

Package: ROOT (via r-universe)

October 21, 2026

Title Rashomon Set of Optimal Trees

Version 0.2.1

Description Implements a general framework for globally optimizing user-specified objective functionals over interpretable binary weight functions represented as sparse decision trees, called ROOT (Rashomon Set of Optimal Trees). It searches over candidate trees to construct a Rashomon set of near-optimal solutions and derives a summary tree highlighting stable patterns in the optimized weights. ROOT includes a built-in generalizability mode for identifying subgroups in trial settings for transportability analyses (Parikh et al. (2025) <[doi:10.1080/01621459.2025.2495319](https://doi.org/10.1080/01621459.2025.2495319)>).

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

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

Config/testthat/edition 3

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

URL <https://github.com/peterliu599/ROOT>

BugReports <https://github.com/peterliu599/ROOT/issues>

VignetteBuilder knitr

Depends R (>= 3.5)

LazyData true

NeedsCompilation no

Config/roxygen2/version 8.1.1

Repository <https://peterliu599.r-universe.dev>

Date/Publication 2026-10-21 19:34:04 UTC

RemoteUrl <https://github.com/peterliu599/root>

RemoteRef HEAD

RemoteSha 46b98c4c7e650320a1b533468a96a59fa47835ba

Contents

characterizing_underrep	2
diabetes_data	6
plot.characterizing_underrep	7
plot.ROOT	8
print.characterizing_underrep	9
print.ROOT	10
ROOT	11
summary.characterizing_underrep	18
summary.ROOT	19

Index	21
--------------	-----------

characterizing_underrep
Characterize Underrepresented Subgroups

Description

A high-level wrapper around `ROOT()` for identifying and characterizing subgroups that are *insufficiently represented* in a randomized controlled trial (RCT) relative to a target population. The function returns an interpretable decision tree describing which subgroups should be included ($w(X) = 1$) or excluded ($w(X) = 0$) from the analysis, along with the corresponding target average treatment effect estimates.

Usage

```
characterizing_underrep(
  data,
  global_objective_fn = NULL,
  generalizability_path = FALSE,
  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",
  max_depth = 8L,
  min_leaf_n = 2L,
  max_rejects_per_node = 10L,
  verbose = FALSE,
  positivity_trim = NULL,
  split_strategy = "midpoint",
```

```

    quantile_range = c(0.1, 0.9),
    summary_max_depth = 3L,
    min_kept = 0.1
  )

```

Arguments

<code>data</code>	A <code>data.frame</code> containing covariates and, in generalizability mode, also columns <code>Y</code> , <code>Tr</code> , and <code>S</code> .
<code>global_objective_fn</code>	A function with signature <code>function(D) -> numeric</code> scoring the entire state and minimized by <code>ROOT</code> . If <code>NULL</code> (the default), a variance-based objective is used that targets the precision of the WTATE estimator.
<code>generalizability_path</code>	Logical. If <code>TRUE</code> , runs the full generalizability analysis (design + analysis stages) and expects columns <code>Y</code> , <code>Tr</code> , and <code>S</code> in <code>data</code> . If <code>FALSE</code> , runs <code>ROOT</code> in general optimization mode. Default <code>FALSE</code> .
<code>leaf_proba</code>	A <code>numeric(1)</code> tuning parameter that increases the chance a node stops splitting by selecting a synthetic "leaf" feature. Internally, the probability of choosing "leaf" is $\text{leaf_proba} / (1 + \text{leaf_proba})$ (assuming the covariate probabilities sum to 1). Higher values produce shallower, more interpretable trees. Default <code>0.25</code> .
<code>seed</code>	Random seed for reproducibility.
<code>num_trees</code>	Number of trees to grow in the <code>ROOT</code> forest. More trees explore the tree space more thoroughly but increase computation time.
<code>vote_threshold</code>	Controls how Rashomon-set tree votes are aggregated into <code>w_opt</code> . Accepts a numeric threshold in $(0, 1]$, one of "majority" / "mean" / "median", or a custom function(<code>votes</code>) -> integer vector. See <code>ROOT</code> for full details. Default <code>2/3</code> (majority vote).
<code>explore_proba</code>	Exploration probability in tree growth. Controls the explore-exploit trade-off: with probability <code>explore_proba</code> , a leaf is assigned a random weight; otherwise, it receives the greedy optimal weight. Default <code>0.05</code> .
<code>feature_est</code>	Either "Ridge", "GBM", or a custom feature importance function. Used to bias which covariates are selected for splitting. Default "Ridge".
<code>feature_est_args</code>	List of extra arguments passed to <code>feature_est</code> when it is a function.
<code>top_k_trees</code>	Logical; if <code>TRUE</code> , selects the top <code>k</code> trees by objective value for the Rashomon set. If <code>FALSE</code> , uses <code>cutoff</code> instead. Default <code>FALSE</code> .
<code>k</code>	Number of trees to retain when <code>top_k_trees = TRUE</code> . Default <code>10</code> .
<code>cutoff</code>	Numeric or "baseline". When <code>top_k_trees = FALSE</code> , trees with objective values below this cutoff are included in the Rashomon set. "baseline" uses the objective value at $w \equiv 1$ (no subgroups excluded). Default "baseline".
<code>max_depth</code>	Maximum depth of each tree grown during the forest construction stage. A node at <code>depth == max_depth</code> is forced to be a leaf. Shallower trees are more interpretable but less flexible. Default <code>8</code> .

<code>min_leaf_n</code>	Minimum number of observations required in a node for splitting to be attempted. If a node contains fewer than <code>min_leaf_n</code> observations it becomes a leaf. Default 2.
<code>max_rejects_per_node</code>	Maximum number of consecutive rejected splits (splits that do not improve the objective) allowed at a single node before the node is forced to become a leaf. This prevents infinite recursion in pathological cases. Default 10.
<code>verbose</code>	Logical; if TRUE, prints the unweighted and weighted estimands with standard errors. Default FALSE.
<code>positivity_trim</code>	Controls pre-trimming of target observations with near-zero trial selection probability. Only used when <code>generalizability_path = TRUE</code> . Accepts NULL (no trimming, default), a numeric in (0, 0.5), or "crump" for the data-driven threshold of Crump et al. (2009). See ROOT for details.
<code>split_strategy</code>	Character; either "midpoint" (default) or "random_quantile". Controls how split points are chosen at each tree node. See ROOT for details.
<code>quantile_range</code>	Numeric vector of length 2 giving the lower and upper bounds of the quantile range when <code>split_strategy = "random_quantile"</code> . Default <code>c(0.1, 0.9)</code> . Ignored when <code>split_strategy = "midpoint"</code> .
<code>summary_max_depth</code>	Positive integer. Maximum depth of the characteristic (summary) tree. Default 3. See ROOT for details.
<code>min_kept</code>	Numeric in [0, 1). Generalizability mode only: the smallest share of trial units a candidate rule may keep under the default objective. Default 0.1. See ROOT for details.

Value

A `characterizing_underrep` S3 object (a list) with:

<code>root</code>	The ROOT object returned by <code>ROOT()</code> . Contains the Rashomon set, weight assignments (<code>root\$D_rash\$w_opt</code>), summary tree (<code>root\$f</code>), and (in generalizability mode) treatment effect estimates (<code>root\$estimate</code>).
<code>combined</code>	The input data.
<code>leaf_summary</code>	A <code>data.frame</code> with one row per terminal node of the summary (characteristic) tree, giving the decision rule, predicted weight, sample size, and a label indicating whether the subgroup is "Represented (keep, $w = 1$)" or "Under-represented (drop, $w = 0$)". NULL if no summary tree was produced.

What does "underrepresented" mean?

In the context of generalizing treatment effects from a trial to a target population, a subgroup is considered **underrepresented** (or *insufficiently represented*) when it occupies a region of the covariate space that both (a) has limited overlap between the trial and the target population, and (b) exhibits heterogeneous treatment effects.

Formally, the contribution of a unit with covariates $X = x$ to the variance of the target average treatment effect (TATE) estimator depends on both the selection ratio $\ell(x) = P(S = 1 \mid X =$

$x)/P(S = 0 \mid X = x)$ and the conditional average treatment effect. Subgroups where $\ell(x)$ is small, and conditional average treatment effect deviates from the overall TATE, contribute disproportionately to estimator variance. These are the subgroups that `characterizing_underrep()` identifies and characterizes. The sample target average treatment effect (STATE), the average treatment effect over the target sample, is a finite sample equivalent version of the TATE.

The generalizability workflow

When `generalizability_path = TRUE`, this function implements the two-stage approach of Parikh et al. (2025):

1. **Design stage:** ROOT learns binary weights $w(X)$ that minimize the variance of the weighted target average treatment effect (WTATE) estimator, subject to interpretability constraints (tree structure). The resulting decision tree characterizes which subgroups are well-represented ($w = 1$) and which are underrepresented ($w = 0$).
2. **Analysis stage:** The WTATE is estimated on the refined target population that excludes the underrepresented subgroups. This estimand trades some generality for greater precision and credibility.

The key estimands are:

- **STATE** (Sample Target Average Treatment Effect): the treatment effect for the full target sample, estimated by transporting the trial results, which may be imprecise if certain subgroups are underrepresented. It is a finite sample equivalent version of the TATE.
- **WTATE** (Weighted Target Average Treatment Effect): the treatment effect restricted to the sufficiently represented subpopulation, estimated with lower variance.

General optimization mode

When `generalizability_path = FALSE`, this function behaves as a convenience wrapper around `ROOT()` for arbitrary binary weight optimization. The user can supply a custom objective function via `global_objective_fn`; ROOT will learn an interpretable tree-based weight function minimizing that objective. See `vignette("optimization_path_example")` for an example.

Data requirements

When `generalizability_path = TRUE`, data must contain the following standardized columns:

- Y: numeric outcome variable.
- Tr: binary treatment indicator (0 = control, 1 = treated).
- S: binary sample indicator (1 = trial/RCT, 0 = target population).

All remaining columns are treated as pretreatment covariates X available for splitting.

References

Parikh H, Ross RK, Stuart E, Rudolph KE (2025). "Who Are We Missing?: A Principled Approach to Characterizing the Underrepresented Population." *Journal of the American Statistical Association*. doi:10.1080/01621459.2025.2495319

See Also

[ROOT](#) for the underlying optimization engine; `vignette("generalizability_path_example")` for a detailed worked example of the generalizability workflow; `vignette("optimization_path_example")` for general optimization mode.

Examples

```
# --- Generalizability analysis ---
# diabetes_data has columns Y, Tr, S, and covariates
data(diabetes_data, package = "ROOT")

char_fit <- characterizing_underrep(
  data          = diabetes_data,
  generalizability_path = TRUE,
  num_trees     = 20,
  top_k_trees   = TRUE,
  k             = 10,
  seed         = 123
)

# View the characterization tree
plot(char_fit)

# Inspect which subgroups are underrepresented
char_fit$leaf_summary

# Treatment effect estimates (STATE and WTATE)
char_fit$root$estimate
```

diabetes_data

Simulated diabetes trial and target data

Description

A simulated dataset for illustrating ROOT in generalizability mode. It mimics a randomized trial of a glucose-lowering drug (about 2,000 participants, $S = 1$) and a target population of adults with type 2 diabetes (about 8,000 people, $S = 0$). Treatment and outcome are observed only in the trial.

Usage

```
data(diabetes_data)
```

Format

A data.frame with 10,000 rows and 13 columns:

Age Age in years (30 to 85).

Sex_Male Indicator in 0/1: male.

Race_Black Indicator in 0/1: race is Black.

Rural Indicator in 0/1: lives in a rural area.

BMI Body mass index (kg/m²).

HbA1c Baseline HbA1c (percent).

Insulin Indicator in 0/1: currently uses insulin.

CKD Indicator in 0/1: chronic kidney disease.

HeartFailure Indicator in 0/1: heart failure.

DietYes Indicator in 0/1: enrolled in a diet program.

S Sample indicator: 1 = trial, 0 = target.

Tr Treatment assignment in 0/1 (trial only; NA in the target).

Y Reduction in HbA1c (percentage points) at 6 months (trial only; NA in the target).

Details

The design contains all four combinations of trial under-sampling and treatment-effect modification: chronic kidney disease (CKD), heart failure (HeartFailure), insulin use (Insulin), and age 70 or older are under-sampled *and* modify the treatment effect; Race_Black and Rural are under-sampled but do *not* modify the effect; baseline HbA1c mildly modifies the effect but is well sampled; Sex_Male, BMI, and DietYes neither modify the effect nor are under-sampled.

Data-generating process

Trial participation follows

$$\text{logit } P(S = 1 \mid X) = -0.9 - 1.25 \text{ CKD} - 0.8 \text{ HF} - 0.4 \text{ Insulin} - 0.018(\text{Age} - 58) - 0.5 \text{ Black} - 0.35 \text{ Rural},$$

where HF denotes HeartFailure. Within the trial, Tr is randomized with probability 0.5. The conditional average treatment effect is

$$\tau(X) = 1 + 0.1(\text{HbA1c} - 8) - 2.0 \text{ CKD} - 2.4 \text{ HF} + 1.5 \text{ Insulin} + 2.0 \mathbf{1}\{\text{Age} \geq 70\},$$

and the outcome is $Y = \mu_0(X) + \text{Tr} \tau(X) + \varepsilon$, with $\mu_0(X) = 0.2(\text{HbA1c} - 8) + 0.1 \text{ Insulin} - 0.2 \text{ Black} + 0.1 \text{ Rural} - 0.01(\text{BMI} - 31) + 0.2 \text{ DietYes} + 0.1 \text{ HF}$ and $\varepsilon \sim N(0, 0.5^2)$. The effects are illustrative. The full simulation code is in data-raw/diabetes_data.R in the package source.

plot.characterizing_underrep

Plot Underrepresented Population Characterization

Description

Visualizes the decision tree derived from the ROOT analysis. Highlights which subgroups are represented where $w = 1$ versus underrepresented where $w = 0$ in generalizability mode, or simply $w(x)$ in $\{0, 1\}$ in general optimization mode.

Usage

```
## S3 method for class 'characterizing_underrep'
plot(
  x,
  main = "Final Characterized Tree from Rashomon Set",
  cex.main = 1.2,
  ...
)
```

Arguments

x	A characterizing_underrep S3 object with x\$root\$f present as an rpart object for the summary or characterization tree.
main	Character string for the plot title. Default is "Final Characterized Tree from Rashomon Set".
cex.main	Numeric scaling factor for the title text size. Default is 1.2.
...	Additional arguments passed to rpart.plot::prp().

Value

NULL. The plot is drawn to the active graphics device.

Examples

```
char.output = characterizing_underrep(diabetes_data, generalizability_path = TRUE,
                                     num_trees = 20, top_k_trees = TRUE, k = 10, seed = 123)
plot(char.output)
```

plot.ROOT	<i>Plot the ROOT summary tree</i>
-----------	-----------------------------------

Description

Visualizes the decision tree that characterizes the weighted subgroup (the weight function $w(d)$ in $\{0, 1\}$) identified by ROOT(), using rpart.plot::prp().

Usage

```
## S3 method for class 'ROOT'
plot(x, ...)
```

Arguments

x	A "ROOT" S3 object returned by ROOT() with x\$f an rpart object representing the summary / characterization tree.
...	Additional arguments passed to rpart.plot::prp().

Value

No return value; the plot is drawn to the active graphics device.

Examples

```
ROOT.output = ROOT(diabetes_data, generalizability_path = TRUE,
                   num_trees = 20, top_k_trees = TRUE, k = 10, seed = 123)
plot(ROOT.output)
```

```
print.characterizing_underrep
```

Print a characterizing_underrep fit

Description

Print the ROOT summary which includes unweighted and (when in generalizability mode) weighted estimates with standard errors, as reported by `summary.ROOT()`.

Usage

```
## S3 method for class 'characterizing_underrep'
print(x, ...)
```

Arguments

<code>x</code>	A <code>characterizing_underrep</code> S3 object. Expected components include <code>root</code> which is a <code>ROOT</code> object (summarized by <code>print.ROOT()</code>) and may contain <code>f</code> which is an <code>rpart</code> object for the summary tree, and <code>leaf_summary</code> which is a <code>data.frame</code> with one row per terminal node and may include a <code>rule</code> column of type <code>character</code> .
<code>...</code>	Currently unused. Included for S3 compatibility.

Details

Delegates core statistics and estimands to `print(x$root)`.

Value

object returned invisibly. Printed output is a readable brief summary.

Abbreviations

- ATE** Average treatment effect.
- RCT** Randomized controlled trial.
- SE** Standard error.
- TATE** Target average treatment effect.
- WTATE** Weighted target average treatment effect.
- STATE** Sample target average treatment effect.

Examples

```
char.output = characterizing_underrep(diabetes_data, generalizability_path = TRUE,
                                     num_trees = 20, top_k_trees = TRUE, k = 10, seed = 123)
print(char.output)
```

```
print.ROOT
```

```
Print a ROOT fit
```

Description

Provides a human-readable brief summary of a ROOT object, including:

1. the summary characterization tree f ,
2. in generalizability mode (`generalizability_path = TRUE`), the unweighted and weighted estimands with their standard errors and an explanatory note for the weighted standard error (SE).

Usage

```
## S3 method for class 'ROOT'
print(x, ...)
```

Arguments

- | | |
|------------------|--|
| <code>x</code> | A "ROOT" S3 object returned by <code>ROOT()</code> . |
| <code>...</code> | Currently unused and included for S3 compatibility. |

Value

object returned invisibly. Printed output is for inspection.

Abbreviations

ATE Average treatment effect.

RCT Randomized controlled trial.

SE Standard error.

TATE Target average treatment effect.

WTATE Weighted target average treatment effect.

STATE Sample target average treatment effect.

When `generalizability_path = TRUE`, the unweighted estimand corresponds to a STATE-type quantity and the weighted estimand to a WTATE-type quantity for the transported target population. When `generalizability_path = FALSE`, ROOT is used for general functional optimization and no causal labels are imposed.

Examples

```
ROOT.output = ROOT(diabetes_data, generalizability_path = TRUE,
                    num_trees = 20, top_k_trees = TRUE, k = 10, seed = 123)
print(ROOT.output)
```

ROOT

Rashomon Set of Optimal Trees (ROOT) for Functional Optimization

Description

ROOT (Rashomon Set of Optimal Trees) is a general-purpose functional optimization algorithm that learns interpretable, tree-structured binary weight functions $w(X) \in \{0, 1\}$. Given a dataset D_n and a global objective function $L(w, D_n)$, ROOT searches over the space of decision trees to find weight assignments that minimize the objective function.

Usage

```
ROOT(
  data,
  global_objective_fn = NULL,
  generalizability_path = FALSE,
  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",
```

```

max_depth = 8L,
min_leaf_n = 2L,
max_rejects_per_node = 10L,
verbose = FALSE,
positivity_trim = NULL,
split_strategy = "midpoint",
quantile_range = c(0.1, 0.9),
summary_max_depth = 3L,
min_kept = 0.1
)

```

Arguments

- | | |
|-----------------------|---|
| data | <p>A data.frame containing the dataset.</p> <p>In general optimization mode (<code>generalizability_path = FALSE</code>), data can contain any covariates and auxiliary columns. The user either supplies a <code>global_objective_fn</code> that takes a data frame with a column <code>w</code> and returns a scalar loss, or relies on the default objective, which reads an optional <code>vsq</code> or <code>v</code> column (see the General optimization mode section). Columns named <code>vsq</code> and <code>v</code> are auxiliary and are never used for splitting.</p> <p>In generalizability mode (<code>generalizability_path = TRUE</code>), data must contain columns <code>"Y"</code> (outcome), <code>"Tr"</code> (treatment indicator, 0/1), and <code>"S"</code> (sample indicator, 1 = trial, 0 = target). ROOT internally constructs transportability scores and, if no custom objective is given, uses a default variance-based loss.</p> |
| global_objective_fn | <p>A function with signature <code>function(D) -> numeric</code> scoring the entire state and minimized by ROOT. <code>D</code> is the working data frame containing at least a column <code>w</code> with the current weight assignments. If <code>NULL</code> (default), a variance-based objective is used.</p> |
| generalizability_path | <p>Logical(1). If <code>TRUE</code>, use the built-in transportability objective for trial-to-target generalizability. If <code>FALSE</code>, treat data as arbitrary and rely on <code>global_objective_fn</code>. Default <code>FALSE</code>.</p> |
| leaf_proba | <p>A numeric tuning parameter that increases the chance a node stops splitting by selecting a synthetic "leaf" feature. Internally, the probability of choosing "leaf" is $\text{leaf_proba} / (1 + \text{leaf_proba})$ (assuming the covariate probabilities sum to 1). Higher values produce shallower, sparser trees. Default 0.25.</p> |
| seed | <p>An optional numeric seed for reproducibility.</p> |
| num_trees | <p>An integer number of trees to grow. More trees explore the solution space more thoroughly. Default 10.</p> |
| vote_threshold | <p>Controls how per-observation votes from the Rashomon set trees are aggregated into the final binary weight <code>w_opt</code>. Accepts one of:</p> <ul style="list-style-type: none"> • A numeric in $(0, 1)$: <code>w_opt = 1</code> when the mean fraction of Rashomon-set trees voting 1 strictly exceeds this value. An error is raised if any observation's mean vote equals the threshold exactly (tie is undefined). Default 2/3. • "majority": equivalent to the numeric threshold 2/3. |

	<ul style="list-style-type: none"> • "mean": $w_{\text{opt}} = 1$ when the mean vote strictly exceeds 0.5. Errors on exact 0.5 ties. • "median": $w_{\text{opt}} = 1$ when the per-row median vote strictly exceeds 0.5. Errors on exact 0.5 ties. • A function with signature <code>function(votes) -> integer vector</code>, where <code>votes</code> is a numeric matrix with one row per observation and one column per Rashomon-set tree (each cell 0 or 1). Must return an integer vector of 0s and 1s of length <code>nrow(votes)</code>.
<code>explore_proba</code>	A numeric giving the exploration probability at leaves. With probability <code>explore_proba</code> , a leaf receives a random weight; otherwise the greedy-optimal weight is used. Default 0.05.
<code>feature_est</code>	Either "Ridge", "GBM", or a custom function(<code>X, y, ...</code>) -> named numeric vector returning nonnegative importance values with names matching columns of <code>X</code> . Used to bias which covariates are chosen for splitting. If the method fails, ROOT falls back to uniform feature sampling with a warning.
<code>feature_est_args</code>	A list of additional arguments passed to a user-supplied <code>feature_est</code> function.
<code>top_k_trees</code>	Logical(1). If TRUE, select the top <code>k</code> trees by objective value for the Rashomon set. If FALSE, use the cutoff threshold. Default FALSE.
<code>k</code>	An integer giving the number of top trees when <code>top_k_trees = TRUE</code> . Default 10.
<code>cutoff</code>	A numeric or "baseline". Used as the Rashomon cutoff when <code>top_k_trees = FALSE</code> . "baseline" uses the objective at $w \equiv 1$ (all weights equal to 1, i.e., no exclusions).
<code>max_depth</code>	Maximum depth of each tree grown during the forest construction stage. A node at <code>depth == max_depth</code> is forced to be a leaf. Shallower trees are more interpretable but less flexible. Default 8.
<code>min_leaf_n</code>	Minimum number of observations required in a node for splitting to be attempted. If a node contains fewer than <code>min_leaf_n</code> observations it becomes a leaf. Default 2.
<code>max_rejects_per_node</code>	Maximum number of consecutive rejected splits (splits that do not improve the objective) allowed at a single node before the node is forced to become a leaf. This prevents infinite recursion in pathological cases. Default 10.
<code>verbose</code>	Logical(1). If TRUE, prints the unweighted and (when available) weighted treatment effect estimates and standard errors in generalizability mode.
<code>positivity_trim</code>	Controls pre-trimming of target observations with near-zero trial selection probability $P(S = 1 X)$, addressing structural positivity violations before ROOT optimization. Only used when <code>generalizability_path = TRUE</code> ; ignored otherwise. <ul style="list-style-type: none"> • NULL (default): no positivity trimming. • A numeric in (0,0.5): target observations with $\hat{P}(S = 1 X)$ below this value are removed.

- "crump": use the data-driven threshold from Crump, Hotz, Imbens & Mitnik (2009, *Biometrika*), which solves for the optimal symmetric trimming cutoff α that minimizes the asymptotic variance of the treatment-effect estimator.

Note that trimming changes the estimand: the WTATE is now estimated over the subpopulation $\{x : \hat{P}(S = 1 | X = x) \geq \alpha\}$.

split_strategy	Character; either "midpoint" (default) or "random_quantile". Controls how the split point for a chosen feature is determined at each tree node. "midpoint" uses the classic $(\min + \max) / 2$ rule; "random_quantile" draws a uniform random quantile within quantile_range, introducing controlled stochasticity that improves covariate-space coverage across the forest.
quantile_range	Numeric vector of length 2 giving the lower and upper bounds of the quantile range when split_strategy = "random_quantile". Default $c(0.1, 0.9)$. Ignored when split_strategy = "midpoint".
summary_max_depth	Positive integer. Maximum depth of the single characteristic (summary) tree fitted to the final rule w_opt with rpart. Default 3. A rule that excludes m separate subgroups may need depth m to be described exactly; larger values describe the ensemble rule more faithfully at the cost of a less compact tree. This affects only the summary tree, not w_opt or the reported estimates.
min_kept	Numeric in $[0, 1)$. Generalizability mode only: the smallest share of analysis (trial) units that a candidate rule may keep when the default objective is used. Rules keeping fewer than $\max(2, \text{ceiling}(\text{min_kept} * n))$ units receive an infinite objective value. Default 0.1. Without this bound, the finite-sample standard error can be made arbitrarily small by keeping a handful of units whose scores happen to be similar, a degenerate solution that a thorough search will find. Set min_kept = 0 to remove the bound. Ignored when a global_objective_fn is supplied or when generalizability_path = FALSE.

Value

An object of class "ROOT" (a list) with elements:

D_rash	Data frame containing the Rashomon-set tree votes and the final aggregated weight w_opt for each observation.
D_forest	Data frame with all forest-level working columns.
w_forest	List of per-tree results from the tree-building routine.
rashomon_set	Integer vector of indices identifying which trees were selected into the Rashomon set.
global_objective_fn	The objective function used.
f	The characteristic (summary) tree fitted to w_opt, as an rpart object. NULL if all observations received the same weight or no covariates were available.
testing_data	Data frame of observations used for optimization (trial units when generalizability_path = TRUE).

<code>estimate</code>	(Only when <code>generalizability_path = TRUE</code>) A list with the unweighted (STATE) and weighted (WTATE) point estimates and their standard errors, the number of analysis observations (<code>n_analysis</code>), the size of the kept set (<code>n_A</code>), the selection-odds re-normalization factor <code>kappa_w</code> , and <code>se_weighted_note</code> .
<code>generalizability_path</code>	Logical flag echoing the input.
<code>positivity_trim_info</code>	(Only when <code>positivity_trim</code> is not NULL) A list with the threshold used (<code>alpha</code>), the number of target observations removed (<code>n_trimmed</code>), and the method (<code>method</code>).

The optimization problem

ROOT solves the functional optimization problem:

$$w^* \in \arg \min_w L(D_n, w)$$

where $w : \mathbb{R}^p \rightarrow \{0, 1\}$ maps a p -dimensional covariate vector to a binary include/exclude decision. The key challenge is that, unlike standard tree algorithms, the global loss $L(D_n, w)$ is not decomposable as a sum of losses over independent subsets of the data. This means conventional greedy, divide-and-conquer tree-building strategies do not apply. ROOT addresses this through a randomization-based tree construction with an explore-exploit strategy.

How ROOT works

The algorithm proceeds in several stages:

1. **Feature importance estimation:** Split probabilities are estimated using Ridge regression, Gradient Boosting Machine (GBM), or a user-supplied function, biasing the search toward covariates likely to be informative.
2. **Stochastic tree construction:** `num_trees` trees are grown. At each internal node, a feature is drawn according to the estimated split probabilities (or a "leaf" token is drawn, terminating the branch). Splits are made at the midpoint of the selected feature's empirical distribution. An explore-exploit strategy assigns leaf weights: with probability `explore_proba` a random weight is chosen; otherwise the greedy optimal weight (reducing the global objective) is used.
3. **Rashomon set selection:** Trees are ranked by their global objective values. The top- k trees (or all trees below a cutoff) form the Rashomon set: a collection of near-optimal but potentially different models, each providing a characterization of the optimal weight function.
4. **Aggregation:** Per-observation votes from the Rashomon set are combined (by default, majority vote) to produce the final weight vector `w_opt`.
5. **Characteristic tree:** A single summary decision tree is fitted to the aggregated `w_opt` assignments, providing a concise, interpretable description of the weight function.

Generalizability mode

When `generalizability_path = TRUE`, ROOT implements the methodology of Parikh et al. (2025) for characterizing underrepresented subgroups in trial-to-target generalizability analyses. In this mode:

- data must contain columns Y (outcome), Tr (treatment, 0/1), and S (sample indicator, 1 = trial, 0 = target).
- ROOT internally computes transportability scores based on inverse-probability weighting (IPW), estimates the selection model $P(S = 1 \mid X)$, and constructs inverse propensity weighting (IPW) transported scores.
- The default objective minimizes the variance of the weighted target average treatment effect (WTATE) estimator. This objective accounts for both the selection odds (trial participation probability) and treatment effect heterogeneity, so that subgroups are flagged as underrepresented primarily when they both lack trial representation and exhibit effect modification.
- The output includes the unweighted sample target average treatment effect (STATE) and the WTATE with standard errors.

See [characterizing_underrep](#) for a higher-level wrapper that additionally produces a leaf-level summary table, and `vignette("generalizability_path_example")` for a worked example.

General optimization mode

When `generalizability_path = FALSE`, ROOT operates as a general functional optimizer. The user supplies any `data.frame` and (optionally) a custom `global_objective_fn`. If no objective is supplied, ROOT uses the default variance-based loss

$$L(w) = \sqrt{\sum_i w_i \text{vsq}_i / (\sum_i w_i)^2},$$

where `vsq` is taken from the data as follows:

- if data has a `vsq` column (per-unit variance contribution), it is used as supplied;
- otherwise, if data has a `v` column (per-unit score), $\text{vsq}_i = (v_i - \bar{v}_w)^2$ is recomputed for every candidate rule, where $\bar{v}_w$ is the mean of `v` over the currently included units;
- otherwise, $\text{vsq}_i = 1$ is used as a fallback. The loss then equals $1/\sqrt{n_{\text{kept}}}$, which is minimized by keeping every observation, so a message is issued and users should supply `vsq`, `v`, or a `global_objective_fn`.

See `vignette("optimization_path_example")` for an example with a user-supplied, covariance-based objective.

References

- Parikh H, Ross RK, Stuart E, Rudolph KE (2025). "Who Are We Missing?: A Principled Approach to Characterizing the Underrepresented Population." *Journal of the American Statistical Association*. doi:10.1080/01621459.2025.2495319
- Crump RK, Hotz VJ, Imbens GW, Mitnik OA (2009). "Dealing with Limited Overlap in Estimation of Average Treatment Effects." *Biometrika*, **96**(1), 187–199.

See Also

[characterizing_underrep](#) for a higher-level wrapper with leaf-summary output; `vignette("generalizability_path_example")` for the generalizability workflow; `vignette("optimization_path_example")` for general optimization.

Examples

```

# --- Generalizability mode ---
data(diabetes_data, package = "ROOT")
root_fit <- ROOT(
  data          = diabetes_data,
  generalizability_path = TRUE,
  num_trees     = 20,
  top_k_trees   = TRUE,
  k             = 10,
  seed         = 123
)
# --- General optimization mode (custom, covariance-based objective) ---
set.seed(123)
n_assets <- 100

# Asset features
volatility <- runif(n_assets, 0.05, 0.40) # annualised volatility
beta      <- runif(n_assets, 0.5, 1.8)    # market beta
sector    <- sample(c("Tech", "Finance", "Energy", "Health"),
  n_assets, replace = TRUE)

# Simulate returns:  $r_i = \beta_i * r_{\text{market}} + \epsilon_i$ 
market    <- rnorm(1000, 0.0005, 0.01)
returns_mat <- sapply(seq_len(n_assets), function(i)
  beta[i] * market + rnorm(1000, 0, volatility[i] / sqrt(252))
)
Sigma_hat <- stats::cov(returns_mat) # 100 x 100 return covariance

# Variance of the equal-weight portfolio of the included assets:
#  $L(w) = w' \Sigma w / (1'w)^2$ . It depends on all pairwise covariances,
# so it cannot be split into independent per-asset contributions.
portfolio_variance <- function(D) {
  w <- D$w
  if (sum(w) == 0) return(Inf)
  sum(Sigma_hat[w == 1, w == 1]) / sum(w)^2 # =  $w' \Sigma w / (1'w)^2$ 
}

my_data <- data.frame(
  vsq    = diag(Sigma_hat), # marginal variances; guide split sampling only
  vol    = volatility,
  beta   = beta,
  sector = as.integer(factor(sector))
)

opt_fit <- ROOT(
  data          = my_data,
  global_objective_fn = portfolio_variance,
  num_trees     = 20,
  top_k_trees   = TRUE,
  k             = 10,
  seed         = 42
)

```

summary.characterizing_underrep

Summarize a characterizing_underrep fit

Description

Summarizes the ROOT summary which includes unweighted and (when in generalizability mode) weighted estimates with standard errors, as reported by `summary.ROOT()`. Provides a brief overview of terminal rules from the annotated summary tree when available.

Usage

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

Arguments

<code>object</code>	A <code>characterizing_underrep</code> S3 object. Expected components include <code>root</code> which is a ROOT object (summarized by <code>summary.ROOT()</code>) and may contain <code>f</code> which is an <code>rpart</code> object for the summary tree, and <code>leaf_summary</code> which is a <code>data.frame</code> with one row per terminal node and may include a <code>rule</code> column of type <code>character</code> .
<code>...</code>	Currently unused. Included for S3 compatibility.

Details

Delegates core statistics and estimands to `summary(object$root)`. Previews up to ten terminal rules when a summary tree exists.

Value

`object` returned invisibly. Printed output is a readable summary.

Abbreviations

ATE Average treatment effect.
RCT Randomized controlled trial.
SE Standard error.
TATE Target average treatment effect.
WTATE Weighted target average treatment effect.
STATE Sample target average treatment effect.

Examples

```
char.output = characterizing_underrep(diabetes_data, generalizability_path = TRUE,
                                     num_trees = 20, top_k_trees = TRUE, k = 10, seed = 123)
summary(char.output)
```

summary.ROOT

*Summarize a ROOT fit***Description**

Provides a readable summary of a ROOT object, including:

1. the summary characterization tree f ,
2. whether the user supplied a custom `global_objective_fn` (Yes/No), and
3. in generalizability mode (`generalizability_path = TRUE`), the unweighted and weighted estimands with their standard errors.

Usage

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

Arguments

`object` A "ROOT" S3 object returned by `ROOT()`.

`...` Currently unused and included for S3 compatibility.

Value

object returned invisibly. Printed output is for inspection.

Abbreviations

ATE Average treatment effect.

RCT Randomized controlled trial.

SE Standard error.

TATE Target average treatment effect.

WTATE Weighted target average treatment effect.

STATE Sample target average treatment effect.

When `generalizability_path = TRUE`, the unweighted estimand corresponds to a STATE-type quantity and the weighted estimand to a WTATE-type quantity for the transported target population. When `generalizability_path = FALSE`, ROOT is used for general functional optimization and no causal labels are imposed; the summary focuses on the tree and diagnostics.

Diagnostics

The summary also reports:

- the number of trees grown,
- the size of the Rashomon set,
- the percentage of observations with ensemble vote $w_{\text{opt}} == 1$.

Examples

```
ROOT.output = ROOT(diabetes_data, generalizability_path = TRUE,  
                   num_trees = 20, top_k_trees = TRUE, k = 10, seed = 123)  
summary(ROOT.output)
```

Index

* **datasets**

diabetes_data, [6](#)

characterizing_underrep, [2](#), [16](#)

diabetes_data, [6](#)

plot.characterizing_underrep, [7](#)

plot.ROOT, [8](#)

print.characterizing_underrep, [9](#)

print.ROOT, [10](#)

ROOT, [2–6](#), [11](#)

summary.characterizing_underrep, [18](#)

summary.ROOT, [19](#)