Package 'xplaineff'

September 14, 2026

Type Package

Title Decomposing Global Feature Effects Based on Feature Interactions

Version 0.1.0

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), iml (>= 0.11.4), mlr3, mlr3learners, ranger, ISLR2, rpart, withr, xgboost

Config/testthat/edition 3

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]

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

Repository CRAN

Date/Publication 2026-09-14 15:00:02 UTC

Contents

`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 Nonlinear 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()`**:** Create an AleStrategy instance (calls `super$initialize("ale")`).

Usage:

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

Arguments:

`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.

Method `preprocess()`**:** Preprocess to Z and Y via `prepare_split_data_ale`.

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")
)
```

Arguments:

`model` (‘any‘)
: Fitted model.

`effect` (‘list()‘ or ‘NULL‘)
: Reserved for future extension. Currently unsupported.

`data` (‘data.frame()‘ or ‘data.table()‘)
: Data.

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

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

`feature_set` (‘character()‘ or ‘NULL‘)
Features for ALE; `NULL` = all.

`split_feature` (‘character()‘ or ‘NULL‘)
Features for splitting; `NULL` = all.

`predict_fun` (‘function()‘ or ‘NULL‘)
Prediction function.

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

`ale_engine` (‘character(1)‘)
ALE engine: `"auto"`, `"cpp"`, or `"r"`.

Returns: (‘list()‘)
`Z`: split features; `Y`: ALE effect data.tables.

Method `node_transform()`**:** Subset ALE by node indices; handle single-interval and categorical.

Usage:

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

Arguments:

`Y` (‘list()‘)
ALE effect list from `calculate_ale`.

`idx` (‘integer()‘)
Row indices in the node.

`grid` (‘list()‘ or ‘NULL‘)
Ignored for ALE; required by interface.

`is_child` (‘logical(1)‘)
Whether the current node is a child node.

Returns: (‘list()‘)
Transformed ALE data.tables.

Method `heterogeneity()`**:** Compute ALE heterogeneity via `calculate_ale_heterogeneity_cpp`.

Usage:

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

Arguments:

`Y` (‘list()‘)
ALE effect list from `calculate_ale`.

Returns: (‘numeric()‘)
Heterogeneity per feature.

Method `get_child_objectives()`**:** Compute left/right child objective values from split result. For ALE, extracts from split_info (computed during sweep).

Usage:

```
AleStrategy$get_child_objectives(
  Z,
  Y,
```

```
  split_info,
  idx_left,
  idx_right,
  grid_left,
  grid_right
)
```

Arguments:

`Z` (‘data.frame()‘ or ‘data.table()‘)
: Split features.

`Y` (‘list()‘)
: ALE effect from `calculate_ale`.

`split_info` (‘list()‘)
: Split metadata.

`idx_left`, `idx_right` (‘integer()‘)
: Child row indices.

`grid_left`, `grid_right` (‘list()‘)
: Child grids.

Returns: (‘list()‘)
`left_objective_value_j`, `right_objective_value_j`, `left_objective_value`, `right_objective_value`.

Method `find_best_split()`**:** Find best split via `search_best_split_ale`.

Usage:

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

Arguments:

`Z` (‘data.frame()‘ or ‘data.table()‘)
: Split features.

`Y` (‘list()‘)
: ALE effect from `calculate_ale`.

`min_node_size` (‘integer(1)‘)
: Minimum node size.

`n_quantiles` (‘integer(1)‘ or ‘NULL‘)
: Quantile candidates for numeric.

Returns: (‘list()‘ or ‘data.frame()‘)
Best split info: `split_feature`, `split_point`, etc.

Method `plot()`**:** Plot ALE curves via `plot_tree_ale`.

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,
  ...
)
```

Arguments:

`tree` (‘list()‘)
: Depth-based list of Node objects.

`effect` (‘list()‘ or ‘NULL‘)
: ALE effect; `NULL` = use cached `effect`.

`data` (‘data.frame()‘ or ‘data.table()‘)
: Data.

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

`depth` (‘integer()‘ or ‘NULL‘)
: Depths to plot.

`node_id` (‘integer()‘ or ‘NULL‘)
: Node IDs to plot.

`features` (‘character()‘ or ‘NULL‘)
: Features to include.

`show_plot`, `show_point`, `mean_center` (‘logical(1)‘)
: Plot options.

`...` Passed to `plot_tree_ale`.

Returns: (‘list()‘)
Nested list (depth -> node -> patchwork).

Method `fit()`**:** Fit tree: preprocess, create root, split recursively.

Usage:

```
AleStrategy$fit(
  tree,
  model,
  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,
  ...
)
```

Arguments:

`tree` (‘GadgetTree‘)
Tree instance.

`model` (‘any‘)
Fitted model.

`effect` (‘list()‘ or ‘NULL‘)
Reserved for future extension. Currently unsupported.

`data` (‘data.frame()‘ or ‘data.table()‘)
Data.

`target_feature_name` (‘character(1)‘)
Target name.

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

`feature_set`, `split_feature` (‘character()‘ or ‘NULL‘)
Feature subsets.

`predict_fun` (‘function()‘ or ‘NULL‘)
Prediction function.

`order_method` (‘character(1)‘)
Categorical order.

`ale_engine` (‘character(1)‘)
ALE engine: `"auto"`, `"cpp"`, or `"r"`.

`categorical_split` (‘character(1)‘ or ‘NULL‘)
Categorical split mode for ALE trees; `NULL` keeps the current strategy setting.

`max_exhaustive_levels` (‘integer(1)‘ or ‘NULL‘)
Maximum observed levels allowed for exhaustive categorical split search; `NULL` keeps the current strategy setting.

`...` Ignored.

Returns: (‘GadgetTree‘)
The tree, invisibly.

Method `clean()`**:** Sets `data` and `model` to `NULL` to free memory after fitting. `effect` is intentionally retained because `plot()` requires it post-fit.

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
```

`node_transform_ale` *Node Transform ALE*

Description

Subsets ALE effect data to the current node's row indices and recomputes per-interval statistics. When `is_child` is `TRUE`, forces `d_l = 0` for any feature whose values are constant in this node (single unique value).

Usage

```
node_transform_ale(Y, idx, is_child = FALSE)
```

Arguments

`Y`	('list()') ALE effect data per feature.
`idx`	('integer()') Sample indices in the current node.
`is_child`	('logical(1)') Whether the current node is a child node; `FALSE` skips constant-feature zeroing.

Value

('list()')
Transformed ALE effects per feature.

`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()`**:** Create a PdStrategy instance (calls `super$initialize("pd")`).

Usage:

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

Arguments:

`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.

Method `preprocess()`**:** Preprocess to Z, Y, grid via `prepare_split_data_pd`.

Usage:

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

Arguments:

`effect` (R6 or 'list()')
: Effect object (e.g. FeatureEffect).

`data` ('data.frame()' or 'data.table()')
: Data.

`target_feature_name` ('character(1)' or 'NULL')
: Target variable name.

`feature_set` ('character()' or 'NULL')
: Features for effect; `NULL` = all.

`split_feature` ('character()' or 'NULL')
: Features for splitting; `NULL` = all.

Returns: ('list()')
`Z`, `Y`, `grid`.

Method `node_transform()`**:** Subset and mean-center via `re_mean_center_ice_cpp`.

Usage:

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

Arguments:

`Y` ('list()')
: Effect matrices per feature.

`idx` ('integer()')
: Sample indices in the node.

`grid` ('list()')
: Feature grids; required for PD.

`is_child` ('logical(1)')
: Ignored for PD; kept for API parity with `AleStrategy`.

Returns: ('list()')
Mean-centered effect matrices.

Method `heterogeneity()`**:** Compute heterogeneity via `node_heterogeneity`.

Usage:

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

Arguments:

`Y` ('list()')
: Effect matrices.

Returns: (‘numeric()‘)
Heterogeneity per feature.

Method `get_child_objectives()`**:** Compute left/right child objective values via node_transform and heterogeneity.

Usage:

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

Arguments:

`Z` (‘data.frame()‘ or ‘data.table()‘)
Split features.

`Y` (‘list()‘)
Effect matrices.

`split_info` (‘list()‘)
Split metadata.

`idx_left`, `idx_right` (‘integer()‘)
Child row indices.

`grid_left`, `grid_right` (‘list()‘)
Child grids.

Returns: (‘list()‘)
`left_objective_value_j`, `right_objective_value_j`, `left_objective_value`, `right_objective_value`.

Method `find_best_split()`**:** Find best split via `search_best_split_cpp`.

Usage:

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

Arguments:

`Z` (‘data.frame()‘ or ‘data.table()‘)
Split features.

`Y` (‘list()‘)
Effect matrices.

`min_node_size` (‘integer(1)‘)
Minimum node size.

`n_quantiles` (‘integer(1)‘ or ‘NULL‘)
Quantile candidates.

Returns: (‘data.frame()‘ or ‘list()‘)
Best split info.

Method `plot()`**:** Plot PD/ICE tree via `plot_tree_pd`.

Usage:

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

Arguments:

`tree` (‘list()‘)
: Depth-based list of Node objects.

`effect` (R6 or ‘list()‘ or ‘NULL‘)
: Effect object.

`data` (‘data.frame()‘)
: Data.

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

`depth` (‘integer()‘ or ‘NULL‘)
: Depths to plot.

`node_id` (‘integer()‘ or ‘NULL‘)
: Node IDs to plot.

`features` (‘character()‘ or ‘NULL‘)
: Features to plot.

`...` Plot arguments.

Returns: (‘list()‘)
Nested list (depth -> node -> patchwork).

Method `fit()`**:** Fit tree: preprocess, create root, split recursively.

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,
  ...
)
```

Arguments:

`tree` (‘GadgetTree‘)
Tree instance.

`effect` (R6 or ‘list()‘ or ‘NULL‘)
Optional precomputed effect object.

`model` (‘any‘)
Fitted model for internal PD/ICE computation.

`data` (‘data.frame()‘)
Data.

`target_feature_name` (‘character(1)‘)
Target name.

`feature_set, split_feature` (‘character()‘ or ‘NULL‘)
Feature subsets.

`predict_fun` (‘function()‘ or ‘NULL‘)
Optional prediction function.

`n_grid` (‘integer(1)‘)
Number of grid points for numeric features.

`pd_engine` (‘character(1)‘)
When computing ICE/PD from `model`: `"auto"`, `"cpp"` (column-wise stacked `newdata`, xplaineff-style), or `"r"` (`data.table::rbindlist`).

`categorical_split` (‘character(1)‘ or ‘NULL‘)
Categorical split mode for PD trees; `NULL` keeps the current strategy setting.

`max_exhaustive_levels` (‘integer(1)‘ or ‘NULL‘)
Maximum observed levels allowed for exhaustive categorical split search; `NULL` keeps the current strategy setting.

`...` Ignored.

Returns: (‘GadgetTree‘)
The tree, invisibly.

Method `clean()`**:** Drops `tree_ref`; effect cache is intentionally retained when present.

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
```

prepare_plot_data_ale *Prepare ALE Plot Data for One or More Nodes*

Description

Given effect (from calculate_ale), idx (row indices or list of such), features, mean_center: subsets ALE rows by idx; calls mean_center_ale per feature for cumulative and optional centering. Returns named list of mean_effect data.tables (or nested list if idx is list).

Usage

```
prepare_plot_data_ale(
  effect,
  idx = NULL,
  features = names(effect),
  mean_center = TRUE
)
```

Arguments

effect	('list()') List returned by calculate_ale().
idx	('integer()' or 'list()' or 'NULL') Row indices (node subset); list of such vectors; or NULL for root.
features	('character()') Features to include (default: all in effect).
mean_center	('logical(1)') Whether to mean-center ALE curves.

Details

Rows are subset with effect[[feat]][row_id %in% idx] when idx is non-NULL.

mean_center_ale() builds plot grids from aggregated intervals: means sample-wise d_l within each (interval_index, x_left, x_right) group (delta_aggr), cumulates, then optionally subtracts a weighted scalar f_j0. Sample-wise d_l == 0 is mapped to NA before aggregation so exact zeros do not enter group means.

For categorical features, each row of mean_effect corresponds to one row of delta_aggr; x_grid uses as.character(delta_aggr$x_left), so the number of plotted points follows the number of distinct aggregated intervals after subsetting (not always one row per factor level). Factor x_grid still carries full levels(feat_val) for axis ordering.

Value

('list()')
Named list of mean_effect data.tables per feature; nested if idx is list.

Index