Package: leaf (via r-universe)

May 22, 2026

Type Package

Title Learning Equations for Automated Function Discovery

Version 0.1.0

Maintainer Francisco Martins <francisco.martins@tecnico.ulisboa.pt>

Description A unified framework for symbolic regression (SR) and multi-view symbolic regression (MvSR) designed for complex, nonlinear systems, with particular applicability to ecological datasets. The package implements a four-stage workflow: data subset generation, functional form discovery, numerical parameter optimization, and multi-objective evaluation. It provides a high-level formula-style interface that abstracts and extends multiple discovery engines: genetic programming (via PySR), Reinforcement Learning with Monte Carlo Tree Search (via RSRM), and exhaustive generalized linear model search. 'leaf' extends these methods by enabling multi-view discovery, where functional structures are shared across groups while parameters are fitted locally, and by supporting the enforcement of domain-specific constraints, such as sign consistency across groups. The framework automatically handles data normalization, link functions, and back-transformation, ensuring that discovered symbolic equations remain interpretable and valid on the original data scale. Implements methods following ongoing work by the authors (2026, in preparation).

URL https://github.com/NabiaAI/Leaf

Note Includes modified Python code from the RSRM project (<https://github.com/intell-sci-comput/RSRM>) under the MIT License.

License MIT + file LICENSE

Copyright see inst/COPYRIGHTS

Encoding UTF-8

Imports R6, utils, reticulate (>= 1.30), ggplot2, dplyr, rlang, rappdirs, rstudioapi

SystemRequirements Conda, Python (>= 3.10)

RoxygenNote 7.3.3

Config/Needs/website rmarkdown

Suggests rmarkdown, knitr, testthat (>= 3.0.0)

VignetteBuilder knitr

NeedsCompilation no

Config/testthat/edition 3

Author Francisco Martins [cre, aut, cph] (ORCID: <https://orcid.org/0009-0007-9941-2994>), Pedro Cardoso [aut] (ORCID: <https://orcid.org/0000-0001-8119-9960>), Manuel Lopes [aut] (ORCID: <https://orcid.org/0000-0002-6238-8974>), Vasco Branco [aut] (ORCID: <https://orcid.org/0000-0001-7797-3183>), INESC-ID [fnd] (Financed by FCT - PTDC/CCI-COM/5060/2021), intell-sci-comput [cph] (Copyright holder of RSRM (<https://github.com/intell-sci-comput/RSRM>))

Config/pak/sysreqs libpng-dev python3

Repository https://1-0-go.r-universe.dev

Date/Publication 2026-04-21 22:03:38 UTC

RemoteUrl https://github.com/cran/leaf

RemoteRef HEAD

RemoteSha 88a5398bf4cc9b88dc3a1ab180231c1a97e18133

Contents

backend_available *Check backend availability*

Description

Checks whether the required Python backend is available and properly configured for the current R session.

Usage

```
backend_available(conda = "auto")
```

Arguments

conda	Path to a conda executable. Default is "auto".

Value

Logical value. 'TRUE' if the configured Python backend is available and usable, 'FALSE' otherwise.

Note

This function performs a lightweight availability check. It does not initialize or modify the Python session.

check_python_environment *Check Python dependencies*

Description

Verifies if required Python modules are accessible by attempting an import.

Usage

```
check_python_environment()
```

Value

Logical. 'TRUE' if all required dependencies are available, 'FALSE' otherwise.

`generate_group_subsets` *Generate group-aware training subsets (wrapper around scikit-learn)*

Description

Convenience wrapper for the Python function ‘generate_group_folds‘. Delegates splitting to scikit-learn (GroupKFold / LeaveOneGroupOut / StratifiedGroupKFold). Behavior is equivalent to the corresponding scikit-learn cross-validator. Returned indices are converted to **1-based** row positions suitable for subsetting R ‘data.frame‘s.

Usage

```
generate_group_subsets(
  data,
  group_cols,
  n_splits = 5L,
  stratify = FALSE,
  shuffle = FALSE,
  random_state = NULL
)
```

Arguments

`data`	A ‘data.frame‘ with one row per sample.
`group_cols`	Character scalar or character vector. Column name(s) defining groups. If multiple names are given, group labels are formed as tuples of those columns’ values.
`n_splits`	Integer number of folds, or the string ‘’logo’‘ to use Leave-One-Group-Out. Default ‘5‘.
`stratify`	Logical; if ‘TRUE‘ uses stratified group-KFold (StratifiedGroupKFold). When ‘TRUE‘, ‘n_splits‘ must be an integer (cannot be ‘’logo’‘). Default ‘FALSE‘.
`shuffle`	Logical; passed to the underlying scikit-learn splitter where applicable (StratifiedGroupKFold supports shuffle). Default ‘FALSE‘.
`random_state`	Integer seed passed to scikit-learn when shuffling is enabled; otherwise ignored. Default ‘NULL‘.

Details

This function calls the Python iterator and collects all folds into an R list of integer vectors. Each element is the training-row positions for a fold.

Value

A list of integer vectors. Each element contains 1-based row positions (suitable for ‘data[indices,]‘) for the training set of a single fold or to use directly in leaf::SymbolicRegressor$search_equations. The list length equals the number of folds produced by the chosen cross-validator.

Examples

```
if(backend_available()){
# Requires install_leaf() to be run once before use
data = leaf_data("GDM")
folds <- generate_group_subsets(data, group_cols = "species", n_splits = 'logo')
}
```

`generate_shuffle_subsets` *Generate shuffle-based training subsets (wrapper around scikit-learn)*

Description

Convenience wrapper for the Python function ‘generate_shuffle_splits‘. Delegates splitting to scikit-learn (ShuffleSplit / StratifiedShuffleSplit / GroupShuffleSplit). Behavior is equivalent to the corresponding scikit-learn cross-validator.

Usage

```
generate_shuffle_subsets(
  data,
  n_splits = 5L,
  test_size = 0.2,
  train_size = NULL,
  stratify = FALSE,
  group_cols = NULL,
  random_state = NULL
)
```

Arguments

`data`	A ‘data.frame‘ with one row per sample.
`n_splits`	Integer number of re-shuffling & splitting iterations. Default ‘5‘.
`test_size`	Numeric (proportion or absolute) for the test set size. Default ‘0.2‘.
`train_size`	Numeric (proportion or absolute) for the train set size, or ‘NULL‘ to let scikit-learn decide. Default ‘NULL‘.
`stratify`	Logical; if ‘TRUE‘ uses StratifiedShuffleSplit. Cannot be combined with ‘group_cols‘. Default ‘FALSE‘.
`group_cols`	Optional character scalar or vector of column names. If provided, GroupShuffleSplit is used and ‘group_cols‘ defines groups. Cannot be combined with ‘stratify = TRUE‘.
`random_state`	Integer seed passed to scikit-learn. Default ‘NULL‘.

Details

Returned indices are converted to **1-based** row positions suitable for subsetting R ‘data.frame‘s. The Python iterator is consumed and a list of integer vectors is returned.

Value

A list of integer vectors. Each element contains 1-based row positions (suitable for ‘data[indices,]‘) for the training set of a single split.

Examples

```
if(backend_available()){
# Requires install_leaf() to be run once before use
data = leaf_data("GDM")
# plain ShuffleSplit
folds <- generate_shuffle_subsets(data, n_splits = 10, test_size = 0.25)

# grouped
folds_grouped <- generate_shuffle_subsets(data, group_cols = "species", n_splits = 5L)
}
```

---

generate_subsets *Generate standard (non-group) training subsets (wrapper around scikit-learn)*

Description

Convenience wrapper for the Python function ‘generate_folds‘. Delegates splitting to scikit-learn (KFold / LeaveOneOut / StratifiedKFold). Behavior is equivalent to the corresponding scikit-learn cross-validator.

Usage

```
generate_subsets(
  data,
  n_splits = 5L,
  stratify = FALSE,
  shuffle = FALSE,
  random_state = NULL
)
```

Arguments

data	A ‘data.frame‘ with one row per sample.
n_splits	Integer number of folds, or the string ‘’loo’‘ to use Leave-One-Out. Default ‘5‘.
stratify	Logical; if ‘TRUE‘ uses StratifiedKFold. When ‘TRUE‘, ‘n_splits‘ must be an integer (cannot be ‘’loo’‘). Default ‘FALSE‘.

shuffle	Logical; passed to the underlying scikit-learn splitter where applicable (KFold/StratifiedKFold support shuffle). Default 'FALSE'.
random_state	Integer seed passed to scikit-learn when shuffling is enabled; otherwise ignored. Default 'NULL'.

Details

Returned indices are converted to **1-based** row positions suitable for subsetting R 'data.frame's. The Python iterator is consumed and a list of integer vectors is returned.

Value

A list of integer vectors. Each element contains 1-based row positions (suitable for 'data[indices,]') for the training set of a single fold.

Examples

```
if(backend_available()){
# Requires install_leaf() to be run once before use
data = leaf_data("GDM")
folds <- generate_subsets(data, n_splits = 5L)
}
```

---

get_python_lib *Load a Python module*

Description

Imports a specific Python module from the leaf environment.

Usage

```
get_python_lib(lib)
```

Arguments

lib	Character. Name of the Python library to import.

Value

A Python module object (reticulate proxy) that can be accessed using the '$' operator.

get_python_path *Get leaf's python environment path*

Description

Get the path to which leaf's python environment was installed.

Usage

```
get_python_path()
```

Value

Character. The absolute path to the Python executable within leaf's dedicated environment.

Examples

```
if(backend_available()){
  # Requires install_leaf() to be run once before use
  reticulate::import("numpy", get_python_path())
}
```

install_leaf *Install the LEAF Python Environment*

Description

Creates a Conda environment called "r-leafr" with Python 3.12 and all required dependencies for the leaf package.

Usage

```
install_leaf(conda = "auto", restart_session = TRUE)
```

Arguments

conda	Path to a conda executable. Default is "auto".
restart_session	Logical. If TRUE, restarts the RStudio session after installation. Default is TRUE.

Details

This function will create a directory in the user's data directory (as resolved by `user_data_dir`) to store the path to the Conda environment. On most systems this resolves to:

- Linux: `~/.local/share/leafr`
- macOS: `~/Library/Application Support/leafr`
- Windows: `C:/Users/<username>/AppData/Local/leafr`

The user will be prompted for consent before the directory is created. This function must be called interactively.

Value

None (invisible).

Examples

```
install_leaf()
```

`leaf_data` *Example data packaged with *leaf**

Description

Load data included in the package. This includes two matrices *eg_train* and *eg_test*, with 3 columns and 40 and 10 rows respectively. These columns include one dependent variable, `y` and two independent variables, `x1`, `x2`.

Usage

```
leaf_data(data = NULL)
```

Arguments

`data`	Name of data in quotes. E.g.: '"eg_train"' If 'NULL', the example files will be listed.

Value

A 'data.frame' containing the requested example dataset. If 'data = NULL', prints the available dataset names and returns 'NULL' invisibly.

Source

This function is inspired by 'palmerpanguins::path_to_file()' which in turn is based on 'readxl::readxl_example()'.

Examples

```
leaf_data()
leaf_data("eg_train")
```

`load_leaf_analysis` *Load a Saved Leaf Analysis*

Description

Load a previously saved ‘SymbolicRegressor‘ analysis from a pickle file.

This function restores a ‘SymbolicRegressor‘ object that was previously saved using the Python implementation. The entire state of the model (including fitted equations, parameters, and configuration) is recovered, allowing the analysis to be continued or used for prediction.

Usage

```
load_leaf_analysis(filepath)
```

Arguments

`filepath`	Character string. Path to the saved model file (for example ‘"my_model.pkl"‘).

Details

Internally, this function calls the Python class method ‘SymbolicRegressor$load()‘ and wraps the returned Python object in the corresponding R interface so it can be used from R.

Value

An R6 ‘SymbolicRegressor‘ object wrapping the loaded Python model.

Examples

```
## Not run:
if(backend_available()){
 # Requires install_leaf() to be run once before use and a previously saved model

 # Load a previously saved model
 regressor <- load_leaf_analysis("my_model.pkl")

 # Use the loaded model for prediction
 regressor$predict(
    data = leaf_data("eg_test"),
   equation_id = 1
 )
}

## End(Not run)
```

pareto_2d *Compute 2D Pareto Front*

Description

Identifies the non-dominated (Pareto-optimal) points for two objectives.

Usage

```
pareto_2d(
  x,
  y,
  x_bigger_better = FALSE,
  y_bigger_better = FALSE,
  na_remove = TRUE
)
```

Arguments

x	Numeric vector representing the first objective.
y	Numeric vector representing the second objective.
x_bigger_better	Logical. TRUE if larger values of x are better, FALSE if smaller is better. Default FALSE.
y_bigger_better	Logical. TRUE if larger values of y are better, FALSE if smaller is better. Default FALSE.
na_remove	Logical. If TRUE, NAs are considered dominated and removed. Default TRUE.

Value

Logical vector indicating which points are Pareto-optimal (TRUE if non-dominated).

Examples

```
x <- c(1, 2, 3)
y <- c(3, 2, 1)
pareto_2d(x, y, x_bigger_better = FALSE, y_bigger_better = FALSE)
```

`plot_pareto` *Plot the Pareto Front*

Description

Plots the Pareto front for two objectives, highlighting the trade-off between metrics and connecting the non-dominated points with a dashed line.

Usage

```
plot_pareto(
  df,
  x_col,
  y_col,
  x_bigger_better = FALSE,
  y_bigger_better = FALSE,
  show_all_points = FALSE,
  point_size = 2.5,
  pareto_point_size = 3.5
)
```

Arguments

`df`	Data frame containing at least two numeric columns.
`x_col`	Name of the column for the x-axis.
`y_col`	Name of the column for the y-axis.
`x_bigger_better`	Logical. TRUE if larger x values are better, FALSE if smaller is better.
`y_bigger_better`	Logical. TRUE if larger y values are better, FALSE if smaller is better.
`show_all_points`	Logical. If TRUE, shows all points; if FALSE, only Pareto points.
`point_size`	Numeric. Size of non-Pareto points.
`pareto_point_size`	Numeric. Size of Pareto points.

Value

A list with:

`plot`	ggplot object of the Pareto front.
`pareto_points`	Data frame of Pareto-optimal points.
`all`	Data frame including a logical column '.pareto' indicating Pareto points.

References

- Giagkiozis, I. and Fleming, P.J. (2014) 'Pareto front estimation for decision making', Evolutionary Computation, 22(4), pp. 651–678. doi:10.1162/evco_a_00128.
- Tušar, T. and Filipič, B. (2015) 'Visualization of pareto front approximations in evolutionary multiobjective optimization: A critical review and the prosection method', IEEE Transactions on Evolutionary Computation, 19(2), pp. 225–245. doi:10.1109/tevc.2014.2313407.

Examples

```
set.seed(123)
df <- data.frame(obj1 = runif(20, 0, 10), obj2 = runif(20, 0, 10))
res <- plot_pareto(df, "obj1", "obj2", x_bigger_better = FALSE, y_bigger_better = TRUE)
print(res$plot)
```

SymbolicRegressor *Symbolic Regressor*

Description

R6 interface to the **LEAF** symbolic regression framework.

#' **Important:** Before using 'SymbolicRegressor', you must run 'leaf::install_leaf()' to install the Python backend. This only needs to be done once per system.

'SymbolicRegressor' provides a high-level interface for performing symbolic regression analyses using the LEAF framework. It wraps the underlying Python implementation and exposes a simplified workflow for discovering, fitting, evaluating, and interpreting symbolic models directly from R.

A typical analysis consists of the following stages:

- **Initialize** – Create a 'SymbolicRegressor' object specifying the symbolic regression engine and loss function.
- **Search** – Discover candidate symbolic equations from the data. This is the main equation discovery phase.
- **Fit** – Estimate the parameters of the discovered equations using training data.
- **Evaluate** – Assess equation performance on new datasets (for example a validation or test set).
- **Analyze** – Inspect results, view equation details, and compare model trade-offs.
- **Persist** – Save or reload analyses to resume work later.

Supported evaluation metrics include: `R2`, `RMSE`, `MAE`, `F1`, `TSS`, `PseudoR2`, `AIC`, `AICc`, `AIC_w`, and `AICc_w`.

See the package documentation (`?leaf`) and examples for a complete workflow demonstrating how to run a symbolic regression analysis.

External documentation

The R interface wraps the Python implementation of the LEAF symbolic regression framework. For detailed information about the underlying algorithms and advanced engine configuration, see:

- PySR documentation: https://ai.damtp.cam.ac.uk/pysr/v1.5.9/
- RSRM documentation: https://github.com/intell-sci-comput/RSRM

Public fields

`py_obj` The underlying Python object.

Methods

Public methods:

- `SymbolicRegressor$new()`
- `SymbolicRegressor$search_equations()`
- `SymbolicRegressor$fit()`
- `SymbolicRegressor$evaluate()`
- `SymbolicRegressor$get_results()`
- `SymbolicRegressor$predict()`
- `SymbolicRegressor$find_optimal_threshold()`
- `SymbolicRegressor$get_pareto_front()`
- `SymbolicRegressor$show_equation()`
- `SymbolicRegressor$save()`
- `SymbolicRegressor$exclude()`
- `SymbolicRegressor$clone()`

Method `new()`**:** Initialize a new symbolic regression analysis.

This constructor creates a 'SymbolicRegressor' object that manages the symbolic regression workflow. The user specifies the backend symbolic regression engine and optional configuration parameters controlling the search process and model complexity.

Some arguments are only supported by specific engines. Unsupported arguments are ignored with a warning.

In addition to automated symbolic regression, 'leafr' supports a '"manual"' engine that allows users to directly specify their own candidate equations. This bypasses the automated search step and enables users to evaluate, fit, and compare user-defined models alongside automatically discovered or baseline models. See detailed example in vignettes.

Usage:

```
SymbolicRegressor$new(engine, loss = "MSE", force_consistent_sign = FALSE, ...)
```

Arguments:

`engine` Character string specifying the symbolic regression engine. Supported engines are '"rsrm"', '"pysr"', '"linear"'and '"manual"'.

`loss` Character string specifying the loss function to optimize. The provided name is automatically mapped to the engine-specific loss implementation. Default is '"MSE"'.
Allowed losses include:

- PySR: MAE, MSE, BinaryCrossEntropy, ExpLoss, PoissonDeviance
- RSRM: MAE, MSE, RMSE, R2, RobustTanh, BinaryCrossEntropy, ExpLoss, PoissonDeviance
- Linear: MSE, PoissonDeviance, BinaryCrossEntropy

force_consistent_sign Logical. If 'TRUE', model parameters are constrained to maintain the same sign across groups during parameter optimization. This enforces consistent effect direction in multi-group settings. Supported by '"rsrm"', '"linear"', '"manual"'. Default is 'FALSE'.

... Additional engine-specific arguments passed directly to the underlying symbolic regression engine for advanced customization.
#' @details See the 'SymbolicRegressor' class documentation for links to the full Python framework documentation.

num_iterations Integer specifying the number of search iterations or generations used by the symbolic regression engine. Supported by '"rsrm"' and '"pysr"'.

operators Character vector specifying the mathematical operators allowed in candidate expressions (for example 'c("+", "-", "*", "/")'). Supported by '"rsrm"' and '"pysr"'.

max_complexity Integer limiting the symbolic complexity of generated expressions. Interpretation may vary between engines. Supported by '"rsrm"' and '"pysr"'.

max_params Integer specifying the maximum number of free parameters (constants) allowed in expressions. Supported by '"rsrm"' and '"pysr"' (multi-view SR mode only).

max_depth Integer limiting the maximum depth of the expression tree. Supported by '"rsrm"' and '"pysr"'.

Returns: A new 'SymbolicRegressor' object.

Examples:

```
if(backend_available()){
# Requires install_leaf() to be run once before use
regressor <- leaf::SymbolicRegressor$new(
  engine = "rsrm",
  loss = "MSE",
  operators = c("+"),
  threshold = 1e-10,
  base = list(verbose = FALSE, epoch = as.integer(2)),
  mcts = list(times = as.integer(20)),
  gp = list(times = as.integer(6))
)
}
```

Method `search_equations()`**:** Discover equation skeletons using the SR engine, with optional sampling. Processes the input data based on the specified formula, applies group-wise normalization, and then initiates the computationally intensive search phase using the designated backend engine. It identifies a set of candidate equations but does not fit their internal parameters.

Usage:

```
SymbolicRegressor$search_equations(
  data,
  formula,
  normalization = "divide_by_gmd",
```

```
  folds = NULL
)
```

Arguments:

`data` A data.frame containing the variables. The input data containing all variables mentioned in the formula.

`formula` A character string specifying the model formula. Functional transformations are allowed inside the formula (e.g. `"y ~ f(log(x1), x2)"`).
The formula follows the structure `y ~ f(...)`, where the arguments inside `f(...)` define the input variables used in the model.
A pipe symbol `|` can be used to specify grouping variables: `y ~ f(x1, x2 | group1, group2)`. Variables after `|` define groups for multi-view modeling.
All variable names must correspond to column names in the dataset provided to `search_equations()`.
Transformations are evaluated using Python's `pandas.DataFrame.eval()` on the provided dataset. As a result, expressions must follow Python syntax rather than R syntax (e.g. use `**` for exponentiation instead of `^`).

`normalization` Method to normalize data (zscore','minmax','divide_by_gmd','none') [SciKit-learn Contributors, 2025a, 2025b]. Default is divide_by_gmd.

`folds` Optional list of train/test indices. Assumes input is in the form list(list(train = c(...), test = c(...)), ...), which is a list of folds where each fold consists of a named list indicating the train and test indices for that fold. See documentation for helpers to more easily generate these folds.

Returns: A data.frame of discovered equations where columns are ID, Equation and Complexity.

Examples:

```
if(backend_available()){
# Requires install_leaf() to be run once before use
# after following the use flow described in ?leaf:

search_results = regressor$search_equations (
  data = leaf_data("eg_train"),
  formula ='y ~ f(x1, x2)'
)
}
```

Method `fit()`**:** Fit the parameters of the discovered equations. Fits the global and group-specific parameters for each equation found during the search phase. By default, it uses the same dataset provided to `search_equations`, unless a new dataset is explicitly passed.

Usage:

```
SymbolicRegressor$fit(data = NULL, do_cv = FALSE, n_cv_folds = NULL)
```

Arguments:

`data` data.frame. The data to use for fitting. If left NULL, the data from the 'search_equations' call is used.

`do_cv` Perform cross-validation? (Default FALSE).

`n_cv_folds` Number of cross validation folds. If NULL, the chosen k_fold will be 10 or the minimum group size if the dataset or any group has less than 10 elements, since each fold must contain at least one example of each group.

Returns: A data.frame of discovered equations where columns are ID, Equation, Complexity and Loss.

Examples:

```
if(backend_available()){
# Requires install_leaf() to be run once before use
# after following the use flow described in ?leaf:

fit_results = regressor$fit(
  data = leaf_data("eg_train"),
  do_cv = FALSE
)
}
```

Method `evaluate()`**:** Evaluate equations on specific metrics.

Usage:

```
SymbolicRegressor$evaluate(
  metrics = NULL,
  metrics_cv = NULL,
  data = NULL,
  equation_ids = NULL
)
```

Arguments:

`metrics` Character vector of metrics.

`metrics_cv` Character vector of CV metrics. Requires `fit` to have been run with `do_cv = True`.

`data` Optional test data.frame. An external dataset used to evaluate fitted equations on `metrics`.

`equation_ids` Integer vector of IDs to evaluate. If NULL, all equations are evaluated. likelihood_for_AIC_and_Elbow : callable, optional Custom likelihood function for AIC (Akaike, 2011) and Elbow calculation. Defaults to the Poisson likelihood if the loss is Poisson-Deviance, Bernoulli likelihood if the loss is 'BinaryCrossEntropy' or 'ExpLoss', and the Gaussian likelihood otherwise.

Returns: A 'data.frame' of equations where columns include the original results (ID, Equation, Complexity, Loss) plus the newly evaluated metrics.

Examples:

```
if(backend_available()){
# Requires install_leaf() to be run once before use
# after following the use flow described in ?leaf:

eval_table = regressor$evaluate(
  metrics=c('RMSE', 'Elbow'),
  data= leaf_data("eg_test")
)
}
```

Method `get_results()`**:** Returns the symbolic regression results as a 'data.frame'.

The returned table always contains: - 'ID' (integer) - 'Equation' (string) - 'Complexity' (integer)

Additional columns may be present depending on which steps have been run: - ‘Loss‘ — if ‘fit_equation_parameters()‘ has been run - ‘Loss (cv)‘ — if cross-validation was enabled - Other metrics — if ‘evaluate()‘ has been run with custom metrics

Usage:

```
SymbolicRegressor$get_results()
```

Arguments:

`py_regressor` A reticulate-wrapped Python symbolic regression object.

`drop_index` Logical; if TRUE (default), Python row indices are replaced with standard R row numbers.

Returns: A ‘data.frame‘ where columns are ID, Equation, Complexity, and any metrics computed in previous steps.

Examples:

```
if(backend_available()){
# Requires install_leaf() to be run once before use
df <- regressor$get_results()
head(df)
}
```

Method `predict()`**:** Make predictions using a specific equation.

Usage:

```
SymbolicRegressor$predict(data, equation_id)
```

Arguments:

`data` New data.frame. Input features must match the formula used in training.

`equation_id` The ID of the equation to use.

Returns: Numeric vector of predictions.

Examples:

```
if(backend_available()){
# Requires install_leaf() to be run once before use
# after following the use flow described in ?leaf:

regressor$predict(
  data = leaf_data("eg_test"),
  equation_id=1
)
}
```

Method `find_optimal_threshold()`**:** Compute the optimal decision threshold for a classification equation using the True Skill Statistic (TSS) via cross-validation.

Usage:

```
SymbolicRegressor$find_optimal_threshold(equation_id, data = NULL, k_folds = 5)
```

Arguments:

`equation_id` Integer. Identifier of the equation for which the optimal threshold should be computed.

`data` Optional data.frame used for threshold selection. If 'NULL', the training data used during model fitting are used.

`k_folds` Integer. Number of folds used for cross-validation (default = 5).

Details: This method identifies the probability cutoff that maximizes the TSS (sensitivity + specificity - 1) using k-fold cross-validation. For multi-view or grouped models, group labels are preserved during optimization to ensure robust estimation.
Note: This method is only applicable to classification models. If called on a regression model, it will raise a 'RuntimeError'. Your target variable must be binary.

Returns: Numeric scalar representing the optimal probability threshold that maximizes the TSS across cross-validation folds.

Examples:

```
if(backend_available()){
# Requires install_leaf() to be run once before use
# after following the workflow described in ?leaf:

regressor$find_optimal_threshold(
  equation_id = 1,
  data = leaf_data("eg_train"),
  k_folds = 5
)
}
```

Method `get_pareto_front()`**:** Returns a filtered view of the symbolic regression results table. This function provides a convenient way to inspect the discovered equations. By default, it restricts the output to the Pareto-optimal subset. The Pareto front consists of equations that are not outperformed by any simpler alternative on the selected performance metric.

Usage:

```
SymbolicRegressor$get_pareto_front(
  metrics = NULL,
  equation_ids = NULL,
  pareto_metric = NULL
)
```

Arguments:

`metrics` Character vector of metric column names to include in the results table. Each entry must match a column name in 'regressor$results_'. Default 'NULL' includes all metrics.

`equation_ids` Integer vector of specific equation IDs to include. If 'NULL', all equations are considered.

`pareto_metric` Character scalar indicating the performance metric used to compute the Pareto front (e.g., '"Loss (cv)"', '"R2 (test)"'). If 'NULL', the default metric is '"Loss (cv)"' if that column is available, otherwise '"Loss"' is used.

Returns: A 'data.frame' corresponding to the full or Pareto-filtered results table where columns are ID, Equation, Complexity, and any metrics computed in previous steps.

Examples:

```
if(backend_available()){
# Requires install_leaf() to be run once before use
```

```
regressor$get_pareto_front()

regressor$get_pareto_front(
  metrics = c('RMSE', 'Elbow'),
  equation_ids = c(1, 5, 7)
)
}
```

Method `show_equation()`**:** Display detailed information about a fitted symbolic equation.

This method prints the full symbolic structure of the selected equation, including the expanded expression for each group after reversing the normalization applied during model training. The output helps users interpret how predictors and coefficients contribute to the final model.

Global coefficients (β) are shared across all groups, while group-specific coefficients (u) are fitted separately within each group.

Usage:

```
SymbolicRegressor$show_equation(equation_id)
```

Arguments:

`equation_id` Integer. Identifier of the equation for which details should be displayed.

Returns: This function prints information to the console and returns 'NULL'.

Examples:

```
if(backend_available()){
# Requires install_leaf() to be run once before use
# after following the workflow described in ?leaf:

regressor$show_equation(
  equation_id = 1
)
}
```

Method `save()`**:** Save the current analysis to a pickle file.

This method serializes the entire 'SymbolicRegressor' instance, including the symbolic regression engine, data transformer, discovered equations, fitted parameters, and the results table. The saved file can later be reloaded with [load_leaf_analysis()] to resume the analysis or make predictions without refitting the model.

Usage:

```
SymbolicRegressor$save(filepath)
```

Arguments:

`filepath` Character string. Path to the output file (for example '"my_model.pkl"').

Returns: None (invisible). The model is saved to the specified 'filepath'.

Examples:

```
if(backend_available()){
# Requires install_leaf() to be run once before use
# after following the workflow described in ?leaf:
```

```
# Use a temporary file for CRAN checks; in practice, provide your own path
path <- tempfile(fileext = ".pkl")

# Save the trained analysis
regressor$save(path)

# Later, reload it
regressor <- load_leaf_analysis(path)
}
```

Method `exclude()`**:** Permanently excludes specified equations from the regressor object.

Usage:

```
SymbolicRegressor$exclude(equation_ids)
```

Arguments:

`equation_ids` Integer vector of IDs to be removed from the candidate pool.

Returns: None (invisible). The specified equations are removed internally.

Examples:

```
if(backend_available()){
# Requires install_leaf() to be run once before use
# after following the use flow described in ?leaf:

regressor$exclude(
  equation_ids=c(1,2,3)
)
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
SymbolicRegressor$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Note

This class provides an R interface to the Python implementation of the LEAF symbolic regression framework via the ‘reticulate‘ package.

References

- Xu, Y., Liu, Y., Sun, H. (2023). *RSRM: Reinforcement Symbolic Regression Machine*. arXiv:2305.14656.
- Cranmer, M. (2023). *Interpretable Machine Learning for Science with PySR and SymbolicRegression.jl*. arXiv:2305.01582.
- Schluchter, M. D. (2005). *Mean Square Error*. Encyclopedia of Biostatistics.
- SciKit-learn Contributors (2025a). ‘StandardScaler‘. In: scikit-learn User Guide v.1.8.0. url: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

- SciKit-learn Contributors (2025b). 'MinMaxScaler'. In: scikit-learn User Guide v.1.8.0. url: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html
- Akaike, H. (2011). 'Akaike's Information Criterion'. In: Lovric, M. (eds) International Encyclopedia of Statistical Science. Springer, Berlin, Heidelberg. doi:10.1007/978-3-642-04898-2_110
- elbow
- SciKit-learn Contributors (2025). '3.4.6.9. Mean Poisson, Gamma, and Tweedie deviances'. In: scikit-learn User Guide v.1.8.0. url: https://scikit-learn.org/stable/modules/model_evaluation.html#mean-tweedie-deviance
- Bernoulli
- PyTorch Contributors. 'Binary Cross Entropy Loss (BCELoss)'. url: https://docs.pytorch.org/docs/stable/generated/torc
- ExpLoss

Examples

```
## ------------------------------------------------
## Method `SymbolicRegressor$new`
## ------------------------------------------------

if(backend_available()){
# Requires install_leaf() to be run once before use
regressor <- leaf::SymbolicRegressor$new(
  engine = "rsrm",
  loss = "MSE",
  operators = c("+"),
  threshold = 1e-10,
  base = list(verbose = FALSE, epoch = as.integer(2)),
  mcts = list(times = as.integer(20)),
  gp = list(times = as.integer(6))
)
}


## ------------------------------------------------
## Method `SymbolicRegressor$search_equations`
## ------------------------------------------------

if(backend_available()){
# Requires install_leaf() to be run once before use
# after following the use flow described in ?leaf:

search_results = regressor$search_equations (
  data = leaf_data("eg_train"),
  formula ='y ~ f(x1, x2)'
)
}

## ------------------------------------------------
## Method `SymbolicRegressor$fit`
```

```
## ------------------------------------------------

if(backend_available()){
# Requires install_leaf() to be run once before use
# after following the use flow described in ?leaf:

fit_results = regressor$fit(
  data = leaf_data("eg_train"),
  do_cv = FALSE
)
}

## ------------------------------------------------
## Method `SymbolicRegressor$evaluate`
## ------------------------------------------------

if(backend_available()){
# Requires install_leaf() to be run once before use
# after following the use flow described in ?leaf:

eval_table = regressor$evaluate(
  metrics=c('RMSE', 'Elbow'),
  data= leaf_data("eg_test")
)
}

## ------------------------------------------------
## Method `SymbolicRegressor$get_results`
## ------------------------------------------------

if(backend_available()){
# Requires install_leaf() to be run once before use
df <- regressor$get_results()
head(df)
}

## ------------------------------------------------
## Method `SymbolicRegressor$predict`
## ------------------------------------------------

if(backend_available()){
# Requires install_leaf() to be run once before use
# after following the use flow described in ?leaf:

regressor$predict(
  data = leaf_data("eg_test"),
  equation_id=1
)
}

## ------------------------------------------------
## Method `SymbolicRegressor$find_optimal_threshold`
## ------------------------------------------------
```

```
if(backend_available()){
# Requires install_leaf() to be run once before use
# after following the workflow described in ?leaf:

regressor$find_optimal_threshold(
  equation_id = 1,
  data = leaf_data("eg_train"),
  k_folds = 5
)
}

## ------------------------------------------------
## Method `SymbolicRegressor$get_pareto_front`
## ------------------------------------------------

if(backend_available()){
# Requires install_leaf() to be run once before use
regressor$get_pareto_front()

regressor$get_pareto_front(
  metrics = c('RMSE', 'Elbow'),
  equation_ids = c(1, 5, 7)
)
}

## ------------------------------------------------
## Method `SymbolicRegressor$show_equation`
## ------------------------------------------------

if(backend_available()){
# Requires install_leaf() to be run once before use
# after following the workflow described in ?leaf:

regressor$show_equation(
  equation_id = 1
)
}

## ------------------------------------------------
## Method `SymbolicRegressor$save`
## ------------------------------------------------

if(backend_available()){
# Requires install_leaf() to be run once before use
# after following the workflow described in ?leaf:

# Use a temporary file for CRAN checks; in practice, provide your own path
path <- tempfile(fileext = ".pkl")

# Save the trained analysis
regressor$save(path)
```

```
# Later, reload it
regressor <- load_leaf_analysis(path)
}

## ------------------------------------------------
## Method `SymbolicRegressor$exclude`
## ------------------------------------------------

if(backend_available()){
# Requires install_leaf() to be run once before use
# after following the use flow described in ?leaf:

regressor$exclude(
  equation_ids=c(1,2,3)
)
}
```

`uninstall_leaf` *Uninstall the LEAF Python Environment*

Description

Removes the Conda environment associated with the leaf package and deletes the environment record file and data directory.

Usage

```
uninstall_leaf(conda = "auto")
```

Arguments

`conda`	Path to a conda executable. Default is `"auto"`.

Details

Removes the Conda environment stored at the path recorded in the leafr data directory, then deletes the record file and data directory itself. The data directory location follows the same conventions as `install_leaf`.

Value

None (invisible).

Examples

```
uninstall_leaf()
```

Index