Package 'ROOT'

September 3, 2025

Title What the Package Does (One Line, Title Case)

Version 0.0.0.9000

Description What the package does (one paragraph).

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Suggests knitr,
rpart.plot,
rmarkdown,
testthat (>= 3.0.0)

Config/testthat/edition 3

Imports gbm,
MASS,
mlbench,
rpart,
stats,
withr

VignetteBuilder knitr

R topics documented:

`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.

`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 ATT in 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.

Examples

```
sim<-get_data(n=2000,seed=599)
dml<-estimate_dml(sim$data,outcome="Yobs",treatment="Tr",sample="S",crossfit= 5)
```

forest_opt *Ensemble of weighted trees (forest) and Rashomon set selection*

Description

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

Usage

```
forest_opt(
  data,
  outcome,
  treatment,
  sample,
  leaf_proba = 0.25,
  seed = NULL,
  num_trees = 10,
  vote_threshold = 2/3,
  explore_proba = 0.05,
  feature_est = "Ridge",
  top_k_trees = FALSE,
  k = 10,
  cutoff = "baseline",
  verbose = FALSE
)
```

Arguments

data	A data frame containing the dataset (must include outcome, treatment, sample indicator).
outcome	Name of the outcome column.
treatment	Name of the treatment indicator column (0/1).
sample	Name of the sample indicator column (0/1).
leaf_proba	Probability mass for the "leaf" option in each tree (see tree_opt; default 0.25).
seed	Integer seed for reproducibility (default 42).
num_trees	Number of trees to grow in the forest (default 10).
vote_threshold	Majority vote threshold in (0.5, 1] for assigning final weight=1 (default 2/3, i.e., at least 67% of trees vote 1).
explore_proba	Probability of exploration at leaves in each tree (default 0.05, see split_node).
feature_est	Method for feature importance estimation, "Ridge" (default) or "GBM".
top_k_trees	Logical. If TRUE, selects the Rashomon set as the top k trees with lowest objective. If FALSE, uses a cutoff threshold.
k	Integer, number of top trees to keep if top_k_trees = TRUE (default 10).
cutoff	Either "baseline" or a numeric value. If top_k_trees = FALSE, this is the loss cutoff for selecting Rashomon set (if "baseline", uses the baseline loss of not weighting any observations).
verbose	Logical, if TRUE, prints progress messages (e.g., estimated PATE and number of trees selected). Default FALSE.

Details

Each tree in the forest is built on the same cross-fitted pseudo-outcomes (from one call to `estimate_dml`) for consistency. If `feature_est = "GBM"`, a gradient boosting model (from **gbm**) is used instead of ridge regression to estimate feature importance for splitting probabilities. The Rashomon set is defined as either the top `k` trees with lowest objective (if `top_k_trees=TRUE`), or all trees with objective below a cutoff. The baseline loss is defined as the loss with no selection (all weights = 1 for S==1 group).

Value

A list with components:

`D_rash`	A data frame similar to `D_forest` but containing only the Rashomon set of trees' weight columns and additional columns: `w_opt` (the ensemble-voted weight for each observation) and `vote_count` (how many trees voted to include that observation).
`D_forest`	A data frame with all trees' weight assignments. It contains covariates, `v`, `vsq`, `S`, `1X` (overlap factor = 1/b), and columns `w_tree_1, ..., w_tree_num_trees` for each tree's weights.
`w_forest`	A list of length `num_trees`, where each element is the full tree object returned by the internal split routine (containing the tree structure and local objective).
`rashomon_set`	An integer vector of indices of trees that were selected into the Rashomon set.
`f`	An `rpart` model fit on `X` vs `w_opt` (the final classifier summarizing the ensemble's selected subgroup).
`testing_data`	The data frame of observations used in tree construction (same as in `tree_opt` and output from `estimate_dml`, i.e., those with `S==1` and finite pseudo-outcomes).

Examples

```
sim<-get_data(n=2000,seed=599)
D <-sim$data
dml<-estimate_dml(D,outcome="Yobs",treatment="Tr",sample="S",crossfit= 5)
fo_res<-forest_opt(D,
                   outcome="Yobs",treatment="Tr",sample="S",
                   leaf_proba=0.25,seed=3,
                   num_trees=50, vote_threshold=2/3,
                   explore_proba=0.05,feature_est="Ridge",
                   top_k_trees=FALSE,cutoff="baseline")
```

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

Description

Generate sample indicator `S` ~ Bernoulli(plogis(a)) 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.

Examples

```
# Create a small example data frame D
D <- data.frame(vsq = c(0.5, 1.0, 0.2), w = c(1, 1, 1))
loss(0, indices = 2, D)  # Set w=0 for row 2 and compute loss
loss(1, indices = c(2,3), D)  # Set w=1 for rows 2 and 3 and compute loss
```

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.

---

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.

---

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

Description

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

Usage

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

Arguments

split_feature	Named numeric vector of feature selection probabilities (should include a "leaf" option).
X	A data frame of current observations (must include at least the feature fj chosen for splitting if applicable; may also include a working copy of weights w).
D	A data frame representing the global state (must include columns w for weights and vsq for squared 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 split at a leaf (randomly flipping the chosen weight).
choose_feature	Function to choose the next feature to split (default uses choose_feature).
loss_fn	Function to compute loss given a tentative weight assignment (default uses loss).
reduce_weight_fn	Function to adjust feature weights when a split is rejected (default uses reduce_weight).
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. By default, a no-op. If provided (e.g., from tree_opt(verbose=TRUE)), it will receive formatted strings to output.

Details

This function works as follows:

1. Selects a feature fj to split using choose_feature and the current probability vector split_feature. If fj == "leaf", no further splitting is done and the node becomes a leaf.
2. If the maximum depth or minimum node size criteria are met, the node becomes a leaf.

3. Otherwise, it computes a split threshold (`cj`) as the midpoint of feature `fj` in the current data `X`, and partitions `X` (and the corresponding indices in `D`) into left (`X_left`) and right (`X_right`) sets.
4. If either side is empty, the node becomes a leaf (no split possible).
5. It then evaluates the loss for assigning all left observations weight 0 vs 1, and all right observations weight 0 vs 1, to determine optimal assignments (`w_left` and `w_right` for each side).
6. If the combined loss `new_loss` is less than or equal to the `parent_loss`, the split is accepted: weights in `D` are updated for left and right, and the function recurses on each side (randomizing the order of recursion to break symmetry).
7. If the split does not improve the loss, it is rejected: the selection probability of `fj` is reduced (via `reduce_weight_fn`), and `split_node` is called again on the same data without increasing depth.

Leaf node assignment: when a node is designated as a leaf, it compares the loss of assigning all observations in that node `w=0` versus `w=1` and picks the assignment that yields lower loss (exploitation). With probability `explore_proba`, it may instead pick the opposite assignment to encourage exploration. This randomness can be controlled via the `explore_proba` parameter.

Value

A list representing the tree/subtree. Each node includes:

`node`	Feature name used for split at this node (or "leaf").
`split`	Numeric value of the splitting threshold (midpoint) if node is a split.
`left_tree`	Left subtree (list) if node is a split.
`right_tree`	Right subtree (list) if node is a split.
`w`	If node is a leaf, the weight (0 or 1) assigned to all observations in this node.
`local objective`	Numeric, the loss objective value at this node after assignment.
`depth`	Integer depth of this node.
`D`	Data frame of global state after this node's assignments (for internal use).
`leaf_reason`	(Leaf nodes only) Text reason for why this node was made a leaf (e.g., "max-depth", "min-leaf", "feature==leaf", etc.).
`feature`	(Leaf nodes only) The feature that would have been split (or NA).
`cut`	(Leaf nodes only) The cut value (or NA) that would have been used.

Note

This function is typically not called directly by the user. It assumes that `D` and `X` are kept in sync and that `D$w` is updated globally as splits are made. The `log_fn` parameter allows optional logging of the splitting process for debugging or insight when `verbose=TRUE` is passed from higher-level functions.

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.

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 $P(Tr = 1 \mid X, S = 1)$.

---

tree_opt *Fit a single weighted tree for optimized subgroup selection*

Description

Runs a Double ML procedure to compute pseudo-outcomes, then grows a single decision tree that optimizes an objective function (approximate treatment effect standard error) by reweighting observations. Finally, fits a shallow classifier to summarize the resulting weights.

Usage

```
tree_opt(
  data,
  outcome,
  treatment,
  sample,
  leaf_proba = 0.25,
  seed = NULL,
  verbose = FALSE
)
```

Arguments

data	A data frame containing the full dataset (must include outcome, treatment, and sample columns).
outcome	Name of the outcome column.
treatment	Name of the treatment indicator column (0/1).

sample	Name of the sample indicator column (0/1).
leaf_proba	Numeric in [0,1], the probability weight assigned to choosing a "leaf" (no split) at each node during tree building (default 0.25).
seed	Integer seed for reproducibility (default 42). A single set.seed is called at the start for all random components.
verbose	Logical, if TRUE enables verbose logging of the tree-building process (prints details of splits and decisions). Defaults to FALSE.

Details

Procedure: First, cross-fitted pseudo-outcomes are computed using estimate_dml (with a default of 5 folds). Then a ridge regression (or GBM, if specified) is used to estimate feature importances for vsq (the variance of the pseudo-outcome), which informs the initial feature selection probabilities. A special "leaf" option is included with probability leaf_proba to allow the tree to terminate splits. The function then calls an internal recursive splitting routine to assign binary weights w to observations, optimizing the treatment effect estimate's precision. The final classifier (f) is a decision tree of fixed depth (by default 3) trained on X to predict the optimized w labels for interpretability.

Randomness and Reproducibility: The function uses random number generation in the tree-building process (for tie-breaking and exploration). Setting the same seed will produce the same result. The verbose flag can be used to trace the splitting process for debugging or insight.

Value

A list with components:

D	A data frame (same rows as data2) including covariates and additional columns: v (pseudo-outcome), vsq (squared pseudo-outcome), w (optimized weight, 0/1), and S (sample indicator).
f	An rpart model fit on X vs w (the classifier summarizing the learned weights).
testing_data	A data frame of the subset of data used in tree optimization (specifically, those with S==1 and finite weights). This is essentially the data2 from the DML step.

Examples

```
sim<-get_data(n=2000,seed=599)
D <-sim$data
dml<-estimate_dml(D,outcome="Yobs",treatment="Tr",sample="S",crossfit= 5)
tr_res <- forest_opt(D,
                 outcome="Yobs",treatment="Tr",sample="S",
                 leaf_proba=0.25,seed=599)
```

Index