

Package ‘Nestimate’

April 8, 2026

Title Network Estimation, Bootstrap, and Higher-Order Analysis

Version 0.3.0

Description Estimate, compare, and analyze dynamic and psychological networks using a unified interface. Provides transition network analysis estimation (transition, frequency, co-occurrence, attention-weighted) Saqr et al. (2025) <[doi:10.1145/3706468.3706513](https://doi.org/10.1145/3706468.3706513)>, psychological network methods (correlation, partial correlation, 'graphical lasso', 'Ising') Saqr, Beck, and Lopez-Pernas (2024) <[doi:10.1007/978-3-031-54464-4_19](https://doi.org/10.1007/978-3-031-54464-4_19)>, and higher-order network methods including higher-order networks, higher-order network embedding, hyper-path anomaly, and multi-order generative model. Supports bootstrap inference, permutation testing, split-half reliability, centrality stability analysis, mixed Markov models, multi-cluster multi-layer networks and clustering.

License MIT + file LICENSE

URL <https://github.com/mohsaqr/Nestimate>

BugReports <https://github.com/mohsaqr/Nestimate/issues>

Language en-US

Encoding UTF-8

RoxygenNote 7.3.3

Imports ggplot2, glasso, data.table, cluster

Suggests testthat (>= 3.0.0), tna, cograph, igraph, glmnet, lavaan, stringdist, nnet, IsingFit, bootnet, gimme, qgraph, reticulate, gridExtra, knitr, rmarkdown, pkgdown

Config/testthat/edition 3

Depends R (>= 3.5)

LazyData true

VignetteBuilder knitr

NeedsCompilation no

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

Maintainer Mohammed Saqr <saqr@saqr.me>

Repository CRAN

Date/Publication 2026-04-08 14:20:09 UTC

Contents

| | |
|-------------------------------------|----|
| action_to_onehot | 4 |
| aggregate_weights | 5 |
| association_rules | 6 |
| as_tna | 8 |
| beti_numbers | 10 |
| bootstrap_network | 10 |
| boot_glasso | 12 |
| build_gimme | 15 |
| build_hon | 17 |
| build_honem | 19 |
| build_hypa | 21 |
| build_mcml | 22 |
| build_mmm | 24 |
| build_mogen | 26 |
| build_network | 27 |
| build_simplicial | 30 |
| centrality | 31 |
| centrality_stability | 32 |
| cluster_data | 34 |
| cluster_mmm | 37 |
| cluster_network | 38 |
| cluster_summary | 40 |
| compare_mmm | 45 |
| convert_sequence_format | 45 |
| estimate_network | 47 |
| euler_characteristic | 48 |
| evaluate_links | 48 |
| extract_edges | 49 |
| extract_initial_probs | 50 |
| extract_transition_matrix | 51 |
| get_estimator | 52 |
| group_regulation_long | 52 |
| human_ai_edges | 53 |
| learning_activities | 54 |
| list_estimators | 55 |
| long_to_wide | 56 |
| mogen_transitions | 57 |
| pathways | 58 |
| path_counts | 60 |
| permutation_test | 61 |
| persistent_homology | 63 |

| | |
|---------------------------------------|-----|
| plot.boot_glasso | 64 |
| plot.mcml | 65 |
| plot.mmm_compare | 66 |
| plot.net_association_rules | 66 |
| plot.net_clustering | 67 |
| plot.net_gimme | 68 |
| plot.net_honem | 69 |
| plot.net_link_prediction | 70 |
| plot.net_mmm | 70 |
| plot.net_mogen | 71 |
| plot.net_reliability | 72 |
| plot.net_stability | 73 |
| plot.persistent_homology | 74 |
| plot.q_analysis | 74 |
| plot.simplicial_complex | 75 |
| predictability | 76 |
| predict_links | 77 |
| prepare_data | 79 |
| prepare_for_tna | 81 |
| prepare_onehot | 82 |
| print.boot_glasso | 83 |
| print.mcml | 84 |
| print.mmm_compare | 85 |
| print.nestimate_data | 86 |
| print.netobject | 87 |
| print.netobject_group | 87 |
| print.netobject_ml | 88 |
| print.net_association_rules | 89 |
| print.net_bootstrap | 90 |
| print.net_bootstrap_group | 90 |
| print.net_clustering | 91 |
| print.net_gimme | 92 |
| print.net_hon | 93 |
| print.net_honem | 94 |
| print.net_hypa | 94 |
| print.net_link_prediction | 95 |
| print.net_mmm | 96 |
| print.net_mogen | 97 |
| print.net_permutation | 98 |
| print.net_permutation_group | 99 |
| print.net_reliability | 100 |
| print.net_stability | 101 |
| print.persistent_homology | 102 |
| print.q_analysis | 102 |
| print.simplicial_complex | 103 |
| print.wtna_boot_mixed | 104 |
| print.wtna_mixed | 105 |
| q_analysis | 105 |

| | |
|---|-----|
| register_estimator | 106 |
| reliability | 107 |
| remove_estimator | 108 |
| simplicial_degree | 109 |
| srl_strategies | 110 |
| state_frequencies | 110 |
| summary.boot_glasso | 111 |
| summary.mcml | 112 |
| summary.net_association_rules | 112 |
| summary.net_bootstrap | 113 |
| summary.net_bootstrap_group | 114 |
| summary.net_clustering | 115 |
| summary.net_gimme | 116 |
| summary.net_hon | 116 |
| summary.net_honem | 117 |
| summary.net_hypa | 118 |
| summary.net_link_prediction | 119 |
| summary.net_mmm | 120 |
| summary.net_mogen | 120 |
| summary.net_permutation | 121 |
| summary.net_permutation_group | 122 |
| summary.net_stability | 123 |
| summary.wtna_boot_mixed | 124 |
| verify_simplicial | 125 |
| vibcoding-data | 125 |
| wide_to_long | 128 |
| wtna | 129 |

Index 131

| | |
|------------------|--|
| action_to_onehot | <i>Convert Action Column to One-Hot Encoding</i> |
|------------------|--|

Description

Convert a categorical Action column to one-hot (binary indicator) columns.

Usage

```
action_to_onehot(
  data,
  action_col = "Action",
  states = NULL,
  drop_action = TRUE,
  sort_states = FALSE,
  prefix = ""
)
```

Arguments

| | |
|-------------|---|
| data | Data frame containing an action column. |
| action_col | Character. Name of the action column. Default: "Action". |
| states | Character vector or NULL. States to include as columns. If NULL, uses all unique values. Default: NULL. |
| drop_action | Logical. Remove the original action column. Default: TRUE. |
| sort_states | Logical. Sort state columns alphabetically. Default: FALSE. |
| prefix | Character. Prefix for state column names. Default: "". |

Value

Data frame with one-hot encoded columns (0/1 integers).

Examples

```
long_data <- data.frame(
  Actor = rep(1:3, each = 4),
  Time = rep(1:4, 3),
  Action = sample(c("A", "B", "C"), 12, replace = TRUE)
)
onehot_data <- action_to_onehot(long_data)
head(onehot_data)
```

aggregate_weights *Aggregate Edge Weights*

Description

Aggregates a vector of edge weights using various methods. Compatible with igraph's edge.attr.comb parameter.

Usage

```
aggregate_weights(w, method = "sum", n_possible = NULL)

wagg(w, method = "sum", n_possible = NULL)
```

Arguments

| | |
|------------|---|
| w | Numeric vector of edge weights |
| method | Aggregation method: "sum", "mean", "median", "max", "min", "prod", "density", "geomean" |
| n_possible | Number of possible edges (for density calculation) |

Value

Single aggregated value

Examples

```
w <- c(0.5, 0.8, 0.3, 0.9)
aggregate_weights(w, "sum") # 2.5
aggregate_weights(w, "mean") # 0.625
aggregate_weights(w, "max") # 0.9
mat <- matrix(c(0, 0.5, 0.5, 0.3, 0, 0.7, 0.4, 0.6, 0), 3, 3, byrow = TRUE)
aggregate_weights(mat)
```

association_rules

Discover Association Rules from Sequential or Transaction Data

Description

Discovers association rules using the Apriori algorithm with proper candidate pruning. Accepts `netobject` (extracts sequences as transactions), data frames, lists, or binary matrices.

Support counting is vectorized via `crossprod()` for 2-itemsets and logical matrix indexing for k-itemsets.

Usage

```
association_rules(
  x,
  min_support = 0.1,
  min_confidence = 0.5,
  min_lift = 1,
  max_length = 5L
)
```

Arguments

| | |
|-----------------------------|---|
| <code>x</code> | Input data. Accepts: netobject Uses <code>\$data</code> sequences — each sequence becomes a transaction of its unique states. list Each element is a character vector of items (one transaction). data.frame Wide format: each row is a transaction, character columns are item occurrences. Or a binary matrix (0/1). matrix Binary transaction matrix (rows = transactions, columns = items). |
| <code>min_support</code> | Numeric. Minimum support threshold. Default: 0.1. |
| <code>min_confidence</code> | Numeric. Minimum confidence threshold. Default: 0.5. |
| <code>min_lift</code> | Numeric. Minimum lift threshold. Default: 1.0. |
| <code>max_length</code> | Integer. Maximum itemset size. Default: 5. |

Details

Algorithm:

Uses level-wise Apriori (Agrawal & Srikant, 1994) with the full pruning step: after the join step generates k-candidates, all (k-1)-subsets are verified as frequent before support counting. This is critical for efficiency at $k \geq 4$.

Metrics:

support $P(A \text{ and } B)$. Fraction of transactions containing both antecedent and consequent.

confidence $P(B | A)$. Fraction of antecedent transactions that also contain the consequent.

lift $P(A \text{ and } B) / (P(A) * P(B))$. Values > 1 indicate positive association; < 1 indicate negative association.

conviction $(1 - P(B)) / (1 - \text{confidence})$. Measures departure from independence. Higher = stronger implication.

Value

An object of class "net_association_rules" containing:

rules Data frame with columns: antecedent (list), consequent (list), support, confidence, lift, conviction, count, n_transactions.

frequent_itemsets List of frequent itemsets per level k.

items Character vector of all items.

n_transactions Integer.

n_rules Integer.

params List of min_support, min_confidence, min_lift, max_length.

References

Agrawal, R. & Srikant, R. (1994). Fast algorithms for mining association rules. In *Proc. 20th VLDB Conference*, 487–499.

See Also

[build_network](#), [predict_links](#)

Examples

```
# From a list of transactions
trans <- list(
  c("plan", "discuss", "execute"),
  c("plan", "research", "analyze"),
  c("discuss", "execute", "reflect"),
  c("plan", "discuss", "execute", "reflect"),
  c("research", "analyze", "reflect")
)
rules <- association_rules(trans, min_support = 0.3, min_confidence = 0.5)
print(rules)
```

```
# From a netobject (sequences as transactions)
seqs <- data.frame(
  V1 = sample(LETTERS[1:5], 50, TRUE),
  V2 = sample(LETTERS[1:5], 50, TRUE),
  V3 = sample(LETTERS[1:5], 50, TRUE)
)
net <- build_network(seqs, method = "relative")
rules <- association_rules(net, min_support = 0.1)
```

as_tna

Convert cluster_summary to tna Objects

Description

Converts a `cluster_summary` object to proper `tna` objects that can be used with all functions from the `tna` package. Creates both a between-cluster `tna` model (cluster-level transitions) and within-cluster `tna` models (internal transitions within each cluster).

Usage

```
as_tna(x)

## S3 method for class 'mcml'
as_tna(x)

## Default S3 method:
as_tna(x)
```

Arguments

`x` A `cluster_summary` object created by `cluster_summary`. The `cluster_summary` should typically be created with `type = "tna"` to ensure row-normalized transition probabilities. If created with `type = "raw"`, the raw counts will be passed to `tna::tna()` which will normalize them.

Details

This is the final step in the MCML workflow, enabling full integration with the `tna` package for centrality analysis, bootstrap validation, permutation tests, and visualization.

Requirements:

The `tna` package must be installed. If not available, the function throws an error with installation instructions.

Workflow:

```

# Full MCML workflow
net <- build_network(data, method = "relative")
net$nodes$clusters <- group_assignments
cs <- cluster_summary(net, type = "tna")
tna_models <- as_tna(cs)

# Now use tna package functions
plot(tna_models$macro)
tna::centralities(tna_models$macro)
tna::bootstrap(tna_models$macro, iter = 1000)

# Analyze within-cluster patterns
plot(tna_models$clusters$ClusterA)
tna::centralities(tna_models$clusters$ClusterA)

```

Excluded Clusters:

A within-cluster tna cannot be created when:

- The cluster has only 1 node (no internal transitions possible)
- Some nodes in the cluster have no outgoing edges (row sums to 0)

These clusters are silently excluded from \$clusters. The between-cluster model still includes all clusters.

Value

A cluster_tna object (S3 class) containing:

between A tna object representing cluster-level transitions. Contains \$weights (k x k transition matrix), \$inits (initial distribution), and \$labels (cluster names). Use this for analyzing how learners/entities move between high-level groups or phases.

within Named list of tna objects, one per cluster. Each tna object represents internal transitions within that cluster. Contains \$weights ($n_i \times n_i$ matrix), \$inits (initial distribution), and \$labels (node labels). Clusters with single nodes or zero-row nodes are excluded (tna requires positive row sums).

A netobject_group with data preserved from each sub-network.

A tna object constructed from the input.

See Also

[cluster_summary](#) to create the input object, `plot()` for visualization without conversion, `tna::tna` for the underlying tna constructor

Examples

```

mat <- matrix(runif(36), 6, 6)
rownames(mat) <- colnames(mat) <- LETTERS[1:6]
clusters <- list(G1 = c("A", "B"), G2 = c("C", "D"), G3 = c("E", "F"))
cs <- cluster_summary(mat, clusters, type = "tna")
tna_models <- as_tna(cs)

```

```
tna_models
tna_models$macro$weights
```

betti_numbers *Betti Numbers*

Description

Computes Betti numbers: β_0 (components), β_1 (loops), β_2 (voids), etc.

Usage

```
betti_numbers(sc)
```

Arguments

sc A simplicial_complex object.

Value

Named integer vector c(b0 = ..., b1 = ..., ...).

Examples

```
mat <- matrix(c(0,.6,.5,.6,0,.4,.5,.4,0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A","B","C")
sc <- build_simplicial(mat, threshold = 0.3)
betti_numbers(sc)
```

bootstrap_network *Bootstrap a Network Estimate*

Description

Non-parametric bootstrap for any network estimated by [build_network](#). Works with all built-in methods (transition and association) as well as custom registered estimators.

For transition methods ("relative", "frequency", "co_occurrence"), uses a fast pre-computation strategy: per-sequence count matrices are computed once, and each bootstrap iteration only resamples sequences via colSums (C-level) plus lightweight post-processing. Data must be in wide format for transition bootstrap; use [convert_sequence_format](#) to convert long-format data first.

For association methods ("cor", "pcor", "glasso", and custom estimators), the full estimator is called on resampled rows each iteration.

Usage

```
bootstrap_network(
  x,
  iter = 1000L,
  ci_level = 0.05,
  inference = "stability",
  consistency_range = c(0.75, 1.25),
  edge_threshold = NULL,
  seed = NULL
)
```

Arguments

| | |
|--------------------------------|--|
| <code>x</code> | A netobject from <code>build_network</code> . The data, method, params, scaling, threshold, and level are all extracted from this object. |
| <code>iter</code> | Integer. Number of bootstrap iterations (default: 1000). |
| <code>ci_level</code> | Numeric. Significance level for CIs and p-values (default: 0.05). |
| <code>inference</code> | Character. "stability" (default) tests whether bootstrap replicates fall within a multiplicative consistency range around the original weight. "threshold" tests whether replicates exceed a fixed edge threshold. |
| <code>consistency_range</code> | Numeric vector of length 2. Multiplicative bounds for stability inference (default: <code>c(0.75, 1.25)</code>). |
| <code>edge_threshold</code> | Numeric or NULL. Fixed threshold for inference = "threshold". If NULL, defaults to the 10th percentile of absolute original edge weights. |
| <code>seed</code> | Integer or NULL. RNG seed for reproducibility. |

Value

An object of class "net_bootstrap" containing:

original The original netobject.
mean Bootstrap mean weight matrix.
sd Bootstrap SD matrix.
p_values P-value matrix.
significant Original weights where $p < ci_level$, else 0.
ci_lower Lower CI bound matrix.
ci_upper Upper CI bound matrix.
cr_lower Consistency range lower bound (stability only).
cr_upper Consistency range upper bound (stability only).
summary Long-format data frame of edge-level statistics.
model Pruned netobject (non-significant edges zeroed).
method, params, iter, ci_level, inference Bootstrap config.
consistency_range, edge_threshold Inference parameters.

See Also

[build_network](#), [print.net_bootstrap](#), [summary.net_bootstrap](#)

Examples

```
net <- build_network(data.frame(V1 = c("A","B","C"), V2 = c("B","C","A")),
  method = "relative")
boot <- bootstrap_network(net, iter = 10)

seqs <- data.frame(
  V1 = sample(LETTERS[1:4], 30, TRUE), V2 = sample(LETTERS[1:4], 30, TRUE),
  V3 = sample(LETTERS[1:4], 30, TRUE), V4 = sample(LETTERS[1:4], 30, TRUE)
)
net <- build_network(seqs, method = "relative")
boot <- bootstrap_network(net, iter = 100)
print(boot)
summary(boot)
```

 boot_glasso

Bootstrap for Regularized Partial Correlation Networks

Description

Fast, single-call bootstrap for EBICglasso partial correlation networks. Combines nonparametric edge/centrality bootstrap, case-dropping stability analysis, edge/centrality difference tests, predictability CIs, and thresholded network into one function. Designed as a faster alternative to boot-net with richer output.

Usage

```
boot_glasso(
  x,
  iter = 1000L,
  cs_iter = 500L,
  cs_drop = seq(0.1, 0.9, by = 0.1),
  alpha = 0.05,
  gamma = 0.5,
  nlambda = 100L,
  centrality = c("strength", "expected_influence", "betweenness", "closeness"),
  centrality_fn = NULL,
  cor_method = "pearson",
  ncores = 1L,
  seed = NULL
)
```

Arguments

| | |
|----------------------------|--|
| <code>x</code> | A data frame, numeric matrix (observations x variables), or a netobject with <code>method = "glasso"</code> . |
| <code>iter</code> | Integer. Number of nonparametric bootstrap iterations (default: 1000). |
| <code>cs_iter</code> | Integer. Number of case-dropping iterations per drop proportion (default: 500). |
| <code>cs_drop</code> | Numeric vector. Drop proportions for CS-coefficient computation (default: <code>seq(0.1, 0.9, by = 0.1)</code>). |
| <code>alpha</code> | Numeric. Significance level for CIs (default: 0.05). |
| <code>gamma</code> | Numeric. EBIC hyperparameter (default: 0.5). |
| <code>nlambda</code> | Integer. Number of lambda values in the regularization path (default: 100). |
| <code>centrality</code> | Character vector. Centrality measures to compute. Built-in: "strength", "expected_influence", "betweenness", "closeness". Custom measures beyond these require <code>centrality_fn</code> . Default: <code>c("strength", "expected_influence", "betweenness", "closeness")</code> . |
| <code>centrality_fn</code> | Optional function. A custom centrality function that takes a weight matrix and returns a named list of centrality vectors. When NULL (default), only "strength" and "expected_influence" are computed via <code>rowSums/colSums</code> . When provided, the function is called as <code>centrality_fn(mat)</code> and should return a named list (e.g., <code>list(closeness = ..., betweenness = ...)</code>). |
| <code>cor_method</code> | Character. Correlation method: "pearson" (default), "spearman", or "kendall". |
| <code>ncores</code> | Integer. Number of parallel cores for <code>mclapply</code> (default: 1, sequential). |
| <code>seed</code> | Integer or NULL. RNG seed for reproducibility. |

Value

An object of class "boot_glasso" containing:

- original_pcor** Original partial correlation matrix.
- original_precision** Original precision matrix.
- original_centrality** Named list of original centrality vectors.
- original_predictability** Named numeric vector of node R-squared.
- edge_ci** Data frame of edge CIs (edge, weight, ci_lower, ci_upper, inclusion).
- edge_inclusion** Named numeric vector of edge inclusion probabilities.
- thresholded_pcor** Partial correlation matrix with non-significant edges zeroed.
- centrality_ci** Named list of data frames (node, value, ci_lower, ci_upper) per centrality measure.
- cs_coefficient** Named numeric vector of CS-coefficients per centrality measure.
- cs_data** Data frame of case-dropping correlations (drop_prop, measure, correlation).
- edge_diff_p** Symmetric matrix of pairwise edge difference p-values.
- centrality_diff_p** Named list of symmetric p-value matrices per centrality measure.
- predictability_ci** Data frame of node predictability CIs (node, r2, ci_lower, ci_upper).
- boot_edges** `iter x n_edges` matrix of bootstrap edge weights.
- boot_centrality** Named list of `iter x p` bootstrap centrality matrices.

boot_predictability iter x p matrix of bootstrap R-squared.

nodes Character vector of node names.

n Sample size.

p Number of variables.

iter Number of nonparametric iterations.

cs_iter Number of case-dropping iterations.

cs_drop Drop proportions used.

alpha Significance level.

gamma EBIC hyperparameter.

nlambda Lambda path length.

centrality_measures Character vector of centrality measures.

cor_method Correlation method.

lambda_path Lambda sequence used.

lambda_selected Selected lambda for original data.

timing Named numeric vector with timing in seconds.

See Also

[build_network](#), [bootstrap_network](#)

Examples

```
set.seed(1)
dat <- as.data.frame(matrix(rnorm(60), ncol = 3))
net <- build_network(dat, method = "glasso")
bg <- boot_glasso(net, iter = 10, cs_iter = 5, centrality = "strength")

set.seed(42)
mat <- matrix(rnorm(60), ncol = 4)
colnames(mat) <- LETTERS[1:4]
net <- build_network(as.data.frame(mat), method = "glasso")
boot <- boot_glasso(net, iter = 100, cs_iter = 50, seed = 42,
  centrality = c("strength", "expected_influence"))
print(boot)
summary(boot, type = "edges")
```

 build_gimme

GIMME: Group Iterative Multiple Model Estimation

Description

Estimates person-specific directed networks from intensive longitudinal data using the unified Structural Equation Modeling (uSEM) framework. Implements a data-driven search that identifies:

1. **Group-level paths:** Directed edges present for a majority (default 75\
2. **Individual-level paths:** Additional edges specific to each person, found after group paths are established.

Uses lavaan for SEM estimation and modification indices. Accepts a single data frame with an ID column (not CSV directories).

Usage

```
build_gimme(
  data,
  vars,
  id,
  time = NULL,
  ar = TRUE,
  standardize = FALSE,
  groupcutoff = 0.75,
  subcutoff = 0.5,
  paths = NULL,
  exogenous = NULL,
  hybrid = FALSE,
  rmsea_cutoff = 0.05,
  srmr_cutoff = 0.05,
  nnfi_cutoff = 0.95,
  cfi_cutoff = 0.95,
  n_excellent = 2L,
  seed = NULL
)
```

Arguments

| | |
|-------------------|---|
| <code>data</code> | A <code>data.frame</code> in long format with columns for person ID, time-varying variables, and optionally a time/beep column. |
| <code>vars</code> | Character vector of variable names to model. |
| <code>id</code> | Character string naming the person-ID column. |
| <code>time</code> | Character string naming the time/order column, or <code>NULL</code> . When provided, data is sorted by <code>id</code> then <code>time</code> before lagging. |

| | |
|--------------|--|
| ar | Logical. If TRUE (default), autoregressive paths (each variable predicting itself at lag 1) are included as fixed paths. |
| standardize | Logical. If TRUE (default FALSE), variables are standardized per person before estimation. |
| groupcutoff | Numeric between 0 and 1. Proportion of individuals for whom a path must be significant to be added at group level. Default 0.75. |
| subcutoff | Numeric. Not used (reserved for future subgrouping). |
| paths | Character vector of lavaan-syntax paths to force into the model (e.g., "V2~V1lag"). Default NULL. |
| exogenous | Character vector of variable names to treat as exogenous. Default NULL. |
| hybrid | Logical. If TRUE, also searches residual covariances. Default FALSE. |
| rmsea_cutoff | Numeric. RMSEA threshold for excellent fit (default 0.05). |
| srmr_cutoff | Numeric. SRMR threshold for excellent fit (default 0.05). |
| nnfi_cutoff | Numeric. NNFI/TLI threshold for excellent fit (default 0.95). |
| cfi_cutoff | Numeric. CFI threshold for excellent fit (default 0.95). |
| n_excellent | Integer. Number of fit indices that must be excellent to stop individual search. Default 2. |
| seed | Integer or NULL. Random seed for reproducibility. |

Value

An S3 object of class "net_gimme" containing:

temporal $p \times p$ matrix of group-level temporal (lagged) path counts – entry $[i, j]$ = number of individuals with path $j(t-1) \rightarrow i(t)$.

contemporaneous $p \times p$ matrix of group-level contemporaneous path counts – entry $[i, j]$ = number of individuals with path $j(t) \rightarrow i(t)$.

coefs List of per-person $p \times 2p$ coefficient matrices (rows = endogenous, cols = [lagged, contemporaneous]).

psi List of per-person residual covariance matrices.

fit Data frame of per-person fit indices (chisq, df, pvalue, rmsea, srmr, nnfi, cfi, bic, aic, logl, status).

path_counts $p \times 2p$ matrix: how many individuals have each path.

paths List of per-person character vectors of lavaan path syntax.

group_paths Character vector of group-level paths found.

individual_paths List of per-person character vectors of individual-level paths (beyond group).

syntax List of per-person full lavaan syntax strings.

labels Character vector of variable names.

n_subjects Integer. Number of individuals.

n_obs Integer vector. Time points per individual.

config List of configuration parameters.

See Also[build_network](#)**Examples**

```
if (requireNamespace("gimme", quietly = TRUE)) {
  # Create simple panel data (3 subjects, 4 variables, 50 time points)
  set.seed(42)
  n_sub <- 3; n_t <- 50; vars <- paste0("V", 1:4)
  rows <- lapply(seq_len(n_sub), function(i) {
    d <- as.data.frame(matrix(rnorm(n_t * 4), ncol = 4))
    names(d) <- vars; d$id <- i; d
  })
  panel <- do.call(rbind, rows)
  res <- build_gimme(panel, vars = vars, id = "id")
  print(res)
}
```

build_hon*Build a Higher-Order Network (HON)*

Description

Constructs a Higher-Order Network from sequential data, faithfully implementing the BuildHON algorithm (Xu, Wickramaratne & Chawla, 2016).

The algorithm detects when a first-order Markov model is insufficient to capture sequential dependencies and automatically creates higher-order nodes. Uses KL-divergence to determine whether extending a node's history provides significantly different transition distributions.

Usage

```
build_hon(
  data,
  max_order = 5L,
  min_freq = 1L,
  collapse_repeats = FALSE,
  method = "hon+"
)
```

Arguments

data One of:

- `data.frame`: rows are trajectories, columns are time steps. Trailing NAs are stripped. All non-NA values are coerced to character.

| | |
|-------------------------------|--|
| | <ul style="list-style-type: none"> • <code>list</code>: each element is a character (or coercible) vector representing one trajectory. • <code>tna</code>: a <code>tna</code> object with sequence data. Numeric state IDs are automatically converted to label names. • <code>netobject</code>: a <code>netobject</code> with sequence data. |
| <code>max_order</code> | Integer. Maximum order of the HON. Default 5. The algorithm may produce lower-order nodes if the data do not justify higher orders. |
| <code>min_freq</code> | Integer. Minimum frequency for a transition to be considered. Transitions observed fewer than <code>min_freq</code> times are treated as zero. Default 1. |
| <code>collapse_repeats</code> | Logical. If TRUE, adjacent duplicate states within each trajectory are collapsed before analysis. Default FALSE. |
| <code>method</code> | Character. Algorithm to use: "hon+" (default, parameter-free BuildHON+ with lazy observation building and MaxDivergence pruning) or "hon" (original BuildHON with eager observation building). |

Details

Node naming convention: Higher-order nodes use readable arrow notation. A first-order node is simply "A". A second-order node representing the context "came from A, now at B" is "A -> B". Third-order: "A -> B -> C", etc.

Algorithm overview:

1. Count all subsequence transitions up to `max_order + 1`.
2. Build probability distributions, filtering by `min_freq`.
3. For each first-order source, recursively test whether extending the history (adding more context) produces a significantly different distribution (via KL-divergence vs. an adaptive threshold).
4. Build the network from the accepted rules, rewiring edges so higher-order nodes are properly connected.

Value

An S3 object of class "net_hon" containing:

matrix Weighted adjacency matrix (rows = from, cols = to). Rows and columns use readable arrow notation (e.g., "A -> B").

edges Data frame with columns: `path` (full state sequence, e.g., "A -> B -> C"), `from` (context/conditioning states), `to` (predicted next state), `count` (raw frequency), `probability` (transition probability), `from_order`, `to_order`.

nodes Character vector of HON node names in arrow notation.

n_nodes Number of HON nodes.

n_edges Number of edges.

first_order_states Character vector of unique original states.

max_order_requested The `max_order` parameter used.

max_order_observed Highest order actually present.
min_freq The min_freq parameter used.
n_trajectories Number of trajectories after parsing.
directed Logical. Always TRUE.

References

Xu, J., Wickramaratne, T. L., & Chawla, N. V. (2016). Representing higher-order dependencies in networks. *Science Advances*, 2(5), e1600028.

Saebi, M., Xu, J., Kaplan, L. M., Ribeiro, B., & Chawla, N. V. (2020). Efficient modeling of higher-order dependencies in networks: from algorithm to application for anomaly detection. *EPJ Data Science*, 9(1), 15.

Examples

```
seqs <- list(c("A","B","C","D"), c("A","B","C","A"), c("B","C","D","A"))
hon <- build_hon(seqs, max_order = 2)

# From list of trajectories
trajs <- list(
  c("A", "B", "C", "D", "A"),
  c("A", "B", "D", "C", "A"),
  c("A", "B", "C", "D", "A")
)
hon <- build_hon(trajs, max_order = 3, min_freq = 1)
print(hon)
summary(hon)

# From data.frame (rows = trajectories)
df <- data.frame(T1 = c("A", "A"), T2 = c("B", "B"),
                 T3 = c("C", "D"), T4 = c("D", "C"))
hon <- build_hon(df, max_order = 2)
```

 build_honem

Build HONEM Embeddings for Higher-Order Networks

Description

Constructs low-dimensional embeddings from a Higher-Order Network (HON) that preserve higher-order dependencies. Uses exponentially-decaying matrix powers of the HON transition matrix followed by truncated SVD.

Usage

```
build_honem(hon, dim = 32L, max_power = 10L)
```

Arguments

| | |
|-----------|---|
| hon | A net_hon object from <code>build_hon</code> , or a square weighted adjacency matrix. |
| dim | Integer. Embedding dimension (default 32). |
| max_power | Integer. Maximum walk length for neighborhood computation (default 10). Higher values capture longer-range structure. |

Details

HONEM is parameter-free and scalable — no random walks, skip-gram, or hyperparameter tuning required.

Value

An object of class `net_honem` with components:

embeddings Numeric matrix (`n_nodes` x `dim`) of node embeddings.

nodes Character vector of node names.

singular_values Numeric vector of top singular values.

explained_variance Proportion of variance explained.

dim Embedding dimension used.

max_power Maximum power used.

n_nodes Number of nodes embedded.

References

Saebi, M., Ciampaglia, G. L., Kazemzadeh, S., & Meyur, R. (2020). HONEM: Learning Embedding for Higher Order Networks. *Big Data*, 8(4), 255–269.

Examples

```
seqs <- list(c("A","B","C","D"), c("A","B","C","A"), c("B","C","D","A"))
hem <- build_honem(build_hon(seqs, max_order = 2), dim = 2)
```

```
trajs <- list(c("A","B","C","D"), c("A","B","D","C"),
             c("B","C","D","A"), c("C","D","A","B"))
hon <- build_hon(trajs, max_order = 2)
emb <- build_honem(hon, dim = 4)
print(emb)
plot(emb)
```

| | |
|------------|---------------------------------------|
| build_hypa | <i>Detect Path Anomalies via HYPa</i> |
|------------|---------------------------------------|

Description

Constructs a k -th order De Bruijn graph from sequential trajectory data and uses a hypergeometric null model to detect paths with anomalous frequencies. Paths occurring more or less often than expected under the null model are flagged as over- or under-represented.

Usage

```
build_hypa(data, k = 3L, alpha = 0.05, min_count = 5L)
```

Arguments

| | |
|------------------|--|
| data | A data.frame (rows = trajectories), list of character vectors, tna object, or netobject with sequence data. For tna/netobject, numeric state IDs are automatically converted to label names. |
| k | Integer. Order of the De Bruijn graph (default 2). Detects anomalies in paths of length k . |
| alpha | Numeric. Significance threshold for anomaly classification (default 0.05). Paths with HYPa score $< \alpha$ are under-represented; paths with score $> 1 - \alpha$ are over-represented. |
| min_count | Integer. Minimum observed count for a path to be classified as anomalous (default 2). Paths with fewer observations are always classified as "normal" regardless of their HYPa score, since single occurrences are unreliable. |

Value

An object of class `net_hypa` with components:

scores Data frame with path, from, to, observed, expected, ratio, hypa_score, anomaly columns. The path column shows the full state sequence (e.g., "A -> B -> C"); from is the context (conditioning states); to is the next state; ratio is observed / expected.

adjacency Weighted adjacency matrix of the De Bruijn graph.

xi Fitted propensity matrix.

k Order of the De Bruijn graph.

alpha Significance threshold used.

n_anomalous Number of anomalous paths detected.

n_over Number of over-represented paths.

n_under Number of under-represented paths.

n_edges Total number of edges.

nodes Node names in the De Bruijn graph.

References

LaRock, T., Nanumyan, V., Scholtes, I., Casiraghi, G., Eliassi-Rad, T., & Schweitzer, F. (2020). HYPA: Efficient Detection of Path Anomalies in Time Series Data on Networks. *SDM 2020*, 460–468.

Examples

```
seqs <- list(c("A","B","C"), c("B","C","A"), c("A","C","B"), c("A","B","C"))
hyp <- build_hypa(seqs, k = 2)
```

```
trajs <- list(c("A","B","C"), c("A","B","C"), c("A","B","C"),
             c("A","B","D"), c("C","B","D"), c("C","B","A"))
h <- build_hypa(trajs, k = 2)
print(h)
```

build_mcml

Build MCML from Raw Transition Data

Description

Builds a Multi-Cluster Multi-Level (MCML) model from raw transition data (edge lists or sequences) by recoding node labels to cluster labels and counting actual transitions. Unlike [cluster_summary](#) which aggregates a pre-computed weight matrix, this function works from the original transition data to produce the TRUE Markov chain over cluster states.

Usage

```
build_mcml(
  x,
  clusters = NULL,
  method = c("sum", "mean", "median", "max", "min", "density", "geomean"),
  type = c("tna", "frequency", "cooccurrence", "semi_markov", "raw"),
  directed = TRUE,
  compute_within = TRUE
)
```

Arguments

x Input data. Accepts multiple formats:

- data.frame with from/to columns** Edge list. Columns named from/source/src/v1/node1/i and to/target/tgt/v2/node2/j are auto-detected. Optional weight column (weight/w/value/strength).
- data.frame without from/to columns** Sequence data. Each row is a sequence, columns are time steps. Consecutive pairs (t, t+1) become transitions.

| | |
|----------------|--|
| | <p>tna object If <code>x\$data</code> is non-NULL, uses sequence path on the raw data. Otherwise falls back to <code>cluster_summary</code>.</p> <p>netobject If <code>x\$data</code> is non-NULL, detects edge list vs sequence data. Otherwise falls back to <code>cluster_summary</code>.</p> <p>cluster_summary Returns as-is.</p> <p>square numeric matrix Falls back to <code>cluster_summary</code>.</p> <p>non-square or character matrix Treated as sequence data.</p> |
| clusters | <p>Cluster/group assignments. Accepts:</p> <p>named list Direct mapping. List names = cluster names, values = character vectors of node labels. Example: <code>list(A = c("N1", "N2"), B = c("N3", "N4"))</code></p> <p>data.frame A data frame where the first column contains node names and the second column contains group/cluster names. Example: <code>data.frame(node = c("N1", "N2", "N3"), group = c("A", "A", "B"))</code></p> <p>membership vector Character or numeric vector. Node names are extracted from the data. Example: <code>c("A", "A", "B", "B")</code></p> <p>column name string For edge list data.frames, the name of a column containing cluster labels. The mapping is built from unique (node, group) pairs in both from and to columns.</p> <p>NULL Auto-detect from <code>netobject\$nodes</code> or <code>\$node_groups</code> (same logic as <code>cluster_summary</code>).</p> |
| method | Aggregation method for combining edge weights: "sum", "mean", "median", "max", "min", "density", "geomean". Default "sum". |
| type | Post-processing: "tna" (row-normalize), "cooccurrence" (symmetrize), "semi_markov", or "raw". Default "tna". |
| directed | Logical. Treat as directed network? Default TRUE. |
| compute_within | Logical. Compute within-cluster matrices? Default TRUE. |

Value

A `cluster_summary` object with `meta$source = "transitions"`, fully compatible with `plot()`, `as_tna()`, and `plot()`.

See Also

`cluster_summary` for matrix-based aggregation, `as_tna()` to convert to tna objects, `plot()` for visualization

Examples

```
# Edge list with clusters
edges <- data.frame(
  from = c("A", "A", "B", "C", "C", "D"),
  to   = c("B", "C", "A", "D", "D", "A"),
  weight = c(1, 2, 1, 3, 1, 2)
)
clusters <- list(G1 = c("A", "B"), G2 = c("C", "D"))
cs <- build_mcml(edges, clusters)
```

```

cs$macro$weights

# Sequence data with clusters
seqs <- data.frame(
  T1 = c("A", "C", "B"),
  T2 = c("B", "D", "A"),
  T3 = c("C", "C", "D"),
  T4 = c("D", "A", "C")
)
cs <- build_mcml(seqs, clusters, type = "raw")
cs$macro$weights

```

 build_mmm

Fit a Mixed Markov Model

Description

Discovers latent subgroups with different transition dynamics using Expectation-Maximization. Each mixture component has its own transition matrix. Sequences are probabilistically assigned to components.

Usage

```

build_mmm(
  data,
  k = 2L,
  n_starts = 50L,
  max_iter = 200L,
  tol = 1e-06,
  smooth = 0.01,
  seed = NULL,
  covariates = NULL
)

```

Arguments

| | |
|----------|---|
| data | A data.frame (wide format), netobject, or tna model. For tna objects, extracts the stored data. |
| k | Integer. Number of mixture components. Default: 2. |
| n_starts | Integer. Number of random restarts. Default: 50. |
| max_iter | Integer. Maximum EM iterations per start. Default: 200. |
| tol | Numeric. Convergence tolerance. Default: 1e-6. |
| smooth | Numeric. Laplace smoothing constant. Default: 0.01. |
| seed | Integer or NULL. Random seed. |

covariates Optional. Covariates integrated into the EM algorithm to model covariate-dependent mixing proportions. Accepts formula, character vector, string, or data.frame (same forms as `cluster_data`). Unlike the post-hoc analysis in `cluster_data()`, these covariates directly influence cluster membership during estimation. Requires the **nnet** package.

Value

An object of class `net_mmm` with components:

models List of netobjects, one per component.

k Number of components.

mixing Numeric vector of mixing proportions.

posterior $N \times k$ matrix of posterior probabilities.

assignments Integer vector of hard assignments (1..k).

quality List: `avepp` (per-class), `avepp_overall`, `entropy`, `relative_entropy`, `classification_error`.

log_likelihood, BIC, AIC, ICL Model fit statistics.

states Character vector of state names.

See Also

[compare_mmm](#), [build_network](#)

Examples

```
seqs <- data.frame(V1 = sample(c("A", "B", "C"), 30, TRUE),
                  V2 = sample(c("A", "B", "C"), 30, TRUE))
mmm <- build_mmm(seqs, k = 2, n_starts = 1, max_iter = 10, seed = 1)
mmm

seqs <- data.frame(
  V1 = sample(LETTERS[1:3], 30, TRUE), V2 = sample(LETTERS[1:3], 30, TRUE),
  V3 = sample(LETTERS[1:3], 30, TRUE), V4 = sample(LETTERS[1:3], 30, TRUE)
)
mmm <- build_mmm(seqs, k = 2, seed = 42)
print(mmm)
summary(mmm)
```

 build_mogen

Build Multi-Order Generative Model (MOGen)

Description

Constructs higher-order De Bruijn graphs from sequential trajectory data and selects the optimal Markov order using AIC, BIC, or likelihood ratio tests.

Usage

```
build_mogen(
  data,
  max_order = 5L,
  criterion = c("aic", "bic", "lrt"),
  lrt_alpha = 0.01
)
```

Arguments

| | |
|-----------|--|
| data | A data.frame (rows = trajectories, columns = time points) or a list of character/numeric vectors (one per trajectory). |
| max_order | Integer. Maximum Markov order to test (default 5). |
| criterion | Character. Model selection criterion: "aic" (default), "bic", or "lrt" (likelihood ratio test). |
| lrt_alpha | Numeric. Significance threshold for LRT (default 0.01). |

Details

At order k , nodes are k -tuples of states and edges represent transitions between overlapping k -tuples. The model tests increasingly complex Markov orders and selects the one that best balances fit and parsimony.

Value

An object of class `net_mogen` with components:

- optimal_order** Selected optimal Markov order.
- criterion** Which criterion was used for selection.
- orders** Integer vector of tested orders (0 to `max_order`).
- aic** Named numeric vector of AIC values per order.
- bic** Named numeric vector of BIC values per order.
- log_likelihood** Named numeric vector of log-likelihoods.
- dof** Named integer vector of cumulative DOF per model.
- layer_dof** Named integer vector of per-layer DOF.

transition_matrices List of transition matrices (index 1 = order 0).

states Unique first-order states.

n_paths Number of trajectories.

n_observations Total number of state observations.

References

Scholtes, I. (2017). When is a Network a Network? Multi-Order Graphical Model Selection in Pathways and Temporal Networks. *KDD 2017*.

Gote, C. & Scholtes, I. (2023). Predicting variable-length paths in networked systems using multi-order generative models. *Applied Network Science*, 8, 62.

Examples

```
seqs <- list(c("A","B","C","D"), c("A","B","C","A"), c("B","C","D","A"))
mg <- build_mogen(seqs, max_order = 2)
```

```
trajs <- list(c("A","B","C","D"), c("A","B","D","C"),
             c("B","C","D","A"), c("C","D","A","B"))
m <- build_mogen(trajs, max_order = 3)
print(m)
plot(m)
```

build_network

Build a Network

Description

Universal network estimation function that supports both transition networks (relative, frequency, co-occurrence) and association networks (correlation, partial correlation, graphical lasso). Uses the global estimator registry, so custom estimators can also be used.

Usage

```
build_network(
  data,
  method,
  actor = NULL,
  action = NULL,
  time = NULL,
  session = NULL,
  order = NULL,
  codes = NULL,
  group = NULL,
```

```

format = "auto",
window_size = 3L,
mode = c("non-overlapping", "overlapping"),
scaling = NULL,
threshold = 0,
level = NULL,
time_threshold = 900,
params = list(),
...
)

```

Arguments

| | |
|----------------|--|
| data | Data frame (sequences or per-observation frequencies) or a square symmetric matrix (correlation or covariance). |
| method | Character. Required. Name of a registered estimator. Built-in methods: "relative", "frequency", "co_occurrence", "cor", "pcor", "glasso". Aliases: "tna" and "transition" map to "relative"; "ftna" and "counts" map to "frequency"; "cna" maps to "co_occurrence"; "corr" and "correlation" map to "cor"; "partial" maps to "pcor"; "ebicglasso" and "regularized" map to "glasso". |
| actor | Character. Name of the actor/person ID column for sequence grouping. Default: NULL. |
| action | Character. Name of the action/state column (long format). Default: NULL. |
| time | Character. Name of the time column (long format). Default: NULL. |
| session | Character. Name of the session column. Default: NULL. |
| order | Character. Name of the ordering column. Default: NULL. |
| codes | Character vector. Column names of one-hot encoded states (for onehot format). Default: NULL. |
| group | Character. Name of a grouping column for per-group networks. Returns a netobject_group (named list of netobjects). Default: NULL. |
| format | Character. Input format: "auto", "wide", "long", or "onehot". Default: "auto". |
| window_size | Integer. Window size for one-hot windowing. Default: 3L. |
| mode | Character. Windowing mode: "non-overlapping" or "overlapping". Default: "non-overlapping". |
| scaling | Character vector or NULL. Post-estimation scaling to apply (in order). Options: "minmax", "max", "rank", "normalize". Can combine: c("rank", "minmax"). Default: NULL (no scaling). |
| threshold | Numeric. Absolute values below this are set to zero in the result matrix. Default: 0 (no thresholding). |
| level | Character or NULL. Multilevel decomposition for association methods. One of NULL, "between", "within", "both". Requires id_col. Default: NULL. |
| time_threshold | Numeric. Maximum time gap (seconds) for long format session splitting. Default: 900. |

| | |
|--------|---|
| params | Named list. Method-specific parameters passed to the estimator function (e.g. <code>list(gamma = 0.5)</code> for <code>glasso</code> , or <code>list(format = "wide")</code> for transition methods). This is the key composability feature: downstream functions like <code>bootstrap</code> or <code>grid search</code> can store and replay the full params list without knowing method internals. |
| ... | Additional arguments passed to the estimator function. |

Details

The function works as follows:

1. Resolves method aliases to canonical names.
2. Retrieves the estimator function from the global registry.
3. For association methods with `level` specified, decomposes the data (between-person means or within-person centering).
4. Calls the estimator: `do.call(fn, c(list(data = data), params))`.
5. Applies scaling and thresholding to the result matrix.
6. Extracts edges and constructs the `netobject`.

Value

An object of class `c("netobject", "cograph_network")` containing:

data The input data used for estimation, as a data frame.

weights The estimated network weight matrix.

nodes Data frame with columns `id`, `label`, `name`, `x`, `y`. Node labels are in `$nodes$label`.

edges Data frame of non-zero edges with integer `from/to` (node IDs) and numeric `weight`.

directed Logical. Whether the network is directed.

method The resolved method name.

params The params list used (for reproducibility).

scaling The scaling applied (or `NULL`).

threshold The threshold applied.

n_nodes Number of nodes.

n_edges Number of non-zero edges.

level Decomposition level used (or `NULL`).

meta List with source, layout, and tna metadata (cograph-compatible).

node_groups Node groupings data frame, or `NULL`.

Method-specific extras (e.g. `precision_matrix`, `cor_matrix`, `frequency_matrix`, `lambda_selected`, etc.) are preserved from the estimator output.

When `level = "both"`, returns an object of class `"netobject_m1"` with `$between` and `$within` sub-networks and a `$method` field.

See Also

[register_estimator](#), [list_estimators](#), [bootstrap_network](#)

Examples

```
seqs <- data.frame(V1 = c("A", "B", "C", "A"), V2 = c("B", "C", "A", "B"))
net <- build_network(seqs, method = "relative")
net

# Transition network (relative probabilities)
seqs <- data.frame(
  V1 = sample(LETTERS[1:4], 30, TRUE), V2 = sample(LETTERS[1:4], 30, TRUE),
  V3 = sample(LETTERS[1:4], 30, TRUE), V4 = sample(LETTERS[1:4], 30, TRUE)
)
net <- build_network(seqs, method = "relative")
print(net)

# Association network (glasso)
freq_data <- convert_sequence_format(seqs, format = "frequency")
net_glasso <- build_network(freq_data, method = "glasso",
  params = list(gamma = 0.5, nlambda = 50))

# With scaling
net_scaled <- build_network(seqs, method = "relative",
  scaling = c("rank", "minmax"))
```

build_simplicial

Build a Simplicial Complex

Description

Constructs a simplicial complex from a network or higher-order pathway object. Three construction methods are available:

- **Clique complex** ("clique"): every clique in the thresholded graph becomes a simplex. The standard bridge from graph theory to algebraic topology.
- **Pathway complex** ("pathway"): each higher-order pathway from a net_hon or net_hypa becomes a simplex.
- **Vietoris-Rips** ("vr"): nodes with edge weight \geq threshold are connected; all cliques in the resulting graph become simplices.

Usage

```
build_simplicial(
  x,
  type = "clique",
```

```

    threshold = 0,
    max_dim = 10L,
    max_pathways = NULL,
    ...
  )

```

Arguments

| | |
|--------------|--|
| x | A square matrix, tna, netobject, net_hon, or net_hypa. |
| type | Construction type: "clique" (default), "pathway", or "vr". |
| threshold | Minimum absolute edge weight to include an edge (default 0). Edges below this are ignored. |
| max_dim | Maximum simplex dimension (default 10). A k-simplex has k+1 nodes. |
| max_pathways | For type = "pathway": maximum number of pathways to include, ranked by count (HON) or ratio (HYPA). NULL includes all. Default NULL. |
| ... | Additional arguments passed to build_hon() when x is a tna/netobject with type = "pathway". |

Value

A simplicial_complex object.

See Also

[betti_numbers](#), [persistent_homology](#), [simplicial_degree](#), [q_analysis](#)

Examples

```

mat <- matrix(c(0,.6,.5,.6,0,.4,.5,.4,0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A","B","C")
sc <- build_simplicial(mat, threshold = 0.3)
print(sc)
betti_numbers(sc)

```

Description

Computes centrality measures from a netobject or netobject_group. For directed networks the default measures are InStrength, OutStrength, and Betweenness. For undirected networks the defaults are Strength (column sums) and Betweenness.

Usage

```
centrality(x, ...)

## S3 method for class 'netobject'
centrality(x, measures = NULL, loops = FALSE, centrality_fn = NULL, ...)

## S3 method for class 'netobject_group'
centrality(x, measures = NULL, loops = FALSE, centrality_fn = NULL, ...)

## S3 method for class 'cograph_network'
centrality(x, measures = NULL, loops = FALSE, centrality_fn = NULL, ...)

## S3 method for class 'mcml'
centrality(x, measures = NULL, loops = FALSE, centrality_fn = NULL, ...)
```

Arguments

| | |
|---------------|--|
| x | A netobject or netobject_group. |
| ... | Additional arguments (ignored). |
| measures | Character vector. Centrality measures to compute. Built-in: "InStrength", "OutStrength", "Betweenness", "InCloseness", "OutCloseness", "Closeness". Default depends on directedness. |
| loops | Logical. Include self-loops (diagonal) in computation? Default: FALSE. |
| centrality_fn | Optional function. Custom centrality function that takes a weight matrix and returns a named list of centrality vectors. |

Value

For a netobject: a data frame with node names as rows and centrality measures as columns. For a netobject_group: a named list of such data frames (one per group).

Examples

```
seqs <- data.frame(
  V1 = c("A", "B", "A", "C"), V2 = c("B", "C", "B", "A"),
  V3 = c("C", "A", "C", "B"))
net <- build_network(seqs, method = "relative")
centrality(net)
```

Description

Estimates the stability of centrality indices under case-dropping. For each drop proportion, sequences are randomly removed and the network is re-estimated. The correlation between the original and subset centrality values is computed. The CS-coefficient is the maximum proportion of cases that can be dropped while maintaining a correlation above threshold in at least certainty of bootstrap samples.

For transition methods, uses pre-computed per-sequence count matrices for fast resampling. Strength centralities (InStrength, OutStrength) are computed directly from the matrix without igrph.

Usage

```
centrality_stability(
  x,
  measures = c("InStrength", "OutStrength", "Betweenness"),
  iter = 1000L,
  drop_prop = seq(0.1, 0.9, by = 0.1),
  threshold = 0.7,
  certainty = 0.95,
  method = "pearson",
  centrality_fn = NULL,
  loops = FALSE,
  seed = NULL
)
```

Arguments

| | |
|---------------|--|
| x | A netobject from build_network . |
| measures | Character vector. Centrality measures to assess. Built-in: "InStrength", "OutStrength", "Betweenness", "InCloseness", "OutCloseness", "Closeness". Custom measures beyond these require centrality_fn. Default: c("InStrength", "OutStrength", "Betweenness"). |
| iter | Integer. Number of bootstrap iterations per drop proportion (default: 1000). |
| drop_prop | Numeric vector. Proportions of cases to drop (default: seq(0.1, 0.9, by = 0.1)). |
| threshold | Numeric. Minimum correlation to consider stable (default: 0.7). |
| certainty | Numeric. Required proportion of iterations above threshold (default: 0.95). |
| method | Character. Correlation method: "pearson", "spearman", or "kendall" (default: "pearson"). |
| centrality_fn | Optional function. A custom centrality function that takes a weight matrix and returns a named list of centrality vectors. When NULL (default), only "InStrength" and "OutStrength" are computed via colSums/rowSums. When provided, the function is called as centrality_fn(mat) and should return a named list (e.g., list(Betweenness = ..., Closeness = ...)). |
| loops | Logical. If FALSE (default), self-loops (diagonal) are excluded from centrality computation. This does not modify the stored matrix. |
| seed | Integer or NULL. RNG seed for reproducibility. |

Value

An object of class "net_stability" containing:

cs Named numeric vector of CS-coefficients per measure.

correlations Named list of matrices (iter x n_prop) of correlation values per measure.

measures Character vector of measures assessed.

drop_prop Drop proportions used.

threshold Stability threshold.

certainty Required certainty level.

iter Number of iterations.

method Correlation method.

See Also

[build_network](#), [reliability](#)

Examples

```
net <- build_network(data.frame(V1 = c("A","B","C","A"),
  V2 = c("B","C","A","B")), method = "relative")
cs <- centrality_stability(net, iter = 10, drop_prop = 0.3)

seqs <- data.frame(
  V1 = sample(LETTERS[1:4], 30, TRUE), V2 = sample(LETTERS[1:4], 30, TRUE),
  V3 = sample(LETTERS[1:4], 30, TRUE), V4 = sample(LETTERS[1:4], 30, TRUE)
)
net <- build_network(seqs, method = "relative")
cs <- centrality_stability(net, iter = 100, seed = 42,
  measures = c("InStrength", "OutStrength"))
print(cs)
```

Description

Clusters wide-format sequences using pairwise string dissimilarity and either PAM (Partitioning Around Medoids) or hierarchical clustering. Supports 9 distance metrics including temporal weighting for Hamming distance. When the **stringdist** package is available, uses C-level distance computation for 100-1000x speedup on edit distances.

Usage

```
cluster_data(
  data,
  k,
  dissimilarity = "hamming",
  method = "pam",
  na_syms = c("*", "%"),
  weighted = FALSE,
  lambda = 1,
  seed = NULL,
  q = 2L,
  p = 0.1,
  covariates = NULL,
  ...
)
```

```
cluster_sequences(
  data,
  k,
  dissimilarity = "hamming",
  method = "pam",
  na_syms = c("*", "%"),
  weighted = FALSE,
  lambda = 1,
  seed = NULL,
  q = 2L,
  p = 0.1,
  covariates = NULL,
  ...
)
```

Arguments

| | |
|---------------|---|
| data | Input data. Accepts multiple formats: data.frame / matrix Wide-format sequences (rows = sequences, columns = time points, values = state names). netobject A network object from <code>build_network</code> . Extracts the stored sequence data. Only valid for sequence-based methods (relative, frequency, co_occurrence, attention). tna A tna model from the tna package. Decodes the integer-encoded sequence data using stored labels. cograph_network A cograph network object. Extracts the stored sequence data. |
| k | Integer. Number of clusters (must be between 2 and <code>nrow(data) - 1</code>). |
| dissimilarity | Character. Distance metric. One of "hamming", "osa" (optimal string alignment), "lv" (Levenshtein), "dl" (Damerau-Levenshtein), "lcs" (longest com- |

| | |
|------------|---|
| | mon subsequence), "qgram", "cosine", "jaccard", "jw" (Jaro-Winkler). Default: "hamming". |
| method | Character. Clustering method. "pam" for Partitioning Around Medoids, or a hierarchical method: "ward.D2", "ward.D", "complete", "average", "single", "mcquitty", "median", "centroid". Default: "pam". |
| na_syms | Character vector. Symbols treated as missing values. Default: c("*", "%"). |
| weighted | Logical. Apply exponential decay weighting to Hamming distance positions? Only valid when <code>dissimilarity = "hamming"</code> . Default: FALSE. |
| lambda | Numeric. Decay rate for weighted Hamming. Higher values weight earlier positions more strongly. Default: 1. |
| seed | Integer or NULL. Random seed for reproducibility. Default: NULL. |
| q | Integer. Size of q-grams for "qgram", "cosine", and "jaccard" distances. Default: 2L. |
| p | Numeric. Winkler prefix penalty for Jaro-Winkler distance (clamped to 0–0.25). Default: 0.1. |
| covariates | Optional. Post-hoc covariate analysis of cluster membership via multinomial logistic regression. Accepts: formula ~ Age + Gender character vector c("Age", "Gender") string "Age + Gender" data.frame All columns used as covariates NULL No covariate analysis (default) Covariates are looked up in <code>netobject\$metadata</code> or non-sequence columns of the input data. For <code>tna</code> and <code>cograph_network</code> inputs, pass covariates as a <code>data.frame</code> . Results stored in <code>\$covariates</code> . Requires the nnet package. |
| ... | Additional arguments (currently unused). |

Value

An object of class "net_clustering" containing:

data The original input data.

k Number of clusters.

assignments Named integer vector of cluster assignments.

silhouette Overall average silhouette width.

sizes Named integer vector of cluster sizes.

method Clustering method used.

dissimilarity Distance metric used.

distance The computed dissimilarity matrix (dist object).

medoids Integer vector of medoid row indices (PAM only; NULL for hierarchical methods).

seed Seed used (or NULL).

weighted Logical, whether weighted Hamming was used.

lambda Lambda value used (0 if not weighted).

Examples

```
seqs <- data.frame(V1 = c("A", "B", "C", "A", "B"), V2 = c("B", "C", "A", "B", "A"),
                  V3 = c("C", "A", "B", "C", "B"))
cl <- cluster_data(seqs, k = 2)
cl

seqs <- data.frame(
  V1 = sample(LETTERS[1:3], 20, TRUE), V2 = sample(LETTERS[1:3], 20, TRUE),
  V3 = sample(LETTERS[1:3], 20, TRUE), V4 = sample(LETTERS[1:3], 20, TRUE)
)
cl <- cluster_data(seqs, k = 2)
print(cl)
summary(cl)
```

cluster_mmm

Cluster sequences using Mixed Markov Models

Description

Convenience alias for `build_mmm`. Fits a mixture of Markov chains to sequence data and returns per-component transition networks with EM-fitted initial state probabilities.

Usage

```
cluster_mmm(
  data,
  k = 2L,
  n_starts = 50L,
  max_iter = 200L,
  tol = 1e-06,
  smooth = 0.01,
  seed = NULL,
  covariates = NULL
)
```

Arguments

| | |
|-----------------------|---|
| <code>data</code> | A data.frame (wide format), netobject, or tna model. For tna objects, extracts the stored data. |
| <code>k</code> | Integer. Number of mixture components. Default: 2. |
| <code>n_starts</code> | Integer. Number of random restarts. Default: 50. |
| <code>max_iter</code> | Integer. Maximum EM iterations per start. Default: 200. |
| <code>tol</code> | Numeric. Convergence tolerance. Default: 1e-6. |
| <code>smooth</code> | Numeric. Laplace smoothing constant. Default: 0.01. |

| | |
|------------|---|
| seed | Integer or NULL. Random seed. |
| covariates | Optional. Covariates integrated into the EM algorithm to model covariate-dependent mixing proportions. Accepts formula, character vector, string, or data.frame (same forms as <code>cluster_data</code>). Unlike the post-hoc analysis in <code>cluster_data()</code> , these covariates directly influence cluster membership during estimation. Requires the nnet package. |

Details

Use `build_network` on the result to extract per-cluster networks with any estimation method, or use `cluster_network` for a one-shot clustering + network call.

Value

A `net_mmm` object. See `build_mmm` for details.

See Also

`build_mmm`, `cluster_network`

Examples

```
seqs <- data.frame(V1 = sample(c("A", "B", "C"), 30, TRUE),
                  V2 = sample(c("A", "B", "C"), 30, TRUE))
mmm <- cluster_mmm(seqs, k = 2, n_starts = 1, max_iter = 10, seed = 1)
mmm

seqs <- data.frame(
  V1 = sample(LETTERS[1:3], 40, TRUE),
  V2 = sample(LETTERS[1:3], 40, TRUE),
  V3 = sample(LETTERS[1:3], 40, TRUE)
)
mmm <- cluster_mmm(seqs, k = 2)
print(mmm)
```

cluster_network

Cluster data and build per-cluster networks in one step

Description

Combines sequence clustering and network estimation into a single call. Clusters the data using the specified algorithm, then calls `build_network` on each cluster subset.

Usage

```
cluster_network(data, k, cluster_by = "pam", dissimilarity = "hamming", ...)
```

Arguments

| | |
|---------------|---|
| data | Sequence data. Accepts a data frame, matrix, or netobject. See cluster_data for supported formats. |
| k | Integer. Number of clusters. |
| cluster_by | Character. Clustering algorithm passed to cluster_data 's method parameter ("pam", "ward.D2", "ward.D", "complete", "average", "single", "mcquitty", "median", "centroid"), or "mmm" for Mixed Markov Model clustering. Default: "pam". |
| dissimilarity | Character. Distance metric for sequence clustering (ignored when cluster_by = "mmm"). Default: "hamming". |
| ... | Passed directly to build_network . Use method to specify the network type; threshold, scaling, and all other build_network arguments are supported. |

Details

If data is a netobject and method is not provided in ..., the original network method is inherited automatically so the per-cluster networks match the type of the input network.

Value

A netobject_group.

See Also

[cluster_data](#), [cluster_mmm](#), [build_network](#)

Examples

```
seqs <- data.frame(V1 = c("A", "B", "C", "A", "B"), V2 = c("B", "C", "A", "B", "A"),
                  V3 = c("C", "A", "B", "C", "B"))
grp <- cluster_network(seqs, k = 2)
grp

seqs <- data.frame(
  V1 = sample(LETTERS[1:4], 50, TRUE), V2 = sample(LETTERS[1:4], 50, TRUE),
  V3 = sample(LETTERS[1:4], 50, TRUE), V4 = sample(LETTERS[1:4], 50, TRUE)
)
# Default: PAM clustering, relative (transition) networks
grp <- cluster_network(seqs, k = 3)

# Specify network method (cor requires numeric panel data)
## Not run:
panel <- as.data.frame(matrix(rnorm(1500), nrow = 300, ncol = 5))
grp <- cluster_network(panel, k = 2, method = "cor")

## End(Not run)

# MMM-based clustering
grp <- cluster_network(seqs, k = 2, cluster_by = "mmm")
```

| | |
|-----------------|-----------------------------------|
| cluster_summary | <i>Cluster Summary Statistics</i> |
|-----------------|-----------------------------------|

Description

Aggregates node-level network weights to cluster-level summaries. Computes both between-cluster transitions (how clusters connect to each other) and within-cluster transitions (how nodes connect within each cluster).

Usage

```
cluster_summary(
  x,
  clusters = NULL,
  method = c("sum", "mean", "median", "max", "min", "density", "geomean"),
  type = c("tna", "cooccurrence", "semi_markov", "raw"),
  directed = TRUE,
  compute_within = TRUE
)

csum(
  x,
  clusters = NULL,
  method = c("sum", "mean", "median", "max", "min", "density", "geomean"),
  type = c("tna", "cooccurrence", "semi_markov", "raw"),
  directed = TRUE,
  compute_within = TRUE
)
```

Arguments

| | |
|-----------------------|---|
| <code>x</code> | <p>Network input. Accepts multiple formats:</p> <ul style="list-style-type: none"> matrix Numeric adjacency/weight matrix. Row and column names are used as node labels. Values represent edge weights (e.g., transition counts, co-occurrence frequencies, or probabilities). netobject A cograph network object. The function extracts the weight matrix from <code>x\$weights</code> or converts via <code>to_matrix()</code>. Clusters can be auto-detected from node attributes. tna A tna object from the tna package. Extracts <code>x\$weights</code>. cluster_summary If already a cluster_summary, returns unchanged. |
| <code>clusters</code> | <p>Cluster/group assignments for nodes. Accepts multiple formats:</p> <ul style="list-style-type: none"> NULL (default) Auto-detect from netobject. Looks for columns named 'clusters', 'cluster', 'groups', or 'group' in <code>x\$nodes</code>. Throws an error if no cluster column is found. This option only works when <code>x</code> is a netobject. |

| | |
|----------------|---|
| | <p>vector Cluster membership for each node, in the same order as the matrix rows/columns. Can be numeric (1, 2, 3) or character ("A", "B"). Cluster names will be derived from unique values. Example: <code>c(1, 1, 2, 2, 3, 3)</code> assigns first two nodes to cluster 1.</p> <p>data.frame A data frame where the first column contains node names and the second column contains group/cluster names. Example: <code>data.frame(node = c("A", "B", "C"), group = c("G1", "G1", "G2"))</code></p> <p>named list Explicit mapping of cluster names to node labels. List names become cluster names, values are character vectors of node labels that must match matrix row/column names. Example: <code>list(Alpha = c("A", "B"), Beta = c("C", "D"))</code></p> |
| method | <p>Aggregation method for combining edge weights within/between clusters. Controls how multiple node-to-node edges are summarized:</p> <p>"sum" (default) Sum of all edge weights. Best for count data (e.g., transition frequencies). Preserves total flow.</p> <p>"mean" Average edge weight. Best when cluster sizes differ and you want to control for size. Note: when input is already a transition matrix (rows sum to 1), "mean" avoids size bias. Example: cluster with 5 nodes won't have 5x the weight of cluster with 1 node.</p> <p>"median" Median edge weight. Robust to outliers.</p> <p>"max" Maximum edge weight. Captures strongest connection.</p> <p>"min" Minimum edge weight. Captures weakest connection.</p> <p>"density" Sum divided by number of possible edges. Normalizes by cluster size combinations.</p> <p>"geomean" Geometric mean of positive weights. Useful for multiplicative processes.</p> |
| type | <p>Post-processing applied to aggregated weights. Determines the interpretation of the resulting matrices:</p> <p>"tna" (default) Row-normalize so each row sums to 1. Creates transition probabilities suitable for Markov chain analysis. Interpretation: "Given I'm in cluster A, what's the probability of transitioning to cluster B?" Required for use with tna package functions. Diagonal represents within-cluster transition probability.</p> <p>"raw" No normalization. Returns aggregated counts/weights as-is. Use for frequency analysis or when you need raw counts. Compatible with igraph's <code>contract + simplify</code> output.</p> <p>"cooccurrence" Symmetrize the matrix: $(A + t(A)) / 2$. For undirected co-occurrence analysis.</p> <p>"semi_markov" Row-normalize with duration weighting. For semi-Markov process analysis.</p> |
| directed | <p>Logical. If TRUE (default), treat network as directed. A->B and B->A are separate edges. If FALSE, edges are undirected and the matrix is symmetrized before processing.</p> |
| compute_within | <p>Logical. If TRUE (default), compute within-cluster transition matrices for each cluster. Each cluster gets its own $n_i \times n_i$ matrix showing internal node-to-node</p> |

transitions. Set to FALSE to skip this computation for better performance when only between-cluster summary is needed.

Details

This is the core function for Multi-Cluster Multi-Level (MCML) analysis. Use `as_tna()` to convert results to tna objects for further analysis with the tna package.

Workflow:

Typical MCML analysis workflow:

```
# 1. Create network
net <- build_network(data, method = "relative")
net$nodes$clusters <- group_assignments

# 2. Compute cluster summary
cs <- cluster_summary(net, type = "tna")

# 3. Convert to tna models
tna_models <- as_tna(cs)

# 4. Analyze/visualize
plot(tna_models$macro)
tna::centralities(tna_models$macro)
```

Between-Cluster Matrix Structure:

The `macro$weights` matrix has clusters as both rows and columns:

- Off-diagonal (row *i*, col *j*): Aggregated weight from cluster *i* to cluster *j*
- Diagonal (row *i*, col *i*): Within-cluster total (sum of internal edges in cluster *i*)

When `type = "tna"`, rows sum to 1 and diagonal values represent "retention rate" - the probability of staying within the same cluster.

Choosing method and type:

| Input data | Recommended | Reason |
|--------------------|---------------------------|---|
| Edge counts | method="sum", type="tna" | Preserves total flow, normalizes to probabilities |
| Transition matrix | method="mean", type="tna" | Avoids cluster size bias |
| Frequencies | method="sum", type="raw" | Keep raw counts for analysis |
| Correlation matrix | method="mean", type="raw" | Average correlations |

Value

A `cluster_summary` object (S3 class) containing:

between List with two elements:

weights $k \times k$ matrix of cluster-to-cluster weights, where *k* is the number of clusters. Row *i*, column *j* contains the aggregated weight from cluster *i* to cluster *j*. Diagonal contains within-cluster totals. Processing depends on type.

inits Numeric vector of length k . Initial state distribution across clusters, computed from column sums of the original matrix. Represents the proportion of incoming edges to each cluster.

within Named list with one element per cluster. Each element contains:

weights $n_i \times n_i$ matrix for nodes within that cluster. Shows internal transitions between nodes in the same cluster.

inits Initial distribution within the cluster.

NULL if `compute_within = FALSE`.

clusters Named list mapping cluster names to their member node labels. Example: `list(A = c("n1", "n2"), B = c("n3", "n4", "n5"))`

meta List of metadata:

type The type argument used ("tna", "raw", etc.)

method The method argument used ("sum", "mean", etc.)

directed Logical, whether network was treated as directed

n_nodes Total number of nodes in original network

n_clusters Number of clusters

cluster_sizes Named vector of cluster sizes

See [cluster_summary](#).

See Also

`as_tna()` to convert results to tna objects, `plot()` for two-layer visualization, `plot()` for flat cluster visualization

Examples

```
# -----
# Basic usage with matrix and cluster vector
# -----
mat <- matrix(runif(100), 10, 10)
rownames(mat) <- colnames(mat) <- LETTERS[1:10]

clusters <- c(1, 1, 1, 2, 2, 2, 3, 3, 3, 3)
cs <- cluster_summary(mat, clusters)

# Access results
cs$weights      # 3x3 cluster transition matrix
cs$inits        # Initial distribution
cs$clusters$1$weights # Within-cluster 1 transitions
cs$meta        # Metadata

# -----
# Named list clusters (more readable)
# -----
clusters <- list(
  Alpha = c("A", "B", "C"),
  Beta  = c("D", "E", "F"),
```

```

Gamma = c("G", "H", "I", "J")
)
cs <- cluster_summary(mat, clusters, type = "tna")
cs$macro$weights      # Rows/cols named Alpha, Beta, Gamma
cs$clusters$Alpha     # Within Alpha cluster

# -----
# Auto-detect clusters from netobject
# -----

seqs <- data.frame(
  V1 = sample(LETTERS[1:10], 30, TRUE), V2 = sample(LETTERS[1:10], 30, TRUE),
  V3 = sample(LETTERS[1:10], 30, TRUE)
)
net <- build_network(seqs, method = "relative")
cs2 <- cluster_summary(net, c(1, 1, 1, 2, 2, 2, 3, 3, 3, 3))

# -----
# Different aggregation methods
# -----
cs_sum <- cluster_summary(mat, clusters, method = "sum") # Total flow
cs_mean <- cluster_summary(mat, clusters, method = "mean") # Average
cs_max <- cluster_summary(mat, clusters, method = "max") # Strongest

# -----
# Raw counts vs TNA probabilities
# -----
cs_raw <- cluster_summary(mat, clusters, type = "raw")
cs_tna <- cluster_summary(mat, clusters, type = "tna")

rowSums(cs_raw$macro$weights) # Various sums
rowSums(cs_tna$macro$weights) # All equal to 1

# -----
# Skip within-cluster computation for speed
# -----
cs_fast <- cluster_summary(mat, clusters, compute_within = FALSE)
cs_fast$clusters # NULL

# -----
# Convert to tna objects for tna package
# -----
cs <- cluster_summary(mat, clusters, type = "tna")
tna_models <- as_tna(cs)
# tna_models$macro      # tna object
# tna_models$clusters$Alpha # tna object
mat <- matrix(runif(16), 4, 4)
rownames(mat) <- colnames(mat) <- LETTERS[1:4]
csum(mat, c(1, 1, 2, 2))

```

| | |
|-------------|--|
| compare_mmm | <i>Compare MMM fits across different k</i> |
|-------------|--|

Description

Compare MMM fits across different k

Usage

```
compare_mmm(data, k = 2:5, ...)
```

Arguments

| | |
|------|---|
| data | Data frame, netobject, or tna model. |
| k | Integer vector of component counts. Default: 2:5. |
| ... | Arguments passed to build_mmm . |

Value

A mmm_compare data frame with BIC, AIC, ICL, AvePP, entropy per k.

Examples

```
seqs <- data.frame(V1 = sample(c("A", "B", "C"), 30, TRUE),
                  V2 = sample(c("A", "B", "C"), 30, TRUE))
comp <- compare_mmm(seqs, k = 2:3, n_starts = 1, max_iter = 10, seed = 1)
comp

seqs <- data.frame(
  V1 = sample(LETTERS[1:3], 30, TRUE), V2 = sample(LETTERS[1:3], 30, TRUE),
  V3 = sample(LETTERS[1:3], 30, TRUE), V4 = sample(LETTERS[1:3], 30, TRUE)
)
comp <- compare_mmm(seqs, k = 2:3, seed = 42)
print(comp)
```

| |
|-------------------------|
| convert_sequence_format |
|-------------------------|

Convert Sequence Data to Different Formats

Description

Convert wide or long sequence data into frequency counts, one-hot encoding, edge lists, or follows format.

Usage

```
convert_sequence_format(
  data,
  seq_cols = NULL,
  id_col = NULL,
  action = NULL,
  time = NULL,
  format = c("frequency", "onehot", "edgelist", "follows")
)
```

Arguments

| | |
|-----------------------|---|
| <code>data</code> | Data frame containing sequence data. |
| <code>seq_cols</code> | Character vector. Names of columns containing sequential states (for wide format input). If NULL, all columns except <code>id_col</code> are used. Default: NULL. |
| <code>id_col</code> | Character vector. Name(s) of the ID column(s). For wide format, defaults to the first column. For long format, required. Default: NULL. |
| <code>action</code> | Character or NULL. Name of the column containing actions/states (for long format input). If provided, data is treated as long format. Default: NULL. |
| <code>time</code> | Character or NULL. Name of the time column for ordering actions within sequences (for long format). Default: NULL. |
| <code>format</code> | Character. Output format: "frequency" Count of each action per sequence (wide, one column per state). "onehot" Binary presence/absence of each action per sequence. "edgelist" Consecutive transition pairs (from, to) per sequence. "follows" Each action paired with the action that preceded it. |

Value

A data frame in the requested format:

frequency ID columns + one integer column per state with counts.

onehot ID columns + one binary column per state (0/1).

edgelist ID columns + from and to columns.

follows ID columns + act and follows columns.

See Also

[frequencies](#) for building transition frequency matrices.

Examples

```
# Wide format input
seqs <- data.frame(V1 = c("A", "B", "A"), V2 = c("B", "A", "C"), V3 = c("A", "C", "B"))
convert_sequence_format(seqs, format = "frequency")
convert_sequence_format(seqs, format = "edgelist")
```

| | |
|------------------|--|
| estimate_network | <i>Estimate a Network (Deprecated)</i> |
|------------------|--|

Description

This function is deprecated. Use [build_network](#) instead.

Usage

```
estimate_network(
  data,
  method = "relative",
  params = list(),
  scaling = NULL,
  threshold = 0,
  level = NULL,
  ...
)
```

Arguments

| | |
|-----------|---|
| data | Data frame (sequences or per-observation frequencies) or a square symmetric matrix (correlation or covariance). |
| method | Character. Defaults to "relative" for backward compatibility. |
| params | Named list. Method-specific parameters passed to the estimator function (e.g. <code>list(gamma = 0.5)</code> for glasso, or <code>list(format = "wide")</code> for transition methods). This is the key composability feature: downstream functions like bootstrap or grid search can store and replay the full params list without knowing method internals. |
| scaling | Character vector or NULL. Post-estimation scaling to apply (in order). Options: "minmax", "max", "rank", "normalize". Can combine: <code>c("rank", "minmax")</code> . Default: NULL (no scaling). |
| threshold | Numeric. Absolute values below this are set to zero in the result matrix. Default: 0 (no thresholding). |
| level | Character or NULL. Multilevel decomposition for association methods. One of NULL, "between", "within", "both". Requires <code>id_col</code> . Default: NULL. |
| ... | Additional arguments passed to build_network . |

Value

A netobject (see [build_network](#)).

See Also

[build_network](#)

Examples

```
data <- data.frame(A = c("x", "y", "z", "x"), B = c("y", "x", "z", "y"))
net <- estimate_network(data, method = "relative")
```

euler_characteristic *Euler Characteristic*

Description

Computes $\chi = \sum_{k=0}^d (-1)^k f_k$ where f_k is the number of k -simplices. By the Euler-Poincare theorem, $\chi = \sum_k (-1)^k \beta_k$.

Usage

```
euler_characteristic(sc)
```

Arguments

sc A simplicial_complex object.

Value

Integer.

Examples

```
mat <- matrix(c(0,.6,.5,.6,0,.4,.5,.4,0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A","B","C")
sc <- build_simplicial(mat, threshold = 0.3)
euler_characteristic(sc)
```

evaluate_links *Evaluate Link Predictions Against Known Edges*

Description

Computes AUC-ROC, precision@k, and average precision for link predictions against a set of known true edges.

Usage

```
evaluate_links(pred, true_edges, k = c(5L, 10L, 20L))
```

Arguments

| | |
|------------|--|
| pred | A net_link_prediction object. |
| true_edges | A data frame with columns from and to, or a binary matrix where 1 indicates a true edge. |
| k | Integer vector. Values of k for precision@k. Default: c(5, 10, 20). |

Value

A data frame with columns: method, auc, average_precision, and one precision_at_k column per k value.

Examples

```
set.seed(42)
seqs <- data.frame(
  V1 = sample(LETTERS[1:5], 50, TRUE),
  V2 = sample(LETTERS[1:5], 50, TRUE),
  V3 = sample(LETTERS[1:5], 50, TRUE)
)
net <- build_network(seqs, method = "relative")
pred <- predict_links(net, exclude_existing = FALSE)

# Evaluate: predict the network's own edges
true <- data.frame(from = pred$predictions$from[1:5],
  to = pred$predictions$to[1:5])
evaluate_links(pred, true)
```

 extract_edges

Extract Edge List with Weights

Description

Extract an edge list from a TNA model, representing the network as a data frame of from-to-weight tuples.

Usage

```
extract_edges(model, threshold = 0, include_self = FALSE, sort_by = "weight")
```

Arguments

| | |
|--------------|---|
| model | A TNA model object or a matrix of weights. |
| threshold | Numeric. Minimum weight to include an edge. Default: 0. |
| include_self | Logical. Whether to include self-loops. Default: FALSE. |
| sort_by | Character. Column to sort by: "weight" (descending), "from", "to", or NULL for no sorting. Default: "weight". |

Details

This function converts the transition matrix into an edge list format, which is useful for visualization, analysis with igraph, or export to other network tools.

Value

A data frame with columns:

from Source state name.

to Target state name.

weight Edge weight (transition probability).

See Also

[extract_transition_matrix](#) for the full matrix, [build_network](#) for network estimation.

Examples

```
seqs <- data.frame(V1 = c("A", "B", "A"), V2 = c("B", "A", "C"), V3 = c("A", "C", "B"))
net <- build_network(seqs, method = "relative")
edges <- extract_edges(net, threshold = 0.05)
head(edges)
```

extract_initial_probs *Extract Initial Probabilities from Model*

Description

Extract the initial state probability vector from a TNA model object.

Usage

```
extract_initial_probs(model)
```

Arguments

model A TNA model object or a list containing an 'initial' element.

Details

Initial probabilities represent the probability of starting a sequence in each state. If the model doesn't have explicit initial probabilities, this function attempts to estimate them from the data or use uniform probabilities.

Value

A named numeric vector of initial state probabilities.

See Also

[extract_transition_matrix](#) for extracting the transition matrix, [extract_edges](#) for extracting an edge list.

Examples

```
seqs <- data.frame(V1 = c("A", "B", "A"), V2 = c("B", "A", "C"), V3 = c("A", "C", "B"))
net <- build_network(seqs, method = "relative")
init_probs <- extract_initial_probs(net)
print(init_probs)
```

extract_transition_matrix

Extract Transition Matrix from Model

Description

Extract the transition probability matrix from a TNA model object.

Usage

```
extract_transition_matrix(model, type = c("raw", "scaled"))
```

Arguments

| | |
|-------|--|
| model | A TNA model object or a list containing a 'weights' element. |
| type | Character. Type of matrix to return: " raw " The raw weight matrix as stored in the model. " scaled " Row-normalized to ensure rows sum to 1. Default: "raw". |

Details

TNA models store transition weights in different locations depending on the model type. This function handles the extraction automatically.

For "scaled" type, each row is divided by its sum to create valid transition probabilities. This is useful when the original weights don't sum to 1.

Value

A square numeric matrix with row and column names as state names.

See Also

[extract_initial_probs](#) for extracting initial probabilities, [extract_edges](#) for extracting an edge list.

Examples

```
seqs <- data.frame(V1 = c("A", "B", "A"), V2 = c("B", "A", "C"), V3 = c("A", "C", "B"))
net <- build_network(seqs, method = "relative")
trans_mat <- extract_transition_matrix(net)
print(trans_mat)
```

| | |
|---------------|--|
| get_estimator | <i>Retrieve a Registered Estimator</i> |
|---------------|--|

Description

Retrieve a registered network estimator by name.

Usage

```
get_estimator(name)
```

Arguments

name Character. Name of the estimator to retrieve.

Value

A list with elements fn, description, directed.

See Also

[register_estimator](#), [list_estimators](#)

Examples

```
est <- get_estimator("relative")
```

| | |
|-----------------------|---|
| group_regulation_long | <i>Group Regulation in Collaborative Learning (Long Format)</i> |
|-----------------------|---|

Description

Students' regulation strategies during collaborative learning, in long format. Contains 27,533 time-stamped action records from multiple students working in groups across two courses.

Usage

```
group_regulation_long
```

Format

A data frame with 27,533 rows and 6 columns:

Actor Integer. Student identifier.

Achiever Character. Achievement level: "High" or "Low".

Group Numeric. Collaboration group identifier.

Course Character. Course identifier ("A", "B", or "C").

Time POSIXct. Timestamp of the action.

Action Character. Regulation action (e.g., cohesion, consensus, discuss, synthesis).

Source

Synthetically generated from the `group_regulation` dataset in the `tna` package.

See Also

[learning_activities](#), [srl_strategies](#)

Examples

```
# Build a transition network per actor
net <- build_network(group_regulation_long,
                    method = "relative",
                    actor = "Actor", action = "Action", time = "Time")
net
```

human_ai_edges

Human-AI Vibe Coding Edge List

Description

Non-aggregated edge list of all consecutive action transitions across 429 sessions. Each row is one transition from one action to the next within a session.

Usage

```
human_ai_edges
```

Format

A data frame with 18,918 rows and 14 columns:

from Character. Source action (code level).

to Character. Target action (code level).

weight Integer. Always 1 (non-aggregated).

session_id Character. Unique session hash.

session Integer. Numeric session ID (1..429).

project Character. Project identifier.

order Integer. Position of the transition within the session.

timepoint Character. ISO 8601 timestamp of the source action.

from_actor Character. Actor of the source action.

to_actor Character. Actor of the target action.

from_category Character. Category of the source action.

to_category Character. Category of the target action.

from_superclass Character. Superclass of the source action.

to_superclass Character. Superclass of the target action.

Source

Saqr, M. (2026). Human-AI vibe coding interaction study.

Examples

```
# Filter to Human -> AI transitions only
handoffs <- human_ai_edges[
  human_ai_edges$from_actor == "Human" &
  human_ai_edges$to_actor == "AI", ]
```

learning_activities *Online Learning Activity Indicators*

Description

Simulated binary time-series data for 200 students across 30 time points. At each time point, one or more learning activities may be active (1) or inactive (0). Activities: Reading, Video, Forum, Quiz, Coding, Review. Includes temporal persistence (activities tend to continue across adjacent time points).

Usage

```
learning_activities
```

Format

A data frame with 6,000 rows and 7 columns:

student Integer. Student identifier (1–200).

Reading Integer (0/1). Reading activity indicator.

Video Integer (0/1). Video watching indicator.

Forum Integer (0/1). Discussion forum indicator.

Quiz Integer (0/1). Quiz/assessment indicator.

Coding Integer (0/1). Coding practice indicator.

Review Integer (0/1). Review/revision indicator.

Examples

```
head(learning_activities)
```

| | |
|-----------------|---------------------------------------|
| list_estimators | <i>List All Registered Estimators</i> |
|-----------------|---------------------------------------|

Description

Return a data frame summarising all registered network estimators.

Usage

```
list_estimators()
```

Value

A data frame with columns name, description, directed.

See Also

[register_estimator](#), [get_estimator](#)

Examples

```
list_estimators()
```

`long_to_wide`*Convert Long Format to Wide Sequences*

Description

Convert sequence data from long format (one row per action) to wide format (one row per sequence, columns as time points).

Usage

```
long_to_wide(  
  data,  
  id_col = "Actor",  
  time_col = "Time",  
  action_col = "Action",  
  time_prefix = "V",  
  fill_na = TRUE  
)
```

Arguments

| | |
|--------------------------|---|
| <code>data</code> | Data frame in long format. |
| <code>id_col</code> | Character. Name of the column identifying sequences. Default: "Actor". |
| <code>time_col</code> | Character. Name of the column identifying time points. Default: "Time". |
| <code>action_col</code> | Character. Name of the column containing actions/states. Default: "Action". |
| <code>time_prefix</code> | Character. Prefix for time point columns in output. Default: "V". |
| <code>fill_na</code> | Logical. Whether to fill missing time points with NA. Default: TRUE. |

Details

This function converts long format data (like that from `simulate_long_data()`) to the wide format expected by `tna::tna()` and related functions.

If `time_col` contains non-integer values (e.g., timestamps), the function will use the ordering within each sequence to create time indices.

Value

A data frame in wide format where each row is a sequence and columns V1, V2, ... contain the actions at each time point.

See Also

[wide_to_long](#) for the reverse conversion, [prepare_for_tna](#) for preparing data for TNA analysis.

Examples

```
long_data <- data.frame(
  Actor = rep(1:3, each = 4),
  Time = rep(1:4, 3),
  Action = sample(c("A", "B", "C"), 12, replace = TRUE)
)
wide_data <- long_to_wide(long_data, id_col = "Actor")
head(wide_data)
```

| | |
|-------------------|--|
| mogen_transitions | <i>Extract Transition Table from a MOGen Model</i> |
|-------------------|--|

Description

Returns a data frame of all transitions at a given Markov order, sorted by count (descending). Each row shows the full path as a readable sequence of states, along with the observed count and transition probability.

Usage

```
mogen_transitions(x, order = NULL, min_count = 1L)
```

Arguments

| | |
|------------------------|---|
| <code>x</code> | A <code>net_mogen</code> object from <code>build_mogen()</code> . |
| <code>order</code> | Integer. Which order's transitions to extract. Defaults to the optimal order selected by the model. |
| <code>min_count</code> | Integer. Minimum observed count to include (default 1). Use this to filter out rare transitions that have unreliable probabilities. |

Details

At order k , each edge in the De Bruijn graph represents a $(k+1)$ -step path. For example, at order 2, the edge from node "AI -> FAIL" to node "FAIL -> SOLVE" represents the three-step path AI -> FAIL -> SOLVE. The path column reconstructs this full sequence for readability.

Value

A data frame with columns:

path The full state sequence (e.g., "AI -> FAIL -> SOLVE").

count Number of times this transition was observed.

probability Transition probability $P(\text{to} \mid \text{from})$.

from The context / conditioning states (k -gram source node).

to The predicted next state.

Examples

```
seqs <- list(c("A","B","C","D"), c("A","B","C","A"), c("B","C","D","A"))
mg <- build_mogen(seqs, max_order = 2)
mogen_transitions(mg, order = 1)
```

```
trajs <- list(c("A","B","C","D"), c("A","B","D","C"),
             c("B","C","D","A"), c("C","D","A","B"))
m <- build_mogen(trajs, max_order = 3)
mogen_transitions(m, order = 1)
```

 pathways

Extract Pathways from Higher-Order Network Objects

Description

Extracts higher-order pathway strings suitable for `cograph::plot_simplicial()`. Each pathway represents a multi-step dependency: source states lead to a target state.

For `net_hon`: extracts edges where the source node is higher-order ($order > 1$), i.e., the transitions that differ from first-order Markov.

For `net_hypa`: extracts anomalous paths (over- or under-represented relative to the hypergeometric null model).

For `net_mogen`: extracts all transitions at the optimal order (or a specified order).

Usage

```
pathways(x, ...)
```

S3 method for class 'net_hon'

```
pathways(x, min_count = 1L, min_prob = 0, top = NULL, order = NULL, ...)
```

S3 method for class 'net_hypa'

```
pathways(x, type = "all", ...)
```

S3 method for class 'net_association_rules'

```
pathways(x, top = NULL, min_lift = NULL, min_confidence = NULL, ...)
```

S3 method for class 'net_link_prediction'

```
pathways(x, method = NULL, top = 10L, evidence = TRUE, max_evidence = 3L, ...)
```

S3 method for class 'net_mogen'

```
pathways(x, order = NULL, min_count = 1L, min_prob = 0, top = NULL, ...)
```

Arguments

| | |
|----------------|--|
| x | A higher-order network object (net_hon, net_hypa, or net_mogen). |
| ... | Additional arguments. |
| min_count | Integer. Minimum transition count to include (default: 1). |
| min_prob | Numeric. Minimum transition probability to include (default: 0). |
| top | Integer or NULL. Return only the top N pathways ranked by count (default: NULL = all). |
| order | Integer or NULL. Markov order to extract. Default: optimal order from model selection. |
| type | Character. Which anomalies to include: "all" (default), "over", or "under". |
| min_lift | Numeric or NULL. Additional lift filter applied on top of the object's original threshold (default: NULL). |
| min_confidence | Numeric or NULL. Additional confidence filter (default: NULL). |
| method | Character or NULL. Which prediction method to use. Default: first method in the object. |
| evidence | Logical. If TRUE, include common neighbor evidence nodes in each pathway. Default: TRUE. |
| max_evidence | Integer. Maximum number of evidence nodes per pathway (default: 3). |

Value

A character vector of pathway strings in arrow notation (e.g. "A B -> C"), suitable for `cograph::plot_simplicial()`.

A character vector of pathway strings.

A character vector of pathway strings.

A character vector of pathway strings.

A character vector of pathway strings.

A character vector of pathway strings.

Methods (by class)

- `pathways(net_hon)`: Extract higher-order pathways from HON
- `pathways(net_hypa)`: Extract anomalous pathways from HYPa
- `pathways(net_association_rules)`: Extract pathways from association rules
Converts association rules $\{A, B\} \Rightarrow \{C\}$ into pathway strings "A B -> C" suitable for `cograph::plot_simplicial()`. Antecedent items become source nodes; consequent items become the target.
- `pathways(net_link_prediction)`: Extract pathways from link predictions
Converts predicted links into pathway strings for `cograph::plot_simplicial()`. When `evidence = TRUE` (default), each predicted edge A -> B is enriched with common neighbors that structurally support the prediction, producing "A cn1 cn2 -> B".
- `pathways(net_mogen)`: Extract transition pathways from MOGen

Examples

```

seqs <- list(c("A","B","C","D"), c("A","B","C","A"))
hon <- build_hon(seqs, max_order = 3)
pw <- pathways(hon)

trans <- list(c("A","B","C"), c("A","B"), c("B","C","D"), c("A","C","D"))
rules <- association_rules(trans, min_support = 0.3, min_confidence = 0.3,
                           min_lift = 0)
pathways(rules)

seqs <- data.frame(
  V1 = sample(LETTERS[1:5], 50, TRUE),
  V2 = sample(LETTERS[1:5], 50, TRUE),
  V3 = sample(LETTERS[1:5], 50, TRUE)
)
net <- build_network(seqs, method = "relative")
pred <- predict_links(net, methods = "common_neighbors")
pathways(pred)

```

path_counts

Count Path Frequencies in Trajectory Data

Description

Counts the frequency of k-step paths (k-grams) across all trajectories. Useful for understanding which sequences dominate the data before applying formal models.

Usage

```
path_counts(data, k = 2L, top = NULL)
```

Arguments

| | |
|------|--|
| data | A list of character vectors (trajectories) or a data.frame (rows = trajectories, columns = time points). |
| k | Integer. Length of the path / n-gram (default 2). A k of 2 counts individual transitions; k of 3 counts two-step paths, etc. |
| top | Integer or NULL. If set, returns only the top N most frequent paths (default NULL = all). |

Value

A data frame with columns: path, count, proportion.

Examples

```
trajs <- list(c("A","B","C","D"), c("A","B","D","C"))
path_counts(trajs, k = 2)
```

```
path_counts(trajs, k = 3, top = 10)
```

permutation_test

Permutation Test for Network Comparison

Description

Compares two networks estimated by `build_network` using a permutation test. Works with all built-in methods (transition and association) as well as custom registered estimators. The test shuffles which observations belong to which group, re-estimates networks, and tests whether observed edge-wise differences exceed chance.

For transition methods ("relative", "frequency", "co_occurrence"), uses a fast pre-computation strategy: per-sequence count matrices are computed once, and each permutation iteration only shuffles group labels and computes group-wise colSums.

For association methods ("cor", "pcor", "glasso", and custom estimators), the full estimator is called on each permuted group split.

Usage

```
permutation_test(
  x,
  y = NULL,
  iter = 1000L,
  alpha = 0.05,
  paired = FALSE,
  adjust = "none",
  nlambda = 50L,
  seed = NULL
)
```

Arguments

| | |
|--------|--|
| x | A netobject (from <code>build_network</code>). |
| y | A netobject (from <code>build_network</code>). Must use the same method and have the same nodes as x. |
| iter | Integer. Number of permutation iterations (default: 1000). |
| alpha | Numeric. Significance level (default: 0.05). |
| paired | Logical. If TRUE, permute within pairs (requires equal number of observations in x and y). Default: FALSE. |

| | |
|----------------------|---|
| <code>adjust</code> | Character. p-value adjustment method passed to <code>p.adjust</code> (default: "none"). Common choices: "holm", "BH", "bonferroni". |
| <code>nlambda</code> | Integer. Number of lambda values for the glassopath regularisation path (only used when <code>method = "glasso"</code>). Higher values give finer lambda resolution at the cost of speed. Default: 50. |
| <code>seed</code> | Integer or NULL. RNG seed for reproducibility. |

Value

An object of class "net_permutation" containing:

x The first netobject.

y The second netobject.

diff Observed difference matrix ($x - y$).

diff_sig Observed difference where $p < \alpha$, else 0.

p_values P-value matrix (adjusted if `adjust != "none"`).

effect_size Effect size matrix (observed diff / SD of permutation diffs).

summary Long-format data frame of edge-level results.

method The network estimation method.

iter Number of permutation iterations.

alpha Significance level used.

paired Whether paired permutation was used.

adjust p-value adjustment method used.

See Also

[build_network](#), [bootstrap_network](#), [print.net_permutation](#), [summary.net_permutation](#)

Examples

```
s1 <- data.frame(V1 = c("A", "B", "C"), V2 = c("B", "C", "A"))
s2 <- data.frame(V1 = c("A", "C", "B"), V2 = c("C", "B", "A"))
n1 <- build_network(s1, method = "relative")
n2 <- build_network(s2, method = "relative")
perm <- permutation_test(n1, n2, iter = 10)

set.seed(1)
d1 <- data.frame(V1 = sample(LETTERS[1:4], 20, TRUE),
                 V2 = sample(LETTERS[1:4], 20, TRUE),
                 V3 = sample(LETTERS[1:4], 20, TRUE))
d2 <- data.frame(V1 = sample(LETTERS[1:4], 20, TRUE),
                 V2 = sample(LETTERS[1:4], 20, TRUE),
                 V3 = sample(LETTERS[1:4], 20, TRUE))
net1 <- build_network(d1, method = "relative")
net2 <- build_network(d2, method = "relative")
perm <- permutation_test(net1, net2, iter = 100, seed = 42)
print(perm)
```

```
summary(perm)
```

persistent_homology *Persistent Homology*

Description

Computes persistent homology by building simplicial complexes at decreasing weight thresholds and tracking the birth/death of topological features.

Usage

```
persistent_homology(x, n_steps = 20L, max_dim = 3L)
```

Arguments

| | |
|---------|---|
| x | A square matrix, tna, or netobject. |
| n_steps | Number of filtration steps (default 20). |
| max_dim | Maximum simplex dimension to track (default 3). |

Value

A persistent_homology object with:

betti_curve Data frame: threshold, dimension, betti at each filtration step.

persistence Data frame of birth-death pairs: dimension, birth, death, persistence.

thresholds Numeric vector of filtration thresholds.

Examples

```
mat <- matrix(c(0,.6,.5,.6,0,.4,.5,.4,0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A","B","C")
ph <- persistent_homology(mat, n_steps = 10)
print(ph)
```

plot.boot_glasso *Plot Method for boot_glasso*

Description

Plots bootstrap results for GLASSO networks.

Usage

```
## S3 method for class 'boot_glasso'  
plot(x, type = "edges", measure = NULL, ...)
```

Arguments

| | |
|---------|--|
| x | A boot_glasso object. |
| type | Character. Plot type: "edges" (default), "stability", "edge_diff", "centrality_diff", or "inclusion". |
| measure | Character. Centrality measure for type = "centrality_diff" (default: first available measure). |
| ... | Additional arguments passed to plotting functions. For type = "edge_diff" and type = "centrality_diff", accepts order: "sample" (default, sorted by value) or "id" (alphabetical). |

Value

A ggplot object, invisibly.

Examples

```
set.seed(1)  
dat <- as.data.frame(matrix(rnorm(60), ncol = 3))  
bg <- boot_glasso(dat, iter = 10, cs_iter = 5, centrality = "strength")  
plot(bg, type = "edges")  
  
set.seed(42)  
mat <- matrix(rnorm(60), ncol = 4)  
colnames(mat) <- LETTERS[1:4]  
boot <- boot_glasso(as.data.frame(mat), iter = 20, cs_iter = 10,  
  centrality = "strength", seed = 42)  
plot(boot, type = "edges")
```

| | |
|-----------|-----------------------------|
| plot.mcml | <i>Plot Method for mcml</i> |
|-----------|-----------------------------|

Description

Plots an MCML network. When **cograph** is available, delegates to `cograph::plot_mcml()` which renders a two-layer visualization (macro summary on top, within-cluster detail on bottom). Otherwise, converts to a `netobject_group` and plots each layer as a separate panel.

Usage

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

Arguments

| | |
|-----|---|
| x | An mcml object. |
| ... | Additional arguments passed to <code>cograph::plot_mcml()</code> (e.g., <code>colors</code> , <code>edge_labels</code> , <code>mode</code>). |

Value

The input object, invisibly.

Examples

```
## Not run:  
seqs <- data.frame(  
  T1 = sample(LETTERS[1:6], 30, TRUE),  
  T2 = sample(LETTERS[1:6], 30, TRUE),  
  T3 = sample(LETTERS[1:6], 30, TRUE)  
)  
clusters <- list(G1 = c("A", "B", "C"), G2 = c("D", "E", "F"))  
cs <- build_mcml(seqs, clusters)  
plot(cs)  
  
## End(Not run)
```

plot.mmm_compare *Plot Method for mmm_compare*

Description

Plot Method for mmm_compare

Usage

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

Arguments

x An mmm_compare object.
... Additional arguments (ignored).

Value

A ggplot object, invisibly.

Examples

```
seqs <- data.frame(V1 = sample(c("A","B","C"), 30, TRUE),
                   V2 = sample(c("A","B","C"), 30, TRUE))
cmp <- compare_mmm(seqs, k = 2:3, n_starts = 1, max_iter = 10, seed = 1)
plot(cmp)

set.seed(1)
seqs <- data.frame(
  V1 = sample(c("A","B","C"), 30, TRUE),
  V2 = sample(c("A","B","C"), 30, TRUE),
  V3 = sample(c("A","B","C"), 30, TRUE)
)
cmp <- compare_mmm(seqs, k = 2:3, n_starts = 5, seed = 1)
plot(cmp)
```

plot.net_association_rules
 Plot Method for net_association_rules

Description

Scatter plot of association rules: support vs confidence, with point size proportional to lift.

Usage

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

Arguments

x A net_association_rules object.
 ... Additional arguments passed to ggplot2 functions.

Value

A ggplot object, invisibly.

Examples

```
trans <- list(c("A","B","C"), c("A","B"), c("B","C","D"),
             c("A","C","D"), c("A","B","D"), c("B","C"))
rules <- association_rules(trans, min_support = 0.3, min_confidence = 0.3,
                          min_lift = 0)
plot(rules)
```

plot.net_clustering *Plot Sequence Clustering Results*

Description

Plot Sequence Clustering Results

Usage

```
## S3 method for class 'net_clustering'
plot(x, type = c("silhouette", "mds", "heatmap", "predictors"), ...)
```

Arguments

x A net_clustering object.
 type Character. Plot type: "silhouette" (per-observation silhouette bars), "mds"
 (2D MDS projection), or "heatmap" (distance matrix heatmap ordered by cluster).
 Default: "silhouette".
 ... Additional arguments (currently unused).

Value

A ggplot object (invisibly).

Examples

```
seqs <- data.frame(V1 = c("A", "B", "C", "A", "B"), V2 = c("B", "C", "A", "B", "A"),
                  V3 = c("C", "A", "B", "C", "B"))
cl <- cluster_data(seqs, k = 2)
plot(cl, type = "silhouette")

set.seed(1)
seqs <- data.frame(
  V1 = sample(c("A", "B", "C"), 20, TRUE),
  V2 = sample(c("A", "B", "C"), 20, TRUE),
  V3 = sample(c("A", "B", "C"), 20, TRUE)
)
cl <- cluster_data(seqs, k = 2)
plot(cl, type = "silhouette")
```

plot.net_gimme

Plot Method for net_gimme

Description

Plot Method for net_gimme

Usage

```
## S3 method for class 'net_gimme'
plot(
  x,
  type = c("temporal", "contemporaneous", "individual", "counts", "fit"),
  subject = NULL,
  ...
)
```

Arguments

| | |
|---------|--|
| x | A net_gimme object. |
| type | Character. Plot type: "temporal", "contemporaneous", "individual", "counts", or "fit". |
| subject | Integer or character. Subject to plot for type = "individual". |
| ... | Additional arguments (ignored). |

Value

The input object, invisibly.

Examples

```

set.seed(1)
panel <- data.frame(
  id = rep(1:5, each = 20),
  t = rep(seq_len(20), 5),
  A = rnorm(100), B = rnorm(100), C = rnorm(100)
)
gm <- build_gimme(panel, vars = c("A","B","C"), id = "id", time = "t")
plot(gm, type = "temporal")

```

plot.net_honem

*Plot Method for net_honem***Description**

Plot Method for net_honem

Usage

```

## S3 method for class 'net_honem'
plot(x, dims = c(1L, 2L), ...)

```

Arguments

| | |
|------|--|
| x | A net_honem object. |
| dims | Integer vector of length 2. Dimensions to plot (default: c(1, 2)). |
| ... | Additional arguments passed to plot . |

Value

The input object, invisibly.

Examples

```

seqs <- list(c("A","B","C","D"), c("A","B","C","A"), c("B","C","D","A"))
hem <- build_honem(build_hon(seqs, max_order = 2), dim = 2)
plot(hem)

```

```

seqs <- list(c("A","B","C","D"), c("A","B","C","A"), c("B","C","D","A"))
hon <- build_hon(seqs, max_order = 3)
he <- build_honem(hon, dim = 2)
plot(he)

```

```
plot.net_link_prediction
```

Plot Method for net_link_prediction

Description

Plots predicted links overlaid on the existing network. Requires the **cograph** package for `splot()`.

Usage

```
## S3 method for class 'net_link_prediction'
plot(x, method = NULL, top_n = 5L, ...)
```

Arguments

| | |
|---------------------|---|
| <code>x</code> | A <code>net_link_prediction</code> object. |
| <code>method</code> | Character. Which method's predictions to plot. Default: first method. |
| <code>top_n</code> | Integer. Number of top predictions to show. Default: 5. |
| <code>...</code> | Additional arguments passed to <code>cograph::splot()</code> . |

Value

The input object, invisibly.

Examples

```
## Not run:
net <- build_network(seqs, method = "relative")
pred <- predict_links(net)
plot(pred, top_n = 10)

## End(Not run)
```

```
plot.net_mmm
```

Plot Method for net_mmm

Description

Plot Method for `net_mmm`

Usage

```
## S3 method for class 'net_mmm'
plot(x, type = c("posterior", "covariates"), ...)
```

Arguments

x A net_mmm object.
 type Character. Plot type: "posterior" (default) or "covariates".
 ... Additional arguments (ignored).

Value

A ggplot object, invisibly.

Examples

```
seqs <- data.frame(V1 = sample(c("A","B","C"), 30, TRUE),
                  V2 = sample(c("A","B","C"), 30, TRUE))
mmm <- build_mmm(seqs, k = 2, n_starts = 1, max_iter = 10, seed = 1)
plot(mmm, type = "posterior")

set.seed(1)
seqs <- data.frame(
  V1 = sample(c("A","B","C"), 30, TRUE),
  V2 = sample(c("A","B","C"), 30, TRUE),
  V3 = sample(c("A","B","C"), 30, TRUE)
)
mmm <- build_mmm(seqs, k = 2, n_starts = 5, seed = 1)
plot(mmm, type = "posterior")
```

 plot.net_mogen

Plot Method for net_mogen

Description

Plot Method for net_mogen

Usage

```
## S3 method for class 'net_mogen'
plot(x, type = c("ic", "likelihood"), ...)
```

Arguments

x A net_mogen object.
 type Character. Plot type: "ic" (default) or "likelihood".
 ... Additional arguments passed to `plot`.

Value

The input object, invisibly.

Examples

```
seqs <- list(c("A","B","C","D"), c("A","B","C","A"), c("B","C","D","A"))
mg <- build_mogen(seqs, max_order = 2)
plot(mg)
```

```
seqs <- data.frame(
  V1 = c("A","B","C","A","B"),
  V2 = c("B","C","A","B","C"),
  V3 = c("C","A","B","C","A")
)
mog <- build_mogen(seqs, max_order = 2L)
plot(mog, type = "ic")
```

plot.net_reliability *Plot Method for net_reliability*

Description

Density plots of split-half metrics faceted by metric type. Multi-model comparisons show overlaid densities colored by model.

Usage

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

Arguments

```
x          A net_reliability object.
...        Additional arguments (ignored).
```

Value

A ggplot object (invisibly).

Examples

```
net <- build_network(data.frame(V1 = c("A","B","C","A"),
  V2 = c("B","C","A","B")), method = "relative")
rel <- reliability(net, iter = 10)
plot(rel)
```

```
set.seed(1)
seqs <- data.frame(
  V1 = sample(c("A","B","C"), 30, TRUE),
  V2 = sample(c("A","B","C"), 30, TRUE),
```

```
V3 = sample(c("A","B","C"), 30, TRUE)
)
net <- build_network(seqs, method = "relative")
rel <- reliability(net, iter = 20, seed = 1)
plot(rel)
```

plot.net_stability *Plot Method for net_stability*

Description

Plots mean correlation vs drop proportion for each centrality measure. The CS-coefficient is marked where the curve crosses the threshold.

Usage

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

Arguments

x A net_stability object.
... Additional arguments (ignored).

Value

A ggplot object (invisibly).

Examples

```
net <- build_network(data.frame(V1 = c("A","B","C","A"),
  V2 = c("B","C","A","B")), method = "relative")
cs <- centrality_stability(net, iter = 10, drop_prop = 0.3)
plot(cs)

set.seed(1)
seqs <- data.frame(
  V1 = sample(c("A","B","C"), 30, TRUE),
  V2 = sample(c("A","B","C"), 30, TRUE),
  V3 = sample(c("A","B","C"), 30, TRUE)
)
net <- build_network(seqs, method = "relative")
stab <- centrality_stability(net, measures = c("InStrength","OutStrength"),
  iter = 10)

plot(stab)
```

```
plot.persistent_homology
```

Plot Persistent Homology

Description

Two panels: Betti curve (threshold vs Betti number) and persistence diagram (birth vs death).

Usage

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

Arguments

```
x          A persistent_homology object.
...        Ignored.
```

Value

A grid grob (invisibly).

Examples

```
seqs <- data.frame(
  V1 = c("A", "B", "C", "A", "B"),
  V2 = c("B", "C", "A", "B", "C"),
  V3 = c("C", "A", "B", "C", "A")
)
net <- build_network(seqs, method = "relative")
ph <- persistent_homology(net)
if (requireNamespace("gridExtra", quietly = TRUE)) plot(ph)
```

```
plot.q_analysis
```

Plot Q-Analysis

Description

Two panels: Q-vector (components at each connectivity level) and structure vector (max simplex dimension per node).

Usage

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

Arguments

x A q_analysis object.
 ... Ignored.

Value

A grid grob (invisibly).

Examples

```
seqs <- data.frame(
  V1 = c("A", "B", "C", "A", "B"),
  V2 = c("B", "C", "A", "B", "C"),
  V3 = c("C", "A", "B", "C", "A")
)
net <- build_network(seqs, method = "relative")
sc <- build_simplicial(net, type = "clique")
qa <- q_analysis(sc)
plot(qa)
```

plot.simplicial_complex

Plot a Simplicial Complex

Description

Produces a multi-panel summary: f-vector, simplicial degree ranking, and degree-by-dimension heatmap.

Usage

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

Arguments

x A simplicial_complex object.
 ... Ignored.

Value

A grid grob (invisibly).

Examples

```
mat <- matrix(c(0,.6,.5,.6,0,.4,.5,.4,0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A","B","C")
sc <- build_simplicial(mat, threshold = 0.3)
if (requireNamespace("gridExtra", quietly = TRUE)) plot(sc)
```

| | |
|----------------|------------------------------------|
| predictability | <i>Compute Node Predictability</i> |
|----------------|------------------------------------|

Description

Computes the proportion of variance explained (R^2) for each node in the network, following Haslbeck & Waldorp (2018).

For method = "glasso" or "pcor", predictability is computed analytically from the precision matrix:

$$R_j^2 = 1 - 1/\Omega_{jj}$$

where Ω is the precision (inverse correlation) matrix.

For method = "cor", predictability is the multiple R^2 from regressing each node on its network neighbors (nodes with non-zero edges).

Usage

```
predictability(object, ...)

## S3 method for class 'netobject'
predictability(object, ...)

## S3 method for class 'netobject_ml'
predictability(object, ...)
```

Arguments

| | |
|--------|-------------------------------------|
| object | A netobject or netobject_ml object. |
| ... | Additional arguments (ignored). |

Value

For netobject: a named numeric vector of R^2 values (one per node, between 0 and 1).

For netobject_ml: a list with elements \$between and \$within, each a named numeric vector.

A named numeric vector of predictability values per node.

A list with within and between predictability vectors.

References

Haslbeck, J. M. B., & Waldorp, L. J. (2018). How well do network models predict observations? On the importance of predictability in network models. *Behavior Research Methods*, 50(2), 853–861. doi:10.3758/s134280170910x

Examples

```
set.seed(42)
mat <- matrix(rnorm(60), ncol = 4)
colnames(mat) <- LETTERS[1:4]
net <- build_network(as.data.frame(mat), method = "glasso")
predictability(net)
```

predict_links

Predict Missing or Future Links in a Network

Description

Computes link prediction scores for all node pairs using one or more structural similarity methods. Accepts netobject, mcml, cograph_network, or a raw weight matrix.

All methods are fully vectorized using matrix operations — no loops. Supports both weighted and binary adjacency, directed and undirected networks.

Usage

```
predict_links(
  x,
  methods = c("common_neighbors", "resource_allocation", "adamic_adar", "jaccard",
    "preferential_attachment", "katz"),
  weighted = TRUE,
  top_n = NULL,
  exclude_existing = TRUE,
  include_self = FALSE,
  katz_damping = NULL
)
```

Arguments

| | |
|----------|---|
| x | A netobject, mcml, cograph_network, or numeric square matrix. |
| methods | Character vector. One or more of: "common_neighbors", "resource_allocation", "adamic_adar", "jaccard", "preferential_attachment", "katz". Default: all six methods. |
| weighted | Logical. If TRUE, use the weight matrix directly instead of binarizing. Default: TRUE. |

| | |
|------------------|---|
| top_n | Integer or NULL. Return only the top N predictions per method. Default: NULL (all pairs). |
| exclude_existing | Logical. If TRUE, exclude node pairs that already have an edge. Default: TRUE. |
| include_self | Logical. If TRUE, include self-loop predictions. Default: FALSE. |
| katz_damping | Numeric or NULL. Attenuation factor for Katz index. If NULL, auto-computed as $0.9 / \text{spectral_radius}(A)$. Default: NULL. |

Details

Methods:

common_neighbors Number of shared neighbors. For directed graphs, sums shared out-neighbors and shared in-neighbors. Vectorized as $A \%*\% t(A) + t(A) \%* \% A$.

resource_allocation Zhou et al. (2009). Like common neighbors but weights each shared neighbor z by $1/\text{degree}(z)$. Penalizes hubs, rewards rare shared connections.

adamic_adar Adamic & Adar (2003). Like resource allocation but weights by $1/\log(\text{degree}(z))$. Less aggressive penalty than RA.

jaccard Ratio of shared neighbors to total neighbors. For directed graphs, computed on combined (out+in) neighbor sets.

preferential_attachment Product of source out-degree and target in-degree. Captures the "rich-get-richer" effect.

katz Katz (1953). Weighted sum of all paths between nodes, exponentially damped by path length. Computed via matrix inversion: $(I - \text{beta} * A)^{-1} - I$. Captures global structure.

Value

An object of class "net_link_prediction" containing:

predictions Data frame with columns: from, to, method, score, rank. Sorted by score (descending) within each method.

scores Named list of score matrices (one per method).

methods Character vector of methods used.

nodes Character vector of node names.

directed Logical.

weighted Logical.

n_nodes Integer.

n_existing Integer. Number of existing edges.

References

Liben-Nowell, D. & Kleinberg, J. (2007). The link-prediction problem for social networks. *JASIST*, 58(7), 1019–1031.

Zhou, T., Lu, L. & Zhang, Y.-C. (2009). Network topology and link prediction. *European Physical Journal B*, 71, 623–630.

Adamic, L. A. & Adar, E. (2003). Friends and neighbors on the Web. *Social Networks*, 25(3), 211–230.

Katz, L. (1953). A new status index derived from sociometric analysis. *Psychometrika*, 18(1), 39–43.

See Also

[evaluate_links](#) for prediction evaluation, [build_network](#) for network estimation.

Examples

```
seqs <- data.frame(  
  V1 = sample(LETTERS[1:5], 50, TRUE),  
  V2 = sample(LETTERS[1:5], 50, TRUE),  
  V3 = sample(LETTERS[1:5], 50, TRUE)  
)  
net <- build_network(seqs, method = "relative")  
pred <- predict_links(net)  
print(pred)  
summary(pred)
```

prepare_data

Prepare Event Log Data for Network Estimation

Description

Converts event log data (actor, action, time) into wide sequence format suitable for [build_network](#). Automatically parses timestamps, detects sessions from time gaps, and handles tie-breaking.

Usage

```
prepare_data(  
  data,  
  actor,  
  action,  
  time = NULL,  
  order = NULL,  
  session = NULL,  
  time_threshold = 900,  
  custom_format = NULL,  
  is_unix_time = FALSE,  
  unix_time_unit = c("seconds", "milliseconds", "microseconds")  
)
```

Arguments

| | |
|-----------------------------|---|
| <code>data</code> | Data frame with event log columns. |
| <code>actor</code> | Character or character vector. Column name(s) identifying who performed the action (e.g. "student" or <code>c("student", "group")</code>). If missing, all data is treated as one actor. |
| <code>action</code> | Character. Column name containing the action/state/code. |
| <code>time</code> | Character or NULL. Column name containing timestamps. Supports ISO8601, Unix timestamps (numeric), and 40+ date/time formats. If NULL, row order defines the sequence. Default: NULL. |
| <code>order</code> | Character or NULL. Column name for tie-breaking when timestamps are identical. If NULL, original row order is used. Default: NULL. |
| <code>session</code> | Character, character vector, or NULL. Column name(s) for explicit session grouping (e.g. "course" or <code>c("course", "semester")</code>). When combined with <code>time</code> , sessions are further split by time gaps. Default: NULL. |
| <code>time_threshold</code> | Numeric. Maximum gap in seconds between consecutive events before a new session starts. Only used when <code>time</code> is provided. Default: 900 (15 minutes). |
| <code>custom_format</code> | Character or NULL. Custom <code>strptime</code> format string for parsing timestamps. Default: NULL (auto-detect). |
| <code>is_unix_time</code> | Logical. If TRUE, treat numeric time values as Unix timestamps. Default: FALSE (auto-detected for numeric columns). |
| <code>unix_time_unit</code> | Character. Unit for Unix timestamps: "seconds", "milliseconds", or "microseconds". Default: "seconds". |

Value

A list with class "nestimate_data" containing:

sequence_data Data frame in wide format (one row per session, columns T1, T2, ...).

long_data The processed long-format data with session IDs.

meta_data Session-level metadata (session ID, actor).

time_data Parsed time values in wide format (if time provided).

statistics List with `total_sessions`, `total_actions`, `max_sequence_length`, `unique_actors`, etc.

See Also

[build_network](#), [prepare_onehot](#)

Examples

```
df <- data.frame(
  student = rep(1:3, each = 5),
  code = sample(c("read", "write", "test"), 15, replace = TRUE),
  timestamp = seq.POSIXt(as.POSIXct("2024-01-01"), by = "min", length.out = 15)
)
prepared <- prepare_data(df, actor = "student", action = "code",
```

```

                                time = "timestamp")
net <- build_network(prepared$sequence_data, method = "relative")

```

```
prepare_for_tna      Prepare Data for TNA Analysis
```

Description

Prepare simulated or real data for use with `tna::tna()` and related functions. Handles various input formats and ensures the output is compatible with TNA models.

Usage

```

prepare_for_tna(
  data,
  type = c("sequences", "long", "auto"),
  state_names = NULL,
  id_col = "Actor",
  time_col = "Time",
  action_col = "Action",
  validate = TRUE
)

```

Arguments

| | |
|--------------------------|---|
| <code>data</code> | Data frame containing sequence data. |
| <code>type</code> | Character. Type of input data: "sequences" Wide format with one row per sequence (default). "long" Long format with one row per action. "auto" Automatically detect format based on column names. |
| <code>state_names</code> | Character vector. Expected state names, or NULL to extract from data. Default: NULL. |
| <code>id_col</code> | Character. Name of ID column for long format data. Default: "Actor". |
| <code>time_col</code> | Character. Name of time column for long format data. Default: "Time". |
| <code>action_col</code> | Character. Name of action column for long format data. Default: "Action". |
| <code>validate</code> | Logical. Whether to validate that all actions are in <code>state_names</code> . Default: TRUE. |

Details

This function performs several preparations:

1. Converts long format to wide format if needed.
2. Validates that all actions/states are recognized.
3. Removes any non-sequence columns (e.g., id, metadata).
4. Converts factors to characters.
5. Ensures consistent column naming (V1, V2, ...).

Value

A data frame ready for use with TNA functions. For "sequences" type, returns a data frame where each row is a sequence and columns are time points (V1, V2, ...). For "long" type, converts to wide format first.

See Also

[wide_to_long](#), [long_to_wide](#) for format conversions.

Examples

```
# From wide format sequences
sequences <- data.frame(
  V1 = c("A", "B", "C", "A"), V2 = c("B", "C", "A", "B"),
  V3 = c("C", "A", "B", "C"), V4 = c("A", "B", "A", "B")
)
tna_data <- prepare_for_tna(sequences, type = "sequences")
```

```
prepare_onehot
```

Import One-Hot Encoded Data into Sequence Format

Description

Converts binary indicator (one-hot) data into the wide sequence format expected by [build_network](#) and `tna::tna()`. Each binary column represents a state; rows where the value is 1 are marked with the column name. Supports optional windowed aggregation.

Usage

```
prepare_onehot(
  data,
  cols,
  actor = NULL,
  session = NULL,
  interval = NULL,
  window_size = 1L,
  window_type = c("non-overlapping", "overlapping"),
  aggregate = FALSE
)
```

Arguments

| | |
|--------------------|--|
| <code>data</code> | Data frame with binary (0/1) indicator columns. |
| <code>cols</code> | Character vector. Names of the one-hot columns to use. |
| <code>actor</code> | Character or NULL. Name of the actor/ID column. If NULL, all rows are treated as a single sequence. Default: NULL. |

| | |
|-------------|--|
| session | Character or NULL. Name of the session column for sub-grouping within actors. Default: NULL. |
| interval | Integer or NULL. Number of rows per time point in the output. If NULL, all rows become a single time point group. Default: NULL. |
| window_size | Integer. Number of consecutive rows to aggregate into each window. Default: 1 (no windowing). |
| window_type | Character. "non-overlapping" (fixed, separate windows) or "overlapping" (rolling, step = 1). Default: "non-overlapping". |
| aggregate | Logical. If TRUE, aggregate within each window by taking the first non-NA indicator per column. Default: FALSE. |

Value

A data frame in wide format with columns named $W\{\text{window}\}_T\{\text{time}\}$ where each cell contains a state name or NA. Attributes `windowed`, `window_size`, `window_span` are set on the result.

See Also

[action_to_onehot](#) for the reverse conversion.

Examples

```
# Simple binary data
df <- data.frame(
  A = c(1, 0, 1, 0, 1),
  B = c(0, 1, 0, 1, 0),
  C = c(0, 0, 0, 0, 0)
)
seq_data <- prepare_onehot(df, cols = c("A", "B", "C"))

# With actor grouping
df$actor <- c(1, 1, 1, 2, 2)
seq_data <- prepare_onehot(df, cols = c("A", "B", "C"), actor = "actor")

# With windowing
seq_data <- prepare_onehot(df, cols = c("A", "B", "C"),
  window_size = 2, window_type = "non-overlapping")
```

print.boot_glasso *Print Method for boot_glasso*

Description

Print Method for boot_glasso

Usage

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

Arguments

x A boot_glasso object.
... Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
set.seed(1)  
dat <- as.data.frame(matrix(rnorm(60), ncol = 3))  
bg <- boot_glasso(dat, iter = 10, cs_iter = 5, centrality = "strength")  
print(bg)
```

```
set.seed(42)  
mat <- matrix(rnorm(60), ncol = 4)  
colnames(mat) <- LETTERS[1:4]  
boot <- boot_glasso(as.data.frame(mat), iter = 20, cs_iter = 10,  
                  centrality = "strength", seed = 42)  
print(boot)
```

print.mcml

Print Method for mcml

Description

Print Method for mcml

Usage

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

Arguments

x An mcml object.
... Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
seqs <- data.frame(V1 = c("A", "B", "C", "A"), V2 = c("B", "C", "A", "B"))
clusters <- list(G1 = c("A", "B"), G2 = c("C"))
cs <- build_mcml(seqs, clusters)
print(cs)
```

```
seqs <- data.frame(
  T1 = c("A", "B", "A"), T2 = c("B", "C", "B"),
  T3 = c("C", "A", "C"), T4 = c("A", "B", "A")
)
clusters <- c("Alpha", "Beta", "Alpha")
cs <- build_mcml(seqs, clusters, type = "raw")
print(cs)
```

```
print.mmm_compare      Print Method for mmm_compare
```

Description

Print Method for mmm_compare

Usage

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

Arguments

```
x          An mmm_compare object.
...        Additional arguments (ignored).
```

Value

The input object, invisibly.

Examples

```
seqs <- data.frame(V1 = sample(c("A", "B", "C"), 30, TRUE),
                  V2 = sample(c("A", "B", "C"), 30, TRUE))
cmp <- compare_mmm(seqs, k = 2:3, n_starts = 1, max_iter = 10, seed = 1)
print(cmp)
```

```
set.seed(1)
seqs <- data.frame(
  V1 = sample(c("A", "B", "C"), 30, TRUE),
  V2 = sample(c("A", "B", "C"), 30, TRUE),
  V3 = sample(c("A", "B", "C"), 30, TRUE)
```

```
)  
cmp <- compare_mmm(seqs, k = 2:3, n_starts = 5, seed = 1)  
print(cmp)
```

print.nestimate_data *Print Method for nestimate_data*

Description

Print Method for nestimate_data

Usage

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

Arguments

| | |
|-----|---------------------------------|
| x | A nestimate_data object. |
| ... | Additional arguments (ignored). |

Value

The input object, invisibly.

Examples

```
events <- data.frame(  
  actor = c("u1", "u1", "u1", "u2", "u2", "u2"),  
  action = c("A", "B", "C", "B", "A", "C"),  
  time = c(1, 2, 3, 1, 2, 3)  
)  
nd <- prepare_data(events, action = "action",  
                   actor = "actor", time = "time")  
print(nd)
```

print.netobject *Print Method for Network Object*

Description

Print Method for Network Object

Usage

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

Arguments

x A netobject.
... Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
seqs <- data.frame(V1 = c("A","B","C","A"), V2 = c("B","C","A","B"))  
net <- build_network(seqs, method = "relative")  
print(net)  
  
seqs <- data.frame(  
  V1 = c("A","B","A","C"), V2 = c("B","C","B","A"),  
  V3 = c("C","A","C","B")  
)  
net <- build_network(seqs, method = "relative")  
print(net)
```

print.netobject_group *Print Method for Group Network Object*

Description

Print Method for Group Network Object

Usage

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

Arguments

x A netobject_group.
 ... Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
seqs <- data.frame(V1 = c("A", "B", "A", "B"), V2 = c("B", "A", "B", "A"),
                  grp = c("X", "X", "Y", "Y"))
nets <- build_network(seqs, method = "relative", group = "grp")
print(nets)

seqs <- data.frame(
  V1 = c("A", "B", "A", "C", "B", "A"),
  V2 = c("B", "C", "B", "A", "C", "B"),
  V3 = c("C", "A", "C", "B", "A", "C"),
  grp = c("X", "X", "X", "Y", "Y", "Y")
)
nets <- build_network(seqs, method = "relative", group = "grp")
print(nets)
```

`print.netobject_ml` *Print Method for Multilevel Network Object*

Description

Print Method for Multilevel Network Object

Usage

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

Arguments

x A netobject_ml.
 ... Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```

set.seed(1)
obs <- data.frame(id = rep(1:3, each = 5),
                 A = rnorm(15), B = rnorm(15), C = rnorm(15))
net_ml <- build_network(obs, method = "cor",
                      params = list(id = "id"), level = "both")
print(net_ml)

set.seed(1)
obs <- data.frame(
  id = rep(1:5, each = 8),
  A = rnorm(40), B = rnorm(40),
  C = rnorm(40), D = rnorm(40)
)
net_ml <- build_network(obs, method = "cor",
                      params = list(id = "id"), level = "both")
print(net_ml)

```

```
print.net_association_rules
```

Print Method for net_association_rules

Description

Print Method for net_association_rules

Usage

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

Arguments

```
x          A net_association_rules object.
...        Additional arguments (ignored).
```

Value

The input object, invisibly.

Examples

```

trans <- list(c("A","B","C"), c("A","B"), c("B","C","D"), c("A","C","D"))
rules <- association_rules(trans, min_support = 0.3, min_confidence = 0.5,
                          min_lift = 0)
print(rules)

```

```
print.net_bootstrap    Print Method for net_bootstrap
```

Description

Print Method for net_bootstrap

Usage

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

Arguments

| | |
|-----|---------------------------------|
| x | A net_bootstrap object. |
| ... | Additional arguments (ignored). |

Value

The input object, invisibly.

Examples

```
net <- build_network(data.frame(V1 = c("A","B","C"), V2 = c("B","C","A")),  
  method = "relative")  
boot <- bootstrap_network(net, iter = 10)  
print(boot)  
  
set.seed(1)  
seqs <- data.frame(  
  V1 = c("A","B","A","C","B"), V2 = c("B","C","B","A","C"),  
  V3 = c("C","A","C","B","A")  
)  
net <- build_network(seqs, method = "relative")  
boot <- bootstrap_network(net, iter = 20)  
print(boot)
```

```
print.net_bootstrap_group  
    Print Method for net_bootstrap_group
```

Description

Print Method for net_bootstrap_group

Usage

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

Arguments

```
x          A net_bootstrap_group object.
...        Ignored.
```

Value

x invisibly.

Examples

```
seqs <- data.frame(V1 = c("A", "B", "A", "C"), V2 = c("B", "C", "C", "A"),
  V3 = c("C", "A", "B", "B"), grp = c("X", "X", "Y", "Y"))
nets <- build_network(seqs, method = "relative", group = "grp")
boot <- bootstrap_network(nets, iter = 10)
print(boot)

set.seed(1)
seqs <- data.frame(
  V1 = c("A", "B", "A", "C", "B", "A"),
  V2 = c("B", "C", "B", "A", "C", "B"),
  V3 = c("C", "A", "C", "B", "A", "C"),
  grp = c("X", "X", "X", "Y", "Y", "Y")
)
nets <- build_network(seqs, method = "relative", group = "grp")
boot <- bootstrap_network(nets, iter = 20)
print(boot)
```

print.net_clustering *Print Method for net_clustering*

Description

Print Method for net_clustering

Usage

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

Arguments

```
x          A net_clustering object.
...        Additional arguments (ignored).
```

Value

The input object, invisibly.

Examples

```
seqs <- data.frame(V1 = c("A", "B", "C", "A", "B"), V2 = c("B", "C", "A", "B", "A"),
                  V3 = c("C", "A", "B", "C", "B"))
cl <- cluster_data(seqs, k = 2)
print(cl)

set.seed(1)
seqs <- data.frame(
  V1 = sample(c("A", "B", "C"), 20, TRUE),
  V2 = sample(c("A", "B", "C"), 20, TRUE),
  V3 = sample(c("A", "B", "C"), 20, TRUE)
)
cl <- cluster_data(seqs, k = 2)
print(cl)
```

print.net_gimme

Print Method for net_gimme

Description

Print Method for net_gimme

Usage

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

Arguments

x A net_gimme object.
... Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```

set.seed(1)
panel <- data.frame(
  id = rep(1:5, each = 20),
  t = rep(seq_len(20), 5),
  A = rnorm(100), B = rnorm(100), C = rnorm(100)
)
gm <- build_gimme(panel, vars = c("A","B","C"), id = "id", time = "t")
print(gm)

```

print.net_hon

Print Method for net_hon

Description

Print Method for net_hon

Usage

```

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

```

Arguments

| | |
|-----|---------------------------------|
| x | A net_hon object. |
| ... | Additional arguments (ignored). |

Value

The input object, invisibly.

Examples

```

seqs <- list(c("A","B","C","D"), c("A","B","C","A"), c("B","C","D","A"))
hon <- build_hon(seqs, max_order = 2)
print(hon)

```

```

seqs <- data.frame(
  V1 = c("A","B","C","A","B"),
  V2 = c("B","C","A","B","C"),
  V3 = c("C","A","B","C","A")
)
hon <- build_hon(seqs, max_order = 2L)
print(hon)

```

print.net_honem *Print Method for net_honem*

Description

Print Method for net_honem

Usage

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

Arguments

x A net_honem object.
 ... Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
seqs <- list(c("A","B","C","D"), c("A","B","C","A"), c("B","C","D","A"))
hem <- build_honem(build_hon(seqs, max_order = 2), dim = 2)
print(hem)
```

```
seqs <- data.frame(
  V1 = c("A","B","C","A","B"),
  V2 = c("B","C","A","B","C"),
  V3 = c("C","A","B","C","A")
)
hon <- build_hon(seqs, max_order = 2L)
honem <- build_honem(hon, dim = 2L)
print(honem)
```

print.net_hypa *Print Method for net_hypa*

Description

Print Method for net_hypa

Usage

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

Arguments

```
x          A net_hypa object.
...        Additional arguments (ignored).
```

Value

The input object, invisibly.

Examples

```
seqs <- list(c("A","B","C"), c("B","C","A"), c("A","C","B"), c("A","B","C"))
hyp <- build_hypa(seqs, k = 2)
print(hyp)
```

```
seqs <- data.frame(
  V1 = c("A","B","C","A","B","C","A","B","C","A"),
  V2 = c("B","C","A","B","C","A","B","C","A","B"),
  V3 = c("C","A","B","C","A","B","C","A","B","C"),
  V4 = c("A","B","C","A","B","C","A","B","C","A")
)
hypa <- build_hypa(seqs, k = 2L)
print(hypa)
```

```
print.net_link_prediction
```

Print Method for net_link_prediction

Description

Print Method for net_link_prediction

Usage

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

Arguments

```
x          A net_link_prediction object.
...        Additional arguments (ignored).
```

Value

The input object, invisibly.

Examples

```
seqs <- data.frame(
  V1 = sample(LETTERS[1:4], 30, TRUE),
  V2 = sample(LETTERS[1:4], 30, TRUE),
  V3 = sample(LETTERS[1:4], 30, TRUE)
)
net <- build_network(seqs, method = "relative")
pred <- predict_links(net)
print(pred)
```

print.net_mmm

Print Method for net_mmm

Description

Print Method for net_mmm

Usage

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

Arguments

x A net_mmm object.
 ... Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
seqs <- data.frame(V1 = sample(c("A","B","C"), 30, TRUE),
  V2 = sample(c("A","B","C"), 30, TRUE))
mmm <- build_mmm(seqs, k = 2, n_starts = 1, max_iter = 10, seed = 1)
print(mmm)

set.seed(1)
seqs <- data.frame(
  V1 = sample(c("A","B","C"), 30, TRUE),
  V2 = sample(c("A","B","C"), 30, TRUE),
  V3 = sample(c("A","B","C"), 30, TRUE)
)
```

```
mmm <- build_mmm(seqs, k = 2, n_starts = 5, seed = 1)
print(mmm)
```

print.net_mogen *Print Method for net_mogen*

Description

Print Method for net_mogen

Usage

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

Arguments

x A net_mogen object.
... Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
seqs <- list(c("A","B","C","D"), c("A","B","C","A"), c("B","C","D","A"))
mg <- build_mogen(seqs, max_order = 2)
print(mg)
```

```
seqs <- data.frame(
  V1 = c("A","B","C","A","B"),
  V2 = c("B","C","A","B","C"),
  V3 = c("C","A","B","C","A")
)
mog <- build_mogen(seqs, max_order = 2L)
print(mog)
```

print.net_permutation *Print Method for net_permutation*

Description

Print Method for net_permutation

Usage

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

Arguments

| | |
|-----|---------------------------------|
| x | A net_permutation object. |
| ... | Additional arguments (ignored). |

Value

The input object, invisibly.

Examples

```
s1 <- data.frame(V1 = c("A","B","C"), V2 = c("B","C","A"))  
s2 <- data.frame(V1 = c("A","C","B"), V2 = c("C","B","A"))  
n1 <- build_network(s1, method = "relative")  
n2 <- build_network(s2, method = "relative")  
perm <- permutation_test(n1, n2, iter = 10)  
print(perm)  
  
set.seed(1)  
d1 <- data.frame(V1 = c("A","B","A"), V2 = c("B","C","B"),  
                 V3 = c("C","A","C"))  
d2 <- data.frame(V1 = c("C","A","C"), V2 = c("A","B","A"),  
                 V3 = c("B","C","B"))  
net1 <- build_network(d1, method = "relative")  
net2 <- build_network(d2, method = "relative")  
perm <- permutation_test(net1, net2, iter = 20, seed = 1)  
print(perm)
```

```
print.net_permutation_group
```

Print Method for net_permutation_group

Description

Print Method for net_permutation_group

Usage

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

Arguments

x A net_permutation_group object.
... Additional arguments (ignored).

Value

x invisibly.

Examples

```
s1 <- data.frame(V1 = c("A", "B", "A", "C"), V2 = c("B", "C", "B", "A"),  
  V3 = c("C", "A", "C", "B"), grp = c("X", "X", "Y", "Y"))  
s2 <- data.frame(V1 = c("C", "A", "C", "B"), V2 = c("A", "B", "A", "C"),  
  V3 = c("B", "C", "B", "A"), grp = c("X", "X", "Y", "Y"))  
nets1 <- build_network(s1, method = "relative", group = "grp")  
nets2 <- build_network(s2, method = "relative", group = "grp")  
perm <- permutation_test(nets1, nets2, iter = 10)  
print(perm)  
  
set.seed(1)  
s1 <- data.frame(V1 = c("A", "B", "A", "C"), V2 = c("B", "C", "B", "A"),  
  V3 = c("C", "A", "C", "B"), grp = c("X", "X", "Y", "Y"))  
s2 <- data.frame(V1 = c("C", "A", "C", "B"), V2 = c("A", "B", "A", "C"),  
  V3 = c("B", "C", "B", "A"), grp = c("X", "X", "Y", "Y"))  
nets1 <- build_network(s1, method = "relative", group = "grp")  
nets2 <- build_network(s2, method = "relative", group = "grp")  
perm <- permutation_test(nets1, nets2, iter = 20, seed = 1)  
print(perm)
```

print.net_reliability *Print Method for net_reliability*

Description

Print Method for net_reliability

Usage

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

Arguments

| | |
|-----|---------------------------------|
| x | A net_reliability object. |
| ... | Additional arguments (ignored). |

Value

The input object, invisibly.

Examples

```
net <- build_network(data.frame(V1 = c("A","B","C","A"),  
  V2 = c("B","C","A","B")), method = "relative")  
rel <- reliability(net, iter = 10)  
print(rel)  
  
set.seed(1)  
seqs <- data.frame(  
  V1 = sample(c("A","B","C"), 30, TRUE),  
  V2 = sample(c("A","B","C"), 30, TRUE),  
  V3 = sample(c("A","B","C"), 30, TRUE)  
)  
net <- build_network(seqs, method = "relative")  
rel <- reliability(net, iter = 20, seed = 1)  
print(rel)
```

print.net_stability *Print Method for net_stability*

Description

Print Method for net_stability

Usage

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

Arguments

x A net_stability object.
... Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
net <- build_network(data.frame(V1 = c("A","B","C","A"),  
  V2 = c("B","C","A","B")), method = "relative")  
cs <- centrality_stability(net, iter = 10, drop_prop = 0.3)  
print(cs)  
  
set.seed(1)  
seqs <- data.frame(  
  V1 = sample(c("A","B","C"), 30, TRUE),  
  V2 = sample(c("A","B","C"), 30, TRUE),  
  V3 = sample(c("A","B","C"), 30, TRUE)  
)  
net <- build_network(seqs, method = "relative")  
stab <- centrality_stability(net, measures = c("InStrength","OutStrength"),  
  iter = 10)  
print(stab)
```

```
print.persistent_homology
      Print persistent homology results
```

Description

Print persistent homology results

Usage

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

Arguments

x A persistent_homology object.
... Additional arguments (unused).

Value

The input object, invisibly.

Examples

```
mat <- matrix(c(0,.6,.5,.6,0,.4,.5,.4,0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A","B","C")
ph <- persistent_homology(mat, n_steps = 10)
print(ph)
```

```
print.q_analysis        Print Q-analysis results
```

Description

Print Q-analysis results

Usage

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

Arguments

x A q_analysis object.
... Additional arguments (unused).

Value

The input object, invisibly.

Examples

```
mat <- matrix(c(0,.6,.5,.6,0,.4,.5,.4,0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A","B","C")
sc <- build_simplicial(mat, threshold = 0.3)
qa <- q_analysis(sc)
print(qa)
```

```
print.simplicial_complex
      Print a simplicial complex
```

Description

Print a simplicial complex

Usage

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

Arguments

- x A simplicial_complex object.
- ... Additional arguments (unused).

Value

The input object, invisibly.

Examples

```
mat <- matrix(c(0,.6,.5,.6,0,.4,.5,.4,0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A","B","C")
sc <- build_simplicial(mat, threshold = 0.3)
print(sc)
```

print.wtna_boot_mixed *Print Method for wtna_boot_mixed*

Description

Print Method for wtna_boot_mixed

Usage

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

Arguments

| | |
|-----|---------------------------------|
| x | A wtna_boot_mixed object. |
| ... | Additional arguments (ignored). |

Value

The input object, invisibly.

Examples

```
oh <- data.frame(A = c(1,0,1,0), B = c(0,1,0,1), C = c(1,1,0,0))  
mixed <- wtna(oh, method = "both")  
boot <- bootstrap_network(mixed, iter = 10)  
print(boot)  
  
set.seed(1)  
oh <- data.frame(  
  A = c(1,0,1,0,1,0,1,0),  
  B = c(0,1,0,1,0,1,0,1),  
  C = c(1,1,0,0,1,1,0,0)  
)  
mixed <- wtna(oh, method = "both")  
boot <- bootstrap_network(mixed, iter = 20)  
print(boot)
```

```
print.wtna_mixed      Print Method for wtna_mixed
```

Description

Print Method for wtna_mixed

Usage

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

Arguments

```
x          A wtna_mixed object.
...        Additional arguments (ignored).
```

Value

The input object, invisibly.

Examples

```
oh <- matrix(c(1,0,0, 0,1,0, 0,0,1, 1,0,0), nrow = 4, byrow = TRUE,
             dimnames = list(NULL, c("A","B","C")))
mixed <- wtna(oh, method = "both")
print(mixed)
```

```
oh <- data.frame(
  A = c(1,0,1,0,1,0,1,0),
  B = c(0,1,0,1,0,1,0,1),
  C = c(1,1,0,0,1,1,0,0)
)
mixed <- wtna(oh, method = "both")
print(mixed)
```

```
q_analysis      Q-Analysis
```

Description

Computes Q-connectivity structure (Atkin 1974). Two maximal simplices are q-connected if they share a face of dimension $\geq q$. Reports:

- **Q-vector:** number of connected components at each q-level
- **Structure vector:** highest simplex dimension per node

Usage

```
q_analysis(sc)
```

Arguments

sc A simplicial_complex object.

Value

A q_analysis object with \$q_vector, \$structure_vector, and \$max_q.

References

Atkin, R. H. (1974). *Mathematical Structure in Human Affairs*.

Examples

```
mat <- matrix(c(0,.6,.5,.6,0,.4,.5,.4,0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A","B","C")
sc <- build_simplicial(mat, threshold = 0.3)
q_analysis(sc)
```

register_estimator *Register a Network Estimator*

Description

Register a custom or built-in network estimator function by name. Estimators registered here can be used by [estimate_network](#) via the method parameter.

Usage

```
register_estimator(name, fn, description, directed)
```

Arguments

| | |
|-------------|---|
| name | Character. Unique name for the estimator (e.g. "relative", "glasso"). |
| fn | Function. The estimator function. Must accept data as its first argument and ... for additional parameters. Must return a list with at least: matrix (square numeric matrix), nodes (character vector), directed (logical). |
| description | Character. Short description of the estimator. |
| directed | Logical. Whether the estimator produces directed networks. |

Value

Invisible NULL.

See Also

[get_estimator](#), [list_estimators](#), [remove_estimator](#), [estimate_network](#)

Examples

```
my_fn <- function(data, ...) {
  m <- cor(data)
  diag(m) <- 0
  list(matrix = m, nodes = colnames(m), directed = FALSE)
}
register_estimator("my_cor", my_fn, "Custom correlation", directed = FALSE)
df <- data.frame(A = rnorm(20), B = rnorm(20), C = rnorm(20))
net <- build_network(df, method = "my_cor")
remove_estimator("my_cor")
```

reliability

Split-Half Reliability for Network Estimates

Description

Assesses the stability of network estimates by repeatedly splitting sequences into two halves, building networks from each half, and comparing them. Supports single-model reliability assessment and multi-model comparison with optional scaling for cross-method comparability.

For transition methods ("relative", "frequency", "co_occurrence"), uses pre-computed per-sequence count matrices for fast resampling (same infrastructure as [bootstrap_network](#)).

Usage

```
reliability(..., iter = 1000L, split = 0.5, scale = "none", seed = NULL)
```

Arguments

| | |
|-------|---|
| ... | One or more netobjects (from build_network). If unnamed, each model is auto-named from its \$method. A netobject_group is flattened into its constituent models. |
| iter | Integer. Number of split-half iterations (default: 1000). |
| split | Numeric. Fraction of sequences assigned to the first half (default: 0.5). |
| scale | Character. Scaling applied to both split-half matrices before computing metrics. One of "none" (default), "minmax", "standardize", or "proportion". Use scaling when comparing models on different scales (e.g. frequency vs relative). |
| seed | Integer or NULL. RNG seed for reproducibility. |

Value

An object of class "net_reliability" containing:

iterations Data frame with columns model, mean_dev, median_dev, cor, max_dev (one row per iteration per model).

summary Data frame with columns model, metric, mean, sd.

models Named list of the original netobjects.

iter Number of iterations.

split Split fraction.

scale Scaling method used.

See Also

[build_network](#), [bootstrap_network](#)

Examples

```
net <- build_network(data.frame(V1 = c("A","B","C","A"),
  V2 = c("B","C","A","B")), method = "relative")
rel <- reliability(net, iter = 10)

seqs <- data.frame(
  V1 = sample(LETTERS[1:4], 30, TRUE), V2 = sample(LETTERS[1:4], 30, TRUE),
  V3 = sample(LETTERS[1:4], 30, TRUE), V4 = sample(LETTERS[1:4], 30, TRUE)
)
net <- build_network(seqs, method = "relative")
rel <- reliability(net, iter = 100, seed = 42)
print(rel)
```

remove_estimator

Remove a Registered Estimator

Description

Remove a network estimator from the registry.

Usage

```
remove_estimator(name)
```

Arguments

name Character. Name of the estimator to remove.

Value

Invisible NULL.

See Also

[register_estimator](#), [list_estimators](#)

Examples

```
register_estimator("test_est", function(data, ...) diag(3),
  description = "test", directed = FALSE)
remove_estimator("test_est")
```

| | |
|-------------------|--------------------------|
| simplicial_degree | <i>Simplicial Degree</i> |
|-------------------|--------------------------|

Description

Counts how many simplices of each dimension contain each node.

Usage

```
simplicial_degree(sc, normalized = FALSE)
```

Arguments

`sc` A `simplicial_complex` object.
`normalized` Divide by maximum possible count. Default FALSE.

Value

Data frame with node, columns `d0` through `d_k`, and `total` (sum of `d1+`). Sorted by total descending.

Examples

```
mat <- matrix(c(0,.6,.5,.6,0,.4,.5,.4,0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A","B","C")
sc <- build_simplicial(mat, threshold = 0.3)
simplicial_degree(sc)
```

| | |
|----------------|---|
| srl_strategies | <i>Self-Regulated Learning Strategy Frequencies</i> |
|----------------|---|

Description

Simulated frequency counts of 9 self-regulated learning (SRL) strategies for 250 university students. Strategies are grouped into three clusters: metacognitive (Planning, Monitoring, Evaluating), cognitive (Elaboration, Organization, Rehearsal), and resource management (Help_Seeking, Time_Mgmt, Effort_Reg). Within-cluster correlations are moderate (0.3–0.6), cross-cluster correlations are weaker.

Usage

```
srl_strategies
```

Format

A data frame with 250 rows and 9 columns. Each column is an integer count of how often the student used that strategy.

Examples

```
net <- build_network(srl_strategies, method = "glasso",
                    params = list(gamma = 0.5))
net
```

| | |
|-------------------|---|
| state_frequencies | <i>Compute State Frequencies from Trajectory Data</i> |
|-------------------|---|

Description

Counts how often each state appears across all trajectories. Returns a data frame sorted by frequency (descending).

Usage

```
state_frequencies(data)
```

Arguments

`data` A list of character vectors (trajectories) or a `data.frame`.

Value

A data frame with columns: `state`, `count`, `proportion`.

Examples

```
trajs <- list(c("A","B","C"), c("A","B","A"))
state_frequencies(trajs)
```

summary.boot_glasso *Summary Method for boot_glasso*

Description

Summary Method for boot_glasso

Usage

```
## S3 method for class 'boot_glasso'
summary(object, type = "edges", ...)
```

Arguments

| | |
|--------|---|
| object | A boot_glasso object. |
| type | Character. Summary type: "edges" (default), "centrality", "cs", "predictability", or "all". |
| ... | Additional arguments (ignored). |

Value

A data frame or list of data frames depending on type.

Examples

```
set.seed(1)
dat <- as.data.frame(matrix(rnorm(60), ncol = 3))
bg <- boot_glasso(dat, iter = 10, cs_iter = 5, centrality = "strength")
summary(bg, type = "edges")
```

```
set.seed(42)
mat <- matrix(rnorm(60), ncol = 4)
colnames(mat) <- LETTERS[1:4]
boot <- boot_glasso(as.data.frame(mat), iter = 20, cs_iter = 10,
  centrality = "strength", seed = 42)
summary(boot, type = "edges")
```

summary.mcml

Summary Method for mcml

Description

Summary Method for mcml

Usage

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

Arguments

object An mcml object.
... Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
seqs <- data.frame(V1 = c("A","B","C","A"), V2 = c("B","C","A","B"))
clusters <- list(G1 = c("A","B"), G2 = c("C"))
cs <- build_mcml(seqs, clusters)
summary(cs)

seqs <- data.frame(
  T1 = c("A","B","A"), T2 = c("B","C","B"),
  T3 = c("C","A","C"), T4 = c("A","B","A")
)
clusters <- c("Alpha", "Beta", "Alpha")
cs <- build_mcml(seqs, clusters, type = "raw")
summary(cs)
```

summary.net_association_rules

Summary Method for net_association_rules

Description

Summary Method for net_association_rules

Usage

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

Arguments

```
object      A net_association_rules object.
...         Additional arguments (ignored).
```

Value

A data frame summarizing the rules, invisibly.

Examples

```
trans <- list(c("A","B","C"), c("A","B"), c("B","C","D"), c("A","C","D"))
rules <- association_rules(trans, min_support = 0.3, min_confidence = 0.5,
                           min_lift = 0)
summary(rules)
```

summary.net_bootstrap *Summary Method for net_bootstrap*

Description

Summary Method for net_bootstrap

Usage

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

Arguments

```
object      A net_bootstrap object.
...         Additional arguments (ignored).
```

Value

A data frame with edge-level bootstrap statistics.

Examples

```
net <- build_network(data.frame(V1 = c("A","B","C"), V2 = c("B","C","A")),
  method = "relative")
boot <- bootstrap_network(net, iter = 10)
summary(boot)

set.seed(1)
seqs <- data.frame(
  V1 = c("A","B","A","C","B"), V2 = c("B","C","B","A","C"),
  V3 = c("C","A","C","B","A")
)
net <- build_network(seqs, method = "relative")
boot <- bootstrap_network(net, iter = 20)
summary(boot)
```

```
summary.net_bootstrap_group
```

Summary Method for net_bootstrap_group

Description

Summary Method for net_bootstrap_group

Usage

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

Arguments

```
object      A net_bootstrap_group object.
...         Ignored.
```

Value

A data frame with group, edge, and bootstrap statistics columns.

Examples

```
seqs <- data.frame(V1 = c("A","B","A","C"), V2 = c("B","C","C","A"),
  V3 = c("C","A","B","B"), grp = c("X","X","Y","Y"))
nets <- build_network(seqs, method = "relative", group = "grp")
boot <- bootstrap_network(nets, iter = 10)
summary(boot)

set.seed(1)
seqs <- data.frame(
```

```

V1 = c("A", "B", "A", "C", "B", "A"),
V2 = c("B", "C", "B", "A", "C", "B"),
V3 = c("C", "A", "C", "B", "A", "C"),
grp = c("X", "X", "X", "Y", "Y", "Y")
)
nets <- build_network(seqs, method = "relative", group = "grp")
boot <- bootstrap_network(nets, iter = 20)
summary(boot)

```

summary.net_clustering

Summary Method for net_clustering

Description

Summary Method for net_clustering

Usage

```

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

```

Arguments

| | |
|--------|---------------------------------|
| object | A net_clustering object. |
| ... | Additional arguments (ignored). |

Value

The input object, invisibly.

Examples

```

seqs <- data.frame(V1 = c("A", "B", "C", "A", "B"), V2 = c("B", "C", "A", "B", "A"),
                  V3 = c("C", "A", "B", "C", "B"))
c1 <- cluster_data(seqs, k = 2)
summary(c1)

set.seed(1)
seqs <- data.frame(
  V1 = sample(c("A", "B", "C"), 20, TRUE),
  V2 = sample(c("A", "B", "C"), 20, TRUE),
  V3 = sample(c("A", "B", "C"), 20, TRUE)
)
c1 <- cluster_data(seqs, k = 2)
summary(c1)

```

summary.net_gimme *Summary Method for net_gimme*

Description

Summary Method for net_gimme

Usage

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

Arguments

object A net_gimme object.
... Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
set.seed(1)  
panel <- data.frame(  
  id = rep(1:5, each = 20),  
  t = rep(seq_len(20), 5),  
  A = rnorm(100), B = rnorm(100), C = rnorm(100)  
)  
gm <- build_gimme(panel, vars = c("A","B","C"), id = "id", time = "t")  
summary(gm)
```

summary.net_hon *Summary Method for net_hon*

Description

Summary Method for net_hon

Usage

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

Arguments

object A net_hon object.
... Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
seqs <- list(c("A","B","C","D"), c("A","B","C","A"), c("B","C","D","A"))
hon <- build_hon(seqs, max_order = 2)
summary(hon)
```

```
seqs <- data.frame(
  V1 = c("A","B","C","A","B"),
  V2 = c("B","C","A","B","C"),
  V3 = c("C","A","B","C","A")
)
hon <- build_hon(seqs, max_order = 2L)
summary(hon)
```

summary.net_honem *Summary Method for net_honem*

Description

Summary Method for net_honem

Usage

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

Arguments

object A net_honem object.
... Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
seqs <- list(c("A","B","C","D"), c("A","B","C","A"), c("B","C","D","A"))
hem <- build_honem(build_hon(seqs, max_order = 2), dim = 2)
summary(hem)
```

```
seqs <- list(c("A","B","C","D"), c("A","B","C","A"), c("B","C","D","A"))
hon <- build_hon(seqs, max_order = 3)
he <- build_honem(hon, dim = 2)
summary(he)
```

summary.net_hypa

Summary Method for net_hypa

Description

Summary Method for net_hypa

Usage

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

Arguments

| | |
|--------|---------------------------------|
| object | A net_hypa object. |
| ... | Additional arguments (ignored). |

Value

The input object, invisibly.

Examples

```
seqs <- list(c("A","B","C"), c("B","C","A"), c("A","C","B"), c("A","B","C"))
hyp <- build_hypa(seqs, k = 2)
summary(hyp)
```

```
seqs <- data.frame(
  V1 = c("A","B","C","A","B","C","A","B","C","A"),
  V2 = c("B","C","A","B","C","A","B","C","A","B"),
  V3 = c("C","A","B","C","A","B","C","A","B","C"),
  V4 = c("A","B","C","A","B","C","A","B","C","A")
)
hypa <- build_hypa(seqs, k = 2L)
summary(hypa)
```

```
summary.net_link_prediction
      Summary Method for net_link_prediction
```

Description

Summary Method for net_link_prediction

Usage

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

Arguments

| | |
|--------|---------------------------------|
| object | A net_link_prediction object. |
| ... | Additional arguments (ignored). |

Value

A data frame with per-method summary statistics, invisibly.

Examples

```
seqs <- data.frame(
  V1 = sample(LETTERS[1:4], 30, TRUE),
  V2 = sample(LETTERS[1:4], 30, TRUE),
  V3 = sample(LETTERS[1:4], 30, TRUE)
)
net <- build_network(seqs, method = "relative")
pred <- predict_links(net)
summary(pred)
```

summary.net_mmm

Summary Method for net_mmm

Description

Summary Method for net_mmm

Usage

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

Arguments

object A net_mmm object.
... Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
seqs <- data.frame(V1 = sample(c("A","B","C"), 30, TRUE),  
                  V2 = sample(c("A","B","C"), 30, TRUE))  
mmm <- build_mmm(seqs, k = 2, n_starts = 1, max_iter = 10, seed = 1)  
summary(mmm)  
  
set.seed(1)  
seqs <- data.frame(  
  V1 = sample(c("A","B","C"), 30, TRUE),  
  V2 = sample(c("A","B","C"), 30, TRUE),  
  V3 = sample(c("A","B","C"), 30, TRUE)  
)  
mmm <- build_mmm(seqs, k = 2, n_starts = 5, seed = 1)  
summary(mmm)
```

summary.net_mogen*Summary Method for net_mogen*

Description

Summary Method for net_mogen

Usage

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

Arguments

```
object      A net_mogen object.
...         Additional arguments (ignored).
```

Value

The input object, invisibly.

Examples

```
seqs <- list(c("A","B","C","D"), c("A","B","C","A"), c("B","C","D","A"))
mg <- build_mogen(seqs, max_order = 2)
summary(mg)
```

```
seqs <- data.frame(
  V1 = c("A","B","C","A","B"),
  V2 = c("B","C","A","B","C"),
  V3 = c("C","A","B","C","A")
)
mog <- build_mogen(seqs, max_order = 2L)
summary(mog)
```

```
summary.net_permutation
```

Summary Method for net_permutation

Description

Summary Method for net_permutation

Usage

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

Arguments

```
object      A net_permutation object.
...         Additional arguments (ignored).
```

Value

A data frame with edge-level permutation test results.

Examples

```
s1 <- data.frame(V1 = c("A", "B", "C"), V2 = c("B", "C", "A"))
s2 <- data.frame(V1 = c("A", "C", "B"), V2 = c("C", "B", "A"))
n1 <- build_network(s1, method = "relative")
n2 <- build_network(s2, method = "relative")
perm <- permutation_test(n1, n2, iter = 10)
summary(perm)

set.seed(1)
d1 <- data.frame(V1 = c("A", "B", "A"), V2 = c("B", "C", "B"),
                V3 = c("C", "A", "C"))
d2 <- data.frame(V1 = c("C", "A", "C"), V2 = c("A", "B", "A"),
                V3 = c("B", "C", "B"))
net1 <- build_network(d1, method = "relative")
net2 <- build_network(d2, method = "relative")
perm <- permutation_test(net1, net2, iter = 20, seed = 1)
summary(perm)
```

summary.net_permutation_group

Summary Method for net_permutation_group

Description

Returns a combined summary data frame across all groups.

Usage

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

Arguments

object A net_permutation_group object.
 ... Additional arguments (ignored).

Value

A data frame with group, edge, p_value, and sig columns.

Examples

```

s1 <- data.frame(V1 = c("A", "B", "A", "C"), V2 = c("B", "C", "B", "A"),
  V3 = c("C", "A", "C", "B"), grp = c("X", "X", "Y", "Y"))
s2 <- data.frame(V1 = c("C", "A", "C", "B"), V2 = c("A", "B", "A", "C"),
  V3 = c("B", "C", "B", "A"), grp = c("X", "X", "Y", "Y"))
nets1 <- build_network(s1, method = "relative", group = "grp")
nets2 <- build_network(s2, method = "relative", group = "grp")
perm <- permutation_test(nets1, nets2, iter = 10)
summary(perm)

set.seed(1)
s1 <- data.frame(V1 = c("A", "B", "A", "C"), V2 = c("B", "C", "B", "A"),
  V3 = c("C", "A", "C", "B"), grp = c("X", "X", "Y", "Y"))
s2 <- data.frame(V1 = c("C", "A", "C", "B"), V2 = c("A", "B", "A", "C"),
  V3 = c("B", "C", "B", "A"), grp = c("X", "X", "Y", "Y"))
nets1 <- build_network(s1, method = "relative", group = "grp")
nets2 <- build_network(s2, method = "relative", group = "grp")
perm <- permutation_test(nets1, nets2, iter = 20, seed = 1)
summary(perm)

```

summary.net_stability *Summary Method for net_stability*

Description

Returns the mean correlation at each drop proportion for each measure.

Usage

```

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

```

Arguments

| | |
|--------|---------------------------------|
| object | A net_stability object. |
| ... | Additional arguments (ignored). |

Value

A data frame with columns measure, drop_prop, mean_cor, sd_cor, prop_above.

Examples

```

net <- build_network(data.frame(V1 = c("A", "B", "C", "A"),
  V2 = c("B", "C", "A", "B")), method = "relative")
cs <- centrality_stability(net, iter = 10, drop_prop = 0.3)
summary(cs)

```

```

set.seed(1)
seqs <- data.frame(
  V1 = sample(c("A", "B", "C"), 30, TRUE),
  V2 = sample(c("A", "B", "C"), 30, TRUE),
  V3 = sample(c("A", "B", "C"), 30, TRUE)
)
net <- build_network(seqs, method = "relative")
stab <- centrality_stability(net, measures = c("InStrength", "OutStrength"),
                             iter = 10)

summary(stab)

```

```
summary.wtna_boot_mixed
```

Summary Method for wtna_boot_mixed

Description

Summary Method for wtna_boot_mixed

Usage

```

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

```

Arguments

| | |
|--------|---------------------------------|
| object | A wtna_boot_mixed object. |
| ... | Additional arguments (ignored). |

Value

A list with \$transition and \$cooccurrence summary data frames.

Examples

```

oh <- data.frame(A = c(1,0,1,0), B = c(0,1,0,1), C = c(1,1,0,0))
mixed <- wtna(oh, method = "both")
boot <- bootstrap_network(mixed, iter = 10)
summary(boot)

set.seed(1)
oh <- data.frame(
  A = c(1,0,1,0,1,0,1,0),
  B = c(0,1,0,1,0,1,0,1),
  C = c(1,1,0,0,1,1,0,0)
)
mixed <- wtna(oh, method = "both")

```

```
boot <- bootstrap_network(mixed, iter = 20)
summary(boot)
```

verify_simplicial *Verify Simplicial Complex Against igraph*

Description

Cross-validates clique finding and Betti numbers against igraph and known topological invariants. Useful for testing.

Usage

```
verify_simplicial(mat, threshold = 0)
```

Arguments

mat A square adjacency matrix.
threshold Edge weight threshold.

Value

A list with \$cliques_match (logical), \$n_simplices_ours, \$n_simplices_igraph, \$betti, and \$euler.

Examples

```
mat <- matrix(c(0,.6,.5,.6,0,.4,.5,.4,0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A","B","C")
verify_simplicial(mat, threshold = 0.3)
```

vibcoding-data *Human-AI Vibe Coding Interaction Data*

Description

Coded interaction sequences from 429 human-AI pair programming sessions across 34 projects. Three coding granularities: code (32 states), category (17 states), and superclass (6 states).

Usage

human_ai

human_ai_cat

human_ai_super

human_detailed

human_cat

human_super

ai_detailed

ai_cat

ai_super

human_wide

ai_wide

Format

Long-format data frames with columns:

id Integer. Turn index within the session.

project Character. Project identifier (Project_1 .. Project_34).

session_id Character. Unique session hash.

timestamp Character. ISO 8601 timestamp.

session_date Character. Date of the session (YYYY-MM-DD).

actor Character. "Human" or "AI".

code Character. Fine-grained action code (32 states).

category Character. Mid-level category (17 states).

superclass Character. High-level superclass (6 states).

An object of class `data.frame` with 19347 rows and 9 columns.

An object of class `data.frame` with 19347 rows and 9 columns.

An object of class `data.frame` with 19347 rows and 9 columns.

An object of class `data.frame` with 10796 rows and 9 columns.

An object of class `data.frame` with 10796 rows and 9 columns.

An object of class `data.frame` with 10796 rows and 9 columns.

An object of class `data.frame` with 8551 rows and 9 columns.

An object of class `data.frame` with 8551 rows and 9 columns.

An object of class `data.frame` with 8551 rows and 9 columns.

An object of class `data.frame` with 429 rows and 164 columns.

An object of class `data.frame` with 428 rows and 138 columns.

Details

Nine long-format datasets are provided, filtered by actor and named by granularity level:

| Dataset | Actor | Granularity |
|----------------|--------------|-----------------------|
| human_ai | Both | code (32 states) |
| human_ai_cat | Both | category (17 states) |
| human_ai_super | Both | superclass (6 states) |
| human_detailed | Human | code (32 states) |
| human_cat | Human | category (17 states) |
| human_super | Human | superclass (6 states) |
| ai_detailed | AI | code (32 states) |
| ai_cat | AI | category (17 states) |
| ai_super | AI | superclass (6 states) |

Two wide-format datasets at category level (rows = sessions, columns = T1, T2, ...):

| | |
|------------|---------------------------------------|
| human_wide | Human actions in wide sequence format |
| ai_wide | AI actions in wide sequence format |

Source

Saqr, M. (2026). Human-AI vibe coding interaction study.

Examples

```
# Build a transition network from human category sequences
net <- build_network(human_wide, method = "relative")
```

```
# Use the edge list directly
head(human_ai_edges)
```

wide_to_long

*Convert Wide Sequences to Long Format***Description**

Convert sequence data from wide format (one row per sequence, columns as time points) to long format (one row per action).

Usage

```
wide_to_long(
  data,
  id_col = NULL,
  time_prefix = "V",
  action_col = "Action",
  time_col = "Time",
  drop_na = TRUE
)
```

Arguments

| | |
|--------------------------|---|
| <code>data</code> | Data frame in wide format with sequences in rows. |
| <code>id_col</code> | Character. Name of the ID column, or NULL to auto-generate IDs. Default: NULL. |
| <code>time_prefix</code> | Character. Prefix for time point columns (e.g., "V" for V1, V2, ...). Default: "V". |
| <code>action_col</code> | Character. Name of the action column in output. Default: "Action". |
| <code>time_col</code> | Character. Name of the time column in output. Default: "Time". |
| <code>drop_na</code> | Logical. Whether to drop NA values. Default: TRUE. |

Details

This function converts data from the format produced by `simulate_sequences()` to the long format used by many TNA functions and analyses.

Value

A data frame in long format with columns:

id Sequence identifier (integer).

Time Time point within the sequence (integer).

Action The action/state at that time point (character).

Any additional columns from the original data are preserved.

See Also

[long_to_wide](#) for the reverse conversion, [prepare_for_tna](#) for preparing data for TNA analysis.

Examples

```
wide_data <- data.frame(
  V1 = c("A", "B", "C"), V2 = c("B", "C", "A"), V3 = c("C", "A", "B")
)
long_data <- wide_to_long(wide_data)
head(long_data)
```

wtna

Window-based Transition Network Analysis

Description

Computes networks from one-hot (binary indicator) data using temporal windowing. Supports transition (directed), co-occurrence (undirected), or both network types.

Usage

```
wtna(
  data,
  method = c("transition", "cooccurrence", "both"),
  type = c("frequency", "relative"),
  codes = NULL,
  window_size = 1L,
  mode = c("non-overlapping", "overlapping"),
  actor = NULL
)
```

Arguments

| | |
|--------------------------|---|
| <code>data</code> | Data frame with one-hot encoded columns (0/1 binary). |
| <code>method</code> | Character. Network type: "transition" (directed), "cooccurrence" (undirected), or "both" (returns list of two networks). Default: "transition". |
| <code>type</code> | Character. Output type: "frequency" (raw counts) or "relative" (row-normalized probabilities). Default: "frequency". |
| <code>codes</code> | Character vector or NULL. Names of the one-hot columns to use. If NULL, auto-detects binary columns. Default: NULL. |
| <code>window_size</code> | Integer. Number of consecutive rows to aggregate per window. Default: 1 (no windowing). |
| <code>mode</code> | Character. Window mode: "non-overlapping" (fixed, separate windows) or "overlapping" (rolling, step = 1). Default: "non-overlapping". |
| <code>actor</code> | Character or NULL. Name of the actor/ID column for per-group computation. If NULL, treats all rows as one group. Default: NULL. |

Details

Transitions: Uses `crossprod(X[-n,], X[-1,])` to count how often state *i* is active at time *t* AND state *j* at time *t+1*.

Co-occurrence: Uses `crossprod(X)` to count states that are simultaneously active in the same row.

Windowing: For `window_size > 1`, rows are aggregated into windows before computing networks. Non-overlapping windows are fixed, separate blocks; overlapping windows roll forward one row at a time. Within each window, any active indicator (1) in any row makes that state active for the window.

Per-actor: When `actor` is specified, networks are computed per group and summed.

Value

For `method = "transition" or "cooccurrence"`: a `netobject` (see [build_network](#)).

For `method = "both"`: a `wtna_mixed` object with elements `$transition` and `$cooccurrence`, each a `netobject`.

See Also

[build_network](#), [prepare_onehot](#)

Examples

```
oh <- matrix(c(1,0,0, 0,1,0, 0,0,1, 1,0,0), nrow = 4, byrow = TRUE,
             dimnames = list(NULL, c("A","B","C")))
w <- wtna(oh)

# Simple one-hot data
df <- data.frame(
  A = c(1, 0, 1, 0, 1),
  B = c(0, 1, 0, 1, 0),
  C = c(0, 0, 1, 0, 0)
)

# Transition network
net <- wtna(df)
print(net)

# Both networks
nets <- wtna(df, method = "both")
print(nets$transition)
print(nets$cooccurrence)

# With windowing
net <- wtna(df, window_size = 2, mode = "non-overlapping")
```

Index

* datasets

- group_regulation_long, 52
 - human_ai_edges, 53
 - learning_activities, 54
 - srl_strategies, 110
 - vibcoding-data, 125
- action_to_onehot, 4, 83
- aggregate_weights, 5
- ai_cat (vibcoding-data), 125
- ai_detailed (vibcoding-data), 125
- ai_super (vibcoding-data), 125
- ai_wide (vibcoding-data), 125
- as_tna, 8
- association_rules, 6
- betti_numbers, 10, 31
- boot_glasso, 12
- bootstrap_network, 10, 14, 30, 62, 107, 108
- build_gimme, 15
- build_hon, 17, 20
- build_honem, 19
- build_hypa, 21
- build_mcml, 22
- build_mmm, 24, 37, 38, 45
- build_mogen, 26
- build_network, 7, 10–12, 14, 17, 25, 27, 33–35, 38, 39, 47, 50, 61, 62, 79, 80, 82, 107, 108, 130
- build_simplicial, 30
- centrality, 31
- centrality_stability, 32
- cluster_data, 25, 34, 38, 39
- cluster_mmm, 37, 39
- cluster_network, 38, 38
- cluster_sequences (cluster_data), 34
- cluster_summary, 8, 9, 22, 23, 40, 43
- compare_mmm, 25, 45
- convert_sequence_format, 10, 45
- csum (cluster_summary), 40
- estimate_network, 47, 106, 107
- euler_characteristic, 48
- evaluate_links, 48, 79
- extract_edges, 49, 51
- extract_initial_probs, 50, 51
- extract_transition_matrix, 50, 51, 51
- frequencies, 46
- get_estimator, 52, 55, 107
- group_regulation_long, 52
- human_ai (vibcoding-data), 125
- human_ai_cat (vibcoding-data), 125
- human_ai_edges, 53
- human_ai_super (vibcoding-data), 125
- human_cat (vibcoding-data), 125
- human_detailed (vibcoding-data), 125
- human_super (vibcoding-data), 125
- human_wide (vibcoding-data), 125
- learning_activities, 53, 54
- list_estimators, 30, 52, 55, 107, 109
- long_to_wide, 56, 82, 129
- mogen_transitions, 57
- p.adjust, 62
- path_counts, 60
- pathways, 58
- permutation_test, 61
- persistent_homology, 31, 63
- plot, 69, 71
- plot.boot_glasso, 64
- plot.mcml, 65
- plot.mmm_compare, 66
- plot.net_association_rules, 66
- plot.net_clustering, 67
- plot.net_gimme, 68

plot.net_honem, 69
plot.net_link_prediction, 70
plot.net_mmm, 70
plot.net_mogen, 71
plot.net_reliability, 72
plot.net_stability, 73
plot.persistent_homology, 74
plot.q_analysis, 74
plot.simplicial_complex, 75
predict_links, 7, 77
predictability, 76
prepare_data, 79
prepare_for_tna, 56, 81, 129
prepare_onehot, 80, 82, 130
print.boot_glasso, 83
print.mcml, 84
print.mmm_compare, 85
print.nestimate_data, 86
print.net_association_rules, 89
print.net_bootstrap, 12, 90
print.net_bootstrap_group, 90
print.net_clustering, 91
print.net_gimme, 92
print.net_hon, 93
print.net_honem, 94
print.net_hypa, 94
print.net_link_prediction, 95
print.net_mmm, 96
print.net_mogen, 97
print.net_permutation, 62, 98
print.net_permutation_group, 99
print.net_reliability, 100
print.net_stability, 101
print.netobject, 87
print.netobject_group, 87
print.netobject_ml, 88
print.persistent_homology, 102
print.q_analysis, 102
print.simplicial_complex, 103
print.wtna_boot_mixed, 104
print.wtna_mixed, 105

q_analysis, 31, 105

register_estimator, 30, 52, 55, 106, 109
reliability, 34, 107
remove_estimator, 107, 108

simplicial_degree, 31, 109

srl_strategies, 53, 110
state_frequencies, 110
summary.boot_glasso, 111
summary.mcml, 112
summary.net_association_rules, 112
summary.net_bootstrap, 12, 113
summary.net_bootstrap_group, 114
summary.net_clustering, 115
summary.net_gimme, 116
summary.net_hon, 116
summary.net_honem, 117
summary.net_hypa, 118
summary.net_link_prediction, 119
summary.net_mmm, 120
summary.net_mogen, 120
summary.net_permutation, 62, 121
summary.net_permutation_group, 122
summary.net_stability, 123
summary.wtna_boot_mixed, 124

verify_simplicial, 125
vibcoding-data, 125

wagg (aggregate_weights), 5
wide_to_long, 56, 82, 128
wtna, 129