Package 'astrolabe'

September 6, 2025

Title Astrolabe: An Entropy-based Method for Causality on Quantitative Outcomes in R

Version 0.0.0.9000

Authors Fabio Chillotti <astrolabe.dispense778@passinbox.com>
Federica Grosso <federicagrosso31@gmail.com>

Description This vignette illustrates the methodology implemented in the astrolabe package, an R tool for inferring non-linear causal relationships among variables. The package combines random forest models with differential entropy estimation of residuals, followed by bootstrap-permutation testing and variable-importance-based selection.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Depends R (>= 3.5)

Imports randomForest,
parallel,
pbmcapply,
FNN,
ggplot2,
grid,
igraph,
ggraph,
dbscan,
PCAmixdata

R topics documented:

`can_coerce_numeric` *Heuristic Check for Numeric Coercibility*

Description

Tests whether a vector can be safely coerced to numeric without excessive missing values.

Usage

```
can_coerce_numeric(v, na_tol = 0.01)
```

Arguments

`v`	A vector of any type.
`na_tol`	Numeric in $[0, 1]$. Maximum tolerated fraction of `NA`s after coercion (default `0.01`).

Value

Logical. `TRUE` if coercion is safe, `FALSE` otherwise.

Examples

```
can_coerce_numeric(c("1", "2", "3"))   # TRUE
can_coerce_numeric(c("1", "a", "3"))   # FALSE (di solito)
```

causal_entropy_combinations
Causal Entropy Scan over All Predictor→Outcome Combinations

Description

For each candidate outcome in `df`, fits a tuned Random Forest to predict that outcome from the remaining variables, computes residuals, and evaluates a multivariate entropy `entropy_nd` on `cbind(predictors, residual)`. The final score per relation is `H = exp(-e_complete)`; larger is better.

Usage

```
causal_entropy_combinations(
  df,
  ntree = 500,
  mtry_grid = 1:sqrt(ncol(df) - 1),
  verbose = FALSE,
  fixed_outcome = NULL,
  always_predictors = NULL,
  as_factor = NULL
)
```

Arguments

`df`	Data frame containing all variables.
`ntree`	Integer. Trees for Random Forest fitting (default `500`).
`mtry_grid`	Integer vector for `mtry` search; default `1:sqrt(p)` with `p = ncol(df)-1`.
`verbose`	Logical. Print per-combination diagnostics (default `FALSE`).
`fixed_outcome`	Optional character scalar. If provided, only this column is used as outcome; otherwise all columns except `always_predictors` are considered outcomes in turn.
`always_predictors`	Optional character vector of predictors that must always be available; excluded from the outcome set when scanning.
`as_factor`	Optional character vector of column names to coerce to factor before fitting (also appended to `always_predictors`).

Details

For each `outcome_col`, predictors are `setdiff(colnames(df), outcome_col)`. A best RF is selected by `tune_rf`. Residuals are computed as `outcome - predict(RF)`. The entropy `e_complete` is calculated by `entropy_nd` on the numeric subset of `cbind(predictors, residual)` with `normalize = "divide"`. The returned score is `H = exp(-e_complete)`.

Value

An invisible list with:

- `entropy`: named numeric vector of `H` scores with names like `"X1 + X2 → Y"`;
- `importances`: list of per-relation named lists of RF importances (one numeric value per predictor).

See Also

tune_rf, entropy_nd

Examples

```
## Not run:
# df must contain only the variables to scan; factors are allowed.
res <- causal_entropy_combinations(df, ntree = 300, verbose = TRUE)
head(res$entropy[order(res$entropy, decreasing = TRUE)])

## End(Not run)
```

---

complete_function *End-to-End Causal Scan (multi-outcome, robust validation, and binary pairs)*

Description

Runs the full pipeline on df: (i) column preprocessing (coercion to factor or numeric with rare-level handling), (ii) multi-outcome scan via scan_all_outcomes_complete(), (iii) robust validation with bootstrap + permutation via robust_scan_all_outcomes(), and (iv) optional binary pairwise direction matrix via cor_forest_matrix_robust_perm(). Optionally draws a causal graph that combines robust multi-outcome and binary evidence.

Usage

```
complete_function(
  df,
  n_boot = 100,
  n_perm = 50,
  alpha = 0.05,
  ntree = 500,
  seed = NULL,
  n_cores = parallel::detectCores() - 1,
  verbose = TRUE,
  always_predictors = NULL,
  which = c("all", "robust", "binary"),
  categorical_thr = 35,
  quantitative_thr = 40,
  importance_method = c("fixed", "neg_exp", "net_clust"),
  prob = 0.75,
  plot = TRUE,
  curved = NULL,
  layout = "auto",
  pad = 0.4,
  arrow_len_pt = 8,
  end_cap_mm = 8,
  linewidth = 0.75,
  node_size = 20,
  node_stroke = 1,
```

```
  strength_curved = 0.6
)
```

Arguments

df	A data frame with variables to analyze (predictors + outcomes).
n_boot	Integer. Number of bootstraps (B) for robust/binary stages. Default 100.
n_perm	Integer. Number of permutations (P) for robust/binary stages. Default 50.
alpha	Numeric in (0, 1). Significance level for permutation tests. Default 0.05.
ntree	Integer. Number of trees for Random Forest fits. Default 500.
seed	Optional integer random seed.
n_cores	Integer. Parallel cores for parallel::mclapply(). Default parallel::detectCores() - 1.
verbose	Logical. If TRUE, print progress messages. Default TRUE.
always_predictors	Optional character vector; column names forced to be considered as predictors.
which	One of "all", "robust", "binary": controls which stages of the pipeline are executed. If df has exactly 2 columns, the mode is coerced to "binary".
categorical_thr	Numeric. Threshold for categorical variables in evaluate_importance(). Default 35.
quantitative_thr	Numeric. Threshold for quantitative variables in evaluate_importance(). Default 40.
importance_method	One of "fixed", "neg_exp", "net_clust"; strategy used by evaluate_importance().
prob	Numeric. Probability cutoff used when importance_method = "net_clust". Default 0.75.
plot	Logical. If TRUE, build and print a combined causal graph. Default TRUE.
curved	See draw_dag: which edges to draw as arcs (NULL, logical vector, character keys "X->Y", or data.frame(from,to)).
layout	Graph layout name passed to ggraph. Examples: "auto", "kk", "fr", "sugiyama", "linear". Default "auto".
pad	Numeric padding added around computed x/y ranges for the plot. Default 0.4.
arrow_len_pt	Arrow length (points) for directed edges. Default 8.
end_cap_mm	End cap radius (millimeters) for edge arrows. Default 8.
linewidth	Edge line width. Default 0.75.
node_size	Node point size. Default 20.
node_stroke	Node point stroke width. Default 1.
strength_curved	Curvature strength for curved edges (passed to ggraph::geom_edge_arc2()). Non-curved edges use 0. Default 0.6.

Details

Preprocessing. Columns are preprocessed to stabilize the scans: (a) low-cardinality numeric vectors are converted to factors; (b) high-cardinality non-numeric vectors are coerced to numeric when `can_coerce_numeric()` returns `TRUE` (dropping rows that cannot be safely coerced), otherwise they are factored with rare-level grouping and dominance checks; (c) strongly dominant categorical columns may be binarized or dropped. During this stage, some columns may be appended to `always_predictors`.

Scanning. If `which %in% c("all","robust")`, `scan_all_outcomes_complete()` runs first, followed by `robust_scan_all_outcomes()` which applies bootstrap (`B = n_boot`) and permutation tests (`P = n_perm`) at level `alpha`.

Binary pairs. If `which %in% c("all","binary")`, the binary direction matrix is computed via `cor_forest_matrix_robust_perm()` with its own bootstrap/permutation routine.

Graph. When `plot = TRUE`, a combined edge set is built by merging robust multi-outcome relations (labeled `"complex"`) and significant binary pairs (labeled `"pairwise"`). Overlaps are labeled `"both"`. The graph is drawn with `draw_dag` using the provided layout/curvature/appearance settings.

Value

A list with:

- `scan_res`: results from `scan_all_outcomes_complete()` (or `NULL`);
- `robust`: results from `robust_scan_all_outcomes()` (or `NULL`);
- `binary`: list of matrices from `cor_forest_matrix_robust_perm()` (or `NULL`);
- `graph`: a **ggplot** object returned by `draw_dag()` (or `NULL`);
- `res_all`: combined edge data.frame used for plotting (from/pairwise/complex/both).

See Also

`scan_all_outcomes_complete`, `robust_scan_all_outcomes`, `cor_forest_matrix_robust_perm`, `can_coerce_numeric`, `draw_dag`

Examples

```
## Not run:
set.seed(123)
out <- complete_function(
  df,
  n_boot = 50, n_perm = 200, ntree = 400,
  which = "all", verbose = TRUE,
  plot = TRUE, layout = "kk"
)

# Combined graph object (ggplot):
out$graph

# Combined edges used for plotting:
out$res_all

## End(Not run)
```

`cor_forest_matrix_robust_perm` *Robust Pairwise Direction Matrix via Bootstrap + Permutation*

Description

For every unordered variable pair (X, Y), infers the preferred direction ($X \to Y$ or $Y \to X$) on real data, then estimates robustness by bootstrap frequency and an empirical permutation test. Aggregates results into four matrices.

Usage

```
cor_forest_matrix_robust_perm(
  df,
  B = 30,
  P = 30,
  seed = NULL,
  ntree = 300,
  n_cores = parallel::detectCores() - 1,
  alpha = 0.05,
  verbose = TRUE,
  always_predictors = NULL,
  categorical_thr = 35,
  quantitative_thr = 40,
  importance_method = c("fixed", "neg_exp", "net_clust"),
  prob = 0.75
)
```

Arguments

`df`	Data frame with all variables.
`B`	Integer. Number of bootstrap repetitions on real data (default `30`).
`P`	Integer. Number of permutations (each with embedded bootstrap) (default `30`).
`seed`	Optional integer random seed.
`ntree`	Integer. Trees for Random Forest in inner scans (default `300`).
`n_cores`	Integer. Parallel cores for `mclapply` (default `parallel::detectCores() - 1`).
`alpha`	Numeric. Significance level for empirical p-values (default `0.05`).
`verbose`	Logical. Print progress (default `TRUE`).
`always_predictors`	Optional character vector of predictors to keep available (pairs where both are in this set are skipped).
`categorical_thr, quantitative_thr`	Thresholds for `evaluate_importance()`.
`importance_method`	One of `"fixed"`, `"neg_exp"`, `"net_clust"`.
`prob`	Numeric. Probability cutoff for neural network when `importance_method="net_clust"`.

Details

For each pair, runs `scan_all_outcomes_complete()` on the 2D subset and keeps only truly binary decisions. Bootstrap counts how often the same direction reappears; permutations shuffle the target outcome to form a null distribution.

Value

A list of four matrices (dimension = `p` × `p`, row/col names = variable names):

- `real`: binary adjacency (1 if direction selected on real data);
- `freq`: bootstrap hit counts for the selected direction;
- `significant`: binary adjacency after permutation test (`p` < `alpha`);
- `pval`: matrix of empirical p-values (rounded).

See Also

`scan_all_outcomes_complete`, `robust_scan_all_outcomes`

Examples

```
## Not run:
mats <- cor_forest_matrix_robust_perm(
  df, B = 50, P = 100, ntree = 300, n_cores = 4, alpha = 0.05
)
image(mats$significant)  # quick look at significant directions

## End(Not run)
```

`dataframe_generation` *Generate a Data Frame from Inline R Formulas*

Description

Evaluates a set of R statements (one per line) inside a temporary environment that contains `n` and the current session scope, then returns all created objects (except `n`) as a data frame.

Usage

```
dataframe_generation(formulas, n = 300)
```

Arguments

`formulas`	Character vector of code lines, or a single multi-line string. Lines starting with `"#"` are ignored.
`n`	Integer. Sample size available inside the temporary environment (bound to `n`).

Value

A data frame with one column per symbol created by the code (excluding `n`).

Examples

```
## Not run:
code <- "
X <- rnorm(n)
Y <- X^3 + rnorm(n, 0, 0.1)
S <- rnorm(n)
W <- rnorm(n)
Z <- X^2 + log(abs(Y)) + W^2 + rnorm(n, 0, 0.1)
"
df <- dataframe_generation(code, n = 500)
str(df)

code1 <- "
X <- rnorm(n)
Y <- X^2 + rnorm(n, 0, 0.1)
W <- rnorm(n)
Z <- log(abs(W) + 1) + rnorm(n, 0, 0.1)
"
df1 <- dataframe_generation(code1, n = 500)

code2 <- "
X <- runif(n, -pi, pi)
Y <- sin(X) + rnorm(n, 0, 0.05)
Z <- cos(Y) + rnorm(n, 0, 0.05)
"
df2 <- dataframe_generation(code2, n = 500)

code3 <- "
L <- rnorm(n)
W <- rnorm(n)
Z <- rnorm(n)
X <- L^4 + rnorm(n, 0 , 0.1)
Y <- W^5 + rnorm(n, 0, 0.1)
"
df3 <- dataframe_generation(code3, n = 500)

code4 <- "
X <- rnorm(n)
Y <- X^2 + rnorm(n, 0, 0.1)
Z <- rnorm(n, 0, 0.1)
S <- exp(abs(Z)) + rnorm(n, 0, 0.1)
"
df4 <- dataframe_generation(code4, n = 500)

code5 <- "
X <- rnorm(n)
Y <- 2*X + rnorm(n, 0, 0.1)
Z <- 1/(X+1)^2 + rnorm(n, 0, 0.1)
S <- atan(Z) + rnorm(n, 0, 0.1)
"
df5 <- dataframe_generation(code5, n = 500)

## End(Not run)
```

`dbscan_1d_augmented` *One-Dimensional DBSCAN with Data Augmentation*

Description

Runs DBSCAN on one-dimensional data, augmented by jittered replicates around each observation to improve clustering stability.

Usage

```
dbscan_1d_augmented(
  x,
  reps_per_point = 60,
  jitter_scale = 0.25,
  eps_factor = 2.5,
  minPts_base = 2,
  scale_data = FALSE,
  seed = 123,
  plot_result = TRUE,
  show_aug_points = FALSE,
  main = "Clustering 1D con DBSCAN (augmented)"
)
```

Arguments

`x`	Numeric vector of data points (no `NA`).
`reps_per_point`	Integer. Number of augmented replicates per point.
`jitter_scale`	Numeric. Scale of jitter relative to MAD (default `0.25`).
`eps_factor`	Numeric. `eps` = `eps_factor` * `sd_jitter` (default `2.5`).
`minPts_base`	Integer. Base `minPts` for DBSCAN (default `2`).
`scale_data`	Logical. Standardize data before clustering.
`seed`	Integer random seed (default `123`).
`plot_result`	Logical. Plot the clustering result (default `TRUE`).
`show_aug_points`	Logical. Show augmented points in the plot.
`main`	Character. Plot title.

Value

List with components:

`clusters`	Integer vector of cluster assignments per original point (0 = noise).
`noise`	Indices of noise points.
`n_clusters`	Number of clusters detected.
`parameters`	List of parameters used.

Examples

```
res <- dbscan_1d_augmented(c(1,2,3,10), plot_result = FALSE)
res$clusters
```

`draw_dag` *Draw a small DAG with smart padding and optional curved edges*

Description

Plots a directed acyclic graph (DAG) from an edge list using **ggraph**, with automatic axis limits padding and optional per-edge curvature. Edges can be color-coded via a `custom_color` column (e.g., `"pairwise"`, `"complex"`, `"both"`), and a subset of edges can be drawn as arcs.

Usage

```
draw_dag(
  edges,
  curved = NULL,
  layout = "auto",
  pad = 0.4,
  arrow_len_pt = 8,
  end_cap_mm = 8,
  linewidth = 0.75,
  node_size = 20,
  node_stroke = 1,
  strength_curved = 0.6
)
```

Arguments

`edges`	A `data.frame` with at least two columns `from` and `to` (character or factor) describing directed edges. If present, a column `custom_color` is used to map edge colors via a manual palette.
`curved`	One of: • `NULL` (default): no curved edges; • a logical vector of length `nrow(edges)` marking which edges are curved; • a character vector of keys `"X->Y"` selecting edges to curve; • a `data.frame` with columns `from` and `to` selecting edges to curve.
`layout`	A layout name passed to `ggraph::create_layout()` / `ggraph()` (e.g., `"auto"`, `"kk"`, `"fr"`, `"sugiyama"`, `"linear"`, ...).
`pad`	Numeric padding added around the computed x/y ranges to prevent clipping (applied symmetrically on both axes).
`arrow_len_pt`	Arrow length (in points) for directed edges.
`end_cap_mm`	End cap radius (in millimeters) for edge arrows.
`linewidth`	Edge line width.
`node_size`	Node point size.

node_stroke Node point stroke width.

strength_curved
Curvature strength for curved edges passed to ggraph::geom_edge_arc2() (non-curved edges use 0).

Details

The function computes an automatic bounding box based on the chosen layout and expands it by pad on both axes to reduce clipping and keep the graph compact but readable. If edges$custom_color exists, it is mapped with a fixed manual scale: "pairwise" → grey, "complex" → black, "both" → greenish.

The curved argument supports multiple convenient notations. When a character vector is supplied, edges are identified by the key paste(from, to, sep = "->").

Value

A **ggplot** object.

See Also

ggraph::ggraph(), ggraph::geom_edge_arc2(), igraph::graph_from_data_frame()

Examples

```
# Minimal example
edges <- data.frame(
  from = c("X","Z","Z","A"),
  to   = c("Y","Y","X","B"),
  custom_color = c("pairwise","both","complex","complex")
)

# Curving a specific edge by key:
p1 <- draw_dag(edges, curved = "Z->X", layout = "kk")
# Curving by logical vector:
p2 <- draw_dag(edges, curved = c(FALSE, TRUE, FALSE, TRUE), layout = "fr")
# Curving via data.frame(from, to):
sel <- data.frame(from = "Z", to = "X")
p3 <- draw_dag(edges, curved = sel, layout = "sugiyama")

# Print one:
# print(p1)
```

---

drill_down_scan *Recursive Drill-Down over Candidate Predictor Sets*

Description

Starting from a current best relation, iteratively explores reduced predictor subsets (removing only those marked as *removable*) to seek higher-entropy combinations for a fixed outcome. At each step it re-fits the causal model and keeps track of entropy gains and importances across layers.

Usage

```
drill_down_scan(
  df,
  verdict,
  importances = list(),
  layer_level,
  removable_predictors,
  ntree = 500,
  verbose = FALSE,
  fixed_outcome,
  always_predictors = NULL,
  quantitative_thr = 40,
  categorical_thr = 35,
  importance_method = c("fixed", "neg_exp", "net_clust"),
  prob = 0.75,
  acc = NULL
)
```

Arguments

df	Data frame with predictors and the fixed outcome column.
verdict	Named numeric vector of current best relations with their entropy (e.g., "X + Y → Z" = 0.42). Passed forward and extended during drilling.
importances	List accumulating variable-importance structures per layer. Typically starts from the first-layer importances returned by causal_entropy_combinations().
layer_level	Integer index of the current drill layer (e.g., 1 for the first drill step).
removable_predictors	Character vector of predictors deemed removable at this step.
ntree	Integer. Number of trees for the underlying Random Forest (default 500).
verbose	Logical. Print step-by-step diagnostics (default FALSE).
fixed_outcome	Character scalar. The outcome variable name kept fixed during drilling.
always_predictors	Optional character vector of predictors that must always be included.
quantitative_thr	Numeric. Threshold for quantitative variables (importance-based pruning).
categorical_thr	Numeric. Threshold for categorical variables (importance-based pruning).
importance_method	One of "fixed", "neg_exp", "net_clust". Strategy used by evaluate_importance() to decide removability.
prob	Numeric. Probability cutoff for the neural network inside evaluate_importance() when importance_method = "net_clust" (default 0.75).
acc	Environment used as an accumulator to propagate stop conditions across recursive calls. Usually left as NULL by the user.

Details

For each removable predictor, the function fits causal_entropy_combinations() on the remaining set and compares entropies, keeping the best. Importances for the winning subset are then

re-evaluated via `evaluate_importance()` to decide the next removable set, and the procedure recurses until stopping criteria are met. If at any step all predictors are deemed removable, the drill-down halts and the root result is discarded to avoid trivial models.

Value

A (possibly nested) list with elements:

- `statistic`: numeric scalar, entropy of the winning combination at this layer;
- `decision`: character, relation label of the winner (e.g., `"X + Y → Z"`);
- `data`: data frame of the winner's predictors + outcome;
- `verdict`: named numeric vector of surviving candidates and entropies;
- `importances`: list accumulating per-layer importances;
- `last_layer`: integer, index of the last processed layer;
- `drill_down`: recursive result for the next layer (or `NULL` if no further drilling).

See Also

`scan_all_outcomes_complete`, `evaluate_importance`, `causal_entropy_combinations`

Examples

```
## Not run:
res <- drill_down_scan(
  df = mydata,
  verdict = c("A + B → Y" = 0.37),
  importances = list(n1th_layer = my_first_layer_imps),
  layer_level = 1,
  removable_predictors = c("A"),
  ntree = 500,
  verbose = TRUE,
  fixed_outcome = "Y",
  always_predictors = NULL,
  quantitative_thr = 40,
  categorical_thr = 35,
  importance_method = "neg_exp",
  prob = 0.75
)

## End(Not run)
```

---

entropy_nd *Multivariate Differential Entropy Estimation with the Kozachenko–Leonenko Method*

Description

This function computes the differential entropy of a multivariate dataset using the k-nearest neighbor estimator (Kozachenko–Leonenko).

Usage

```
entropy_nd(data, k = 10, normalize = c("none", "divide", "sqrt"))
```

Arguments

`data`	A data frame or a matrix of size $n \times d$, where n is the number of observations and d the dimensionality.
`k`	Number of neighbors to consider in the estimator. Must be a positive integer, typically $k \geq 2$. Default: 10.
`normalize`	Normalization mode of the returned entropy. Can be: • `"none"`: raw entropy in bits; • `"divide"`: per-dimension entropy (bits per dimension); • `"sqrt"`: square-root scaling of entropy (non-standard heuristic). Default: `"none"`.

Details

The Kozachenko–Leonenko estimator is based on the distance to the k-th nearest neighbor for each point, and the volume of the unit ball in R^d. The entropy is computed in nats and then converted to bits.

Value

A single numeric value representing the estimated entropy (possibly normalized) in bits.

References

Kozachenko, L. F., & Leonenko, N. N. (1987). Sample estimate of the entropy of a random vector. *Problemy Peredachi Informatsii*, 23(2), 9–16.

Examples

```
set.seed(123)
data <- matrix(rnorm(100 * 3), ncol = 3)
entropy_nd(data, k = 5, normalize = "none")
entropy_nd(data, k = 5, normalize = "divide")
```

`evaluate_importance` *Evaluate Variable Importance and Decide Removability*

Description

Classifies predictors as removable or to be kept based on importance values, using one of:

- `"fixed"` — fixed thresholds for quantitative vs categorical;
- `"net_clust"` — 1D DBSCAN clustering + neural network on high cluster(s);
- `"neg_exp"` — adaptive exponential-decay thresholds depending on p.

Usage

```
evaluate_importance(
  df,
  imp_vec,
  predictors,
  importance_method = c("fixed", "net_clust", "neg_exp"),
  quantitative_thr = 40,
  categorical_thr = 35,
  verbose = FALSE,
  prob = 0.75
)
```

Arguments

df	Data frame containing the predictors.
imp_vec	Named numeric vector of importance values.
predictors	Character vector of predictor names (order for the output).
importance_method	One of "fixed", "net_clust", "neg_exp".
quantitative_thr	Numeric. Threshold for quantitative predictors (default 40).
categorical_thr	Numeric. Threshold for categorical predictors (default 35).
verbose	Logical. Print verbose diagnostics (default FALSE).
prob	Numeric. Probability cutoff for the neural network (default 0.75).

Value

Named logical vector (aligned with predictors), where TRUE means "removable".

Examples

```
imp <- c(x1 = 10, x2 = 50)
df  <- data.frame(x1 = 1:10, x2 = letters[1:10])
evaluate_importance(df, imp, predictors = names(imp), importance_method = "fixed")
```

---

find_maximum *Find the Index of the Maximum Value*

Description

Returns the index of the maximum element in a numeric vector.

Usage

```
find_maximum(v)
```

Arguments

v	Numeric vector.

Value

Integer scalar: the index of the maximum value. If `length(v) == 1`, returns `1`. If `length(v) == 0`, raises an error.

Examples

```
find_maximum(c(3, 5, 2))   # 2
find_maximum(7)            # 1
```

---

generate_multivariate_time_series

Simulate Multivariate Short Panel Time Series from Symbolic Formulas

---

Description

Generates a panel dataset with `n` independent series over `T_points` time points. Each variable is defined by a symbolic formula `lhs ~ rhs` where `rhs` can depend on current values, a per-time deterministic `trend`, Gaussian noise term `error`, and individual lags via `lag(var)`.

Usage

```
generate_multivariate_time_series(
  n = 1000,
  T_points = 4,
  formulas = list(),
  trend_fun_list = list(),
  sd = 0.1,
  seed = NULL
)
```

Arguments

n	Integer. Number of individuals (panel units).
T_points	Integer. Number of time points per individual.
formulas	List of parsed formulas (e.g., created with `as.formula`), each with a single `lhs ~ rhs`.
trend_fun_list	Named list of functions; for each `lhs` name, a function of the time index `i` returning a deterministic trend contribution.
sd	Numeric. Standard deviation of the Gaussian noise `error`.
seed	Optional integer. Random seed for reproducibility.

Details

Within each formula's environment at time `i`:

- `trend` is provided from `trend_fun_list[[lhs]](i)` if present, else 0;
- `error` is `rnorm(1, 0, sd)`;
- `lag(var)` returns the previous-time value of `var` for the same `id`.

Value

A data frame with columns `id`, `t`, and one column per variable defined in `formulas`. Initial state at `t=1` is standard normal.

Examples

```
## Not run:
fmls <- list(
  as.formula(X ~ 0.7 * lag(X) + 0.2 * Y + trend + error),
  as.formula(Y ~ 0.5 * lag(Y) + error)
)
trends <- list(X = function(i) 0.05 * i)
panel <- generate_multivariate_time_series(n = 200, T_points = 10,
                                           formulas = fmls,
                                           trend_fun_list = trends,
                                           sd = 0.1, seed = 123)
head(panel)

## End(Not run)
```

`get_weights_dense` *Retrieve the neural-network weights (lazy-loaded, cached)*

Description

Returns the weights object (previously saved with `base::saveRDS()`) used by functions in this package. The object is loaded on first call and cached in memory for subsequent calls. If a custom path was set via `set_weights_path()`, that file is used. Otherwise, the function looks for `inst/models/weights_nn_model.rds` bundled with the package using `base::system.file()`.

Usage

```
get_weights_dense()
```

Details

This function avoids reading files at top-level (package load time) to keep `devtools::load_all()` and `devtools::document()` robust. The cache is stored in a private environment and persists while the package is loaded.

Value

The R object read from the `.rds` file (typically a list/matrix of layer weights).

See Also

set_weights_path()

Examples

```
## Not run:
# Default (uses the file inside the installed package, if present):
W <- get_weights_dense()

# After setting a custom path:
set_weights_path("/path/to/weights_nn_model.rds")
W2 <- get_weights_dense()

## End(Not run)
```

---

mixed_pca_topvars *Mixed PCA and Top-Contributing Variables*

Description

Runs a mixed PCA (PCAmix) on numeric + categorical data, selects the number of components needed to reach a cumulative explained variance threshold, and extracts the top-top_k contributing variables for each retained component.

Usage

```
mixed_pca_topvars(
  df,
  ncomp = 5,
  top_k = 5,
  var_threshold = 0.9,
  always_predictors = NULL,
  verbose = FALSE
)
```

Arguments

df	Data frame containing numeric and/or categorical variables. Character columns are coerced to factors.
ncomp	Integer. Maximum number of components to compute (default 5).
top_k	Integer. Number of top contributing variables to keep per component (default 5).
var_threshold	Numeric in (0,1]. Target cumulative explained variance (default 0.90).
always_predictors	Optional character vector (kept for API symmetry; not used internally).
verbose	Logical. If TRUE, prints progress (default FALSE).

Value

A list with:

- components: matrix/data frame of individual component scores;
- top_vars: data frame with columns component, variable, contribution_percent;
- contributions: matrix of variable contributions per component;
- fit: the fitted PCAmix object.

Examples

```
## Not run:
library(PCAmixdata)
set.seed(1)
df <- data.frame(
  x1 = rnorm(200),
  x2 = rnorm(200),
  g1 = sample(letters[1:3], 200, TRUE),
  g2 = sample(c("A","B","C","D"), 200, TRUE)
)
res <- mixed_pca_topvars(df, ncomp = 4, top_k = 3, var_threshold = 0.8)
head(res$top_vars)

## End(Not run)
```

---

pca_scan_and_augment *PCA-Guided Causal Scan and Augmentation*

Description

Preprocesses variables, runs mixed PCA to find top-contributing variables, then re-runs the full causal pipeline (complete_function) restricted to those variables (optionally re-adding fixed_variables). Optionally renders a causal graph from the restricted scan.

Usage

```
pca_scan_and_augment(
  df,
  ncomp = 5,
  top_k = 5,
  alpha = 0.05,
  n_boot = 30,
  n_perm = 100,
  ntree = 500,
  seed = 101,
  n_cores = parallel::detectCores() - 1,
  which = c("all", "robust", "binary"),
  always_predictors = NULL,
  verbose = TRUE,
  plot = TRUE,
  categorical_thr = 35,
```

```
  quantitative_thr = 40,
  fixed_variables = NULL,
  importance_method = c("fixed", "neg_exp", "net_clust")
)
```

Arguments

df	Data frame with variables to analyze.
ncomp	Integer. Maximum number of PCA components (passed to mixed_pca_topvars).
top_k	Integer. Top variables per component to retain.
alpha	Numeric. Significance level for permutation tests (default 0.05).
n_boot	Integer. Number of bootstraps (default 30).
n_perm	Integer. Number of permutations (default 100).
ntree	Integer. Trees for Random Forest (default 500).
seed	Optional integer seed.
n_cores	Integer. Parallel cores (default parallel::detectCores()-1).
which	One of "all", "robust", "binary"; passed to complete_function.
always_predictors	Optional character vector of predictors to always include.
verbose	Logical. Verbose output (default TRUE).
plot	Logical. If TRUE, build a causal graph from the restricted scan.
categorical_thr, quantitative_thr	Numeric thresholds for evaluate_importance().
fixed_variables	Optional character vector of columns to exclude from PCA but re-attach for scanning.
importance_method	One of "fixed", "neg_exp", "net_clust" for evaluate_importance().

Value

A list with:

- pca: result from mixed_pca_topvars;
- top_variables: unique variables selected from PCA;
- scan_topvars: list returned by complete_function() on the restricted set;
- plot: recorded plot (if plot=TRUE), else NULL.

See Also

mixed_pca_topvars, complete_function

Examples

```
## Not run:
set.seed(101)
res <- pca_scan_and_augment(
  df,
  ncomp = 5, top_k = 5, which = "all",
  n_boot = 30, n_perm = 100, ntree = 500
)
res$top_variables

## End(Not run)
```

`plot_causal_graph_igraph` *Plot Causal Graph (auto-detect robust relations and/or binary matrices)*

Description

Builds and plots a directed causal graph by auto-detecting inputs passed via `...`: you can provide (i) robust scan results (the list produced by `robust_scan_all_outcomes()`), and/or (ii) a binary direction matrix (or the `$significant` matrix from `cor_forest_matrix_robust_perm()`). If **ggraph/tidygraph** are available, a polished plot is produced; otherwise a base-**igraph** fallback is used. Labels are drawn *inside* circular nodes.

Usage

```
plot_causal_graph_igraph(
  ...,
  base_curvature = 0.22,
  arrow_size = 1,
  arrow_width = 1.25,
  seed = 1
)
```

Arguments

`...`	One or more of: • robust scan results (a named list of per-relation lists with fields `predictors`, `outcome`, `significant`); • a list from `cor_forest_matrix_robust_perm()` or a plain numeric matrix/data frame (interpreted as a binary adjacency where 1 = edge). You may mix multiple robust results and/or multiple matrices; they will be merged.
`base_curvature`	Numeric in $[0, 1]$. Curvature magnitude used to separate opposite-direction edges in the fallback plot (default `0.22`).
`arrow_size, arrow_width`	Numeric. Arrow size and width multipliers for the fallback base-**igraph** plot (defaults `1.0`, `1.25`).
`seed`	Integer random seed for reproducible layouts (default `1`).

Details

Inputs are merged as follows:

- **Robust relations:** for each significant relation, every predictor creates a directed edge `predictor → outcome` tagged as `"complex"`.
- **Binary matrix:** any nonzero entry `[i,j]` adds `i → j` tagged as `"binary"`.
- If both sources yield the same edge, its type becomes `"both"` and is rendered thicker/colored accordingly.

Layout choice: DAGs attempt `layout_with_sugiyama`; otherwise `kk`/`fr`. When **ggraph** is available, node diameters are sized to fit labels.

Value

Invisibly returns:

- a `tidygraph::tbl_graph` object when using the **ggraph** pipeline, or
- an `igraph` object when using the fallback base plot,
- `NULL` if no nodes/edges could be inferred.

The function draws the plot as a side effect.

Examples

```
## Not run:
# From robust + binary:
TG <- plot_causal_graph_igraph(robust_results, binary_results)

# Only binary matrix:
M <- matrix(0, 3, 3, dimnames = list(letters[1:3], letters[1:3]))
M["a","b"] <- 1; M["b","c"] <- 1
plot_causal_graph_igraph(M)

## End(Not run)
```

---

predict_manual *Manual prediction with a dense neural network*

Description

Performs a forward pass of a feed-forward neural network given a list of weights and biases, applying the specified activation functions at each layer.

Usage

```
predict_manual(
  X,
  weights_dense,
  activations = "relu",
  last_activation = NULL,
  verbose = FALSE
)
```

Arguments

X	Numeric input matrix, with rows = observations and columns = features.
weights_dense	A list of length 2*L containing, for each layer, the weight matrix (W) and the bias vector (b), in the order [W1, b1, W2, b2, ...].
activations	Either: • a single activation name/function (recycled for all layers), or • a vector/list of length L with one activation per layer. Supported names: "relu", "sigmoid", "tanh", or custom functions.
last_activation	Optional. If provided, overrides the activation of the last layer.
verbose	Logical; if TRUE, prints layer shapes and debugging info.

Value

A numeric matrix (or vector if 1D) with the output of the network.

---

relu *ReLU (Rectified Linear Unit) activation*

Description

Sets all negative values to zero.

Usage

```
relu(x)
```

Arguments

x	Numeric vector or matrix input.

Value

A matrix with negative values replaced by zero.

---

remove_outliers *Remove Outliers from a Data Frame*

Description

Removes all rows containing outliers in any numeric column, using the interquartile range (IQR) rule.

Usage

```
remove_outliers(df)
```

Arguments

df	A data frame with numeric and/or non-numeric columns.

Value

A data frame with the same columns as df, with rows removed if any numeric column contains an outlier in that row.

Examples

```
df <- data.frame(a = c(1,2,3,100), b = c(5,6,7,8))
remove_outliers(df)
```

---

robust_scan_all_outcomes

Robust Validation of Causal Scan Results (Bootstrap + Permutation)

Description

Validates each relation found by a prior scan (e.g. scan_all_outcomes_complete) using (1) bootstrap repetitions on real data to count how often the relation (or a subset with same outcome) reappears, and (2) an empirical permutation test with embedded bootstrap to compute p-values.

Usage

```
robust_scan_all_outcomes(
  df,
  scan_results,
  seed = NULL,
  ntree = 500,
  n_boot = 300,
  n_perm = 30,
  alpha = 0.05,
  n_cores = parallel::detectCores() - 1,
  verbose = TRUE,
  always_predictors = NULL,
  categorical_thr = 35,
  quantitative_thr = 40,
  importance_method = c("fixed", "neg_exp", "net_clust"),
  prob = 0.75
)
```

Arguments

df	Data frame with all variables.
scan_results	List of results returned by scan_all_outcomes_complete().
seed	Optional integer random seed.
ntree	Integer. Number of trees for Random Forest during re-scans (default 500).
n_boot	Integer. Bootstrap repetitions on real data (default 300).

n_perm	Integer. Number of permutations (each with embedded bootstrap) (default 30).
alpha	Numeric. Significance level for the empirical p-value (default 0.05).
n_cores	Integer. Number of parallel cores for mclapply (default parallel::detectCores() - 1).
verbose	Logical. Print progress (default TRUE).
always_predictors	Optional character vector of predictors to keep available.
categorical_thr	Numeric. Threshold for categorical variables in evaluate_importance() (default 35).
quantitative_thr	Numeric. Threshold for quantitative variables in evaluate_importance() (default 40).
importance_method	One of "fixed", "neg_exp", "net_clust" for evaluate_importance().
prob	Numeric. Probability cutoff for neural network inside evaluate_importance() when importance_method="net_clust".

Details

Bootstrap step: resamples rows with replacement (and shuffles columns order), reruns scan_all_outcomes_complete() on the subset of variables for the target relation, and counts partial matches (same outcome and predictors included). Permutation step: shuffles all variables within the subset and repeats the bootstrap to build the null distribution of the count; computes p-value.

Value

A named list (one entry per tested relation) with elements:

- predictors, outcome;
- bootstrap_freq: total bootstrap hits (partial-match counting);
- max_freq: theoretical maximum hits (n_boot * length(predictors));
- freq_perm: vector of permutation bootstrap counts (or NA if skipped);
- p_empirical: empirical p-value;
- significant: logical flag (p_empirical < alpha).

See Also

scan_all_outcomes_complete, evaluate_importance

Examples

```
## Not run:
set.seed(42)
val <- robust_scan_all_outcomes(
  df, scan_results,
  ntree = 500, n_boot = 200, n_perm = 50, alpha = 0.05, n_cores = 4
)

## End(Not run)
```

scan_all_outcomes_complete *Full Multi-Outcome Causal Scan with Drill-Down*

Description

Fits causal_entropy_combinations() on all outcomes in df, ranks candidate relations by entropy, applies a drill-down search on each top candidate (recursively removing low-importance predictors), and finally removes bidirectional duplicates by keeping the higher-entropy direction.

Usage

```
scan_all_outcomes_complete(
  df,
  ntree = 500,
  verbose = FALSE,
  always_predictors = NULL,
  seed = NULL,
  categorical_thr = 35,
  quantitative_thr = 40,
  importance_method = c("fixed", "neg_exp", "net_clust"),
  prob = 0.75
)
```

Arguments

df	Data frame containing all variables to scan (predictors and outcomes).
ntree	Integer. Number of trees for Random Forest (default 500).
verbose	Logical. Print progress and diagnostics (default FALSE).
always_predictors	Optional character vector of predictors that must always be included.
seed	Optional integer random seed for reproducibility.
categorical_thr	Numeric. Threshold for categorical predictors in evaluate_importance() (default 35).
quantitative_thr	Numeric. Threshold for quantitative predictors in evaluate_importance() (default 40).
importance_method	One of "fixed", "neg_exp", "net_clust". Strategy to decide removability in evaluate_importance().
prob	Numeric. Probability cutoff for the neural network used when importance_method = "net_clust" (default 0.75).

Details

Pipeline.

1. Fit causal_entropy_combinations(df, ...) to obtain entropies and first-layer importances;

2. Rank candidates by entropy and, for each, compute removable predictors via `evaluate_importance()`;
3. Run `drill_down_scan()` to explore reduced subsets until convergence;
4. Remove bidirectional duplicates by keeping the higher-entropy direction.

Pruning. After drill-down, candidates can be discarded if any retained predictor shows too-low importance (see inline checks). Note: current implementation compares against an internal threshold in code; keep documentation consistent with that setting.

Reproducibility. If `seed` is provided, `set.seed(seed)` is used before fitting. Predictors listed in `always_predictors` are kept throughout the scan and drill-down.

Value

A list of results (one per retained relation) where each element contains:

- `outer_layer_statistics`: named numeric vector of first-layer candidates and entropies;
- `outer_layer_decision`: character, root relation label examined;
- `drill_down_statistic`: named numeric scalar, final winning relation and its entropy;
- `drill_down_decision`: character, final winning relation label (after drilling);
- `verdict`: named numeric vector of surviving candidates at the final depth;
- `importances`: list of per-layer importances accumulated during drilling.

Returns `invisible(NULL)` if no candidates are found.

See Also

`drill_down_scan`, `evaluate_importance`, `causal_entropy_combinations`

Examples

```
## Not run:
results <- scan_all_outcomes_complete(
  df = mydata,
  ntree = 500,
  verbose = TRUE,
  always_predictors = NULL,
  seed = 123,
  categorical_thr = 35,
  quantitative_thr = 40,
  importance_method = "neg_exp",
  prob = 0.75
)

## End(Not run)
```

set_weights_path *Configure a custom path for the neural-network weights (.rds)*

Description

Sets (and validates) a custom file path for the pre-saved weights object used by the package. This function does **not** read the file immediately; it only stores the path and clears the in-memory cache so the next call to `get_weights_dense()` will reload from disk.

Package-default behavior looks for a file named `inst/models/weights_nn_model.rds` at install time and resolves it via `base::system.file()`. Use this function if you want to override that path.

Usage

```
set_weights_path(path)
```

Arguments

`path`	`character(1)`. Absolute or relative path to a `.rds` file that exists and is readable.

Value

Invisibly returns the path (invisible character scalar).

See Also

`get_weights_dense()`

Examples

```
## Not run:
# Point to a custom weights file:
set_weights_path("/path/to/weights_nn_model.rds")

# Then use it (will be loaded lazily on first call):
W <- get_weights_dense()

## End(Not run)
```

sigmoid *Sigmoid activation*

Description

Computes the sigmoid transformation.

Usage

```
sigmoid(x)
```

Arguments

x	Numeric vector or matrix input.

Value

Transformed values in (0, 1).

---

thr_exp *Negative-Exponential Threshold Function*

Description

Computes a decaying threshold as a function of the number of predictors (p), optionally lower-bounded by L.

Usage

```
thr_exp(p, thr, k, L = NULL)
```

Arguments

p	Integer. Number of predictors/dimension.
thr	Numeric. Initial threshold value (at $p = 1$).
k	Numeric. Decay rate.
L	Optional numeric. Lower bound (default NULL).

Value

Numeric vector of thresholds.

Examples

```
thr_exp(10, thr = 40, k = 0.05)
thr_exp(1:5, thr = 40, k = 0.2, L = 10)
```

---

tune_rf *Random Forest Tuning over mtry with %IncMSE Importances*

Description

Fits multiple Random Forests over a grid of mtry values and selects the model with the lowest in-sample MSE. Returns the best model and its variable importances (\

Usage

```
tune_rf(x, y, mtry_grid = 1:floor(sqrt(ncol(x))), ntree = 500)
```

Arguments

x	A data frame or matrix of predictors (rows = samples, cols = features).
y	A numeric response vector (same length as nrow(x)).
mtry_grid	Integer vector of mtry values to try. If NULL, a small grid is generated automatically.
ntree	Integer. Number of trees for each Random Forest (default 500).

Details

The score minimized is the in-sample Mean Squared Error on y vs predict(model, x). Importances are extracted from the best model via randomForest::importance() and the "%IncMSE" column is returned.

Value

A list with:

- model: the best randomForest object found;
- feature_importance: a one-column matrix with rownames = predictors and column "%IncMSE".

Examples

```
## Not run:
set.seed(1)
x <- as.data.frame(matrix(rnorm(200*5), 200, 5))
y <- x[[1]] * 2 + rnorm(200)
out <- tune_rf(x, y, mtry_grid = 1:5, ntree = 300)
str(out$feature_importance)

## End(Not run)
```

Index