Package 'niarules'

September 15, 2025

Type Package

Title Numerical Association Rule Mining using Population-Based Nature-Inspired Algorithms

Version 0.3.1

Classification/ACM G.4, H.2.8

Description Framework is devoted to mining numerical association rules through the utilization of nature-inspired algorithms for optimization. Drawing inspiration from the 'NiaARM' 'Python' and the 'NiaARM' 'Julia' packages, this repository introduces the capability to perform numerical association rule mining in the R programming language.
Fister Jr., Iglesias, Galvez, Del Ser, Osaba and Fister (2018) <doi:10.1007/978-3-030-03493-1_9>.

URL https://github.com/firefly-cpp/niarules

BugReports https://github.com/firefly-cpp/niarules/issues

Depends R (>= 4.0.0)

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Imports stats, utils, Rcpp, dplyr, rlang, rgl

Suggests testthat, withr

LinkingTo Rcpp

NeedsCompilation yes

Author Iztok Jr. Fister [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-6418-1272>), Gerlinde Emsenhuber [aut] (ORCID: <https://orcid.org/0000-0001-7776-586X>), Jan Hendrik Plümer [aut] (ORCID: <https://orcid.org/0009-0000-3722-1702>)

Maintainer Iztok Jr. Fister <iztok@iztok.space>

Repository CRAN

Date/Publication 2025-09-15 09:10:12 UTC

Contents

`add_attribute` *Add an attribute to the "rule" list.*

Description

This function adds an attribute to the existing list.

Usage

```
add_attribute(rules, name, type, border1, border2, value)
```

Arguments

rules	The current rules list.
name	The name of the feature in the rule.
type	The type of the feature in the rule.
border1	The first border value in the rule.
border2	The second border value in the rule.
value	The value associated with the rule.

Value

The updated rules list.

Examples

```
rules <- list()
new_rules <- add_attribute(rules, "feature1", "numerical", 0.2, 0.8, "EMPTY")
```

---

build_coral_plots *Build coral plot layout (nodes + edges) from a parsed rules object*

Description

Produces the node and edge layout consumed by ‘render_coral_rgl()‘ or ‘render_coral_rgl_experimental()‘. Given a parsed association-rules object (from ‘parse_rules()‘), this function groups rules by RHS itemset (one coral per unique RHS), arranges those corals on a square grid, and emits geometry and metadata for drawing.

Input expectations (‘parsed‘) - ‘parsed$items‘: ‘data.frame‘ with at least ‘item_id‘ (integer, **0-based**), ‘label‘ (character). - ‘parsed$rules‘: ‘data.frame‘ with at least ‘support‘, ‘confidence‘, ‘lift‘ (numeric) and ‘lhs_item_ids‘, ‘rhs_item_ids‘ (list-columns of **0-based** integer vectors).

Usage

```
build_coral_plots(
  parsed,
  lhs_sort_metric = c("confidence", "support", "lift"),
  bin_breaks = NULL,
  bin_digits = 3
)
```

Arguments

parsed	A list as returned by ‘parse_rules()‘, containing components ‘items‘ and ‘rules‘ with the schema above.
lhs_sort_metric	character; how to order items **within each LHS path** when building the layout. One of ‘"confidence"‘, ‘"support"‘, ‘"lift"‘. Typically interpreted as **descending** by the chosen metric.
bin_breaks	Optional named list of numeric break vectors used to bin numeric features (e.g., ‘list(Age = c(0, 18, 30, 50, Inf))‘). If ‘NULL‘, bins are inferred later (and may be provided to the renderer via ‘bin_legend‘).
bin_digits	Integer number of decimal places used when formatting numeric interval labels (e.g., ‘[0.405,0.485)‘). Default is ‘3‘.

Details

Grid sizing. The number of corals (‘n_plots‘) is computed as the number of distinct, non-empty RHS itemsets across rules. An RHS itemset's display label is recomposed by joining the ‘items$label‘ values for its ‘rhs_item_ids‘ (comma-separated). The grid is arranged as a near-square: ‘grid_size = ceiling(sqrt(n_plots))‘, with a minimum of 1.

The heavy lifting (node positions, radii, edge routing) is delegated to the C++ backend ‘build_layout_cpp()‘, which receives the ‘parsed‘ object, the computed ‘grid_size‘, and the chosen ‘lhs_sort_metric‘.

Output schema (for ‘render_coral_rgl()‘). - ‘nodes‘ includes (at least): ‘x‘, ‘z‘, ‘x_offset‘, ‘z_offset‘, ‘radius‘, ‘path‘ (character key), ‘node_id‘, ‘is_root‘, ‘coral_id‘, ‘interval_brackets‘, ‘bin_index‘ and optionally ‘item‘, ‘feature‘, ‘step‘, ‘interval_label‘, ‘interval_label_short‘. - ‘edges‘ includes (at least): ‘x‘, ‘y‘, ‘z‘, ‘x_end‘, ‘y_end‘, ‘z_end‘, ‘parent_path‘, ‘child_path‘, and the rule metrics ‘support‘, ‘confidence‘, ‘lift‘. (Initial ‘y‘/‘y_end‘ are typically on the base plane; vertical styling can be added later by ‘render_coral_rgl()‘ via ‘y_scale‘/jitter.)

Indexing note. Item identifiers remain **0-based** as produced by ‘parse_rules()‘ for cross-language stability.

Value

A list with components: - ‘nodes‘: ‘data.frame‘ of node geometry and labels, - ‘edges‘: ‘data.frame‘ of edge geometry and attached metrics, - ‘grid_size‘: integer grid side length used to arrange corals. - ‘bin_legend‘: data.frame (or NULL) mapping feature -> bin index -> interval.

---

build_rule *Build rules based on a candidate solution.*

Description

This function takes a candidate solution vector and a features list and builds rule.

Usage

```
build_rule(solution, features)
```

Arguments

solution	The solution vector.
features	The features list.

Value

A rule.

`calculate_border` *Calculate the border value based on feature information and a given value.*

Description

This function calculates the border value for a feature based on the feature information and a given value.

Usage

```
calculate_border(feature_info, value)
```

Arguments

feature_info	Information about the feature.
value	The value to calculate the border for.

Value

The calculated border value.

Examples

```
feature_info <- list(type = "numerical", lower_bound = 0, upper_bound = 1)
border_value <- calculate_border(feature_info, 0.5)
```

calculate_fitness *Calculate the fitness of an association rule.*

Description

This function calculates the fitness of an association rule using support and confidence.

Usage

```
calculate_fitness(supp, conf)
```

Arguments

supp	The support of the association rule.
conf	The confidence of the association rule.

Value

The fitness of the association rule.

calculate_selected_category *Calculate the selected category based on a value and the number of categories.*

Description

This function calculates the selected category based on a given value and the total number of categories.

Usage

```
calculate_selected_category(value, num_categories)
```

Arguments

value	The value to calculate the category for.
num_categories	The total number of categories.

Value

The calculated selected category.

Examples

```
selected_category <- calculate_selected_category(0.3, 5)
```

check_attribute *Check if the attribute conditions are satisfied for an instance.*

Description

This function checks if the attribute conditions specified in the association rule are satisfied for a given instance row.

Usage

```
check_attribute(attribute, instance_row)
```

Arguments

attribute	An attribute with type and name information.
instance_row	A row representing an instance in the dataset.

Value

TRUE if conditions are satisfied, FALSE otherwise.

coral_get_theme *Get a theme object (experimental)*

Description

‘r lifecycle::badge('experimental')‘

Usage

```
coral_get_theme(
  name = c("default", "studio", "flat", "dark", "none"),
  overrides = NULL
)
```

Arguments

name	one of coral_list_themes()
overrides	named list to tweak values

Value

an object of class 'coral_theme'

coral_list_themes *Available themes (experimental)*

Description

'r lifecycle::badge('experimental')'

Usage

```
coral_list_themes()
```

Value

character vector of theme names

cut_point *Calculate the cut point for an association rule.*

Description

This function calculates the cut point, denoting which part of the vector belongs to the antecedent and which to the consequent of the mined association rule.

Usage

```
cut_point(sol, num_attr)
```

Arguments

sol	The cut value from the solution vector.
num_attr	The number of attributes in the association rule.

Value

The cut point value.

`differential_evolution` *Implementation of Differential Evolution metaheuristic algorithm.*

Description

This function uses Differential Evolution, a stochastic population-based optimization algorithm, to find the optimal numerical association rule.

Usage

```
differential_evolution(
  d = 10,
  np = 10,
  f = 0.5,
  cr = 0.9,
  nfes = 1000,
  features,
  data,
  is_time_series = FALSE
)
```

Arguments

`d`	Dimension of the problem (default: 10).
`np`	Population size (default: 10).
`f`	The differential weight, controlling the amplification of the difference vector (default: 0.5).
`cr`	The crossover probability, determining the probability of a component being replaced (default: 0.9).
`nfes`	The maximum number of function evaluations (default: 1000).
`features`	A list containing information about features, including type and bounds.
`data`	A data frame representing instances in the dataset.
`is_time_series`	A boolean indicating whether the dataset is time series.

Value

A list containing the best solution, its fitness value, and the number of function evaluations and list of identified association rules.

References

Storn, R., & Price, K. (1997). "Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces." Journal of Global Optimization, 11(4), 341–359. doi:10.1023/A:1008202821328

evaluate *Evaluate a candidate solution, with optional time series filtering.*

Description

This function evaluates the fitness of an association rule using support and confidence. If time series data is used, it restricts evaluation to the specified time range.

Usage

```
evaluate(solution, features, instances, is_time_series = FALSE)
```

Arguments

solution	A vector representing a candidate solution.
features	A list containing information about features.
instances	A data frame representing dataset instances.
is_time_series	A boolean flag indicating if time series filtering is required.

Value

A list containing fitness and identified rules.

References

Fister, I., Iglesias, A., Galvez, A., Del Ser, J., Osaba, E., & Fister, I. (2018). "Differential evolution for association rule mining using categorical and numerical attributes." In Intelligent Data Engineering and Automated Learning–IDEAL 2018: 19th International Conference, Madrid, Spain, November 21–23, 2018, Proceedings, Part I (pp. 79-88). Springer International Publishing. doi:10.1007/9783030034962_9

Fister Jr, I., Podgorelec, V., & Fister, I. (2021). "Improved nature-inspired algorithms for numeric association rule mining." In Intelligent Computing and Optimization: Proceedings of the 3rd International Conference on Intelligent Computing and Optimization 2020 (ICO 2020) (pp. 187-195). Springer International Publishing. doi:10.1007/9783030681548_19

extract_feature_info *Extract feature information from a dataset, excluding timestamps.*

Description

This function analyzes the given dataset and extracts information about each feature.

Usage

```
extract_feature_info(data, timestamp_col = "timestamp")
```

Arguments

data	The dataset to analyze.
timestamp_col	Optional. The name of the timestamp column to exclude from features.

Value

A list containing information about each feature, including type and bounds/categories.

---

feature_position *Get the position of a feature.*

Description

This function returns the position of a feature in the vector, considering the type of the feature.

Usage

```
feature_position(features, feature)
```

Arguments

features	The features list.
feature	The name of the feature to find.

Value

The position of the feature.

Examples

```
features <- list(
  feature1 = list(type = "numerical"),
  feature2 = list(type = "categorical"),
  feature3 = list(type = "numerical")
)
position <- feature_position(features, "feature2")
```

fix_borders *Fix Borders of a Numeric Vector*

Description

This function ensures that all values greater than 1.0 are set to 1.0, and all values less than 0.0 are set to 0.0.

Usage

```
fix_borders(vector)
```

Arguments

vector	A numeric vector to be processed.

Value

A numeric vector with borders fixed.

format_rule_parts *Format Rule Parts*

Description

This function formats the parts of an association rule into a string.

Usage

```
format_rule_parts(parts)
```

Arguments

parts	A list containing parts of an association rule.

Value

A formatted string representing the rule parts.

map_to_ts *Map solution boundaries to time series instances.*

Description

This function maps the lower and upper bounds of the solution vector to a subset of the dataset.

Usage

```
map_to_ts(lower, upper, instances)
```

Arguments

lower	The lower bound in [0, 1].
upper	The upper bound in [0, 1].
instances	The full dataset.

Value

A list with ‘low‘, ‘up‘, and ‘filtered_instances‘.

metric_domains *Compute metric domains (experimental)*

Description

‘r lifecycle::badge('experimental')‘

Returns numeric ranges [min, max] for the metrics ‘support‘, ‘confidence‘, and ‘lift‘ from various input types.

Usage

```
metric_domains(x, ...)

## S3 method for class 'parsed'
metric_domains(x, ...)

## S3 method for class 'data.frame'
metric_domains(x, ...)

## Default S3 method:
metric_domains(x, ...)
```

Arguments

x	An object containing rule metrics. Methods are provided for objects of class ‘parsed‘, and for a ‘data.frame‘. (Optionally also for class ‘niarules_edges‘ if you tag it.)
...	Passed to methods (currently unused).

Value

A named list with numeric vectors of length 2: ‘list(support = c(min, max), confidence = c(min, max), lift = c(min, max))‘.

---

parse_rules *Parse association rules into a reusable, layout-agnostic structure*

Description

Converts association rules into a normalized representation for downstream layout/rendering. Accepts either: - a ‘data.frame‘ with **required** columns ‘Antecedent‘, ‘Consequence‘, ‘Support‘, ‘Confidence‘, ‘Fitness‘, or - a native ‘niarules‘ rules object (which is exported to CSV internally via ‘niarules::write_association_rules_to_csv()‘ and then parsed).

The output separates **items** from **rules** and uses stable **0-based** item identifiers suitable for cross-language use.

Usage

```
parse_rules(arules = NULL)
```

Arguments

arules	A ‘data.frame‘ with columns ‘Antecedent‘, ‘Consequence‘, ‘Support‘, ‘Confidence‘, ‘Fitness‘, **or** a ‘niarules‘ rules object.

Details

Input requirements - ‘Antecedent‘, ‘Consequence‘: character encodings of itemsets per rule. - ‘Support‘, ‘Confidence‘, ‘Fitness‘: numeric metrics; ‘Fitness‘ is interpreted as the **lift-like** metric and is exposed as ‘lift‘ in the returned ‘rules‘.

When ‘arules‘ is not a ‘data.frame‘, the function requires the **niarules** package at runtime to serialize the rules to CSV. Missing required columns trigger an error.

Output schema - ‘items‘ (‘data.frame‘): ‘item_id‘ (integer, **0-based**), ‘label‘, ‘feature‘, ‘kind‘, ‘category_value‘, ‘lo‘, ‘hi‘, ‘incl_low‘, ‘incl_high‘, ‘op‘, ‘label_long‘, ‘label_short‘. - ‘rules‘ (‘data.frame‘): ‘rule_id‘, ‘support‘, ‘confidence‘, ‘lift‘, ‘lhs_item_ids‘ (list of integer vectors; **0-based ids**), ‘rhs_item_ids‘ (list of integer vectors; **0-based ids**), ‘antecedent_length‘, ‘consequent_length‘.

Indexing note ‘item_id‘ values are **0-based** for stability across languages. In R, convert to 1-based with ‘items$item_id + 1L‘ if needed.

Value

A list with components: - ‘items‘: ‘data.frame‘ describing unique items, - ‘rules‘: ‘data.frame‘ describing association rules.

`particle_swarm_optimization` *Implementation of Particle Swarm Optimization (PSO) metaheuristic algorithm.*

Description

This function uses PSO, a stochastic population-based optimization algorithm, to find the optimal numerical association rule.

Usage

```
particle_swarm_optimization(
  d = 10,
  np = 10,
  w = 0.7,
  c1 = 1.5,
  c2 = 1.5,
  nfes = 1000,
  features,
  data,
  is_time_series = FALSE
)
```

Arguments

`d`	Dimension of the problem (default: 10).
`np`	Population size (default: 10).
`w`	Inertia weight (default: 0.7).
`c1`	Cognitive coefficient (default: 1.5).
`c2`	Social coefficient (default: 1.5).
`nfes`	The maximum number of function evaluations (default: 1000).
`features`	A list containing information about features, including type and bounds.
`data`	A data frame representing instances in the dataset.
`is_time_series`	A boolean indicating whether the dataset is time series.

Value

A list containing the best solution, its fitness value, and the number of function evaluations and list of identified association rules.

References

Kennedy, J., & Eberhart, R. (1995). "Particle swarm optimization." Proceedings of ICNN'95 - International Conference on Neural Networks, 4, 1942–1948. IEEE. doi:10.1109/ICNN.1995.488968

`print_association_rules`

Print Numerical Association Rules

Description

This function prints association rules including antecedent, consequence, support, confidence, and fitness. For time series datasets, it also includes the start and end timestamps instead of indices.

Usage

```
print_association_rules(rules, is_time_series = FALSE, timestamps = NULL)
```

Arguments

`rules`	A list containing association rules.
`is_time_series`	A boolean flag indicating if time series information should be included.
`timestamps`	A vector of timestamps corresponding to the time series data.

Value

Prints the association rules.

`print_feature_info` *Print feature information extracted from a dataset.*

Description

This function prints the information extracted about each feature.

Usage

```
print_feature_info(feature_info)
```

Arguments

`feature_info`	The list containing information about each feature.

Value

A message is printed to the console for each feature, providing information about the feature's type, and additional details such as lower and upper bounds for numerical features, or categories for categorical features. No explicit return value is generated.

problem_dimension *Calculate the dimension of the problem, excluding timestamps.*

Description

Calculate the dimension of the problem, excluding timestamps.

Usage

```
problem_dimension(feature_info, is_time_series = FALSE)
```

Arguments

feature_info	A list containing information about each feature.
is_time_series	Boolean indicating if time series data is present.

Value

The calculated dimension based on the feature types.

read_dataset *Read a CSV Dataset*

Description

Reads a dataset from a CSV file and optionally parses a timestamp column.

Usage

```
read_dataset(
  dataset_path,
  timestamp_col = "timestamp",
  timestamp_formats = c("%d/%m/%Y %H:%M:%S", "%H:%M:%S %d/%m/%Y")
)
```

Arguments

dataset_path	A string specifying the path to the CSV file.
timestamp_col	A string specifying the timestamp column name (default: ‘"timestamp"‘).
timestamp_formats	A vector of date-time formats to try for parsing timestamps.

Value

A data frame containing the dataset.

`render_coral_rgl` *Apply styling to coral plots and render them with rgl*

Description

Renders a 3D "coral" plot produced by ‘build_coral_layout()‘, with edge width/color/alpha mapped from association rule metrics and node colors derived from item/type groupings. The function draws a floor grid, edges as 3D segments, nodes as spheres, and optional labels/legend.

Required columns - ‘edges‘: ‘x‘, ‘y‘, ‘z‘, ‘x_end‘, ‘y_end‘, ‘z_end‘, ‘parent_path‘, ‘child_path‘, and metric columns ‘support‘, ‘confidence‘, ‘lift‘. - ‘nodes‘: ‘x‘, ‘z‘, ‘x_offset‘, ‘z_offset‘, ‘radius‘, ‘path‘.

Optional columns - ‘nodes$item‘, ‘nodes$feature‘ (for labels/legend & color-by), ‘nodes$step‘ (roots identified as ‘step == 0‘), ‘nodes$interval_label‘, ‘nodes$interval_label_short‘ (label text when requested).

Usage

```
render_coral_rgl(
  nodes,
  edges,
  grid_size,
  grid_color = "grey80",
  legend = FALSE,
  label_mode = c("none", "interval", "item", "interval_short"),
  label_cex = 0.7,
  label_offset = 1.5,
  max_labels = 100,
  edge_width_metric = c("confidence", "lift", "support"),
  edge_color_metric = c("confidence", "lift", "support"),
  edge_alpha_metric = NULL,
  edge_width_range = c(1, 5),
  edge_width_transform = c("linear", "sqrt", "log"),
 edge_gradient = c("#2166AC", "#67A9CF", "#D1E5F0", "#FDDBC7", "#EF8A62", "#B2182B"),
  edge_color_transform = c("linear", "sqrt", "log"),
  edge_alpha = 0.5,
  edge_alpha_range = c(0.25, 0.5),
  edge_alpha_transform = c("linear", "sqrt", "log"),
 node_color_by = c("type", "item", "none", "edge_incoming", "edge_outgoing_mean"),
  node_gradient = "match",
  node_gradient_map = c("even", "hash", "frequency"),
  y_scale = 0,
  jitter_sd = 0,
  jitter_mode = c("deterministic", "random"),
  jitter_seed = NULL,
  return_data = FALSE
)
```

Arguments

nodes	data.frame; typically ‘build_coral_layout()$nodes‘. Must contain ‘x‘, ‘z‘, ‘x_offset‘, ‘z_offset‘, ‘radius‘, ‘path‘. Optional: ‘item‘, ‘feature‘, ‘step‘, ‘interval_label‘, ‘interval_label_short‘.
edges	data.frame; typically ‘build_coral_layout()$edges‘. Must contain ‘x‘, ‘y‘, ‘z‘, ‘x_end‘, ‘y_end‘, ‘z_end‘, ‘parent_path‘, ‘child_path‘, and metric columns ‘support‘, ‘confidence‘, ‘lift‘.
grid_size	integer; the layout grid size (usually ‘build_coral_layout()$grid_size‘).
grid_color	background grid color. Any R color spec. Default ‘"grey80"‘.
legend	logical; draw a node legend keyed by base feature (‘nodes$feature‘). Requires that ‘nodes$feature‘ and node colors are available. Default ‘FALSE‘.
label_mode	one of ‘"none"‘, ‘"interval"‘, ‘"item"‘, ‘"interval_short"‘. Controls label text: interval labels, item labels, or no labels.
label_cex	numeric; label size passed to ‘rgl::text3d()‘. Default ‘0.7‘.
label_offset	numeric; vertical offset (in **node radii**) applied to labels (positive values move labels downward from sphere tops). Default ‘1.5‘.
max_labels	integer; maximum number of **non-root** labels to keep (largest radii first). Root nodes are always kept. Default ‘100‘.
edge_width_metric	character; which metric to map to edge **width**. One of ‘"confidence"‘, ‘"lift"‘, ‘"support"‘. Default ‘"confidence"‘.
edge_color_metric	character; which metric to map to edge **color**. One of ‘"confidence"‘, ‘"lift"‘, ‘"support"‘. Default ‘"confidence"‘.
edge_alpha_metric	character or ‘NULL‘; which metric to map to edge **alpha** (transparency). One of ‘"support"‘, ‘"lift"‘, ‘"confidence"‘, or ‘NULL‘ to use the constant ‘edge_alpha‘. Default ‘NULL‘.
edge_width_range	numeric length-2; min/max line width for edges after scaling. Default ‘c(1, 5)‘.
edge_width_transform	character; transformation for width scaling from normalized metric in ‘[0,1]‘. One of ‘"linear"‘, ‘"sqrt"‘, ‘"log"‘. Default ‘"linear"‘.
edge_gradient	character vector (>= 2); color ramp for edges, passed to ‘grDevices::colorRamp()‘. Default ‘c("#2166AC","#67A9CF","#D1E5F0","#FDDBC7","#EF8A62","#B2182B")‘.
edge_color_transform	character; transformation for color scaling from normalized metric in ‘[0,1]‘. One of ‘"linear"‘, ‘"sqrt"‘, ‘"log"‘. Default ‘"linear"‘.
edge_alpha	numeric in ‘[0,1]‘; constant alpha used **only when** ‘edge_alpha_metric‘ is ‘NULL‘. Default ‘0.6‘.
edge_alpha_range	numeric length-2 in ‘[0,1]‘; min/max alpha used **only when** ‘edge_alpha_metric‘ is not ‘NULL‘. Default ‘c(0.25, 0.5)‘.

`edge_alpha_transform`
: character; transformation for alpha scaling from normalized metric in ‘[0,1]‘. One of ‘"linear"‘, ‘"sqrt"‘, ‘"log"‘. Default ‘"linear"‘.

`node_color_by`
: one of ‘"type"‘, ‘"item"‘, ‘"none"‘, ‘"edge_incoming"‘, ‘"edge_outgoing_mean"‘. Controls node coloring: - ‘"type"‘ colors by ‘nodes$feature‘ (recommended). - ‘"item"‘ colors by ‘nodes$item‘. - ‘"none"‘ leaves default colors. - ‘"edge_incoming"‘ / ‘"edge_outgoing_mean"‘ are reserved for future use. **Note:** current implementation applies custom colors only for ‘"type"‘ and ‘"item"‘. Default ‘"type"‘.

`node_gradient`
: either the string ‘"match"‘ to reuse ‘edge_gradient‘ for nodes, or a character vector (>= 2) of colors to build the node palette. Default ‘"match"‘.

`node_gradient_map`
: one of ‘"even"‘, ‘"hash"‘, ‘"frequency"‘; how unique labels are placed along the gradient: - ‘"even"‘: evenly spaced by sorted unique label order, - ‘"hash"‘: stable per-label positions via a lightweight hash (good for reproducibility), - ‘"frequency"‘: labels ordered by frequency (most frequent near one end). Default ‘"even"‘.

`y_scale`
: numeric scalar; vertical scale factor applied to each node’s normalized radial distance from its local center (‘x_offset‘,‘z_offset‘). ‘0‘ keeps the plot flat; try ‘0.5‘–‘0.8‘ for gentle relief. Default ‘0‘.

`jitter_sd`
: numeric; standard deviation of vertical jitter added to nodes, multiplied by the normalized radius so jitter fades toward the center. Default ‘0‘.

`jitter_mode`
: one of ‘"deterministic"‘ or ‘"random"‘. Deterministic jitter derives noise from ‘nodes$path‘ (requires that column); random jitter uses ‘rnorm()‘. Default ‘"deterministic"‘.

`jitter_seed`
: integer or ‘NULL‘; RNG seed for reproducible **random** jitter. Ignored for ‘"deterministic"‘ mode. Default ‘NULL‘.

`return_data`
: logical; if ‘TRUE‘, returns a list with augmented ‘nodes‘ and ‘edges‘ (including computed ‘color‘, ‘width‘, ‘y‘, etc.) instead of just drawing. The plot is still created. Default ‘FALSE‘.

Details

Metric scaling uses the helper ‘.norm_metric()‘ which: 1) rescales the chosen metric to ‘[0,1]‘ over finite values, and 2) applies the selected transform: - ‘"linear"‘: identity, - ‘"sqrt"‘: emphasizes differences at the low end, - ‘"log"‘: ‘log1p(9*t)/log(10)‘, emphasizing very small values.

Node elevation (‘y‘) is computed as ‘y_scale * r_norm‘ where ‘r_norm‘ is the node’s radial distance from its center normalized to the max within that coral. Optional jitter is added (fading to zero at the center). Root nodes (‘step == 0‘) that overlap are vertically stacked with small stems for readability.

Value

Invisibly returns ‘NULL‘ after drawing. If ‘return_data = TRUE‘, returns (invisibly) a list with components: - ‘nodes‘: input ‘nodes‘ with added columns ‘y‘, ‘color‘ (and possibly stacked draw positions for roots), - ‘edges‘: input ‘edges‘ with added columns ‘width‘, ‘color‘, ‘t_color_norm‘, ‘y‘, ‘y_end‘, and ‘width_binned‘.

Requirements

Requires an interactive OpenGL device (‘rgl‘). On headless systems, consider using an off-screen context or skipping examples.

`render_coral_rgl_experimental` *Coral 3D renderer (experimental)*

Description

‘r lifecycle::badge('experimental')‘ API and argument names may change before 0.4.0.

Renders a 3D "coral" plot produced by ‘build_coral_layout()‘, with edge width/color/alpha mapped from association rule metrics and node colors derived from item/type groupings. The function draws a floor grid, edges as 3D segments, nodes as spheres, and optional labels/legend.

Required columns - ‘edges‘: ‘parent_path‘, ‘child_path‘, and metric columns ‘support‘, ‘confidence‘, ‘lift‘. (Endpoint coordinates ‘x‘, ‘y‘, ‘z‘, ‘x_end‘, ‘y_end‘, ‘z_end‘ are recomputed by this function.) - ‘nodes‘: ‘x‘, ‘z‘, ‘x_offset‘, ‘z_offset‘, ‘radius‘, ‘path‘.

Optional columns - ‘nodes$item‘, ‘nodes$feature‘ (for labels/legend & color-by), ‘nodes$step‘ (roots identified as ‘step == 0‘), ‘nodes$interval_label‘, ‘nodes$interval_label_short‘ (label text when requested).

Usage

```
render_coral_rgl_experimental(
  nodes,
  edges,
  grid_size,
  grid_outline = FALSE,
  grid_color = "grey92",
  legend = FALSE,
  legend_style = c("auto", "feature", "grouped", "feature_bins"),
  legend_cex = 1,
  legend_pos = c("inside_right", "topright", "topleft", "custom"),
  legend_items_per_feature = 6L,
  legend_features_max = 10L,
  legend_xy = c(0.92, 0.96),
  legend_panel_width = 0.28,
  legend_panel_margin = 0.02,
  legend_reserve = NULL,
  legend_title_cex = NULL,
  legend_row_cex = NULL,
  legend_col_gap = 0.004,
  label_mode = c("none", "interval", "item", "interval_short", "bin"),
  label_cex = 0.7,
  label_offset = 1.5,
```

```
  label_color = NULL,
  label_non_numeric = c("none", "category", "item"),
  max_labels = 0,
  bin_legend = NULL,
  bin_breaks = NULL,
  bin_infer = TRUE,
  bin_label_fmt = c("index", "roman"),
  theme = c("default", "studio", "flat", "dark", "none"),
  theme_overrides = NULL,
  apply_theme = TRUE,
  edge_width_domain = NULL,
  edge_color_domain = NULL,
  edge_alpha_domain = NULL,
  edge_width_metric = c("confidence", "lift", "support"),
  edge_color_metric = c("confidence", "lift", "support"),
  edge_alpha_metric = NULL,
  edge_width_range = c(1, 5),
  edge_width_transform = c("linear", "sqrt", "log"),
 edge_gradient = c("#2166AC", "#67A9CF", "#D1E5F0", "#FDDBC7", "#EF8A62", "#B2182B"),
  edge_color_transform = c("linear", "sqrt", "log"),
  edge_alpha = 0.5,
  edge_alpha_range = c(0.25, 0.5),
  edge_alpha_transform = c("linear", "sqrt", "log"),
 node_color_by = c("type", "item", "none", "edge_incoming", "edge_outgoing_mean"),
  node_gradient = "match",
  node_gradient_map = c("even", "hash", "frequency"),
  node_scale = 1,
  radial_expand = 1,
  radial_gamma = 1,
  y_scale = 0,
  jitter_sd = 0,
  jitter_mode = c("deterministic", "random"),
  jitter_seed = NULL,
  keep_camera = FALSE,
  view_theta = 0,
  view_phi = NULL,
  view_fov = 60,
  view_zoom = NULL,
  view_userMatrix = NULL,
  return_data = FALSE
)
```

Arguments

nodes	data.frame; typically ‘build_coral_layout()$nodes‘. Must contain ‘x‘, ‘z‘, ‘x_offset‘, ‘z_offset‘, ‘radius‘, ‘path‘. Optional: ‘item‘, ‘feature‘, ‘step‘, ‘interval_label‘, ‘interval_label_short‘.
edges	data.frame; typically ‘build_coral_layout()$edges‘. Must contain ‘parent_path‘,

'child_path', and metric columns 'support', 'confidence', 'lift'. Endpoint columns are ignored if present and will be recomputed.

`grid_size` integer; the layout grid size (usually 'build_coral_layout()$grid_size').

`grid_outline` logical; if 'TRUE' draws the reference grid/guide (using 'grid_color' or the theme's grid color). Defaults to 'FALSE' for clean screenshots.

`grid_color` color for the grid (if 'grid_outline = TRUE'). If missing, the active theme's grid color is used. Default '"grey92"'.

`legend` logical; draw a node legend keyed by base feature ('nodes$feature'). Requires that 'nodes$feature' and node colors are available. Default 'FALSE'.

`legend_style` Character; how to compose the legend. One of '"auto"', '"feature"', '"grouped"', '"feature_bins"'. '"auto"' picks '"feature_bins"' when binning info is available, otherwise '"feature"'.

`legend_cex` Numeric scaling factor for all legend text.

`legend_pos` Character; legend position. One of '"inside_right"', '"topright"', '"topleft"', '"custom"'. '"custom"' uses 'legend_xy'.

`legend_items_per_feature`
Integer; maximum number of items to list per feature before eliding with "...".

`legend_features_max`
Integer; maximum number of distinct features shown in the legend.

`legend_xy` Numeric length-2; normalized device coordinates '(x, y)' used when 'legend_pos = "custom"'.

`legend_panel_width`
Numeric (0-1); fraction of the viewport width reserved for the legend panel when drawing inside the 3D device.

`legend_panel_margin`
Numeric (0-1); margin around the legend panel within the reserved area.

`legend_reserve` Optional numeric (0-1); if provided, overrides 'legend_panel_width' as the fraction of viewport width to carve out for the legend.

`legend_title_cex`
Optional numeric; cex override for the legend title. If 'NULL', a sensible default is used.

`legend_row_cex` Optional numeric; cex override for item rows in the legend. If 'NULL', a sensible default is used.

`legend_col_gap` Numeric; horizontal gap between legend columns (in normalized device coordinates).

`label_mode` one of '"none"', '"interval"', '"item"', '"interval_short"'. Controls label text: interval labels, item labels, or no labels.

`label_cex` numeric; label size passed to 'rgl::text3d()'. Default '0.7'.

`label_offset` numeric; vertical offset (in **node radii**) applied to labels (positive values move labels downward from sphere tops). Default '1.5'.

`label_color` 'NULL' to color labels like their nodes, or a single color / vector to override.

`label_non_numeric`
Character; how to label non-numeric feature values. One of '"none"', '"category"', '"item"'. '"none"' suppresses labels for non-numeric nodes.

`max_labels` integer; maximum number of non-root labels (largest radii first). Root nodes are always kept. If ‘<= 0‘, only root (RHS) labels are drawn.

`bin_legend` Optional ‘data.frame‘ with columns ‘feature‘, ‘bin‘, ‘interval‘ (character), typically passed from ‘build_coral_plots()‘ to drive a ‘"feature_bins"‘ legend.

`bin_breaks` Optional named list ‘list(Feature = numeric breaks)‘ used to compute per-feature bins when ‘bin_legend‘ is not provided.

`bin_infer` Logical; if ‘TRUE‘, infer binning from ‘nodes‘ when neither ‘bin_legend‘ nor ‘bin_breaks‘ is provided.

`bin_label_fmt` how numbers are printed, One of ‘"index‘" or ‘"roman‘". Default ‘"index‘".

`theme` character; one of ‘"default"‘, ‘"studio"‘, ‘"flat"‘, ‘"dark"‘, ‘"none"‘. Selects a preset for lights, materials, and background: - **default**: balanced lighting with subtle specular highlights. - **studio**: brighter, glossy look for screenshots. - **flat**: low-specular, diagram-style shading. - **dark**: dark background with rim lighting. - **none**: no lights configured (use existing rgl state/ambient).

`theme_overrides`
optional named list to partially override the selected theme. Supported keys: ‘background‘ (color), ‘grid_color‘ (color), ‘materials‘ (list with sublists ‘nodes‘, ‘edges‘, ‘labels‘ - each passed to [rgl::material3d()]), and ‘lights‘ (list of argument lists for [rgl::light3d()]). Example: ‘list(lights = list(list(theta = 60, phi = 30)))‘.

`apply_theme` logical; if ‘TRUE‘ (default) the theme is applied at the start of rendering (background, lights, global/material defaults). Set to ‘FALSE‘ to keep the current rgl device state (useful when you configure lights/materials once for batch rendering).

`edge_width_domain`, `edge_color_domain`, `edge_alpha_domain`
optional numeric length-2 vectors giving the global domain (min, max) to use when scaling the respective metric. If ‘NULL‘ (default), the domain is computed from the provided ‘edges‘. Use these to enforce consistent scaling across multiple plots (e.g., faceting).

`edge_width_metric`
character; which metric to map to edge **width**. One of ‘"confidence"‘, ‘"lift"‘, ‘"support"‘. Default ‘"confidence"‘.

`edge_color_metric`
character; which metric to map to edge **color**. One of ‘"confidence"‘, ‘"lift"‘, ‘"support"‘. Default ‘"confidence"‘.

`edge_alpha_metric`
character or ‘NULL‘; which metric to map to edge **alpha** (transparency). One of ‘"support"‘, ‘"lift"‘, ‘"confidence"‘, or ‘NULL‘ to use the constant ‘edge_alpha‘. Default ‘NULL‘.

`edge_width_range`
numeric length-2; min/max line width for edges after scaling. Default ‘c(1, 5)‘.

`edge_width_transform`
character; transformation for width scaling from normalized metric in ‘[0,1]‘. One of ‘"linear"‘, ‘"sqrt"‘, ‘"log"‘. Default ‘"linear"‘.

edge_gradient character vector (>= 2); color ramp for edges, passed to ‘grDevices::colorRamp()‘. Default ‘c("#2166AC","#67A9CF","#D1E5F0","#FDDBC7","#EF8A62","#B2182B")‘.

edge_color_transform character; transformation for color scaling from normalized metric in ‘[0,1]‘. One of ‘"linear"‘, ‘"sqrt"‘, ‘"log"‘. Default ‘"linear"‘.

edge_alpha numeric in ‘[0,1]‘; constant alpha used **only when** ‘edge_alpha_metric‘ is ‘NULL‘. Default ‘0.5‘.

edge_alpha_range numeric length-2 in ‘[0,1]‘; min/max alpha used **only when** ‘edge_alpha_metric‘ is not ‘NULL‘. Default ‘c(0.25, 0.5)‘.

edge_alpha_transform character; transformation for alpha scaling from normalized metric in ‘[0,1]‘. One of ‘"linear"‘, ‘"sqrt"‘, ‘"log"‘. Default ‘"linear"‘.

node_color_by one of ‘"type"‘, ‘"item"‘, ‘"none"‘, ‘"edge_incoming"‘, ‘"edge_outgoing_mean"‘. Controls node coloring: - ‘"type"‘ colors by ‘nodes$feature‘. - ‘"item"‘ colors by ‘nodes$item‘. - ‘"none"‘ leaves existing/implicit colors. - ‘"edge_incoming"‘ uses the mean of incoming edges’ normalized color metric (‘t_color_norm‘). - ‘"edge_outgoing_mean"‘ uses the mean of outgoing edges’ normalized color metric. Default ‘"type"‘.

node_gradient either the string ‘"match"‘ to reuse ‘edge_gradient‘ for nodes, or a character vector (>= 2) of colors to build the node palette. Default ‘"match"‘.

node_gradient_map one of ‘"even"‘, ‘"hash"‘, ‘"frequency"‘; how unique labels are placed along the gradient: - ‘"even"‘: evenly spaced by sorted unique label order, - ‘"hash"‘: stable per-label positions via a lightweight hash (reproducible), - ‘"frequency"‘: labels ordered by frequency (most frequent near one end). Default ‘"even"‘.

node_scale Numeric; global multiplier applied to node radii (size).

radial_expand numeric; global expand/contract factor applied radially from each plot center. ‘1‘ = no change; ‘>1‘ pushes nodes outward. Default ‘1‘.

radial_gamma numeric; curvature of the radial remap. ‘1‘ = linear, ‘>1‘ separates outer rings (more spacing near the edge), ‘<1‘ compresses them. Default ‘1‘.

y_scale numeric scalar; vertical scale factor applied to each node’s normalized radial distance from its local center (‘x_offset‘,‘z_offset‘). ‘0‘ keeps the plot flat; try ‘0.5‘-‘0.8‘ for gentle relief. Default ‘0‘.

jitter_sd numeric; standard deviation of vertical jitter added to nodes, multiplied by the normalized radius so jitter fades toward the center. Default ‘0‘.

jitter_mode one of ‘"deterministic"‘ or ‘"random"‘. Deterministic jitter derives noise from ‘nodes$path‘ (requires that column); random jitter uses ‘rnorm()‘. Default ‘"deterministic"‘.

jitter_seed integer or ‘NULL‘; RNG seed for reproducible **random** jitter. Ignored for ‘"deterministic"‘ mode. Default ‘NULL‘.

keep_camera Logical; if ‘TRUE‘, keep the current rgl camera settings (angle, FOV, zoom). If ‘FALSE‘, apply the view parameters below.

view_theta Numeric; azimuth angle passed to ‘rgl::view3d()‘.

view_phi	Optional numeric; elevation angle. If ‘NULL‘, a sensible default based on grid size is used.
view_fov	Numeric; field of view in degrees (passed to ‘rgl::view3d()‘).
view_zoom	Optional numeric; zoom factor passed to ‘rgl::par3d(zoom=)‘.
view_userMatrix	Optional 4x4 matrix passed to ‘rgl::par3d(userMatrix=)‘ to fully specify the camera transform (overrides theta/phi when provided).
return_data	logical; if ‘TRUE‘, returns a list with augmented ‘nodes‘ and ‘edges‘ (including computed ‘color‘, ‘width‘, ‘y‘, etc.) instead of just drawing. The plot is still created. Default ‘FALSE‘.

Details

Metric scaling uses a helper that: 1) rescales the chosen metric to ‘[0,1]‘ over finite values, and 2) applies the selected transform: - ‘"linear"‘: identity, - ‘"sqrt"‘: emphasizes differences at the low end, - ‘"log"‘: ‘log1p(9*t)/log(10)‘, emphasizing very small values.

Node elevation (‘y‘) is computed as ‘y_scale * r_norm‘ where ‘r_norm‘ is the node’s radial distance from its center normalized to the max within that coral. Optional jitter is added (fading to zero at the center). Root nodes (‘step == 0‘) that overlap are vertically stacked (with small stems drawn). Labels are rendered on top of geometry.

Value

Invisibly returns ‘NULL‘ after drawing. If ‘return_data = TRUE‘, returns (invisibly) a list with components: - ‘nodes‘: input ‘nodes‘ with added columns like ‘y‘ (base elevation) and ‘color‘. - ‘edges‘: input ‘edges‘ with added columns ‘width‘, ‘color‘, ‘alpha‘, ‘t_color_norm‘, ‘y‘, ‘y_end‘, ‘width_binned‘, ‘alpha_binned‘.

Requirements

Requires an interactive OpenGL device (‘rgl‘). On headless systems, consider using an off-screen context or skipping examples.

See Also

[rgl::material3d()], [rgl::light3d()]

---

rs *Simple Random Search*

Description

This function generates a vector of random solutions for a specified length.

Usage

```
rs(candidate_len)
```

Arguments

candidate_len	The length of the vector of random solutions.

Value

A vector of random solutions between 0 and 1.

Examples

```
candidate_len <- 10
random_solutions <- rs(candidate_len)
print(random_solutions)
```

---

supp_conf *Calculate support and confidence for an association rule.*

Description

This function calculates the support and confidence for the given antecedent and consequent in the dataset instances.

Usage

```
supp_conf(antecedent, consequent, instances, features)
```

Arguments

antecedent	The antecedent part of the association rule.
consequent	The consequent part of the association rule.
instances	A data frame representing instances in the dataset.
features	A list containing information about features, including type and bounds.

Value

A list containing support and confidence values.

`write_association_rules_to_csv` *Write Association Rules to CSV file*

Description

This function writes association rules to a CSV file. For time series datasets, it also includes start and end timestamps instead of indices.

Usage

```
write_association_rules_to_csv(
  rules,
  file_path,
  is_time_series = FALSE,
  timestamps = NULL
)
```

Arguments

`rules`	A list of association rules.
`file_path`	The file path for the CSV output.
`is_time_series`	A boolean flag indicating if time series information should be included.
`timestamps`	A vector of timestamps corresponding to the time series data.

Value

No explicit return value. The function writes association rules to a CSV file.

Index