Package ‘ROOT’

November 4, 2025

Title Identifying Underrepresented Subpopulations With Interpretable Trees

Version 0.0.0.9000

Description ROOT (Rashomon set of Optimal Trees) is a framework for learning interpretable binary weight functions represented as sparse decision trees. It constructs a Rashomon set of near-optimal trees and extracts a characteristic tree to summarize patterns. Given trial and target data, the package identifies trial subpopulations that contribute disproportionately to the variance of the target treatment-effect estimate (underrepresented groups).

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Suggests gbm,
mlbench,
testthat (>= 3.0.0),
knitr,
rmarkdown,
ragg

Config/testthat/edition 3

Imports MASS,
rpart,
stats,
withr,
rpart.plot

VignetteBuilder knitr

Contents

characterize_tree 2
characterizing_underrep 3
choose_feature 4
estimate 5
estimate_dml 6
estimate_dml_single 7
estimate_single 7
gen_S 8

gen_T 8
gen_XY 9
get_data 10
loss 11
loss_from_objective 11
midpoint 12
objective_default 12
objective_if 13
reduce_weight 13
ROOT 14
split_node 15
stratified_kfold 17
summary.characterizing_underrep 17
summary.ROOT 18
train 19
train_single 20

Index **21**

`characterize_tree` *Fit a shallow decision tree to characterize learned weights* `w`

Description

Trains a classification tree on the covariates `X` to predict the binary membership `w`. This provides an interpretable summary of how the weighted subgroup can be distinguished by `X`.

Usage

```
characterize_tree(X, w, max_depth = 3)
```

Arguments

`X`	A data frame of covariates (features).
`w`	A binary vector (0/1 or a factor with two levels) indicating class membership for each observation (e.g., whether an observation is in the selected subgroup).
`max_depth`	Integer, the maximum tree depth (default 3).

Details

The tree is grown using the Gini index (classification) and is not pruned (complexity parameter `cp = 0`), relying solely on `max_depth` to control complexity. This mirrors the default behavior of scikit-learn's `DecisionTreeClassifier(max_depth=...)`. If `w` is not already a factor, it will be converted internally. The tree's rules can be interpreted to understand which covariates (and what splits) best separate the two classes defined by `w`.

Value

An `rpart` object representing the fitted decision tree.

characterizing_underrep *Characterize under-represented subgroups (wraps ROOT)*

Description

Combines an RCT (S=1) and a target dataset (S=0), calls `ROOT()` to learn a binary selector `w(x)`, and (optionally) renders an annotated tree that highlights represented (w=1) vs underrepresented (w=0) leaves.

Usage

```
characterizing_underrep(
  DataRCT,
  covariate_DataRCT,
  treatment_DataRCT,
  outcome_DataRCT,
  DataTarget,
  covariate_DataTarget,
  leaf_proba = 0.25,
  seed = 123,
  num_trees = 10,
  vote_threshold = 2/3,
  explore_proba = 0.05,
  feature_est = "Ridge",
  feature_est_args = list(),
  top_k_trees = FALSE,
  k = 10,
  cutoff = "baseline",
  verbose = FALSE,
  objective_fn = objective_default,
  loss_fn = NULL,
  root_plot_tree = TRUE,
  root_plot_args = list(type = 2, extra = 109, under = TRUE, faclen = 0, tweak = 1.1,
    fallen.leaves = TRUE, box.palette = c("pink", "palegreen3"), shadow.col = c("gray"),
    branch.lty = 3, main = "Final Characterized Tree from Rashomon Set"),
  plot_underrep = TRUE,
  keep_threshold = 0.5,
  lX_threshold = NULL,
  plot_main = "Underrepresented Population Characterization Tree"
)
```

Arguments

`DataRCT`	data.frame with trial data; must include `treatment_DataRCT`, `outcome_DataRCT`, and the covariates named in `covariate_DataRCT`.
`covariate_DataRCT`	character vector of covariate column names in `DataRCT`.
`treatment_DataRCT`	single string: treatment column name in `DataRCT` (0/1).

`outcome_DataRCT`	single string: outcome column name in `DataRCT`.
`DataTarget`	data.frame with target-population covariates (no treatment/outcome required).
`covariate_DataTarget`	character vector of covariate column names in `DataTarget`.
`leaf_proba`, `seed`, `num_trees`, `vote_threshold`, `explore_proba`, `feature_est`, `feature_est_args`, `top_k_trees`, `k`, `cutoff`, `verbose`	Passed to `ROOT()`.
`objective_fn`	Objective function for ROOT (default `objective_default`).
`loss_fn`	Optional loss micro-evaluator; if `NULL`, ROOT will wrap `objective_fn`.
`root_plot_tree`	Logical; pass-through to `ROOT(plot_tree=...)`.
`root_plot_args`	Optional list passed to `ROOT(plot_tree_args=...)`.
`plot_underrep`	Logical; if `TRUE`, draws an annotated represented/underrepresented tree.
`keep_threshold`, `lX_threshold`	Kept for API compatibility (unused).
`plot_main`	Title for the annotated plot.

Value

A `characterizing_underrep` object with

`root`	the fitted `ROOT` object
`combined`	stacked RCT+Target data used for fitting
`leaf_summary`	data.frame with terminal rules and sizes (if a summary tree exists)
`tree_plot_root`	recorded plot of the ROOT summary tree (if produced)
`tree_plot_underrep`	recorded plot of the annotated underrep tree (if produced)

`choose_feature` *Randomly choose a split feature based on provided probabilities*

Description

Given a probability distribution over features (and possibly a "leaf" option), selects one feature at random according to those probabilities.

Usage

```
choose_feature(split_feature, depth)
```

Arguments

`split_feature`	A named numeric vector of feature selection probabilities. Names should correspond to feature IDs (and may include a special "leaf" entry).
`depth`	Current tree depth (an integer, used for parity with Python implementation but not affecting probabilities in this implementation).

Value

A single feature name (or "leaf") chosen randomly according to the provided probability weights.

Note

The factor $2^{(0*depth/4)}$ present in the code is effectively 1 (no effect on the first element's weight) and is included only for parity with an equivalent Python implementation. All probabilities are normalized to sum to 1 before sampling.

`estimate` *Compute pseudo-outcome components (a, b) and their product (v)*

Description

Using the outputs of the nuisance models, computes intermediate values for the treatment effect estimation via inverse probability weighting (IPW) for the Average Treatment Effect (ATE) in trial sample.

Usage

```
estimate(testing_data, outcome, treatment, sample, pi, pi_m, e_m)
```

Arguments

`testing_data`	A data frame of test data (or evaluation data) containing at least the columns for outcome, treatment, and sample indicators.
`outcome`	Name of the outcome column in `testing_data`.
`treatment`	Name of the treatment column in `testing_data` (0/1).
`sample`	Name of the sample indicator column in `testing_data` (0/1).
`pi`	Numeric scalar, the estimated $P(S = 1)$ (prevalence) from the training data.
`pi_m`	A fitted model (e.g., `glm`) for $P(S = 1 \mid X)$; typically from `train()`.
`e_m`	A fitted model (`glm`) for $P(Tr = 1 \mid X, S = 1)$; typically from `train()`.

Details

Specifically, it computes:

- `a`: IPW-adjusted outcome difference, $a_i = S_i \left(\frac{Tr_i Y_i}{p_{t1|x,i}} - \frac{(1-Tr_i)Y_i}{1-p_{t1|x,i}} \right)$.
- `b`: Overlap weight factor, $b_i = \frac{1}{\ell(X_i)}$, where $\ell(X) = \frac{P(S=1|X)/\pi}{P(S=0|X)/(1-\pi)}$.
- `v`: The pseudo-outcome, defined as $v_i = a_i \times b_i$.

The predicted probabilities from `pi_m` and `e_m` are constrained to `[1e-8, 1-1e-8]` to avoid instability (extremely small or large probabilities are clamped). If the provided `pi` is 0 or 1 (indicating no variation in sample inclusion in training), the computation is undefined and an error will be thrown. Ensure that `pi_m` and `e_m` correspond to models trained on compatible data (same covariates) for accurate predictions.

Value

A list with numeric vectors:

`v`	Pseudo-outcome values for each observation (numeric vector length = nrow(testing_data)).
`a`	Intermediate "IPW-adjusted outcome" values (same length as v).
`b`	Overlap weight factors (same length as v).

See Also

`train` for obtaining `pi`, `pi_m`, and `e_m`; `estimate_dml` for cross-fitted estimation.

`estimate_dml` *Cross-fitted estimation of pseudo-outcomes (Double ML)*

Description

Trains nuisance models on each training fold and computes pseudo-outcomes on the corresponding test fold, then aggregates results. Returns only what is needed downstream: the pseudo-outcome table and the aligned evaluation data.

Usage

```
estimate_dml(data, outcome, treatment, sample, crossfit = 5)
```

Arguments

`data`	A data frame containing at least the outcome, treatment, and sample indicator columns.
`outcome`	Name of the outcome column.
`treatment`	Name of the treatment column (0/1).
`sample`	Name of the sample indicator column (0/1).
`crossfit`	Integer number of folds for cross-fitting (>= 2).

Value

A list with:

`df_v`	Data frame with one row per kept observation (indexed by `primary_index`), containing: `te` (pseudo-outcome v), `a`, `b`, and squared deviations `te_sq`, `a_sq`. Only S==1 rows with finite values are kept.
`data2`	Subset of original `data` corresponding to `df_v$primary_index`.

Note

Rows with infinite or undefined weights (e.g., where the predicted propensity scores were 0 or 1) are removed from `df_v` (and the corresponding rows in `data2`). The `primary_index` in `df_v` corresponds to the row index in the original `data`. Squared deviation columns (`te_sq`, `a_sq`) are centered around the mean of `te` and `a` for the `S==1` group.

`estimate_dml_single` *Cross-fitted Double ML (single-sample mode)*

Description

Runs K-fold cross-fitting to produce pseudo-outcomes for ATE estimation when no sample-membership indicator is available (or has no variation). On each fold, a treatment propensity model is trained on the training split and used to compute pseudo-outcomes on the test split. Results are combined and centered to create variance proxies.

Usage

```
estimate_dml_single(data, outcome, treatment, crossfit = 5)
```

Arguments

`data`	A data frame containing `outcome`, `treatment`, and covariates.
`outcome`	Name of the outcome column.
`treatment`	Name of the binary treatment indicator column (0/1).
`crossfit`	Integer number of folds for cross-fitting (default 5; must be ≥ 2).

Value

A list with:

`df_v`	Data frame with one row per kept observation, containing: `te` (pseudo-outcome v), `a`, `b` (all ones), and centered squares `te_sq`, `a_sq`, plus `primary_index` mapping back to the original `data` rows.
`data2`	Subset of `data` corresponding to `df_v$primary_index` (i.e., rows kept after cross-fitting and finite checks).

`estimate_single` *Compute single-sample pseudo-outcomes*

Description

Computes the single-sample pseudo-outcome components for ATE-style estimation: $a_i = T_i Y_i / e_i - (1 - T_i) Y_i / (1 - e_i)$, with $v_i = a_i$ and $b_i \equiv 1$. The treatment propensity e_i is predicted from a supplied model.

Usage

```
estimate_single(testing_data, outcome, treatment, e_m)
```

Arguments

`testing_data`	A data frame containing at least `outcome`, `treatment`, and covariates (the latter are used for prediction).
`outcome`	Name of the outcome column (character).
`treatment`	Name of the binary treatment indicator column (0/1).
`e_m`	A fitted `glm`(binomial) model for $P(T = 1 \mid X)$; typically the result of `train_single`.

Value

A list with numeric vectors of length `nrow(testing_data)`:

`v`	Pseudo-outcome values (equal to `a` in single-sample mode).
`a`	IPW-adjusted outcome contrast.
`b`	Vector of ones (no sample-overlap weighting in single-sample mode).

`gen_S` *Generate sample indicator* `S` *~ Bernoulli(plogis(a))*

Description

Generates a binary sample inclusion indicator `S` for each observation, using a logistic model influenced by a rectangular region in the first two covariates (`X0` and `X1`).

Usage

```
gen_S(X, seed = NULL)
```

Arguments

`X`	A data frame of covariates (must contain at least columns `X0` and `X1`).
`seed`	Optional numeric seed for RNG. If provided, `set.seed(seed + 1)` is invoked for reproducibility. If `NULL` (default), no specific seed is set.

Details

The inclusion probability is defined as $p = \text{plogis}(a)$, where $a = 0.25 - 2 * I\{X0, X1 \text{ in region } (0.5, 1)\}$. In other words, observations for which both `X0` and `X1` lie in (0.5, 1) have a lower odds of being included (due to a negative contribution in the linear predictor). This mirrors a scenario where a specific region in feature space is under-sampled. If a seed is set, it uses `seed + 1` to differentiate from other generators.

Value

A data frame with a single column `S` of 0/1 values indicating inclusion (1) or exclusion (0).

`gen_T` *Generate treatment indicator* `Tr` *~ Bernoulli(pi)*

Description

Assigns a treatment indicator for each observation, combining an experimental design for included samples (S==1) and an observational assignment for excluded samples (S==0).

Usage

```
gen_T(X, S, seed = NULL)
```

Arguments

`X`	A data frame of covariates.
`S`	A data frame with column `S` (0/1 indicating sample inclusion for each observation).
`seed`	Optional numeric seed for RNG. If provided, `set.seed(seed - 1)` is used. Default `NULL` means no explicit seeding.

Details

For observations with `S==1` (in sample), treatment is assigned with probability `0.5` (mimicking a randomized experiment). For those with `S==0` (out of sample), treatment probability is $\text{plogis}(X0)$, i.e., it increases with the value of covariate `X0`. The overall assignment probability for each observation is $\pi_i = S_i * 0.5 + (1 - S_i) * \text{plogis}(X0_i)$. If a seed is provided, an offset `seed - 1` is used to differentiate from other generation steps.

Value

A list with two elements:

`Tr`	A data frame with a single column `Tr` (treatment assignments 0/1 for each observation).
`pi`	A numeric vector of length equal to number of observations, giving the treatment probability used for each observation.

`gen_XY` *Generate covariates* `X` *and potential outcomes (*`Y0`*,* `Y1`*)*

Description

Simulates a regression problem (Friedman #1) and defines a treatment effect. Uses `mlbench.friedman1` to generate `X` features and a baseline outcome `Y0`. The treatment potential outcome `Y1` is defined as `Y1 = Y0 + log(Y0 + 1)`, introducing a heterogeneous treatment effect.

Usage

```
gen_XY(n = 1000, seed = NULL)
```

Arguments

`n`	Integer or numeric. Number of observations to simulate (must be positive).
`seed`	Optional. Single numeric value for RNG seed. If provided, a global seed is set for reproducibility. If `NULL` (default), no seed is set (results will vary on each run).

Details

The `mlbench.friedman1` function from the **mlbench** package is used to generate 10 independent continuous features and a baseline outcome `Y0` with additive noise. The treatment outcome `Y1` is defined by adding a non-linear term `log(Y0 + 1)` to the baseline. If a seed is specified, the random number generator state is reset at the start of the function (which affects other random operations).

Value

A list with two components:

X	A data frame of simulated covariates with columns `X0, X1, ...` up to `X(p-1)`.
Y	A data frame of potential outcomes with columns `Y0` (baseline outcome) and `Y1` (outcome under treatment).

---

get_data — *Convenience wrapper to generate a full simulated dataset*

Description

Generates covariates, sample inclusion, treatment assignments, and observed outcomes for a specified sample size. This wraps `gen_XY()`, `gen_S()`, and `gen_T()` in sequence.

Usage

```
get_data(n = 1000, seed = NULL)
```

Arguments

n	Integer or numeric. Sample size (number of observations to generate).
seed	Optional base seed for reproducibility. If provided, internal generators use offsets of this seed to ensure independent randomness. Default `NULL` means no explicit seeding.

Details

This function first generates covariates and potential outcomes with `gen_XY`. It then generates `S` (sample inclusion) and `Tr` (treatment assignment). The observed outcome `Yobs` is computed as $Y_{\text{obs}} = Tr * Y1 + (1 - Tr) * Y0$ for each observation.

Value

A list with two components:

data	A data frame of length `n` containing covariates `X0`, . . . , sample indicator `S`, treatment indicator `Tr`, and observed outcome `Yobs`.
Y	A data frame of length `n` containing the potential outcomes `Y0` and `Y1` for each observation.

Examples

```
sim <- get_data(n = 100, seed = 599)
dim(sim$data)    # should be 100 x (p + 3) columns (p features + S + Tr + Yobs)
head(sim$data$Yobs)  # observed outcomes
head(sim$Y)     # potential outcomes corresponding to those observations
```

`loss` *Evaluate splitting objective loss for a given weight assignment*

Description

Computes the approximate standard error (loss) of the treatment effect estimate under a hypothetical assignment of weights `w` for specified rows.

Usage

```
loss(val, indices, D)
```

Arguments

`val`	Numeric scalar (must be 0 or 1). The weight value to assign (0 = exclude, 1 = include).
`indices`	Indices or row names in `D` for which the weight should be set to `val`. Can be a numeric vector of row positions or a character vector of row names.
`D`	A data frame containing at least columns `vsq` (squared pseudo-outcome) and `w` (current weights).

Value

Numeric value representing the loss, defined as $\sqrt{\sum_i vsq_i * w_i \,/\, (\sum_i w_i)^2}$. Returns `Inf` if the denominator is 0 or if the result is not a number.

Note

This function mimics the behavior of numpy's `nan_to_num(..., nan=Inf)` by returning `Inf` when the computation is undefined (e.g., no weights selected). It is used internally to decide whether a proposed split improves the objective.

`loss_from_objective` *Backward/fast-path micro-evaluator adaptor*

Description

Wrap a global objective `objective_fn(D)` into a splitter-compatible loss function `loss_fn(val, indices, D)` by evaluating `objective_if` on a temporary copy of `D`.

Usage

```
loss_from_objective(objective_fn)
```

Arguments

`objective_fn`	Function of one argument `D` returning a numeric scalar to be minimized (e.g., `objective_default`).

Value

A function `loss_fn(val, indices, D)` suitable for use in `ROOT` and `split_node`. It sets `w = val` on `indices` (non-mutating), then returns `objective_fn(D)`.

`midpoint` *Compute the midpoint of a numeric vector*

Description

Calculates the midpoint defined as $(\max(X) + \min(X))/2$, ignoring any NA values.

Usage

```
midpoint(X)
```

Arguments

`X`	A numeric vector.

Value

A numeric scalar giving the midpoint of the finite values in `X`. If `X` is empty or has no finite values, `NA` is returned.

`objective_default` *Default objective: SE proxy of (W)TATE/PATE*

Description

Computes $\sqrt{\sum_i vsq_i * w_i / (\sum_i w_i)^2}$. Requires columns `vsq` and `w` in `D`. Minimize this. Supply your own function(D) -> scalar to use a different objective.

Usage

```
objective_default(D)
```

Arguments

`D`	data.frame with at least numeric columns `vsq` and `w`.

Value

numeric scalar objective value; `Inf` if undefined.

`objective_if` *Helper: evaluate objective after a hypothetical local change*

Description

Helper: evaluate objective after a hypothetical local change

Usage

```
objective_if(val, indices, D, objective_fn)
```

Arguments

`val`	0/1 assignment to apply
`indices`	integer or rownames to receive `val`
`D`	data.frame used by `objective_fn`
`objective_fn`	function(D)->scalar

Value

numeric scalar objective after the hypothetical change

`reduce_weight` *Reduce a feature's selection weight by half and renormalize*

Description

Lowers the probability weight of a given feature by 50%, and then re-normalizes the entire probability vector.

Usage

```
reduce_weight(fj, split_feature)
```

Arguments

`fj`	A feature name (character string) present in the names of `split_feature`.
`split_feature`	A named numeric vector of probabilities for features (as used in splitting).

Details

This is typically used when a particular feature split was rejected; the feature's probability is halved to reduce its chance of being chosen again immediately, encouraging exploration of other features. If `fj` is "leaf", its weight is also halved similarly.

Value

A numeric vector of the same length as `split_feature`, giving the updated probabilities that sum to 1.

ROOT *Ensemble of weighted trees (loss/objective-agnostic) and Rashomon selection*

Description

Builds multiple weighted trees, then identifies a "Rashomon set" of top-performing trees and aggregates their weight assignments by majority vote.

Usage

```
ROOT(
  data,
  outcome,
  treatment,
  sample,
  leaf_proba = 0.25,
  seed = NULL,
  num_trees = 10,
  vote_threshold = 2/3,
  explore_proba = 0.05,
  feature_est = "Ridge",
  feature_est_args = list(),
  top_k_trees = FALSE,
  k = 10,
  cutoff = "baseline",
  verbose = FALSE,
  objective_fn = objective_default,
  loss_fn = NULL,
  plot_tree = TRUE,
 plot_tree_args = list(type = 2, extra = 109, under = TRUE, faclen = 0, tweak = 1.1,
   fallen.leaves = TRUE, box.palette = c("pink", "palegreen3"), shadow.col = c("gray"),
    branch.lty = 3, main = "Final Characterized Tree from Rashomon Set")
)
```

Arguments

`data`	A data frame containing the dataset (must include outcome, treatment, sample indicator).
`outcome`	Name of the outcome column in `data`.
`treatment`	Name of the treatment indicator column (0/1) in `data`.
`sample`	Name of the sample indicator column (0/1) in `data`. Use NULL for single-sample SATE mode.
`leaf_proba`	Probability mass for the "leaf" option in each tree (default 0.25).
`seed`	Integer seed for reproducibility (default NULL).
`num_trees`	Number of trees to grow in the forest (default 10).
`vote_threshold`	Majority vote threshold in (0.5, 1] for final weight=1 (default 2/3).
`explore_proba`	Probability of exploration at leaves in each tree (default 0.05).

`feature_est`	"Ridge", "GBM", or a function(X, y, ...) returning a named, non-negative vector of importances; normalized to probabilities (default "Ridge").
`feature_est_args`	Named list of extra args for a user-supplied `feature_est` function.
`top_k_trees`	If TRUE, select top-k trees by objective; else use `cutoff` (default FALSE).
`k`	Number of top trees if `top_k_trees = TRUE` (default 10).
`cutoff`	If `top_k_trees = FALSE`, numeric cutoff or "baseline" (default "baseline"). With "baseline", the cutoff is computed by evaluating `objective_fn` on the state with all `w=1`.
`verbose`	If TRUE, prints 2 lines with (unweighted and weighted) estimate + SE. Default FALSE.
`objective_fn`	Function `function(D) -> numeric` that scores the **entire state** (minimize). Default `objective_default()` reproduces prior behavior (SE proxy of PATE/TATE using `vsq` and `w`).
`loss_fn`	Optional micro-evaluator `function(val, indices, D) -> numeric` that returns the objective after hypothetically setting `w=val` on `indices`. If `NULL`, ROOT wraps `objective_fn` via `loss_from_objective(objective_fn)`.
`plot_tree`	If TRUE, plots the characterized tree (default TRUE). Guarded by `interactive()`.
`plot_tree_args`	Named list forwarded to `rpart.plot::rpart.plot()`.

Value

S3 object of class "ROOT" with components: D_rash, D_forest, w_forest, rashomon_set, f, testing_data, tree_plot, estimate

---

`split_node` *Recursive split builder for weighted tree (internal function) Recursive split builder for weighted tree (internal function)*

Description

Recursively builds a weighted decision tree to optimize an objective function, using an exploration/exploitation trade-off. This function is internal and is used by `tree_opt() / ROOT()` to construct a single tree.

Usage

```
split_node(
  split_feature,
  X,
  D,
  parent_loss,
  depth,
  explore_proba = 0.05,
  choose_feature_fn = choose_feature,
  loss_fn = loss,
  reduce_weight_fn = reduce_weight,
  objective_fn = objective_default,
  max_depth = 8,
  min_leaf_n = 5,
  log_fn = function(...) {
 },
  max_rejects_per_node = 1000
)
```

Arguments

`split_feature`	Named numeric vector of feature selection probabilities (should include a "leaf" option).
`X`	Data frame of current observations (must include at least the feature chosen for splitting if applicable; may also include a working copy of weights `w`).
`D`	Data frame representing the global state (must include columns `w` for weights and `vsq` for squared pseudo-outcomes, with row names aligning to observations).
`parent_loss`	Numeric, the loss value of the parent node (used to decide if a split improves the objective).
`depth`	Integer, current depth of the node in the tree.
`explore_proba`	Numeric in `[0,1]`, probability of exploring the suboptimal assignment at a leaf (randomly flipping the chosen weight).
`choose_feature_fn`	Function to choose the next feature to split; default `choose_feature`.
`loss_fn`	Function to compute loss given a tentative weight assignment; if `NULL`, it is set via `loss_from_objective(objective_fn)`.
`reduce_weight_fn`	Function to adjust feature probabilities when a split is rejected; default `reduce_weight`.
`objective_fn`	Function that maps `D` $\to$ scalar objective; used for node summaries and to derive `loss_fn` when `loss_fn` is `NULL`.
`max_depth`	Integer maximum depth of the tree. If the current depth equals `max_depth`, the node is made a leaf.
`min_leaf_n`	Integer minimum number of observations required to attempt a split. If `X` has <= `min_leaf_n` rows, the node becomes a leaf.
`log_fn`	Function for logging debug messages. Default no-op.
`max_rejects_per_node`	Integer. A safety budget for how many times a node may *reject* non-improving splits before the algorithm gives up and finalizes the node as a leaf. Each rejection halves the probability of the last-tried feature (via `reduce_weight_fn`) and samples a new candidate feature to try again. This prevents infinite recursion / stack overflows in adversarial or flat objective landscapes. Larger values allow more exploration at a node (potentially better splits but slower), while smaller values make the builder more conservative (fewer retries, faster, more leaves). Default: `1000`.

Value

A list representing the tree/subtree.

`stratified_kfold` *Stratified K-fold index generator*

Description

Splits indices into K folds while preserving the class distribution of a binary factor. This mimics scikit-learn's `StratifiedKFold`, ensuring each fold has a representative ratio of the two classes in `S`.

Usage

```
stratified_kfold(S, K = 5)
```

Arguments

`S`	A vector or factor indicating class membership (typically 0/1 or two-class factor) for stratification.
`K`	Integer number of folds (K >= 2). If K is larger than the number of observations, it will be reduced to that number.

Details

The function deterministically allocates indices to folds by class. For each class in `S`, indices are cyclically assigned to folds to balance counts. If `K == 1`, a single fold containing all indices is returned (though typically K should be >= 2 for cross-validation).

Value

A list of length K, where each element is an integer vector of row indices assigned to that fold. The union of all folds equals `1:length(S)`, and folds are roughly equal in size.

`summary.characterizing_underrep` *Summarize a characterizing_underrep fit*

Description

Prints the `ROOT` summary (un/weighted estimates with standard deviations) and a brief overview of terminal rules from the annotated summary tree, if available.

Usage

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

Arguments

`object`	A `characterizing_underrep` object.
`...`	Unused; included for S3 compatibility.

Details

Delegates core statistics to `summary(object$root)`; previews up to five terminal rules when a summary tree exists, and reports plot availability.

Value

`object`, invisibly.

`summary.ROOT` *Summarize a ROOT fit*

Description

Summarizes a `ROOT` object by reporting the primary estimands and key model diagnostics. The first two lines report:

1. the **unweighted** estimate (ATE in RCT for single-sample or TATE for two-sample) and its **standard error (SE)**;
2. the **weighted** estimate (Weighted ATE in RCT or WTATE, using the learned subgroup weights `w_opt`) and its **SE**.

Subsequent lines describe the estimand type, number of trees, size of the Rashomon set, presence of a summary tree, covariate count, observation count, baseline loss, selected-tree losses, and the proportion kept by `w_opt`.

Usage

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

Arguments

`object`	A `ROOT` object returned by `ROOT()`.
`...`	Unused; included for S3 compatibility.

Details

This method prefers pre-computed estimates under `object$estimate` (as stored by `ROOT()`). If unavailable, it recomputes:

- Unweighted mean as $\mathrm{mean}(v)$ over the analysis set (all rows in single-sample; `S==1` in two-sample);
- Unweighted SE as $\sqrt{\mathrm{var}(v)/(n)}$ on the same set;
- Weighted mean as $\sum wv / \sum w$ where `w = w_opt` (binary), and weighted SE as $\sqrt{\sum w(v - \bar{v}_w)^2 / \sum w}$.

Value

The input `object`, invisibly. Printed output is a human-readable summary.

`train` *Train nuisance models for weighting*

Description

Fits models to estimate sampling and treatment propensities on training data. Specifically, it computes:

- `pi`: The prevalence of sample inclusion (estimated as mean of `S`).
- `pi_m`: A logistic regression model for $P(S = 1 \mid X)$ using all covariates.
- `e_m`: A logistic regression model for $P(Tr = 1 \mid X, S = 1)$, fit only on the subset where `S==1`.

Usage

```
train(training_data, outcome, treatment, sample)
```

Arguments

`training_data`	A data frame containing the training dataset.
`outcome`	Name of the outcome column (typically observed outcome, e.g. "Yobs").
`treatment`	Name of the treatment indicator column (e.g. "Tr").
`sample`	Name of the sample inclusion indicator column (e.g. "S").

Details

This function uses simple logistic regression (`glm` with logit link) to estimate the necessary nuisance parameters for weighting. It requires that both `S` and `Tr` have variation (both 0 and 1 must be present in training data); if not, the fitting is not possible and an error is raised. All covariates other than the specified outcome, treatment, and sample columns are used as predictors.

Value

A list with components:

`pi`	Numeric scalar giving the overall sample inclusion rate $P(S = 1)$ in the training data.
`pi_m`	A fitted `glm` model (binomial family) for $P(S = 1 \mid X)$.
`e_m`	A fitted `glm` model (binomial family) for $P(Tr = 1 \mid X, S = 1)$.

`train_single` *Train treatment propensity model (single-sample mode)*

Description

Fits a logistic regression for $P(T = 1 \mid X)$ on the provided training data. Used by the single-sample Double ML path where no sample-selection model is needed.

Usage

```
train_single(training_data, outcome, treatment)
```

Arguments

`training_data`	A data frame containing the outcome, treatment, and covariates. Only `treatment` and covariates are used for fitting.
`outcome`	Name of the outcome column (character). Present for a consistent signature but not used in this function.
`treatment`	Name of the binary treatment indicator column (0/1).

Value

A list with one element:

`e_m`	A `glm` (binomial) object for the treatment propensity model.

Index

characterize_tree, 2
characterizing_underrep, 3
choose_feature, 4

estimate, 5
estimate_dml, 6, *6*
estimate_dml_single, 7
estimate_single, 7

gen_S, 8
gen_T, 8
gen_XY, 9
get_data, 10

loss, 11
loss_from_objective, 11

midpoint, 12

objective_default, *11*, 12
objective_if, *11*, 13

reduce_weight, 13
ROOT, *12*, 14

split_node, *12*, 15
stratified_kfold, 17
summary.characterizing_underrep, 17
summary.ROOT, 18

train, *6*, 19
train_single, *7*, 20