Package 'evoFE'

June 9, 2026

Type Package

Title Evolutionary Feature Engineering

Version 0.1.0

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) <https://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-framework>, 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>.

License MIT + file LICENSE

Encoding UTF-8

Imports data.table, lightgbm, xgboost, stats, digest, uwot, quitefastmst, genieclust

Suggests RhpcBLASctl, testthat, knitr, rmarkdown, lumbermark, deadwood

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

RoxygenNote 7.3.2

NeedsCompilation no

Author Gustavo Pereira [aut, cre]

Maintainer Gustavo Pereira <tanopereira@gmail.com>

Repository CRAN

Date/Publication 2026-06-09 15:50:14 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
)
```

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.

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).

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)
)
```

Arguments

genes	List of genes
numeric_cols	Vector of numeric column names
categorical_cols	Vector of categorical column names

Value

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

`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`.

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.

evaluate_fitness *Evaluate the fitness of an individual*

Description

Evaluate the fitness of an individual

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
)
```

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. "train", "val", "holdout").
shared_splits	Optional list of shared data.table splits for in-place caching.
evaluator	The ML model to use ("lightgbm" or "xgboost").
fold_ids	Optional vector of pre-defined fold assignments.
shared_folds	Optional list of shared data.table CV folds for in-place caching.
shared_full	Optional data.table of the full dataset for in-place caching.
state_cache	Optional environment to cache full-dataset fitted states of stateful transformers.
threads	Number of threads to use for parallel execution (default 2)

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,
  early_stopping_rounds = 3,
```

```
  evaluator = "lightgbm",
  dynamic_population = TRUE,
  crossover_type = "both",
  threads = 2,
  max_clustering_size = 5000,
  verbose = TRUE
)
```

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. "train", "val", "holdout").
early_stopping_rounds	Stop if fitness doesn't improve for this many generations
evaluator	The ML model to use ("lightgbm" or "xgboost")
dynamic_population	Logical. If TRUE, population expands dynamically during stagnation.
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.

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).

evo_transformers *Built-in feature transformers*

Description

A list of default transformer definitions available for feature engineering.

Usage

```
evo_transformers
```

Value

A named list of `evo_transformer` objects, each defining a feature transformation (e.g. `log`, `pca`, `target_encode`).

gene_to_formula *Convert a gene to a formula string*

Description

Convert a gene to a formula string

Usage

```
gene_to_formula(gene)
```

Arguments

gene	A gene list

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,
  initial_genes = 2,
  task = "classification"
)
```

Arguments

pop_size	Population size.
numeric_cols	Vector of numeric column names.
categorical_cols	Vector of categorical column names.
initial_genes	Number of initial genes per individual.
task	Task type ("classification", "regression", or "multiclass").

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.

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
)
```

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).

Value

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

`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.

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

Description

Predict target values using the fully evolved model

Usage

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

Arguments

`object`	An evo_recipe object containing the trained model and best individual
`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).

`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.

Index