumulative number of neighbors of each TRN vertex at all possible geodesic distances in its output topology, stored in a `nV` x `OT_max_dist` matrix.

`CADJ_lrank` The CADJvis local rank of each CADJ edge in `CADJ_EL`.

`CADJ_grank` The CADJvis global rank of each CADJ edge in `CADJ_EL`.

`CONN_lrank` The CONNvis local rank of each CONN edge in `CONN_EL`.

`CONN_grank` The CONNvis global rank of each CONN edge in `CONN_EL`.

`CADJ_active` Indicator flag identifying whether each edge in `CADJ_EL` is active (=1) or not (=0). "active" = un-pruned, i.e. not removed by thresholding (so all edges are active until any of the `prune_CADJ_*` methods are called).

`CONN_active` Indicator flag identifying whether each edge in `CONN_EL` is active (=1) or not (=0). "active" = un-pruned, i.e. not removed by thresholding (so all edges are active until any of the `prune_CONN_*` methods are called).

`TopoProd` The Topographic Product of the TRN. See `?set_TopoPresMeasures` for details.

CADJ_TopoFxns The suite of Topographic Functions of the CADJ graph. See ?set_TopoPresMeasures for details.

CONN_TopoFxns The suite of Topographic Functions of the CONN graph. See ?set_TopoPresMeasures for details.

CADJ_BootSig The significance values for each edge in CADJ_EL, as derived from a bootstrapped re-sampling procedure.

CONN_BootSig The significance values for each edge in CONN_EL, as derived from a bootstrapped re-sampling procedure.

DMP_prior The DM-Prune priors for each edge in CADJ_EL.

DMP_mu_G The DM-Prune global μ score for each edge in CADJ_EL.

DMP_mu_N The DM-Prune local μ score for each edge in CADJ_EL.

DMP_mu_L The DM-Prune length μ score for each edge in CADJ_EL.

DMP_mu_S The DM-Prune bootstrap significance μ score for each edge in CADJ_EL.

DMP_mu The DM-Prune composite μ score for each edge in CADJ_EL.

DMP_mu_rank The rank of each edge in CADJ_EL, according to their DM-Prune composite μ scores. The ranking mechanism is ascending and dense, meaning the edge with the lowest μ has rank = 1, and any edges with identical mu scores share the same rank.

DMP_pruneStep The unique prune steps at which the DM-Prune Λ-path is calculated. These steps are the unique values in DMP_mu_rank.

DMP_Lambda The DM-Prune Λ path, computed at each step in DMP_pruneStep.

isinit Flag indicating whether initialize_TRN has been called.

isset_TPM Flag indicating whether calc_TopoMeasures has been called.

isset_DMPrune Flag indicating whether calc_DMPrune_LambdaPath has been called.

Methods

Each class method has its own documentation, accessible via ?TopoRNet::<method_name>. For completeness, the list is repeated here in entirety. Additional functionality for visualizing a TRN object is available through the vis_* functions. See their documentation for more information.

TRN$new Instantiate a TRN object.

set_parallel Set the parallel computation flag.

initialize_TRN Initialize a TRN object with CADJ and output topology adjacency matrices.

get_CADJvis_stats Get the CADJvis summary statistics, grouped by lrank, grank, or length.

get_CADJvis_colors Get the CADJvis colors of each edge in CADJ_EL, used for producing the CADJvis.

get_CADJvis_widths Get the CADJvis line widths of each edge in CADJ_EL, used for producing the CADJvis.

get_CONNvis_stats Get the CONNvis summary statistics, grouped by lrank, grank, or length.

get_CONNvis_colors Get the CONNvis colors of each edge in CONN_EL, used for producing the CONNvis.

get_CONNvis_widths Get the CONNvis line widths of each edge in CONN_EL, used for producing the CONNvis.

prune_CADJ_edge_list Prune the CADJ graph given a list of edges to remove.

prune_CADJ_edge_id Prune the CADJ graph given a list of edge IDs to remove.

prune_CADJ_vertex_id Prune the CADJ graph by removing edges connecting vertices in a list of given vertex IDs.

prune_CADJ_edge_weight Prune the CADJ graph by removing edges whose weights are < a given minimum edge weight.

prune_CADJ_lrank Prune the CADJ graph by removing edges whose CADJ_lrank is > a given maximum lrank.

prune_CADJ_grank Prune the CADJ graph by removing edges whose CADJ_grank is > a given maximum grank.

prune_CADJ_length Prune the CADJ graph by removing edges whose CADJ_fwdflen is > a given maximim forward folding length.

prune_CADJ_wdeg Prune the CADJ graph by removing edges incident to vertices whose CADJ_wdeg is < a given minimum weighted degree.

get_CADJ Get the CADJ adjacency matrix of all CADJ_active edges (i.e., the adjacency matrix returned excludes any pruned edges).

restore_CADJ_edges Set all edges in CADJ_EL as active, which erases any graph pruning that has been done via the prune_CADJ_* methods.

prune_CONN_edge_list Prune the CONN graph given a list of edges to remove.

prune_CONN_edge_id Prune the CONN graph given a list of edge IDs to remove.

prune_CONN_vertex_id Prune the CONN graph by removing edges connecting vertices in a list of given vertex IDs.

prune_CONN_edge_weight Prune the CONN graph by removing edges whose weights are < a given minimum edge weight.

prune_CONN_lrank Prune the CONN graph by removing edges whose CONN_lrank is > a given maximum lrank.

prune_CONN_grank Prune the CONN graph by removing edges whose CONN_grank is > a given maximum grank.

prune_CONN_length Prune the CONN graph by removing edges whose CONN_fwdflen is > a given maximim forward folding length.

prune_CONN_wdeg Prune the CONN graph by removing edges incident to vertices whose CONN_wdeg is < a given minimum weighted degree.

prune_CONN_CADJ Prune the CONN graph by removing CADJ inactive edges.

get_CONN Get the CONN adjacency matrix of all CONN_active edges (i.e., the adjacency matrix returned excludes any pruned edges).

restore_CONN_edges Set all edges in CONN_EL as active, which erases any graph pruning that has been done via the prune_CONN_* methods.

get_OTADJ Get the adjacency matrix of the output space topology of the TRN. This is the same adjacency that was given (and stored) during a call to initialize_TRN.

CADJ_active_verts Return a list of vertex IDs that are connected by any CADJ_active edge.

CONN_active_verts Return a list of vertex IDs that are connected by any CONN_active edge.

calc_TopoMeasures Sets the Topology Preserving Measures of both the CADJ and CONN graphs (Topographic Product and several Topographic Functions).

set_BootSig Set the Bootstrap significance values of each CADJ and CONN edge.

calc_DMPrune_LambdaPath Calculate the DM-Prune Lambda path.

DMPrune_CADJ_step Prune the CADJ graph by removing edges in CADJ_EL whose DMP_mu_rank is < a given minimum prune step.

`save` Save a TRN object to disk

`load` Load a previously saved TRN object from disk

`as_list` Convert and return all fields of a TRN object to an R list.

`load_list` Populate an instance of a TRN object from an R list.

`vis_CADJvis` *CADJvis Visualization*

Description

In CADJvis, plotted edges only extend to the midpoint between source and sink vertices in order to highlight the asymmetries in the CADJ graph. The colors and line widths are computed relative to CADJ (not CONN).

Usage

```
vis_CADJvis(
  TRN,
  add = F,
  vertex.xy,
  vertex.pch = 16,
  vertex.cex = 1,
  vertex.col = "black",
  edge.lwd_range = NULL,
  vertex.active = T,
  vertex.subset = NULL
)
```

Arguments

`TRN` a TRN object

`add` whether to create a new plotting device (=FALSE, default), or add to an existing one (=TRUE)

`vertex.xy` a matrix of (x,y) coordinates (nrows = TRN$nV, ncols = 2) defining the plot coordinates of the vertices of the TRN. It is assumed that the `i-th` row of `vertex.xy` gives the coordinates for vertex `i` of the graph.

`vertex.pch` the pch symbol plotted vertices, default = 16.

`vertex.cex` the cex of plotted vertices, default = 1. Set = 0 to suppress vertex plotting.

`vertex.col` the color of plotted vertices, default = 'black'

`edge.lwd_range` the min/max range of the plotted line widths. Default = NULL means line widths inherit from the `TRN$CADJ_grank` of each edge. If supplied as something other than NULL, it must be a length=2 vector giving the (lower,upper) bounds of plotted edge widths.

`vertex.active` whether to restrict plotted vertices to those which have an edge connecting them, default = TRUE.

`vertex.subset` a vector of vertex indices to restrict the plotting to. Default = NULL, meaning all edges and vertices are plotted. If given, any edges connecting vertices in this list (along with the vertices themselves) will not be plotted.

Details

Only active CADJ edges are plotted (i.e., those with `TRN$CADJ_active = 1`), so that any previously pruned CADJ edges are not shown. If all edges are desired, called `TRN$restore_CADJ_edges` prior to plotting.

`vis_CADJvis_stats` *Visualize CADJ Statistics*

Description

The CADJvis statistics will be plotted by each local rank, global rank, or output topology distance, as requested.

Usage

```
vis_CADJvis_stats(TRN, group_by = "lrank")
```

Arguments

`TRN`	a TRN object
`group_by`	which grouping to visualize, default = 'lrank'. Can also be 'grank' or 'length'

`vis_CADJ_TopoFunctions` *Visualize the Topographic Functions for CADJ*

Description

Visualize the Topographic Functions for CADJ

Usage

```
vis_CADJ_TopoFunctions(TRN)
```

Arguments

`TRN`	a TRN object

Details

Three subplots are returned, corresponding to the Topographic Function, the Discrete Topographic Function, and the Weighted Discrete Topographic Function, respectively. See `?Topographic_Functions` for further details.

vis_CONNvis *CONNvis Visualization*

Description

CONNvis Visualization

Usage

```
vis_CONNvis(
  TRN,
  add = F,
  vertex.xy,
  vertex.pch = 16,
  vertex.cex = 1,
  vertex.col = "black",
  edge.lwd_range = NULL,
  vertex.active = T,
  vertex.subset = NULL
)
```

Arguments

TRN	a TRN object
add	whether to create a new plotting device (=FALSE, default), or add to an existing one (=TRUE)
vertex.xy	a matrix of (x,y) coordinates (nrows = TRN$nV, ncols = 2) defining the plot coordinates of the vertices of the TRN. It is assumed that the i-th row of vertex.xy gives the coordinates for vertex i of the graph.
vertex.pch	the pch symbol plotted vertices, default = 16.
vertex.cex	the cex of plotted vertices, default = 1. Set = 0 to suppress vertex plotting.
vertex.col	the color of plotted vertices, default = 'black'
edge.lwd_range	the min/max range of the plotted line widths. Default = NULL means line widths inherit from the TRN$CONN_grank of each edge. If supplied as something other than NULL, it must be a length=2 vector giving the (lower,upper) bounds of plotted edge widths.
vertex.active	whether to restrict plotted vertices to those which have an edge connecting them, default = TRUE.
vertex.subset	a vector of vertex indices to restrict the plotting to. Default = NULL, meaning all edges and vertices are plotted. If given, any edges connecting vertices in this list (along with the vertices themselves) will not be plotted.

Details

Only active CONN edges are plotted (i.e., those with TRN$CONN_active = 1), so that any previously pruned CONN edges are not shown. If all edges are desired, called TRN$restore_CONN_edges prior to plotting.

`vis_CONNvis_stats` *Visualize CONN Statistics*

Description

The CONNvis statistics will be plotted by each local rank, global rank, or output topology distance, as requested.

Usage

```
vis_CONNvis_stats(TRN, group_by = "lrank")
```

Arguments

`TRN`	a TRN object
`group_by`	which grouping to visualize, default = 'lrank'. Can also be 'grank' or 'length'

`vis_CONN_TopoFunctions` *Visualize the Topographic Functions for CONN*

Description

Visualize the Topographic Functions for CONN

Usage

```
vis_CONN_TopoFunctions(TRN)
```

Arguments

`TRN`	a TRN object

Details

Three subplots are returned, corresponding to the Topographic Function, the Discrete Topographic Function, and the Weighted Discrete Topographic Function, respectively. See `?Topographic_Functions` for further details.

vis_DMPrune_LambdaPath *Visualize the DM-Prune Lambda Path*

Description

Visualize the DM-Prune Lambda Path

Usage

```
vis_DMPrune_LambdaPath(
  TRN,
  min_step = NULL,
  max_step = NULL,
  scale = 1,
  plot.cpts = T
)
```

Arguments

TRN	a TRN object
min_step	the minimum pruning step to show
max_step	the maximum pruning step to show
scale	a scale factor controlling the size of plotted lines, points, and text labels. Higher values increase the size.
plot.cpts	whether to compute and plot the variance changepoints of the second-order differenced Lambda path. Default = TRUE.

Value

None, a plot is produced.

vis_DMPrune_prior *Visualize the DM-Prune prior*

Description

Plots the CADJ prior values vs. the CADJ values for comparison.

Usage

```
vis_DMPrune_prior(TRN)
```

Arguments

TRN	a TRN object

Value

None, a plot is produced.

`vis_TopoView` *TopoView Visualization*

Description

TopoView Visualization

Usage

```
vis_TopoView(
  ADJ,
  add = F,
  vertex.xy,
  vertex.pch = 16,
  vertex.cex = 1,
  vertex.col = "black",
  edge.col = "darkorange",
  edge.lwd_range = c(1, 5),
  vertex.active = T,
  vertex.subset = NULL
)
```

Arguments

`ADJ`	a (possibly weighted) adjacency matrix of a Topology Representing Network.
`add`	whether to create a new plotting device (=FALSE, default), or add to an existing one (=TRUE)
`vertex.xy`	a matrix of (x,y) coordinates (nrows = TRN$nV, ncols = 2) defining the plot coordinates of the vertices of the TRN. It is assumed that the `i-th` row of `vertex.xy` gives the coordinates for vertex `i` of the graph.
`vertex.pch`	the pch symbol plotted vertices, default = 16.
`vertex.cex`	the cex of plotted vertices, default = 1. Set = 0 to suppress vertex plotting.
`vertex.col`	the color of plotted vertices.
`edge.lwd_range`	the min/max range of the plotted line widths, Default = c(1, 5). This parameter is only valid if input `ADJ` is weighted, in which case the line widths represent the edge weights via a linear scaling from `(min weight,max weight)` to `edge.lwd_range`. If `ADJ` is unweighted, the plotted edges have width = `max(edge.lwd_range)`.
`vertex.active`	whether to restrict plotted vertices to those which have an edge connecting them, default = TRUE.
`vertex.subset`	a vector of vertex indices to restrict the plotting to. Default = NULL, meaning all edges and vertices are plotted. If given, any edges connected vertices in this list (along with the vertices themselves) will not be plotted.
`edge.color`	line color of plotted edges, default = "darkorange".

Details

It is assumed that any 0 value in the input ADJ_{ij} means no edge connects vertices i and j.

Index