Package 'evoFE'

August 25, 2026

Type Package

Title Evolutionary Feature Engineering

Version 1.0.0

Description Automates feature engineering using evolutionary algorithms inspired by genetic programming. Starting from raw input features, the package evolves candidate transformation recipes through selection, crossover, and mutation, evaluating fitness via cross-validation or train/validation splits with gradient-boosted tree models ('LightGBM' or 'XGBoost'). Built-in transformers include arithmetic, logarithmic, and power operations, interaction terms, target encoding, quantile and log-based binning, principal component analysis, truncated singular value decomposition, Uniform Manifold Approximation and Projection (UMAP) dimensionality reduction, and minimum spanning tree (MST) graph-based clustering. The evolutionary search yields an optimised feature recipe that can be applied to new data for prediction. Methods are described in McInnes et al. (2018) <doi:10.21105/joss.00861>, Ke et al. (2017) <doi:10.48550/arXiv.1711.08789>, Chen and Guestrin (2016) <doi:10.1145/2939672.2939785>, Gagolewski (2021) <doi:10.1016/j.softx.2021.100722>, Gagolewski (2026) <doi:10.32614/CRAN.package.lumbermark>, and Gagolewski (2026) <doi:10.32614/CRAN.package.deadwood>.

URL https://github.com/tanopereira/evoFE

BugReports https://github.com/tanopereira/evoFE/issues

License MIT + file LICENSE

Encoding UTF-8

Imports data.table, lightgbm, xgboost, stats, digest, uwot, quitefastmst, genieclust, paradox, bbotk, mlr3mbo, lhs

Suggests glmnet, RhpcBLASctl, testthat, knitr, rmarkdown, lumbermark, deadwood, keras3, httpuv, jsonlite, farff, ranger, DiceKriging, rgenoud, mlr3learners

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

RoxygenNote 8.0.0

NeedsCompilation no

Author Gustavo Pereira [aut, cre]

Maintainer Gustavo Pereira <tanopereira@gmail.com>

Repository CRAN

Date/Publication 2026-08-25 12:30:02 UTC

Contents

`apply_gene` *Apply a single gene to a dataset*

Description

Apply a single gene to a dataset

Usage

```
apply_gene(
  gene,
  train_data,
  val_data = NULL,
  target_col = NULL,
  state_cache = NULL,
  data_hash = NULL
)
```

Arguments

`gene`	A gene list representing a feature transformation.
`train_data`	A data.frame or data.table representing the training data.
`val_data`	Optional validation data.frame or data.table.
`target_col`	Name of the target column.
`state_cache`	Optional environment to cache full-dataset fitted states of stateful transformers.
`data_hash`	Optional pre-computed xxhash64 digest of the target column, to avoid redundant hashing when applying multiple genes.

Value

A list with three elements: `train` (the modified training `data.table` with the new gene column appended), `val` (the modified validation `data.table` or `NULL`), and `gene` (the gene list, with its `state` element populated if the transformer is stateful).

`apply_individual` *Apply an entire individual's recipe to data*

Description

Apply an entire individual's recipe to data

Usage

```
apply_individual(
  ind,
  train_data,
  val_data = NULL,
  target_col = NULL,
  state_cache = NULL,
  allow_prune = TRUE
)
```

Arguments

`ind`	An evo_individual object.
`train_data`	A data.frame or data.table representing the training data.
`val_data`	Optional validation data.frame or data.table.
`target_col`	Name of the target column.
`state_cache`	Optional environment to cache full-dataset fitted states of stateful transformers.
`allow_prune`	Logical. If TRUE, genes that fail application are skipped instead of failing the entire individual.

Value

A list with three elements: `train` (the transformed training `data.table` with all gene columns applied), `val` (the transformed validation `data.table` or `NULL`), and `ind` (the updated `evo_individual` whose genes now carry fitted states).

`as.matrix.evo_topology` *Convert evo_topology to Adjacency Matrix*

Description

Convert evo_topology to Adjacency Matrix

Usage

```
## S3 method for class 'evo_topology'
as.matrix(x, ...)
```

Arguments

x	An evo_topology object.
...	Unused.

Value

N x N integer adjacency matrix.

---

compute_calibrated_mae *Compute Calibrated MAE*

Description

Computes the Mean Absolute Error (MAE) of y_pred after optimal L1 linear post-calibration. Finds intercept 'a' and slope 'b' minimizing Mean(|y_true - (a + b * y_pred)|) via 2D optimization.

Usage

```
compute_calibrated_mae(y_true, y_pred)
```

Arguments

y_true	Numeric vector of true target values.
y_pred	Numeric vector of predicted values.

Value

Numeric calibrated MAE score.

`compute_calibrated_rmse`

Compute Calibrated RMSE

Description

Computes the Root Mean Squared Error (RMSE) of y_pred after optimal linear post-calibration. Mathematically equivalent to SD(y_true) * sqrt(1 - R^2) where R is the Pearson correlation.

Usage

```
compute_calibrated_rmse(y_true, y_pred)
```

Arguments

`y_true`	Numeric vector of true target values.
`y_pred`	Numeric vector of predicted values.

Value

Numeric calibrated RMSE score.

`compute_complexity_penalty`

Compute Complexity Penalty for an Individual

Description

Computes the parsimony complexity penalty using Bayesian Information Criterion (BIC) scaling (`ln(N) / (2N)`) and optional dynamic convergence scaling based on the relative gap to the metric ceiling.

Usage

```
compute_complexity_penalty(
  n_genes,
  n_samples,
  running_best_fitness = NULL,
  baseline_fitness = NULL,
  metric = "default",
  task = "classification",
  complexity_penalty = 0,
  complexity_mode = "bic_dynamic",
  complexity_floor = 0.2,
  complexity_target = "all_features",
```

```
  n_features = NULL,
  epsilon_floor = 0.2
)
```

Arguments

n_genes	Integer. Number of evolved genes in the recipe.
n_samples	Integer. Number of dataset samples (rows).
running_best_fitness	Numeric. Current running best fitness in the population/island.
baseline_fitness	Numeric. Generation 0 baseline fitness (raw features only).
metric	Character or function. Metric being optimized.
task	Character. "classification", "multiclass", or "regression".
complexity_penalty	Numeric. Dimensionless penalty multiplier (default 0).
complexity_mode	Character. "bic_dynamic" (default), "bic", "pac_bayes_dynamic", "pac_bayes", or "none".
complexity_floor	Numeric. Minimum safety floor factor for dynamic penalties (default 0.20, representing 20% of base penalty).
complexity_target	Character. "all_features" (default, penalizes total active features) or "genes" (penalizes only derived genes).
n_features	Optional integer. Total number of active features (active raw features plus genes). If NULL, defaults to n_genes.
epsilon_floor	Deprecated alias for complexity_floor.

Value

Non-negative numeric penalty to subtract from raw fitness.

---

compute_ts_refinement *Temperature Scaled Refinement Metric*

Description

Computes the Temperature Scaled Refinement (TS-Refinement) metric for binary or multiclass classification. The metric finds the temperature T that minimizes the Laplace-smoothed log-loss of the temperature-scaled prediction margins (logits).

Usage

```
compute_ts_refinement(
  y_true,
  y_pred,
  task = "classification",
  num_class = NULL,
  alpha = 1,
  is_logits = FALSE
)
```

Arguments

y_true	Numeric vector of true labels (0/1 for classification, or 0 to C-1 for multiclass classification).
y_pred	Numeric vector or matrix of predicted probabilities (or logits, if is_logits = TRUE).
task	Character. Either "classification" (binary) or "multiclass".
num_class	Integer. Number of classes (required for multiclass).
alpha	Numeric. Laplace smoothing parameter (default is 1).
is_logits	Logical. If TRUE, the input predictions y_pred are treated directly as prediction margins (logits). If FALSE, they are treated as probabilities and converted to logits.

Value

Numeric. The minimized smoothed log-loss.

---

create_gene *Create a single gene*

Description

Create a single gene

Usage

```
create_gene(transformer_name, input_cols)
```

Arguments

transformer_name	Name of the transformer
input_cols	Vector of input column names

Value

A gene list with elements `transformer_name`, `input_cols`, `params` (transformer-specific parameters), `state` (`NULL` until fitted), and `output_col` (auto-generated column name).

`create_individual` *Create an individual*

Description

Create an individual

Usage

```
create_individual(
  genes = list(),
  numeric_cols = character(0),
  categorical_cols = character(0),
  datetime_cols = character(0),
  all_numeric_cols = NULL,
  all_categorical_cols = NULL,
  all_datetime_cols = NULL
)
```

Arguments

`genes`	List of genes
`numeric_cols`	Vector of active numeric column names visible to this individual.
`categorical_cols`	Vector of active categorical column names visible to this individual.
`datetime_cols`	Vector of active datetime column names visible to this individual.
`all_numeric_cols`	Vector of all numeric column names in the dataset (superset of `numeric_cols`). Used to initialize the full feature pool for mask toggling. Defaults to `numeric_cols` when `NULL`.
`all_categorical_cols`	Vector of all categorical column names in the dataset. Defaults to `categorical_cols` when `NULL`.
`all_datetime_cols`	Vector of all datetime column names in the dataset. Defaults to `datetime_cols` when `NULL`.

Value

An `evo_individual` S3 object: a list with elements `genes` (topologically sorted), `numeric_cols`, `categorical_cols`, and `fitness` (initialised to `NA_real_`).

Examples

```
ind <- create_individual(
  genes = list(),
  numeric_cols = c("a", "b"),
  categorical_cols = c("c")
)
print(ind)
```

`create_transformer` *Create a transformer definition*

Description

Create a transformer definition

Usage

```
create_transformer(
  name,
  type,
  input_type = "numeric",
  output_type = "numeric",
  fit_func = NULL,
  apply_func,
  name_generator,
  allow_replace = FALSE
)
```

Arguments

`name`	Transformer name
`type`	Type: "unary", "binary", "supervised_unary"
`input_type`	Type of input: "numeric" or "categorical"
`output_type`	Type of output: "numeric" or "categorical"
`fit_func`	function(data, input_cols, target_col = NULL) returning state
`apply_func`	function(data, input_cols, state = NULL) returning new column vector
`name_generator`	function(input_cols) returning output column name
`allow_replace`	Logical. Whether column sampling allows replacement.

Value

An `evo_transformer` S3 object: a list with elements `name`, `type`, `input_type`, `output_type`, `fit_func`, `apply_func`, `name_generator`, and `allow_replace`.

Examples

```
# Define a transformer that adds a constant value of 10 to a variable
add_ten_trans <- create_transformer(
  name = "add_ten",
  type = "unary",
  input_type = "numeric",
  apply_func = function(data, gene, state = NULL) {
    data[[gene$input_cols[1]]] + 10
  },
  name_generator = function(gene) paste0("add10_", gene$input_cols[1])
)
print(add_ten_trans)
```

---

crossover *Crossover two individuals*

Description

Crossover two individuals

Usage

```
crossover(ind1, ind2, verbose = FALSE)
```

Arguments

ind1	Parent 1
ind2	Parent 2
verbose	Logical. Whether to print crossover details.

Value

An `evo_individual` child created by randomly sampling genes from both parents with duplicate gene outputs removed.

Examples

```
ind1 <- create_individual(numeric_cols = c("a", "b"))
ind1 <- mutate(ind1, force_add = TRUE)
ind2 <- create_individual(numeric_cols = c("a", "b"))
ind2 <- mutate(ind2, force_add = TRUE)
child <- crossover(ind1, ind2)
```

`ensemble_islands` *Caruana Island Ensemble Selection*

Description

Performs Caruana ensemble selection (greedy forward selection with replacement) over the validation/out-of-fold predictions from evolved islands, creating an optimal multi-model ensemble.

Usage

```
ensemble_islands(
  recipe,
  data,
  target_col = NULL,
  caruana_rounds = 50,
  bag_samples = TRUE,
  sample_ratio = 0.8,
  seed = NULL,
  threads = 2,
  verbose = TRUE,
  ...
)
```

Arguments

`recipe`	An `evo_recipe` object produced by `evolve_features`.
`data`	A data.frame or data.table containing the original training data used during evolution. Required for lazy final model training of surviving islands.
`target_col`	Character string. Name of the target column. If `NULL`, it is inferred from the recipe or data.
`caruana_rounds`	Positive integer. Number of greedy selection rounds (default: 50).
`bag_samples`	Logical. If `TRUE` (default), uses bootstrap sampling of validation predictions during selection rounds to prevent validation set overfitting.
`sample_ratio`	Numeric between 0 and 1. Fraction of validation samples used when `bag_samples = TRUE` (default: 0.8).
`seed`	Optional integer seed for reproducible bagged sampling. Does not mutate the user's global RNG state.
`threads`	Integer. Number of threads to use for model training.
`verbose`	Logical. Whether to print progress messages.
`...`	Additional arguments passed to `train_model`.

Value

An `evo_ensemble` object containing:

`active_recipes` Named list of feature engineering recipes for surviving islands.

`active_models` Named list of trained models for surviving islands.

`weights` Named numeric vector of Caruana ensemble weights (summing to 1).

`caruana_history`
Data frame of validation loss trajectory across selection rounds.

`single_best_fitness`
Fitness of the single best island model.

`ensemble_val_fitness`
Validation fitness achieved by the Caruana ensemble.

`task` The learning task ("classification", "regression", or "multiclass").

`evaluator` The evaluator model engine used.

`classes` Target class levels (for multiclass classification).

`metric` Evaluation metric used.

Examples

```
data(mtcars)
df <- mtcars
df$am <- as.integer(df$am)

# Evolve features across 3 islands
recipe <- evolve_features(
  data = df,
  target_col = "am",
  task = "classification",
  evaluator = "xgboost",
  generations = 2,
  pop_size = 2,
  islands = 3,
  cv_folds = 2,
  verbose = FALSE
)

# Build Caruana ensemble from island predictions
ens <- ensemble_islands(recipe, data = df, caruana_rounds = 20, verbose = FALSE)
print(ens)
```

`evaluate_fitness` *Evaluate the fitness of an individual*

Description

Trains a model using the features specified by the individual's recipe and evaluates performance using cross-validation or train/val split.

Usage

```
evaluate_fitness(
  ind,
  data,
  target_col,
  task = "classification",
  cv_folds = 3,
  evaluation_strategy = "cv",
  split_ids = NULL,
  shared_splits = NULL,
  evaluator = "lightgbm",
  fold_ids = NULL,
  shared_folds = NULL,
  shared_full = NULL,
  state_cache = NULL,
  threads = 2,
  metric = "default",
  verbose = FALSE,
  allow_prune = TRUE,
  complexity_penalty = 0,
  complexity_mode = "bic_dynamic",
  complexity_floor = 0.2,
  complexity_target = "all_features",
  running_best_fitness = NULL,
  baseline_fitness = NULL,
  n_samples = NULL,
  cv_strategy = "random",
  time_col = NULL,
  group_col = NULL,
  ...
)
```

Arguments

`ind`	An evo_individual object.
`data`	A data.frame or data.table containing the dataset.
`target_col`	Name of the target column.

task	"classification" or "regression".
cv_folds	Number of cross-validation folds.
evaluation_strategy	Character string, either "cv" (cross-validation) or "split" (train/validation split).
split_ids	Optional vector of pre-defined split assignments (e.g. `c("train", "train", "val", "holdout", "train")`). Must have the same length as the number of rows in `data` and contain only "train", "val", or "holdout" labels.
shared_splits	Optional list of shared data.table splits for in-place caching.
evaluator	Character string specifying the model backend: "lightgbm", "xgboost", "catboost", "rf", or "lm".
fold_ids	Optional integer vector of pre-assigned fold indices.
shared_folds	Optional list of shared data.table fold splits for in-place caching.
shared_full	Optional shared full data.table.
state_cache	Optional environment used to cache transformer training states across evaluations.
threads	Number of threads for model training.
metric	Character string or evaluation metric.
verbose	Logical.
allow_prune	Logical.
complexity_penalty	Numeric. Dimensionless penalty multiplier (default 0).
complexity_mode	Character. Complexity penalty strategy: "bic_dynamic" (default), "bic", "pac_bayes_dynamic", "pac_bayes", or "none".
complexity_floor	Numeric in `[0, 1]`. Minimum safety floor factor for dynamic penalty (default `0.20`).
complexity_target	Character. "all_features" (default, penalizes total active features) or "genes" (penalizes only derived genes).
running_best_fitness	Optional numeric. Current running best fitness for dynamic BIC.
baseline_fitness	Optional numeric. Generation 0 baseline fitness for dynamic BIC.
n_samples	Optional integer. Dataset sample size N for BIC calculations. Defaults to nrow(data).
cv_strategy	Fold construction strategy for CV: `"random"` (default), `"time"`, or `"group"`.
time_col	Column name used when `cv_strategy = "time"`.
group_col	Column name used when `cv_strategy = "group"`.
...	Additional arguments passed to the underlying evaluator training functions.

Value

The input `evo_individual` with its `fitness` field set to the computed score (higher is better), `importances` set to a named numeric vector of feature importances, `holdout_fitness` set to `NULL`, and `genes` updated with fitted transformer states.

`evolve_features` *Run evolutionary feature engineering*

Description

Run evolutionary feature engineering

Usage

```
evolve_features(
  data,
  target_col,
  task = "classification",
  generations = 10,
  pop_size = 10,
  cv_folds = 3,
  evaluation_strategy = "cv",
  split_ratio = c(0.6, 0.2, 0.2),
  split_ids = NULL,
  holdout_frac = 0,
  cv_strategy = "random",
  time_col = NULL,
  group_col = NULL,
  multi_fidelity = FALSE,
  mf_sample_frac = 0.5,
  mf_warmup_frac = 0.5,
  early_stopping_generations = 3,
  evaluator = "lightgbm",
  seed = NULL,
  dynamic_population = TRUE,
  dynamic_population_growth_rate = 1.5,
  dynamic_population_decay_rate = 0.7,
  crossover_type = "both",
  threads = 2,
  max_clustering_size = 5000,
  verbose = TRUE,
  metric = "default",
  model_all_final_genes = FALSE,
  model_all_historical_genes = FALSE,
  allowed_transformers = "all",
  complexity_penalty = 0,
```

```
  complexity_mode = "bic_dynamic",
  complexity_floor = 0.2,
  complexity_target = "all_features",
  migration = NULL,
  islands = 1,
  migration_interval = 5,
  migration_rate = 1,
  gene_migration_prob = 0.2,
  migration_topology = "ring",
  migration_temperature = 1,
  pull_stagnation_threshold = 3,
  raw_toggle_prob = 0.15,
  recalculate_mask_prob = 0.05,
  mask_temp_factor = 0.5,
  row_split_islands = FALSE,
  per_island_validation = FALSE,
  record = FALSE,
  port = NULL,
  ...
)
```

Arguments

`data`	A data.frame or data.table
`target_col`	Name of the target column
`task`	"classification" or "regression"
`generations`	Number of generations (max iterations)
`pop_size`	Population size
`cv_folds`	Number of cross-validation folds
`evaluation_strategy`	"cv" or "split". Strategy to evaluate candidate recipes.
`split_ratio`	A numeric vector of length 2 or 3 defining train/validation/holdout proportions (e.g. c(0.6, 0.2, 0.2)).
`split_ids`	An optional character vector of split assignments (e.g. `c("train", "train", "val", "holdout", "train")`). Must have the same length as the number of rows in `data` and contain only "train", "val", or "holdout" labels (with at least "train" and "val" present). When provided, `evaluation_strategy` is automatically set to "split" and the actual split proportions are computed from the vector.
`holdout_frac`	Numeric in `[0, 0.5)`. When greater than 0 with `evaluation_strategy = "cv"`, this fraction of rows is stratified and held out of the entire evolutionary search. After evolution, the winning recipe (with its frozen transformer states) and the final model are scored once on these never-seen rows; the result is exposed as `holdout_fitness` / `search_gap` on the returned object — an unbiased estimate of generalization that reveals how much the search overfit its selection folds.
`cv_strategy`	Fold construction strategy for CV: `"random"` (default, rows shuffled into folds), `"time"` (rows ordered by `time_col` and split into contiguous chronological

blocks so validation always lies in the future of training), or "group" (all rows sharing a group_col value land in the same fold). Use "time" for temporal data and "group" for clustered data to avoid leakage.

time_col Column name used when cv_strategy = "time". Must be datetime or numeric.

group_col Column name used when cv_strategy = "group".

multi_fidelity Logical (default FALSE). If TRUE, individuals during warm-up generations are first screened on row-subsampled folds (mf_sample_frac); the most promising half is then re-evaluated at full fidelity before any selection decision, so all fitness comparisons remain apples-to-apples. Reduces compute cost with minimal search-quality loss.

mf_sample_frac Row fraction kept per fold during multi-fidelity screening, in (0, 1).

mf_warmup_frac Fraction of generations (of generations) run in low-fidelity screening mode before full-fidelity-only evaluation begins.

early_stopping_generations
Stop if fitness doesn't improve for this many generations

evaluator The ML model to use ("lightgbm", "xgboost", "catboost", or a custom registered evaluator name).

seed Optional integer. Seeds the entire stochastic pipeline (fold construction, hold-out split, population initialization, mutation and crossover) without touching the caller's .Random.seed: the user's RNG state is saved on entry and restored on exit (CRAN-safe). Island j derives its initial population from seed + 1000*j, so island identities are stable regardless of island count. The seed is also forwarded to the final model fit (and to evaluators that accept one). Multi-threaded LightGBM/XGBoost remain only statistically reproducible due to floating-point reduction order; use threads = 1 for bitwise identical reruns.

dynamic_population
Logical. If TRUE, population expands dynamically during stagnation.

dynamic_population_growth_rate
Growth rate multiplier for population expansion during stagnation (default 1.5).

dynamic_population_decay_rate
Decay rate multiplier for population contraction back to baseline (default 0.7).

crossover_type Crossover type: "both" (default, 50% random / 50% union), "random", or "union"

threads Number of threads to use for parallel execution (default 2)

max_clustering_size
Maximum unique training rows to cluster (default 5000, 0/NULL for unlimited)

verbose Logical. If TRUE, prints progress.

metric The metric to optimize ("default", "auc", "f1", "mae", "cal_rmse", "cal_mae", or a custom function).

model_all_final_genes
Logical. If TRUE, the final model is trained using the union of all unique genes evolved in the final population, rather than only the best individual's genes.

model_all_historical_genes
Logical. If TRUE, the final model is trained using the union of all unique genes evolved across all generations, rather than only the best individual's genes.

allowed_transformers
: Character vector of allowed transformer names, or "all" / "basic" / "robust" / "clustering".

complexity_penalty
: Non-negative numeric multiplier for complexity penalty (default 0). When set to 1.0, applies standard BIC or PAC-Bayes parsimony pressure. A value of 0 disables complexity penalisation.

complexity_mode
: Character string specifying the complexity penalty strategy: "bic_dynamic" (default, asymptotic BIC scaling ln(N) / (2N) dynamically relaxed with progress), "bic" (constant asymptotic BIC penalty ln(N) / (2N) throughout evolution), "pac_bayes_dynamic" (PAC-Bayes generalization bound scaling 1 / (2*sqrt(N)) dynamically relaxed with progress), "pac_bayes" (constant PAC-Bayes generalization bound scaling 1 / (2*sqrt(N))), or "none" (disabled).

complexity_floor
: Numeric in [0, 1]. Minimum safety floor factor for dynamic penalties (default 0.20, representing a 20% minimum floor of base penalty).

complexity_target
: Character string specifying the complexity count target: "all_features" (default, penalizes total number of active features including raw features and genes, rewarding active feature pruning) or "genes" (penalizes only derived genes).

migration
: Optional evo_migration_config object created by migration_config().

islands
: Integer. Number of islands for multi-island parallel evolution (default 1).

migration_interval
: Integer. Number of generations between migrations (default 5).

migration_rate
: Integer. Number of top individuals to migrate from each island to its neighbor (default 1).

gene_migration_prob
: Numeric. Probability of injecting a migrated gene during mutation (default 0.2).

migration_topology
: Character string specifying the island migration scheme: "ring" (default unidirectional ring), "gibbs_stagnation" (probabilistic push targeting stagnated islands), "gibbs_fitness" (probabilistic push targeting lower-fitness islands), "dual_gibbs_pull" (demand-driven pull where stagnated islands request migrants from high-fitness donors), or "random" (uniform random destination).

migration_temperature
: Numeric > 0. Temperature parameter for Gibbs softmax migration probability distributions (default 1.0).

pull_stagnation_threshold
: Integer >= 1. Stagnation generation threshold used as sigmoid midpoint for pull requests in "dual_gibbs_pull" (default 3).

raw_toggle_prob
: Numeric in [0, 1]. Probability that a mutation event toggles one or more raw input features in an individual's active mask rather than adding/modifying/removing a gene. A dynamic geometric distribution determines how many features are toggled per event. Default 0.15.

recalculate_mask_prob
: Numeric in [0, 1]. Probability that a mutation event completely recalculates the individual's active raw feature mask from scratch using feature importances and a sigmoid inclusion probability. Default 0.05.

mask_temp_factor
: Numeric > 0. Temperature scaling factor applied to feature importances during active mask initialization and recalculation. Higher values flatten the importance distribution (more uniform sampling); lower values concentrate sampling on the highest-importance features. Default 0.5.

row_split_islands
: Logical. If TRUE, splits data rows across islands (default FALSE).

per_island_validation
: Logical. If TRUE, evaluates candidate recipes using each island's specific row split (default FALSE).

record
: Logical. If TRUE, records detailed evolutionary logs and launches the interactive evolution live viewer (default FALSE).

port
: Optional port number for the live viewer server. If NULL, a random free port is used (or retrieves from the global option 'evoFE.viewer_port').

...
: Additional arguments passed to the underlying evaluator training functions.

Value

An evo_recipe S3 object: a list with elements best_individual (the top-scoring evo_individual), history (list of all evaluated individuals across generations), task, best_model (the trained model object), evaluator, and classes (class levels for multiclass tasks, otherwise NULL).

Examples

```
# Quick classification example using mtcars
data(mtcars)
df <- mtcars
df$am <- as.integer(df$am)

set.seed(42)
recipe <- evolve_features(
  data = df,
  target_col = "am",
  task = "classification",
  evaluator = "xgboost",
  generations = 2,
  pop_size = 2,
  cv_folds = 2,
  verbose = FALSE
)
print(recipe)
```

evo_evaluators *Global environment for registered model evaluators*

Description

Global environment for registered model evaluators

Usage

```
evo_evaluators
```

Value

An `environment` containing registered model evaluators.

evo_transformers *Built-in feature transformers*

Description

An environment containing all built-in transformer definitions available for the evolutionary feature-engineering search. Every entry is an `evo_transformer` object produced by `create_transformer`.

Usage

```
evo_transformers
```

Details

Arithmetic (numeric -> numeric)

`log` Safe natural logarithm: `log1p(|x|)`.

`sqrt` Safe square root: `sqrt(|x|)`.

`reciprocal` Reciprocal: `1/x` (0 where `x == 0`).

`power` Signed exponentiation: `sign(x) * |x|^p` where `p` is sampled from {0.5, 1/3, 2, 3}.

`displaced_log` Displaced log: `log1p(|x + displacement|)` where `displacement` is sampled from `[10, 1000]`.

`add` Element-wise sum of 2+ numeric columns.

`subtract` Element-wise difference of two numeric columns.

`multiply` Element-wise product of 2+ numeric columns.

`divide` Element-wise ratio (0 where denominator is 0).

`normalized_difference` `(a - b) / (|a| + |b| + 1e-6)`.

`log_ratio` `log1p(|a|) - log1p(|b|)`.

Rank / distribution (numeric -> numeric, stateful)

`rank_transform` ECDF-based percentile rank mapped to $[0, 1]$. Fit on training data; robust to outliers.

Group-by aggregations (mixed cat x num -> numeric, stateful)

`groupby_mean` Per-group mean.

`groupby_sd` Per-group standard deviation.

`groupby_max` Per-group maximum.

`groupby_min` Per-group minimum.

`groupby_ratio` `value / group_mean`.

`groupby_zscore` `(value - group_mean) / group_sd`.

`groupby_median` Per-group median (robust to outliers).

`groupby_quantile` Per-group Q1 or Q3 (`q` sampled from $\{0.25, 0.75\}$).

Supervised categorical encodings (categorical -> numeric, stateful)

`target_encode` Smoothed mean-target encoding for binary / regression tasks.

`pooled_target_encode` Empirical Bayes pooled target encoding for binary / regression tasks using dynamic shrinkage based on target variance.

`target_encode_multiclass` Class-wise smoothed target encoding for multiclass tasks.

`target_quantile_encode` Category target encoding using smoothed target quantiles (`q` sampled from $\{0.25, 0.50, 0.75\}$).

`cat_interaction_target_encode` Smoothed mean-target encoding for joint Cartesian interaction of two categorical columns.

`woe_encode` Weight of Evidence encoding for binary classification: `ln(P(event|cat) / P(non-event|cat))` with Laplace smoothing. Falls back to 0 for non-binary targets.

Unsupervised categorical / text / datetime

`concat` Concatenates 2 or 3 categorical columns row-wise using an underscore separator.

`frequency_encode` Count of each category level in training data.

`one_hot_encode` Binary indicator for up to 5 top categories plus an "other" bucket (`comp_idx` 1-6).

`similarity_encode` Character 3-gram Jaccard similarity between string levels and top-K prototype categories (inspired by **skrub**).

`minhash_encode` Fast sub-string MinHash hashing for high-cardinality strings (inspired by **skrub**).

`gap_encode` Character 3-gram TF-IDF projection via SVD to extract latent sub-string topics (inspired by **skrub**).

`quantile_binning` Assigns quantile-based bin index (numeric output).

`quantile_binning_cat` Same, with categorical output.

`log_binning` Log-scale bin index (numeric output).

`log_binning_cat` Same, with categorical output.

`datetime_extract` Extracts year, month, day, hour, day-of-week, or weekend indicator from date/datetime columns.

`datetime_cyclic` Sine and cosine cyclic encoding for periodic date/time components (hour, day of week, month, day of year).

`date_diff` Signed difference in days between two datetime columns.

Dimensionality reduction (numeric -> numeric, stateful)

`pca` Selected principal component from `prcomp`.

`truncated_svd` Selected component from truncated SVD.

`random_projection` Random unit-vector linear combination.

`umap` UMAP projection component (requires **uwot**).

Manifold / graph learning (numeric -> numeric or categorical, stateful)

`genie` Genie hierarchical cluster label (requires **genieclust**).

`genie_centroid_dist` Distance to each Genie cluster centroid.

`umap_genie` Genie cluster label computed on low-dimensional UMAP embedding (requires **uwot** and **genieclust**).

`umap_lumbermark` Lumbermark cluster label computed on a low-dimensional UMAP embedding. Combines the non-linear structure discovery of UMAP with the minimum-spanning-tree-based hierarchical clustering of Lumbermark (requires **uwot** and **lumbermark**).

`lumbermark` Lumbermark hierarchical cluster label (requires **lumbermark**).

`lumbermark_centroid_dist` Distance to each Lumbermark cluster centroid.

`mst_score` MST-based anomaly score (requires **quitefastmst**).

`deadwood` Deadwood outlier indicator (requires **deadwood**).

Value

An `environment` containing registered feature transformers.

See Also

`create_transformer`, `register_transformer`

`gene_to_formula` *Convert a gene to a formula string*

Description

Convert a gene to a formula string

Usage

```
gene_to_formula(gene, truncate = TRUE)
```

Arguments

gene	A gene list
truncate	Logical. If TRUE (default), long list of input columns is truncated for display.

Value

A character string representing the gene as a human-readable formula, e.g. "log(col1)" or "pca2(col1, col2)".

---

gene_to_state_formula *Convert a gene to a formula string for state caching (ignoring component index)*

Description

Convert a gene to a formula string for state caching (ignoring component index)

Usage

```
gene_to_state_formula(gene)
```

Arguments

gene	A gene list

Value

A character string representing the gene formula suitable for state caching. For multi-component transformers (PCA, SVD, UMAP) the component index is omitted so that all components share one cache key.

---

individual_to_recipe_string *Convert an individual to a recipe string of formulas*

Description

Convert an individual to a recipe string of formulas

Usage

```
individual_to_recipe_string(ind)
```

Arguments

ind	An evo_individual

Value

A character string listing all gene formulas in bracket notation, e.g. `"[log(x), sqrt(y)]"`, or `"[Original features only]"` when the individual has no genes.

`initialize_population` *Initialize a population*

Description

Initialize a population

Usage

```
initialize_population(
  pop_size,
  numeric_cols,
  categorical_cols,
  datetime_cols = character(0),
  initial_genes = 2,
  task = "classification",
  importances = NULL,
  allowed_transformers = NULL,
  mask_temp_factor = 0.5
)
```

Arguments

`pop_size`	Population size.
`numeric_cols`	Vector of numeric column names.
`categorical_cols`	Vector of categorical column names.
`datetime_cols`	Vector of datetime column names.
`initial_genes`	Number of initial genes per individual.
`task`	Task type ("classification", "regression", or "multiclass").
`importances`	Optional numeric vector of feature importances.
`allowed_transformers`	A character vector of allowed transformer names, or NULL/"all" to allow all.
`mask_temp_factor`	Numeric > 0. Temperature scaling factor applied to feature importances during active mask initialization. Higher values flatten the importance distribution (more uniform sampling); lower values concentrate sampling on the highest-importance features. Default `0.5`.

Value

A list of `evo_individual` objects of length `pop_size`. The first individual is a baseline with no genes; the remaining individuals each carry `initial_genes` randomly generated genes.

`make_tunable` *Create a Tunable Evaluator from a Registered Base Model*

Description

Wraps an existing registered model evaluator in a Bayesian Optimization tuning loop using the **mlr3mbo** framework. It automatically generates a parameter space, constructs a cross-validation or split-validation objective function, searches for the optimal hyperparameters, and registers the tuned evaluator.

Usage

```
make_tunable(
  base_model_name,
  param_ranges,
  tuner_name = paste0(base_model_name, "_mbo")
)
```

Arguments

`base_model_name`
: Character. Name of the registered base evaluator (e.g., `"xgboost"`, `"lightgbm"`).

`param_ranges`
: List. A nested list defining the parameter names, types, and bounds/values. Each parameter definition must be a list containing:

  `type` Character: `"numeric"`, `"integer"`, or `"discrete"`.

  `lower` Numeric/Integer: Lower bound of the search space (required for `"numeric"` and `"integer"`).

  `upper` Numeric/Integer: Upper bound of the search space (required for `"numeric"` and `"integer"`).

  `values` Vector: Set of valid values (required for `"discrete"`).

`tuner_name`
: Character. The name under which to register the tuned evaluator. Defaults to `paste0(base_model_name, "_mbo")`.

Details

The tuning loop uses a Latin Hypercube Design (LHS) for the initial parameters layout. It uses the **mlr3mbo** package to run Bayesian Optimization to optimize hyperparameters.

Evaluators registered via `make_tunable` accept several control parameters passed via `...`:

`mbo_iters` Integer: Number of Bayesian Optimization iterations (default 5).

`mbo_init_design` Integer: Number of initial layout designs generated (default 8).

`mbo_folds` Integer: Number of internal CV folds used for evaluation when no validation split is provided (default 3).

`mbo_infill_opt` Character: Strategy for infill optimization to find the next candidate parameter set. Supported values are `"focussearch"` (default) and `"ea"` (deprecated).

`best_params` List: Optional list of initial parameters to seed the MBO search.

Value

Invisibly returns NULL. Registers the tuned evaluator in the global evo_evaluators environment.

Examples

```
## Not run:
# 1. Register a simple mock evaluator
register_evaluator(
  "mock_base",
  train_func = function(x_train, y_train, x_val = NULL, y_val = NULL,
                        task = "regression", ...) {
    args <- list(...)
    val_score <- 100 - abs(args$param_a - 4.5)
    list(
      model = list(args = args, val_score = val_score),
      predictions = if (!is.null(x_val)) {
        rep(val_score, nrow(x_val))
      } else {
        NULL
      }
    )
  },
  predict_func = function(model, x_new, task, ...) {
    rep(model$val_score, nrow(x_new))
  }
)

# 2. Make it tunable
param_ranges <- list(
  param_a = list(type = "numeric", lower = 1.0, upper = 8.0)
)
make_tunable("mock_base", param_ranges, tuner_name = "mock_tuned")

# 3. Train the tuned model on mock data
x_train <- matrix(rnorm(20), ncol = 2)
colnames(x_train) <- c("x1", "x2")
y_train <- rnorm(10)
x_val <- matrix(rnorm(10), ncol = 2)
y_val <- rnorm(5)

fit <- train_model(
  x_train, y_train, x_val = x_val, y_val = y_val,
  task = "regression", evaluator = "mock_tuned",
  mbo_iters = 3, mbo_init_design = 5, mbo_folds = 2
)
print(fit$best_params)

## End(Not run)
```

migration_policy *Migration Policy Constructors and Generics for Island Models*

Description

Functions to specify and resolve migration dynamics, target selection, and admission rules.

Usage

```
policy_push_uniform()

policy_gibbs_push(
  temperature = 0.5,
  weight_by = c("stagnation", "fitness", "feature_distance", "uniform")
)

policy_gibbs_pull(
  temperature = 0.5,
  stagnation_threshold = 3,
  weight_by = c("fitness", "feature_distance", "uniform")
)

policy_tiered_admission(min_fitness_threshold = "min_peer")

migration_config(
  topology = topology_ring(),
  policy = policy_push_uniform(),
  payload = "gene_only"
)

resolve_migration_transactions(policy, topology, state, ...)

## S3 method for class 'evo_policy_push_uniform'
resolve_migration_transactions(policy, topology, state, ...)

## S3 method for class 'evo_policy_gibbs_push'
resolve_migration_transactions(policy, topology, state, ...)

## S3 method for class 'evo_policy_gibbs_pull'
resolve_migration_transactions(policy, topology, state, ...)

## S3 method for class 'evo_policy_tiered_admission'
resolve_migration_transactions(policy, topology, state, ...)
```

Arguments

temperature	Numeric. Softmax temperature for Gibbs routing (default: 0.5).

weight_by	Character. Criterion for Gibbs push weighting ("stagnation" or "fitness").
stagnation_threshold	Integer. Stagnation generation threshold for Gibbs pull (default: 3).
min_fitness_threshold	Character. Tier admission rule (default: "min_peer").
topology	An evo_topology object or string name.
policy	An evo_policy object or string name.
payload	Character. Payload strategy: "gene_only" (default) or "full_individual".
state	List containing current evolution state (pop_list, island_best_fitness, island_gens_without_improvement).
...	Additional arguments.

Value

An object of class evo_policy or evo_migration_config, or a transaction list.

---

mutate *Mutate an individual*

Description

Mutate an individual

Usage

```
mutate(
  ind,
  verbose = FALSE,
  force_add = FALSE,
  importances = numeric(0),
  temperature = 1,
  task = "classification",
  tested_gene_outputs = NULL,
  allowed_transformers = NULL,
  migrated_genes = list(),
  gene_migration_prob = 0.2,
  raw_toggle_prob = 0.15,
  recalculate_mask_prob = 0.05
)
```

Arguments

ind	An evo_individual.
verbose	Logical. Whether to print mutation details.
force_add	Logical. If TRUE, forces adding a new gene.
importances	A numeric vector of feature importances.

`temperature` A numeric temperature value controlling selection weights.

`task` The task type ("classification", "regression", or "multiclass")

`tested_gene_outputs`
Character vector of gene output names that have been evaluated in a previous generation and are safe for chaining. When NULL (default), all existing gene outputs are available. Pass character(0) to block all chaining (e.g. during initialization).

`allowed_transformers`
A character vector of allowed transformer names, or NULL/"all" to allow all.

`migrated_genes` A list of genes migrated from other islands.

`gene_migration_prob`
Probability of selecting a migrated gene during mutation.

`raw_toggle_prob`
Numeric in `[0, 1]`. Probability that a mutation event toggles one or more raw input features in the individual's active mask rather than adding/modifying/removing a gene. Default `0.15`.

`recalculate_mask_prob`
Numeric in `[0, 1]`. Probability that a mutation event completely recalculates the active raw feature mask from scratch using feature importances. Default `0.05`.

Value

An `evo_individual` with the mutation applied (gene added, removed, or modified) and `fitness` reset to `NA_real_`.

Examples

```
ind <- create_individual(
  numeric_cols = c("a", "b"),
  categorical_cols = c("c")
)
mutated_ind <- mutate(ind)
```

---

plot.evo_recipe *Plot an evo_recipe object*

Description

Plots either the fitness trajectory over generations or the feature importances of the best individual.

Usage

```
## S3 method for class 'evo_recipe'
plot(x, type = "fitness", ...)
```

Arguments

x	An evo_recipe object.
type	Character string, either "fitness" (default) to plot the fitness trajectory, or "importance" to plot a bar chart of the top feature importances of the winning model.
...	Additional arguments passed to plot or barplot.

Value

Invisible NULL. Called for its side effect of plotting the fitness curve or feature importances.

Examples

```
data(mtcars)
df <- mtcars
df$am <- as.integer(df$am)

recipe <- evolve_features(
  data = df,
  target_col = "am",
  task = "classification",
  evaluator = "xgboost",
  generations = 2,
  pop_size = 2,
  cv_folds = 2,
  seed = 42,
  verbose = FALSE
)

# Plot the fitness curve
plot(recipe, type = "fitness")

# Plot feature importances
plot(recipe, type = "importance")
```

---

predict.evo_ensemble *Apply engineered features from an ensemble of recipes*

Description

Apply engineered features from an ensemble of recipes

Usage

```
## S3 method for class 'evo_ensemble'
predict(object, newdata, ...)
```

Arguments

object	An evo_ensemble object
newdata	A data.frame or data.table
...	Additional arguments

Value

A data.table containing the combined engineered features across all active recipes in the ensemble.

---

predict.evo_recipe *Apply feature engineering recipe to new data*

Description

Apply feature engineering recipe to new data

Usage

```
## S3 method for class 'evo_recipe'
predict(object, newdata, ...)
```

Arguments

object	An evo_recipe object
newdata	A data.frame or data.table
...	Additional arguments

Value

A `data.table` containing the engineered feature columns (original plus all gene-derived columns) for `newdata`, ready for downstream modelling.

Examples

```
data(mtcars)
df <- mtcars
df$am <- as.integer(df$am)

recipe <- evolve_features(
  data = df,
  target_col = "am",
  task = "classification",
  evaluator = "xgboost",
  generations = 2,
  pop_size = 2,
  cv_folds = 2,
  seed = 42,
```

```
  verbose = FALSE
)

# Extract engineered features
engineered_features <- predict(recipe, df[1:5, ])
print(engineered_features)
```

`predict_model` *Predict target values using the fully evolved model or ensemble*

Description

Predict target values using the fully evolved model or ensemble

Usage

```
predict_model(object, newdata, ...)

## S3 method for class 'evo_recipe'
predict_model(object, newdata, ...)

## S3 method for class 'evo_ensemble'
predict_model(object, newdata, ...)
```

Arguments

`object`	An evo_recipe or evo_ensemble object containing trained model(s)
`newdata`	A data.frame or data.table to make predictions on
`...`	Additional arguments (currently unused)

Value

For binary classification and regression tasks a numeric vector of predictions. For multiclass tasks a numeric matrix with one column per class (columns named after class levels).

Examples

```
data(mtcars)
df <- mtcars
df$am <- as.integer(df$am)

recipe <- evolve_features(
  data = df,
  target_col = "am",
  task = "classification",
  evaluator = "xgboost",
  generations = 2,
```

```
  pop_size = 2,
  cv_folds = 2,
  seed = 42,
  verbose = FALSE
)

# Get model predictions
predictions <- predict_model(recipe, df[1:5, ])
print(predictions)
```

`print.evo_ensemble` *Print an evo_ensemble object*

Description

Prints a human-readable summary of the Caruana island ensemble.

Usage

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

Arguments

`x`	An `evo_ensemble` object.
`...`	Additional arguments (currently unused).

Value

Invisible `x`. Called for its side effect of printing the ensemble overview.

`print.evo_recipe` *Print an evo_recipe object*

Description

Prints a human-readable summary of the evolutionary feature engineering recipe.

Usage

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

Arguments

x	An evo_recipe object.
...	Additional arguments (currently unused).

Value

Invisible x. Called for its side effect of printing the recipe overview.

---

print.summary_evo_ensemble *Print summary of an evo_ensemble object*

Description

Prints summary details of the Caruana island ensemble.

Usage

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

Arguments

x	A summary_evo_ensemble object.
...	Additional arguments (currently unused).

Value

Invisible x. Called for its side effect of printing the ensemble summary.

---

print.summary_evo_recipe *Print summary of an evo_recipe object*

Description

Prints the summary details of the evolutionary feature engineering recipe.

Usage

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

Arguments

x	A summary_evo_recipe object.
...	Additional arguments (currently unused).

Value

Invisible x. Called for its side effect of printing the recipe summary.

---

register_evaluator *Register a model evaluator*

Description

Register a model evaluator

Usage

```
register_evaluator(
  name,
  train_func,
  predict_func,
  base_evaluator = NULL,
  cleanup_func = NULL
)
```

Arguments

name	Name of the evaluator.
train_func	Function to train the model. Must accept x_train, y_train, x_val, task, threads, num_class, and any additional parameters, and return a list with model, predictions, and importances.
predict_func	Function to make predictions. Must accept model, x_new, task, and any additional parameters, and return a vector or matrix of predictions.
base_evaluator	Optional character name of the base registered model.
cleanup_func	Optional function to clean up model resources/states after evaluation.

Value

Invisible NULL. Called for the side effect of registering the evaluator in evo_evaluators.

Examples

```
# Register a simple mock evaluator
register_evaluator(
  "mock_eval",
  train_func = function(x_train, y_train, x_val = NULL,
                        task = "regression", ...) {
    list(
      model = list(weights = colMeans(x_train)),
      predictions = if (!is.null(x_val)) rowMeans(x_val) else NULL,
      importances = stats::setNames(
        rep(1, ncol(x_train)), colnames(x_train)
      )
    )
  },
  predict_func = function(model, x_new, task, ...) {
    rowMeans(x_new)
  }
)

# Verify it is registered
exists("mock_eval", envir = evo_evaluators)
```

---

register_transformer *Register a custom feature transformer*

Description

Adds a user-defined feature transformer to the available pool for feature evolution.

Usage

```
register_transformer(name, transformer)
```

Arguments

`name`	Unique character string naming the transformer.
`transformer`	An object of class `evo_transformer` created via `create_transformer`.

Value

Invisible `transformer`, the registered transformer object.

Examples

```
# Create a custom transformer
add_ten_trans <- create_transformer(
  name = "add_ten",
  type = "unary",
  input_type = "numeric",
```

```
  apply_func = function(data, gene, state = NULL) {
    data[[gene$input_cols[1]]] + 10
  },
  name_generator = function(gene) paste0("add10_", gene$input_cols[1])
)

# Register it
register_transformer("add_ten", add_ten_trans)

# Verify it is registered
exists("add_ten", envir = evo_transformers)
```

`summary.evo_ensemble` *Summary of an evo_ensemble object*

Description

Computes and formats a detailed summary of the Caruana island ensemble.

Usage

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

Arguments

`object`	An `evo_ensemble` object.
`...`	Additional arguments (currently unused).

Value

An object of class `summary_evo_ensemble` containing detailed ensemble statistics.

`summary.evo_recipe` *Summary of an evo_recipe object*

Description

Computes and formats a detailed summary of the evolutionary feature engineering recipe.

Usage

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

Arguments

object	An evo_recipe object.
...	Additional arguments (currently unused).

Value

An object of class summary_evo_recipe containing detailed recipe statistics.

Examples

```
data(mtcars)
df <- mtcars
df$am <- as.integer(df$am)

recipe <- evolve_features(
  data = df,
  target_col = "am",
  task = "classification",
  evaluator = "xgboost",
  generations = 2,
  pop_size = 2,
  cv_folds = 2,
  seed = 42,
  verbose = FALSE
)

# Print the recipe overview
print(recipe)

# Inspect a detailed summary
recipe_summary <- summary(recipe)
print(recipe_summary)
```

---

topology *Graph Topology Constructors and Generics for Island Models*

Description

Functions to specify and query graph topologies connecting evolution islands. Topologies define island adjacency ($G = (V, E)$) independent of migration movement policies. Every topology object pre-builds and exposes an explicit adj_list and adj_matrix.

Usage

```
topology_ring(islands = 10)

topology_grid(islands = 10, rows = NULL, cols = NULL)
```

```
topology_torus(islands = 10, rows = NULL, cols = NULL)

topology_hypercube(islands = NULL, dimension = NULL)

topology_tiered(islands = 10, tiers = 3)

topology_complete(islands = 10)

get_neighbors(topology, island_id, state = NULL, ...)

## S3 method for class 'evo_topology'
get_neighbors(topology, island_id, state = NULL, ...)

get_in_neighbors(topology, island_id, state = NULL, ...)

## S3 method for class 'evo_topology'
get_in_neighbors(topology, island_id, state = NULL, ...)

## S3 method for class 'evo_topology_ring'
get_neighbors(topology, island_id, state = NULL, ...)

## S3 method for class 'evo_topology_grid'
get_neighbors(topology, island_id, state = NULL, ...)

## S3 method for class 'evo_topology_torus'
get_neighbors(topology, island_id, state = NULL, ...)

## S3 method for class 'evo_topology_hypercube'
get_neighbors(topology, island_id, state = NULL, ...)

## S3 method for class 'evo_topology_tiered'
get_neighbors(topology, island_id, state = NULL, ...)

## S3 method for class 'evo_topology_complete'
get_neighbors(topology, island_id, state = NULL, ...)

## S3 method for class 'evo_topology_custom'
get_neighbors(topology, island_id, state = NULL, ...)
```

Arguments

islands	Integer. Number of active islands (default: 10).
rows	Optional integer. Number of grid rows.
cols	Optional integer. Number of grid columns.
dimension	Optional integer. Hypercube dimension.
tiers	Integer. Number of tiers for pyramid topology (default: 3).

topology	An evo_topology object.
island_id	Integer. 1-indexed island ID (1 to islands).
state	Optional list containing runtime evolution state (populations, fitnesses, stagnation counts).
...	Additional arguments passed to methods.

Value

An object of class evo_topology (and specific subclass), or neighbor node IDs.

---

topology_custom *Custom Graph Topology Constructor*

Description

Custom Graph Topology Constructor

Usage

```
topology_custom(adj)
```

Arguments

adj	N x N binary matrix or list of integer vectors specifying adjacency.

Value

Object of class evo_topology_custom.

---

union_crossover *Union Crossover of two individuals*

Description

Union Crossover of two individuals

Usage

```
union_crossover(ind1, ind2, verbose = FALSE)
```

Arguments

ind1	Parent 1
ind2	Parent 2
verbose	Logical. Whether to print crossover details.

Value

An `evo_individual` child created by taking the union of all genes from both parents with duplicate gene outputs removed.

`view` *View the evolution history of an evo_recipe*

Description

Opens an interactive HTML page in the browser to visualize the evolutionary feature engineering process, either in real-time or post-hoc.

Usage

```
view(recipe, ...)

## S3 method for class 'evo_recipe'
view(recipe, ...)
```

Arguments

`recipe`	An `evo_recipe` object.
`...`	Additional arguments (not used).

Value

Invisible file path string to the generated HTML viewer file.

Index