

Package ‘treehunt’

October 16, 2026

Type Package

Title Evolutionary Search for Optimal Node Combinations in Trees

Version 0.0.2

Author Jules Bangard [aut, cre]

Maintainer Jules Bangard <jules.bangard@etu.unistra.fr>

Description Provides Markov chain Monte Carlo (MCMC) sampling and genetic algorithms for searching combinations of nodes in hierarchical trees. Includes scores for binary outcomes, continuous outcomes, and recurrent events, with applications to pharmacovigilance and hospital data. Importance weighting estimates uniform combination score references from tempered MCMC samples and reports weight-based effective sample sizes. Computational routines are implemented in C++ via 'Rcpp'.

License GPL-3

Imports Rcpp (>= 1.1.0), jsonlite, survival

LinkingTo Rcpp

Encoding UTF-8

URL <https://github.com/JulesBa-Git/treehunt>

BugReports <https://github.com/JulesBa-Git/treehunt/issues>

Suggests testthat (>= 3.0.0), umap, dbscan, ggplot2

Config/testthat/edition 3

SystemRequirements C++17

Config/roxygen2/version 8.1.0

NeedsCompilation yes

Repository CRAN

Date/Publication 2026-10-16 13:40:02 UTC

Contents

aggregate_ga_results	2
clustering_genetic_algorithm	3
compute_score	4
decode_treehunt_solutions	5
filter_out_cocktails	6
fit_pwp_rao_context	6
get_dissimilarity_of_list	8
get_taker	9
map_cocktail_names	10
mcmc_size_2_true_score_distribution	10
plot_ga_clusters	12
process_ga_scores	13
refit_pwp_combinations	14
run_ga_batch	15
run_genetic_algorithm	16
run_genetic_algorithm_df_tree	19
run_mcmc	21
run_mcmc_df_tree	24
run_pwp_genetic_algorithm	27
run_pwp_mcmc	28
score_pwp_combinations	30
summarize_ga_pipeline	31
uniform_score_reference	32
weight_ess	33
Index	35

aggregate_ga_results	<i>Aggregate GA Results from Multiple JSON Files returned by run_ga_batch</i>
----------------------	---

Description

Aggregate GA Results from Multiple JSON Files returned by run_ga_batch

Usage

aggregate_ga_results(folder_path = "results")

Arguments

folder_path Path to the folder containing result JSON files.

Value

A data frame of unique cocktails with their scores.

clustering_genetic_algorithm

Clustering of a Cocktail List using DBSCAN on UMAP Projection

Description

Projects cocktail dissimilarities into 2D space using UMAP and identifies clusters using DBSCAN.

Usage

```
clustering_genetic_algorithm(
  cocktails,
  patient_data,
  node_column,
  target_column,
  tree_df,
  depth_column,
  upper_bound_column = NULL,
  name_column = NULL,
  score_type = "wilcoxon",
  umap_config = NULL,
  eps_dbscan = 0.2,
  min_pts_dbscan = 5
)
```

Arguments

cocktails	A list of 1-based integer vectors or a data frame containing them. For a data frame, <code>vec_cocktails</code> is the preferred list-column name; otherwise exactly one list-column must be present.
patient_data	The dataset used for scoring.
node_column	Column in <code>patient_data</code> containing zero-based node indices.
target_column	Column containing the outcome (for example, "QT_c").
tree_df	The tree structure data frame.
depth_column	Column in <code>tree_df</code> containing node depths.
upper_bound_column	Column in <code>tree_df</code> containing zero-based subtree bounds.
name_column	Optional node-label column in <code>tree_df</code> .
score_type	The scoring method (e.g., "Wilcoxon", "RR", "phyper").
umap_config	Optional configuration list for UMAP. Defaults to <code>umap.defaults</code> .
eps_dbscan	Distance parameter for DBSCAN. Default is 0.2.
min_pts_dbscan	Minimum points to form a cluster in DBSCAN. Default is 5.

Value

A data frame with one row per input combination, in the same order. UMAP1 and UMAP2 are numeric coordinates of the two-dimensional UMAP embedding of the combination dissimilarities. `cluster` is the integer DBSCAN assignment: positive values identify clusters and zero marks noise. If `cocktails` is a data frame, its original columns are retained; otherwise a `cocktails` list-column contains the supplied one-based node vectors.

compute_score	<i>Compute scores for supplied node combinations</i>
---------------	--

Description

Compute scores for supplied node combinations

Usage

```
compute_score(
  cocktail_list,
  patient_data,
  node_column,
  target_column,
  tree,
  depth_column,
  id_column = NULL,
  upper_bound_column = NULL,
  name_column = NULL,
  score_type = "hypergeometric"
)
```

Arguments

<code>cocktail_list</code>	List of integer vectors identifying tree rows with one-based R indices. The returned solutions retain these indices.
<code>patient_data</code>	Observation data frame. Its node list-column uses the package's zero-based internal tree indices.
<code>node_column</code>	Name or one-based position of the node list-column.
<code>target_column</code>	Name or one-based position of the outcome column.
<code>tree</code>	Tree data frame in depth-first order.
<code>depth_column</code>	Name or one-based position of the tree-depth column.
<code>id_column</code>	Optional name or one-based position of the observation-unit identifier. It is required by patient-level continuous-outcome scores.
<code>upper_bound_column</code>	Optional name or one-based position of the zero-based inclusive subtree upper-bound column.
<code>name_column</code>	Optional name or one-based position of the node-label column.
<code>score_type</code>	Registered score implementation to evaluate.

Details

Candidate vectors deliberately use ordinary one-based R row positions at this user-facing boundary. They are checked against the tree before being converted once to the zero-based representation used by the C++ scoring engine.

Value

A list containing the supplied combinations, their scores, coverage counts, and score-specific summaries.

`decode_treehunt_solutions`*Decode treehunt node indexes*

Description

Decode treehunt node indexes

Usage

```
decode_treehunt_solutions(  
  combinations,  
  tree,  
  index_base = 0L,  
  code_column = "Code"  
)
```

Arguments

<code>combinations</code>	List of node-index vectors.
<code>tree</code>	Tree data frame.
<code>index_base</code>	Index base of combinations.
<code>code_column</code>	Column containing node labels.

Value

A list of character vectors, one per input combination in the same order. Each vector contains the labels from `tree[[code_column]]` for the supplied node indices, in their original order. The input index base is handled by `index_base`.

filter_out_cocktails	<i>Filter a list of cocktails according to the mean depth of nodes inside each cocktail. A cocktail is kept if the mean depth of his nodes is greater than mean_depth_cocktail (default = 3). This function aim is to provide more interpretable cocktail to the user.</i>
----------------------	--

Description

Filter a list of cocktails according to the mean depth of nodes inside each cocktail. A cocktail is kept if the mean depth of his nodes is greater than mean_depth_cocktail (default = 3). This function aim is to provide more interpretable cocktail to the user.

Usage

```
filter_out_cocktails(
  results,
  tree_depth,
  mean_depth_cocktail = 3,
  one_index = FALSE
)
```

Arguments

results	results returned by function summarize_ga_pipeline. Or a list of string of 0 indexed vector of tree nodes. (If nodes are 1-indexed, make sure that the parameter one_index is set to TRUE).
tree_depth	The tree structure depth.
mean_depth_cocktail	The desired minimal mean depth of nodes in a cocktail.
one_index	A boolean telling if the cocktails are 1-indexed or 0-indexed. Default is FALSE

Value

The filtered list of cocktail

fit_pwp_rao_context	<i>Fit the reduced PWP model and prepare a Rao-score context</i>
---------------------	--

Description

The reduced Cox model is fitted once. Its linear predictor, nuisance design matrix and model-based nuisance variance are stored in a compact context used by the optimized C++ score engine. Patient-clustered variance is retained in the fitted model for reporting, while the high-throughput Rao statistic uses the exact model-based efficient information.

Usage

```

fit_pwp_rao_context(
  data,
  status_column,
  time_column = "gap_time",
  strata_column = "event_stratum_25",
  id_column = "id",
  covariates = c("age_start", "index_emergency", "year_start"),
  ties = "efron",
  positive_only = TRUE,
  min_covered_patients = 1L,
  min_covered_events = 1L
)

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

```

Arguments

<code>data</code>	Interval-level data frame.
<code>status_column</code>	Binary endpoint column.
<code>time_column</code>	Gap-time column.
<code>strata_column</code>	PWP event-order stratum.
<code>id_column</code>	Patient identifier used for robust clustering.
<code>covariates</code>	Character vector of reduced-model covariates.
<code>ties</code>	Cox tied-time method. The optimized engine currently requires "efron".
<code>positive_only</code>	If TRUE, the search fitness is $\max(0, \text{signed_z})$; full signed statistics are still returned during detailed evaluation.
<code>min_covered_patients</code>	Minimum number of distinct exposed patients for a candidate to receive non-zero search fitness.
<code>min_covered_events</code>	Minimum number of exposed endpoint events for a candidate to receive non-zero search fitness.
<code>x</code>	A <code>pwp_rao_context</code> to print.
<code>...</code>	Additional print arguments (currently ignored).

Value

A list of class `pwp_rao_context` containing the reduced coxph fit, the numeric arrays in `score_data` used by the score engine, `row_index` (one-based indices of retained analysis rows), and specification (model settings and analysis counts). The print method displays a summary and returns the same object invisibly.

```
get_dissimilarity_of_list
```

Compute the dissimilarity matrix on a list of cocktails

Description

Compute the dissimilarity matrix on a list of cocktails

Usage

```
get_dissimilarity_of_list(
  cocktail_list,
  patient_data,
  node_column,
  target_column,
  tree,
  depth_column,
  upper_bound_column = NULL,
  name_column = NULL,
  score_type = "wilcoxon"
)
```

Arguments

<code>cocktail_list</code>	List of integer vectors identifying tree rows with one-based R indices. The returned solutions retain these indices.
<code>patient_data</code>	Observation data frame. Its node list-column uses the package's zero-based internal tree indices.
<code>node_column</code>	Name or one-based position of the node list-column.
<code>target_column</code>	Name or one-based position of the outcome column.
<code>tree</code>	Tree data frame in depth-first order.
<code>depth_column</code>	Name or one-based position of the tree-depth column.
<code>upper_bound_column</code>	Optional name or one-based position of the zero-based inclusive subtree upper-bound column.
<code>name_column</code>	Optional name or one-based position of the node-label column.
<code>score_type</code>	Registered score implementation to evaluate.

Value

A symmetric numeric matrix of normalized greedy tree-edit dissimilarities.

get_taker

*Identify observations covered by hierarchical combinations***Description**

Identify observations covered by hierarchical combinations

Usage

```
get_taker(
  cocktail_list,
  patient_data,
  node_column,
  target_column,
  id_column,
  hadm_column,
  tree,
  depth_column,
  upper_bound_column = NULL,
  name_column = NULL
)
```

Arguments

cocktail_list	List of zero-based node-index vectors.
patient_data	Observation data frame. Its node list-column uses the package's zero-based internal tree indices.
node_column	Name or one-based position of the node list-column.
target_column	Name or one-based position of the outcome column.
id_column	Name of the integer patient identifier column.
hadm_column	Name of the integer admission identifier column.
tree	Tree data frame in depth-first order.
depth_column	Name or one-based position of the tree-depth column.
upper_bound_column	Optional name or one-based position of the zero-based inclusive subtree upper-bound column.
name_column	Optional name or one-based position of the node-label column.

Value

A list with the supplied combinations (cocktail_list), distinct patient identifiers (Takers_set), admission identifiers (hadm_set), and one-based observation rows (idx_set) for each combination.

map_cocktail_names	<i>Map Cocktail Indices to Names Saved GA combinations use zero-based node indices; tree rows use one-based indices.</i>
--------------------	--

Description

Map Cocktail Indices to Names Saved GA combinations use zero-based node indices; tree rows use one-based indices.

Usage

```
map_cocktail_names(aggregated_df, new_tree)
```

Arguments

aggregated_df	The data frame returned by aggregate_ga_results.
new_tree	Tree data frame with Name and Code columns.

Value

The data frame with cocktail_names and cocktail_codes columns.

mcmc_size_2_true_score_distribution	<i>Run MCMC Algorithm to Compute the Score Distribution Among Node Combinations of Size 2</i>
-------------------------------------	---

Description

Performs a Modified Metropolis-Hastings MCMC sampling to compute the score distribution of nodes combination of size *cocktail_size*.

Usage

```
mcmc_size_2_true_score_distribution(
  patient_data,
  node_column,
  target_column,
  tree,
  depth_column,
  upper_bound_column = NULL,
  name_column = NULL,
  id_column = NULL,
  beta = 4,
  max_score = 200,
  score_type = "hypergeometric"
)
```

Arguments

<code>patient_data</code>	A data.frame containing patient information with at least a node column and a target column.
<code>node_column</code>	Either a string (column name) or integer (column index, 1-based) specifying the column containing drug codes. This column should be either: • A list of integer vectors: <code>list(c(1,2), c(3), c(4,5))</code> • A character vector with comma-separated values: <code>c("1,2", "3", "4,5")</code>
<code>target_column</code>	Either a string (column name) or integer (column index, 1-based) specifying the target/outcome column. Integer values are treated as binary for now, numeric values with non-0/1 entries are treated as continuous.
<code>tree</code>	A data.frame containing the structural definition of the tree.
<code>depth_column</code>	Either a string or integer specifying the column in <code>tree</code> that contains the node depth levels.
<code>upper_bound_column</code>	(Optional) Either a string or integer (1-based index) specifying the column in <code>tree_depth</code> that contains upper bound of nodes. Defaults to NULL.
<code>name_column</code>	(Optional) Either a string or integer (1-based index) specifying the column in <code>tree</code> that contains the corresponding name of nodes. Defaults to NULL.
<code>id_column</code>	Optional observation-unit identifier column, given by name or one-based column position; needed for patient-level scoring.
<code>beta</code>	Strict support threshold: filtered results require more than <code>beta</code> covered observations (or distinct patients for patient-level scores). Default: 4.
<code>max_score</code>	Finite positive cap applied to scores before Metropolis-Hastings acceptance, importance weighting and binning. Default: 200.0.
<code>score_type</code>	Scoring function to use. Either "hypergeometric" for the hypergeometric test, "relative_risk" for relative risk calculation, or "wilcoxon" for the wilcoxon test with continuous output. Default: "hypergeometric".

Details

Enumerates each unordered pair of distinct nodes exactly once and retains pairs present in at least one observation, including ancestor-descendant pairs. It uses the same state space, cap and strict support filter as `run_mcmc_df_tree(cocktail_size = 2)`. No MCMC or importance weighting is performed: each enumerated pair has unit weight. The returned `uniform_reference` is exact; its `weight_ess` equals the number of enumerated pairs. Acceptance statistics are not applicable.

Value

A list containing:

- top_solutions** List of zero-based node vectors for the top scoring solutions
- top_scores** Numeric vector of scores for the top solutions
- top_solutions_filtered** Top solutions meeting the beta threshold
- top_scores_filtered** Scores for the filtered solutions

score_distribution Histogram of retained capped scores (0.1-wide bins and a cap bin)
score_distribution_filtered Histogram for solutions meeting beta threshold
outstanding_scores Retained capped scores equal to max_score
uniform_reference Uniform-reference distribution and weight ESS
uniform_reference_filtered Support-filtered uniform reference and ESS
reference_parameters Temperature, score cap, support and burn-in settings
trace Retained trajectory, only when store_trace is TRUE
statistics List of run statistics including acceptance rates

Examples

```
dat <- data.frame(outcome = c(1L, 1L, 0L, 0L))
dat$nodes <- list(c(0L, 1L), 0L, 1L, 1L)
tree <- data.frame(Depth = c(1L, 1L))
result <- mcmc_size_2_true_score_distribution(dat, "nodes", "outcome", tree, "Depth")
result$uniform_reference$weight_ess
```

plot_ga_clusters	<i>Plot GA Clustering Results</i>
------------------	-----------------------------------

Description

This wrapper runs the clustering pipeline and produces a UMAP scatter plot where points are colored by their identified cluster.

Usage

```
plot_ga_clusters(cocktails, point_size = 2, alpha = 0.7, ...)
```

Arguments

cocktails	A list of 1-based integer vectors or a data frame containing them.
point_size	Numeric. Size of the points in the scatter plot. Default is 2.
alpha	Numeric. Transparency of the points. Default is 0.7.
...	Arguments passed directly to clustering_genetic_algorithm .

Value

A list containing:

- data: The data frame with UMAP coordinates and cluster assignments.
- plot: A ggplot2 object.

process_ga_scores	<i>Process and Attach GA Scores and Statistics (Generic)</i>
-------------------	--

Description

Process and Attach GA Scores and Statistics (Generic)

Usage

```
process_ga_scores(
  df,
  patient_data,
  tree_df,
  node_column,
  target_column,
  depth_column,
  upper_bound_column,
  score_type = "composite",
  ...,
  id_column = NULL,
  name_column = NULL
)
```

Arguments

<code>df</code>	The data frame from <code>map_cocktail_names</code> .
<code>patient_data</code>	The dataset used for scoring.
<code>tree_df</code>	The tree structure data frame.
<code>node_column</code>	Column in <code>patient_data</code> containing zero-based node indices.
<code>target_column</code>	Column containing the outcome (for example, "QT_c").
<code>depth_column</code>	Column in <code>tree_df</code> containing node depths.
<code>upper_bound_column</code>	Column in <code>tree_df</code> containing zero-based subtree bounds.
<code>score_type</code>	The scoring method (e.g., "Wilcoxon", "RR", "phyper").
<code>...</code>	Additional arguments passed to <code>compute_score</code> .
<code>id_column</code>	Optional observation-unit identifier required by patient-level continuous-outcome scores.
<code>name_column</code>	Optional node-label column in <code>tree_df</code> .

Value

A data frame preserving the rows and columns of `df`, with `taker_count` (support reported by [compute_score\(\)](#)) and `scores` (the recomputed search score). For continuous-outcome scores, `QT_mean`, `QT_median`, `QT_min`, and `QT_max` summarize each combination's `QT_diff_distribution`

returned by `compute_score()`. These are summaries of outcomes among covered observations, not contrasts with the unexposed group. For the Wilcoxon score, this distribution contains one median per covered patient; for the other continuous scores it contains covered observation-level outcomes. An empty or entirely missing distribution is summarized as zero. Existing columns with these names are replaced.

refit_pwp_combinations

Refit selected combinations with patient-clustered Cox models

Description

This step is intentionally separate from high-throughput screening. It estimates hazard ratios and robust confidence intervals for a small set of selected combinations.

Usage

```
refit_pwp_combinations(
  combinations,
  data,
  context,
  tree,
  index_base = 0L,
  node_column = "icd10_indices",
  depth_column = "Depth",
  upper_bound_column = "UpperBound",
  name_column = "Code",
  keep_fits = FALSE
)
```

Arguments

<code>combinations</code>	List of node-index vectors.
<code>data</code>	Original interval-level data used to fit context.
<code>context</code>	A <code>pwp_rao_context</code> .
<code>tree</code>	Tree data frame.
<code>index_base</code>	Either 0 or 1.
<code>node_column</code>	List-column of 0-based tree indexes.
<code>depth_column, upper_bound_column, name_column</code>	Tree column names.
<code>keep_fits</code>	Retain fitted coxph objects in a list-column.

Value

For a non-empty list of combinations, a data frame with one row per combination in input order, containing numeric columns:

- `estimate`: fitted log hazard ratio for the combination indicator.
- `robust_se`: patient-clustered standard error of estimate.
- `hazard_ratio`: $\exp(\text{estimate})$, conditional on the model covariates and event-order strata.
- `conf_low`, `conf_high`: lower and upper limits of the 95% Wald confidence interval for the hazard ratio, using `robust_se`.
- `robust_z`, `robust_p_value`: Wald statistic and two-sided normal p-value for a zero log hazard ratio.
- `exposed_intervals`, `exposed_events`: counts of covered intervals and endpoint events among the analysis rows.

The combination list-column retains the input node vectors and index base. When `keep_fits = TRUE`, a fit list-column also contains the corresponding fitted coxph objects. With no combinations, the existing interface returns a list with an empty combination component (and an empty fit component if requested).

run_ga_batch

Run Multiple Genetic Algorithm Replicates from Configuration

Description

Run Multiple Genetic Algorithm Replicates from Configuration

Usage

```
run_ga_batch(
  config_path,
  patient_data,
  tree_df,
  seed_population = NULL,
  replicates = 5,
  output_dir
)
```

Arguments

<code>config_path</code>	Path to a JSON file containing GA hyperparameters.
<code>patient_data</code>	The data frame containing patient/node observations.
<code>tree_df</code>	The data frame defining the tree structure.
<code>seed_population</code>	Optional initial combinations, using one-based tree indices.
<code>replicates</code>	Number of times to run each configuration. Default is 5.

output_dir Required output directory, supplied explicitly by the caller. It is created if it does not exist. One JSON file is written per configuration; an existing file with the same name is overwritten.

Value

No return value (NULL, invisibly). Called for its side effects: runs the requested GA replicates and writes one `results_<name>.json` file per configuration to `output_dir`. Each file contains a list of replicate results with `final_population` (zero-based node-index vectors), `final_scores`, parameters, statistics, and metadata (replicate number, configuration name, and timestamp). Files can be read with [aggregate_ga_results\(\)](#).

run_genetic_algorithm *Run Genetic Algorithm for High Score Nodes Combination Search*

Description

Performs a genetic algorithm search to find optimal node combinations that maximize a specified score function. The algorithm evolves a population of solutions through selection, crossover, and mutation operations.

Usage

```
run_genetic_algorithm(
  patient_data,
  node_column,
  target_column,
  tree_depth,
  seed_population = NULL,
  population_size = 100L,
  epochs = 1000L,
  mutation_rate = 0.1,
  probab_mutation_type1 = 0.2,
  crossover_rate = 0.8,
  elite_count = 0L,
  tournament_size = 3L,
  alpha = 1,
  score_type = "hypergeometric",
  diversity = FALSE,
  verbose = FALSE,
  seed = NULL
)
```

Arguments

patient_data A data.frame containing patient information with at least a node column and a target column.

node_column	Either a string (column name) or integer (column index, 1-based) specifying the column containing node indexes. This column should be either: • A list of integer vectors: <code>list(c(1,2), c(3), c(4,5))</code> (0 indexed Note : Might be wise to change this) • A character vector with comma-separated values: <code>c("1,2", "3", "4,5")</code>
target_column	Either a string (column name) or integer (column index, 1-based) specifying the target/outcome column. Integer values are treated as binary, numeric values with non-0/1 entries are treated as continuous.
tree_depth	An integer vector specifying the depth of each node in the tree structure. Must start at depth 1 and children must be at depth+1 of their parent.
seed_population	A list of integer vectors representing an initial population (vector of 1-based tree index). If provided, these individuals will be included in the first generation . If the list contains fewer individuals than <code>population_size</code> , the remainder will be initialized randomly. If NULL (the default), the entire initial population is generated randomly.
population_size	Number of solutions in the population. Default: 100.
epochs	Number of generations to evolve. Default: 1000.
mutation_rate	Probability of mutating each offspring. Default: 0.1.
prob_mutation_type1	When mutation occurs, probability of using Type 1 (add/remove) vs Type 2 (swap) mutation. Default: 0.2.
crossover_rate	Probability of applying crossover to selected parents. Default: 0.8.
elite_count	Number of top solutions to preserve unchanged each generation. Default: 0.
tournament_size	Number of solutions competing in tournament selection. Default: 2.
alpha	Parameter controlling the add/remove mutation bias. Higher values favor adding nodes. Default: 1.0.
score_type	Scoring function to use. Either "hypergeometric" for the hypergeometric test, "relative_risk" for relative risk calculation, or "wilcoxon" for the wilcoxon test with continuous output.
diversity	If TRUE, applies a diversity penalty to encourage exploration of different solutions. Default: FALSE.
verbose	If TRUE, prints progress during the run. Default: FALSE.
seed	Optional non-negative integer seed for the C++ random-number generator. If NULL, the generator is initialized non-deterministically.

Details

The genetic algorithm uses the following operators:

- **Selection:** Tournament selection with configurable size
- **Crossover:** Single-point crossover on tree subtrees

- **Mutation Type 1:** Add or remove a node (probability controlled by alpha)
- **Mutation Type 2:** Swap a node with its parent or child
- **Elitism:** Top solutions preserved unchanged

The algorithm maintains a score cache to avoid redundant computations for solutions that have been seen before (identified by hash).

When diversity = TRUE, solutions are penalized based on their similarity to other solutions in the population, encouraging exploration of diverse regions.

Value

A list containing:

final_population The complete final population of solutions

final_scores Scores for all solutions in the final population

parameters List of parameters used for the run

statistics Additional information about cache hits etc.

See Also

[run_mcmc](#) for an MCMC-based optimization approach

Examples

```
# A root and three leaf nodes; observations use zero-based leaf indices.
patient_df <- data.frame(outcome = c(1L, 0L, 1L, 0L, 1L, 0L, 1L, 0L))
patient_df$drugs <- list(1L, 2L, c(1L, 3L), 3L,
                        c(1L, 2L), c(2L, 3L), 1L, 2L)
tree_depth <- c(1L, 2L, 2L, 2L)

results <- run_genetic_algorithm(
  patient_data = patient_df,
  node_column = "drugs",
  target_column = "outcome",
  tree_depth = tree_depth,
  # Initial combinations use one-based tree indices.
  seed_population = list(2L, 3L, 4L, c(2L, 3L), c(2L, 4L), c(3L, 4L)),
  population_size = 6,
  epochs = 10,
  score_type = "hypergeometric",
  seed = 123
)
head(results$final_scores)
head(results$final_population)
```

run_genetic_algorithm_df_tree

Run Genetic Algorithm for High Score Nodes Combination Search

Description

Performs a genetic algorithm search to find optimal node combinations that maximize a specified score function. This version builds the internal tree structure from a provided data frame mapping node depths and bounds.

Usage

```
run_genetic_algorithm_df_tree(
  patient_data,
  node_column,
  target_column,
  tree,
  depth_column,
  upper_bound_column = NULL,
  name_column = NULL,
  id_column = NULL,
  seed_population = NULL,
  population_size = 100L,
  epochs = 1000L,
  mutation_rate = 0.1,
  probab_mutation_type1 = 0.2,
  crossover_rate = 0.8,
  elite_count = 0L,
  tournament_size = 3L,
  alpha = 1,
  score_type = "hypergeometric",
  diversity = FALSE,
  verbose = FALSE,
  seed = NULL
)
```

Arguments

patient_data	A data.frame containing patient information.
node_column	Either a string (column name) or integer (1-based index) specifying the column containing node indexes (list of vectors or comma-separated strings).
target_column	Either a string (column name) or integer (1-based index) specifying the target/outcome column.
tree	A data.frame containing the structural definition of the tree.
depth_column	Either a string or integer specifying the column in tree that contains the node depth levels.

upper_bound_column	(Optional) Either a string or integer (1-based index) specifying the column in tree_depth that contains upper bound of nodes. Defaults to NULL.
name_column	(Optional) Either a string or integer (1-based index) specifying the column in tree that contains the corresponding name of nodes. Defaults to NULL.
id_column	Optional observation-unit identifier column, given by name or one-based column position; needed for patient-level scoring.
seed_population	A list of integer vectors representing an initial population (vector of 1-based tree index). If provided, these individuals will be included in the first generation. If the list contains fewer individuals than population_size, the remainder will be initialized randomly. If NULL (the default), the entire initial population is generated randomly.
population_size	Number of solutions in the population. Default: 100.
epochs	Number of generations to evolve. Default: 1000.
mutation_rate	Probability of mutating each offspring. Default: 0.1.
prob_mutation_type1	Probability of using Type 1 (add/remove) vs Type 2 (swap) mutation. Default: 0.2.
crossover_rate	Probability of applying crossover to selected parents. Default: 0.8.
elite_count	Number of top solutions to preserve unchanged each generation. Default: 0.
tournament_size	Number of solutions competing in tournament selection. Default: 3.
alpha	Parameter controlling add/remove mutation bias. Higher values favor adding nodes. Default: 1.0.
score_type	Scoring function: "hypergeometric", "relative_risk", or "wilcoxon".
diversity	If TRUE, applies a diversity penalty to encourage exploration. Default: FALSE.
verbose	If TRUE, prints progress during the run. Default: FALSE.
seed	Optional non-negative integer seed for the C++ random-number generator. If NULL, the generator is initialized non-deterministically.

Details

Unlike the vector-based version, this function extracts tree hierarchy from the tree_depth data frame. It uses depth_column and optionally upper_bound_column and name_column to define the tree.

Value

A list containing:

final_population The complete final population of solutions, 0-based tree index. TODO -> return 1 based (R-like) index for more coherence

final_scores Scores for all solutions in the final population

parameters List of parameters used for the run

statistics Additional information about cache hits, etc.

Examples

```
# A root and three leaf nodes; observations use zero-based leaf indices.
patient_df <- data.frame(outcome = c(1L, 0L, 1L, 0L, 1L, 0L, 1L, 0L))
patient_df$drugs <- list(1L, 2L, c(1L, 3L), 3L,
                        c(1L, 2L), c(2L, 3L), 1L, 2L)

tree_df <- data.frame(
  node_id = 0:3,
  depth_level = c(1L, 2L, 2L, 2L)
)

results <- run_genetic_algorithm_df_tree(
  patient_data = patient_df,
  node_column = "drugs",
  target_column = "outcome",
  tree = tree_df,
  depth_column = "depth_level",
  # Initial combinations use one-based tree indices.
  seed_population = list(2L, 3L, 4L, c(2L, 3L), c(2L, 4L), c(3L, 4L)),
  population_size = 6,
  epochs = 10,
  score_type = "hypergeometric",
  seed = 123
)
head(results$final_scores)
head(results$final_population)
```

run_mcmc

Run MCMC Algorithm for Estimation of Score Distribution Among Nodes of The Tree

Description

Performs a Modified Metropolis-Hastings MCMC sampling to estimate the score distribution of nodes combination of a given *cocktail_size*. The algorithm explores the space of tree combinations using a proposal law composed of two mutation types.

Usage

```
run_mcmc(
  patient_data,
  node_column,
  target_column,
  tree_depth,
  epochs,
  temperature = 1,
  n_results = 10,
  cocktail_size = 2,
```

```

    prob_type1 = 0.01,
    beta = 4,
    max_score = 200,
    score_type = "hypergeometric",
    verbose = FALSE,
    seed = NULL,
    burn_in = 0,
    store_trace = FALSE,
    id_column = NULL
)

```

Arguments

patient_data	A data.frame containing patient information with at least a node column and a target column.
node_column	Either a string (column name) or integer (column index, 1-based) specifying the column containing drug codes. This column should be either: • A list of integer vectors: <code>list(c(1,2), c(3), c(4,5))</code> • A character vector with comma-separated values: <code>c("1,2", "3", "4,5")</code>
target_column	Either a string (column name) or integer (column index, 1-based) specifying the target/outcome column. Integer values are treated as binary for now, numeric values with non-0/1 entries are treated as continuous.
tree_depth	An integer vector specifying the depth of each node in the tree structure. Must start at depth 1 and children must be at depth+1 of their parent.
epochs	Number of MCMC iterations to run.
temperature	Temperature parameter for the Metropolis-Hastings acceptance probability. Higher values lead to an easiest acceptance of lower score. Default: 1.0.
n_results	Number of top solutions to track and return. Default: 10.
cocktail_size	Target size of drug combinations to search for. Default: 2.
prob_type1	Probability of using Type 1 mutation (random generation) vs Type 2 mutation (local swap). Default: 0.01.
beta	Strict support threshold: filtered results require more than beta covered observations (or distinct patients for patient-level scores). Default: 4.
max_score	Finite positive cap applied to scores before Metropolis-Hastings acceptance, importance weighting and binning. Default: 200.0.
score_type	Scoring function to use. Either "hypergeometric" for the hypergeometric test, "relative_risk" for relative risk calculation, or "wilcoxon" for the wilcoxon test with continuous output. Default: "hypergeometric".
verbose	If TRUE, prints progress and statistics during the run. Default: FALSE.
seed	Optional non-negative integer seed for the C++ random-number generator. If NULL, the generator is initialized non-deterministically.
burn_in	Number of initial iterations excluded from all distributions, top solutions and the optional trace. Must be smaller than epochs. Default: 0.

store_trace	If TRUE, return the retained capped score, support and zero-based node vector for each iteration. Default FALSE avoids this storage.
id_column	Optional observation-unit identifier column, given by name or one-based position. Required for Wilcoxon and residual patient-level scores.

Details

The stationary target on observed subsets of `cocktail_size` distinct nodes is proportional to $\exp(S(C)/T)$, where S is capped at `max_score`. Ancestor-descendant pairs are included in this MCMC space; genetic-algorithm validity filtering is a separate operation.

Type 1 proposes a uniform subset of distinct nodes. Type 2 replaces a node by a parent or child not already in the combination. Its Hastings correction is $|V(C)|/|V(C')|$, counting these possible local moves before rejecting proposals absent from the data. Each acceptance probability is the minimum of 1 and $\exp((S(C') - S(C))/T)$ times the applicable proposal correction. A positive `prob_type1` connects the observed state space.

At every retained iteration, including rejected moves, weights proportional to $\exp(-S(C)/T)$ recover the uniform combination reference. Weights are computed from exact capped scores before grouping into bins. They are rescaled internally to avoid numerical underflow. The filtered reference conditions on support strictly greater than `beta`, with its own ESS. This estimates a uniform law on combinations, not on distinct score values.

`uniform_reference` and `uniform_reference_filtered` each contain `distribution`, `n_samples`, and `weight_ess`. The distribution columns are `lower`, `upper`, `probability`, `cdf` and `upper_tail`. Bins are `[lower, upper)`, except the final singleton bin at the cap. CDF includes the current bin; `upper_tail` includes it and all subsequent bins. Empty references have NA probabilities and ESS. ESS is $(\sum w)^2 / \sum w^2$; it measures weight concentration and does not account for MCMC autocorrelation. Check mixing separately and choose an appropriate `burn_in`. See [uniform_score_reference](#) for post-processing an optionally stored trace without binning.

Value

A list containing:

top_solutions List of zero-based node vectors for the top scoring solutions

top_scores Numeric vector of scores for the top solutions

top_solutions_filtered Top solutions meeting the `beta` threshold

top_scores_filtered Scores for the filtered solutions

score_distribution Histogram of retained capped scores (0.1-wide bins and a cap bin)

score_distribution_filtered Histogram for solutions meeting `beta` threshold

outstanding_scores Retained capped scores equal to `max_score`

uniform_reference Uniform-reference distribution and weight ESS

uniform_reference_filtered Support-filtered uniform reference and ESS

reference_parameters Temperature, score cap, support and burn-in settings

trace Retained trajectory, only when `store_trace` is TRUE

statistics List of run statistics including acceptance rates

Examples

```
dat <- data.frame(outcome = c(1L, 1L, 0L, 0L))
dat$nodes <- list(c(0L, 1L), 0L, 1L, 1L)
tree <- data.frame(Depth = c(1L, 1L))
result <- run_mcmc(dat, "nodes", "outcome", c(1L, 1L),
  epochs = 100, cocktail_size = 1, seed = 42)
result$uniform_reference$weight_ess
```

run_mcmc_df_tree	<i>Run MCMC Algorithm for Estimation of Score Distribution Among Nodes of The Tree</i>
------------------	--

Description

Performs a Modified Metropolis-Hastings MCMC sampling to estimate the score distribution of nodes combination of a given *cocktail_size*. The algorithm explores the space of tree combinations using a proposal law composed of two mutation types.

Usage

```
run_mcmc_df_tree(
  patient_data,
  node_column,
  target_column,
  tree,
  depth_column,
  upper_bound_column = NULL,
  name_column = NULL,
  id_column = NULL,
  epochs = 1e+06,
  temperature = 1,
  n_results = 10,
  cocktail_size = 2,
  prob_type1 = 0.01,
  beta = 4,
  max_score = 200,
  score_type = "hypergeometric",
  verbose = FALSE,
  seed = NULL,
  burn_in = 0,
  store_trace = FALSE
)
```

Arguments

patient_data	A data.frame containing patient information with at least a node column and a target column.
node_column	Either a string (column name) or integer (column index, 1-based) specifying the column containing drug codes. This column should be either: • A list of integer vectors: <code>list(c(1,2), c(3), c(4,5))</code> • A character vector with comma-separated values: <code>c("1,2", "3", "4,5")</code>
target_column	Either a string (column name) or integer (column index, 1-based) specifying the target/outcome column. Integer values are treated as binary for now, numeric values with non-0/1 entries are treated as continuous.
tree	A data.frame containing the structural definition of the tree.
depth_column	Either a string or integer specifying the column in tree that contains the node depth levels.
upper_bound_column	(Optional) Either a string or integer (1-based index) specifying the column in tree_depth that contains upper bound of nodes. Defaults to NULL.
name_column	(Optional) Either a string or integer (1-based index) specifying the column in tree that contains the corresponding name of nodes. Defaults to NULL.
id_column	Optional observation-unit identifier column, given by name or one-based column position; needed for patient-level scoring.
epochs	Number of MCMC iterations to run.
temperature	Temperature parameter for the Metropolis-Hastings acceptance probability. Higher values lead to an easiest acceptance of lower score. Default: 1.0.
n_results	Number of top solutions to track and return. Default: 10.
cocktail_size	Target size of drug combinations to search for. Default: 2.
prob_type1	Probability of using Type 1 mutation (random generation) vs Type 2 mutation (local swap). Default: 0.01.
beta	Strict support threshold: filtered results require more than beta covered observations (or distinct patients for patient-level scores). Default: 4.
max_score	Finite positive cap applied to scores before Metropolis-Hastings acceptance, importance weighting and binning. Default: 200.0.
score_type	Scoring function to use. Either "hypergeometric" for the hypergeometric test, "relative_risk" for relative risk calculation, or "wilcoxon" for the wilcoxon test with continuous output. Default: "hypergeometric".
verbose	If TRUE, prints progress and statistics during the run. Default: FALSE.
seed	Optional non-negative integer seed for the C++ random-number generator. If NULL, the generator is initialized non-deterministically.
burn_in	Number of initial iterations excluded from all distributions, top solutions and the optional trace. Must be smaller than epochs. Default: 0.
store_trace	If TRUE, return the retained capped score, support and zero-based node vector for each iteration. Default FALSE avoids this storage.

Details

The stationary target on observed subsets of `cocktail_size` distinct nodes is proportional to $\exp(S(C)/T)$, where S is capped at `max_score`. Ancestor-descendant pairs are included in this MCMC space; genetic-algorithm validity filtering is a separate operation.

Type 1 proposes a uniform subset of distinct nodes. Type 2 replaces a node by a parent or child not already in the combination. Its Hastings correction is $|V(C)|/|V(C')|$, counting these possible local moves before rejecting proposals absent from the data. Each acceptance probability is the minimum of 1 and $\exp((S(C') - S(C))/T)$ times the applicable proposal correction. A positive `prob_type1` connects the observed state space.

At every retained iteration, including rejected moves, weights proportional to $\exp(-S(C)/T)$ recover the uniform combination reference. Weights are computed from exact capped scores before grouping into bins. They are rescaled internally to avoid numerical underflow. The filtered reference conditions on support strictly greater than `beta`, with its own ESS. This estimates a uniform law on combinations, not on distinct score values.

`uniform_reference` and `uniform_reference_filtered` each contain `distribution`, `n_samples`, and `weight_ess`. The distribution columns are `lower`, `upper`, `probability`, `cdf` and `upper_tail`. Bins are $[lower, upper)$, except the final singleton bin at the cap. CDF includes the current bin; `upper_tail` includes it and all subsequent bins. Empty references have NA probabilities and ESS. ESS is $(\sum w)^2 / \sum w^2$; it measures weight concentration and does not account for MCMC autocorrelation. Check mixing separately and choose an appropriate `burn_in`. See [uniform_score_reference](#) for post-processing an optionally stored trace without binning.

Value

A list containing:

top_solutions List of zero-based node vectors for the top scoring solutions

top_scores Numeric vector of scores for the top solutions

top_solutions_filtered Top solutions meeting the `beta` threshold

top_scores_filtered Scores for the filtered solutions

score_distribution Histogram of retained capped scores (0.1-wide bins and a cap bin)

score_distribution_filtered Histogram for solutions meeting `beta` threshold

outstanding_scores Retained capped scores equal to `max_score`

uniform_reference Uniform-reference distribution and weight ESS

uniform_reference_filtered Support-filtered uniform reference and ESS

reference_parameters Temperature, score cap, support and burn-in settings

trace Retained trajectory, only when `store_trace` is TRUE

statistics List of run statistics including acceptance rates

Examples

```
dat <- data.frame(outcome = c(1L, 1L, 0L, 0L))
dat$nodes <- list(c(0L, 1L), 0L, 1L, 1L)
tree <- data.frame(Depth = c(1L, 1L))
result <- run_mcmc_df_tree(dat, "nodes", "outcome", tree, "Depth",
```

```
epochs = 100, cocktail_size = 1, seed = 42)
result$uniform_reference$weight_ess
```

```
run_pwp_genetic_algorithm
```

Run the genetic algorithm with the PWP Rao fitness

Description

Run the genetic algorithm with the PWP Rao fitness

Usage

```
run_pwp_genetic_algorithm(
  data,
  context,
  tree,
  node_column = "icd10_indices",
  depth_column = "Depth",
  upper_bound_column = "UpperBound",
  name_column = "Code",
  seed_population = NULL,
  population_size = 100L,
  epochs = 1000L,
  mutation_rate = 0.1,
  probab_mutation_type1 = 0.2,
  crossover_rate = 0.8,
  elite_count = 2L,
  tournament_size = 3L,
  alpha = 1,
  diversity = FALSE,
  seed = 1L,
  verbose = FALSE
)
```

Arguments

data	Original interval-level data used to fit context.
context	A pwp_rao_context.
tree	Tree data frame.
node_column	List-column of 0-based tree indexes.
depth_column, upper_bound_column, name_column	Tree column names.
seed_population	Optional list of 1-based node vectors, following the historical treehunt seed convention.

population_size	Number of solutions per generation.
epochs	Number of generations.
mutation_rate	Probability of mutation.
prob_mutation_type1	Probability of the first mutation operator.
crossover_rate	Probability of crossover.
elite_count	Number of elite solutions copied to the next generation.
tournament_size	Number of solutions sampled per tournament.
alpha	Add/remove mutation bias. Higher values favor adding nodes and therefore tend to produce longer combinations.
diversity	Whether to apply the engine's diversity rule.
seed	Integer seed for the C++ random-number generator.
verbose	Whether to print progress.

Value

A named list containing:

- `final_population`: a list of integer vectors, each describing a final combination with zero-based tree indices.
- `final_scores`: a numeric vector of search fitness values in the same order. Eligible combinations use the signed Rao statistic, truncated at zero when `context` specifies `positive_only = TRUE`; combinations that fail the coverage or information requirements receive zero fitness.
- `parameters`: a list containing `population_size`, `epochs`, and `score_type` ("pwp_rao").
- `statistics`: a list containing `total_generations` and `cache_hits`.
- `pwp_context_statistics`: a list containing `event_time_groups` (the number of distinct event-time groups within strata) and `nuisance_covariates` (the number of adjustment covariates).

Use `score_pwp_combinations()` for detailed statistics and `decode_treehunt_solutions()` to convert node indices to labels.

run_pwp_mcmc

Run MCMC with the PWP Rao fitness

Description

Run MCMC with the PWP Rao fitness

Usage

```
run_pwp_mcmc(
  data,
  context,
  tree,
  node_column = "icd10_indices",
  depth_column = "Depth",
  upper_bound_column = "UpperBound",
  name_column = "Code",
  epochs = 100000L,
  temperature = 1,
  n_results = 20L,
  cocktail_size = 2L,
  prob_type1 = 0.01,
  beta = 20L,
  max_score = 50,
  seed = 1L,
  verbose = FALSE,
  burn_in = 0L,
  store_trace = FALSE
)
```

Arguments

<code>data</code>	Original interval-level data used to fit context.
<code>context</code>	A <code>pwp_rao_context</code> .
<code>tree</code>	Tree data frame.
<code>node_column</code>	List-column of 0-based tree indexes.
<code>depth_column, upper_bound_column, name_column</code>	Tree column names.
<code>epochs</code>	Number of MCMC iterations.
<code>temperature</code>	Metropolis-Hastings temperature.
<code>n_results</code>	Maximum number of retained top solutions.
<code>cocktail_size</code>	Fixed combination size.
<code>prob_type1</code>	Probability of the first proposal operator.
<code>beta</code>	Strict threshold: filtered output requires more than <code>beta</code> distinct covered patients.
<code>max_score</code>	Finite positive cap applied to scores before Metropolis-Hastings acceptance, importance weighting and binning. Default: 200.0.
<code>seed</code>	Integer seed for the C++ random-number generator.
<code>verbose</code>	Whether to print progress.
<code>burn_in</code>	Number of initial iterations excluded from all distributions, top solutions and the optional trace. Must be smaller than <code>epochs</code> . Default: 0.
<code>store_trace</code>	If TRUE, return the retained capped score, support and zero-based node vector for each iteration. Default FALSE avoids this storage.

Details

Uses the MCMC target, importance weights and output format described in [run_mcmc\(\)](#). The context must use `positive_only = TRUE` so that fitness is non-negative. The support filter counts distinct patients.

Value

See [run_mcmc\(\)](#) for the distributions, weight ESS and optional trace.

score_pwp_combinations

Evaluate full Rao statistics for combinations

Description

Evaluate full Rao statistics for combinations

Usage

```
score_pwp_combinations(
  combinations,
  data,
  context,
  tree,
  index_base = 0L,
  node_column = "icd10_indices",
  depth_column = "Depth",
  upper_bound_column = "UpperBound",
  name_column = "Code"
)
```

Arguments

<code>combinations</code>	List of node-index vectors.
<code>data</code>	Original interval-level data used to fit context.
<code>context</code>	A <code>pwp_rao_context</code> .
<code>tree</code>	Tree data frame.
<code>index_base</code>	Either 0 or 1.
<code>node_column</code>	List-column of 0-based tree indexes.
<code>depth_column, upper_bound_column, name_column</code>	Tree column names.

Value

A data frame with one row per combination, in input order, containing:

- `eligible`: logical indicator that coverage thresholds are met and the efficient information is finite and greater than the numerical tolerance.
- `fitness`: numeric search score, equal to `signed_z` or `max(0, signed_z)` according to context; zero for ineligible combinations.
- `signed_z`: signed Rao statistic, positive for an increased event hazard.
- `rao_chisq`: squared signed statistic, with one degree of freedom under the null model.
- `p_value_model_based`: the corresponding chi-squared upper-tail probability, using model-based information.
- `score_u`: score for the combination indicator at the reduced-model fit.
- `efficient_information`: information for that score after adjustment for the nuisance covariates.
- `covered_intervals`, `covered_patients`, `covered_events`: integer counts of covered intervals, distinct patients, and endpoint events.
- `combination`: a list-column retaining the supplied node vectors and their original index base.

Statistics are numeric. If there are no covered intervals or the efficient information is unusable, `signed_z`, `rao_chisq`, and `fitness` are zero and `p_value_model_based` is one. If only a coverage threshold fails, the full statistics are retained while `fitness` is zero.

`summarize_ga_pipeline` *Complete GA Results Analysis Pipeline (Generic)*

Description

Complete GA Results Analysis Pipeline (Generic)

Usage

```
summarize_ga_pipeline(folder_path, patient_data, tree_df, min_score = 0, ...)
```

Arguments

<code>folder_path</code>	Path to JSON results.
<code>patient_data</code>	The data frame containing patient observations.
<code>tree_df</code>	The tree structure data frame.
<code>min_score</code>	Minimum score threshold to keep a cocktail. Default 0.
<code>...</code>	Arguments passed down to <code>process_ga_scores</code> (and then to <code>compute_score</code>).

Value

A data frame with one row per distinct combination whose saved score is strictly greater than `min_score`. Columns include `cocktail` (comma-separated zero-based node indices), `cocktail_names`, `cocktail_codes`, `score` (the first saved score for that combination), `occurrence_count` (appearances across saved populations), and `found_in_configs` (configuration names separated by semicolons). For retained combinations, `process_ga_scores()` adds the recomputed scores, `taker_count`, and continuous-outcome summaries described there. If no combination passes the threshold, a warning is issued and the zero-row data frame is returned without these additional score columns.

`uniform_score_reference`

Recover the uniform score reference from a tempered MCMC trajectory

Description

Estimates the distribution of scores when combinations are sampled uniformly from the state space explored by a chain targeting $f_T(C) \propto \exp(S(C)/T)$. Scores are weighted before any aggregation.

Usage

```
uniform_score_reference(scores, temperature = 1, max_score = Inf)
```

Arguments

<code>scores</code>	Numeric vector of scores from successive retained MCMC states, including repeats after rejection. Remove burn-in before calling this function.
<code>temperature</code>	Finite, positive temperature used by the chain.
<code>max_score</code>	Upper score cap used by the chain. Use <code>Inf</code> only for a chain that used uncapped scores. For <code>run_mcmc()</code> output, scores in <code>trace</code> are already capped, so this argument may be omitted.

Details

With $s_i = \min(S_i, \text{max_score})$, the normalized weights are proportional to $\exp(-s_i/T)$. For a threshold x , the uniform upper-tail estimate is $\sum_i w_i 1(s_i \geq x) / \sum_i w_i$. This is a self-normalized importance-sampling estimate; its consistency requires an ergodic chain with the stated stationary distribution.

Uniformity refers to combinations, not to distinct score values. Retaining only accepted moves or unique combinations changes the sampling law and invalidates these weights. Legacy binned counts alone are insufficient for exact reweighting because scores within a bin can have different weights. The returned distribution concerns capped scores when a cap is used.

Filtering the retained trajectory by a support criterion estimates the uniform reference conditional on that criterion. Compute its ESS using only those retained iterations. See `weight_ess()` for the diagnostic's scope.

Value

A list with distribution (columns score, probability, cdf, and inclusive upper_tail), normalized per-iteration weights, n_samples, and weight_ess. Empty input returns an empty distribution and NA_real_ ESS.

See Also

[run_mcmc\(\)](#), [run_mcmc_df_tree\(\)](#), [run_pwp_mcmc\(\)](#)

Examples

```
# Two combinations with scores 0 and log(4) have target probabilities 1/5, 4/5.
ref <- uniform_score_reference(c(0, rep(log(4), 4)), temperature = 1)
ref$distribution # Equal estimated mass under the uniform combination law
ref$weight_ess
```

weight_ess	<i>Effective sample size based on importance weights</i>
------------	--

Description

Measures weight concentration using $(\sum_i w_i)^2 / \sum_i w_i^2$. Multiplying all weights by a common positive constant leaves this value unchanged. Computation is rescaled to avoid overflow and underflow.

Usage

```
weight_ess(weights, log = FALSE)
```

Arguments

weights	Non-negative finite weights, or log weights when log = TRUE. Log weights may include -Inf (zero weight).
log	Whether weights are on the natural-log scale.

Details

This diagnostic measures the concentration of importance weights. It does not account for serial dependence in an MCMC trajectory and is not a convergence diagnostic or an estimate of the number of independent draws.

Value

A numeric scalar between 1 and the number of supplied weights. Returns NA_real_ for an empty vector; all-zero weights are an error.

Examples

```
weight_ess(c(1, 1, 1))  
weight_ess(c(1, 0.25, 0.25, 0.25, 0.25))  
weight_ess(c(-1000, -1001), log = TRUE)
```

Index

aggregate_ga_results, [2](#)
aggregate_ga_results(), [16](#)

clustering_genetic_algorithm, [3](#), [12](#)
compute_score, [4](#)
compute_score(), [13](#), [14](#)

decode_treehunt_solutions, [5](#)
decode_treehunt_solutions(), [28](#)

filter_out_cocktails, [6](#)
fit_pwp_rao_context, [6](#)

get_dissimilarity_of_list, [8](#)
get_taker, [9](#)

map_cocktail_names, [10](#)
mcmc_size_2_true_score_distribution,
 [10](#)

plot_ga_clusters, [12](#)
print.pwp_rao_context
 (fit_pwp_rao_context), [6](#)
process_ga_scores, [13](#)
process_ga_scores(), [32](#)

refit_pwp_combinations, [14](#)
run_ga_batch, [15](#)
run_genetic_algorithm, [16](#)
run_genetic_algorithm_df_tree, [19](#)
run_mcmc, [18](#), [21](#)
run_mcmc(), [30](#), [33](#)
run_mcmc_df_tree, [24](#)
run_mcmc_df_tree(), [33](#)
run_pwp_genetic_algorithm, [27](#)
run_pwp_mcmc, [28](#)
run_pwp_mcmc(), [33](#)

score_pwp_combinations, [30](#)
score_pwp_combinations(), [28](#)
summarize_ga_pipeline, [31](#)

uniform_score_reference, [23](#), [26](#), [32](#)

weight_ess, [33](#)
weight_ess(), [32](#)