Package 'tna'

May 8, 2026

Title Transition Network Analysis (TNA)

Version 1.2.3

Description Provides tools for performing Transition Network Analysis (TNA) to study relational dynamics, including functions for building and plotting TNA models, calculating centrality measures, and identifying dominant events and patterns. TNA statistical techniques (e.g., bootstrapping and permutation tests) ensure the reliability of observed insights and confirm that identified dynamics are meaningful. See (Saqr et al., 2025) <doi:10.1145/3706468.3706513> for more details on TNA.

License MIT + file LICENSE

URL https://github.com/sonsoleslp/tna/, https://sonsoles.me/tna/

BugReports https://github.com/sonsoleslp/tna/issues/

Depends R (>= 4.1.0)

Imports checkmate, cli, cluster, cograph, colorspace, dplyr, ggplot2, graphics, igraph, RColorBrewer, rlang, stats, tibble, tidyr, tidyselect

Suggests gt, knitr, pracma, rmarkdown, seqHMM, stringdist, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

LazyData true

Config/Needs/website rmarkdown

NeedsCompilation no

Author Mohammed Saqr [aut],
Santtu Tikka [aut],
Sonsoles López-Pernas [aut, cre, cph]

Maintainer Sonsoles López-Pernas <sonsoles.lopez@uef.fi>

Repository CRAN

Date/Publication 2026-04-27 00:00:03 UTC

Contents

`tna-package` *The* `tna` *Package.*

Description

Provides tools for performing transition network analysis (TNA), including functions for building TNA models, plotting transition networks, and calculating centrality measures. The package relies on the `igraph` for centrality measure calculations.

Author(s)

Sonsoles López-Pernas, Santtu Tikka, Mohammed Saqr

References

Saqr M., López-Pernas S., Törmänen T., Kaliisa R., Misiejuk K., Tikka S. (2025). Transition Network Analysis: A Novel Framework for Modeling, Visualizing, and Identifying the Temporal Patterns of Learners and Learning Processes. In *Proceedings of the 15th International Learning Analytics and Knowledge Conference (LAK '25)*, 351-361.

Banerjee A., Chandrasekhar A., Duflo E., Jackson M. (2014). Gossip: Identifying Central Individuals in a Social Network. Working Paper.

Kivimaki, I., Lebichot, B., Saramaki, J., Saerens, M. (2016). Two betweenness centrality measures based on Randomized Shortest Paths. *Scientific Reports*, 6, 19668.

Serrano, M. A., Boguna, M., Vespignani, A. (2009). Extracting the multiscale backbone of complex weighted networks. *Proceedings of the National Academy of Sciences*, 106, 6483-6488.

Zhang, B., Horvath, S. (2005). A general framework for weighted gene co-expression network analysis. *Statistical Applications in Genetics and Molecular Biology*, 4(1).

See Also

Useful links:

- https://github.com/sonsoleslp/tna/
- https://sonsoles.me/tna/
- Report bugs at https://github.com/sonsoleslp/tna/issues/

Basic functions `build_model()`, `hist.group_tna()`, `hist.tna()`, `plot.group_tna()`, `plot.tna()`, `plot_frequencies()`, `plot_frequencies.group_tna()`, `plot_mosaic()`, `plot_mosaic.group_tna()`, `plot_mosaic.tna_data()`, `print.group_tna()`, `print.summary.group_tna()`, `print.summary.tna()`, `print.tna()`, `summary.group_tna()`, `summary.tna()`

as.igraph.group_tna *Coerce a Specific Group from a* group_tna *Object into an* igraph *Object.*

Description

Coerce a Specific Group from a group_tna Object into an igraph Object.

Usage

```
## S3 method for class 'group_tna'
as.igraph(x, which, ...)
```

Arguments

x	The object to convert.
which	The number or name of the group.
...	Additional arguments. None currently.

Value

An igraph object.

See Also

Helper functions as.igraph.matrix(), as.igraph.tna()

as.igraph.matrix *Coerce a Weight Matrix into an* igraph *Object.*

Description

Coerce a Weight Matrix into an igraph Object.

Usage

```
## S3 method for class 'matrix'
as.igraph(x, mode = "directed", ...)
```

Arguments

x	A matrix of edge weights.
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. See details below.
...	Ignored.

Value

An igraph object.

See Also

Helper functions as.igraph.group_tna(), as.igraph.tna()

---

as.igraph.tna *Coerce a* tna *Object into an* igraph *Object.*

Description

Coerce a tna Object into an igraph Object.

Usage

```
## S3 method for class 'tna'
as.igraph(x, mode = "directed", ...)
```

Arguments

x	A tna object.
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. See details below.
...	Ignored.

Value

An igraph object.

See Also

Helper functions as.igraph.group_tna(), as.igraph.matrix()

betweenness_network *Build and Visualize a Network with Edge Betweenness*

Description

This function builds a network from a transition matrix in a tna object and computes edge betweenness for the network.

Usage

```
betweenness_network(x, directed = TRUE, invert = TRUE)

## S3 method for class 'tna'
betweenness_network(x, directed = TRUE, invert = TRUE)
```

Arguments

x	A tna object.
directed	A logical value. If TRUE, the network is considered directed.
invert	A logical value indicating whether the weights should be inverted for distance-based measures. The default is TRUE.

Value

A tna object where the edge weights are edge betweenness values.

See Also

Centrality measure functions centralities(), plot.group_tna_centralities(), plot.tna_centralities(), print.group_tna_centralities(), print.tna_centralities()

Examples

```
model <- tna(group_regulation)
betweenness_network(model)
```

bootstrap *Bootstrap Transition Networks from Sequence Data*

Description

Perform bootstrapping on transition networks created from sequence data stored in a tna object. Bootstrapped estimates of edge weights are returned with confidence intervals and significance testing.

Usage

```
bootstrap(x, iter, level, method, threshold, consistency_range)

## S3 method for class 'tna'
bootstrap(
  x,
  iter = 1000,
  level = 0.05,
  method = "stability",
  threshold,
  consistency_range = c(0.75, 1.25)
)

## S3 method for class 'group_tna'
bootstrap(
  x,
  iter = 1000,
  level = 0.05,
  method = "stability",
  threshold,
  consistency_range = c(0.75, 1.25)
)
```

Arguments

x	A tna or a group_tna object created from sequence data.
iter	An integer specifying the number of bootstrap samples to draw. Defaults to 1000.
level	A numeric value representing the significance level for hypothesis testing and confidence intervals. Defaults to 0.05.
method	A character string. This argument defines the bootstrap test statistic. The "stability" option (the default) compares edge weights against a range of "consistent" values defined by consistency_range. Weights that fall outside this range are considered insignificant. In other words, an edge is considered significant if its value is within the range in (1 - level) * 100% of the bootstrap samples. The "threshold" option instead compares the edge weights against a user-specified threshold value.

threshold A numeric value to compare edge weights against. The default is the 10th percentile of the edge weights. Used only when method = "threshold".

consistency_range A numeric vector of length 2. Determines how much the edge weights may deviate (multiplicatively) from their observed values (below and above) before they are considered insignificant. The default is c(0.75, 1.25) which corresponds to a symmetric 25% deviation range. Used only when method = "stability".

Details

The function first computes the original edge weights for the specified cluster from the tna object. It then performs bootstrapping by resampling the sequence data and recalculating the edge weights for each bootstrap sample. The mean and standard deviation of the transitions are computed, and confidence intervals are derived. The function also estimates p-values for each edge and identifies significant edges based on the specified significance level. A matrix of significant edges (those with estimated p-values below the significance level) is generated. Additional statistics on removed edges (those not considered significant) are provided.

All results, including the original transition matrix, bootstrapped estimates, and summary statistics for removed edges, are returned in a structured list.

Value

A tna_bootstrap object which is a list containing the following elements:

- weights_orig: The original edge weight matrix.
- weights_sig: The matrix of significant transitions (those with estimated p-values below the significance level).
- weights_mean: The mean weight matrix from the bootstrap samples.
- weights_sd: The standard deviation matrix from the bootstrap samples.
- cr_lower: The lower bound matrix of the consistency range for the edge weights.
- cr_upper: The upper bound matrix of the consistency range for the edge weights.
- ci_lower: The lower bound matrix of the bootstrap confidence intervals for the edge weights.
- ci_upper: The upper bound matrix of the bootstrap confidence intervals for the edge weights.
- p_values: The matrix of estimated p-values for the edge weights.
- summary: A data.frame summarizing the edges, their weights, p-values, statistical significance, consistency ranges, and confidence intervals.

If x is a group_tna object, the output is a group_tna_bootstrap object, which is a list of tna_bootstrap objects.

See Also

Validation functions deprune(), estimate_cs(), permutation_test(), permutation_test.group_tna(), plot.group_tna_bootstrap(), plot.group_tna_permutation(), plot.group_tna_stability(), plot.tna_bootstrap(), plot.tna_permutation(), plot.tna_reliability(), plot.tna_stability(), print.group_tna_bootstrap(), print.group_tna_permutation(), print.group_tna_stability(),

print.summary.group_tna_bootstrap(), print.summary.tna_bootstrap(), print.tna_bootstrap(), print.tna_clustering(), print.tna_permutation(), print.tna_reliability(), print.tna_stability(), prune(), pruning_details(), reliability(), reprune(), summary.group_tna_bootstrap(), summary.tna_bootstrap()

Examples

```
model <- tna(group_regulation)
# Small number of iterations for CRAN
bootstrap(model, iter = 10)
```

`bootstrap_cliques` *Bootstrap Cliques of Transition Networks from Sequence Data*

Description

Bootstrap the edge weights of all cliques of a given size in a `tna` model, producing per-clique mean weights, p-values, confidence intervals, and consistency-range bounds.

Usage

```
bootstrap_cliques(x, size, threshold, iter, level, consistency_range)

## S3 method for class 'tna'
bootstrap_cliques(
  x,
  size = 2L,
  threshold = 0,
  iter = 1000,
  level = 0.05,
  consistency_range = c(0.75, 1.25)
)
```

Arguments

`x`	A `tna` or a `group_tna` object.
`size`	An `integer` specifying the size of the cliques to identify. Defaults to `2` (dyads).
`threshold`	A `numeric` value that sets the minimum edge weight for an edge to be considered in the clique. Edges below this value are ignored. Defaults to `0`.
`iter`	An `integer` specifying the number of bootstrap samples to draw. Defaults to `1000`.
`level`	A `numeric` value representing the significance level for hypothesis testing and confidence intervals. Defaults to `0.05`.

consistency_range
: A numeric vector of length 2. Determines how much the edge weights may deviate (multiplicatively) from their observed values (below and above) before they are considered insignificant. The default is c(0.75, 1.25) which corresponds to a symmetric 25% deviation range. Used only when method = "stability".

Value

A data.frame (also of class tna_bootstrap_cliques) with one row per clique and the columns clique, mean_weight, p_values, sig, cr_lower, cr_upper, ci_lower, ci_upper.

Examples

```
model <- tna(group_regulation)
# Small number of iterations for CRAN
boot_cliq <- bootstrap_cliques(model, size = 2, iter = 10)
```

---

build_model *Build a Transition Network Analysis Model*

Description

Construct a transition network analysis (TNA) model from sequence data. The function takes a data set of sequence of events or states as input and builds a TNA model. It extracts the edge weights and initial probabilities from the data along with the state labels. The function also accepts weight matrices and initial state probabilities directly.

Usage

```
build_model(x, ...)

## Default S3 method:
build_model(
  x,
  type = "relative",
  scaling = character(0L),
  params = list(),
  inits,
  ...
)

## S3 method for class 'matrix'
build_model(
  x,
  type = "relative",
  scaling = character(0L),
```

```
  params = list(),
  inits,
  ...
)

## S3 method for class 'stslist'
build_model(
  x,
  type = "relative",
  scaling = character(0L),
  cols = tidyselect::everything(),
  params = list(),
  concat = 1L,
  begin_state,
  end_state,
  ...
)

## S3 method for class 'data.frame'
build_model(
  x,
  type = "relative",
  scaling = character(0L),
  cols = tidyselect::everything(),
  concat = 1L,
  params = list(),
  begin_state,
  end_state,
  ...
)

## S3 method for class 'tna_data'
build_model(
  x,
  type = "relative",
  scaling = character(0L),
  params = list(),
  concat = 1L,
  begin_state,
  end_state,
  ...
)

## S3 method for class 'tsn'
build_model(
  x,
  type = "relative",
  scaling = character(0L),
```

```
  params = list(),
  concat = 1L,
  begin_state,
  end_state,
  ...
)

tna(x, ...)

ftna(x, ...)

ctna(x, ...)

atna(x, ...)

tsn(x, ...)
```

Arguments

x	A stslist (from TraMineR), data.frame, a matrix, or a tna_data object (see prepare_data()). For stslist and data.frame objects x should describe a sequence of events or states to be used for building the Markov model. If x is a matrix, it is assumed that the element on row i and column j is the weight of the edge representing the transition from state i to state j. If x is a data.frame, then it must be in wide format (see cols on how to define columns for the time points).
...	Ignored. For the build_model aliases (e.g., tna), this argument matches the actual arguments to build_model beside x.
type	A character string describing the weight matrix type. Currently supports the following types: • "relative" for relative frequencies (probabilities, the default) • "frequency" for frequencies. • "co-occurrence" for co-occurrences. • "n-gram" for n-gram transitions. Captures higher-order transitions by considering sequences of n states, useful for identifying longer patterns. • "gap" allows transitions between non-adjacent states, with transitions weighted by the gap size. • "reverse" considers the sequences in reverse order (resulting in what is called a reply network in some contexts). The resulting weight matrix is the transpose of the "frequency" option. • "attention" aggregates all downstream pairs of states with an exponential decay for the gap between states. The parameter lambda can be used to control the decay rate (the default is 1)-
scaling	A character vector describing how to scale the weights defined by type. When a vector is provided, the scaling options are applied in the respective order. For example, c("rank", "minmax") would first compute the ranks, then scale them

to the unit interval using min-max normalization. An empty vector corresponds to no scaling. Currently supports the following options:

- `"minmax"` performs min-max normalization to scale the weights to the unit interval. Note that if the smallest weight is positive, it will be zero after scaling.
- `"max"` Multiplies the weights by the reciprocal of the largest weight to scale the weights to the unit interval. This options preserves positive ranks, unlike `"minmax"` when all weights are positive.
- `"rank"` Computes the ranks of the weights using `base::rank()` with `ties.method = "average"`.

`params` A `list` of additional arguments for models of specific `type`. The potential elements of this list are:

- `n_gram`: An `integer` for n-gram transitions specifying the number of adjacent events. The default value is 2.
- `max_gap`: An `integer` for the gap-allowed transitions specifying the largest allowed gap size. The default is 1.
- `windowed`: Perform the model estimation by window. Supported for `relative`, `frequency` and `co-occurrence` types.
- `window_size`: An `integer` for the sliding window transitions specifying the window size. The default is 2.
- `window_type`: A `character` string that defines the window type. Either `"rolling"` or `"tumbling"`.
- `weighted`: A `logical` value. If `TRUE`, the transitions are weighted by the inverse of the sequence length. Can be used for frequency, co-occurrence and reverse model types. The default is `FALSE`.
- `direction`: A `character` string specifying the direction of attention for models of `type = "attention"`. The available options are `"backward"`, `"forward"`, and `"both"`, for backward attention, forward attention, and bidirectional attention, respectively. The default is `"forward"`.
- `decay`: A `function` that specifies the decay of the weights between two time points at a specific distance. The function should take three arguments: `i`, `j` and `lambda`, where `i` and `j` are `numeric` vectors of time values, and `lambda` is a `numeric` value for the decay rate. The function should return a `numeric` vector of weights. The default is `function(i, j, lambda) exp(-abs(i - j) / lambda)`.
- `lambda`: A `numeric` value for the decay rate. The default is 1.
- `time`: A `matrix` or a `data.frame` providing the time values for each sequence and at time index. For `tna_data` objects, this can also be a logical value, where `TRUE` will use the `time_data` element of `x` for the time values. `Date` values are converted to `numeric`.
- `duration`: A `matrix` or a `data.frame` providing the time spent in each state for each sequence and time index. This is an alternative to `time`.

`inits` An optional `numeric` vector of initial state probabilities for each state. The vector will be scaled to unity.

`cols` An `expression` giving a tidy selection of columns that should be considered as sequence data. By default, all columns are used.

concat	An integer for the number of consecutive sequences to concatenate. The default is 1 (no concatenation).
begin_state	A character string for an additional begin state. This state is added as the first observation for every sequence to signify the beginning of the sequence
end_state	A character string for an additional end state. This state is added as the last observation for every sequence to signify the end of the sequence.

Value

An object of class tna which is a list containing the following elements:

- weights: An adjacency matrix of the model (weight matrix).
- inits: A numeric vector of initial values for each state. For matrix type x, this element will be NULL if inits is not directly provided
- labels: A character vector of the state labels, or NULL if there are no labels.
- data: The original sequence data that has been converted to an internal format used by the package when x is a stslist or a data.frame object. Otherwise NULL.

See Also

Basic functions hist.group_tna(), hist.tna(), plot.group_tna(), plot.tna(), plot_frequencies(), plot_frequencies.group_tna(), plot_mosaic(), plot_mosaic.group_tna(), plot_mosaic.tna_data(), print.group_tna(), print.summary.group_tna(), print.summary.tna(), print.tna(), summary.group_tna(), summary.tna(), tna-package

Examples

```
model <- build_model(group_regulation)
print(model)

model <- tna(group_regulation)

model <- ftna(group_regulation)

model <- ctna(group_regulation)

model <- atna(group_regulation)
```

---

centralities *Calculate Centrality Measures for a Transition Matrix*

Description

Calculates several centrality measures. See 'Details' for information about the measures.

Usage

```
centralities(x, loops = FALSE, normalize = FALSE, invert = TRUE, measures)

## S3 method for class 'tna'
centralities(x, loops = FALSE, normalize = FALSE, invert = TRUE, measures)

## S3 method for class 'matrix'
centralities(x, loops = FALSE, normalize = FALSE, invert = TRUE, measures)

## S3 method for class 'group_tna'
centralities(x, loops = FALSE, normalize = FALSE, invert = TRUE, measures)
```

Arguments

x	A tna object, a group_tna object, or a square matrix representing edge weights.
loops	A logical value indicating whether to include loops in the network when computing the centrality measures. The default is FALSE.
normalize	A logical value indicating whether the centralities should be normalized. The default is FALSE.
invert	A logical value indicating whether the weights should be inverted for distance-based measures. The default is TRUE.
measures	A character vector indicating which centrality measures should be computed. If missing, all available measures are returned. See 'Details' for the available measures.

Details

The following measures are provided:

- OutStrength: Outgoing strength centrality, calculated using igraph::strength() with mode = "out". It measures the total weight of the outgoing edges from each node.
- InStrength: Incoming strength centrality, calculated using igraph::strength() with mode = "in". It measures the total weight of the incoming edges to each node.
- ClosenessIn: Closeness centrality (incoming), calculated using igraph::closeness() with mode = "in". It measures how close a node is to all other nodes based on the incoming paths.
- ClosenessOut: Closeness centrality (outgoing), calculated using igraph::closeness() with mode = "out". It measures how close a node is to all other nodes based on the outgoing paths.
- Closeness: Closeness centrality (overall), calculated using igraph::closeness() with mode = "all". It measures how close a node is to all other nodes based on both incoming and outgoing paths.
- Betweenness: Betweenness centrality defined by the number of geodesics calculated using igraph::betweenness().
- BetweennessRSP: Betweenness centrality based on randomized shortest paths (Kivimäki et al. 2016). It measures the extent to which a node lies on the shortest paths between other nodes.

- `Diffusion`: Diffusion centrality of Banerjee et.al. (2014). It measures the influence of a node in spreading information through the network.
- `Clustering`: Signed clustering coefficient of Zhang and Horvath (2005) based on the symmetric adjacency matrix (sum of the adjacency matrix and its transpose). It measures the degree to which nodes tend to cluster together.

Value

A `tna_centralities` object which is a tibble (`tbl_df`). containing centrality measures for each state.

See Also

Centrality measure functions `betweenness_network()`, `plot.group_tna_centralities()`, `plot.tna_centralities()`, `print.group_tna_centralities()`, `print.tna_centralities()`

Examples

```
model <- tna(group_regulation)

# Centrality measures including loops in the network
centralities(model)

# Centrality measures excluding loops in the network
centralities(model, loops = FALSE)

# Centrality measures normalized
centralities(model, normalize = TRUE)
```

---

cliques *Identify Cliques in a Transition Network*

Description

This function identifies cliques of a specified size in a transition network. It searches for cliques, i.e., complete subgraphs where every pair of nodes is connected, of size `n` in the transition matrix for the specified cluster in the `tna` object.

Usage

```
cliques(x, ...)

## S3 method for class 'tna'
cliques(x, size = 2, threshold = 0, sum_weights = FALSE, ...)

## S3 method for class 'group_tna'
cliques(x, size = 2, threshold = 0, sum_weights = FALSE, ...)
```

Arguments

x	A tna or a group_tna object.
...	Ignored.
size	An integer specifying the size of the cliques to identify. Defaults to 2 (dyads).
threshold	A numeric value that sets the minimum edge weight for an edge to be considered in the clique. Edges below this value are ignored. Defaults to 0.
sum_weights	A logical value specifying whether the sum of the weights should be above the threshold instead of individual weights of the directed edges. Defaults to FALSE.

Value

A tna_cliques object which is a list of two elements:

- weights is a matrix of the edge weights in the clique.
- inits is a numeric vector of initial weights for the clique.

If x is a group_tna object, a group_tna_cliques object is returned instead, which is a list or tna_cliques objects.

See Also

Clique-related functions plot.group_tna_cliques(), plot.tna_cliques(), print.group_tna_cliques(), print.tna_cliques()

Examples

```
model <- tna(group_regulation)

# Find  2-cliques (dyads)
cliq <- cliques(model, size = 2)

model <- group_tna(engagement_mmm)
cliques(model)
```

---

cluster_data *Clustering via Dissimilarity Matrix based on String Distances*

Description

Performs clustering using specified dissimilarity measures and clustering methods. The rows of the data are first converted to strings and compared using the dissimilarity measures available in the stringdist package.

Usage

```
cluster_data(
  data,
  k,
  dissimilarity = "hamming",
  method = "pam",
  na_syms = c("*", "%"),
  weighted = FALSE,
  lambda = 1,
  ...
)

cluster_sequences(
  data,
  k,
  dissimilarity = "hamming",
  method = "pam",
  na_syms = c("*", "%"),
  weighted = FALSE,
  lambda = 1,
  ...
)
```

Arguments

data	A data.frame, a matrix in wide format, or a tna_data object (in which case, the sequence data is extracted automatically).
k	An integer giving the number of clusters.
dissimilarity	A character string specifying the dissimilarity measure. The available options are: "osa", "lv", "dl", "hamming", "lcs", "qgram", "cosine", "jaccard", and "jw". See stringdist::stringdist-metrics for more information on these measures.
method	A character string specifying clustering method. The available methods are "pam", "ward.D", "ward.D2", "complete","average", "single", "mcquitty", "median", and "centroid". See cluster::pam() and stats::hclust() for more information on these methods.
na_syms	A character vector of symbols or factor levels to convert to explicit missing values.
weighted	A logical value indicating whether the dissimilarity measure should be weighted (the default is FALSE for no weighting). If TRUE, earlier observations of the sequences receive a greater weight in the distance calculation with an exponential decay. Currently only supported for the Hamming distance.
lambda	A numeric value defining the strength of the decay when weighted = TRUE. The default is 1.0.
...	Additional arguments passed to stringdist::stringdist().

Value

A tna_clustering object which is a list containing:

- data: The original data.
- k: The number of clusters.
- assignments: An integer vector of cluster assignments.
- silhouette: Silhouette score measuring clustering quality.
- sizes: An integer vector of cluster sizes.
- method: The clustering method used.
- distance: The distance matrix.

Examples

```
data <- data.frame(
  T1 = c("A", "B", "A", "C", "A", "B"),
  T2 = c("B", "A", "B", "A", "C", "A"),
  T3 = c("C", "C", "A", "B", "B", "C")
)

# PAM clustering with optimal string alignment (default)
result <- cluster_sequences(data, k = 2)
```

---

communities *Community Detection for Transition Networks*

Description

This function detects communities within the transition networks (represented by the tna object). It uses various algorithms to find communities in the graph representation of transitions and returns a list of communities for each cluster or a specified cluster. If multiple transition matrices exist, the function iterates over each cluster in the tna object to find communities using different algorithms. The function uses the igraph package to convert the transition matrices into graphs and then applies community detection algorithms (e.g., Walktrap, Fast Greedy, Label Propagation, Infomap, Edge Betweenness, Leading Eigenvector, and Spin Glass).

Usage

```
communities(x, methods, gamma)

## S3 method for class 'tna'
communities(x, methods, gamma = 1)

## S3 method for class 'group_tna'
communities(x, methods, gamma = 1)
```

Arguments

x	A tna or a group_tna object.
methods	A character vector of community detection algorithms to apply to the network. The supported options are: • "walktrap": A community detection method using short random walks. • "fast_greedy": A method based on modularity optimization. • "label_prop": A method that uses label propagation. • "infomap": A method that uses information flow to detect communities. • "edge_betweenness": A method that uses edge betweenness to find communities. • "leading_eigen": A method using the leading eigenvector of the modularity matrix. • "spinglass": A method based on the spinglass model. If not provided, all methods are applied.
gamma	A numeric value depicting a parameter that affects the behavior of certain algorithms like the Spin Glass method. Defaults to 1.

Value

An object of class tna_communities which is a list with an element for each cluster containing:

- counts: A list with the number of communities found by each algorithm.
- assignments: A data.frame where each row corresponds to a node and each column to a community detection algorithm, with color-coded community assignments.

If x is a group_tna object, a group_tna_communities object is returned instead, which is a list of tna_communities objects.

See Also

Community detection functions plot.group_tna_communities(), plot.tna_communities(), print.group_tna_commun
print.tna_communities()

Cluster-related functions group_model(), mmm_stats(), rename_groups()

Examples

```
model <- tna(group_regulation)
comm <- communities(model)
```

compare *Compare Two Matrices or TNA Models with Comprehensive Metrics*

Description

Various distances, measures of dissimilarity and similarity, correlations and other metrics are computed to compare the models. Optionally, the weight matrices of the models can be scaled before comparison. The resulting object can be used to produce heatmap plots and scatterplots to further illustrate the differences.

Usage

```
compare(x, ...)

## S3 method for class 'tna'
compare(x, y, scaling = "none", measures = character(0), network = TRUE, ...)

## S3 method for class 'matrix'
compare(x, y, scaling = "none", measures = character(0), network = TRUE, ...)
```

Arguments

x	A tna object or a matrix of weights.
...	Ignored.
y	A tna object or a matrix of weights.
scaling	A character string naming a scaling method to apply to the weights before comparing them. The supported options are: • "none": No scaling is performed. The weights are used as is. • "minmax": Performs min-max normalization, i.e., the minimum value is subtracted and the differences are scaled by the range. • "max": Max-normalization: the values are divided by the maximum value. • "rank": Applies min-max normalization to the ranks of the weights (computed with ties.method = "average"). • "zscore": Computes the standard score, i.e. the mean weight is subtracted and the differences are scaled by the standard deviation. • "robust": Computes the robust z-score, i.e. the median weight is subtracted and the differences are scaled by the median absolute deviation (using stats::mad). • "log": Simply the natural logarithm of the weights. • "log1p": As above, but adds 1 to the values before taking the logarithm. Useful for scenarios with zero weights. • "softmax": Performs softmax normalization. • "quantile": Uses the empirical quantiles of the weights via stats::ecdf.

measures — A character vector indicating which centrality measures should be computed. See centralities() for the available measures. No measures are included by default.

network — A logical value indicating whether network metrics should be included in the comparison. The default is TRUE.

Value

A tna_comparison object, which is a list containing the following elements:

- matrices: A list containing the scaled matrices of the input tna objects or the scaled inputs themselves in the case of matrices.
- difference_matrix: A matrix of differences x - y.
- edge_metrics: A data.frame of edge-level metrics about the differences.
- summary_metrics: A data.frame of summary metrics of the differences across all edges.
- network_metrics: A data.frame of network metrics for both x and y.
- centrality_differences: A data.frame of differences in centrality measures computes from x and y.
- centrality_correlations: A numeric vector of correlations of the centrality measures between x and y.

See Also

Model comparison functions compare.group_tna(), compare_sequences(), plot.tna_comparison(), plot.tna_sequence_comparison(), plot_compare(), plot_compare.group_tna(), print.tna_comparison(), print.tna_sequence_comparison()

Examples

```
# Comparing TNA models
model_x <- tna(group_regulation[1:200, ])
model_y <- tna(group_regulation[1001:1200, ])
comp1 <- compare(model_x, model_y)

# Comparing matrices
mat_x <- model_x$weights
mat_y <- model_y$weights
comp2 <- compare(mat_x, mat_y)

# Comparing a matrix to a TNA model
comp3 <- compare(mat_x, model_y)
```

compare.group_tna *Compare Grouped TNA Models with Comprehensive Metrics*

Description

Compare Grouped TNA Models with Comprehensive Metrics

Usage

```
## S3 method for class 'group_tna'
compare(
  x,
  i = 1L,
  j = 2L,
  scaling = "none",
  measures = character(0),
  network = TRUE,
  ...
)
```

Arguments

x	A group_tna object.
i	An integer index or the name of the principal cluster as a character string.
j	An integer index or the name of the secondary cluster as a character string.
scaling	A character string naming a scaling method to apply to the weights before comparing them. The supported options are: • "none": No scaling is performed. The weights are used as is. • "minmax": Performs min-max normalization, i.e., the minimum value is subtracted and the differences are scaled by the range. • "max": Max-normalization: the values are divided by the maximum value. • "rank": Applies min-max normalization to the ranks of the weights (computed with ties.method = "average"). • "zscore": Computes the standard score, i.e. the mean weight is subtracted and the differences are scaled by the standard deviation. • "robust": Computes the robust z-score, i.e. the median weight is subtracted and the differences are scaled by the median absolute deviation (using stats::mad). • "log": Simply the natural logarithm of the weights. • "log1p": As above, but adds 1 to the values before taking the logarithm. Useful for scenarios with zero weights. • "softmax": Performs softmax normalization. • "quantile": Uses the empirical quantiles of the weights via stats::ecdf.

measures	A character vector indicating which centrality measures should be computed. See centralities() for the available measures. No measures are included by default.
network	A logical value indicating whether network metrics should be included in the comparison. The default is TRUE.
...	Additional arguments passed to compare.tna().

Value

A tna_comparison object. See compare.tna() for details.

See Also

Model comparison functions compare(), compare_sequences(), plot.tna_comparison(), plot.tna_sequence_compari
plot_compare(), plot_compare.group_tna(), print.tna_comparison(), print.tna_sequence_comparison()

Examples

```
model <- group_model(engagement_mmm)
compare(model, i = 1, j = 2)
```

---

compare_sequences *Compare Sequences Between Groups*

Description

Performs comprehensive sequence comparison analysis between groups. All patterns of the sequences (subsequences of specific length) are extracted from all sequences in each group. The pattern frequencies are compared between the groups using a permutation test. The reported effect size is the difference between the observed test statistic (sum of squared differences between the observed and expected counts) and the mean value over the permutation samples divided by their standard deviation times square root of the number of observations.

Usage

```
compare_sequences(x, ...)

## Default S3 method:
compare_sequences(
  x,
  group,
  sub,
  min_freq = 5L,
  test = TRUE,
  iter = 1000L,
  adjust = "bonferroni",
```

```
  ...
)

## S3 method for class 'group_tna'
compare_sequences(
  x,
  sub,
  min_freq = 5L,
  test = TRUE,
  iter = 1000L,
  adjust = "bonferroni",
  ...
)
```

Arguments

x	A group_tna object or a data.frame containing sequence data in wide format.
...	Not used.
group	A vector indicating the group assignment of each row of the data/sequence. Must have the same length as the number of rows/sequences of x. Alternatively, a single character string giving the column name of the data that defines the group when x is a wide format data.frame or a tna_data object.
sub	An integer vector of pattern lengths to analyze. The default is 2:5.
min_freq	An integer giving the minimum number of times that a specific pattern has to be observed in each group to be included in the analysis. The default is 5.
test	A logical value indicating whether to test the differences of pattern counts between the groups using a permutation test. The default is TRUE.
iter	An integer giving the number of iterations for the permutation test. The default is 1000.
adjust	A character string naming the multiple comparison correction method (default: "bonferroni"). Supports all stats::p.adjust methods: "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none". The adjustment is carried out within sequences of the same length.

Value

A tna_sequence_comparison object, which is a data.frame with columns giving the names of the patterns, pattern frequencies, pattern proportions (within patterns of the same length), effect sizes, and p-values of the tests.

See Also

Model comparison functions compare(), compare.group_tna(), plot.tna_comparison(), plot.tna_sequence_compar
plot_compare(), plot_compare.group_tna(), print.tna_comparison(), print.tna_sequence_comparison()

Examples

```
# Use a subset to keep the example fast.
idx <- c(1:500, 1001:1500)
group <- c(rep("High", 500), rep("Low", 500))
comp <- compare_sequences(group_regulation[idx, ], group)

# With permutation test (small number of iterations for CRAN)
comp_test <- compare_sequences(
  group_regulation[idx, ],
  group,
  test = TRUE,
  iter = 5
)
```

---

deprune *Restore a Pruned Transition Network Analysis Model*

Description

Restore a Pruned Transition Network Analysis Model

Usage

```
deprune(x, ...)

## S3 method for class 'tna'
deprune(x, ...)

## S3 method for class 'tna'
reprune(x, ...)

## S3 method for class 'group_tna'
deprune(x, ...)
```

Arguments

x	A tna or group_tna object.
...	Ignored.

Value

A tna or group_tna object that has not been pruned.

See Also

Validation functions bootstrap(), estimate_cs(), permutation_test(), permutation_test.group_tna(), plot.group_tna_bootstrap(), plot.group_tna_permutation(), plot.group_tna_stability(), plot.tna_bootstrap(), plot.tna_permutation(), plot.tna_reliability(), plot.tna_stability(), print.group_tna_bootstrap(), print.group_tna_permutation(), print.group_tna_stability(), print.summary.group_tna_bootstrap(), print.summary.tna_bootstrap(), print.tna_bootstrap(), print.tna_clustering(), print.tna_permutation(), print.tna_reliability(), print.tna_stability(), prune(), pruning_details(), reliability(), reprune(), summary.group_tna_bootstrap(), summary.tna_bootstrap()

Examples

```
model <- tna(group_regulation)
pruned_model <- prune(model, method = "threshold", threshold = 0.1)
depruned_model <- deprune(pruned_model) # restore original model
```

---

engagement *Example Data on Student Engagement*

Description

Students' engagement states (Active / Average / Disengaged) throughout a whole study program. The data was generated synthetically based on the article "The longitudinal association between engagement and achievement varies by time, students' profiles, and achievement state: A full program study"

Usage

```
engagement
```

Format

A `stslist` object (sequence data).

Source

doi:10.1016/j.compedu.2023.104787

See Also

Datasets engagement_mmm, group_regulation, group_regulation_long

engagement_mmm *Example Mixture Markov Model Object for the* engagement *Data*

Description

A tna_mmm object structurally equivalent to a fitted seqHMM mixture Markov model, used by the package's examples for group_model() and mmm_stats(). Generated deterministically by random_tna_mmm() with engagement-flavored state labels and three clusters; carries no foreign class references so it is safe to bundle.

Usage

```
engagement_mmm
```

Format

A tna_mmm object containing three clusters over the states "Engaged", "Moderate", "Disengaged".

Details

Re-generate via data-raw/engagement_mmm.R whenever the simulator changes.

Source

Generated by data-raw/engagement_mmm.R using random_tna_mmm().

See Also

Datasets engagement, group_regulation, group_regulation_long

estimate_cs *Estimate Centrality Stability*

Description

Estimates the stability of centrality measures in a network using subset sampling without replacement. It allows for dropping varying proportions of cases and calculates correlations between the original centralities and those computed using sampled subsets.

Usage

```
estimate_cs(
  x,
  loops,
  normalize,
  invert,
  measures,
  iter,
  method,
  drop_prop,
  threshold,
  certainty,
  progressbar = NULL,
  detailed = NULL
)

estimate_centrality_stability(
  x,
  loops,
  normalize,
  invert,
  measures,
  iter,
  method,
  drop_prop,
  threshold,
  certainty,
  progressbar = NULL,
  detailed = NULL
)

## S3 method for class 'tna'
estimate_cs(
  x,
  loops = FALSE,
  normalize = FALSE,
  invert = TRUE,
  measures = c("InStrength", "OutStrength", "Betweenness"),
  iter = 1000,
  method = "pearson",
  drop_prop = seq(0.1, 0.9, by = 0.1),
  threshold = 0.7,
  certainty = 0.95,
  progressbar = FALSE,
  detailed = NULL
)

## S3 method for class 'tna'
```

```
estimate_centrality_stability(
  x,
  loops = FALSE,
  normalize = FALSE,
  invert = TRUE,
  measures = c("InStrength", "OutStrength", "Betweenness"),
  iter = 1000,
  method = "pearson",
  drop_prop = seq(0.1, 0.9, by = 0.1),
  threshold = 0.7,
  certainty = 0.95,
  progressbar = FALSE,
  detailed = NULL
)

## S3 method for class 'group_tna'
estimate_cs(
  x,
  loops = FALSE,
  normalize = FALSE,
  invert = TRUE,
  measures = c("InStrength", "OutStrength", "Betweenness"),
  iter = 1000,
  method = "pearson",
  drop_prop = seq(0.1, 0.9, by = 0.1),
  threshold = 0.7,
  certainty = 0.95,
  progressbar = FALSE,
  detailed = NULL
)

## S3 method for class 'group_tna'
estimate_centrality_stability(
  x,
  loops = FALSE,
  normalize = FALSE,
  invert = TRUE,
  measures = c("InStrength", "OutStrength", "Betweenness"),
  iter = 1000,
  method = "pearson",
  drop_prop = seq(0.1, 0.9, by = 0.1),
  threshold = 0.7,
  certainty = 0.95,
  progressbar = FALSE,
  detailed = NULL
)
```

Arguments

x	A tna or a group_tna object representing the temporal network analysis data. The object should be created from a sequence data object.
loops	A logical value indicating whether to include loops in the network when computing the centrality measures. The default is FALSE.
normalize	A logical value indicating whether to normalize the centrality measures. The default is FALSE.
invert	A logical value indicating whether the weights should be inverted for distance-based measures. The default is TRUE.
measures	A character vector of centrality measures to estimate. The default measures are "InStrength", "OutStrength", and "Betweenness". See centralities() for a list of available centrality measures.
iter	An integer specifying the number of resamples to draw. The default is 1000.
method	A character string indicating the correlation coefficient type. The default is "pearson". See stats::cor() for details.
drop_prop	A numeric vector specifying the proportions of cases to drop in each sampling iteration. Default is a sequence from 0.1 to 0.9 in increments of 0.1.
threshold	A numeric value specifying the correlation threshold for calculating the CS-coefficient. The default is 0.7.
certainty	A numeric value specifying the desired level of certainty for the CS-coefficient. Default is 0.95.
progressbar	A logical value. If TRUE, a progress bar is displayed Defaults to FALSE
detailed	Deprecated. This argument is ignored and will be removed in a future version.

Details

The function works by repeatedly resampling the data, dropping varying proportions of cases, and calculating centrality measures on the subsets. The correlation between the original centralities and the resampled centralities is calculated for each drop proportion. The stability of each centrality measure is then summarized using a centrality stability (CS) coefficient, which represents the proportion of dropped cases at which the correlations drop below a given threshold (default 0.7).

The results can be visualized by plotting the output object showing the stability of the centrality measures across different drop proportions, along with confidence intervals. The CS-coefficients are displayed in the subtitle.

Value

A tna_stability object which is a list with an element for each measure with the following elements:

- cs_coefficient: The centrality stability (CS) coefficient of the measure.
- correlations: A matrix of correlations between the original centrality and the resampled centralities for each drop proportion.

If x is a group_tna object, a group_tna_stability object is returned instead, which is a list of tna_stability objects.

See Also

Validation functions bootstrap(), deprune(), permutation_test(), permutation_test.group_tna(), plot.group_tna_bootstrap(), plot.group_tna_permutation(), plot.group_tna_stability(), plot.tna_bootstrap(), plot.tna_permutation(), plot.tna_reliability(), plot.tna_stability(), print.group_tna_bootstrap(), print.group_tna_permutation(), print.group_tna_stability(), print.summary.group_tna_bootstrap(), print.summary.tna_bootstrap(), print.tna_bootstrap(), print.tna_clustering(), print.tna_permutation(), print.tna_reliability(), print.tna_stability(), prune(), pruning_details(), reliability(), reprune(), summary.group_tna_bootstrap(), summary.tna_bootstrap()

Examples

```
model <- tna(group_regulation)
# Small number of iterations and drop proportions for CRAN
estimate_cs(
  model,
  drop_prop = seq(0.3, 0.9, by = 0.2),
  measures = c("InStrength", "OutStrength"),
  iter = 10
)
```

---

group_model *Build a Grouped Transition Network Analysis Model*

Description

This function constructs a transition network analysis (TNA) model for each group from a given sequence, wide-format dataframe or a mixture Markov model.

Usage

```
group_model(x, ...)

## Default S3 method:
group_model(
  x,
  group,
  type = "relative",
  scaling = character(0L),
  groupwise = FALSE,
  cols = tidyselect::everything(),
  params = list(),
  concat = 1L,
  na.rm = TRUE,
  ...
```

```
)

## S3 method for class 'tna_mmm'
group_model(
  x,
  type = "relative",
  scaling = character(0L),
  groupwise = FALSE,
  params = list(),
  na.rm = TRUE,
  ...
)

## S3 method for class 'mhmm'
group_model(
  x,
  type = "relative",
  scaling = character(0L),
  groupwise = FALSE,
  params = list(),
  na.rm = TRUE,
  ...
)

## S3 method for class 'tna_clustering'
group_model(
  x,
  type = "relative",
  scaling = character(0L),
  groupwise = FALSE,
  params = list(),
  na.rm = TRUE,
  ...
)

group_tna(x, ...)

group_ftna(x, ...)

group_ctna(x, ...)

group_atna(x, ...)
```

Arguments

x	An `stslist` object describing a sequence of events or states to be used for building the Markov model. The argument `x` also accepts `data.frame` objects in wide format, and `tna_data` objects. This can also be the output of clustering from `cluster_sequences()`.

`...` Ignored.

`group` A `vector` indicating the group assignment of each row of the data/sequence. Must have the same length as the number of rows/sequences of `x`. Alternatively, a single `character` string giving the column name of the data that defines the group when `x` is a wide format `data.frame` or a `tna_data` object. If not provided, each row of the data forms a cluster. Not used when `x` is a mixture Markov model or a clustering result.

`type` A `character` string describing the weight matrix type. Currently supports the following types:

- `"relative"` for relative frequencies (probabilities, the default)
- `"frequency"` for frequencies.
- `"co-occurrence"` for co-occurrences.
- `"n-gram"` for n-gram transitions. Captures higher-order transitions by considering sequences of n states, useful for identifying longer patterns.
- `"gap"` allows transitions between non-adjacent states, with transitions weighted by the gap size.
- `"reverse"` considers the sequences in reverse order (resulting in what is called a reply network in some contexts). The resulting weight matrix is the transpose of the `"frequency"` option.
- `"attention"` aggregates all downstream pairs of states with an exponential decay for the gap between states. The parameter `lambda` can be used to control the decay rate (the default is 1)-

`scaling` A `character` vector describing how to scale the weights defined by `type`. When a vector is provided, the scaling options are applied in the respective order. For example, `c("rank", "minmax")` would first compute the ranks, then scale them to the unit interval using min-max normalization. An empty vector corresponds to no scaling. Currently supports the following options:

- `"minmax"` performs min-max normalization to scale the weights to the unit interval. Note that if the smallest weight is positive, it will be zero after scaling.
- `"max"` Multiplies the weights by the reciprocal of the largest weight to scale the weights to the unit interval. This options preserves positive ranks, unlike `"minmax"` when all weights are positive.
- `"rank"` Computes the ranks of the weights using `base::rank()` with `ties.method = "average"`.

`groupwise` A `logical` value that indicates whether scaling methods should be applied by group (`TRUE`) or globally (`FALSE`, the default).

`cols` An `expression` giving a tidy selection of the columns that should be considered as sequence data. The default is all columns. The columns are automatically determined for `tna_data` objects. The `group` column is automatically removed from these columns if provided.

`params` A `list` of additional arguments for models of specific `type`. The potential elements of this list are:

- `n_gram`: An `integer` for n-gram transitions specifying the number of adjacent events. The default value is 2.

- max_gap: An integer for the gap-allowed transitions specifying the largest allowed gap size. The default is 1.
- windowed: Perform the model estimation by window. Supported for relative, frequency and co-occurrence types.
- window_size: An integer for the sliding window transitions specifying the window size. The default is 2.
- window_type: A character string that defines the window type. Either "rolling" or "tumbling".
- weighted: A logical value. If TRUE, the transitions are weighted by the inverse of the sequence length. Can be used for frequency, co-occurrence and reverse model types. The default is FALSE.
- direction: A character string specifying the direction of attention for models of type = "attention". The available options are "backward", "forward", and "both", for backward attention, forward attention, and bidirectional attention, respectively. The default is "forward".
- decay: A function that specifies the decay of the weights between two time points at a specific distance. The function should take three arguments: i, j and lambda, where i and j are numeric vectors of time values, and lambda is a numeric value for the decay rate. The function should return a numeric vector of weights. The default is function(i, j, lambda) exp(-abs(i - j) / lambda).
- lambda: A numeric value for the decay rate. The default is 1.
- time: A matrix or a data.frame providing the time values for each sequence and at time index. For tna_data objects, this can also be a logical value, where TRUE will use the time_data element of x for the time values. Date values are converted to numeric.
- duration: A matrix or a data.frame providing the time spent in each state for each sequence and time index. This is an alternative to time.

concat — An integer for the number of consecutive sequences to concatenate. The default is 1 (no concatenation).

na.rm — A logical value that determines if observations with NA value in group be removed. If FALSE, an additional category for NA values will be added. The default is FALSE and a warning is issued if NA values are detected.

Value

An object of class group_tna which is a list containing one element per cluster. Each element is a tna object.

See Also

Cluster-related functions communities(), mmm_stats(), rename_groups()

Examples

```
# Manually specified groups
group <- c(rep("High", 1000), rep("Low", 1000))
```

```
model <- group_model(group_regulation, group = group)

# Groups defined by a mixed Markov model
model <- group_model(engagement_mmm)

model <- group_tna(group_regulation, group = gl(2, 1000))

model <- group_ftna(group_regulation, group = gl(2, 1000))

model <- group_ctna(group_regulation, group = gl(2, 1000))

model <- group_atna(group_regulation, group = gl(2, 1000))
```

`group_regulation` *Example Wide Data on Group Regulation*

Description

Students' regulation during collaborative learning. Students' interactions were coded as: "adapt", "cohesion", "consensus", "coregulate", "discuss", "emotion", "monitor", "plan", "synthesis"

Usage

```
group_regulation
```

Format

A `data.frame` object.

Source

The data was generated synthetically.

See Also

Datasets `engagement`, `engagement_mmm`, `group_regulation_long`

group_regulation_long *Example Long Data on Group Regulation*

Description

Students' regulation during collaborative learning. This is the same dataset as `group_regulation` but in long format. In addition to students' actions (`Action`), it contains the student identifier (`Actor`), timestamp (`Time`), `Course` name, and collaboration `Group`. It also includes a column (`Achiever`) indicating whether the student is a high or low achiever.

Usage

```
group_regulation_long
```

Format

A `data.frame` object.

Source

The data was generated synthetically from `group_regulation`

See Also

Datasets `engagement`, `engagement_mmm`, `group_regulation`

hist.group_tna *Plot a Histogram of Edge Weights for a* `group_tna` *Object.*

Description

Plot a Histogram of Edge Weights for a `group_tna` Object.

Usage

```
## S3 method for class 'group_tna'
hist(x, ...)
```

Arguments

`x`	A `group_tna` object.
`...`	Additional arguments passed to `graphics::hist()`.

Value

A `list` (invisibly) of `histogram` objects of the edge weights of each cluster.

See Also

Basic functions build_model(), hist.tna(), plot.group_tna(), plot.tna(), plot_frequencies(), plot_frequencies.group_tna(), plot_mosaic(), plot_mosaic.group_tna(), plot_mosaic.tna_data(), print.group_tna(), print.summary.group_tna(), print.summary.tna(), print.tna(), summary.group_tna(), summary.tna(), tna-package

Examples

```
model <- group_model(engagement_mmm)
hist(model)
```

---

hist.tna *Plot a Histogram of Edge Weights in the Network*

Description

Plot a Histogram of Edge Weights in the Network

Usage

```
## S3 method for class 'tna'
hist(x, breaks, col = "lightblue", main, xlab, border = "white", ...)
```

Arguments

x	a vector of values for which the histogram is desired.
breaks	one of: • a vector giving the breakpoints between histogram cells, • a function to compute the vector of breakpoints, • a single number giving the number of cells for the histogram, • a character string naming an algorithm to compute the number of cells (see 'Details'), • a function to compute the number of cells. In the last three cases the number is a suggestion only; as the breakpoints will be set to pretty values, the number is limited to 1e6 (with a warning if it was larger). If breaks is a function, the x vector is supplied to it as the only argument (and the number of breaks is only limited by the amount of available memory).
col	a colour to be used to fill the bars.
main	A character string defining the title of the plot.
xlab	A character string defining the vertical axis label.
border	the color of the border around the bars. The default is to use the standard foreground color.
...	Additional arguments passed to graphics::hist().

Value

A histogram object of edge weights.

See Also

Basic functions build_model(), hist.group_tna(), plot.group_tna(), plot.tna(), plot_frequencies(), plot_frequencies.group_tna(), plot_mosaic(), plot_mosaic.group_tna(), plot_mosaic.tna_data(), print.group_tna(), print.summary.group_tna(), print.summary.tna(), print.tna(), summary.group_tna(), summary.tna(), tna-package

Examples

```
model <- tna(group_regulation)
hist(model)
```

---

import_data *Import Wide Format Sequence Data as Long Format Sequence Data*

Description

This function transforms wide format data where features are in separate columns into a long format suitable for sequence analysis. It creates windows of data based on row order and generates sequence order within these windows.

Usage

```
import_data(data, cols, id_cols, window_size = 1, replace_zeros = TRUE)
```

Arguments

data	A data.frame in wide format.
cols	An expression giving a tidy selection of column names to be transformed into long format (actions). This can be a vector of column names (e.g., c(feature1, feature2)) or a range specified as feature1:feature6 (without quotes) to include all columns from 'feature1' to 'feature6' in the order they appear in the data frame. For more information on tidy selections, see dplyr::select().
id_cols	An expression giving a tidy selection of column names that uniquely identify each observation (IDs).
window_size	An integer specifying the size of the window for sequence grouping. Default is 1 (each row is a separate window).
replace_zeros	A logical value indicating whether to replace 0s in cols with NA. The default is TRUE.

Value

A data.frame in long format with added columns for window and sequence order.

See Also

Other data: import_onehot(), list_random_state_pools(), prepare_data(), print.tna_data(), random_group_tna(), random_tna(), random_tna_mmm(), simulate.group_tna(), simulate.tna()

Examples

```
data <- data.frame(
  ID = c("A", "A", "B", "B"),
  Time = c(1, 2, 1, 2),
  feature1 = c(10, 0, 15, 20),
  feature2 = c(5, 8, 0, 12),
  feature3 = c(2, 4, 6, 8),
  other_col = c("X", "Y", "Z", "W")
)

# Using a vector
long_data1 <- import_data(
  data = data,
  cols = c(feature1, feature2),
  id_cols = c("ID", "Time"),
  window_size = 2,
  replace_zeros = TRUE
)

# Using a column range
long_data2 <- import_data(
  data = data,
  cols = feature1:feature3,
  id_cols = c("ID", "Time"),
  window_size = 2,
  replace_zeros = TRUE
)
```

---

import_onehot *Import One-Hot Data*

Description

Import One-Hot Data

Usage

```
import_onehot(
  data,
  cols,
  actor,
  session,
  interval,
```

```
  window_size = 1L,
  window_type = "tumbling",
  aggregate = FALSE
)
```

Arguments

data	A data.frame in wide format.
cols	An expression giving a tidy selection of columns to be considered as one-hot data.
actor	An optional character string giving the column name of data containing the actor identifiers.
session	An optional character string giving the column name of data containing the session identifiers.
interval	An integer that defines how many windows can appear at most per one row of the output data. If not provided (the default), all windows are concatenated per actor/session combination.
window_size	An integer specifying the window size for grouping.
window_type	A character string. Either "tumbling" (the default) for non-overlapping windows or "sliding" for one-step sliding window.
aggregate	A logical value that determines how multiple occurrences of the same event within a window are processed. Option TRUE aggregates multiple occurrences into a single occurrence. Option FALSE keeps all occurrences (the default).

Value

The processed data as a data.frame.

See Also

Other data: import_data(), list_random_state_pools(), prepare_data(), print.tna_data(), random_group_tna(), random_tna(), random_tna_mmm(), simulate.group_tna(), simulate.tna()

Examples

```
d <- data.frame(
  actor = gl(100, 5),
  session = gl(10, 50),
  feature1 = rbinom(500, 1, prob = 0.33),
  feature2 = rbinom(500, 1, prob = 0.25),
  feature3 = rbinom(500, 1, prob = 0.50)
)
onehot1 <- import_onehot(d, feature1:feature3)
onehot2 <- import_onehot(d, feature1:feature3, "actor", "session")
```

list_random_state_pools *List Built-in Label Pools for* random_tna()

Description

Return the names of the curated state-label pools available to random_tna() and random_group_tna(), with their sizes.

Usage

```
list_random_state_pools()
```

Value

A named integer vector mapping pool names to pool sizes.

See Also

Other data: import_data(), import_onehot(), prepare_data(), print.tna_data(), random_group_tna(), random_tna(), random_tna_mmm(), simulate.group_tna(), simulate.tna()

Examples

```
list_random_state_pools()
```

mmm_stats *Retrieve Statistics from a Mixture Markov Model (MMM)*

Description

Retrieve Statistics from a Mixture Markov Model (MMM)

Usage

```
mmm_stats(x, level = 0.05)

## S3 method for class 'tna_mmm'
mmm_stats(x, level = 0.05)

## S3 method for class 'mhmm'
mmm_stats(x, level = 0.05)
```

Arguments

x	A mhmm object (from the seqHMM package) or a tna_mmm object built by random_tna_mmm().
level	A numeric value representing the significance level for hypothesis testing and confidence intervals. Defaults to 0.05.

Value

A data.frame object.

See Also

Cluster-related functions communities(), group_model(), rename_groups()

Examples

```
mmm_stats(engagement_mmm)
```

---

permutation_test *Compare Two Networks from Sequence Data using a Permutation Test*

Description

This function compares two networks built from sequence data using permutation tests. The function builds Markov models for two sequence objects, computes the transition probabilities, and compares them by performing permutation tests. It returns the differences in transition probabilities, effect sizes, estimated p-values, and confidence intervals.

Usage

```
permutation_test(x, ...)

## S3 method for class 'tna'
permutation_test(
  x,
  y,
  adjust = "none",
  iter = 1000,
  paired = FALSE,
  level = 0.05,
  measures = character(0),
  ...
)
```

Arguments

x	A tna object containing sequence data for the first tna model.
...	Additional arguments passed to centralities().
y	A tna object containing sequence data for the second tna model.
adjust	A character string for the method to adjust p-values with for multiple comparisons. The default is "none" for no adjustment. See the method argument of stats::p.adjust() for details and available adjustment methods.
iter	An integer giving the number of permutations to perform. The default is 1000.
paired	A logical value. If TRUE, perform paired permutation tests; if FALSE, perform unpaired tests. The default is FALSE.
level	A numeric value giving the significance level for the permutation tests. The default is 0.05.
measures	A character vector of centrality measures to test. See centralities() for a list of available centrality measures.

Value

A tna_permutation object which is a list with two elements: edges and centralities, both containing the following elements:

- stats: A data.frame of original differences, effect sizes, and estimated p-values for each edge or centrality measure. The effect size is computed as the observed difference divided by the standard deviation of the differences of the permuted samples.
- diffs_true: A matrix of differences in the data.
- diffs_sig: A matrix showing the significant differences.

See Also

Validation functions bootstrap(), deprune(), estimate_cs(), permutation_test.group_tna(), plot.group_tna_bootstrap(), plot.group_tna_permutation(), plot.group_tna_stability(), plot.tna_bootstrap(), plot.tna_permutation(), plot.tna_reliability(), plot.tna_stability(), print.group_tna_bootstrap(), print.group_tna_permutation(), print.group_tna_stability(), print.summary.group_tna_bootstrap(), print.summary.tna_bootstrap(), print.tna_bootstrap(), print.tna_clustering(), print.tna_permutation(), print.tna_reliability(), print.tna_stability(), prune(), pruning_details(), reliability(), reprune(), summary.group_tna_bootstrap(), summary.tna_bootstrap()

Examples

```
model_x <- tna(group_regulation[1:200, ])
model_y <- tna(group_regulation[1001:1200, ])
# Small number of iterations for CRAN
permutation_test(model_x, model_y, iter = 20)
```

`permutation_test.group_tna` *Compare Networks using a Permutation Test*

Description

Test edge weight differences between all pairs or a subset of pairs of a `group_tna` object. See `permutation_test.tna()` for more details.

Usage

```
## S3 method for class 'group_tna'
permutation_test(
  x,
  groups,
  adjust = "none",
  iter = 1000,
  paired = FALSE,
  level = 0.05,
  measures = character(0),
  consecutive = FALSE,
  ...
)
```

Arguments

`x`	A `group_tna` object
`groups`	An `integer` vector or a `character` vector of group indices or names, respectively, defining which groups to compare. When not provided, all pairs are compared (the default).
`adjust`	A `character` string for the method to adjust p-values with for multiple comparisons. The default is `"none"` for no adjustment. See the `method` argument of `stats::p.adjust()` for details and available adjustment methods.
`iter`	An `integer` giving the number of permutations to perform. The default is 1000.
`paired`	A `logical` value. If `TRUE`, perform paired permutation tests; if `FALSE`, perform unpaired tests. The default is `FALSE`.
`level`	A `numeric` value giving the significance level for the permutation tests. The default is 0.05.
`measures`	A `character` vector of centrality measures to test. See `centralities()` for a list of available centrality measures.
`consecutive`	A `logical` value. If `FALSE` (the default), all pairwise comparisons are performed in lexicographic order with respect to the order of the groups. If `TRUE`, only comparisons between consecutive pairs of groups are performed.
`...`	Additional arguments passed to `centralities()`.

See Also

Validation functions bootstrap(), deprune(), estimate_cs(), permutation_test(), plot.group_tna_bootstrap(), plot.group_tna_permutation(), plot.group_tna_stability(), plot.tna_bootstrap(), plot.tna_permutation(), plot.tna_reliability(), plot.tna_stability(), print.group_tna_bootstrap(), print.group_tna_permutation(print.group_tna_stability(), print.summary.group_tna_bootstrap(), print.summary.tna_bootstrap(), print.tna_bootstrap(), print.tna_clustering(), print.tna_permutation(), print.tna_reliability(), print.tna_stability(), prune(), pruning_details(), reliability(), reprune(), summary.group_tna_bootstrap(summary.tna_bootstrap()

Examples

```
model <- group_model(engagement_mmm)
# Small number of iterations for CRAN
permutation_test(model, iter = 20)
```

---

plot.group_tna *Plot a Grouped Transition Network Analysis Model*

Description

Plots a transition network of each cluster using cograph.

Usage

```
## S3 method for class 'group_tna'
plot(x, title, which, ...)
```

Arguments

x	A group_model object.
title	A title for each plot. It can be a single string (the same one will be used for all plots) or a list (one per group)
which	An optional integer vector of groups to plot. By default, all groups are plotted.
...	Arguments passed on to plot.tna

node_list An optional list of two character vectors that define two mutually exclusive groups of node labels.

use_list_order A logical value. If node_list is provided, defines how the order of the nodes in the plot is defined. A TRUE value uses the order in node_list. Otherwise, the nodes are ranked based on edge weights and ordered according to the rank.

scale_nodes A character string giving the name of a centrality measure to scale the node size by. See centralities() for valid names. If missing (the default), uses default cograph::splot() scaling. The value of node_size provided via ... is used as baseline size.

scaling_factor A numeric value specifying how strongly to scale the nodes when scale_nodes is provided. Values between 0 and 1 will result in smaller differences and values larger than 1 will result in greater differences. The default is 0.5.

Value

NULL (invisibly).

See Also

Basic functions build_model(), hist.group_tna(), hist.tna(), plot.tna(), plot_frequencies(), plot_frequencies.group_tna(), plot_mosaic(), plot_mosaic.group_tna(), plot_mosaic.tna_data(), print.group_tna(), print.summary.group_tna(), print.summary.tna(), print.tna(), summary.group_tna(), summary.tna(), tna-package

Examples

```
model <- group_model(engagement_mmm)
plot(model, which = 1)
```

---

plot.group_tna_bootstrap

Plot a Bootstrapped Grouped Transition Network Analysis Model

---

Description

Plot a Bootstrapped Grouped Transition Network Analysis Model

Usage

```
## S3 method for class 'group_tna_bootstrap'
plot(x, title, ...)
```

Arguments

x	A group_tna_bootstrap object.
title	A character vector of titles to use for each plot.
...	Additional arguments passed to plot.tna().

See Also

Validation functions bootstrap(), deprune(), estimate_cs(), permutation_test(), permutation_test.group_tna(), plot.group_tna_permutation(), plot.group_tna_stability(), plot.tna_bootstrap(), plot.tna_permutation(), plot.tna_reliability(), plot.tna_stability(), print.group_tna_bootstrap(), print.group_tna_permutation(), print.group_tna_stability(), print.summary.group_tna_bootstrap(), print.summary.tna_bootstrap(), print.tna_bootstrap(), print.tna_clustering(), print.tna_permutation(), print.tna_reliability(), print.tna_stability(), prune(), pruning_details(), reliability(), reprune(), summary.group_tna_bootstrap(), summary.tna_bootstrap()

Examples

```
model <- group_model(engagement_mmm)
# Small number of iterations for CRAN
boot <- bootstrap(model, iter = 10)
plot(boot)
```

`plot.group_tna_centralities` *Plot Centrality Measures*

Description

Plot Centrality Measures

Usage

```
## S3 method for class 'group_tna_centralities'
plot(
  x,
  reorder = TRUE,
  ncol = 3,
  scales = c("free_x", "fixed"),
  colors,
  palette = "Set2",
  labels = TRUE,
  ...
)
```

Arguments

`x`	A `group_tna_centralities` object.
`reorder`	A `logical` value indicating whether to reorder the values for each centrality in a descending order. The default is `TRUE`.
`ncol`	Number of columns to use for the facets. The default is 3.
`scales`	Either `"fixed"` or `"free_x"` (the default). If `"free_x"`, the horizontal axis is scaled individually in each facet. If `"fixed"`, the same values are used for all axes.
`colors`	The colors for each node (default is the model colors if the `tna` model object is passed, otherwise `"black"`).
`palette`	A color palette to be applied if `colors` is not specified.
`labels`	A `logical` value indicating whether to show the centrality numeric values. The default is `TRUE`.
`...`	Ignored.

Value

A ggplot object displaying a line chart for each centrality with one line per cluster.

See Also

Centrality measure functions betweenness_network(), centralities(), plot.tna_centralities(), print.group_tna_centralities(), print.tna_centralities()

Examples

```
model <- group_model(engagement_mmm)
cm <- centralities(model)
plot(cm)
```

---

plot.group_tna_cliques

Plot Found Cliques

Description

Plot Found Cliques

Usage

```
## S3 method for class 'group_tna_cliques'
plot(x, title, ...)
```

Arguments

x	A group_tna_cliques object.
title	A character vector of titles to use for each plot.
...	Arguments passed to plot.tna_cliques().

Value

A list (invisibly) with one element per cluster. Each element contains a cograph_network plot when only one clique is present per cluster, otherwise the element is NULL.

See Also

Clique-related functions cliques(), plot.tna_cliques(), print.group_tna_cliques(), print.tna_cliques()

Examples

```
model <- group_model(engagement_mmm)
cliq <- cliques(model, size = 2)
plot(cliq, ask = FALSE)
```

plot.group_tna_communities *Plot Detected Communities*

Description

Plot Detected Communities

Usage

```
## S3 method for class 'group_tna_communities'
plot(x, title, colors, ...)
```

Arguments

x	A group_tna_communities object.
title	A character vector of titles to use for each plot.
colors	A character vector of colors to use.
...	Arguments passed to plot.tna_communities().

Value

A list (invisibly) of cograph_network objects in which the nodes are colored by community for each cluster.

See Also

Community detection functions communities(), plot.tna_communities(), print.group_tna_communities(), print.tna_communities()

Examples

```
model <- group_model(engagement_mmm)
comm <- communities(model)
plot(comm)
```

plot.group_tna_permutation *Plot Permutation Test Results*

Description

Plot Permutation Test Results

Usage

```
## S3 method for class 'group_tna_permutation'
plot(x, title, ...)
```

Arguments

x	A group_tna_permutation object.
title	An optional character vector of titles for each plot. When not provided, the title shows the names of the clusters being contrasted.
...	Arguments passed to plot.tna_permutation().

Value

A list (invisibly) of cograph_network objects depicting the significant difference between each pair.

See Also

Validation functions bootstrap(), deprune(), estimate_cs(), permutation_test(), permutation_test.group_tna(), plot.group_tna_bootstrap(), plot.group_tna_stability(), plot.tna_bootstrap(), plot.tna_permutation(), plot.tna_reliability(), plot.tna_stability(), print.group_tna_bootstrap(), print.group_tna_permutation(), print.group_tna_stability(), print.summary.group_tna_bootstrap(), print.summary.tna_bootstrap(), print.tna_bootstrap(), print.tna_clustering(), print.tna_permutation(), print.tna_reliability(), print.tna_stability(), prune(), pruning_details(), reliability(), reprune(), summary.group_tna_bootstrap(), summary.tna_bootstrap()

Examples

```
model <- group_tna(engagement_mmm)
# Small number of iterations for CRAN
perm <- permutation_test(model, iter = 20)
plot(perm)
```

plot.group_tna_stability *Plot Centrality Stability Results*

Description

Plot Centrality Stability Results

Usage

```
## S3 method for class 'group_tna_stability'
plot(x, ...)
```

Arguments

x	A group_tna_stability object.
...	Arguments passed to plot.tna_stability().

Value

A list (invisibly) of ggplot objects displaying the stability analysis plot.

See Also

Validation functions bootstrap(), deprune(), estimate_cs(), permutation_test(), permutation_test.group_tna(), plot.group_tna_bootstrap(), plot.group_tna_permutation(), plot.tna_bootstrap(), plot.tna_permutation(), plot.tna_reliability(), plot.tna_stability(), print.group_tna_bootstrap(), print.group_tna_permutation(), print.group_tna_stability(), print.summary.group_tna_bootstrap(), print.summary.tna_bootstrap(), print.tna_bootstrap(), print.tna_clustering(), print.tna_permutation(), print.tna_reliability(), print.tna_stability(), prune(), pruning_details(), reliability(), reprune(), summary.group_tna_bootstrap(), summary.tna_bootstrap()

Examples

```
model <- group_model(engagement_mmm)
# Low number of iterations for CRAN
stability <- estimate_cs(
  model,
  drop_prop = c(0.3, 0.5, 0.7, 0.9),
  iter = 10
)
plot(stability)
```

plot.tna *Plot a Transition Network Analysis Model*

Description

This function plots a transition network analysis (TNA) model using the cograph package. The nodes in the graph represent states, with node sizes corresponding to initial state probabilities. Edge labels represent the edge weights of the network. See cograph::splot() for details on how to further configure the plot.

Usage

```
## S3 method for class 'tna'
plot(
  x,
  node_list,
  use_list_order = TRUE,
  scale_nodes,
  scaling_factor = 0.5,
  ...
)
```

Arguments

x	A tna object from build_model().
node_list	An optional list of two character vectors that define two mutually exclusive groups of node labels.
use_list_order	A logical value. If node_list is provided, defines how the order of the nodes in the plot is defined. A TRUE value uses the order in node_list. Otherwise, the nodes are ranked based on edge weights and ordered according to the rank.
scale_nodes	A character string giving the name of a centrality measure to scale the node size by. See centralities() for valid names. If missing (the default), uses default cograph::splot() scaling. The value of node_size provided via ... is used as baseline size.
scaling_factor	A numeric value specifying how strongly to scale the nodes when scale_nodes is provided. Values between 0 and 1 will result in smaller differences and values larger than 1 will result in greater differences. The default is 0.5.
...	Additional arguments passed to cograph::splot() or cograph::plot_htna().

Value

A cograph_network plot of the transition network.

See Also

Basic functions build_model(), hist.group_tna(), hist.tna(), plot.group_tna(), plot_frequencies(), plot_frequencies.group_tna(), plot_mosaic(), plot_mosaic.group_tna(), plot_mosaic.tna_data(), print.group_tna(), print.summary.group_tna(), print.summary.tna(), print.tna(), summary.group_tna(), summary.tna(), tna-package

Examples

```
model <- tna(group_regulation)
plot(model)
```

---

plot.tna_bootstrap *Plot a Bootstrapped Transition Network Analysis Model*

Description

Plot a Bootstrapped Transition Network Analysis Model

Usage

```
## S3 method for class 'tna_bootstrap'
plot(x, ...)
```

Arguments

x	A tna_bootstrap object.
...	Additional arguments passed to plot.tna().

See Also

Validation functions bootstrap(), deprune(), estimate_cs(), permutation_test(), permutation_test.group_tna() plot.group_tna_bootstrap(), plot.group_tna_permutation(), plot.group_tna_stability(), plot.tna_permutation(), plot.tna_reliability(), plot.tna_stability(), print.group_tna_bootstrap(), print.group_tna_permutation(), print.group_tna_stability(), print.summary.group_tna_bootstrap(), print.summary.tna_bootstrap(), print.tna_bootstrap(), print.tna_clustering(), print.tna_permutation(), print.tna_reliability(), print.tna_stability(), prune(), pruning_details(), reliability(), reprune(), summary.group_tna_bootstrap(), summary.tna_bootstrap()

Examples

```
model <- tna(group_regulation)
# Small number of iterations for CRAN
boot <- bootstrap(model, iter = 50)
plot(boot)
```

`plot.tna_centralities` *Plot Centrality Measures*

Description

Plots the centrality measures of a `tna_centralities` object as a lollipop chart. The resulting plot includes facets for each centrality measure, showing the values for each state. The returned plot is a `ggplot2` object, so it can be easily modified and styled. See `centralities()` for details on the centrality measures.

Usage

```
## S3 method for class 'tna_centralities'
plot(
  x,
  reorder = TRUE,
  ncol = 3,
  scales = c("free_x", "fixed"),
  colors,
  labels = TRUE,
  ...
)
```

Arguments

`x`	An object of class `tna_centralities`.
`reorder`	A `logical` value indicating whether to reorder the values for each centrality in a descending order. The default is `TRUE`.
`ncol`	Number of columns to use for the facets. The default is 3.
`scales`	Either `"fixed"` or `"free_x"` (the default). If `"free_x"`, the horizontal axis is scaled individually in each facet. If `"fixed"`, the same values are used for all axes.
`colors`	The colors for each node (default is the model colors if the `tna` model object is passed, otherwise `"black"`).
`labels`	A `logical` value indicating whether to show the centrality numeric values. The default is `TRUE`.
`...`	Ignored.

Value

A `ggplot` object displaying the lollipop charts for each centrality measure.

See Also

Centrality measure functions `betweenness_network()`, `centralities()`, `plot.group_tna_centralities()`, `print.group_tna_centralities()`, `print.tna_centralities()`

Examples

```
tna_model <- tna(group_regulation)
cm <- centralities(tna_model)
plot(cm, ncol = 3, reorder = TRUE)
```

`plot.tna_cliques` *Plot Cliques of a TNA Network*

Description

Plot Cliques of a TNA Network

Usage

```
## S3 method for class 'tna_cliques'
plot(x, n = 6, first = 1, show_loops = FALSE, colors, ask = TRUE, ...)
```

Arguments

`x`	A `tna_cliques` object.
`n`	An `integer` defining the maximum number of cliques to show. The defaults is `6`.
`first`	An `integer` giving the index of the first clique to show. The default index is `1`.
`show_loops`	A `logical` value indicating whether to include loops in the plots or not.
`colors`	Optional `character` vector of colors values to use for the nodes.
`ask`	A `logical` value. When `TRUE`, show plots one by one and asks to plot the next plot in interactive mode.
`...`	Ignored.

Value

`NULL` (invisibly).

See Also

Clique-related functions `cliques()`, `plot.group_tna_cliques()`, `print.group_tna_cliques()`, `print.tna_cliques()`

Examples

```
model <- tna(group_regulation)
cliq <- cliques(model, size = 2)
plot(cliq, n = 1, ask = FALSE)
```

`plot.tna_communities` *Plot Communities*

Description

This function visualizes the communities detected within a `tna` object based on different community detection algorithms and their corresponding color mappings.

Usage

```
## S3 method for class 'tna_communities'
plot(x, colors, method, ...)
```

Arguments

`x`	A `communities` object generated by the `find_communities` method. Each community detection method maps nodes or points in to a specific communities.
`colors`	A `character` vector of color values used for visualizing community assignments.
`method`	A `character` string naming a community detection method to use for coloring the plot. The default is to use the first available method in `x`. See `communities()` for details.
`...`	Additional arguments passed to `cograph::splot()`.

Value

A `cograph_network` object in which the nodes are colored by community.

See Also

Community detection functions `communities()`, `plot.group_tna_communities()`, `print.group_tna_communities()`, `print.tna_communities()`

Examples

```
model <- tna(group_regulation)
comm <- communities(model)
plot(comm, method = "leading_eigen")
```

plot.tna_comparison *Plot the Comparison of Two TNA Models or Matrices*

Description

Plot the Comparison of Two TNA Models or Matrices

Usage

```
## S3 method for class 'tna_comparison'
plot(
  x,
  type = "heatmap",
  population = "difference",
  method = "pearson",
  name_x = "x",
  name_y = "y",
  ...
)
```

Arguments

x	A tna_comparison object.
type	A character string naming the type of plot to produce. The available options are "heatmap" (the default), "scatterplot", "centrality_heatmap", and "weight_density".
population	A "character" string naming the population for which to produce the heatmaps, i.e, one of "x", "y", or "difference" for the differences. Ignored for type = "scatterplot". Defaults to "diff".
method	A character string naming the correlation coefficient to use when plotting a scatterplot. The available options are "pearson" (the default), "kendall", "spearman", and "distance". The final option is the distance correlation coefficient of Szekely, Rizzo, and Bakirov (2007). See also the energy package for further information on this measure.
name_x	An optional character string to use as the name of the first population in the plots. The default is "x".
name_y	An optional character string to use as the name of the second population in the plots. The default is "y".
...	Ignored.

Value

A ggplot object.

References

Szekely, G.J., Rizzo, M.L., and Bakirov, N.K. (2007), Measuring and Testing Dependence by Correlation of Distances, *Annals of Statistics*, 35(6), 2769-2794. doi:10.1214/009053607000000505

See Also

Model comparison functions `compare()`, `compare.group_tna()`, `compare_sequences()`, `plot.tna_sequence_comparis`
`plot_compare()`, `plot_compare.group_tna()`, `print.tna_comparison()`, `print.tna_sequence_comparison()`

Examples

```
model_x <- tna(group_regulation[1:200, ])
model_y <- tna(group_regulation[1001:1200, ])
comp <- compare(model_x, model_y)
plot(comp)
```

---

plot.tna_permutation *Plot the Significant Differences from a Permutation Test*

Description

Plot the Significant Differences from a Permutation Test

Usage

```
## S3 method for class 'tna_permutation'
plot(x, colors, posCol = "#009900", negCol = "red", edge_labels = TRUE, ...)
```

Arguments

`x`	A `tna_permutation` object.
`colors`	See `cograph::splot()`.
`posCol`	Color for plotting edges the difference in edge weights is positive. See `cograph::splot()`.
`negCol`	Color for plotting edges when the the difference in edge weights is negative. See `cograph::splot()`.
`edge_labels`	Boolean indicating whether edge labels should be ploted. Defaults to `TRUE`
`...`	Arguments passed to `plot_model()`.

Value

A `cograph_network` object containing only the significant edges according to the permutation test.

See Also

Validation functions bootstrap(), deprune(), estimate_cs(), permutation_test(), permutation_test.group_tna(), plot.group_tna_bootstrap(), plot.group_tna_permutation(), plot.group_tna_stability(), plot.tna_bootstrap(), plot.tna_reliability(), plot.tna_stability(), print.group_tna_bootstrap(), print.group_tna_permutation(), print.group_tna_stability(), print.summary.group_tna_bootstrap(), print.summary.tna_bootstrap(), print.tna_bootstrap(), print.tna_clustering(), print.tna_permutation(), print.tna_reliability(), print.tna_stability(), prune(), pruning_details(), reliability(), reprune(), summary.group_tna_bootstrap(), summary.tna_bootstrap()

Examples

```
model_x <- tna(group_regulation[1:200, ])
model_y <- tna(group_regulation[1001:1200, ])
# Small number of iterations for CRAN
perm <- permutation_test(model_x, model_y, iter = 20)
plot(perm)
```

---

plot.tna_reliability *Plot Reliability Analysis Results*

Description

Plot Reliability Analysis Results

Usage

```
## S3 method for class 'tna_reliability'
plot(x, type = "histogram", metric = "Median Abs. Diff.", ...)
```

Arguments

x	A tna_reliability object.
type	A character string specifying the plot type. The options are: "histogram" (default), "density", or "boxplot".
metric	A character string specifying the metric to plot. The default is the median absolute difference ("Median Abs. Diff.").
...	Ignored

See Also

Validation functions bootstrap(), deprune(), estimate_cs(), permutation_test(), permutation_test.group_tna(), plot.group_tna_bootstrap(), plot.group_tna_permutation(), plot.group_tna_stability(), plot.tna_bootstrap(), plot.tna_permutation(), plot.tna_stability(), print.group_tna_bootstrap(), print.group_tna_permutation(), print.group_tna_stability(), print.summary.group_tna_bootstrap(), print.summary.tna_bootstrap(), print.tna_bootstrap(), print.tna_clustering(), print.tna_permutation(), print.tna_reliability(), print.tna_stability(), prune(), pruning_details(), reliability(), reprune(), summary.group_tna_bootstrap(), summary.tna_bootstrap()

Examples

```
# Small number of iterations for CRAN
model <- tna(engagement)
rel <- reliability(model, iter = 20)
plot(rel)
```

`plot.tna_sequence_comparison`

Plot a Sequence Comparison

Description

Visualize the differences in pattern counts by plotting the standardized residuals by pattern and group.

Usage

```
## S3 method for class 'tna_sequence_comparison'
plot(
  x,
  n = 10,
  legend = TRUE,
  cells = TRUE,
  text_color = "white",
  digits = 2L,
  ...
)
```

Arguments

`x`	A `tna_sequence_comparison` object.
`n`	An `integer` giving the number of patterns to plot. The default is `10`.
`legend`	A `logical` value indicating whether to show the color scale legend. The default is `TRUE`.
`cells`	A `logical` value indicating whether to display the numeric values in each cell. The default is `TRUE`.
`text_color`	A `character` string specifying the text color to use for the cell values. The default is `"white"`.
`digits`	An `integer` specifying the number of digits for the cell values.
`...`	Not used.

Value

A `ggplot` object.

See Also

Model comparison functions `compare()`, `compare.group_tna()`, `compare_sequences()`, `plot.tna_comparison()`, `plot_compare()`, `plot_compare.group_tna()`, `print.tna_comparison()`, `print.tna_sequence_comparison()`

Examples

```
idx <- c(1:500, 1001:1500)
group <- c(rep("High", 500), rep("Low", 500))
comp <- compare_sequences(group_regulation[idx, ], group)
plot(comp)
```

---

plot.tna_stability *Plot Centrality Stability Results*

Description

This function visualizes the centrality stability results produced by the `estimate_centrality_stability` function. It shows how different centrality measures' correlations change as varying proportions of cases are dropped, along with their confidence intervals (CIs).

Usage

```
## S3 method for class 'tna_stability'
plot(x, level = 0.05, ...)
```

Arguments

`x`	A `tna_stability` object produced by `estimate_cs`.
`level`	A `numeric` value representing the significance level for the confidence intervals. Defaults to `0.05`.
`...`	Ignored.

Details

The function aggregates the results for each centrality measure across multiple proportions of dropped cases (e.g., 0.1, 0.2, ..., 0.9) and calculates the mean and the desired quantiles for each proportion. The confidence intervals (CIs) are computed based on the quantiles and displayed in the plot.

If no valid data is available for a centrality measure (e.g., missing or NA values), the function skips that measure with a warning.

The plot includes:

- The mean correlation for each centrality measure as a function of the proportion of dropped cases.

- Shaded confidence intervals representing CIs for each centrality measure.
- A horizontal dashed line at the threshold value used for calculating the CS-coefficient.
- A subtitle listing the CS-coefficients for each centrality measure.

Value

A ggplot object displaying the stability analysis plot.

See Also

Validation functions bootstrap(), deprune(), estimate_cs(), permutation_test(), permutation_test.group_tna(), plot.group_tna_bootstrap(), plot.group_tna_permutation(), plot.group_tna_stability(), plot.tna_bootstrap(), plot.tna_permutation(), plot.tna_reliability(), print.group_tna_bootstrap(), print.group_tna_permutation(), print.group_tna_stability(), print.summary.group_tna_bootstrap(), print.summary.tna_bootstrap(), print.tna_bootstrap(), print.tna_clustering(), print.tna_permutation(), print.tna_reliability(), print.tna_stability(), prune(), pruning_details(), reliability(), reprune(), summary.group_tna_bootstrap(), summary.tna_bootstrap()

Examples

```
model <- tna(group_regulation)
cs <- estimate_cs(model, iter = 10)
plot(cs)
```

---

plot_associations *Plot an Association Network*

Description

Plot an Association Network

Usage

```
plot_associations(x, ...)

## S3 method for class 'tna'
plot_associations(x, edge_color, ...)
```

Arguments

x	A tna object.
...	Additional arguments passed to plot_model().
edge_color	An optional character vector of colors for the edges. By default, the colors are specified by the magnitude of the standardized residual.

Value

A cograph_network object.

Examples

```
model <- ftna(group_regulation)
plot_associations(model)
```

---

plot_compare *Plot the Difference Network Between Two Models*

Description

Plots the difference network between model x and model y. The edges are computed from subtracting the two models. The pie chart is the difference in initial probabilities between model x and model y. Green color indicates that xis greater than yand red indicates otherwise.

Usage

```
plot_compare(x, ...)

## S3 method for class 'tna'
plot_compare(x, y, posCol = "#009900", negCol = "red", ...)
```

Arguments

x	A tna object. This is the the principal model.
...	Additional arguments passed to cograph::splot().
y	A tna object. This is the model subtracted from the principal model.
posCol	Color for plotting edges and pie when the first group has a higher value. See cograph::splot().
negCol	Color for plotting edges and pie when the second group has a higher value. See cograph::splot().

Value

A cograph_network object displaying the difference network between the two models.

See Also

Model comparison functions compare(), compare.group_tna(), compare_sequences(), plot.tna_comparison(), plot.tna_sequence_comparison(), plot_compare.group_tna(), print.tna_comparison(), print.tna_sequence_comparison()

Examples

```
model_x <- tna(group_regulation[group_regulation[, 1] == "plan", ])
model_y <- tna(group_regulation[group_regulation[, 1] != "plan", ])
plot_compare(model_x, model_y)
```

`plot_compare.group_tna` *Plot the Difference Network Between Two Groups*

Description

Plot the Difference Network Between Two Groups

Usage

```
## S3 method for class 'group_tna'
plot_compare(x, i = NULL, j = NULL, ...)
```

Arguments

`x`	A `group_tna` object.
`i`	An `integer` index or the name of the principal cluster as a `character` string. When `NULL`, defaults are chosen automatically (see `cograph::plot_compare()`).
`j`	An `integer` index or the name of the secondary cluster as a `character` string. When `NULL`, defaults are chosen automatically (see `cograph::plot_compare()`).
`...`	Additional arguments passed to `cograph::plot_compare()`.

Value

A `cograph_network` object displaying the difference network between the two clusters

See Also

Model comparison functions `compare()`, `compare.group_tna()`, `compare_sequences()`, `plot.tna_comparison()`, `plot.tna_sequence_comparison()`, `plot_compare()`, `print.tna_comparison()`, `print.tna_sequence_comparison()`

Examples

```
model <- group_model(engagement_mmm)
plot_compare(model)
```

plot_frequencies *Plot the Frequency Distribution of States*

Description

Plot the Frequency Distribution of States

Usage

```
plot_frequencies(x, ...)

## S3 method for class 'tna'
plot_frequencies(x, width = 0.7, hjust = 1.2, show_label = TRUE, colors, ...)
```

Arguments

x	A tna object created from sequence data.
...	Ignored.
width	A numeric value for the Width of the bars. Default is 0.7,
hjust	A numeric value for the horizontal adjustment of the labels. Default is 1.2.
show_label	A logical value indicating whether to show a label with the frequency counts. Default is TRUE.
colors	A character vector of colors to be used in the plot (one per label) or a single color.

Value

A ggplot object.

See Also

Basic functions build_model(), hist.group_tna(), hist.tna(), plot.group_tna(), plot.tna(), plot_frequencies.group_tna(), plot_mosaic(), plot_mosaic.group_tna(), plot_mosaic.tna_data(), print.group_tna(), print.summary.group_tna(), print.summary.tna(), print.tna(), summary.group_tna(), summary.tna(), tna-package

Examples

```
model <- tna(group_regulation)
plot_frequencies(model)
plot_frequencies(model, width =  0.5, colors = "pink")
```

plot_frequencies.group_tna *Plot the Frequency Distribution of States*

Description

Plot the Frequency Distribution of States

Usage

```
## S3 method for class 'group_tna'
plot_frequencies(
  x,
  label,
  colors,
  width = 0.7,
  palette = "Set2",
  show_label = TRUE,
  position = "dodge",
  hjust = 1.2,
  ...
)
```

Arguments

x	A group_tna object.
label	An optional character string that can be provided to specify the grouping factor name if x was not constructed using a column name of the original data.
colors	A character vector of colors to be used in the plot (one per group).
width	A numeric value for the width of the bars. The default is 0.7.
palette	A character string that specifies the palette to be used if colors are not passed.
show_label	A logical value indicating whether to show a label with the frequency counts. Default is TRUE.
position	Position of the bars:"dodge", "dodge2", "fill" or "stack".
hjust	A numeric value for the horizontal adjustment of the labels. The default is 1.2.
...	Ignored.

Value

A ggplot object.

See Also

Basic functions build_model(), hist.group_tna(), hist.tna(), plot.group_tna(), plot.tna(), plot_frequencies(), plot_mosaic(), plot_mosaic.group_tna(), plot_mosaic.tna_data(), print.group_tna(), print.summary.group_tna(), print.summary.tna(), print.tna(), summary.group_tna(), summary.tna(), tna-package

Examples

```
model <- group_model(engagement_mmm)
# Default
plot_frequencies(model)
# Default labels outside and custom colors
plot_frequencies(
  model,
  width = 0.9,
  hjust = -0.3,
  colors = c("#218516", "#f9c22e", "#53b3cb")
)
# Stacked with no labels
plot_frequencies(model, position = "stack", show_label = FALSE)
# Fill
plot_frequencies(model, position = "fill", hjust = 1.1)
```

---

plot_mosaic *Create a Mosaic Plot of Transitions or Events*

Description

Create a Mosaic Plot of Transitions or Events

Usage

```
plot_mosaic(x, ...)

## S3 method for class 'tna'
plot_mosaic(x, ...)
```

Arguments

x	A tna or a group_tna object.
...	Ignored.

Value

A ggplot object.

See Also

Basic functions build_model(), hist.group_tna(), hist.tna(), plot.group_tna(), plot.tna(), plot_frequencies(), plot_frequencies.group_tna(), plot_mosaic.group_tna(), plot_mosaic.tna_data(), print.group_tna(), print.summary.group_tna(), print.summary.tna(), print.tna(), summary.group_tna(), summary.tna(), tna-package

Examples

```
ftna_model <- ftna(group_regulation)
plot_mosaic(ftna_model)
```

`plot_mosaic.group_tna` *Plot State Frequencies as a Mosaic Between Two Groups*

Description

Plot State Frequencies as a Mosaic Between Two Groups

Usage

```
## S3 method for class 'group_tna'
plot_mosaic(x, label, ...)
```

Arguments

`x`	A `group_tna` object.
`label`	An optional `character` string that can be provided to specify the grouping factor name if `x` was not constructed using a column name of the original data.
`...`	Ignored.

Value

A `ggplot` object.

See Also

Basic functions `build_model()`, `hist.group_tna()`, `hist.tna()`, `plot.group_tna()`, `plot.tna()`, `plot_frequencies()`, `plot_frequencies.group_tna()`, `plot_mosaic()`, `plot_mosaic.tna_data()`, `print.group_tna()`, `print.summary.group_tna()`, `print.summary.tna()`, `print.tna()`, `summary.group_tna()`, `summary.tna()`, `tna-package`

Examples

```
model <- group_model(engagement, group = rep(1:3, length.out = 1000))
plot_mosaic(model)
```

`plot_mosaic.tna_data` *Plot State Frequencies as a Mosaic Between Two Groups*

Description

Plot State Frequencies as a Mosaic Between Two Groups

Usage

```
## S3 method for class 'tna_data'
plot_mosaic(x, group, label = "Group", ...)
```

Arguments

`x`	A `tna_data` object.
`group`	A `character` string giving the column name of the (meta) data to contrast the frequencies with or a vector of group indicators with the the same length as the number of rows in the sequence data.
`label`	An optional `character` string that specifies a label for the grouping variable when `group` is not a column name of the data.
`...`	Ignored.

Value

A `ggplot` object.

See Also

Basic functions `build_model()`, `hist.group_tna()`, `hist.tna()`, `plot.group_tna()`, `plot.tna()`, `plot_frequencies()`, `plot_frequencies.group_tna()`, `plot_mosaic()`, `plot_mosaic.group_tna()`, `print.group_tna()`, `print.summary.group_tna()`, `print.summary.tna()`, `print.tna()`, `summary.group_tna()`, `summary.tna()`, `tna-package`

Examples

```
d <- data.frame(
  time = rep(1:5, rep = 4),
  group = rep(1:4, each = 5),
  event = sample(LETTERS[1:3], 20, replace = TRUE)
)
sequence_data <- prepare_data(
  d,
  time = "time",
  actor = "group",
  action = "event"
)
plot_mosaic(sequence_data, group = "group")
```

plot_sequences *Create a Sequence Index Plot or a Distribution Plot*

Description

Create a Sequence Index Plot or a Distribution Plot

Usage

```
plot_sequences(x, ...)

## S3 method for class 'tna'
plot_sequences(
  x,
  group,
  type = "index",
  scale = "proportion",
  geom = "bar",
  include_na = FALSE,
  na_color = "white",
  sort_by,
  show_n = TRUE,
  border,
  title,
  legend_title,
  xlab,
  ylab,
  tick = 5,
  ncol = 2L,
  ...
)

## S3 method for class 'tna_data'
plot_sequences(
  x,
  group,
  type = "index",
  scale = "proportion",
  geom = "bar",
  include_na = FALSE,
  colors,
  na_color = "white",
  sort_by,
  show_n = TRUE,
  border,
  title,
  legend_title,
```

```
  xlab,
  ylab,
  tick = 5,
  ncol = 2L,
  ...
)

## Default S3 method:
plot_sequences(
  x,
  cols = tidyselect::everything(),
  group,
  type = "index",
  scale = "proportion",
  geom = "bar",
  include_na = FALSE,
  colors,
  na_color = "white",
  sort_by,
  show_n = TRUE,
  border,
  title,
  legend_title,
  xlab,
  ylab,
  tick = 5,
  ncol = 2L,
  ...
)

## S3 method for class 'group_tna'
plot_sequences(
  x,
  type = "index",
  scale = "proportion",
  geom = "bar",
  include_na = FALSE,
  na_color = "white",
  sort_by,
  show_n = TRUE,
  border,
  title,
  legend_title,
  xlab,
  ylab,
  tick = 1,
  ncol = 2L,
  ...
```

```
)
```

Arguments

x	A tna, group_tna, tna_data or a data.frame object with sequence data in wide format.
...	Ignored.
group	A vector indicating the group assignment of each row of the data. Must have the same length as the number of rows of x. Alternatively, a single character string giving the column name of the data that defines the group when x is a wide format data.frame or a tna_data object. Used for faceting the plot.
type	A character string for the type of plot to generate. The available options are "index" (the default) for a sequence index plot, and "distribution" showing the distribution of the states over time.
scale	A character string that determines the scaling of the vertical axis for distribution plots. The options are "proportion" (the default) and "count" for proportions and raw counts of states, respectively.
geom	A character string for the type of geom to use for distribution plots. The options are "bar" (the default) and "area".
include_na	A logical value for whether to include missing values for distribution plots. The default is FALSE. If TRUE, the missing values are converted to a new state and included in the plot.
na_color	A character string giving the color to use for missing values. The default is "white".
sort_by	An optional expression giving a tidy selection of column names of x to sort by or "everything".
show_n	A logical value for whether to add the number of observations (total or by group) to the plot title.
border	A character string giving the color for borders. For index plots, this is the color of borders between cells (tiles). For distribution plot with geom = "bar", this is the color of bar outlines. Not applicable to geom = "area".
title	An optional character string providing a title for the plot.
legend_title	An optional character string providing a title for the legend.
xlab	A character string giving the label for the horizontal axis. The default is "Time".
ylab	A character string giving the label for the vertical axis. The default is "Sequence" for index plots, and "Proportion" or "Count" based on scale for distribution plots.
tick	An integer specifying the horizontal axis label interval. The default value tick = 5 shows every 5th label. Setting this to 1 will show every label.
ncol	Number of columns to use for the facets. The default is 2.
colors	A named character vector mapping states to colors, or an unnamed character vector. If missing, a default palette is used.
cols	An expression giving a tidy selection of column names to be treated as time points. By default, all columns will be used.

Value

A ggplot object containing either a sequence index plot or a state distribution plot, faceted by group when supplied.

Examples

```
# Sequence index plot (default)
plot_sequences(
  group_regulation,
  group = rep(1:2, each = 1000),
)
# State distribution plot
plot_sequences(
  group_regulation,
  group = rep(1:2, each = 1000),
  type = "distribution",
)
```

---

prepare_data *Compute User Sessions from Event Data*

Description

Processes a dataset to create user sessions based on time gaps, ordering columns, or actor groupings. It supports different ways to understand order in user behavior and provides flexibility when widening the data.

Usage

```
prepare_data(
  data,
  actor,
  time,
  action,
  order,
  time_threshold = 900,
  custom_format = NULL,
  is_unix_time = FALSE,
  unix_time_unit = "seconds",
  unused_fn = dplyr::first
)
```

Arguments

data	A data.frame or containing the action/event data.

actor	A character vector or an expression that represents a tidy selection of the names of the columns that represent a user/actor identifiers. If not provided and neither time nor order is specified, the entire dataset is treated as a single session. In the case of multiple actors, a new .actor column is added that represents the interaction of the given columns.
time	A character string or an expression giving the name of the column representing timestamps of the action events.
action	A character string or an expression giving the name of the column holding the information about the action taken.
order	A character string or an expression giving the name of a column with sequence numbers or non-unique orderable values that indicate order within an actor group, if not present it will be ordered with all the data if no actor is available, used when widening the data. If both actor and time are specified, then the sequence order should be specified such that it determines the order of events within actor and each session.
time_threshold	An integer specifying the time threshold in seconds for creating new time-based sessions. Defaults to 900 seconds.
custom_format	A character string giving the format used to parse the time column.
is_unix_time	A logical value indicating whether the time column is in Unix time. The default is FALSE.
unix_time_unit	A character string giving the Unix time unit when is_unix_time is TRUE. The default is "seconds". Valid options are "seconds", "milliseconds", or "microseconds".
unused_fn	How to handle extra columns when pivoting to wide format. See tidyr::pivot_wider(). The default is to keep all columns and to use the first value.

Value

A tna_data object, which is a list with the following elements:

- long_data: The processed data in long format.
- sequence_data: The processed data on the sequences in wide format, with actions/events as different variables structured with sequences.
- meta_data: Other variables from the original data in wide format.
- statistics: A list containing summary statistics: total sessions, total actions, unique users, time range (if applicable), and top sessions and user by activities.

See Also

Other data: import_data(), import_onehot(), list_random_state_pools(), print.tna_data(), random_group_tna(), random_tna(), random_tna_mmm(), simulate.group_tna(), simulate.tna()

Examples

```
results <- prepare_data(
  group_regulation_long, actor = "Actor", time = "Time", action = "Action"
```

```
)
print(results$sequence_data)
print(results$meta_data)
print(results$statistics)

# Custom order column
data_ordered <- tibble::tibble(
   user = c("A", "A", "A", "B", "B", "C", "C", "C"),
   order = c(1, 2, 3, 1, 2, 1, 2, 3),
   action = c(
     "view", "click", "add_cart", "view",
     "checkout", "view", "click", "share"
   )
)
results_ordered <- prepare_data(
  data_ordered, actor = "user", order = "order", action = "action"
)
print(results_ordered$sequence_data)
print(results_ordered$meta_data)
print(results_ordered$statistics)

# No actor scenario leading to a single session
data_single_session <- tibble::tibble(
  action = c(
    "view", "click", "add_cart", "view",
    "checkout", "view", "click", "share"
   )
)
results_single <- prepare_data(data_single_session, action = "action")
print(results_single$sequence_data)
print(results_single$meta_data)
print(results_single$statistics)

# Multiple actors
data_multi_actor <- tibble::tibble(
  user = c("A", "A", "A", "A", "B", "B", "B", "B"),
  session = c(1, 1, 2, 2, 1, 1, 2, 2),
  action = c(
    "view", "click", "add_cart", "view",
    "checkout", "view", "click", "share"
  )
)
results_multi_actor <- prepare_data(
  data_multi_actor, actor = c("user", "session"), action = "action"
)
print(results_multi_actor$sequence_data)
print(results_multi_actor$meta_data)
print(results_multi_actor$statistics)
```

---

print.group_tna *Print a* group_tna *Object*

Description

Print a group_tna Object

Usage

```
## S3 method for class 'group_tna'
print(x, ...)
```

Arguments

x	A group_tna object.
...	Arguments passed to print.tna().

Value

x (invisibly).

See Also

Basic functions build_model(), hist.group_tna(), hist.tna(), plot.group_tna(), plot.tna(), plot_frequencies(), plot_frequencies.group_tna(), plot_mosaic(), plot_mosaic.group_tna(), plot_mosaic.tna_data(), print.summary.group_tna(), print.summary.tna(), print.tna(), summary.group_tna(), summary.tna(), tna-package

Examples

```
model <- group_model(engagement_mmm)
print(model)
```

---

print.group_tna_bootstrap

Print group_tna *Bootstrap Results*

Description

Print group_tna Bootstrap Results

Usage

```
## S3 method for class 'group_tna_bootstrap'
print(x, ...)
```

Arguments

x	A group_tna_bootstrap object.
...	Arguments passed to print.tna_bootstrap().

Value

x (invisibly).

See Also

Validation functions bootstrap(), deprune(), estimate_cs(), permutation_test(), permutation_test.group_tna(), plot.group_tna_bootstrap(), plot.group_tna_permutation(), plot.group_tna_stability(), plot.tna_bootstrap(), plot.tna_permutation(), plot.tna_reliability(), plot.tna_stability(), print.group_tna_permutation(), print.group_tna_stability(), print.summary.group_tna_bootstrap(), print.summary.tna_bootstrap(), print.tna_bootstrap(), print.tna_clustering(), print.tna_permutation(), print.tna_reliability(), print.tna_stability(), prune(), pruning_details(), reliability(), reprune(), summary.group_tna_bootstrap(), summary.tna_bootstrap()

Examples

```
model <- group_model(engagement_mmm)
# Low number of iteration for CRAN
boot <- bootstrap(model, iter = 10)
print(boot)
```

---

print.group_tna_centralities

Print Centrality Measures

Description

Print Centrality Measures

Usage

```
## S3 method for class 'group_tna_centralities'
print(x, ...)
```

Arguments

x	A group_tna_centralities object.
...	Ignored.

Value

x (invisibly).

See Also

Centrality measure functions betweenness_network(), centralities(), plot.group_tna_centralities(), plot.tna_centralities(), print.tna_centralities()

Examples

```
model <- group_model(engagement_mmm)
cm <- centralities(model)
print(cm)
```

`print.group_tna_cliques` *Print Found Cliques*

Description

Print Found Cliques

Usage

```
## S3 method for class 'group_tna_cliques'
print(x, ...)
```

Arguments

`x`	A `group_tna_cliques` object.
`...`	Arguments passed to `print.tna_cliques()`.

Value

`x` (invisibly).

See Also

Clique-related functions `cliques()`, `plot.group_tna_cliques()`, `plot.tna_cliques()`, `print.tna_cliques()`

Examples

```
model <- group_model(engagement_mmm)
cliq <- cliques(model, size = 2)
print(cliq)
```

print.group_tna_communities
Print Detected Communities

Description

Print Detected Communities

Usage

```
## S3 method for class 'group_tna_communities'
print(x, ...)
```

Arguments

x	A group_tna_communities object.
...	Arguments passed to print.tna_communities().

Value

x (invisibly).

See Also

Community detection functions communities(), plot.group_tna_communities(), plot.tna_communities(), print.tna_communities()

Examples

```
model <- group_model(engagement_mmm)
comm <- communities(model)
print(comm)
```

print.group_tna_permutation
Print Permutation Test Results

Description

Print Permutation Test Results

Usage

```
## S3 method for class 'group_tna_permutation'
print(x, ...)
```

Arguments

x	A group_tna_permutation object.
...	Arguments passed to print.tna_permutation().

Value

x (invisibly).

See Also

Validation functions bootstrap(), deprune(), estimate_cs(), permutation_test(), permutation_test.group_tna() plot.group_tna_bootstrap(), plot.group_tna_permutation(), plot.group_tna_stability(), plot.tna_bootstrap(), plot.tna_permutation(), plot.tna_reliability(), plot.tna_stability(), print.group_tna_bootstrap(), print.group_tna_stability(), print.summary.group_tna_bootstrap(), print.summary.tna_bootstrap(), print.tna_bootstrap(), print.tna_clustering(), print.tna_permutation(), print.tna_reliability(), print.tna_stability(), prune(), pruning_details(), reliability(), reprune(), summary.group_tna_bootstrap(), summary.tna_bootstrap()

Examples

```
model <- group_model(engagement_mmm)
# Small number of iterations for CRAN
perm <- permutation_test(model, iter = 20)
print(perm)
```

---

print.group_tna_stability

Print Centrality Stability Results

Description

Print Centrality Stability Results

Usage

```
## S3 method for class 'group_tna_stability'
print(x, ...)
```

Arguments

x	A group_tna_stability object.
...	Arguments passed to print.tna_stability().

Value

x (invisibly).

See Also

Validation functions bootstrap(), deprune(), estimate_cs(), permutation_test(), permutation_test.group_tna(), plot.group_tna_bootstrap(), plot.group_tna_permutation(), plot.group_tna_stability(), plot.tna_bootstrap(), plot.tna_permutation(), plot.tna_reliability(), plot.tna_stability(), print.group_tna_bootstrap(), print.group_tna_permutation(), print.summary.group_tna_bootstrap(), print.summary.tna_bootstrap(), print.tna_bootstrap(), print.tna_clustering(), print.tna_permutation(), print.tna_reliability(), print.tna_stability(), prune(), pruning_details(), reliability(), reprune(), summary.group_tna_bootstrap(), summary.tna_bootstrap()

Examples

```
model <- group_model(engagement_mmm)
# Low number of iterations for CRAN
stability <- estimate_cs(
  model,
  drop_prop = c(0.3, 0.5, 0.7, 0.9),
  iter = 10
)
print(stability)
```

---

print.summary.group_tna

Print a Summary of a Grouped Transition Network Analysis Model

---

Description

Print a Summary of a Grouped Transition Network Analysis Model

Usage

```
## S3 method for class 'summary.group_tna'
print(x, ...)
```

Arguments

x	A summary.group_tna object.
...	Arguments passed to the tibble print method

Value

x (invisibly).

See Also

Basic functions build_model(), hist.group_tna(), hist.tna(), plot.group_tna(), plot.tna(), plot_frequencies(), plot_frequencies.group_tna(), plot_mosaic(), plot_mosaic.group_tna(), plot_mosaic.tna_data(), print.group_tna(), print.summary.tna(), print.tna(), summary.group_tna(), summary.tna(), tna-package

Examples

```
model <- group_model(engagement_mmm)
print(summary(model))
```

`print.summary.group_tna_bootstrap` *Print a Bootstrap Summary for a Grouped Transition Network Model*

Description

Print a Bootstrap Summary for a Grouped Transition Network Model

Usage

```
## S3 method for class 'summary.group_tna_bootstrap'
print(x, ...)
```

Arguments

`x`	A `summary.group_tna_bootstrap` object.
`...`	Arguments passed to the generic `print` method.

Value

`x` (invisibly).

See Also

Validation functions `bootstrap()`, `deprune()`, `estimate_cs()`, `permutation_test()`, `permutation_test.group_tna()`, `plot.group_tna_bootstrap()`, `plot.group_tna_permutation()`, `plot.group_tna_stability()`, `plot.tna_bootstrap()`, `plot.tna_permutation()`, `plot.tna_reliability()`, `plot.tna_stability()`, `print.group_tna_bootstrap()`, `print.group_tna_permutation()`, `print.group_tna_stability()`, `print.summary.tna_bootstrap()`, `print.tna_bootstrap()`, `print.tna_clustering()`, `print.tna_permutation()`, `print.tna_reliability()`, `print.tna_stability()`, `prune()`, `pruning_details()`, `reliability()`, `reprune()`, `summary.group_tna_bootstrap()`, `summary.tna_bootstrap()`

Examples

```
model <- group_model(engagement_mmm)
# Low number of iteration for CRAN
boot <- bootstrap(model, iter = 10)
print(summary(boot))
```

print.summary.tna *Print a TNA Summary*

Description

Print a TNA Summary

Usage

```
## S3 method for class 'summary.tna'
print(x, ...)
```

Arguments

x	A summary.tna object.
...	Ignored.

Value

A summary.tna object (invisibly) containing the TNA model network metrics and values.

See Also

Basic functions build_model(), hist.group_tna(), hist.tna(), plot.group_tna(), plot.tna(), plot_frequencies(), plot_frequencies.group_tna(), plot_mosaic(), plot_mosaic.group_tna(), plot_mosaic.tna_data(), print.group_tna(), print.summary.group_tna(), print.tna(), summary.group_tna(), summary.tna(), tna-package

Examples

```
model <- tna(group_regulation)
print(summary(model))
```

print.summary.tna_bootstrap *Print a Bootstrap Summary*

Description

Print a Bootstrap Summary

Usage

```
## S3 method for class 'summary.tna_bootstrap'
print(x, ...)
```

Arguments

x	A summary.tna_bootstrap object.
...	Arguments passed to the generic print method.

Value

A summary.tna_bootstrap object (invisibly) containing the weight, estimated p-value and confidence interval of each edge.

See Also

Validation functions bootstrap(), deprune(), estimate_cs(), permutation_test(), permutation_test.group_tna() plot.group_tna_bootstrap(), plot.group_tna_permutation(), plot.group_tna_stability(), plot.tna_bootstrap(), plot.tna_permutation(), plot.tna_reliability(), plot.tna_stability(), print.group_tna_bootstrap(), print.group_tna_permutation(), print.group_tna_stability(), print.summary.group_tna_bootstrap(), print.tna_bootstrap(), print.tna_clustering(), print.tna_permutation(), print.tna_reliability(), print.tna_stability(), prune(), pruning_details(), reliability(), reprune(), summary.group_tna_bootstrap(), summary.tna_bootstrap()

Examples

```
model <- tna(group_regulation)
# Small number of iterations for CRAN
boot <- bootstrap(model, iter = 10)
print(summary(boot))
```

---

print.tna *Print a* tna *Object*

Description

Print a tna Object

Usage

```
## S3 method for class 'tna'
print(x, generic = FALSE, ...)
```

Arguments

x	A tna object.
generic	A logical value. If TRUE, use generic print method instead. Defaults to FALSE.
...	Additional arguments passed to the generic print methods.

Value

The tna object passed as argument x (invisibly).

See Also

Basic functions build_model(), hist.group_tna(), hist.tna(), plot.group_tna(), plot.tna(), plot_frequencies(), plot_frequencies.group_tna(), plot_mosaic(), plot_mosaic.group_tna(), plot_mosaic.tna_data(), print.group_tna(), print.summary.group_tna(), print.summary.tna(), summary.group_tna(), summary.tna(), tna-package

Examples

```
model <- tna(group_regulation)
print(model)
```

---

print.tna_bootstrap *Print Bootstrap Results*

Description

Print Bootstrap Results

Usage

```
## S3 method for class 'tna_bootstrap'
print(x, digits = getOption("digits"), type = "both", ...)
```

Arguments

x	A tna_bootstrap object.
digits	An integer giving the minimal number of *significant* digits to print.
type	A character vector giving the type of edges to print. The default option "both" prints both statistically significant and non-significant edges, "sig" prints only significant edges, and "nonsig" prints only the non-significant edges.
...	Ignored.

Value

x (invisibly).

See Also

Validation functions bootstrap(), deprune(), estimate_cs(), permutation_test(), permutation_test.group_tna() plot.group_tna_bootstrap(), plot.group_tna_permutation(), plot.group_tna_stability(), plot.tna_bootstrap(), plot.tna_permutation(), plot.tna_reliability(), plot.tna_stability(), print.group_tna_bootstrap(), print.group_tna_permutation(), print.group_tna_stability(), print.summary.group_tna_bootstrap(), print.summary.tna_bootstrap(), print.tna_clustering(), print.tna_permutation(), print.tna_reliability(), print.tna_stability(), prune(), pruning_details(), reliability(), reprune(), summary.group_tna_bootstrap(), summary.tna_bootstrap()

Examples

```
model <- tna(group_regulation)
# Small number of iterations for CRAN
boot <- bootstrap(model, iter = 10)
print(boot)
```

`print.tna_centralities` *Print Centrality Measures*

Description

Print Centrality Measures

Usage

```
## S3 method for class 'tna_centralities'
print(x, ...)
```

Arguments

`x`	A `centralities` object.
`...`	Ignored.

Value

`x` (invisibly).

See Also

Centrality measure functions `betweenness_network()`, `centralities()`, `plot.group_tna_centralities()`, `plot.tna_centralities()`, `print.group_tna_centralities()`

Examples

```
model <- tna(group_regulation)
cm <- centralities(model)
print(cm)
```

`print.tna_cliques` *Print Found Cliques of a TNA Network*

Description

Print Found Cliques of a TNA Network

Usage

```
## S3 method for class 'tna_cliques'
print(x, n = 6, first = 1, digits = getOption("digits"), ...)
```

Arguments

`x`	A `tna_cliques` object.
`n`	An `integer` defining the maximum number of cliques to show. The defaults is `6`.
`first`	An `integer` giving the index of the first clique to show. The default index is `1`.
`digits`	An `integer` giving the minimal number of *significant* digits to print.
`...`	Ignored.

Value

`x` (invisibly).

See Also

Clique-related functions `cliques()`, `plot.group_tna_cliques()`, `plot.tna_cliques()`, `print.group_tna_cliques()`

Examples

```
model <- tna(group_regulation)
cliq <- cliques(model, size = 2)
print(cliq)
```

`print.tna_clustering` *Print the Results of Clustering*

Description

Print the Results of Clustering

Usage

```
## S3 method for class 'tna_clustering'
print(x, ...)
```

Arguments

`x`	A `tna_clustering` object.
`...`	Additional arguments passed to the generic print method.

Value

`x` (invisibly).

See Also

Validation functions `bootstrap()`, `deprune()`, `estimate_cs()`, `permutation_test()`, `permutation_test.group_tna()`, `plot.group_tna_bootstrap()`, `plot.group_tna_permutation()`, `plot.group_tna_stability()`, `plot.tna_bootstrap()`, `plot.tna_permutation()`, `plot.tna_reliability()`, `plot.tna_stability()`, `print.group_tna_bootstrap()`, `print.group_tna_permutation()`, `print.group_tna_stability()`, `print.summary.group_tna_bootstrap()`, `print.summary.tna_bootstrap()`, `print.tna_bootstrap()`, `print.tna_permutation()`, `print.tna_reliability()`, `print.tna_stability()`, `prune()`, `pruning_details()`, `reliability()`, `reprune()`, `summary.group_tna_bootstrap()`, `summary.tna_bootstrap()`

Examples

```
data <- data.frame(
  T1 = c("A", "B", "A", "C", "A", "B"),
  T2 = c("B", "A", "B", "A", "C", "A"),
  T3 = c("C", "C", "A", "B", "B", "C")
)

# PAM clustering with optimal string alignment (default)
result <- cluster_sequences(data, k = 2)
print(result)
```

print.tna_communities *Print Detected Communities*

Description

Print Detected Communities

Usage

```
## S3 method for class 'tna_communities'
print(x, ...)
```

Arguments

x	A tna_communities object.
...	Additional arguments passed to the generic print method.

Value

x (invisibly).

See Also

Community detection functions communities(), plot.group_tna_communities(), plot.tna_communities(), print.group_tna_communities()

Examples

```
model <- tna(group_regulation)
comm <- communities(model)
print(comm)
```

print.tna_comparison *Print Comparison Results*

Description

Print Comparison Results

Usage

```
## S3 method for class 'tna_comparison'
print(x, ...)
```

Arguments

x	A tna_comparison object.
...	Additional arguments passed to the tibble print method.

Value

x (invisibly).

See Also

Model comparison functions compare(), compare.group_tna(), compare_sequences(), plot.tna_comparison(), plot.tna_sequence_comparison(), plot_compare(), plot_compare.group_tna(), print.tna_sequence_comparison

Examples

```
model_x <- tna(group_regulation[1:200, ])
model_y <- tna(group_regulation[1001:1200, ])
comp <- compare(model_x, model_y)
print(comp)
```

---

print.tna_data *Print a TNA Data Object*

Description

Print a TNA Data Object

Usage

```
## S3 method for class 'tna_data'
print(x, data = "sequence", ...)
```

Arguments

x	A tna_data object.
data	A character string that defines the data to be printed tibble. Accepts either "sequence" (default) for wide format sequence data, "meta", for the wide format metadata, or "long" for the long format data.
...	Arguments passed to the tibble print method.

Value

x (invisibly).

See Also

Other data: import_data(), import_onehot(), list_random_state_pools(), prepare_data(), random_group_tna(), random_tna(), random_tna_mmm(), simulate.group_tna(), simulate.tna()

Examples

```
res <- prepare_data(group_regulation_long, action = "Action", actor = "Actor",
time = "Time")
print(res, which = "sequence")
print(res, which = "meta")
print(res, which = "long")
```

---

print.tna_permutation *Print Permutation Test Results*

Description

Print Permutation Test Results

Usage

```
## S3 method for class 'tna_permutation'
print(x, ...)
```

Arguments

x	A tna_permutation object.
...	Additional arguments passed to the tibble print method.

Value

x (invisibly).

See Also

Validation functions bootstrap(), deprune(), estimate_cs(), permutation_test(), permutation_test.group_tna() plot.group_tna_bootstrap(), plot.group_tna_permutation(), plot.group_tna_stability(), plot.tna_bootstrap(), plot.tna_permutation(), plot.tna_reliability(), plot.tna_stability(), print.group_tna_bootstrap(), print.group_tna_permutation(), print.group_tna_stability(), print.summary.group_tna_bootstrap(), print.summary.tna_bootstrap(), print.tna_bootstrap(), print.tna_clustering(), print.tna_reliability(), print.tna_stability(), prune(), pruning_details(), reliability(), reprune(), summary.group_tna_bootstrap(), summary.tna_bootstrap()

Examples

```
model_x <- tna(group_regulation[1:200, ])
model_y <- tna(group_regulation[1001:1200, ])
# Small number of iterations for CRAN
perm <- permutation_test(model_x, model_y, iter = 20)
print(perm)
```

---

print.tna_reliability *Print Reliability Analysis Results*

Description

Print Reliability Analysis Results

Usage

```
## S3 method for class 'tna_reliability'
print(x, summary_metrics, ...)
```

Arguments

x	A tna_reliability object.
summary_metrics	A character vector of metrics to display.
...	Arguments passed to the generic print method.

Value

x (invisibly).

See Also

Validation functions bootstrap(), deprune(), estimate_cs(), permutation_test(), permutation_test.group_tna(), plot.group_tna_bootstrap(), plot.group_tna_permutation(), plot.group_tna_stability(), plot.tna_bootstrap(), plot.tna_permutation(), plot.tna_reliability(), plot.tna_stability(), print.group_tna_bootstrap(), print.group_tna_permutation(), print.group_tna_stability(), print.summary.group_tna_bootstrap(), print.summary.tna_bootstrap(), print.tna_bootstrap(), print.tna_clustering(), print.tna_permutation(), print.tna_stability(), prune(), pruning_details(), reliability(), reprune(), summary.group_tna_bootstrap(), summary.tna_bootstrap()

Examples

```
# Small number of iterations for CRAN
model <- tna(engagement)
rel <- reliability(model, iter = 20)
print(rel)
```

print.tna_sequence_comparison *Print a Comparison of Sequences*

Description

Print a Comparison of Sequences

Usage

```
## S3 method for class 'tna_sequence_comparison'
print(x, ...)
```

Arguments

x	A tna_sequence_comparison object.
...	Arguments passed to the generic print method.

Value

x (invisibly).

See Also

Model comparison functions compare(), compare.group_tna(), compare_sequences(), plot.tna_comparison(), plot.tna_sequence_comparison(), plot_compare(), plot_compare.group_tna(), print.tna_comparison()

Examples

```
idx <- c(1:500, 1001:1500)
group <- c(rep("High", 500), rep("Low", 500))
comp <- compare_sequences(group_regulation[idx, ], group)
print(comp)
```

print.tna_stability *Print Centrality Stability Results*

Description

Print Centrality Stability Results

Usage

```
## S3 method for class 'tna_stability'
print(x, ...)
```

Arguments

x	A tna_stability object.
...	Additional arguments passed to the generic print method.

Value

x (invisibly).

See Also

Validation functions bootstrap(), deprune(), estimate_cs(), permutation_test(), permutation_test.group_tna() plot.group_tna_bootstrap(), plot.group_tna_permutation(), plot.group_tna_stability(), plot.tna_bootstrap(), plot.tna_permutation(), plot.tna_reliability(), plot.tna_stability(), print.group_tna_bootstrap(), print.group_tna_permutation(), print.group_tna_stability(), print.summary.group_tna_bootstrap(), print.summary.tna_bootstrap(), print.tna_bootstrap(), print.tna_clustering(), print.tna_permutation(), print.tna_reliability(), prune(), pruning_details(), reliability(), reprune(), summary.group_tna_bootstrap(), summary.tna_bootstrap()

Examples

```
model <- tna(group_regulation)
# Small number of iterations and drop proportions for CRAN
cs <- estimate_cs(
  model,
  measures = c("InStrength", "OutStrength"),
  drop_prop = seq(0.3, 0.9, by = 0.2),
  iter = 10
)
print(cs)
```

---

prune *Prune a Transition Network based on Transition Probabilities*

Description

Prunes a network represented by a tna object by removing edges based on a specified threshold, lowest percent of non-zero edge weights, or the disparity filter algorithm (Serrano et al., 2009). It ensures the network remains weakly connected.

Prunes a network represented by a tna object by removing edges based on a specified threshold, lowest percent of non-zero edge weights, or the disparity filter algorithm (Serrano et al., 2009). It ensures the network remains weakly connected.

Usage

```
prune(x, ...)

## S3 method for class 'tna'
prune(
  x,
  method = "threshold",
  threshold = 0.1,
  lowest = 0.05,
  level = 0.5,
  boot = NULL,
  ...
)

## S3 method for class 'group_tna'
prune(x, ...)
```

Arguments

x	An object of class tna or group_tna
...	Arguments passed to bootstrap() when using method = "bootstrap" and when a tna_bootstrap is not supplied.
method	A character string describing the pruning method. The available options are "threshold", "lowest", "bootstrap" and "disparity", corresponding to the methods listed in Details. The default is "threshold".
threshold	A numeric value specifying the edge weight threshold. Edges with weights below or equal to this threshold will be considered for removal.
lowest	A numeric value specifying the lowest percentage of non-zero edges. This percentage of edges with the lowest weights will be considered for removal. The default is 0.05.
level	A numeric value representing the significance level for the disparity filter. Defaults to 0.5.
boot	A tna_bootstrap object to be used for pruning with method "boot". The method argument is ignored if this argument is supplied.

Value

A pruned tna or group_tna object. Details on the pruning can be viewed with pruning_details(). The original model can be restored with deprune().

See Also

Validation functions bootstrap(), deprune(), estimate_cs(), permutation_test(), permutation_test.group_tna() plot.group_tna_bootstrap(), plot.group_tna_permutation(), plot.group_tna_stability(), plot.tna_bootstrap(), plot.tna_permutation(), plot.tna_reliability(), plot.tna_stability(), print.group_tna_bootstrap(), print.group_tna_permutation(), print.group_tna_stability(), print.summary.group_tna_bootstrap(), print.summary.tna_bootstrap(), print.tna_bootstrap(),

print.tna_clustering(), print.tna_permutation(), print.tna_reliability(), print.tna_stability(), pruning_details(), reliability(), reprune(), summary.group_tna_bootstrap(), summary.tna_bootstrap()

Validation functions bootstrap(), deprune(), estimate_cs(), permutation_test(), permutation_test.group_tna() plot.group_tna_bootstrap(), plot.group_tna_permutation(), plot.group_tna_stability(), plot.tna_bootstrap(), plot.tna_permutation(), plot.tna_reliability(), plot.tna_stability(), print.group_tna_bootstrap(), print.group_tna_permutation(), print.group_tna_stability(), print.summary.group_tna_bootstrap(), print.summary.tna_bootstrap(), print.tna_bootstrap(), print.tna_clustering(), print.tna_permutation(), print.tna_reliability(), print.tna_stability(), pruning_details(), reliability(), reprune(), summary.group_tna_bootstrap(), summary.tna_bootstrap()

Examples

```
model <- tna(group_regulation)
pruned_threshold <- prune(model, method = "threshold", threshold = 0.1)
pruned_percentile <- prune(model, method = "lowest", lowest = 0.05)
pruned_disparity <- prune(model, method = "disparity", level = 0.5)
```

---

pruning_details *Print Detailed Information on the Pruning Results*

Description

Print Detailed Information on the Pruning Results

Usage

```
pruning_details(x, ...)

## S3 method for class 'tna'
pruning_details(x, ...)

## S3 method for class 'group_tna'
pruning_details(x, ...)
```

Arguments

x	A tna or group_tna object.
...	Ignored.

Value

A data.frame containing the removed edges if x is a tna object, or a list of data.frame objects in the case of group_tna object.

See Also

Validation functions bootstrap(), deprune(), estimate_cs(), permutation_test(), permutation_test.group_tna(), plot.group_tna_bootstrap(), plot.group_tna_permutation(), plot.group_tna_stability(), plot.tna_bootstrap(), plot.tna_permutation(), plot.tna_reliability(), plot.tna_stability(), print.group_tna_bootstrap(), print.group_tna_permutation(), print.group_tna_stability(), print.summary.group_tna_bootstrap(), print.summary.tna_bootstrap(), print.tna_bootstrap(), print.tna_clustering(), print.tna_permutation(), print.tna_reliability(), print.tna_stability(), prune(), reliability(), reprune(), summary.group_tna_bootstrap(), summary.tna_bootstrap()

Examples

```
model <- tna(group_regulation)
pruned_threshold <- prune(model, method = "threshold", threshold = 0.1)
pruning_details(pruned_threshold)
```

---

random_group_tna *Build a Random Group Transition Network Analysis Model*

Description

Construct a fully-functional group_tna object from synthetic parameters. Each group receives its own randomly drawn transition matrix and initial probabilities over a shared alphabet, so groups have heterogeneous dynamics by default. Per-group overrides allow custom group sizes, sparsity, stickiness, or hand-supplied transition matrices.

Usage

```
random_group_tna(
  n_groups = NULL,
  group_names = NULL,
  n_states = NULL,
  states = NULL,
  category = NULL,
  alpha = NULL,
  diag_boost = NULL,
  n_sequences = NULL,
  seq_length = NULL,
  type = "relative",
  per_group = NULL,
  seed = NULL
)
```

Arguments

n_groups	An integer giving the number of groups. If NULL (the default), a value is drawn from 2:4 on each call.

group_names — An optional character vector of group names of length n_groups. Defaults to "Group 1", "Group 2", ...

n_states — An integer >= 2 giving the number of states. If NULL (the default), a value is drawn from 7:11 on each call.

states — An optional character vector of state labels of length at least n_states. The first n_states are used. If NULL (the default), labels are taken from category or auto-picked from a built-in pool that fits n_states.

category — An optional character string naming a built-in label pool. Available pools are returned by list_random_state_pools(). When NULL (the default), a pool whose size is at least n_states is sampled at random. Ignored when states is supplied.

alpha — A positive numeric Dirichlet concentration parameter. Small values (e.g. 0.3) produce sparse, peaked transitions; large values (e.g. 5) produce near-uniform transitions. If NULL (the default), a value is drawn from Uniform(0.5, 1.0) on each call.

diag_boost — A non-negative numeric added to the diagonal of the transition matrix before re-normalising rows. Larger values make states "stickier" (more self-transitions). If NULL (the default), a value is drawn from Uniform(1.5, 3.0) on each call.

n_sequences — An integer giving the number of sequences to simulate from the true parameters. If NULL (the default), a value is drawn from 500:800 on each call.

seq_length — An integer giving the length of each simulated sequence. If NULL (the default), a value is drawn from 6:20 on each call.

type — A character string giving the model type. One of "relative" (the default), "frequency", "co-occurrence", "attention".

per_group — An optional list of length n_groups. Each element is itself a list of overrides applied to that group only. Recognised override names: alpha, diag_boost, trans_matrix, init_probs, n_sequences, seq_length. Unset entries fall back to the top-level defaults (which may themselves be drawn at random when NULL).

seed — An integer random seed for reproducibility, or NULL (the default) for fresh randomness on every call.

Value

A group_tna object.

See Also

Other data: import_data(), import_onehot(), list_random_state_pools(), prepare_data(), print.tna_data(), random_tna(), random_tna_mmm(), simulate.group_tna(), simulate.tna()

Examples

```
# Fresh random group model on every call
model <- random_group_tna()
```

```
# Explicit two-group engagement demo with per-group differences
model <- random_group_tna(
  n_groups  = 2,
  n_states  = 3,
  category  = "engagement",
  per_group = list(
    list(n_sequences = 400, diag_boost = 3),
    list(n_sequences = 100, alpha = 0.3)
  ),
  seed = 42
)
```

`random_tna` *Build a Random Transition Network Analysis Model*

Description

Construct a fully-functional `tna` object from synthetic parameters without needing pre-existing sequence data. Random transition probabilities are drawn from a Dirichlet distribution, sequences are simulated from the resulting model, and a canonical `tna` object is fitted on those sequences.

Calling `random_tna()` with no arguments returns a fresh, sticky network with a coherent alphabet, drawn fresh on every call.

Usage

```
random_tna(
  n_states = NULL,
  states = NULL,
  category = NULL,
  alpha = NULL,
  diag_boost = NULL,
  trans_matrix = NULL,
  init_probs = NULL,
  n_sequences = NULL,
  seq_length = NULL,
  type = "relative",
  return_params = FALSE,
  seed = NULL
)
```

Arguments

`n_states`	An `integer` >= 2 giving the number of states. If `NULL` (the default), a value is drawn from `7:11` on each call.
`states`	An optional `character` vector of state labels of length at least `n_states`. The first `n_states` are used. If `NULL` (the default), labels are taken from `category` or auto-picked from a built-in pool that fits `n_states`.

category	An optional character string naming a built-in label pool. Available pools are returned by list_random_state_pools(). When NULL (the default), a pool whose size is at least n_states is sampled at random. Ignored when states is supplied.
alpha	A positive numeric Dirichlet concentration parameter. Small values (e.g. 0.3) produce sparse, peaked transitions; large values (e.g. 5) produce near-uniform transitions. If NULL (the default), a value is drawn from Uniform(0.5, 1.0) on each call.
diag_boost	A non-negative numeric added to the diagonal of the transition matrix before re-normalising rows. Larger values make states "stickier" (more self-transitions). If NULL (the default), a value is drawn from Uniform(1.5, 3.0) on each call.
trans_matrix	An optional square numeric matrix of transition probabilities. When supplied, alpha and diag_boost are ignored for the transition matrix. Rows are renormalised to sum to one.
init_probs	An optional numeric vector of initial state probabilities. Renormalised to sum to one. If NULL, drawn from a Dirichlet on the same alphabet.
n_sequences	An integer giving the number of sequences to simulate from the true parameters. If NULL (the default), a value is drawn from 500:800 on each call.
seq_length	An integer giving the length of each simulated sequence. If NULL (the default), a value is drawn from 6:20 on each call.
type	A character string giving the model type. One of "relative" (the default), "frequency", "co-occurrence", "attention".
return_params	A logical. If TRUE, returns a list with the fitted model and the ground-truth parameters used to generate it. Default is FALSE.
seed	An integer random seed for reproducibility, or NULL (the default) for fresh randomness on every call.

Value

A tna object, or a list with elements model, trans_matrix, init_probs, sequences, labels, and category when return_params = TRUE.

See Also

Other data: import_data(), import_onehot(), list_random_state_pools(), prepare_data(), print.tna_data(), random_group_tna(), random_tna_mmm(), simulate.group_tna(), simulate.tna()

Examples

```
# Fresh random model on every call
model <- random_tna()

# Explicit small-state demo using the engagement pool
model <- random_tna(n_states = 3, category = "engagement")

# Reproducible model
model <- random_tna(seed = 42)
```

```
# Recover the ground-truth parameters
out <- random_tna(seed = 7, return_params = TRUE)
out$trans_matrix
```

`random_tna_mmm` *Build a Random Mixture Markov Model Object*

Description

Construct a synthetic `tna_mmm` object that mirrors the *structure* a fitted seqHMM mixed Markov model exposes to tna without depending on seqHMM at runtime. The returned object can be passed to `group_model()` (dispatching via `group_model.tna_mmm`) and to `mmm_stats()` (dispatching via `mmm_stats.tna_mmm`).

Real seqHMM `mhmm` objects continue to dispatch via the original `*.mhmm` methods.

Usage

```
random_tna_mmm(
  n_clusters = NULL,
  n_states = NULL,
  states = NULL,
  category = NULL,
  alpha = NULL,
  diag_boost = NULL,
  n_sequences = NULL,
  seq_length = NULL,
  n_covariates = 1L,
  seed = NULL
)
```

Arguments

`n_clusters`	An `integer` giving the number of mixture clusters. If `NULL` (the default), drawn from `2:4` on each call.
`n_states`	An `integer` >= 2 giving the number of states. If `NULL` (the default), a value is drawn from `7:11` on each call.
`states`	An optional `character` vector of state labels of length at least `n_states`. The first `n_states` are used. If `NULL` (the default), labels are taken from `category` or auto-picked from a built-in pool that fits `n_states`.
`category`	An optional `character` string naming a built-in label pool. Available pools are returned by `list_random_state_pools()`. When `NULL` (the default), a pool whose size is at least `n_states` is sampled at random. Ignored when `states` is supplied.

alpha	A positive numeric Dirichlet concentration parameter. Small values (e.g. 0.3) produce sparse, peaked transitions; large values (e.g. 5) produce near-uniform transitions. If NULL (the default), a value is drawn from Uniform(0.5, 1.0) on each call.
diag_boost	A non-negative numeric added to the diagonal of the transition matrix before re-normalising rows. Larger values make states "stickier" (more self-transitions). If NULL (the default), a value is drawn from Uniform(1.5, 3.0) on each call.
n_sequences	An integer giving the number of sequences to simulate from the true parameters. If NULL (the default), a value is drawn from 500:800 on each call.
seq_length	An integer giving the length of each simulated sequence. If NULL (the default), a value is drawn from 6:20 on each call.
n_covariates	An integer >= 1 giving the number of regression variables (including the intercept) used to predict cluster membership. Default is 1 (intercept only). When > 1, additional rows are added to the coefficient matrix.
seed	An integer random seed for reproducibility, or NULL (the default) for fresh randomness on every call.

Value

An object of class tna_mmm containing fields observations, transition_probs, initial_probs, coefficients, vcov, most_probable_cluster, cluster_names, state_names, n_clusters, n_states, n_sequences, n_covariates.

See Also

Other data: import_data(), import_onehot(), list_random_state_pools(), prepare_data(), print.tna_data(), random_group_tna(), random_tna(), simulate.group_tna(), simulate.tna()

Examples

```
model <- random_tna_mmm(seed = 1)
mmm_stats(model)
grp <- group_model(model)
```

---

reliability *Assess Model Reliability*

Description

Performs reliability analysis and outputs a concise summary of key metrics. The results can also be visualized.

Usage

```
reliability(x, ...)

## S3 method for class 'tna'
reliability(
  x,
  types = "relative",
  split = 0.5,
  iter = 1000,
  scaling = "none",
  ...
)
```

Arguments

`x`	A `tna` object.
`...`	Ignored.
`types`	A `character` vector giving the model types to fit. See `build_model()` for available options.
`split`	A `numeric` value between `0` and `1` specifying the proportion of data for the split. The default is `0.5` for an even split.
`iter`	An `integer` specifying number of iterations (splits). The default is `1000`.
`scaling`	See `compare()`.

Value

A `tna_reliability` object.

See Also

Validation functions `bootstrap()`, `deprune()`, `estimate_cs()`, `permutation_test()`, `permutation_test.group_tna()`, `plot.group_tna_bootstrap()`, `plot.group_tna_permutation()`, `plot.group_tna_stability()`, `plot.tna_bootstrap()`, `plot.tna_permutation()`, `plot.tna_reliability()`, `plot.tna_stability()`, `print.group_tna_bootstrap()`, `print.group_tna_permutation()`, `print.group_tna_stability()`, `print.summary.group_tna_bootstrap()`, `print.summary.tna_bootstrap()`, `print.tna_bootstrap()`, `print.tna_clustering()`, `print.tna_permutation()`, `print.tna_reliability()`, `print.tna_stability()`, `prune()`, `pruning_details()`, `reprune()`, `summary.group_tna_bootstrap()`, `summary.tna_bootstrap()`

Examples

```
# Small number of iterations for CRAN
model <- tna(engagement)
rel <- reliability(model, iter = 20)
```

rename_groups *Rename Groups*

Description

Rename Groups

Usage

```
rename_groups(x, new_names)
```

Arguments

x	A group_tna object.
new_names	A character vector containing one name per cluster.

Value

A renamed group_tna object.

See Also

Cluster-related functions communities(), group_model(), mmm_stats()

Examples

```
model <- group_model(engagement_mmm)
model_renamed <- rename_groups(model, c("A", "B", "C"))
```

reprune *Restore Previous Pruning of a Transition Network Analysis Model*

Description

Restore Previous Pruning of a Transition Network Analysis Model

Usage

```
reprune(x, ...)

## S3 method for class 'group_tna'
reprune(x, ...)
```

Arguments

x	A tna or group_tna object.
...	Ignored.

Value

A tna or group_tna object that has not been pruned. The previous pruning result can be reactivated with reprune().

See Also

Validation functions bootstrap(), deprune(), estimate_cs(), permutation_test(), permutation_test.group_tna() plot.group_tna_bootstrap(), plot.group_tna_permutation(), plot.group_tna_stability(), plot.tna_bootstrap(), plot.tna_permutation(), plot.tna_reliability(), plot.tna_stability(), print.group_tna_bootstrap(), print.group_tna_permutation(), print.group_tna_stability(), print.summary.group_tna_bootstrap(), print.summary.tna_bootstrap(), print.tna_bootstrap(), print.tna_clustering(), print.tna_permutation(), print.tna_reliability(), print.tna_stability(), prune(), pruning_details(), reliability(), summary.group_tna_bootstrap(), summary.tna_bootstrap()

Examples

```
model <- tna(group_regulation)
pruned_model <- prune(model, method = "threshold", threshold = 0.1)
depruned_model <- deprune(pruned_model) # restore original model
repruned_model <- reprune(depruned_model) # reapply the previous pruning
```

---

simulate *Simulate Method (Re-export of stats::simulate)*

Description

The stats::simulate() generic is re-exported so that the simulate.tna() and simulate.group_tna() methods provided by this package are dispatched correctly when the user only loads tna. See the method-specific help pages for argument and return details.

Usage

```
simulate(object, nsim = 1, seed = NULL, ...)
```

Arguments

object	A statistical model object. The tna package provides methods for objects of class tna and group_tna.
nsim	An integer giving the number of sequences to simulate.
seed	An integer random seed for reproducibility, or NULL.
...	Further arguments passed to the dispatched method.

Value

The value returned by the dispatched method (a `data.frame` of simulated sequences for `tna` and `group_tna` objects). See the method-specific help pages for details.

Examples

```
model <- tna(group_regulation)
sim <- simulate(model, nsim = 5, max_len = 10)
```

---

simulate.group_tna *Simulate Data from a Group Transition Network Analysis Model*

Description

Simulate Data from a Group Transition Network Analysis Model

Usage

```
## S3 method for class 'group_tna'
simulate(
  object,
  nsim = 1,
  seed = NULL,
  max_len = 100L,
  na_range = c(0L, 0L),
  zero_row = "self",
  format = "wide",
  ...
)
```

Arguments

`object`	A `group_tna` object. The edge weights must be transition probabilities or frequencies, i.e., the model must have `type = "relative"` or `type = "frequency"`.
`nsim`	An `integer` vector giving the number of sequences to simulate per group. If a single integer is provided, the same number of sequences is generated per each group. The default is 1.
`seed`	an object specifying if and how the random number generator should be initialized ('seeded'). For the `"lm"` method, either `NULL` or an integer that will be used in a call to `set.seed` before simulating the response vectors. If set, the value is saved as the `"seed"` attribute of the returned value. The default, `NULL` will not change the random generator state, and return `.Random.seed` as the `"seed"` attribute, see 'Value'.

max_len	An integer vector giving the maximum length of the simulated sequences per group. When no missing values are generated, this is the length of all simulated sequences. If a single integer is provided, the maximum length is the same for each group.
na_range	An integer vector of length 2 giving the minimum and maximum number of missing values to generate for each sequence. The number of missing values is drawn uniformly from this range. If both values are zero (the default), no missing values are generated.
zero_row	A character string describing how to process zero rows in the weight matrix. The option "self" (the default) assigns probability 1 to the corresponding state (self loop) and option "uniform" assigns a uniform distribution.
format	A character string indicating whether the data should be returned in wide or long format.
...	Ignored.

Value

A data.frame of the simulated sequence data.

See Also

Other data: import_data(), import_onehot(), list_random_state_pools(), prepare_data(), print.tna_data(), random_group_tna(), random_tna(), random_tna_mmm(), simulate.tna()

Examples

```
model <- group_tna(
  group_regulation,
  group = rep(c("High", "Low"), each = 1000)
)
sim <- simulate(model, nsim = 10, max_len = 10)
```

---

simulate.tna *Simulate Data from a Transition Network Analysis Model*

Description

Simulate Data from a Transition Network Analysis Model

Usage

```
## S3 method for class 'tna'
simulate(
  object,
  nsim = 1,
  seed = NULL,
```

```
  max_len = 100L,
  na_range = c(0L, 0L),
  zero_row = "self",
  format = "wide",
  ...
)
```

Arguments

object	A tna object. The edge weights must be transition probabilities or frequencies, i.e., the model must have type = "relative" or type = "frequency".
nsim	An integer giving the number of sequences to simulate. The default is 1.
seed	an object specifying if and how the random number generator should be initialized ('seeded'). For the "lm" method, either NULL or an integer that will be used in a call to set.seed before simulating the response vectors. If set, the value is saved as the "seed" attribute of the returned value. The default, NULL will not change the random generator state, and return .Random.seed as the "seed" attribute, see 'Value'.
max_len	An integer giving the maximum length of the simulated sequences. When no missing values are generated, this is the length of all simulated sequences.
na_range	An integer vector of length 2 giving the minimum and maximum number of missing values to generate for each sequence. The number of missing values is drawn uniformly from this range. If both values are zero (the default), no missing values are generated.
zero_row	A character string describing how to process zero rows in the weight matrix. The option "self" (the default) assigns probability 1 to the corresponding state (self loop) and option "uniform" assigns a uniform distribution.
format	A character string indicating whether the data should be returned in wide or long format.
...	Ignored.

Value

A data.frame of the simulated sequence data.

See Also

Other data: import_data(), import_onehot(), list_random_state_pools(), prepare_data(), print.tna_data(), random_group_tna(), random_tna(), random_tna_mmm(), simulate.group_tna()

Examples

```
model <- tna(group_regulation)
sim <- simulate(model, nsim = 10, max_len = 10)
```

sna — *Build a Social Network Analysis Model*

Description

Build a Social Network Analysis Model

Usage

```
sna(x, aggregate = sum, ...)
```

Arguments

x	A data.frame or a matrix with three columns: the first two representing the states and the third giving the weights.
aggregate	A function to use for aggregating the weights. The default is sum().
...	Additional arguments passed to aggregate.

Value

A tna object representing the model.

Examples

```
set.seed(123)
d <- data.frame(
  from = sample(LETTERS[1:4], 100, replace = TRUE),
  to = sample(LETTERS[1:4], 100, replace = TRUE),
  weight = rexp(100)
)
model <- sna(d)
```

summary.group_tna — *Calculate Summary of Network Metrics for a grouped Transition Network*

Description

This function calculates a variety of network metrics for a tna object. It computes key metrics such as node and edge counts, network density, mean distance, strength measures, degree centrality, and reciprocity.

Usage

```
## S3 method for class 'group_tna'
summary(object, combined = TRUE, ...)
```

Arguments

object	A group_tna object.
combined	A logical indicating whether the summary results should be combined into a single data frame for all clusters (defaults to TRUE)
...	Ignored

Details

The function extracts the igraph network for each cluster and computes the following network metrics:

- Node count: Total number of nodes in the network.
- Edge count: Total number of edges in the network.
- Network density: Proportion of possible edges that are present in the network.
- Mean distance: The average shortest path length between nodes.
- Mean and standard deviation of out-strength and in-strength: Measures of the total weight of outgoing and incoming edges for each node.
- Mean and standard deviation of out-degree: The number of outgoing edges from each node.
- Centralization of out-degree and in-degree: Measures of how centralized the network is based on the degrees of nodes.
- Reciprocity: The proportion of edges that are reciprocated (i.e., mutual edges between nodes).

Value

A summary.group_tna object which is a list of lists or a combined data.frame containing the following network metrics:

- node_count: The total number of nodes.
- edge_count: The total number of edges.
- network_Density: The density of the network.
- mean_distance: The mean shortest path length.
- mean_out_strength: The mean out-strength of nodes.
- sd_out_strength: The standard deviation of out-strength.
- mean_in_strength: The mean in-strength of nodes.
- sd_in_strength: The standard deviation of in-strength.
- mean_out_degree: The mean out-degree of nodes.
- sd_out_degree: The standard deviation of out-degree.
- centralization_out_degree: The centralization of out-degree.
- centralization_in_degree: The centralization of in-degree.
- reciprocity: The reciprocity of the network.

See Also

Basic functions build_model(), hist.group_tna(), hist.tna(), plot.group_tna(), plot.tna(), plot_frequencies(), plot_frequencies.group_tna(), plot_mosaic(), plot_mosaic.group_tna(), plot_mosaic.tna_data(), print.group_tna(), print.summary.group_tna(), print.summary.tna(), print.tna(), summary.tna(), tna-package

Examples

```
group <- c(rep("High", 1000), rep("Low", 1000))
model <- group_model(group_regulation, group = group)
summary(model)
```

---

summary.group_tna_bootstrap

Summarize Bootstrap Results for a Grouped Transition Network

Description

Summarize Bootstrap Results for a Grouped Transition Network

Usage

```
## S3 method for class 'group_tna_bootstrap'
summary(object, ...)
```

Arguments

object	A group_tna_bootstrap object.
...	Ignored.

Value

A summary.group_tna_bootstrap object containing the weight, estimated p-value and confidence interval of each edge for each cluster.

See Also

Validation functions bootstrap(), deprune(), estimate_cs(), permutation_test(), permutation_test.group_tna(), plot.group_tna_bootstrap(), plot.group_tna_permutation(), plot.group_tna_stability(), plot.tna_bootstrap(), plot.tna_permutation(), plot.tna_reliability(), plot.tna_stability(), print.group_tna_bootstrap(), print.group_tna_permutation(), print.group_tna_stability(), print.summary.group_tna_bootstrap(), print.summary.tna_bootstrap(), print.tna_bootstrap(), print.tna_clustering(), print.tna_permutation(), print.tna_reliability(), print.tna_stability(), prune(), pruning_details(), reliability(), reprune(), summary.tna_bootstrap()

Examples

```
model <- group_tna(engagement_mmm)
# Small number of iterations for CRAN
boot <- bootstrap(model, iter = 10)
summary(boot)
```

`summary.tna` *Calculate Summary of Network Metrics for a Transition Network*

Description

This function calculates a variety of network metrics for a `tna` object. It computes key metrics such as node and edge counts, network density, mean distance, strength measures, degree centrality, and reciprocity.

Usage

```
## S3 method for class 'tna'
summary(object, ...)
```

Arguments

`object`	A `tna` object.
`...`	Ignored.

Details

The function extracts the `igraph` network and computes the following network metrics:

- Node count: Total number of nodes in the network.
- Edge count: Total number of edges in the network.
- Network density: Proportion of possible edges that are present in the network.
- Mean distance: The average shortest path length between nodes.
- Mean and standard deviation of out-strength and in-strength: Measures of the total weight of outgoing and incoming edges for each node.
- Mean and standard deviation of out-degree: The number of outgoing edges from each node.
- Centralization of out-degree and in-degree: Measures of how centralized the network is based on the degrees of nodes.
- Reciprocity: The proportion of edges that are reciprocated (i.e., mutual edges between nodes).

A summary of the metrics is printed to the console.

Value

A named `list` containing the following network metrics (invisibly):

- `node_count`: The total number of nodes.
- `edge_count`: The total number of edges.
- `network_Density`: The density of the network.
- `mean_distance`: The mean shortest path length.
- `mean_out_strength`: The mean out-strength of nodes.
- `sd_out_strength`: The standard deviation of out-strength.
- `mean_in_strength`: The mean in-strength of nodes.
- `sd_in_strength`: The standard deviation of in-strength.
- `mean_out_degree`: The mean out-degree of nodes.
- `sd_out_degree`: The standard deviation of out-degree.
- `centralization_out_degree`: The centralization of out-degree.
- `centralization_in_degree`: The centralization of in-degree.
- `reciprocity`: The reciprocity of the network.

See Also

Basic functions `build_model()`, `hist.group_tna()`, `hist.tna()`, `plot.group_tna()`, `plot.tna()`, `plot_frequencies()`, `plot_frequencies.group_tna()`, `plot_mosaic()`, `plot_mosaic.group_tna()`, `plot_mosaic.tna_data()`, `print.group_tna()`, `print.summary.group_tna()`, `print.summary.tna()`, `print.tna()`, `summary.group_tna()`, `tna-package`

Examples

```
model <- tna(group_regulation)
summary(model)
```

`summary.tna_bootstrap` *Summarize Bootstrap Results*

Description

Summarize Bootstrap Results

Usage

```
## S3 method for class 'tna_bootstrap'
summary(object, ...)
```

Arguments

object	A tna_bootstrap object.
...	Ignored.

Value

A summary.tna_bootstrap object containing the weight, estimated p-value and confidence interval of each edge.

See Also

Validation functions bootstrap(), deprune(), estimate_cs(), permutation_test(), permutation_test.group_tna() plot.group_tna_bootstrap(), plot.group_tna_permutation(), plot.group_tna_stability(), plot.tna_bootstrap(), plot.tna_permutation(), plot.tna_reliability(), plot.tna_stability(), print.group_tna_bootstrap(), print.group_tna_permutation(), print.group_tna_stability(), print.summary.group_tna_bootstrap(), print.summary.tna_bootstrap(), print.tna_bootstrap(), print.tna_clustering(), print.tna_permutation(), print.tna_reliability(), print.tna_stability(), prune(), pruning_details(), reliability(), reprune(), summary.group_tna_bootstrap()

Examples

```
model <- tna(group_regulation)
# Small number of iterations for CRAN
boot <- bootstrap(model, iter = 50)
summary(boot)
```

Index