Package: xplaineff (via r-universe)

October 29, 2026

Type Package

Title Decomposing Global Feature Effects Based on Feature Interactions

Version 0.1.1

Description Implements the GADGET (Generalized Additive Decomposition of Global EffecTs) algorithm for interpretable machine learning. The package recursively partitions the feature space to minimize heterogeneity of feature effects (e.g., Accumulated Local Effects or Partial Dependence), producing a tree of regions where effects are more stable. It supports both ALE and PD strategies, works with 'mlr3' learners and provides visualization of the interaction tree and regional effect plots. The method is described in Herbinger, J., Wright, M. N., Nagler, T., Bischl, B., and Casalicchio, G. (2024), ``Decomposing Global Feature Effects Based on Feature Interactions" <https://jmlr.org/papers/volume25/23-0699/23-0699.pdf>.

License MIT + file LICENSE

Depends R (>= 4.3.0)

URL https://github.com/mlr-org/xplaineff

BugReports https://github.com/mlr-org/xplaineff/issues

Encoding UTF-8

Imports checkmate (>= 2.3.2), cli (>= 3.0.0), data.table (>= 1.14.0), ggplot2 (>= 3.5.2), ggraph (>= 2.2.1), igraph (>= 2.1.4), mlr3misc (>= 0.14.0), patchwork (>= 1.3.0), R6 (>= 2.6.1), Rcpp (>= 1.0.0)

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 7.3.2

Suggests testthat (>= 3.0.0), knitr, rmarkdown, iml (>= 0.11.4), mlr3, mlr3learners, ranger, ISLR2, rpart, withr, xgboost

Config/testthat/edition 3

VignetteBuilder knitr

Collate 'EffectStrategy.R' 'AleStrategy.R' 'GadgetTree.R' 'Node.R' 'PdStrategy.R' 'RcppExports.R' 'calculate_ale.R' 'calculate_ale_fast.R' 'calculate_ale_heterogeneity.R' 'calculate_pd.R' 'categorical_split_utils.R' 'choose_operator.R' 'convert_tree_to_list.R' 'extract_split_info.R' 'factor_to_numeric.R' 'find_node_by_id.R' 'mean_center_ice.R' 'node_heterogeneity.R' 'node_transform_ale.R' 'order_categorical_levels.R' 'plot_regional_ale.R' 'plot_regional_pd.R' 'plot_tree_ale.R' 'plot_tree_pd.R' 'plot_tree_structure.R' 'plot_utils.R' 'prepare_layout_data.R' 'prepare_plot_data_ale.R' 'prepare_split_data_ale.R' 'prepare_split_data_pd.R' 'prepare_split_data_utils.R' 'search_best_split_ale.R' 'search_best_split_point_ale.R' 'track_split_condition.R' 'xplaineff_internal.R' 'xplaineff_package.R'

NeedsCompilation yes

Author Zizheng Zhang [aut, cre] (ORCID: <https://orcid.org/0009-0009-4252-8018>), Coco Bögel [aut] (ORCID: <https://orcid.org/0009-0002-0683-4957>), Giuseppe Casalicchio [aut] (ORCID: <https://orcid.org/0000-0001-5324-5966>), Bernd Bischl [aut] (ORCID: <https://orcid.org/0000-0001-6002-6980>)

Maintainer Zizheng Zhang <Zizheng.Zhang@stat.uni-muenchen.de>

Repository https://cran.r-universe.dev

Date/Publication 2026-09-29 11:30:02 UTC

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

RemoteRef HEAD

RemoteSha 6958470630ec130ca8264cb25689e2433d5b4270

Contents

xplaineff-package 2
AleStrategy 4
GadgetTree 7
PdStrategy 10

Index **14**

`xplaineff-package` *xplaineff: Generalized Additive Decomposition of Global EffecTs*

Description

The **xplaineff** package implements the GADGET algorithm for interpretable machine learning. It builds a tree by recursively partitioning the feature space to minimize the heterogeneity of feature effects (e.g., Accumulated Local Effects or Partial Dependence), so that within each region the effects are more stable and easier to interpret.

Details

Main components (user-facing):

- `GadgetTree`: R6 class to grow and visualize effect-based trees.
- `AleStrategy`: Strategy for ALE-based trees (ALE computed internally from a fitted model).
- `PdStrategy`: Strategy for PD-based trees (uses precomputed ICE/PD from **iml** or similar tools).

Typical workflow:

1. Train a model (e.g., with **mlr3**).
2. Create a tree: `tree = GadgetTree$new(strategy = AleStrategy$new(), n_split = 3, min_node_size = 50)`.
3. Fit: `tree$fit(data, target_feature_name, ...)`. Strategy-specific `...` arguments include:
   - **AleStrategy**: `model` (required), `n_intervals = 10`, `predict_fun = NULL`, `order_method = "raw"`, `ale_engine` (default `"auto"`), `categorical_split`, and `max_exhaustive_levels`.
   - **PdStrategy**: either `effect`, or `model` with optional `predict_fun`, `n_grid`, `pd_engine` (default `"auto"`), `categorical_split`, and `max_exhaustive_levels`.
   - Both strategies accept `feature_set` and `split_feature`.
   - Tree parameters include `impr_par`, `min_node_size`, and `n_quantiles`.
4. Visualize: `tree$plot_tree_structure()`, `tree$plot(...)`, `tree$extract_split_info()`.

For PD-based trees, either pass an effect object from `iml::FeatureEffects(..., method = "ice")` to `tree$fit(effect = ..., data = ..., target_feature_name = ...)`, or pass a fitted model and let xplaineff compute PD/ICE internally.

Author(s)

Maintainer: Zizheng Zhang <Zizheng.Zhang@stat.uni-muenchen.de>

References

Herbinger, J., Wright, M. N., Nagler, T., Bischl, B., and Casalicchio, G. (2024). Decomposing Global Feature Effects Based on Feature Interactions. *Journal of Machine Learning Research*, 25(23-0699), 1–65. URL: https://jmlr.org/papers/volume25/23-0699/23-0699.pdf.

Apley, D.W. and Zhu, J. (2016). Visualizing the Effects of Predictors on the Response in Non-linear and Generalized Linear Models. *Journal of Computational and Graphical Statistics*, 25(2), 590–600.

See Also

GadgetTree, AleStrategy, PdStrategy

`AleStrategy` *AleStrategy: Generalized Additive Decomposition Based on ALE Effects*

Description

ALE-based effect strategy (inherits from EffectStrategy). Given model and data, preprocesses to Z/Y via `prepare_split_data_ale`; transforms ALE effects per node; computes ALE-derivative heterogeneity; finds best split via `search_best_split_ale`; fits tree and plots ALE curves.

Format

R6::R6Class object inheriting from EffectStrategy.

Details

Intended for use through `GadgetTree$new(strategy = AleStrategy$new())` and `tree$fit(...)`. Can be instantiated directly for custom pipelines.

Construction

```
s = AleStrategy$new(categorical_split = "ordered_prefix")
```

Super class

`::EffectStrategy -> AleStrategy`

Public fields

`model` (any)
: Fitted model (persistent after `$fit()`).

`data` (`data.frame()` or `data.table()`)
: Data (persistent after `$fit()`).

`target_feature_name` (`character(1)`)
: Target variable name.

`n_intervals` (`integer(1)`)
: Intervals for numeric ALE.

`predict_fun` (`function()`)
: `function(model, data)` returning predictions.

`order_method` (`character(1)`)
: Categorical order: `"mds"`, `"pca"`, `"random"`, `"raw"`.

`ale_engine` (`character(1)`)
: ALE backend selected after `$fit()`: `"cpp"` or `"r"`.

`categorical_split` (`character(1)`)
: Categorical split mode for ALE trees: `"ordered_prefix"` or `"exhaustive"`.

`max_exhaustive_levels` (`integer(1)`)
: Maximum observed levels allowed for exhaustive categorical split search.

`effect` (`list()` or `NULL`)
: Cached ALE effect used when `$plot()` omits `effect`.

Methods

Public methods:

- `AleStrategy$new()`
- `AleStrategy$preprocess()`
- `AleStrategy$node_transform()`
- `AleStrategy$heterogeneity()`
- `AleStrategy$get_child_objectives()`
- `AleStrategy$find_best_split()`
- `AleStrategy$plot()`
- `AleStrategy$fit()`
- `AleStrategy$clean()`
- `AleStrategy$clone()`

Method `new()`:

Usage:

```
AleStrategy$new(
  categorical_split = "ordered_prefix",
  max_exhaustive_levels = 12L
)
```

Method `preprocess()`:

Usage:

```
AleStrategy$preprocess(
  model,
  effect = NULL,
  data,
  target_feature_name,
  n_intervals,
  feature_set = NULL,
  split_feature = NULL,
  predict_fun = NULL,
  order_method = "raw",
  ale_engine = c("auto", "cpp", "r")
)
```

Method `node_transform()`:

Usage:

```
AleStrategy$node_transform(Y, idx, grid = NULL, is_child = FALSE)
```

Method `heterogeneity()`**:**

Usage:

```
AleStrategy$heterogeneity(Y)
```

Method `get_child_objectives()`**:**

Usage:

```
AleStrategy$get_child_objectives(
  Y,
  split_info,
  idx_left,
  idx_right,
  grid_left,
  grid_right
)
```

Method `find_best_split()`**:**

Usage:

```
AleStrategy$find_best_split(Z, Y, min_node_size, n_quantiles)
```

Method `plot()`**:**

Usage:

```
AleStrategy$plot(
  tree,
  effect = NULL,
  data,
  target_feature_name,
  depth = NULL,
  node_id = NULL,
  features = NULL,
  show_plot = TRUE,
  show_point = TRUE,
  mean_center = TRUE,
  ...
)
```

Method `fit()`**:**

Usage:

```
AleStrategy$fit(
  tree,
  model = NULL,
  effect = NULL,
  data,
  target_feature_name,
  n_intervals = 10,
  feature_set = NULL,
  split_feature = NULL,
  predict_fun = NULL,
```

```
  order_method = "raw",
  ale_engine = c("auto", "cpp", "r"),
  categorical_split = NULL,
  max_exhaustive_levels = NULL,
  ...
)
```

Method `clean()`**:**

Usage:

```
AleStrategy$clean()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

`calculate_ale_heterogeneity_cpp`

Examples

```
ale_strat = AleStrategy$new()
ale_strat$categorical_split
```

`GadgetTree` *GadgetTree: Regional feature-effect tree*

Description

Wrapper for effect-based trees: given a strategy (pd/ale), fits tree via `$fit()`, plots effects via `$plot()`, extracts splits via `$extract_split_info()`. Delegates all effect logic to the strategy.

Format

R6::R6Class object.

Details

This class manages the overall tree structure and delegates effect-specific operations (such as splitting and plotting) to the provided strategy object. It is the main entry point for fitting and visualizing effect-based decision trees in the xplaineff package.

Construction

```
t = GadgetTree$new(strategy, n_split = 2, impr_par = 0.1, min_node_size = 10, n_quantiles = NULL)
```

- `strategy` :: (AleStrategy | PdStrategy)
  Effect strategy object.
- `n_split` :: `integer(1)`
  Maximum split depth, counted as splits along any root-to-leaf path.
- `impr_par` :: `numeric(1)`
  Improvement threshold.
- `min_node_size` :: `integer(1)`
  Minimum samples per node.
- `n_quantiles` :: `integer(1)` or `NULL`
  Number of quantile cut points for numeric split candidates; `NULL` = use all unique values.

Public fields

`strategy` (PdStrategy | AleStrategy)
: Effect-specific logic.

`root` (Node)
: Root node.

`n_split` (`integer(1)`)
: Maximum split depth, counted as splits along any root-to-leaf path.

`impr_par` (`numeric(1)`)
: Improvement threshold.

`min_node_size` (`integer(1)`)
: Minimum samples per node.

`n_quantiles` (`integer(1)` or `NULL`)
: Number of quantile cut points for numeric split candidates.

`split_benchmark` (`list()`)
: Internal split timing records.

`tree_list_cache` (`list()` or `NULL`)
: Cached depth-based tree list; invalidated on `$fit()`.

Methods

Public methods:

- `GadgetTree$new()`
- `GadgetTree$fit()`
- `GadgetTree$plot()`
- `GadgetTree$plot_tree_structure()`
- `GadgetTree$extract_split_info()`
- `GadgetTree$get_tree_list()`
- `GadgetTree$clone()`

Method `new()`**:**

Usage:

```
GadgetTree$new(
  strategy,
  n_split = 2,
  impr_par = 0.1,
  min_node_size = 10,
  n_quantiles = NULL
)
```

Method `fit()`**:**

Usage:

```
GadgetTree$fit(
  data,
  target_feature_name,
  feature_set = NULL,
  split_feature = NULL,
  ...
)
```

Method `plot()`**:**

Usage:

```
GadgetTree$plot(
  data,
  target_feature_name,
  effect = NULL,
  depth = NULL,
  node_id = NULL,
  features = NULL,
  ...
)
```

Method `plot_tree_structure()`**:**

Usage:

```
GadgetTree$plot_tree_structure(
  label_wrap_width = 34L,
  node_spread_x = 1.55,
  node_spread_y = 1.12
)
```

Method `extract_split_info()`**:**

Usage:

```
GadgetTree$extract_split_info(include_timing = FALSE)
```

Method `get_tree_list()`**:**

Usage:

```
GadgetTree$get_tree_list()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
tree = GadgetTree$new(strategy = PdStrategy$new(), n_split = 2L)
tree$n_split
```

`PdStrategy` *PdStrategy: Generalized Additive Decomposition Based on PD Effects*

Description

PD-based effect strategy (inherits from EffectStrategy). Given effect or model and data, preprocesses to Z/Y/grid; mean-centers effects per node; computes sum-of-variances heterogeneity; finds best split via C++; fits tree and plots PD/ICE. Character feature columns are coerced to `factor` before ICE/PD computation so they match split-matrix treatment and learner conventions (same as `prepare_split_data_common`).

Format

R6::R6Class object inheriting from EffectStrategy.

Details

This class is used internally by the GadgetTree framework to implement partial dependence tree growing, splitting, and visualization. It is not intended to be used directly by end users, but can be instantiated for advanced customization.

Construction

```
s = PdStrategy$new(categorical_split = "one_vs_rest")
```

Super class

`::EffectStrategy` -> `PdStrategy`

Public fields

`effect` (`list()` or `R6` or `NULL`)
: Cached PD/ICE effect used when `$plot()` omits `effect`.

`categorical_split` (`character(1)`)
: Categorical split mode for PD trees: `"one_vs_rest"` or `"exhaustive"`.

`max_exhaustive_levels` (`integer(1)`)
: Maximum observed levels allowed for exhaustive categorical split search.

Methods

Public methods:

- PdStrategy$new()
- PdStrategy$preprocess()
- PdStrategy$node_transform()
- PdStrategy$heterogeneity()
- PdStrategy$get_child_objectives()
- PdStrategy$find_best_split()
- PdStrategy$plot()
- PdStrategy$fit()
- PdStrategy$clean()
- PdStrategy$clone()

Method `new()`:

Usage:

```
PdStrategy$new(categorical_split = "one_vs_rest", max_exhaustive_levels = 12L)
```

Method `preprocess()`:

Usage:

```
PdStrategy$preprocess(
  effect,
  data,
  target_feature_name = NULL,
  feature_set = NULL,
  split_feature = NULL
)
```

Method `node_transform()`:

Usage:

```
PdStrategy$node_transform(Y, idx, grid, is_child = FALSE)
```

Method `heterogeneity()`:

Usage:

```
PdStrategy$heterogeneity(Y)
```

Method `get_child_objectives()`:

Usage:

```
PdStrategy$get_child_objectives(
  Y,
  split_info,
  idx_left,
  idx_right,
  grid_left,
  grid_right
)
```

Method `find_best_split()`**:**

Usage:

```
PdStrategy$find_best_split(Z, Y, min_node_size, n_quantiles)
```

Method `plot()`**:**

Usage:

```
PdStrategy$plot(
  tree,
  effect = NULL,
  data,
  target_feature_name,
  depth = NULL,
  node_id = NULL,
  features = NULL,
  ...
)
```

Method `fit()`**:**

Usage:

```
PdStrategy$fit(
  tree,
  effect = NULL,
  model = NULL,
  data,
  target_feature_name,
  feature_set = NULL,
  split_feature = NULL,
  predict_fun = NULL,
  n_grid = 20L,
  pd_engine = c("auto", "cpp", "r"),
  categorical_split = NULL,
  max_exhaustive_levels = NULL,
  ...
)
```

Method `clean()`**:**

Usage:

```
PdStrategy$clean()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
pd_strat = PdStrategy$new()
pd_strat$categorical_split
```

Index

AleStrategy, *3*, 4, *4*, *8*

calculate_ale_heterogeneity_cpp, *7*

EffectStrategy, *4*, *10*

GadgetTree, *3*, *4*, 7

PdStrategy, *3*, *4*, *8*, 10

R6::R6Class, *4*, *7*, *10*

xplaineff *(*xplaineff-package*)*, 2
xplaineff-package, 2