Package: oda (via r-universe)

October 28, 2026

Title Pure-R Core Engine for Optimal Data Analysis (ODA / MultiODA)

Version 0.1.6

Description Pure-R implementation of univariate binary-class ODA (UniODA), univariate multiclass ODA (MultiODA), and binary Classification Tree Analysis (CTA). Supports ordered and categorical attributes, priors-on inverse-frequency weighting, MAXSENS / SAMPLEREP / first-identified tie-breaking, true leave-one-out cross-validation, and Monte Carlo Fisher-randomization p-values. Covered UniODA, MultiODA, and binary CTA fixtures are tested for parity against MegaODA.exe and CTA.exe outputs.

License GPL-3

Encoding UTF-8

Depends R (>= 4.1.0)

Imports graphics, grDevices, stats, utils

Suggests testthat (>= 3.0.0), knitr, rmarkdown, pkgdown, ggplot2 (>= 3.4.0), patchwork (>= 1.1.0)

Config/testthat/edition 3

VignetteBuilder knitr

URL https://njrhodes.github.io/oda_r/, https://github.com/njrhodes/oda_r

BugReports https://github.com/njrhodes/oda_r/issues

Config/roxygen2/version 8.1.0

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

Date/Publication 2026-10-27 22:50:01 UTC

RemoteUrl https://github.com/njrhodes/oda_r

RemoteRef HEAD

RemoteSha 86ac42eff10c82e4be35e927b270166b8cf77ca1

Contents

.lort_parent_maps . . . 4
as_confusion_matrix . . . 4
as_cta_candidates . . . 5
as_sda_anchor . . . 6
auto_sda_plan . . . 7
cta_assign_endpoints . . . 8
cta_balance_effect_summary . . . 10
cta_balance_plot_data . . . 12
cta_balance_table . . . 13
cta_confusion_matrix . . . 15
cta_confusion_table . . . 16
cta_d_stat . . . 17
cta_demo . . . 18
cta_descendant_family . . . 18
cta_endpoint_counts . . . 19
cta_endpoint_denominators . . . 20
cta_endpoint_summary . . . 21
cta_endpoint_table . . . 22
cta_family_table . . . 24
cta_fit . . . 25
cta_min_terminal_denom . . . 27
cta_node_table . . . 28
cta_observation_weights . . . 29
cta_ort_node_table . . . 31
cta_plot_data . . . 32
cta_propensity_weights . . . 34
cta_staging_table . . . 36
cta_strata . . . 38
lort_fit . . . 39
lort_index_path . . . 41
lort_local_tree . . . 41
lort_path_table . . . 42
lort_propensity_weights . . . 43
myeloma . . . 44
novo_boot_ci . . . 45
oda_balance_effect_table . . . 48
oda_balance_plot_data . . . 50
oda_balance_table . . . 52
oda_best_ordered_multiclass_partition . . . 54
oda_clean_missing_codes . . . 55
oda_confusion . . . 55
oda_confusion_binary . . . 56
oda_confusion_multiclass . . . 56
oda_cta_fit . . . 57
oda_d_stat . . . 59
oda_ess_from_mean . . . 60

oda_ess_from_meanpac . . . 60
oda_fit . . . 61
oda_infer_attr_types . . . 63
oda_loo_multiclass_ordered . . . 64
oda_mc_p_value . . . 65
oda_mean_pac . . . 66
oda_metrics . . . 67
oda_multiclass_unioda_core . . . 67
oda_power . . . 69
oda_predictions . . . 71
oda_propensity_weights . . . 71
oda_readiness_check . . . 72
oda_rule_predict . . . 73
oda_rule_predict_multiclass . . . 74
oda_sample_size . . . 74
oda_univariate_core . . . 76
oda_validate_group . . . 78
oda_validate_weights . . . 78
ort_plot_data . . . 79
plot.cta_ort . . . 79
plot.cta_tree . . . 81
plot_balance_love . . . 83
plot_cta_balance . . . 84
plot_cta_balance_effects . . . 85
plot_cta_family . . . 86
plot_cta_tree . . . 88
plot_lort_path . . . 90
plot_lort_tree . . . 91
plot_oda_balance . . . 93
plot_oda_balance_effects . . . 94
plot_smd_balance . . . 96
predict.cta_ort . . . 97
predict.cta_tree . . . 98
predict.oda_fit . . . 98
predict.sda_fit . . . 99
print.auto_sda_plan . . . 100
print.cta_family . . . 100
print.cta_family_summary . . . 101
print.cta_ort . . . 101
print.cta_ort_summary . . . 102
print.cta_tree . . . 102
print.cta_tree_summary . . . 103
print.oda_fit . . . 103
print.oda_fit_summary . . . 104
print.sda_anchor . . . 104
print.sda_fit . . . 105
print.sda_fit_summary . . . 105
propensity_ess_balance . . . 106

sda_anchor . . . 108
sda_candidate_table . . . 110
sda_fit . . . 110
sda_selected_attributes . . . 112
sda_step_table . . . 112
sda_to_cta_data . . . 113
smd_balance_table . . . 113
summary.cta_family . . . 114
summary.cta_ort . . . 115
summary.cta_tree . . . 116
summary.oda_fit . . . 117
summary.sda_anchor . . . 118
summary.sda_fit . . . 118
validate_sda_anchor . . . 119

Index **120**

`.lort_parent_maps` *Build parent map and endpoint-index map for LORT nodes (internal helper used by lort_index_path and lort_path_table)*

Description

Build parent map and endpoint-index map for LORT nodes (internal helper used by lort_index_path and lort_path_table)

Usage

```
.lort_parent_maps(ort_nodes)
```

Arguments

`ort_nodes`	Named list of LORT node objects from a `cta_ort` fit.

`as_confusion_matrix` *Convert a tidy confusion data frame to a 2x2 integer matrix*

Description

Converts the `data.frame` returned by `cta_confusion_table` (columns `actual`, `predicted`, `n`) to a 2x2 integer matrix suitable for `novo_boot_ci`.

Usage

```
as_confusion_matrix(df)
```

Arguments

df	A data.frame with integer columns actual, predicted, and n. Must represent a binary (2-class) classification with class labels 0 and 1.

Value

A 2x2 integer matrix with rows = actual class (0/1) and columns = predicted class (0/1), matching the training_confusion convention used throughout oda. Row and column names are "0" and "1".

See Also

cta_confusion_table, novo_boot_ci

Examples

```
# From raw data frame:
df <- data.frame(
  actual    = c(0L, 0L, 1L, 1L),
  predicted = c(0L, 1L, 0L, 1L),
  n         = c(146L, 40L, 36L, 33L)
)
m <- as_confusion_matrix(df)
novo_boot_ci(m, nboot = 200L, seed = 1L)

# From a fitted tree:

fit <- cta_fit(data.frame(x = seq_len(8L)),
               c(0L, 0L, 0L, 0L, 1L, 1L, 1L, 1L),
               mindenom = 2L, mc_iter = 100L, loo = "off")
ct <- cta_confusion_table(fit)
m  <- as_confusion_matrix(ct)
novo_boot_ci(m, nboot = 200L, seed = 42L)
```

---

as_cta_candidates *Subset a data frame to the SDA-selected candidate columns*

Description

Returns X restricted to the columns identified by sda_selected_attributes(fit). Intended to produce the constrained candidate frame for cta_fit or cta_descendant_family.

Usage

```
as_cta_candidates(fit, X)
```

Arguments

fit	An sda_fit object.
X	Data frame or matrix containing at least all selected attribute columns. Extra columns are dropped silently.

Value

Data frame with columns matching sda_selected_attributes(fit), in SDA step order.

---

as_sda_anchor *Convert an object to an* sda_anchor

Description

Generic converter. Methods are provided for sda_fit and data.frame. Use sda_anchor for direct construction.

Usage

```
as_sda_anchor(x, ...)

## S3 method for class 'sda_fit'
as_sda_anchor(x, ...)

## S3 method for class 'data.frame'
as_sda_anchor(
  x,
  selected_attributes,
  candidate_universe = NULL,
  group_levels = NULL,
  canon_notes = c("Explicit / manual anchor - user-declared stage table",
   "Not derived from sda_fit", "This anchor is for future SORT / staged CTA workflows",
    "SORT is not implemented", "GORT is not implemented"),
  ...
)
```

Arguments

x	A data frame with at least columns stage_id (integer) and attribute (character). Represents an explicit / manually-declared stage table.
...	Additional arguments passed to methods.
selected_attributes	Character vector of attribute names in stage order. Must match x$attribute entries.
candidate_universe	Character vector of all candidate attributes, or NULL (defaults to selected_attributes).
group_levels	Integer vector, or NULL.
canon_notes	Character vector describing the source.

Value

Object of class `c("sda_anchor", "list")`.

See Also

`sda_anchor`, `validate_sda_anchor`

`auto_sda_plan` *Dry-run planning and validation layer for SDA*

Description

Validates and constructs a candidate set for `sda_fit` without fitting. Returns an auditable plan object that records which columns were accepted, which were excluded and why, and what settings would be passed to `sda_fit()`.

Usage

```
auto_sda_plan(
  data,
  outcome,
  candidates = NULL,
  exclude = NULL,
  role_map = NULL,
  time_map = NULL,
  stage_map = NULL,
  attr_types = NULL,
  collinearity_threshold = 1,
  min_n = NULL,
  min_class_n = NULL,
  mode = c("unioda_max_ess", "novometric_min_d"),
  dry_run = TRUE
)
```

Arguments

`data`	A data frame.
`outcome`	Character scalar: name of the binary outcome column in `data`.
`candidates`	Character vector of candidate column names, or `NULL` (default) to use all non-outcome columns after exclusions.
`exclude`	Character vector of column names to force-exclude from candidates regardless of other checks.
`role_map`	Named list mapping column names to declared roles: `"assignment_mechanism"`, `"outcome"`, `"id"`, or `"leakage"`. Columns declared as `"id"` or `"leakage"` are excluded from the candidate set.

time_map	Named numeric/integer vector mapping column names to time indices. Columns with time_map value greater than the outcome's time index generate a warning flagging potential leakage. Temporal validity is a scientific judgment; auto_sda_plan flags but does not auto-exclude on this basis.
stage_map	Named integer vector mapping column names to stage assignments. Stored for downstream use; not used for exclusions.
attr_types	Named character vector mapping column names to declared attribute types ("ordered", "categorical", "binary"). Overrides type inference for named columns.
collinearity_threshold	Numeric threshold for collinearity detection. Default 1.0 detects exact-duplicate columns only.
min_n	Passed through to proposed_call for sda_fit().
min_class_n	Passed through to proposed_call.
mode	SDA mode: "unioda_max_ess" (legacy/iterative UniODA) or "novometric_min_d" (MPE-canon; per-attribute MDSA via cta_descendant_family(); requires mindenom).
dry_run	Logical. Must be TRUE (default). Fitting is not performed in SDA-3; dry_run = FALSE errors.

Details

Agent principle: auto_sda_plan() proposes and validates. It does not silently decide causal validity, temporal ordering, exposure roles, or outcome roles. If temporal or causal structure is required, declare it via role_map, time_map, or stage_map.

Value

Object of class c("auto_sda_plan", "odacore_plan").

---

cta_assign_endpoints *Assign observations to CTA terminal endpoints*

Description

Traverses the fitted cta_tree for each row of newdata and returns the terminal leaf reached, expressed as both its stored node identifier (endpoint_node_id) and its sequential endpoint index (endpoint_id) matching cta_endpoint_summary.

No endpoint membership is stored at fit time. This function performs the traversal on demand so the cta_tree object remains lean. The returned endpoint_id can be joined with the output of cta_propensity_weights to assign endpoint-level stabilized weights to individual observations.

Column order requirement: newdata must have the same attribute column order as the X matrix passed to oda_cta_fit. Traversal uses the stored integer column positions (attr_col) from the fit, not column names. If both names(newdata) and tree$attr_names are non-NULL, a warning is issued when they disagree at the split attribute positions.

Missingness:

`"na"` **(default)** Canonical path-local behaviour: when a split attribute value is `NA` or a stored miss-code on the observation's actual traversal path, the row returns `NA` for both output columns. This matches the canonical `missing_action = "na"` semantics of `predict`.

`"majority"` Routes the observation to the child subtree with the larger `n_obs`, then continues traversal to a terminal leaf. Ties are resolved by selecting the first child.

Usage

```
cta_assign_endpoints(tree, newdata, missing_action = c("na", "majority"))
```

Arguments

`tree`	A `cta_tree` from `oda_cta_fit`.
`newdata`	A `data.frame` (or coercible object) with the same column order as the training `X` supplied to `oda_cta_fit`.
`missing_action`	Character; one of `"na"` (default) or `"majority"`. See Description.

Details

Observation-level propensity weights (workflow sketch):

```
ep  <- cta_assign_endpoints(tree, X_train, missing_action = "na")
pw  <- cta_propensity_weights(tree, target_class = 1L, adjusted = TRUE)

# One row per classified training observation with its weight:
obs <- merge(
  data.frame(row_id = seq_len(nrow(X_train)),
             class  = as.character(y_train)),
  merge(ep, pw[, c("endpoint_id", "class", "adjusted_propensity_weight")],
        by = "endpoint_id"),
  by = c("row_id", "class")
)
# Rows with NA endpoint_id (missing root attribute) drop naturally.
```

Observation-level propensity weight expansion is intentionally left to the caller so that the `cta_tree` object stores no observation indices.

Value

A `data.frame` with one row per row of `newdata` and columns:

`row_id` Integer; positional row index in `newdata` (1 to `nrow(newdata)`).

`endpoint_node_id` Integer; `node_id` of the terminal leaf reached by traversal. `NA_integer_` when the observation cannot be routed to a terminal leaf (missing split attribute with `missing_action = "na"`, or no-tree fit).

`endpoint_id` Integer; sequential endpoint index matching `cta_endpoint_summary`. `NA_integer_` under the same conditions as `endpoint_node_id`.

For no-tree fits all rows have `endpoint_node_id = NA_integer_` and `endpoint_id = NA_integer_`.

See Also

oda_cta_fit, cta_endpoint_summary, cta_propensity_weights, predict.cta_tree

Examples

```
data(mtcars)
X    <- mtcars[, c("cyl", "disp", "hp", "wt")]
y    <- as.integer(mtcars$am)
tree <- oda_cta_fit(X, y, mindenom = 5L, mc_iter = 500L, mc_seed = 42L)
ep   <- cta_assign_endpoints(tree, X)
head(ep)
```

---

cta_balance_effect_summary

CTA covariate balance evidence-interval summary

Description

Builds one row per analysis scale (multivariate CTA) containing the observed full-tree ESS/WESS, a bootstrap confidence interval, and a chance interval. This is the multivariate analogue of oda_balance_effect_table: a single CTA ENUMERATE run per bootstrap or permutation iteration classifies all covariates jointly.

Usage

```
cta_balance_effect_summary(
  group,
  X,
  w = NULL,
  compare_weights = FALSE,
  mindenom = 1L,
  nboot = 200L,
  chance_iter = 200L,
  ci = 0.95,
  mc_seed = NULL,
  mc_iter = 5000L,
  ...
)
```

Arguments

group	Integer (or coercible) binary group indicator.
X	Data frame of baseline covariate columns.
w	Optional numeric case-weight vector.
compare_weights	Logical; when TRUE and w is supplied, produces two rows: "unweighted" and "weighted". Default FALSE.

mindenom	Integer minimum endpoint denominator. Default 1L.
nboot	Integer bootstrap resamples. Default 200L (CTA ENUMERATE is expensive per iteration).
chance_iter	Integer group-label permutations. Default 200L.
ci	Numeric nominal coverage. Default 0.95.
mc_seed	Integer RNG seed set once at function entry. NULL for unseeded.
mc_iter	Integer CTA MC iterations per node for the observed fit. Default 5000L.
...	Additional arguments forwarded to cta_fit for the observed fit (e.g., alpha_split, prune_alpha). mindenom, mc_iter, mc_seed, and loo are controlled internally.

Details

Three passes are run:

1. **Observed:** full cta_fit() with mc_iter – point estimate and tree metadata.
2. **Bootstrap:** nboot row-resamples, loo = "off" – ESS/WESS percentile CI. no_tree results contribute 0.
3. **Chance:** chance_iter group-label permutations – null percentile interval. no_tree results contribute 0.

no_tree convention: when CTA finds no admissible tree on a bootstrap or chance iteration, ESS = 0 (no discrimination above chance). The observed no_tree result is also recorded as estimate = 0.

Value

A list of class "cta_balance_effect_summary" with:

rows Data frame; one row per analysis scale. Columns: analysis, metric, estimate, boot_lo, boot_hi, chance_lo, chance_hi, d_stat, n_endpoints, root_attribute, status, balance_interpretation.

meta List: n_obs, has_weights, compare_weights, analyses, mindenom, nboot, chance_iter, ci, mc_iter, mc_seed.

References

Linden A, Yarnold PR (2016). Using machine learning to assess covariate balance in matching studies. *Journal of Evaluation in Clinical Practice*, **22**(6), 861-867.

See Also

cta_balance_table, plot_cta_balance_effects

Examples

```
X <- data.frame(
  A = c(rep(0L, 20), rep(1L, 20), rep(1L, 20)),
  B = c(rep(0L, 20), rep(0L, 20), rep(1L, 20))
)
group <- c(rep(0L, 40), rep(1L, 20))
```

```
ces <- cta_balance_effect_summary(group, X, mindenom = 5L,
                                  mc_iter = 200L, mc_seed = 42L,
                                  nboot = 20L, chance_iter = 20L)
ces$rows[, c("analysis", "estimate", "boot_lo", "boot_hi",
             "chance_lo", "chance_hi", "status")]
```

`cta_balance_plot_data` *Renderer-ready plot data for CTA covariate balance*

Description

Transforms a `cta_balance_table` result into a renderer-independent data structure suitable for Graphics v3 plotting. For `no_tree` results, populates `no_tree_message` with the favorable-balance interpretation.

Usage

```
cta_balance_plot_data(cta_balance, target_class = 1L, digits = 1L)
```

Arguments

`cta_balance`	A `"cta_balance_table"` object from `cta_balance_table`.
`target_class`	Integer; target class for endpoint coloring in the embedded tree diagram. Default `1L` (group 1 = treated).
`digits`	Integer; decimal digits passed to `cta_plot_data`. Default `1L`.

Details

This function does not fit any CTA models. It is a pure transformation of the pre-computed `cta_balance_table` result.

Value

A list of class `"cta_balance_plot_data"` with elements:

`status` Character; `"valid_tree"`, `"stump"`, `"no_tree"`, or `"fit_error"`.

`balance_interpretation` Character.

`no_tree_message` Character; human-readable no-tree annotation for renderers; `NA` when status is not `"no_tree"`.

`cta_pd` List from `cta_plot_data` when a valid tree or stump was found; `NULL` for no_tree or fit_error.

`ess_display` Numeric; full-tree ESS/WESS (%); `NA` for no_tree.

`d_stat` Numeric; `NA` for no_tree.

`has_weights` Logical.

`ess_label` Character; `"WESS"` or `"ESS"`.

See Also

cta_balance_table, cta_plot_data

Examples

```
X <- data.frame(
  A = c(rep(0L, 20), rep(1L, 20), rep(1L, 20)),
  B = c(rep(0L, 20), rep(0L, 20), rep(1L, 20))
)
group <- c(rep(0L, 40), rep(1L, 20))
ct  <- cta_balance_table(group, X, mindenom = 5L,
                         mc_iter = 200L, mc_seed = 42L)
cpd <- cta_balance_plot_data(ct)
cpd$status
```

---

cta_balance_table *Multivariate CTA covariate balance diagnostics*

Description

Fits a single cta_fit model with group as the class variable and all columns of X as candidate predictors. Returns a structured summary of the CTA balance result.

Usage

```
cta_balance_table(
  group,
  X,
  w = NULL,
  mindenom = 1L,
  alpha = 0.05,
  loo = "off",
  mc_iter = 5000L,
  mc_seed = NULL,
  ...
)
```

Arguments

group	Integer (or coercible) binary group indicator. Must have exactly two distinct non-missing values.
X	Data frame of baseline covariate columns.
w	Optional numeric case-weight vector. When supplied, CTA uses case weights and has_weights = TRUE in the result.
mindenom	Integer minimum endpoint denominator passed to cta_fit. Default 1L.

alpha	Numeric significance threshold stored in the result and used in the no_tree_message of cta_balance_plot_data. Default 0.05. Does not override alpha_split; pass alpha_split via ... to change the CTA node-level threshold.
loo	LOO gate mode passed to cta_fit. Default "off".
mc_iter	Integer MC iterations per CTA node. Default 5000L.
mc_seed	Integer RNG seed; NULL for unseeded.
...	Additional arguments forwarded to cta_fit (e.g., alpha_split, prune_alpha, priors_on).

Details

A status = "no_tree" result means no combination of baseline covariates in X predicted group membership at the declared significance level, LOO constraint, and minimum endpoint denominator. This is **favorable evidence of multivariable covariate balance** under the declared analytic constraints. It must not be interpreted as a model failure; in balance analysis, inability to discriminate groups is the goal.

group vs. outcome: group is the binary class variable. The scientific outcome is strictly out of scope.

Implementation constraint: this function calls cta_fit once; it does not reimplement ENUMERATE or node-growth logic.

Value

A list of class "cta_balance_table" with fields:

status Character: "valid_tree", "stump", "no_tree", or "fit_error".

balance_interpretation Character: "discriminating" or "no_discriminating_combinations" (when no_tree); NA on fit error.

root_attribute Character; root split variable name; NA when no_tree.

n_endpoints Integer; number of terminal endpoints; NA when no_tree.

overall_ess Numeric; full-tree ESS (%) when weights not active; NA otherwise.

overall_wess Numeric; full-tree WESS (%) when weights active; NA otherwise.

ess_display Numeric; operative measure (overall_wess when weights active, else overall_ess); NA for no_tree.

d_stat Numeric; parsimony-adjusted D statistic; NA for no_tree.

mindenom Integer; MINDENOM used.

alpha Numeric; significance threshold stored for downstream use.

has_weights Logical; whether case weights were active.

tree The raw cta_tree object; NULL on fit error.

endpoint_table Data frame from cta_endpoint_table; zero-row for no_tree.

node_table Data frame from cta_node_table.

fit_error Logical; TRUE when cta_fit threw.

fit_reason Character; error message when fit_error; NA otherwise.

References

Linden A, Yarnold PR (2016). Using machine learning to assess covariate balance in matching studies. *Journal of Evaluation in Clinical Practice*, **22**(6), 861-867.

See Also

cta_balance_plot_data, oda_balance_table, cta_fit

Examples

```
X <- data.frame(
  A = c(rep(0L, 20), rep(1L, 20), rep(1L, 20)),
  B = c(rep(0L, 20), rep(0L, 20), rep(1L, 20))
)
group <- c(rep(0L, 40), rep(1L, 20))
ct <- cta_balance_table(group, X, mindenom = 5L,
                        mc_iter = 200L, mc_seed = 42L)
ct$status
ct$balance_interpretation
```

---

cta_confusion_matrix *Extract training confusion matrix from a fitted CTA tree*

Description

Convenience wrapper: returns the 2x2 integer training confusion matrix for a binary oda_cta_fit result directly, without the intermediate tidy long-format step required by cta_confusion_table and as_confusion_matrix.

Usage

```
cta_confusion_matrix(tree)
```

Arguments

tree	A cta_tree from oda_cta_fit.

Details

Rows are actual class (0/1), columns are predicted class (0/1). Returns NULL invisibly when tree$no_tree is TRUE.

Value

A 2x2 integer matrix (actual x predicted) or NULL when no tree was found.

See Also

cta_confusion_table, as_confusion_matrix

Examples

```
data(mtcars)
X    <- mtcars[, c("cyl", "disp", "hp", "wt")]
y    <- as.integer(mtcars$am)
tree <- oda_cta_fit(X, y, mindenom = 5L, mc_iter = 500L, mc_seed = 42L)
if (!isTRUE(tree$no_tree)) cta_confusion_matrix(tree)
```

---

cta_confusion_table *Final selected tree training confusion table*

Description

Returns the stored full-tree training confusion matrix for the final selected CTA model in tidy long format (one row per actual x predicted class pair).

The confusion matrix is captured at fit time at the exact moment the winning candidate is selected, using the same scoring predictions. For the expanded ENUMERATE phase, predictions use majority-fallback for missing attributes. For the root-only stump phase, predictions are path-local (observations whose root attribute is missing are excluded).

This function does **not** report split-node local confusion. Split-node confusion reflects all observations at a node classified by that node's rule alone; it is not the same as full-tree confusion for trees with more than one split. The two coincide incidentally for stumps but the semantics here are always final-tree.

Usage

```
cta_confusion_table(tree)
```

Arguments

`tree`	A `cta_tree` from `oda_cta_fit`.

Value

A `data.frame` with columns:

`actual` Integer actual class label.

`predicted` Integer predicted class label.

`n` Integer raw count of observations with this actual x predicted combination in the final selected tree.

Rows are sorted by `actual` then `predicted`. For a no-tree fit (or if `training_confusion` is absent), the returned data frame has zero rows but the correct column structure.

See Also

`oda_cta_fit`, `summary.cta_tree`, `cta_endpoint_table`, `cta_node_table`

Examples

```
data(mtcars)
X    <- mtcars[, c("cyl", "disp", "hp", "wt")]
y    <- as.integer(mtcars$am)
tree <- oda_cta_fit(X, y, mindenom = 5L, mc_iter = 500L, mc_seed = 42L)
cta_confusion_table(tree)
```

`cta_d_stat` *D statistic for a fitted CTA tree*

Description

Computes the parsimony-normalized classification criterion:

Usage

```
cta_d_stat(tree)
```

Arguments

`tree`	A `cta_tree` from `oda_cta_fit`.

Details

$$D = \frac{100}{\mathrm{ESS}/\mathrm{strata}} - \mathrm{strata}$$

where `strata` is the number of terminal leaf endpoints and `ESS` is `tree$overall_ess` (WESS when case weights are active, ESS otherwise).

Returns `NA_real_` when:

- `tree$no_tree` is `TRUE`;
- `tree$overall_ess` is missing, non-finite, or ≤ 0;
- `strata` < 2.

Value

Numeric scalar D, or `NA_real_`.

See Also

`cta_strata`, `cta_min_terminal_denom`

cta_demo *CTA demonstration dataset*

Description

A simulated data frame with 200 observations and 6 variables, designed to illustrate Classification Tree Analysis with `cta_fit`. This is the dataset used in the CTA.exe demonstration program.

Format

A data frame with 200 rows and 6 columns:

V1 Class label (integer; 1 or 2).

V2 Ordered attribute (root in MINDENOM = 1 solution).

V3 Ordered attribute.

V4 Binary attribute (0/1).

V5 Ordered attribute.

V6 Ordered attribute.

Details

The CTA.exe golden output for MINDENOM = 1 selects V2 as root (cut = 4.5, ESS = 52.63%). MINDENOM = 8 requires `mc_iter = 25000` for parity.

Simulated dataset; no real subjects or PHI. Used as the primary introductory CTA example in the oda package vignettes and in the CTA.exe demonstration program (`CTA_DEMO.pgm`).

cta_descendant_family *MDSA descendant family for CTA*

Description

Traces the MDSA descendant family by fitting CTA models starting at `start_mindenom` and stepping according to the novometric MDSA rule: next MINDENOM = minimum terminal endpoint denominator + 1. The family terminates when a no-tree fit is produced or `max_steps` is reached.

Usage

```
cta_descendant_family(
  X,
  y,
  w = NULL,
  ...,
  start_mindenom = 1L,
  max_steps = 20L
)
```

Arguments

X	Data frame of predictor attributes; passed to oda_cta_fit.
y	Integer class vector; passed to oda_cta_fit.
w	Optional numeric case-weight vector; passed to oda_cta_fit.
...	Additional arguments forwarded to oda_cta_fit (e.g. alpha_split, prune_alpha, mc_iter, mc_seed, loo, miss_codes, verbose).
start_mindenom	Integer MINDENOM for the first family member. Defaults to 1L.
max_steps	Integer safety cap on the number of CTA fits; prevents unbounded loops. Defaults to 20L.

Value

A list of class cta_family with fields:

members List of new_cta_family_member objects in order, including the terminal no-tree member.

mindenoms Integer vector of MINDENOM values tried.

summary Data frame with one row per member: mindenom, status ("valid_tree", "stump", or "no_tree"), strata, min_terminal_denom, overall_ess, d, no_tree.

min_d_idx Integer index of the feasible (non-no-tree) member with minimum D; NA_integer_ if no feasible member exists.

terminated Logical; always TRUE.

termination_reason Character: one of "no_tree", "max_steps", "no_next_mindenom".

See Also

oda_cta_fit, cta_d_stat, cta_min_terminal_denom, cta_strata

---

cta_endpoint_counts *Per-endpoint class count table for a fitted CTA tree*

Description

Returns one row per terminal endpoint (leaf) per actual class, read directly from stored leaf node fields. No refitting, no prediction, and no recomputation from training data is performed.

Class counts are stored at fit time by oda_cta_fit on every terminal leaf. Row order within each endpoint follows the order of names(leaf$class_counts_raw), which is ascending by class label. Endpoints are ordered by node_id, matching cta_endpoint_summary.

Scope: This function exposes stored raw and weighted class counts only. It does *not* include target-class proportions, event rates, odds, or staging order. Staging-table and event-rate summaries are available via cta_staging_table.

If any terminal leaf is missing the stored class counts (i.e., the cta_tree was fitted by an earlier version of oda that did not store endpoint counts), the function stops with a clear error.

Usage

```
cta_endpoint_counts(tree)
```

Arguments

tree	A cta_tree from oda_cta_fit.

Value

A data.frame with one row per terminal endpoint per actual class and columns:

endpoint_id Integer sequential endpoint index 1..n in node order, matching cta_endpoint_summary.

endpoint_node_id Integer tree node identifier for this endpoint leaf.

path Character; AND-joined branch labels from root to this leaf (e.g. "V14<=0.5 AND V15>0.5").

terminal_prediction Integer class label assigned to this endpoint (stored leaf majority_class).

class Character; actual class label for this row (e.g. "0", "1").

n_raw Integer raw count of observations of this actual class reaching this endpoint.

n_weighted Numeric weighted total for this actual class reaching this endpoint. Equals n_raw when case weights are not active.

For a no-tree fit the returned data frame has zero rows but the correct column structure and types.

See Also

oda_cta_fit, cta_endpoint_summary, cta_confusion_table, cta_endpoint_table, cta_staging_table, cta_propensity_weights

Examples

```
data(mtcars)
X    <- mtcars[, c("cyl", "disp", "hp", "wt")]
y    <- as.integer(mtcars$am)
tree <- oda_cta_fit(X, y, mindenom = 5L, mc_iter = 500L, mc_seed = 42L)
cta_endpoint_counts(tree)
```

---

cta_endpoint_denominators

Terminal endpoint denominators of a CTA tree

Description

Returns the observation counts (n_obs) for each terminal leaf node, named by node ID. These are the raw row counts stored at fit time - they are not recomputed from training data or predictions.

Usage

```
cta_endpoint_denominators(tree)
```

Arguments

tree	A cta_tree from oda_cta_fit.

Details

Returns integer(0) for no-tree fits.

Value

Named integer vector of leaf n_obs values, named by node ID (as character); integer(0) for no-tree fits.

See Also

cta_strata, cta_min_terminal_denom

---

cta_endpoint_summary *Endpoint reporting summary for a fitted CTA tree*

Description

Returns one row per terminal leaf (endpoint) with stable endpoint identifiers and stored node fields suitable for downstream reporting. All values are read directly from stored node fields; no refitting, no prediction, and no recomputation of tree metrics is performed.

Scope: This function reports structural endpoint fields only. It does *not* include endpoint class counts, target-class proportions, event rates, odds, or staging order. Per-endpoint class counts are available via cta_endpoint_counts. Staging-table and event-rate summaries are available via cta_staging_table.

Usage

```
cta_endpoint_summary(tree)
```

Arguments

tree	A cta_tree from oda_cta_fit.

Value

A data.frame with one row per terminal leaf and columns:

endpoint_id Integer sequential index 1..n in node order.

endpoint_node_id Integer tree node identifier for this leaf, corresponding to node_id in cta_endpoint_table.

path Character; AND-joined branch labels from root to this leaf (e.g. "V14<=0.5 AND V15>0.5").

depth Integer depth from root (root = 1).

terminal_prediction Integer class label assigned to this endpoint (stored leaf majority_class).

`n_obs` Integer raw observation count at this endpoint.

`n_weighted` Numeric weighted observation count. Equals `n_obs` when case weights are not active (not `NA`).

`denominator` Integer endpoint denominator (equal to `n_obs`); included to align with MPE/MDSA terminology.

For a no-tree fit the returned data frame has zero rows but the correct column structure and types.

See Also

`oda_cta_fit`, `cta_endpoint_table`, `cta_strata`, `cta_endpoint_denominators`, `cta_endpoint_counts`, `cta_staging_table`

Examples

```
data(mtcars)
X    <- mtcars[, c("cyl", "disp", "hp", "wt")]
y    <- as.integer(mtcars$am)
tree <- oda_cta_fit(X, y, mindenom = 5L, mc_iter = 500L, mc_seed = 42L)
cta_endpoint_summary(tree)
```

`cta_endpoint_table` *Canonical terminal endpoint map for a fitted CTA tree*

Description

Returns one row per terminal leaf (endpoint) of a `cta_tree`. All values are read directly from stored node fields; no refitting or prediction is performed. This is the canonical endpoint map for reporting, translation, ORT, and staged workflows.

Leaf class counts are stored on every terminal node at fit time (`class_counts_raw`, `class_counts_weighted`). `target_n` and `target_prop` are derived from the stored counts.

ESS, WESS, p, LOO status, LOO ESS/WESSL, and LOOp are canonical split-node report metrics (see `cta_node_table`). Terminal endpoints are connected to those metrics through their parent split-node lineage. The `parent_split_*` columns expose the immediate parent split's canonical metrics for auditability. They are not recomputed ESS at the leaf.

Usage

```
cta_endpoint_table(tree, target_class = NULL)
```

Arguments

`tree`	A `cta_tree` from `oda_cta_fit`.
`target_class`	Integer class label to use as the target (positive) class for `target_n` and `target_prop`. When `NULL` (default), the function auto-detects: for binary trees with classes `0` and `1`, class `1` is used; otherwise `target_n` and `target_prop` are `NA`.

Value

A `data.frame` with one row per terminal leaf and columns:

`endpoint_id` Integer sequential endpoint index 1..n.

`leaf_node_id` Integer tree node identifier for this leaf.

`terminal_marker` Character `"*"` on every row.

`terminal` Logical `TRUE` on every row.

`depth` Integer depth from root (root = 1).

`parent_split_node_id` Integer parent split node identifier.

`path` Character; AND-joined branch labels from root to this leaf (e.g. `"V14<=0.5 AND V15>0.5"`).

`n` Integer raw observation count at this endpoint.

`class_counts_raw` List column; each element is a named integer vector of raw per-class counts, or `NULL`.

`class_counts_weighted` List column; each element is a named numeric vector of weighted per-class counts, or `NULL`.

`predicted_class` Integer class label assigned to this endpoint (stored leaf majority class).

`target_n` Integer count of `target_class` observations at this endpoint (`NA` when not resolvable).

`target_prop` Numeric proportion `target_n / n` (`NA` when not resolvable).

`parent_split_attribute` Attribute name of the parent split.

`parent_split_ess` ESS of the parent split node.

`parent_split_wess` WESS of the parent split node.

`parent_split_loo_status` LOO status of the parent split node.

`parent_split_loo_ess` LOO ESS/WESSL of the parent split node.

`parent_split_p_mc` MC p-value of the parent split node.

For a no-tree fit the returned data frame has zero rows but the correct column structure and types.

See Also

`oda_cta_fit`, `cta_node_table`, `summary.cta_tree`, `cta_strata`, `cta_endpoint_denominators`, `cta_endpoint_summary`, `cta_endpoint_counts`

Examples

```
data(mtcars)
X    <- mtcars[, c("cyl", "disp", "hp", "wt")]
y    <- as.integer(mtcars$am)
tree <- oda_cta_fit(X, y, mindenom = 5L, mc_iter = 500L, mc_seed = 42L)
cta_endpoint_table(tree)
```

`cta_family_table` *Tidy table of a CTA descendant family*

Description

Returns a `data.frame` with one row per family member, reading all values from the stored `cta_family` object. No refitting or recomputation is performed.

Usage

```
cta_family_table(family)
```

Arguments

`family`	A `cta_family` from `cta_descendant_family`.

Value

A `data.frame` with columns:

`index` Integer position of the member in the chain.

`mindenom` Integer MINDENOM used for this fit.

`status` Character: `"valid_tree"`, `"stump"`, or `"no_tree"`.

`no_tree` Logical; `TRUE` for the terminal no-tree member.

`strata` Integer number of terminal leaf endpoints; `NA` for no-tree members.

`min_terminal_denom` Integer minimum leaf `n_obs`; `NA` for no-tree members.

`next_mindenom` Integer MINDENOM for the next chain step (`min_terminal_denom + 1`); `NA` for no-tree members.

`overall_ess` Numeric overall ESS or WESS stored at fit time; `NA` for no-tree members.

`has_weights` Logical; `TRUE` when case weights were active for this fit.

`d` Numeric D statistic (`100 / (ESS / strata) - strata`); `NA` for no-tree members.

`selected_min_d` Logical; `TRUE` for the feasible member with minimum D (index `family$min_d_idx`). All `FALSE` when no feasible member exists.

See Also

`cta_descendant_family`, `summary.cta_family`

Examples

```
data(mtcars)
X <- mtcars[, c("cyl", "disp", "hp", "wt")]
y <- as.integer(mtcars$am)
fam <- suppressMessages(
  cta_descendant_family(X, y, start_mindenom = 1L, mc_iter = 200L,
                        mc_seed = 42L, loo = "off")
)
cta_family_table(fam)
```

`cta_fit` *Fit a Classification Tree Analysis (CTA) model (public wrapper)*

Description

Public entry point for CTA. Currently supports binary (two-class) outcome variables only.

When `recursive = FALSE` (default), validates the class variable and delegates to `oda_cta_fit`. When `recursive = TRUE`, runs the Locally Optimal Recursive Tree (LORT) engine: at each endpoint a full MDSA family scan (`cta_descendant_family`) is performed, the min-D member is selected, and recursion continues until no further structure is found or a compute guard fires. Returns a dual-tagged `cta_ort` / `cta_tree` object.

Usage

```
cta_fit(X, y, verbose = FALSE,
        recursive        = FALSE,
        min_n            = 30L,
        max_depth        = 8L,
        max_nodes        = 31L,
        family_max_steps = 20L,
        ...)
```

Arguments

`X`	Data frame or matrix of attribute columns. For recursive CTA, `X` is the declared candidate predictor frame; pass only variables eligible for CTA search. Prediction may be performed on wider `newdata` as long as the split variable names are present.
`y`	Integer class variable vector. Must have exactly two distinct values.
`verbose`	Logical; if `TRUE`, emit `[CTA]` and `[ORT]` progress messages. Default `FALSE`.
`recursive`	Logical; if `TRUE`, run the Locally Optimal Recursive Tree (LORT) engine. Default `FALSE`. Cannot be combined with an explicit `mindenom` argument (error).
`min_n`	Integer; minimum endpoint n to attempt recursion. Endpoints smaller than `min_n` become terminal with stop reason `"min_n"`. Default `30L`. Only used when `recursive = TRUE`.

`max_depth` Integer; safety cap on recursion depth. Nodes at `depth >= max_depth` become terminal with stop reason `"max_depth"`. Default `8L`. Only used when `recursive = TRUE`.

`max_nodes` Integer; safety cap on total ORT nodes allocated. When the node count exceeds `max_nodes` the current endpoint becomes terminal with stop reason `"max_nodes"`. Default `31L`. Only used when `recursive = TRUE`.

`family_max_steps` Integer or `NULL` (default `20L`); maximum number of MDSA family members evaluated at each recursive node. This bounds the per-node `cta_descendant_family` scan (its `max_steps` argument). The default of `20L` preserves `cta_descendant_family()` behavior. Smaller values reduce the per-node compute budget at the cost of possibly missing the true min-D member if the family is long. Stored in `ort_settings$family_max_steps`. Only used when `recursive = TRUE`; error if explicitly supplied with `recursive = FALSE`.

`...` Additional arguments passed to `oda_cta_fit` (non-recursive) or used as ORT settings (recursive). Supported ORT settings via `...`: `w`, `mc_seed`, `mc_iter`, `alpha_split`, `prune_alpha`, `loo`. When `recursive = TRUE`, `mc_seed` initializes the RNG once at ORT start; child-node MDSA scans consume the stream in deterministic right-then-left traversal order without resetting the seed.

Value

Non-recursive: a `cta_tree` object.

Recursive: a dual-tagged `cta_ort` / `cta_tree` object. All existing `cta_tree` S3 methods (`predict`, `print`, `summary`, `plot`) operate on the root-level model. `cta_ort`-aware methods (`predict.cta_ort`, `print.cta_ort`, `summary.cta_ort`, `plot.cta_ort`) operate on the full composite tree. Use `predict(obj, newdata, type="all")` to retrieve stratum assignments.

Note

`oda_cta_fit()` is the internal engine name; `cta_fit()` is the preferred public entry point for non-recursive CTA. Both are exported and functionally equivalent for non-recursive use.

`cta_fit(..., recursive = TRUE)` is a legacy-compatible interface for the LORT workflow layer. Prefer `lort_fit()` for new code. SORT and GORT are reserved and not implemented.

See Also

`oda_fit`, `cta_descendant_family`, `cta_node_table`, `cta_staging_table`, `plot.cta_tree`, `plot.cta_ort`, `ort_plot_data`

Examples

```
# Small synthetic two-class example (non-recursive)
X <- data.frame(
  x1 = c(1, 2, 3, 4, 5, 6, 7, 8),
  x2 = c(0L, 0L, 1L, 0L, 1L, 1L, 0L, 1L)
)
y <- c(1L, 1L, 1L, 1L, 2L, 2L, 2L, 2L)
```

```
tree <- cta_fit(X, y,
  priors_on   = TRUE,
  mindenom    = 1L,
  mc_iter     = 500L,
  mc_seed     = 42L,
  loo         = "off",
  attr_names  = c("x1", "x2")
)
print(tree)

# Recursive ORT  -  two-level synthetic dataset
X2 <- data.frame(
  A = c(rep(0, 20), rep(1, 20), rep(1, 20)),
  B = c(rep(0, 20), rep(0, 20), rep(1, 20))
)
y2 <- c(rep(0L, 20), rep(0L, 20), rep(1L, 20))
ort <- cta_fit(X2, y2, recursive = TRUE,
               mc_iter = 100L, mc_seed = 42L, loo = "off",
               min_n = 5L)
print(ort)
```

`cta_min_terminal_denom` *Minimum terminal endpoint denominator of a CTA tree*

Description

Returns the smallest leaf `n_obs` across all terminal endpoints. This value drives the next MINDENOM step in the MDSA descendant family: `next_mindenom = cta_min_terminal_denom(tree) + 1L`.

Usage

```
cta_min_terminal_denom(tree)
```

Arguments

`tree`	A `cta_tree` from `oda_cta_fit`.

Details

Returns `NA_integer_` for no-tree fits.

Value

Minimum leaf `n_obs` as an integer, or `NA_integer_` for no-tree fits.

See Also

cta_strata, cta_endpoint_denominators

---

cta_node_table *Canonical CTA node report table*

Description

Returns a data frame with one row per node, mirroring the CTA.exe-style node report (ATTRIBUTE, NODE, LEV, OBS, p, ESS/WESS, LOO, WESSL/LOO ESS, LOOp, TYP, MODEL columns).

Split nodes carry canonical split metrics (ESS, WESS, p, LOO status, LOO ESS/WESSL, LOOp) and a MODEL field with branch strings and terminal-leaf * markers. Leaf rows have NA for all split metrics and MODEL.

Usage

```
cta_node_table(tree)
```

Arguments

tree	A cta_tree from oda_cta_fit.

Value

Data frame with columns:

node_id Integer node identifier.

parent_id Integer parent node identifier (0 for root).

level Integer level from root (root = 1); alias for depth.

depth Integer depth from root (root = 1).

leaf Logical; TRUE for terminal leaf nodes.

attribute Character attribute name (NA for leaves).

attr_type Character attribute type (NA for leaves).

n_obs Integer observation count at this node.

n_weighted Numeric weighted observation count.

p_mc Numeric Monte Carlo p-value (NA for leaves).

ess Numeric ESS at this split (NA for leaves).

ess_weighted Numeric WESS at this split (NA for leaves); equals ess when case weights are not active.

loo_status Character LOO status, e.g. "STABLE" (NA for leaves).

loo_ess Numeric LOO ESS/WESSL (NA for leaves).

loo_p Numeric LOO p-value (NA for leaves).

model Character CTA.exe-style branch string with terminal-leaf * markers, e.g. "<=0.5-->0,101/131,77.10%*; >0.5-->1,21/55,38.18%*". NA for leaf nodes.

See Also

oda_cta_fit, cta_endpoint_table, summary.cta_tree

---

cta_observation_weights *Assign per-observation CTA propensity weights*

Description

Convenience wrapper that calls cta_assign_endpoints and cta_propensity_weights and returns a joined observation-level data frame. The cta_tree object is not mutated; all computation is on demand.

Column order requirement: newdata must have the same attribute column order as the X matrix passed to oda_cta_fit. Traversal uses the stored integer column positions (attr_col) from the fit, not column names.

Unroutable observations: Observations with NA endpoint (missing root split attribute under missing_action = "na") or NA class label receive assigned = FALSE and NA for all weight columns. The output always contains nrow(newdata) rows.

Unmatched classified observations: When a non-NA endpoint observation's class is not present in the propensity weight table (e.g., a class unseen at fit time), a warning is issued and assigned = FALSE.

Usage

```
cta_observation_weights(tree, newdata, y, target_class = NULL,
                        adjusted = TRUE,
                        missing_action = c("na", "majority"))
```

Arguments

tree	A cta_tree from oda_cta_fit.
newdata	A data.frame (or coercible object) with the same column order as the training X.
y	Class labels for each row of newdata. Any type coercible to character; length must equal nrow(newdata).
target_class	Passed to cta_propensity_weights as an annotation parameter. Identifies which class is treated as the design target (high-risk class) for the target_class output column; it does *not* filter the endpoint × class rows used for the join. Each observation is matched to its own actual_class regardless of this value. NULL (default) lets cta_propensity_weights resolve the target class automatically (numerically largest class for binary trees; must be supplied explicitly for trees with three or more classes).
adjusted	Logical; passed to cta_propensity_weights. Default TRUE.
missing_action	Character; one of "na" (default) or "majority". Passed to cta_assign_endpoints.

Details

No observation-level data are stored in the cta_tree object at fit time. This function performs traversal and weight lookup on demand.

No-tree fits: When the tree has no splits (leaf-only), all rows have endpoint_id = NA_integer_ and assigned = FALSE.

Join semantics: The join key is paste(endpoint_id, actual_class). Each observation is matched to the propensity weight row whose class equals its actual_class. The target_class parameter annotates all rows with the resolved design target class but does not affect which rows participate in the join.

Value

A data.frame with nrow(newdata) rows and columns:

row_id Integer; positional row index (1 to nrow(newdata)).

actual_class Character; class label from y, coerced to character.

endpoint_node_id Integer; node ID of the terminal leaf reached by traversal, or NA_integer_ when unroutable.

endpoint_id Integer; sequential endpoint index matching cta_endpoint_summary, or NA_integer_.

target_class Integer; resolved design target class annotation from cta_propensity_weights, or NA_integer_ when unassigned.

propensity_weight Numeric; unadjusted propensity weight for the observation's endpoint–class cell, or NA when unassigned.

adjusted_propensity_weight Numeric; adjusted propensity weight (Yarnold-Linden correction for perfectly predicted endpoints), or NA when unassigned.

undefined_empirical Logical; TRUE when the endpoint–class cell has zero observed frequency, or NA when unassigned.

perfectly_predicted_endpoint Logical; TRUE when all observations at the endpoint belong to one class, or NA when unassigned.

adjusted Logical; TRUE when the adjusted weight was applied at this endpoint, or NA when unassigned.

assigned Logical; TRUE when a propensity weight was successfully matched for this observation.

See Also

cta_assign_endpoints, cta_propensity_weights, oda_cta_fit, cta_endpoint_summary

Examples

```
data(mtcars)
X    <- mtcars[, c("cyl", "disp", "hp", "wt")]
y    <- as.integer(mtcars$am)
tree <- oda_cta_fit(X, y, mindenom = 5L, mc_iter = 500L, mc_seed = 42L)
ow   <- cta_observation_weights(tree, X, y)
head(ow)
```

`cta_ort_node_table` *Node-level summary table for a fitted LORT (legacy name: cta_ort)*

Description

Returns one row per LORT node from a `cta_ort` (LORT) object. Each row exposes the embedded CTA member selected at that node (MINDENOM, ESS, D, root attribute, split/leaf counts, endpoint count) plus the LORT method taxonomy metadata (`method`, `selection_scope`, `global_optimization`, `sda_anchored`).

Terminal nodes have `NA` for all selected-model columns. Non-terminal nodes have `NA` for `stop_reason` and non-empty `child_ids`.

Naming note: The function name `cta_ort_node_table` and the class `cta_ort` are legacy compatibility names for the implemented LORT method. They are retained for backward compatibility; new code and documentation should refer to the method as LORT.

Usage

```
cta_ort_node_table(object)
```

Arguments

`object`	A `cta_ort` from `cta_fit(..., recursive = TRUE)`.

Value

A `data.frame` with one row per ORT node and columns:

`ort_node_id` Integer ORT node identifier.

`parent_ort_node_id` Integer parent ORT node id; `NA` for root.

`depth` Integer recursion depth (root = 0).

`n` Integer observations at this ORT node.

`class_counts` Character; named class counts, e.g. `"0=60, 1=40"`.

`terminal` Logical; `TRUE` for terminal leaf ORT nodes.

`stop_reason` Character stop reason for terminal nodes; `NA` for non-terminal.

`selected_mindenom` Integer MINDENOM of the embedded CTA member.

`selected_ess` Numeric ESS of the embedded CTA member (%).

`selected_d` Numeric D-statistic of the embedded CTA member.

`selected_root_attribute` Character root attribute of the embedded CTA member.

`selected_tree_nodes` Integer split-node count in the embedded CTA member.

`selected_tree_leaves` Integer leaf count in the embedded CTA member.

`selected_endpoint_count` Integer endpoint (terminal leaf) count of the embedded CTA member; equals number of ORT child nodes.

`child_ids` Character comma-separated child ORT node ids; empty string for terminal nodes.

`method` Character; always `"lort"` for current fits.

`selection_scope` Character; always `"local_node"` for LORT.

`global_optimization` Logical; always `FALSE` for LORT.

`sda_anchored` Logical; always `FALSE` for LORT.

See Also

`cta_fit`, `predict.cta_ort`, `summary.cta_ort`

Examples

```
X <- data.frame(A = c(rep(0L, 20), rep(1L, 20), rep(1L, 20)),
                B = c(rep(0L, 20), rep(0L, 20), rep(1L, 20)))
y <- c(rep(0L, 20), rep(0L, 20), rep(1L, 20))
ort <- cta_fit(X, y, recursive = TRUE, mc_iter = 100L, mc_seed = 42L,
               loo = "off", min_n = 5L)
cta_ort_node_table(ort)
```

---

cta_plot_data *Extract layout data for plotting a CTA tree*

Description

Returns a pure-data list describing tree topology and layout coordinates. No graphics are produced. Use this as input to `plot.cta_tree` or to custom rendering code.

Layout algorithm: leaves receive sequential integer x-positions in depth-first (left-to-right) order; internal nodes are centred over their children. `y = -depth` so the root sits at the top.

Target-class enrichment: when `target_class` is supplied, each terminal leaf is joined to `cta_staging_table` and annotated with target-class counts, proportions, and a continuous display color derived from the endpoint's rank among all endpoints by ascending target-class proportion. Colors encode relative position within this tree's endpoint distribution and do *not* imply clinical thresholds or categories.

Usage

```
cta_plot_data(tree, target_class = NULL, class_labels = NULL,
              digits = 1, endpoint_palette = NULL)
```

Arguments

`tree`	A `cta_tree` from `oda_cta_fit`.
`target_class`	Integer (or `NULL`); the class label treated as the target when enriching endpoint annotations. `NULL` (default) returns the structural layout only - no endpoint enrichment columns are added.

`class_labels` Optional character vector of display names for class labels. Supply as a *named* vector, e.g. `c("0" = "Alive", "1" = "Deceased")`, or as a positional vector (index $k+1$ maps to class k). `NULL` (default) uses `"class=C"` formatting.

`digits` Integer number of decimal places for percentage formatting in `endpoint_label`. Default `1`.

`endpoint_palette`
Palette for endpoint fill colors, used only when `target_class` is supplied. Accepts `NULL` (default gradient: warm pink to pale yellow to soft green), a palette `function(n)` returning `n` color strings, or a character vector of colors interpolated via `colorRampPalette`.

Value

When `target_class = NULL`: a list with elements `nodes`, `edges`, `no_tree`, `has_weights`.

When `target_class` is supplied: the same list plus `endpoints` (a staging data.frame with layout coordinates) and `target_class_used` (the integer target class used).

`nodes` A `data.frame` with one row per node. Always-present columns: `node_id` (integer), `parent_id` (integer), `depth` (integer), `x` (numeric), `y` (numeric), `leaf` (logical), `attribute` (character; `NA` for leaves), `n_obs` (integer), `majority_class` (integer), `ess` (numeric; `NA` for leaves), `label` (character multi-line display text).

Additional columns present when `target_class` is supplied (values are `NA` on split nodes): `endpoint_id` (integer), `stage` (integer), `target_class` (integer), `target_n` (numeric), `denominator` (numeric), `target_proportion` (numeric; raw continuous proportion, not binned), `target_rank` (integer; ascending rank of proportion, ties broken by `ties.method = "first"`), `endpoint_fill_color` (character hex color assigned by rank within this tree - does not imply clinical thresholds or categories), `predicted_label` (character), `target_label` (character), `endpoint_label` (character multi-line display text).

`edges` A `data.frame` with one row per parent-to-child edge and columns `from_node_id` (integer), `to_node_id` (integer), `x0`, `y0`, `x1`, `y1` (numeric centre-to-centre coordinates), `label` (character branch condition, e.g. `"V14<=0.5"`).

`endpoints` (`target_class` only) A `data.frame` with one row per endpoint, ordered by ascending stage. Columns include all staging fields from `cta_staging_table` plus layout coordinates `x`, `y`, display columns `predicted_label`, `target_label`, `endpoint_fill_color`, and integer `target_rank`.

`target_class_used` (`target_class` only) The integer `target_class` argument used for enrichment.

`no_tree` Logical; `TRUE` for leaf-only fits.

`has_weights` Logical; `TRUE` when case weights are active.

See Also

`plot.cta_tree`, `cta_staging_table`, `oda_cta_fit`

Examples

```
data(mtcars)
X    <- mtcars[, c("cyl", "disp", "hp", "wt")]
y    <- as.integer(mtcars$am)
tree <- suppressMessages(
  oda_cta_fit(X, y, mindenom = 5L, mc_iter = 500L, mc_seed = 42L,
              loo = "off")
)

# Structural layout only
pd <- cta_plot_data(tree)
head(pd$nodes)
pd$edges

# Target-class enrichment
pd2 <- cta_plot_data(tree, target_class = 1L,
                     class_labels = c("0" = "Manual", "1" = "Auto"))
pd2$endpoints[, c("stage", "target_proportion", "endpoint_fill_color")]
```

---

cta_propensity_weights

Endpoint-level propensity-score weights for a fitted CTA tree

---

Description

Returns one row per terminal endpoint per actual class, containing the CTA-derived stabilized propensity-style weights described in Yarnold and Linden (2017). All values are computed on demand from the stored leaf class counts; no refitting, no prediction, and no training-data recomputation is performed.

Formula: For endpoint s and actual class z,

$$w_{s,z} = \frac{n_s \cdot \Pr(Z = z)}{n_{s,z}}$$

where n_s is the endpoint denominator, $n_{s,z}$ is the raw count of class z observations at endpoint s, and $\Pr(Z = z)$ is the marginal class probability across the full classified analytic sample.

Perfect endpoints: When $n_{s,z} = 0$ for some class, the empirical weight is undefined (`Inf`). When `adjusted = TRUE` (default), one hypothetical misclassified observation is added to the absent class profile - and to the global marginal totals - so that all endpoint x class cells yield finite adjusted weights. This is the canon remedy from Yarnold and Linden (2017).

Scope: Raw observation counts (`n_raw`) are used exclusively. The function does not return observation-level weights; those require endpoint membership per training observation, which is not stored on the fitted tree.

Usage

```
cta_propensity_weights(tree, target_class = NULL, adjusted = TRUE)
```

Arguments

tree	A cta_tree from oda_cta_fit.
target_class	Integer (or coercible); annotation column only - does not filter output rows. NULL (default) uses the numerically largest class label for binary trees, and stops for trees with three or more classes.
adjusted	Logical. TRUE (default) applies the one-hypothetical-misclassification adjustment so that all cells yield finite adjusted weights. FALSE leaves undefined weights as Inf and adjusted columns equal to empirical.

Value

A data.frame with one row per terminal endpoint per actual class, with columns:

endpoint_id Integer sequential endpoint index.

endpoint_node_id Integer tree node identifier.

path Character; AND-joined branch labels from root.

terminal_prediction Integer majority-class prediction.

class Character; actual class label for this row.

target_class Integer; design-annotation class label.

class_n Integer; raw count of this class at this endpoint (empirical $n_{s,z}$).

endpoint_n Integer; total raw observations at this endpoint (empirical n_s).

marginal_class_n Integer; total raw observations of this class across all endpoints (empirical N_z).

marginal_total_n Integer; total classified observations across all endpoints (empirical N).

marginal_class_probability Numeric; empirical marginal class probability $\Pr(Z = z) = N_z/N$.

propensity_weight Numeric; empirical stabilized weight $n_s \cdot \Pr(Z = z)/n_{s,z}$. Inf when class_n == 0.

undefined_empirical Logical; TRUE when class_n == 0.

perfectly_predicted_endpoint Logical; TRUE when any class has class_n == 0 at this endpoint.

adjusted Logical; TRUE when the one-hypothetical-observation adjustment was applied to this row.

adjusted_class_n Numeric; class_n + 1 where adjusted, otherwise class_n.

adjusted_endpoint_n Numeric; endpoint denominator after adjustment.

adjusted_marginal_class_n Numeric; global class count after all hypothetical additions.

adjusted_marginal_total_n Numeric; global total after all hypothetical additions.

adjusted_marginal_class_probability Numeric; adjusted marginal class probability.

adjusted_propensity_weight Numeric; adjusted weight. Finite whenever adjusted_class_n > 0.

For a no-tree fit the returned data frame has zero rows but the correct column structure and types.

References

Yarnold PR, Linden A (2017). Computing propensity score weights for CTA models involving perfectly predicted endpoints. *Optimal Data Analysis*, **6**, 43-46.

See Also

oda_cta_fit, cta_endpoint_counts, cta_staging_table

Examples

```
data(mtcars)
X    <- mtcars[, c("cyl", "disp", "hp", "wt")]
y    <- as.integer(mtcars$am)
tree <- oda_cta_fit(X, y, mindenom = 5L, mc_iter = 500L, mc_seed = 42L)
cta_propensity_weights(tree)
```

---

cta_staging_table *Staging table for a fitted CTA tree*

Description

Returns one row per terminal endpoint ordered by ascending target-class propensity (lowest to highest risk stratum). Empirical counts, proportions, and odds are computed from the stored leaf class counts. When an endpoint is perfectly predicted (100 percent one class), the empirical odds and proportion are undefined; the `adjust_perfect` option adds one hypothetical misclassified observation to the undefined profile so all endpoints can be ranked and compared - a canon remedy anchored in Yarnold and Linden (2017).

Scope: The two-class case is handled automatically when `target_class = NULL` (defaults to the numerically larger class label, typically 1). For trees with three or more classes `target_class` must be supplied explicitly.

Usage

```
cta_staging_table(tree, target_class = NULL, weighted = FALSE,
                  adjust_perfect = TRUE)
```

Arguments

`tree`	A `cta_tree` from oda_cta_fit.
`target_class`	Integer (or coercible); the class label treated as the target (positive / high-risk) class. `NULL` (default) uses the numerically largest class label for binary trees, and stops for trees with three or more classes.
`weighted`	Logical. `FALSE` (default) uses raw observation counts; `TRUE` uses case-weighted counts.
`adjust_perfect`	Logical. `TRUE` (default) applies the one-hypothetical-misclassification adjustment to perfectly predicted endpoints so that all endpoints can be ordered by propensity.

Value

A `data.frame` with one row per terminal endpoint, ordered by ascending target-class propensity (lowest to highest risk stratum), with columns:

`stage` Integer rank 1..n, ascending by target proportion.

`endpoint_id` Integer sequential endpoint index, matching `cta_endpoint_summary`.

`endpoint_node_id` Integer tree node identifier.

`path` Character; AND-joined branch labels from root.

`terminal_prediction` Integer majority-class prediction.

`target_class` Integer; the target class used for this table.

`target_n` Numeric; raw (or weighted) count of target-class observations at this endpoint.

`denominator` Numeric; total raw (or weighted) observations at this endpoint.

`target_proportion` Numeric; empirical target-class proportion (`target_n / denominator`).

`non_target_n` Numeric; denominator minus target_n.

`odds` Numeric; empirical odds (`target_n / non_target_n`); `NA` when `perfectly_predicted` is `TRUE`.

`perfectly_predicted` Logical; `TRUE` when the endpoint is 100 percent one class (`target_n == 0` or `non_target_n == 0`).

`adjusted` Logical; `TRUE` when the one-hypothetical-misclassification adjustment has been applied. Always `FALSE` when `adjust_perfect = FALSE`.

`adjusted_target_n` Numeric; target_n after adjustment. Equal to `target_n` when `adjusted` is `FALSE`.

`adjusted_denominator` Numeric; denominator after adjustment.

`adjusted_target_proportion` Numeric; adjusted proportion.

`adjusted_non_target_n` Numeric; adjusted non-target count.

`adjusted_odds` Numeric; adjusted odds.

`weighted` Logical; the value of the `weighted` argument.

`n_obs` Integer; raw observation count at this endpoint (from `cta_endpoint_summary`).

`n_weighted` Numeric; weighted observation count.

For a no-tree fit the returned data frame has zero rows but the correct column structure and types.

References

Yarnold PR, Linden A (2017). Computing propensity score weights for CTA models involving perfectly predicted endpoints. *Optimal Data Analysis*, **6**, 43-46.

See Also

`oda_cta_fit`, `cta_endpoint_summary`, `cta_endpoint_counts`, `cta_propensity_weights`

Examples

```
data(mtcars)
X    <- mtcars[, c("cyl", "disp", "hp", "wt")]
y    <- as.integer(mtcars$am)
tree <- oda_cta_fit(X, y, mindenom = 5L, mc_iter = 500L, mc_seed = 42L)
cta_staging_table(tree)
```

`cta_strata` *Number of terminal leaf endpoints in a CTA tree*

Description

Returns the *count* of terminal leaf nodes in a fitted `cta_tree` (an integer scalar, not a table). Returns `NA_integer_` for no-tree fits (where `tree$no_tree` is `TRUE`).

Usage

```
cta_strata(tree)
```

Arguments

`tree`	A `cta_tree` from `oda_cta_fit`.

Details

To obtain endpoint details (node IDs, path labels, class counts, predicted class), use `cta_endpoint_table`.

Value

Integer scalar: number of terminal leaf nodes, or `NA_integer_` for no-tree fits. This is a count, not a data frame - use `cta_endpoint_table` for per-endpoint rows.

See Also

`cta_endpoint_table`, `cta_endpoint_denominators`, `cta_min_terminal_denom`

lort_fit *Fit a Locally Optimal Recursive Tree (LORT)*

Description

Preferred explicit entry point for the LORT workflow layer. LORT is a non-canonical workflow composition: at each recursive endpoint it runs a full MDSA family scan (`cta_descendant_family`), selects the min-D member, and recurses until no further structure is found or a compute guard fires. It uses canon CTA/MDSA components but is not itself a canon CTA.exe behavior.

Usage

```
lort_fit(
  X,
  y,
  w = NULL,
  mc_iter = 5000L,
  mc_seed = 42L,
  mc_stop = 99.9,
  mc_stopup = NA,
  alpha_split = 0.05,
  prune_alpha = 0.05,
  loo = "stable",
  min_n = 30L,
  max_depth = 8L,
  max_nodes = 31L,
  family_max_steps = 20L,
  verbose = FALSE
)
```

Arguments

`X`	Data frame or matrix of candidate predictor columns.
`y`	Integer class variable vector. Must have exactly two distinct values.
`w`	Optional numeric case-weight vector. Default `NULL` (unit weights).
`mc_iter`	Integer; maximum Monte Carlo iterations per node. Default `5000L`.
`mc_seed`	Integer or `NULL`; RNG seed set once at LORT start. Child-node MDSA scans consume the stream in deterministic right-then-left traversal order without re-setting the seed. Default `42L`.
`mc_stop`	Numeric; confidence bound for lower-tail early MC stopping (percent). Default `99.9`.
`mc_stopup`	Numeric; confidence bound for upper-tail early MC stopping (percent). Default `NA` (disabled; matches MegaODA behavior).
`alpha_split`	Numeric; node-level significance threshold. Default `0.05`.
`prune_alpha`	Numeric; pruning significance threshold. Default `0.05`.

`loo` LOO gate mode per node: `"off"` (no gate), `"stable"` (MegaODA LOO STABLE; accept when |WESSL - WESS| <= 0.01 pp; default), `"pvalue"` (Fisher p strictly less than 0.05), or a single numeric in (0, 1) (Fisher p strictly less than the supplied threshold).

`min_n` Integer; minimum endpoint n to attempt recursion. Endpoints smaller than `min_n` become terminal (stop reason `"min_n"`). Default `30L`.

`max_depth` Integer; safety cap on recursion depth. Nodes at `depth >= max_depth` become terminal (stop reason `"max_depth"`). Default `8L`.

`max_nodes` Integer; safety cap on total ORT nodes. When node count exceeds `max_nodes` the current endpoint becomes terminal (stop reason `"max_nodes"`). Default `31L`.

`family_max_steps` Integer; maximum MDSA family members evaluated at each recursive node. Default `20L`.

`verbose` Logical; emit `[ORT]` progress messages. Default `FALSE`.

Details

`lort_fit()` is functionally equivalent to `cta_fit(..., recursive = TRUE)`. `cta_fit(..., recursive = TRUE)` is retained as a legacy-compatible alias and will continue to work; prefer `lort_fit()` for new code. SORT and GORT are reserved and not implemented.

Value

A dual-tagged `cta_ort` / `cta_tree` object. `cta_ort`-aware methods (`predict.cta_ort`, `print.cta_ort`, `summary.cta_ort`, `plot.cta_ort`, `cta_ort_node_table`) operate on the full composite tree. `ort_settings$method` is always `"lort"`.

See Also

`cta_fit`, `predict.cta_ort`, `cta_ort_node_table`, `ort_plot_data`

Examples

```
X <- data.frame(
  A = c(rep(0L, 20), rep(1L, 20), rep(1L, 20)),
  B = c(rep(0L, 20), rep(0L, 20), rep(1L, 20))
)
y <- c(rep(0L, 20), rep(0L, 20), rep(1L, 20))
fit <- lort_fit(X, y, mc_iter = 100L, mc_seed = 42L, loo = "off", min_n = 5L)
print(fit)
```

lort_index_path *LORT path from root to a given node index*

Description

Returns a data frame tracing the LORT recursion path from the root node (index 1) to the requested node, one row per LORT node on the path.

Usage

```
lort_index_path(x, index)
```

Arguments

x	A cta_ort object from lort_fit.
index	Integer; target LORT node index.

Value

A data frame with columns: lort_index, parent_lort_index, depth, n, stop_reason, is_terminal, incoming_endpoint_id (which endpoint of the parent led here), incoming_path_condition (condition string for that endpoint), incoming_path_label (human-readable label), local_status, local_ess, local_d, local_n_endpoints.

See Also

lort_local_tree, lort_path_table, plot_lort_path

lort_local_tree *Extract the local CTA model embedded at a LORT node*

Description

Returns the full cta_tree object selected at LORT node index. This is the complete CTA/MDSA family member fitted on the observations that reached that node – not a summary, not a stump approximation, not reconstructed from plot-data.

Usage

```
lort_local_tree(x, index)
```

Arguments

x	A cta_ort object from lort_fit.
index	Integer; LORT node index.

Details

Returns NULL when the node is terminal due to min_n, max_depth, max_nodes, or a pure-class guard (no fit was attempted). A no_tree result at a non-forced-terminal node yields a cta_tree with $no_tree = TRUE.

Value

A cta_tree object or NULL (with a message for forced-terminal nodes).

See Also

lort_index_path, plot_lort_path

---

lort_path_table *Formatted path table for a LORT recursion path*

Description

Prints and returns (invisibly) a summary of the LORT recursion path from the root node to the requested index. For each node on the path it shows the local CTA model's key metrics and the endpoint condition that led to the next recursive call.

Usage

```
lort_path_table(x, index)
```

Arguments

x	A cta_ort object from lort_fit.
index	Integer; target LORT node index.

Value

Invisibly, the data frame from lort_index_path. Printed output goes to stdout().

See Also

lort_index_path, lort_local_tree, plot_lort_path

lort_propensity_weights *LORT terminal strata propensity weights*

Description

Computes propensity weights from the terminal strata of a fitted LORT (Locally Optimal Recursive Tree) model. Uses stored `class_counts` per terminal node. Implements the Yarnold/Linden stratum-weight formula (same as `cta_propensity_weights`):

$$w = n_s \times \Pr(Z = z)/n_{z,s}$$

Usage

```
lort_propensity_weights(ort, target_class = NULL, adjusted = TRUE)
```

Arguments

`ort`	A `cta_ort` object produced by `lort_fit` or `cta_fit(recursive = TRUE)`.
`target_class`	Integer target class for annotation (optional; if `NULL`, defaults to the numerically higher class in binary models).
`adjusted`	Logical; if `TRUE` (default), applies a one-hypothetical-misclassification adjustment when a class is absent from a terminal stratum.

Details

The fitted model must have been trained with the treatment/exposure/group membership as the class variable, not a clinical outcome. The user is responsible for this labeling decision.

Value

Data frame with one row per (stratum, class) combination. Columns: `stratum_id` (integer), `path` (character), `depth` (integer), `stratum_n` (integer), `terminal_class` (integer), `class` (character), `class_n` (integer), `target_class` (integer), `marginal_class_n` (integer), `marginal_total_n` (integer), `marginal_class_probability` (numeric), `propensity_weight` (numeric), `undefined_empirical` (logical), `adjusted` (logical), `adjusted_propensity_weight` (numeric), `model_family` (`"lort"`), `global_optimization` (`FALSE`), `sda_anchored` (`FALSE`).

See Also

`cta_propensity_weights`, `oda_propensity_weights`, `lort_fit`

`myeloma` *Myeloma gene-expression dataset (CTA benchmark)*

Description

A data frame with 256 observations and 19 variables, formatted for use with `cta_fit` and `oda_fit`. Derived from the publicly available myeloma gene-expression dataset (GEO accession GSE4581), as distributed in the **survminer** package.

Format

A data frame with 256 rows and 19 columns:

V1 Survival event indicator (0 = censored, 1 = event). Used as the class variable `y` in CTA/ODA.

V2 Case weight (observation time in months). Use as `w` in `cta_fit`; rows with V2 == 0 should be excluded.

V3 CCND1 gene expression.

V4 CRIM1 gene expression.

V5 DEPDC1 gene expression.

V6 IRF4 gene expression.

V7 TP53 expression / mutation burden.

V8 WHSC1 gene expression.

V9 Molecular group: Cyclin D-1 (binary).

V10 Molecular group: Cyclin D-2 (binary).

V11 Molecular group: Hyperdiploid (binary).

V12 Molecular group: Low bone disease (binary).

V13 Molecular group: MAF (binary).

V14 Molecular group: MMSET (binary).

V15 Molecular group: Proliferation (binary).

V16 Chr1q21 status: 2 copies (binary).

V17 Chr1q21 status: 3 copies (binary).

V18 Chr1q21 status: 4+ copies (binary).

V19 Chr1q21 status: NA-coded (binary). Missing values are coded as -9 (`miss_codes = -9`).

Details

This dataset is used throughout the oda documentation and vignettes to illustrate weighted CTA, MINDENOM constraints, LOO STABLE validation, and missing-code handling. Reference CTA.exe golden outputs for MINDENOM = 1, 30, and 56 are used as regression anchors.

Use `miss_codes = -9` and `w = myeloma$V2` when calling `cta_fit`. With `mindenom = 1`, the enumerated CTA tree roots at V14 with a V15 child (OVERALL ESS = 26.32%, WEIGHTED ESS = 27.69%). With `mindenom = 30`, the selected tree is a V17 stump (WEIGHTED ESS = 16.51%). With `mindenom = 56`, no admissible tree exists.

Source

Derived from the `myeloma` dataset in the **survminer** package. Original data: NCBI GEO accession GSE4581. No PHI; no institutional data. See `tests/testthat/fixtures/myeloma/README.md` in the source tree.

`novo_boot_ci` *Novometric bootstrap CI from a fixed 2x2 confusion matrix*

Description

Estimates the precision of an observed binary classification effect by comparing model and chance distributions via permutation/resampling bootstrap. Based on the NOVOboot methodology (Yarnold 2020; Yarnold & Soltysik 2016).

Fixed-confusion bootstrap: This function samples from the observed confusion matrix structure. It does *not* refit ODA or CTA models and does *not* estimate model-selection variability. The model distribution is generated by resampling paired (actual, predicted) rows from the expanded confusion table; the chance distribution is generated by independently resampling actual and predicted labels, breaking their association. Novometric significance (Axiom 1) is declared when the 95% confidence intervals for model and chance ESS do not overlap.

Usage

```
novo_boot_ci(x, ...)

## Default S3 method:
novo_boot_ci(x,
             nboot       = 5000L,
             seed        = NULL,
             sample_frac = 0.5,
             probs       = c(0, .025, .05, .25, .5, .75, .95, .975, 1),
             alternative = c("two.sided", "greater", "less"),
             ...)

## S3 method for class 'oda_fit'
novo_boot_ci(x,
             nboot       = 5000L,
             seed        = NULL,
             sample_frac = 0.5,
             probs       = c(0, .025, .05, .25, .5, .75, .95, .975, 1),
             alternative = c("two.sided", "greater", "less"),
             ...)

## S3 method for class 'cta_tree'
novo_boot_ci(x,
             nboot       = 5000L,
             seed        = NULL,
```

```
              sample_frac = 0.5,
              probs       = c(0, .025, .05, .25, .5, .75, .95, .975, 1),
              alternative = c("two.sided", "greater", "less"),
              node_id     = NULL,
              weighted    = FALSE,
              ...)

## S3 method for class 'cta_ort'
novo_boot_ci(x,
              nboot       = 5000L,
              seed        = NULL,
              sample_frac = 0.5,
              probs       = c(0, .025, .05, .25, .5, .75, .95, .975, 1),
              alternative = c("two.sided", "greater", "less"),
              stratum_id  = NULL,
              weighted    = FALSE,
              ...)

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

Arguments

x	For the default method: a 2x2 integer matrix, rows = actual class, columns = predicted class. Same [actual, predicted] convention as training_confusion in a cta_tree and as oda_confusion(). Use byrow = TRUE when constructing with matrix(). For S3 methods: a fitted model object (oda_fit, cta_tree, or cta_ort) from which the training confusion matrix is extracted.
nboot	Number of bootstrap replicates. Default 5000.
seed	Integer seed passed to set.seed, or NULL to use the current RNG state. Use a fixed seed for reproducibility.
sample_frac	Fraction of n sampled per replicate (with replacement). Default 0.5, matching NOVOboot.
probs	Quantile probability levels for the summary table.
alternative	Direction for exact Fisher p-values: "two.sided" (default), "greater", or "less".
node_id	Integer node id of a *terminal* (leaf) node in a cta_tree. When supplied, the bootstrap uses the class counts for that specific terminal node rather than the full-tree confusion. Only valid for novo_boot_ci.cta_tree.
stratum_id	Integer stratum id from cta_ort$strata. When supplied, the bootstrap uses the class counts for that single terminal LORT stratum rather than the full-LORT confusion. Only valid for novo_boot_ci.cta_ort.
weighted	Logical. When node_id or stratum_id is supplied, weighted = TRUE uses case-weighted class counts and weighted = FALSE (default) uses raw integer counts. Ignored for full-tree paths.
...	For the generic and S3 fit methods: additional arguments passed to novo_boot_ci.default. For print.novo_boot_ci: currently ignored.

Details

Model distribution: The input confusion matrix is expanded to n paired (actual, predicted) observation rows. For each replicate, k row indices are drawn with replacement, preserving the observed (actual, predicted) joint distribution. This mirrors the NOVOboot row-resampling approach.

Chance distribution: Actual and predicted labels are resampled independently for each replicate, breaking any association between them. This generates the null distribution against which the model effect is compared.

p-values: An exact 2x2 Fisher p-value is computed for every replicate confusion matrix for both model and chance distributions. These form precision distributions and complement the CI non-overlap criterion; they are not a substitute for it.

Novometric Axiom 1: A statistically significant effect exists when the exact discrete confidence intervals for model and chance performance do not overlap. `significant = TRUE` indicates the ESS model 95% CI lies entirely above the ESS chance 95% CI.

ESS formula: `ESS(%) = 100 * (mean_PAC - 0.5) / 0.5`, consistent with `oda_ess_from_meanpac`.

OR: Diagnostic odds ratio `(TP * TN) / (FP * FN)`. `NA` when `FP = 0` or `FN = 0` in a replicate.

RR: Positive predictive value / false omission rate `[TP / (TP+FP)] / [FN / (FN+TN)]`. `NA` when undefined.

Value

An object of class `novo_boot_ci`, a list with:

`call` The matched call.

`confusion` Input confusion matrix (integer, 2x2).

`n` Total observations (`sum(x)`).

`k` Observations sampled per replicate (`round(sample_frac * n)`).

`nboot`, `sample_frac`, `probs`, `alternative` Input parameters.

`has_zero_cells` Logical; `TRUE` if any cell of `x` is zero. Does not stop computation; NA propagates for affected metrics in affected replicates.

`observed` Data frame with one row per metric. Columns: `metric`, `value`. Reports the observed (not bootstrapped) sensitivity, specificity, mean_pac, ess, odds_ratio, and risk_ratio computed directly from the input confusion matrix.

`model` Data frame (`nboot` rows). Per-replicate model bootstrap distributions: `sensitivity`, `specificity`, `mean_pac`, `ess` (all in %), `odds_ratio`, `risk_ratio`, `p_value`. `NA` for undefined OR/RR.

`chance` Data frame (`nboot` rows). Same columns as `model`. Generated by independently resampling actual and predicted labels (null of no classification association).

`quantiles` Data frame (`length(probs)` rows). Quantiles of each metric for model and chance across all replicates, including `p_value_model` and `p_value_chance`.

`ci` Data frame (one row per metric). Fixed 95% CI bounds (2.5th and 97.5th percentiles) for model and chance. Columns: `metric`, `model_lower`, `model_upper`, `chance_lower`, `chance_upper`, `overlap`.

`significant` Logical scalar. `TRUE` if the ESS model 95% CI lower bound exceeds the ESS chance 95% CI upper bound - novometric Axiom 1 CI non-overlap criterion.

`source_type` Character. Evidence provenance tag: `"matrix"`, `"oda_fit"`, `"cta_tree"`, `"cta_tree_node"`, `"cta_ort"`, or `"cta_ort_stratum"`.

`source_id` Integer or `NA`. Node or stratum id when evidence came from a specific sub-unit; `NA` for full-tree paths.

`weighted` Logical or `NA`. `TRUE` when weighted class counts were used; `FALSE` for raw counts; `NA` for the default matrix path.

References

Yarnold PR (2020). Reformulating the First Axiom of Novometric Theory: Assessing Minimum Sample Size in Experimental Design. *Optimal Data Analysis* **9**, 7–8.

Yarnold PR, Soltysik RC (2016). *Maximizing Predictive Accuracy*. ODA Books.

Examples

```
# Myeloma MINDENOM=1 confusion (actual x predicted, byrow = TRUE)
conf <- matrix(c(146, 40,
                 36, 33), nrow = 2, byrow = TRUE)
ci <- novo_boot_ci(conf, nboot = 200L, seed = 42L)
ci$significant
print(ci)
```

---

oda_balance_effect_table

ODA covariate balance evidence-interval table

Description

Builds one row per covariate × analysis scale containing the observed ESS/WESS, a bootstrap confidence interval (model sampling variability), and a chance interval (null distribution from group-label permutation). The resulting table answers whether each covariate's model confidence interval clears the chance interval.

Usage

```
oda_balance_effect_table(
  group,
  X,
  w = NULL,
  compare_weights = FALSE,
  covariate_types = NULL,
  nboot = 2000L,
  chance_iter = 2000L,
  ci = 0.95,
  mc_seed = NULL,
  mc_iter = 1000L,
  ...
)
```

Arguments

group	Integer (or coercible) binary group indicator with exactly two distinct non-missing values. Plays y in ODA.
X	Data frame of baseline covariate columns (nrow(X) == length(group)).
w	Optional numeric case-weight vector. When supplied, weighted ODA is used and metric = "WESS".
compare_weights	Logical; when TRUE and w is supplied, produces two rows per covariate: analysis = "unweighted" (w ignored) and analysis = "weighted" (w used). Default FALSE.
covariate_types	Optional named character vector mapping column names to ODA attribute types. Unmapped columns use "auto".
nboot	Integer; number of bootstrap resamples. Default 2000L.
chance_iter	Integer; number of group-label permutations for the null interval. Default 2000L.
ci	Numeric; nominal coverage for both intervals. Default 0.95.
mc_seed	Integer RNG seed set once at function entry. Controls all bootstrap and permutation sampling deterministically. NULL for unseeded.
mc_iter	Integer; MC iterations passed to the observed oda_fit call. Default 1000L.
...	Additional arguments forwarded to each oda_fit call (e.g., priors_on, loo). mcarlo and mc_iter are controlled internally and must not be passed here.

Details

Three passes are run per covariate:

1. **Observed:** oda_fit(mcarlo = TRUE) – point estimate and Monte Carlo p-value.
2. **Bootstrap:** nboot resamples (rows with replacement), mcarlo = FALSE – percentile confidence interval.
3. **Chance:** chance_iter group-label permutations, mcarlo = FALSE – null percentile interval.

When compare_weights = TRUE and w is supplied, both an "unweighted" and a "weighted" row are produced per covariate. Multiplicity corrections (Sidak, Bonferroni) are applied within each analysis scale across covariates.

Interpretation:

- balanced_by_interval = TRUE: model bootstrap CI overlaps the chance interval (boot_lo <= chance_hi) – no evidence of residual imbalance for this covariate.
- residual_imbalance = TRUE: model CI clears chance (boot_lo > chance_hi) – residual imbalance detected.

Value

A list of class "oda_balance_effect_table" with:

rows Data frame; one row per covariate × analysis scale. Columns: attribute, analysis, metric, estimate, boot_lo, boot_hi, chance_lo, chance_hi, p_mc, p_sidak, p_bonferroni, rule_summary, sensitivity, specificity, n_total, balanced_by_interval, residual_imbalance.

meta List of metadata: n_covariates, n_obs, has_weights, compare_weights, analyses, nboot, chance_iter, ci, mc_iter, mc_seed.

References

Linden A, Yarnold PR (2016). Using machine learning to assess covariate balance in matching studies. *Journal of Evaluation in Clinical Practice*, **22**(6), 861-867.

See Also

oda_balance_table, plot_oda_balance_effects

Examples

```
set.seed(1)
group <- c(rep(0L, 30), rep(1L, 30))
X <- data.frame(
  age   = c(rnorm(30, 45, 8), rnorm(30, 55, 8)),
  score = rnorm(60, 50, 10)
)
et <- oda_balance_effect_table(group, X,
                               nboot = 50L, chance_iter = 50L,
                               mc_iter = 200L, mc_seed = 1L)
et$rows[, c("attribute", "estimate", "boot_lo", "boot_hi",
            "chance_lo", "chance_hi", "balanced_by_interval")]
```

---

oda_balance_plot_data *Renderer-ready plot data for univariate ODA covariate balance*

Description

Transforms an oda_balance_table result (and optionally an smd_balance_table result) into a renderer-independent data structure suitable for Graphics v3 plotting.

Usage

```
oda_balance_plot_data(
  balance_table,
  smd_table = NULL,
  p_col = c("p_mc", "p_sidak", "p_bonferroni"),
  rank_by = c("abs_ess", "p", "abs_smd")
)
```

Arguments

balance_table	An "oda_balance_table" object from oda_balance_table.
smd_table	Optional "smd_balance_table" from smd_balance_table. When supplied, SMD columns are joined by attribute name and included in rows.
p_col	Character; which p-value column to use for the p_plot and significant columns in the output rows. One of "p_mc" (default), "p_sidak", "p_bonferroni".
rank_by	Character; how to rank covariates for display order. "abs_ess" (default): descending ESS/WESS (most imbalanced first). "p": ascending p (most significant first). "abs_smd": descending absolute SMD (requires smd_table).

Details

This function does not fit any ODA models and does not accept group **or** X **arguments.** It is a pure transformation of pre-computed balance tables.

Value

A list of class "oda_balance_plot_data" with elements:

rows Data frame; one row per covariate, sorted by rank_by. Columns: attribute, attr_type, ess_display, ess_display_bar (clipped to [0, 100]), p_plot (selected p column), significant, significance_label ("*" or ""), rule_summary, abs_smd, wsmd_available, abs_smd_display (weighted if active), fit_ok, rank.

has_weights Logical.

ess_label Character; "WESS" or "ESS".

p_col_used Character; selected p column name.

alpha Numeric; significance threshold from metadata.

n_covariates Integer.

n_significant Integer; covariates significant on p_col_used.

rank_by Character.

See Also

oda_balance_table, smd_balance_table

Examples

```
set.seed(1)
group <- c(rep(0L, 30), rep(1L, 30))
X     <- data.frame(age   = c(rnorm(30, 45, 8), rnorm(30, 55, 8)),
                    score = rnorm(60, 50, 10))
bt  <- oda_balance_table(group, X, mcarlo = TRUE, mc_iter = 500L)
smd <- smd_balance_table(group, X)
pd  <- oda_balance_plot_data(bt, smd_table = smd)
pd$rows[, c("attribute", "ess_display", "p_plot", "significant", "abs_smd")]
```

`oda_balance_table` *Univariate ODA covariate balance diagnostics*

Description

Fits a univariate ODA model for each covariate in `X` with `group` as the class variable. Returns one row per covariate summarizing ODA-based balance diagnostics: rule, sensitivity, specificity, Mean PAC, ESS/WESS, and permutation p-value with Sidak and Bonferroni multiplicity corrections.

Usage

```
oda_balance_table(
  group,
  X,
  w = NULL,
  covariate_types = NULL,
  loo = "off",
  mcarlo = TRUE,
  mc_iter = 1000L,
  alpha = 0.05,
  adjust = c("none", "sidak", "bonferroni"),
  ...
)
```

Arguments

`group`	Integer (or coercible) binary group indicator. Must have exactly two distinct non-missing values. Plays the role of the class variable (`y`) in ODA.
`X`	Data frame of baseline covariate columns. Plays the role of attributes (`x`) in ODA. Must have `nrow(X) == length(group)`.
`w`	Optional numeric case-weight vector (length `length(group)`). When supplied, ODA fits use case weights and `ess_label = "WESS"`.
`covariate_types`	Optional named character vector mapping column names to ODA attribute types (`"auto"`, `"ordered"`, `"categorical"`, `"binary"`). Unmapped columns use `"auto"`.
`loo`	LOO gate mode passed to each `oda_fit` call. Default `"off"`. See `oda_fit` for options.
`mcarlo`	Logical; run Monte Carlo permutation p-value? Default `TRUE`. Set `FALSE` to skip p-value computation.
`mc_iter`	Integer; maximum MC iterations per covariate. Default `1000L`. Higher values (e.g., `25000L`) are recommended for final published analyses but increase runtime.
`alpha`	Numeric significance threshold for the `significant` flag. Default `0.05`.

adjust	Character; which p-value drives the primary significant column: "none" (raw p, default), "sidak", or "bonferroni". All three significance flags are always returned regardless of this choice.
...	Additional arguments forwarded to each oda_fit call (e.g., mc_seed, priors_on, mc_stop).

Details

Balance asks whether group membership (treatment, exposure, or study arm) can be predicted from observed baseline covariates. When no covariate predicts group membership above chance, the groups are considered balanced on those covariates under the declared analytic constraints.

group vs. outcome: group is the binary class variable in every ODA call. The scientific outcome of interest is strictly out of scope; do not pass the outcome as group or as a column of X.

SMD: conventional standardized mean difference is a companion diagnostic, not the oda balance objective. Use smd_balance_table for the conventional companion table.

Value

A list of class "oda_balance_table" with elements:

rows Data frame; one row per covariate. Key columns: attribute, attr_type, n_total, n_group_0, n_group_1, sensitivity, specificity, mean_pac, ess, wess, ess_display (operative measure), p_mc, p_sidak, p_bonferroni, significant_raw, significant_sidak, significant_bonferroni, significant (driven by adjust), rule_type, rule_summary, loo_status, ess_loo, has_weights, fit_ok, fit_reason.

meta List of metadata: n_covariates, n_obs, has_weights, ess_label, alpha, adjust, k_valid (number of covariates with valid p_mc used for multiplicity correction), loo_mode, mcarlo, mc_iter.

References

Linden A, Yarnold PR (2016). Using machine learning to assess covariate balance in matching studies. *Journal of Evaluation in Clinical Practice*, **22**(6), 861-867.

See Also

smd_balance_table, oda_balance_plot_data, oda_fit

Examples

```
set.seed(1)
n <- 60
group <- c(rep(0L, 30), rep(1L, 30))
X <- data.frame(
  age    = c(rnorm(30, 45, 8), rnorm(30, 55, 8)),   # imbalanced
  score  = rnorm(60, 50, 10)                         # balanced
)
bt <- oda_balance_table(group, X, mcarlo = TRUE, mc_iter = 500L)
bt$rows[, c("attribute", "ess_display", "p_mc", "significant_raw")]
```

`oda_best_ordered_multiclass_partition` *Select the best K-segment ordered partition by MegaODA spec: PRIMARY -> SECONDARY -> FIRST IDENTIFIED (enum order via tick()).*

Description

Select the best K-segment ordered partition by MegaODA spec: PRIMARY -> SECONDARY -> FIRST IDENTIFIED (enum order via tick()).

Usage

```
oda_best_ordered_multiclass_partition(
  x_rep,
  counts_obj,
  counts_raw,
  K,
  priors_on_eff = TRUE,
  degen = FALSE,
  primary = NULL,
  secondary = NULL,
  cut_value_mode = c("midpoint", "lower", "upper"),
  debug_return_ties = FALSE,
  debug_max_ties = 200L,
  direction = "off"
)
```

Arguments

`x_rep`	Representative x value per unique block.
`counts_obj`	m x C count matrix in objective (priors-weighted) space.
`counts_raw`	m x C count matrix in raw (case-weight) space.
`K`	Number of segments (cuts = K-1).
`priors_on_eff`	Logical.
`degen`	Allow degenerate solutions?
`primary, secondary`	Heuristic strings (NULL = spec defaults).
`cut_value_mode`	"midpoint", "lower", or "upper".
`debug_return_ties`	Return all primary-tied candidates for diagnostics.
`debug_max_ties`	Cap on number of ties stored.
`direction`	Directional constraint (MPE Chapter 4 ordered DIRECTIONAL). "ascending" forces segment s to map to class s; "descending" forces class C+1-s. Default "off" (nondirectional; all assignments evaluated).

Value

List with ok, cuts_idx, cut_values, seg_cls_idx, primary_obj, secondary_obj, best_enum_id, ties, classes.

---

oda_clean_missing_codes

Replace missing-code values with NA

Description

Replaces all values in `miss_codes` with `replacement` (default `NA`). Accepts a numeric vector or a data frame. Does not modify the class variable or weight vector — pass those separately if needed.

Usage

```
oda_clean_missing_codes(X, miss_codes, replacement = NA)
```

Arguments

`X`	Numeric vector or data frame of predictors.
`miss_codes`	Numeric vector of values to treat as missing (e.g. `-9`).
`replacement`	Replacement value (default `NA`).

Value

Object of the same class and dimensions as `X` with `miss_codes` values replaced.

---

oda_confusion

Retrieve a confusion matrix from a fitted ODA model

Description

Retrieve a confusion matrix from a fitted ODA model

Usage

```
oda_confusion(fit, split = c("train", "loo"), weighted = FALSE)
```

Arguments

`fit`	An `oda_fit` object.
`split`	One of `"train"` or `"loo"`.
`weighted`	Logical; if `TRUE` return the priors-weighted confusion. Weighted LOO confusion is not stored separately.

Value

The confusion object stored on the fit, or NULL.

---

oda_confusion_binary *Binary confusion table*

Description

Compute a weighted binary confusion table from actual and predicted labels.

Usage

```
oda_confusion_binary(y, y_pred, w = NULL)
```

Arguments

y	Actual class labels (0/1 integer).
y_pred	Predicted class labels (0/1 integer).
w	Optional numeric weights. Default: unit weights.

Value

Named list with integer count fields TP, TN, FP, FN (weighted sums), and rate fields sensitivity, specificity (proportions in [0, 1]), and mean_pac (proportion in [0, 1]).

Note: With unit weights these are raw integer counts. With prior-odds weights (from oda_univariate_core with priors_on = TRUE) they are weighted counts, not raw integers.

---

oda_confusion_multiclass *Multiclass confusion matrix*

Description

Compute a weighted multiclass confusion matrix with PAC and PV summaries.

Usage

```
oda_confusion_multiclass(y, y_pred, w = NULL)
```

Arguments

y	Actual integer class labels.
y_pred	Predicted integer class labels.
w	Optional numeric weights. Default: unit weights.

Value

Named list:

`confusion` C x C numeric matrix of weighted counts. With unit weights these are raw integer observation counts. Rows are actual classes; columns are predicted classes.

`correct` Total weighted count of correct classifications.

`overall_acc` Overall accuracy as a proportion [0, 1].

`pac_by_class` Per-class sensitivity as proportions [0, 1].

`mean_pac` Mean sensitivity across classes, proportion [0,1].

`pv_by_class` Per-class predictive value, proportions [0, 1].

`mean_pv` Mean predictive value, proportion [0, 1].

`oda_cta_fit` *Fit a Classification Tree Analysis (CTA) model (internal engine)*

Description

Internal CTA engine name retained for backward compatibility. **Users should prefer `cta_fit()` as the public entry point.**

Builds a classification tree by recursively applying ODA at each node. At each split, all attributes are evaluated and the attribute with the highest ESS passing the significance threshold is selected. Matches MegaODA CTA behaviour including MINDENOM, PRUNE, ENUMERATE, LOO STABLE, and WEIGHT parameters.

Usage

```
oda_cta_fit(X, y, w = NULL, priors_on = TRUE, miss_codes = NULL,
  alpha_split = 0.05, mindenom = 5L, prune_alpha = 1.0,
  max_depth = 10L, ess_min = 0,
  mc_iter = 25000L, mc_target = 0.05, mc_stop = 99.9, mc_stopup = NULL,
  mc_seed = NULL, loo = "off", attr_names = NULL, K_segments = NULL,
  verbose = FALSE, diag_env = NULL)
```

Arguments

`X`	Data frame or matrix of attribute columns.
`y`	Class variable vector.
`w`	Optional numeric case weights (MegaODA WEIGHT). Same length as y.
`priors_on`	Use prior-odds weighting at each node. Default TRUE.
`miss_codes`	Numeric vector of missing-value codes (MegaODA MISSING).
`alpha_split`	Significance threshold to split a node (MegaODA MC CUTOFF). Default 0.05.
`mindenom`	Minimum weighted node size to attempt a split (MegaODA MINDENOM). Default 5.

prune_alpha	Branches with p >= prune_alpha are not grown (MegaODA PRUNE). Default 1.0 = no pruning (unpruned tree).
max_depth	Maximum tree depth. Default 10.
ess_min	Minimum ESS required to split. Default 0.
mc_iter	Maximum MC iterations per node. Default 25000.
mc_target, mc_stop, mc_stopup	MC stopping parameters.
mc_seed	Base RNG seed; each node uses mc_seed + node_id * 1000 + attr_j.
loo	LOO mode per node: "off" (default), "stable" (MegaODA LOO STABLE; accept when |WESSL - WESS| <= 0.01 pp; reports loo_status = "STABLE"), "pvalue" (Fisher p *strictly less than* 0.05; reports loo_status = "PVALUE"), or a single numeric in (0, 1) (Fisher p *strictly less than* the supplied threshold; reports loo_status = "PVALUE").
attr_names	Attribute names. Defaults to column names of X.
K_segments	Segments for multiclass ordered splits. Default = C.
verbose	Logical; if TRUE, emit [CTA] progress messages via message() at each major stage. Default FALSE.
diag_env	Internal diagnostic environment used to collect CTA timing and Monte Carlo instrumentation. Intended for development diagnostics only; leave as NULL for normal use.

Value

An object of class cta_tree containing:

nodes Named list of node objects, each with fields: node_id, parent_id, depth, n_obs, n_weighted, attribute, rule, ess, p_mc, loo_status, loo_ess, confusion, child_ids, split_labels, majority_class, leaf.

root_id Integer ID of the root node.

n_nodes Total number of nodes grown.

Use predict.cta_tree to classify new data and cta_node_table to extract the node summary table.

See Also

predict.cta_tree, cta_node_table

Examples

```
## Binary CTA on mtcars
data(mtcars)
mt <- mtcars
X  <- mt[, c("cyl","disp","hp","wt")]
y  <- as.integer(mt$am)
tree <- oda_cta_fit(X, y, alpha_split = 0.05, mindenom = 5L,
                    mc_iter = 500L, mc_seed = 42L)
```

```
print(tree)
preds <- predict(tree, X)
mean(preds == y)   # training accuracy
```

`oda_d_stat` *Compute the D statistic for a fitted ODA model*

Description

D measures the distance between a model's classification accuracy (ESS) and chance, expressed relative to the number of terminal prediction strata. Formula: $D = \frac{100}{ESS/strata} - strata$, where *strata* counts terminal prediction endpoints only.

Usage

```
oda_d_stat(fit)
```

Arguments

`fit`	An `oda_fit` object from `oda_fit`.

Details

Supported rule types and strata definitions:

- Binary (`oda_fit_binary`): strata = 2, ESS = `fit$ess`.
- Multiclass ordered (`multiclass_ordered` rule): strata = `length(fit$rule$seg_classes)`, ESS = `fit$ess`.
- Multiclass nominal/categorical: returns `NA_real_` (strata count is ambiguous without additional canon specification).
- Failed fit (`ok = FALSE`): returns `NA_real_`.

Value

A scalar `numeric` D value, or `NA_real_` when the fit failed or the rule type does not have an unambiguous strata count.

oda_ess_from_mean *ESS from mean metric for a C-class problem*

Description

Compute Effect Strength for Sensitivity from mean PAC or mean PV for a problem with C classes.

Usage

```
oda_ess_from_mean(mean_metric, C)
```

Arguments

mean_metric	Mean PAC or mean PV as a proportion [0, 1].
C	Number of classes.

Value

ESS as a percentage [0, 100]. Chance baseline is 1/C.

oda_ess_from_meanpac *Effect Strength for Sensitivity from mean PAC*

Description

Compute ESS (Effect Strength for Sensitivity) in percent, scaled against a chance baseline.

Usage

```
oda_ess_from_meanpac(mean_pac, chance)
```

Arguments

mean_pac	Mean PAC as a proportion [0, 1].
chance	Chance baseline as a proportion (e.g. 0.5 for 2-class).

Value

ESS as a percentage [0, 100].

oda_fit *Fit an ODA model*

Description

Unified entry point for Optimal Data Analysis. Dispatches to the binary-class engine when the outcome has exactly two distinct values, or the multiclass engine for three or more classes. This is the function CTA nodes call at each split candidate.

Usage

```
oda_fit(x, y, w = NULL,
        attr_type = c("auto","ordered","categorical","binary"),
        priors_on = TRUE, K_segments = NULL, degen = FALSE,
        miss_codes = NULL, missing_code = NULL,
        mcarlo = TRUE, mc_iter = 25000L, mc_target = 0.05,
        mc_stop = 99.9, mc_stopup = NA, mc_seed = NULL,
        loo = "off",
        boundary_mode = c("megaoda_halfopen","right_closed"),
        eval_order = c("mc_then_loo","loo_then_mc"),
        mindenom = 1L,
        direction = c("both","off","greater","less","ascending","descending"),
        direction_map = NULL)
```

Arguments

x	Attribute values (numeric, factor, character, or logical).
y	Class labels; must have 2 or 3+ distinct values.
w	Optional numeric case weights. Default: unit weights. These are economic or importance weights, distinct from prior-odds weighting which is controlled by priors_on.
attr_type	Attribute measurement type: "auto" (default), "ordered", "categorical", or "binary".
priors_on	Logical; if TRUE (default), weight classes by the reciprocal of their sample frequency so that the objective maximizes mean PAC rather than overall PAC.
K_segments	Number of segments for multiclass ordered models. Default equals the number of classes C.
degen	Logical; if FALSE (default) require all C classes to appear in the predicted labels.
miss_codes	Numeric vector of values to treat as missing (excluded from analysis).
missing_code	Scalar alias for miss_codes; accepted for convenience.
mcarlo	Logical; run Monte Carlo Fisher-randomization p-value? Default TRUE.
mc_iter	Maximum Monte Carlo iterations. Default 25000.
mc_target	Significance threshold for STOP early stopping. Default 0.05.

`mc_stop` Confidence level (percent) for lower-tail STOP. Default 99.9.

`mc_stopup` Confidence level (percent) for upper-tail STOPUP. Default NA (disabled; matches MegaODA behavior).

`mc_seed` Optional integer RNG seed for reproducibility.

`loo` LOO mode. `"off"` (default): no LOO filter. `"on"`: synonym for `"pvalue"` when used with the multiclass engine. `"stable"`: binary only; accept when LOO ESS equals training ESS (|WESSL - WESS| <= 0.01 pp); split node reports `loo_status = "STABLE"`. `"pvalue"`: binary only; accept when LOO Fisher p is *strictly less than* 0.05 (default threshold); split node reports `loo_status = "PVALUE"`. Numeric in (0, 1): binary only; accept when LOO Fisher p is *strictly less than* the supplied value; must be a single finite value strictly in (0, 1); split node reports `loo_status = "PVALUE"`. Do not describe the p-value gate as "STABLE" - the two modes are distinct.

`boundary_mode` Boundary convention for multiclass ordered rules. Default `"megaoda_halfopen"` matches MegaODA.exe behaviour.

`eval_order` Controls whether Monte Carlo testing is run before LOO validation or whether eligible ordered-cut LOO stability is checked before Monte Carlo. The default `"mc_then_loo"` preserves standalone UniODA behaviour. CTA tree building uses `"loo_then_mc"` internally to reject LOO-unstable ordered-cut candidates before spending MC iterations.

`mindenom` Minimum raw observation count required in each child node for a candidate cut to be evaluated. Default 1 (no enforcement).

`direction` Directional hypothesis control. `"both"` (default, non-directional) or its synonym `"off"`. `"greater"` / `"less"`: MPE Chapter 2 binary ordered DIRECTIONAL (high / low attribute values predict class 1; binary class only - error on multiclass). `"ascending"` / `"descending"`: MPE Chapter 4 DIRECTIONAL for multiclass ordered (segment s maps to class s or C+1-s) or multiclass categorical with L == C (auto-creates identity or reverse mapping). Error on binary class (use `"greater"` / `"less"` for binary ordered).

`direction_map` Named integer vector for categorical fixed-partition DIRECTIONAL (MPE Chapter 4). Names are attribute levels (character); values are predicted class labels. All attribute levels must be covered exactly once with at least two distinct target classes. When supplied, ODA evaluates only the specified mapping and skips the partition search. For binary class, values should be the original class labels (recoded to 0/1 internally). For multiclass, values should be 1..C class labels. Compatible with `direction = "both"` (default). Default `NULL`.

Value

A named list with components:

`ok` Logical; `TRUE` if a valid model was found.

`reason` Character reason string if `ok = FALSE`.

`rule` The fitted rule (list; structure depends on `attr_type` and engine).

`n_eff` Number of observations used (after missing removal).

`ess` Effect Strength for Sensitivity (percent), scaled 0–100.

pac Percentage Accuracy in Classification (training).

p_mc Monte Carlo p-value, or NA if mcarlo = FALSE.

loo LOO results list, or NULL if loo = "off".

engine Character; "binary" or "multiclass".

confusion Confusion table. For the binary engine this is a list with integer counts TP, TN, FP, FN plus sensitivity and specificity as proportions [0,1]. For the multiclass engine this is a numeric matrix of (possibly weighted) counts.

Examples

```
## Binary (C = 2)
x <- c(1,2,3,4,5,6,7,8)
y <- c(0L,0L,0L,0L,1L,1L,1L,1L)
fit <- oda_fit(x, y, mcarlo = FALSE)
fit$ok
fit$rule$cut_value

## Multiclass (C = 3)
x3 <- c(1,2,3,4,5,6,7,8,9)
y3 <- c(1L,1L,1L,2L,2L,2L,3L,3L,3L)
fit3 <- oda_fit(x3, y3, mcarlo = FALSE)
fit3$rule$cut_values
fit3$rule$seg_classes
```

---

oda_infer_attr_types *Infer attribute types from a predictor data frame*

Description

Uses the same type-inference logic as oda_fit() ("auto" mode) to report the likely ODA attribute type for each column.

Usage

```
oda_infer_attr_types(X, miss_codes = NULL)
```

Arguments

X	Data frame of predictors.
miss_codes	Numeric vector of missing-code values to exclude when counting unique levels (default NULL).

Value

Data frame with one row per column in X: attribute (character), inferred_type (one of "ordered", "categorical", "binary"), n_unique (integer, excluding miss_codes and NA), n_missing (integer, NA count), n_miss_code (integer, miss_code hit count).

See Also

oda_fit, oda_clean_missing_codes

`oda_loo_multiclass_ordered` *Leave-one-out cross-validation for ordered multiclass ODA.*

Description

Leave-one-out cross-validation for ordered multiclass ODA.

Usage

```
oda_loo_multiclass_ordered(
  x,
  y,
  w0,
  priors_on_eff,
  degen,
  K_segments,
  miss_codes = NULL,
  cut_value_mode = c("midpoint", "lower", "upper"),
  grid_mode = c("fixed", "refit"),
  boundary_mode = c("megaoda_halfopen", "right_closed"),
  loo_use_samplerep = FALSE,
  loo_return_folds = FALSE,
  loo_priors_mode = c("fold", "global")
)
```

Arguments

x, y	Attribute and class vectors.
w0	Raw case weights.
priors_on_eff	Logical.
degen	Logical.
K_segments	Number of segments.
miss_codes	Optional missing codes.
cut_value_mode	"midpoint","lower","upper".
grid_mode	"refit" (true per-fold rebuild) or "fixed" (global grid).
boundary_mode	"megaoda_halfopen" or "right_closed".
loo_use_samplerep	Include samplerep in fold selection.
loo_return_folds	Return per-fold rules and debug info.
loo_priors_mode	"fold" (renorm each fold) or "global" (global wts).

Value

List with allowed, confusion_raw, confusion_weighted, y_pred, and optional fold_rule, fold_debug, fold_best_enum_id.

`oda_mc_p_value` *Monte Carlo Fisher-randomization p-value with Clopper-Pearson early stopping.*

Description

Monte Carlo Fisher-randomization p-value with Clopper-Pearson early stopping.

Usage

```
oda_mc_p_value(
  x,
  y,
  w = NULL,
  attr_type,
  priors_on,
  primary,
  secondary,
  miss_codes = NULL,
  chance_model = c("class", "attribute"),
  mc_iter = 25000L,
  mc_target = 0.05,
  mc_stop = 99.9,
  mc_stopup = NA,
  mc_adjust = FALSE,
  seed = NULL,
  ess_obs = NULL,
  obj_obs = NULL,
  direction = c("both", "off", "greater", "less"),
  direction_map = NULL
)
```

Arguments

`x`, `y`, `w`	Data for the current attribute (already cleaned).
`attr_type`	"ordered", "categorical", or "binary".
`priors_on`	Logical.
`primary`, `secondary`	Tie-break heuristic strings.
`miss_codes`	Optional numeric vector of additional missing codes.
`chance_model`	"class" (1/2) or "attribute" (1/k_attr).

mc_iter	Maximum iterations.
mc_target	Significance threshold (e.g. 0.05).
mc_stop	Confidence level for lower-tail stop (e.g. 99.9).
mc_stopup	Confidence level for upper-tail stop (e.g. 20 -> 0.20). Default NA (disabled).
mc_adjust	Kept for API compatibility; not used.
seed	Optional RNG seed.
ess_obs	Observed ESS (must be supplied).
obj_obs	Observed count-priors objective, used instead of ESS as the statistic when weights are non-uniform and priors are on (default NULL: computed from the data).
direction	Directional constraint forwarded from oda_univariate_core(): "both" (canonical non-directional default), "off" (synonym for "both"), "greater", or "less". Each permutation refit uses the same constraint.
direction_map	Named integer vector for categorical fixed-partition DIRECTIONAL. When supplied, each permutation evaluates the SAME fixed mapping on permuted y labels. Default NULL.

Value

List with p_mc, ge_count, iter_used, ess_obs.

---

oda_mean_pac *Mean PAC from sensitivity and specificity*

Description

Compute mean Percentage Accuracy in Classification.

Usage

```
oda_mean_pac(sens, spec)
```

Arguments

sens	Sensitivity (proportion [0, 1]).
spec	Specificity (proportion [0, 1]).

Value

Mean PAC as a proportion [0, 1].

oda_metrics *Retrieve scalar performance metrics from a fitted ODA model*

Description

Returns a list of scalar metrics present on the fit. No quantities are recomputed; absent fields appear as NA_real_. LOO p-value uses 2x2 Fisher exact when stored and available (p_status = "computed"); if the value is absent or NA the status is "not_computed" with an explicit reason. Multiclass/polychotomous LOO always returns p_status = "not_computed".

Usage

```
oda_metrics(fit, split = c("train", "loo"))
```

Arguments

fit	An oda_fit object.
split	One of "train" or "loo".

Value

Named list of scalar metrics.

oda_multiclass_unioda_core *Fit a univariate multiclass ODA model*

Description

Low-level engine for multiclass (C >= 3) Optimal Data Analysis. Handles ordered and categorical attributes. Most users should call oda_fit instead.

Usage

```
oda_multiclass_unioda_core(x, y, w = NULL,
  attr_type = c("auto","ordered","categorical","binary"),
  priors_on = TRUE, miss_codes = NULL, missing_code = NULL,
  K_segments = NULL, degen = FALSE,
  mcarlo = TRUE, mc_iter = 25000L, mc_target = 0.05,
  mc_stop = 99.9, mc_stopup = NA, mc_adjust = FALSE, mc_seed = NULL,
  loo = c("off","on"),
  boundary_mode = c("megaoda_halfopen","right_closed"),
  loo_opts = list(),
  direction = "off", direction_map = NULL)
```

Arguments

x	Attribute values (numeric or factor).
y	Integer class labels (will be re-coded to 1..C internally).
w	Optional numeric case weights.
attr_type	Attribute type.
priors_on	Inverse-frequency weighting.
miss_codes	Additional missing codes (scalar or vector).
missing_code	Alias for miss_codes.
K_segments	Number of segments; default = C.
degen	Allow degenerate solutions?
mcarlo	Run Monte Carlo p-value?
mc_iter, mc_target, mc_stop, mc_stopup, mc_adjust, mc_seed	MC parameters.
loo	"off" or "on".
boundary_mode	Boundary convention for ordered cut values.
loo_opts	Named list of LOO options passed to the LOO engine.
direction	Directional constraint (MPE Chapter 4). "ascending" forces segment s to class s for ordered multiclass; auto-creates identity direction_map for categorical when L == C. "descending" uses reverse assignment. Default "off" (nondirectional).
direction_map	Named integer vector for categorical fixed-partition DIRECTIONAL. Names are attribute levels; values are class labels 1..C. When supplied, bypasses the partition search and evaluates only the specified mapping. Default NULL.

Value

Named list. Key fields: ok, rule (with cut_values and seg_classes), confusion (weighted count matrix), pac, mean_pac, ess_pac, p_mc, loo, n_eff.

Note on confusion matrix: confusion contains weighted counts (priors-adjusted when priors_on = TRUE). For raw integer counts use loo$confusion_raw.

See Also

oda_fit

oda_power *ODA power analysis via simulation*

Description

Estimates planning power for unit-weighted binary 2×2 ODA-equivalent designs. The implemented design assumes fixed group sizes n1/n2 and binomial outcome probabilities p1/p2, then evaluates whether the resulting 2×2 table is significant by Fisher's exact test at the (optionally Sidak-adjusted) alpha.

Usage

```
oda_power(
  n1,
  n2 = n1,
  p1 = NULL,
  p2 = NULL,
  ess = NULL,
  alpha = 0.05,
  comp = 1L,
  nsim = 10000L,
  mc_seed = NULL
)
```

Arguments

n1	Integer (or integer vector) giving the per-group sample size for class 0. When a vector is supplied, power is estimated at each element.
n2	Integer (or integer vector) giving the per-group sample size for class 1. Defaults to n1 (balanced design). If scalar and n1 is a vector, n2 is recycled.
p1	Probability of the event in class 0. Ignored when ess is supplied.
p2	Probability of the event in class 1. Ignored when ess is supplied.
ess	Effect Strength for Sensitivity (percent, $0 < \text{ESS} < 100$) under the symmetric balanced convention. Mutually exclusive with p1/p2.
alpha	Nominal significance level. Default 0.05. May be a vector to evaluate power at multiple alpha levels simultaneously.
comp	Number of comparisons for Sidak multiple-comparison correction. Default 1 (no correction). Must be a single positive integer.
nsim	Number of Monte Carlo replications per cell. Default 10000.
mc_seed	Integer seed passed to set.seed() once before all simulations, or NULL to use the current RNG state.

Details

This is the binary lowest-measurement planning case discussed by Rhodes (2020). See also Yarnold and Soltysik (2005) for the underlying ODA/Fisher isomorphism. **Scope:** unit-weighted, binary class, binary (2-level) attribute only. This is not a general CTA, LORT, SDA, weighted, or multiclass power method.

Method: For each Monte Carlo replicate, binomial draws are generated under (`p1`, `p2`) with fixed group sizes (`n1`, `n2`). The resulting 2×2 table is tested by Fisher's exact test; power is the proportion of replicates in which the null is rejected. The prospective sampling treats group sizes as fixed and outcomes as binomial within each group; the Fisher test is then applied to the generated table with its realized marginals. This is the standard simulation-based power approach for 2×2 contingency analyses.

Effect-size input: Specify the effect either as per-group proportions `p1` and `p2` directly, or as `ess` (Effect Strength for Sensitivity, percent) under the symmetric balanced convention: $p_2 = \mathrm{ESS}/200 + 0.5$, $p_1 = 1 - p_2$.

Sidak correction: When `comp > 1`, the working α is Sidak-adjusted: $\alpha_{\mathrm{adj}} = 1 - (1 - \alpha)^{1/\mathrm{comp}}$.

Value

An object of class `"oda_power"`, a list with elements:

`power` Numeric matrix (rows = n1, cols = alpha_adj) of estimated power. Simplified to a named vector if one dimension is scalar, or to a scalar if both are.

`n1`, `n2` Per-group sample sizes.

`p1`, `p2` Per-group event rates used.

`ess_input` ESS supplied, or `NA` if `p1`/`p2` used.

`alpha`, `alpha_adj` Input and Sidak-adjusted alpha.

`comp`, `nsim`, `mc_seed` Input parameters.

References

Rhodes, N. J. (2020). Statistical power analysis in ODA, CTA and Novometrics. *Optimal Data Analysis*, 9. https://odajournal.files.wordpress.com/2020/02/v9a5.pdf

Yarnold PR, Soltysik RC (2005). *Optimal Data Analysis: A Guidebook with Software for Windows.* Washington, DC: APA Books.

Examples

```
# Power for ESS = 48%, n = 50 per group (CRAN-safe nsim; use 10000L for publication)
oda_power(n1 = 50, ess = 48, nsim = 500L, mc_seed = 42L)

# Power curve across a range of n
oda_power(n1 = c(30, 50, 80), ess = 48, nsim = 500L, mc_seed = 42L)

# Direct proportions (p1 = 0.26, p2 = 0.74)
oda_power(n1 = 50, p1 = 0.26, p2 = 0.74, nsim = 500L, mc_seed = 42L)

# Sidak correction for 3 comparisons
```

```
oda_power(n1 = 80, ess = 48, comp = 3L, nsim = 500L, mc_seed = 42L)
```

`oda_predictions` *Retrieve predictions from a fitted ODA model*

Description

Returns stored LOO predictions when available, or calls `predict.oda_fit()` on supplied `newdata`. Training predictions are not stored by the engine; supply `newdata` to obtain them.

Usage

```
oda_predictions(fit, split = c("train", "loo"), newdata = NULL, ...)
```

Arguments

`fit`	An `oda_fit` object.
`split`	One of `"train"` or `"loo"`.
`newdata`	For `split = "train"`: numeric vector or single-column data frame; if `NULL`, returns `NULL` with a message.
`...`	Passed to `predict.oda_fit()` when `newdata` is supplied.

Value

Integer vector of predictions or `NULL`.

`oda_propensity_weights`
ODA rule strata propensity weights

Description

Computes propensity weights from the two rule strata (left and right of the ODA cutpoint) using stored training confusion counts. Implements the Yarnold/Linden stratum-weight formula:

$$w = n_s \times \Pr(Z = z)/n_{z,s}$$

Usage

```
oda_propensity_weights(fit, adjusted = TRUE)
```

Arguments

`fit`	An `oda_fit` object with `ok == TRUE`.
`adjusted`	Logical; if `TRUE` (default), applies a one-hypothetical-misclassification adjustment when a class is absent from a rule stratum.

Details

Currently implemented for **binary (C=2) ODA fits only**.

The fitted model must have been trained with the treatment/exposure/group membership as the class variable (y), not a clinical outcome. The user is responsible for this labeling decision.

Value

Data frame with one row per (stratum, class) combination: `stratum_id` (1L = rule predicts class 0, 2L = rule predicts class 1), `predicted_class` (integer), `class` (character), `class_n` (integer), `stratum_n` (integer), `marginal_class_n` (integer), `marginal_total_n` (integer), `marginal_class_probability` (numeric), `propensity_weight` (numeric), `undefined_empirical` (logical), `adjusted` (logical), `adjusted_propensity_weight` (numeric), `model_family` ("oda").

See Also

`cta_propensity_weights`, `lort_propensity_weights`

---

oda_readiness_check *Preflight readiness check for ODA / CTA analysis*

Description

Validates a predictor frame, class vector, and optional weight vector before fitting. Returns a structured report. Does not modify inputs.

Usage

```
oda_readiness_check(
  X,
  y,
  w = NULL,
  miss_codes = NULL,
  binary_only = FALSE,
  min_class_n = 5L
)
```

Arguments

`X`	Data frame of predictors.
`y`	Integer class/group vector.
`w`	Optional numeric weight vector.
`miss_codes`	Numeric vector of missing-code values (default `NULL`).
`binary_only`	Logical; flag > 2 classes as an issue (default `FALSE`).
`min_class_n`	Minimum observations per class; flags if any class is below this threshold (default `5L`).

Details

Flags:

- Missing class/group variable.
- Non-binary group when `binary_only = TRUE`.
- Non-numeric weights, wrong-length weights, NA/Inf/zero weights.
- Missing-code patterns in predictors (if `miss_codes` supplied).
- Constant attributes (zero variance after miss-code removal).
- Insufficient class counts (< `min_class_n`).
- Attribute-type uncertainty (logical/factor columns).

Value

Named list with: `ok` (logical, TRUE if no issues), `issues` (character vector), `warnings` (character vector, non-fatal), `n_obs` (integer), `group_report` (from `oda_validate_group()`), `weight_report` (from `oda_validate_weights()`), `attr_types` (from `oda_infer_attr_types()`), `constant_attrs` (character vector of constant columns).

See Also

`oda_validate_group`, `oda_validate_weights`, `oda_infer_attr_types`, `oda_clean_missing_codes`

---

oda_rule_predict *Apply a binary ODA rule to new data*

Description

Predict class labels (0 or 1) for new attribute values using a fitted binary ODA rule.

Usage

```
oda_rule_predict(x, rule)
```

Arguments

`x`	Numeric or character attribute values.
`rule`	A rule list returned in `fit$rule` from `oda_univariate_core` or `oda_fit`.

Value

Integer vector of predicted class labels (0 or 1).

oda_rule_predict_multiclass *Apply a multiclass ODA rule to new data*

Description

Predict class labels for new attribute values using a fitted multiclass ODA rule.

Usage

```
oda_rule_predict_multiclass(x, rule,
  boundary = c("megaoda_halfopen","right_closed"))
```

Arguments

x	Numeric attribute values.
rule	A rule list from oda_multiclass_unioda_core or oda_fit.
boundary	Boundary convention. Default "megaoda_halfopen" matches MegaODA.exe.

Value

Integer vector of predicted class labels.

oda_sample_size *ODA minimum sample size via bisection*

Description

Finds the minimum per-group sample size n (balanced design) at which power reaches or exceeds power_target. Uses bisection over oda_power() with a fixed RNG seed for stable search.

Usage

```
oda_sample_size(
  power_target = 0.8,
  p1 = NULL,
  p2 = NULL,
  ess = NULL,
  alpha = 0.05,
  comp = 1L,
  nsim = 10000L,
  mc_seed = 42L,
  n_min = 2L,
  n_max = 2000L
)
```

Arguments

power_target	Target power. Default 0.80.
p1	Probability of the event in class 0. Ignored when ess is supplied.
p2	Probability of the event in class 1. Ignored when ess is supplied.
ess	Effect Strength for Sensitivity (percent, $0 <$ ESS < 100). Mutually exclusive with p1/p2.
alpha	Nominal significance level. Default 0.05.
comp	Number of comparisons for Sidak correction. Default 1.
nsim	Number of Monte Carlo replications per candidate n. Default 10000.
mc_seed	Integer seed used for every oda_power() call during the bisection. Using a fixed seed yields stable, reproducible results. Default 42L.
n_min	Minimum n to search. Default 2.
n_max	Maximum n to search. If power at n_max is still below power_target, an error is raised. Default 2000.

Details

Scope: unit-weighted, binary class, binary (2-level) attribute only. This is not a general CTA, LORT, SDA, weighted, or multiclass sample-size method. For unbalanced designs, call oda_power() directly across a candidate grid.

Value

An object of class "oda_sample_size", a list with elements:

n Minimum per-group sample size achieving power_target.

power_achieved Estimated power at n.

power_target Input target power.

p1, p2, ess_input Effect-size inputs.

alpha, alpha_adj, comp Alpha parameters.

nsim, mc_seed Simulation parameters.

References

Rhodes, N. J. (2020). Statistical power analysis in ODA, CTA and Novometrics. *Optimal Data Analysis*, 9. https://odajournal.files.wordpress.com/2020/02/v9a5.pdf

Yarnold PR, Soltysik RC (2005). *Optimal Data Analysis: A Guidebook with Software for Windows.* Washington, DC: APA Books.

Examples

```
# Minimum n for ESS = 48%, 80% power (use nsim >= 500L for publication-quality estimates)
oda_sample_size(ess = 48, nsim = 200L, mc_seed = 42L)


# 90% power target (publication-quality nsim)
oda_sample_size(ess = 48, power_target = 0.90, nsim = 500L, mc_seed = 42L)
```

---

oda_univariate_core *Fit a univariate binary-class ODA model*

Description

Low-level engine for binary-class Optimal Data Analysis. Handles ordered, categorical, and binary attributes with optional prior-odds weighting, Monte Carlo p-value, and leave-one-out validity analysis. Most users should call `oda_fit` instead.

Usage

```
oda_univariate_core(x, y, w = NULL,
  attr_type = c("auto","ordered","categorical","binary"),
  priors_on = TRUE, primary = NULL, secondary = NULL,
  miss_codes = NULL, missing_code = NULL,
  loo = c("off","stable","pvalue"), loo_alpha = 0.05,
  mcarlo = TRUE, mc_iter = 25000L, mc_target = 0.05,
  mc_stop = 99.9, mc_stopup = NA_real_, mc_adjust = FALSE,
  mc_seed = NULL, chance_model = c("class","attribute"),
  eval_order = c("mc_then_loo","loo_then_mc"),
  mindenom = 1L,
  direction = c("both","off","greater","less"),
  direction_map = NULL)
```

Arguments

`x`	Attribute values.
`y`	Binary class labels, coercible to 0/1 integers.
`w`	Optional numeric case weights.
`attr_type`	Attribute type. `"auto"` detects from data.
`priors_on`	If `TRUE`, use inverse-frequency weighting.
`primary`	Primary tie-break heuristic. `NULL` = default by `priors_on`.
`secondary`	Secondary tie-break. `NULL` = `"samplerep"`.
`miss_codes`	Additional missing-value codes.
`missing_code`	Scalar alias for `miss_codes`.

loo	"off", "stable", or "pvalue".
loo_alpha	Alpha threshold for loo = "pvalue".
mcarlo	Run Monte Carlo p-value?
mc_iter	Maximum MC iterations.
mc_target	Significance threshold.
mc_stop	Confidence level (percent) for STOP early stopping.
mc_stopup	Confidence level (percent) for STOPUP.
mc_adjust	Legacy parameter; unused.
mc_seed	RNG seed.
chance_model	"class" (1/2) or "attribute" (1/k_attr) baseline.
eval_order	Controls whether Monte Carlo testing is run before LOO validation or whether eligible ordered-cut LOO stability is checked before Monte Carlo. The default "mc_then_loo" preserves standalone UniODA behaviour. CTA tree building uses "loo_then_mc" internally to reject LOO-unstable ordered-cut candidates before spending MC iterations.
mindenom	Minimum raw observation count required in each child node for a candidate cut to be evaluated. Default 1 (no enforcement).
direction	Directional hypothesis (MPE Chapter 2 scope): "both" (default) or its synonym "off", "greater" (high x predicts class 1; Chapter 2 DIRECTION < 0 1), or "less" (low x predicts class 1; Chapter 2 DIRECTION > 0 1). Ordered and binary attributes only. When direction_map is supplied, categorical DIRECTIONAL is also supported via a fixed mapping; passing direction in "greater"/"less" with a categorical attribute and no direction_map returns ok = FALSE.
direction_map	Named integer vector for categorical fixed-partition DIRECTIONAL (MPE Chapter 4). Names are attribute levels (character); values are 0/1 coded class labels. All attribute levels must be covered. When supplied for a categorical attribute, the specified partition is evaluated without searching alternatives; LOO predictions are trivially stable. Default NULL.

Value

Named list. Key fields: ok, rule, confusion (list with integer counts TP, TN, FP, FN and rate fields sensitivity, specificity as proportions in [0,1]), ess, pac, p_mc, loo, n_eff.

See Also

oda_fit, oda_multiclass_unioda_core

oda_validate_group *Validate a class / group variable*

Description

Returns a structured report list rather than erroring. Useful as a preflight check before passing y to oda_fit() or cta_fit().

Usage

```
oda_validate_group(y, binary_only = FALSE)
```

Arguments

y	Integer (or coercible to integer) class vector.
binary_only	Logical; if TRUE, flags more than 2 classes as an issue (useful for UniODA workflows).

Value

Named list with: ok (logical), n_classes (integer), class_levels (integer vector), class_counts (named integer table), issues (character vector, empty if ok).

oda_validate_weights *Validate a case weight vector*

Description

Returns a structured report rather than throwing an error. NULL weights are valid (interpreted as unit weights) and return ok = TRUE.

Usage

```
oda_validate_weights(w, n)
```

Arguments

w	Numeric weight vector or NULL.
n	Expected length of w.

Value

Named list with: ok (logical), issues (character vector, empty if ok), n_weights (integer or NA), range (numeric(2) or NULL).

ort_plot_data *Renderer-independent layout data for a LORT composite tree*

Description

Computes node positions and edge metadata for `plot.cta_ort`. Terminal nodes receive integer x-slot positions (left-to-right in DFS right-first order); internal nodes are centered over their children.

Usage

```
ort_plot_data(object, target_class = NULL, class_labels = NULL, digits = 1L)
```

Arguments

`object`	A `cta_ort` from `cta_fit(..., recursive = TRUE)`.
`target_class`	Integer target class for terminal node annotation, or `NULL` for a structural plot.
`class_labels`	Optional named character vector of class display names.
`digits`	Integer decimal places for proportion labels. Default 1.

Value

A list with elements:

`nodes` data.frame: `node_id`, `depth`, `x`, `y`, `is_terminal`, `label`, `n`, `stop_reason`.

`edges` data.frame: `from_id`, `to_id`, `x0`, `y0`, `x1`, `y1`, `label`.

`strata` The strata table from the LORT object.

Note

`ort_plot_data` is a legacy compatibility name for the LORT method. See `print.cta_ort` for the naming note.

plot.cta_ort *Plot method for Locally Optimal Recursive Tree (LORT)*

Description

Renders the composite LORT using G1 base-R conventions: ellipses for split nodes, rectangles for terminal nodes, directed arrows for edges.

Usage

```
## S3 method for class 'cta_ort'
plot(
  x,
  target_class = NULL,
  class_labels = NULL,
  digits = 1L,
  main = "LORT",
  split_fill = "#D9EAF7",
  endpoint_fill = "#D9F7E6",
  endpoint_palette = NULL,
  border_col = "grey30",
  text_col = "black",
  edge_col = "grey40",
  arrow_col = NULL,
  show_caption = FALSE,
  cex = 0.75,
  ...
)
```

Arguments

x	A cta_ort object.
target_class	Integer target class for terminal node annotation; NULL for a structural plot.
class_labels	Optional named character vector of class display names.
digits	Decimal places for proportion labels. Default 1.
main	Plot title. Default "ORT".
split_fill	Fill color for split (internal) ellipse nodes.
endpoint_fill	Default fill for terminal rectangle nodes.
endpoint_palette	Palette for terminal nodes when target_class is supplied; a function(n) or character color vector.
border_col	Border color for all nodes. Default "grey30".
text_col	Text color for node labels. Default "black".
edge_col	Color for directed edge arrows. Default "grey40".
arrow_col	Arrow color; NULL (default) uses edge_col.
show_caption	Logical; add color-encoding caption when target_class is supplied. Default FALSE.
cex	Text expansion factor. Default 0.75.
...	Unused.

Value

invisible(pd), the layout list from ort_plot_data.

Note

plot.cta_ort and ort_plot_data are legacy compatibility names for the LORT method. See print.cta_ort for the naming note.

See Also

ort_plot_data, plot.cta_tree

---

plot.cta_tree *Plot a fitted CTA tree*

Description

Native base-R CTA visualization. Calls cta_plot_data for layout; uses only base graphics - no external package dependencies.

Split (internal) nodes are drawn as **ellipses**; terminal endpoint nodes are drawn as **rectangles**; edges are **directed arrows**. Split nodes show the split attribute, node-level ESS or WESS, and observation count. Without target_class, leaf nodes show the majority-class prediction and observation count. With target_class, leaf nodes show the target-class count, percentage, predicted class, and stage from cta_staging_table. Edge labels show the branch condition (e.g. "V14<=0.5").

Color note: when target_class is supplied, endpoint fill colors are assigned by ascending rank of each endpoint's target-class proportion within this tree. Colors encode relative position in the endpoint distribution and do *not* imply clinical thresholds or categories. Supply a custom palette via endpoint_palette to change the color encoding. Use show_caption = TRUE to render an explicit note on the plot.

cta_plot_data is the renderer-independent data contract. This function (plot.cta_tree) is the current native base-R renderer.

Usage

```
## S3 method for class 'cta_tree'
plot(x,
     target_class = NULL, class_labels = NULL, digits = 1,
     main = "CTA Tree", show_counts = TRUE, show_stage = TRUE,
     endpoint_palette = NULL, endpoint_fill = "#D9F7E6",
     split_fill = "#D9EAF7", node_col_split = NULL,
     node_col_leaf = NULL, edge_col = "grey40",
     border_col = "grey30", text_col = "black",
     arrow_col = NULL, show_caption = FALSE,
     cex = 0.75, ...)
```

Arguments

x	A cta_tree from oda_cta_fit.
target_class	Integer target class for endpoint annotation; passed to cta_plot_data. NULL (default) produces a structural plot without endpoint enrichment.

class_labels	Optional display names for class labels; passed to cta_plot_data. See cta_plot_data for accepted formats.
digits	Decimal places for percentage labels in enriched endpoint nodes; passed to cta_plot_data. Default 1.
main	Character plot title. Default "CTA Tree".
show_counts	Logical; include n=a/b raw counts in endpoint labels when target_class is supplied. Default TRUE.
show_stage	Logical; include Stage s line in endpoint labels when target_class is supplied. Default TRUE.
endpoint_palette	Palette for endpoint fill colors when target_class is supplied; passed to cta_plot_data. NULL uses the default gradient. Accepts a palette function(n) or a character vector of colors.
endpoint_fill	Default fill colour for leaf (terminal) nodes when target_class is NULL. Default "#D9F7E6" (light green).
split_fill	Fill colour for split (internal) ellipse nodes. Default "#D9EAF7" (light blue).
node_col_split	Legacy alias for split_fill; overrides it when non-NULL.
node_col_leaf	Legacy alias for endpoint_fill; overrides it when non-NULL.
edge_col	Colour for directed edge arrows. Default "grey40".
border_col	Border colour for all nodes. Default "grey30".
text_col	Text colour for node labels. Default "black".
arrow_col	Arrow colour for directed edges. NULL (default) uses edge_col.
show_caption	Logical; if TRUE and target_class is supplied, adds a bottom caption: *"Endpoint fill: relative target-class proportion within this tree. Not a clinical threshold."* Default FALSE.
cex	Text expansion factor for node labels. Default 0.75.
...	Unused; included for S3 compatibility.

Value

invisible(pd), where pd is the cta_plot_data list used to render the plot. The caller can inspect layout coordinates, enrichment columns, and endpoint annotations from the returned object.

See Also

cta_plot_data, cta_staging_table, oda_cta_fit

Examples

```
data(mtcars)
X    <- mtcars[, c("cyl", "disp", "hp", "wt")]
y    <- as.integer(mtcars$am)
tree <- suppressMessages(
  oda_cta_fit(X, y, mindenom = 5L, mc_iter = 500L, mc_seed = 42L,
              loo = "off")
```

```
)

# Structural plot
plot(tree)

# Target-class enriched plot with custom labels
plot(tree, target_class = 1L,
     class_labels = c("0" = "Manual", "1" = "Auto"))

# Custom palette (white to dark red)
plot(tree, target_class = 1L,
     endpoint_palette = c("#ffffff", "#c62828"))
```

`plot_balance_love` *Love plot for covariate balance (SMD)*

Description

A direct alias for `plot_smd_balance`. Produces a Cleveland-style Love plot of absolute SMD with conventional threshold reference lines.

Usage

```
plot_balance_love(x, ...)
```

Arguments

`x`	A `"smd_balance_table"` object from `smd_balance_table`.
`...`	Arguments forwarded to `plot_smd_balance`.

Value

A `ggplot` object.

See Also

`plot_smd_balance`, `smd_balance_table`

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  group <- c(rep(0L, 20), rep(1L, 20))
  X     <- data.frame(A = c(rep(0L,20), rep(1L,20)),
                      B = rnorm(40))
  smd <- smd_balance_table(group, X)
  p   <- plot_balance_love(smd)
  print(p)
}
```

plot_cta_balance *Plot CTA multivariate covariate balance*

Description

Renders the CTA covariate balance result. When no discriminating tree was found (`status = "no_tree"`), a message panel confirms favorable evidence of multivariable balance under the declared constraints. When a valid tree or stump was found, the tree diagram is rendered via `plot_cta_tree`.

Usage

```
plot_cta_balance(
  x,
  target_class = 1L,
  color_by = c("target_rate", "prediction", "none"),
  main = NULL,
  subtitle = NULL,
  ...
)
```

Arguments

`x`	A `"cta_balance_plot_data"` object from `cta_balance_plot_data`, or a `"cta_balance_table"` object from `cta_balance_table` (coerced internally via `cta_balance_plot_data`; never calls the fitting function).
`target_class`	Integer; target class for leaf-node coloring. Default `1L`.
`color_by`	Character; leaf-node fill: `"target_rate"` (default), `"prediction"`, `"none"`.
`main`	Character; plot title. Default: auto-generated from ESS/WESS.
`subtitle`	Character; plot subtitle.
`...`	Additional arguments forwarded to `plot_cta_tree` when a tree is rendered.

Details

This function is a pure renderer. It does not fit any CTA models and does not accept `group` or `X` arguments.

Value

A `ggplot` object.

See Also

`cta_balance_plot_data`, `cta_balance_table`, `plot_cta_tree`

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  X <- data.frame(
    A = c(rep(0L,20), rep(1L,20), rep(1L,20)),
    B = c(rep(0L,20), rep(0L,20), rep(1L,20))
  )
  group <- c(rep(0L, 40), rep(1L, 20))
  ct  <- cta_balance_table(group, X, mindenom = 5L,
                           mc_iter = 200L, mc_seed = 42L)
  cpd <- cta_balance_plot_data(ct)
  p   <- plot_cta_balance(cpd)
  print(p)
}
```

---

plot_cta_balance_effects

Evidence card for CTA multivariate covariate balance

Description

Renders an evidence-interval card from a cta_balance_effect_summary object. Each row of the card corresponds to one analysis scale. The plot uses the same interval encoding as plot_oda_balance_effects: thick black = bootstrap CI, thin gray = chance CI, open circle = observed ESS/WESS.

Usage

```
plot_cta_balance_effects(x, main = NULL, subtitle = NULL, xlim = NULL, ...)
```

Arguments

x	A "cta_balance_effect_summary" object from cta_balance_effect_summary.
main	Optional character; plot title.
subtitle	Optional character; plot subtitle.
xlim	Optional numeric(2); x-axis limits.
...	Ignored; reserved for future use.

Details

When status = "no_tree" for all rows, a favorable-balance message panel is returned instead of an interval plot.

This function does not fit any models.

Value

A ggplot object.

See Also

cta_balance_effect_summary

Examples

```
group <- c(0L, 0L, 0L, 0L, 1L, 1L, 1L, 1L)
X     <- data.frame(v1 = c(1, 2, 3, 4, 5, 6, 7, 8),
                    v2 = c(0L, 1L, 0L, 1L, 0L, 1L, 0L, 1L))
ces <- cta_balance_effect_summary(group, X, mindenom = 5L,
                                  mc_iter = 200L, mc_seed = 42L,
                                  nboot = 20L, chance_iter = 20L)
plot_cta_balance_effects(ces)
```

---

plot_cta_family *Plot a CTA descendant family member using ggplot2*

Description

Renders a publication-quality CTA tree diagram for a single member of a `cta_family` object (indexed inspection), or a named list of plots for all members (`show_all = TRUE`). Requires the **ggplot2** package.

Usage

```
plot_cta_family(
  family,
  index = 1L,
  min_d = FALSE,
  show_all = FALSE,
  layout = c("multipanel", "list"),
  ncol = 1L,
  target_class = 1L,
  color_by = c("none", "target_rate", "prediction"),
  label_detail = c("simple", "full"),
  show_node_ess = FALSE,
  show_p = TRUE,
  show_loo = TRUE,
  main = NULL,
  subtitle = NULL,
  show_rule = TRUE,
  show_metrics = FALSE,
  short_edge_labels = TRUE,
  node_text_size = 3.5,
  edge_text_size = 3.2,
  palette = NULL
)
```

Arguments

family	A cta_family object from cta_descendant_family.
index	Integer or "min_d"; which family member to render. Default 1L. Use "min_d" to render the member with minimum D-statistic. Ignored when show_all = TRUE.
min_d	Logical; convenience shorthand for index = "min_d". When TRUE, renders the minimum-D family member regardless of index. Default FALSE.
show_all	Logical; if TRUE, render all family members. Output format is controlled by layout. Default FALSE.
layout	Character; "multipanel" (default, requires **patchwork**) returns a single combined figure; "list" returns a named list of ggplot objects. Only relevant when show_all = TRUE.
ncol	Integer; number of columns in the multipanel grid. Default 1L. Only used when show_all = TRUE and layout = "multipanel".
target_class	Integer; target class for endpoint coloring (default 1L).
color_by	Character; leaf-node fill. "none" (default), "target_rate", or "prediction".
label_detail	Character; "simple" (default) or "full".
show_node_ess	Logical; append node ESS to split labels. Default FALSE.
show_p	Logical; append MC p = X.XXX to split-node labels. Default TRUE.
show_loo	Logical; append LOO status/p to split-node labels. Default TRUE.
main	Character; plot title. Default: auto-generated with MINDENOM and D.
subtitle	Character; plot subtitle.
show_rule	Logical; show edge condition labels. Default TRUE.
show_metrics	Logical; append ESS/D to subtitle. Default FALSE.
short_edge_labels	Logical; strip attribute prefix from edge labels. Default TRUE.
node_text_size	Numeric; text size for node labels. Default 3.5.
edge_text_size	Numeric; text size for edge labels. Default 3.2.
palette	Named list for color overrides.

Value

A ggplot object (single member or multipanel), or (when show_all = TRUE and layout = "list") a named list of ggplot objects.

See Also

cta_descendant_family, plot_cta_tree, plot_lort_tree, ggsave

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  X <- data.frame(x1 = c(rep(0L,20), rep(1L,20)),
                  x2 = c(rep(0L,10), rep(1L,10), rep(0L,10), rep(1L,10)))
  y <- c(rep(0L,30), rep(1L,10))
  fam <- cta_descendant_family(X, y, mc_iter=200L, mc_seed=42L, loo="off")
  p   <- plot_cta_family(fam, index=1L)
  print(p)
}
```

`plot_cta_tree` *Plot a CTA tree using ggplot2*

Description

Renders a publication-quality tree diagram for a fitted CTA tree. Requires the **ggplot2** package (listed in `Suggests`); if unavailable, a clear error is raised.

Usage

```
plot_cta_tree(
  x,
  target_class = 1L,
  color_by = c("none", "target_rate", "prediction"),
  label_detail = c("simple", "full"),
  show_node_ess = FALSE,
  show_p = TRUE,
  show_loo = TRUE,
  main = NULL,
  subtitle = NULL,
  show_rule = TRUE,
  show_metrics = FALSE,
  short_edge_labels = TRUE,
  node_text_size = 3.5,
  edge_text_size = 3.2,
  palette = NULL
)
```

Arguments

`x`	A `cta_tree` object from `cta_fit` / `oda_cta_fit`, or the list returned by `cta_plot_data`.
`target_class`	Integer; target class for endpoint coloring and target-rate annotation (default `1L`). Ignored when `x` is already `cta_plot_data` output.
`color_by`	Character; controls leaf-node fill color. `"none"` (default): white fill (B/W publication default); `"target_rate"`: continuous gradient by target-class proportion; `"prediction"`: discrete fill by predicted class.

label_detail	Character; node label verbosity. "simple" (default): canonical MPE-style labels (attribute + condition + n for split nodes; Stage/class/n/rate for terminal nodes). "full": same content (reserved for future extension).
show_node_ess	Logical; if TRUE, append the node-level ESS/WESS to split-node labels. Default FALSE.
show_p	Logical; if TRUE (default), append MC p = X.XXX to each split-node label when the MC permutation p-value is available.
show_loo	Logical; if TRUE (default), append the LOO result to each split-node label: LOO: STABLE when loo = "stable", LOO p = X.XXX when loo = "pvalue". Nothing is shown when loo = "off".
main	Character; plot title. Default: auto-generated from tree structure (n, endpoints, ESS/D).
subtitle	Character; plot subtitle.
show_rule	Logical; show branch condition labels on edges. Default TRUE.
show_metrics	Logical; if TRUE, appends an ESS/WESS and D line to the plot subtitle. Default FALSE.
short_edge_labels	Logical; if TRUE (default), strip the attribute-name prefix from edge labels so that "x1 <= 24.5" renders as "<= 24.5".
node_text_size	Numeric; ggplot text size for node labels. Default 3.5.
edge_text_size	Numeric; ggplot text size for edge labels. Default 3.2.
palette	Named list for color overrides: internal, low, high. NULL uses defaults.

Value

A ggplot object. Print it, modify it, or save with ggplot2::ggsave().

See Also

cta_fit, cta_plot_data, plot_lort_tree, plot_cta_family, ggsave

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  X <- data.frame(x1 = c(1,2,3,4,5,6,7,8),
                  x2 = c(0L,0L,1L,0L,1L,1L,0L,1L))
  y <- c(1L,1L,1L,1L,2L,2L,2L,2L)
  tree <- cta_fit(X, y, mindenom=1L, mc_iter=500L, mc_seed=42L, loo="off")
  p <- plot_cta_tree(tree)
  print(p)
}
```

plot_lort_path *Plot the full local CTA models along a LORT recursion path*

Description

Returns a named list of ggplot objects, one per LORT node on the path from the root to the requested `index`. Each panel shows the *full* local CTA model embedded at that LORT node – not a stump summary.

Usage

```
plot_lort_path(
  x,
  index = 1L,
  layout = c("multipanel", "list"),
  ncol = 1L,
  target_class = 1L,
  color_by = c("none", "target_rate", "prediction"),
  label_detail = c("simple", "full"),
  show_node_ess = FALSE,
  show_p = TRUE,
  show_loo = TRUE,
  show_rule = TRUE,
  show_metrics = FALSE,
  short_edge_labels = TRUE,
  node_text_size = 3.5,
  edge_text_size = 3.2,
  palette = NULL,
  ...
)
```

Arguments

`x`	A `cta_ort` object from `lort_fit`.
`index`	Integer; target LORT node index (end of path).
`layout`	Character; `"multipanel"` (default) returns a single arranged figure with one panel per path node – requires the **patchwork** package. `"list"` returns the current named list of ggplot objects.
`ncol`	Integer; number of columns in the multipanel layout. Default `1L` (vertical stack).
`target_class`	Integer; target class for node coloring. Default `1L`.
`color_by`	Character; leaf fill mode. Default `"none"`.
`label_detail`	Character; `"simple"` (default) or `"full"`.
`show_node_ess`	Logical. Default `FALSE`.
`show_p`	Logical; append `MC p = X.XXX` to split-node labels. Default `TRUE`.

show_loo	Logical; append LOO status/p to split-node labels. Default TRUE.
show_rule	Logical. Default TRUE.
show_metrics	Logical. Default FALSE.
short_edge_labels	Logical. Default TRUE.
node_text_size	Numeric. Default 3.5.
edge_text_size	Numeric. Default 3.2.
palette	Named list; color overrides.
...	Ignored; reserved.

Details

The list is named index_1, index_2, etc. (one name per LORT node on the path). Terminal nodes with no model get a message panel.

Value

With layout = "multipanel": a single patchwork/ggplot object containing all path panels. With layout = "list": a named list of ggplot objects.

See Also

lort_index_path, lort_local_tree, lort_path_table, plot_lort_tree

---

plot_lort_tree *Plot a LORT (Locally Optimal Recursive Tree) using ggplot2*

Description

Renders a publication-quality CTA tree diagram for a single sub-tree within a LORT object (indexed inspection), or a named list of plots for all sub-trees (show_all = TRUE). Requires the **ggplot2** package.

Usage

```
plot_lort_tree(
  x,
  index = 1L,
  show_all = FALSE,
  show_path = FALSE,
  target_class = 1L,
  color_by = c("none", "target_rate", "prediction"),
  label_detail = c("simple", "full"),
  show_node_ess = FALSE,
  show_p = TRUE,
  show_loo = TRUE,
```

```
  main = NULL,
  subtitle = NULL,
  show_rule = TRUE,
  show_metrics = FALSE,
  short_edge_labels = TRUE,
  node_text_size = 3.5,
  edge_text_size = 3.2,
  palette = NULL,
  ...
)
```

Arguments

x	A cta_ort object from lort_fit.
index	Integer or character; which LORT node (sub-tree) to render. Default 1L (root sub-tree). Ignored when show_all = TRUE.
show_all	Logical; if TRUE, return a named list of ggplot objects, one per LORT node, in node-id order. Default FALSE.
show_path	Logical; if TRUE, delegates to plot_lort_path to render the full path from root to index. Default FALSE.
target_class	Integer; target class for endpoint coloring and target-rate annotation (default 1L).
color_by	Character; controls leaf-node fill color. "none" (default): white fill; "target_rate": gradient; "prediction": discrete fill by predicted class.
label_detail	Character; "simple" (default) or "full".
show_node_ess	Logical; append node-level ESS to split labels. Default FALSE.
show_p	Logical; append MC p = X.XXX to split-node labels. Default TRUE.
show_loo	Logical; append LOO: STABLE or LOO p = X.XXX to split-node labels when LOO was active. Default TRUE.
main	Character; plot title. Default: auto-generated. When show_all = TRUE each plot's title includes the node id.
subtitle	Character; plot subtitle.
show_rule	Logical; show branch condition labels on edges.
show_metrics	Logical; append ESS/D to subtitle. Default FALSE.
short_edge_labels	Logical; strip attribute-name prefix from edge labels. Default TRUE.
node_text_size	Numeric; text size for node labels. Default 3.5.
edge_text_size	Numeric; text size for edge labels. Default 3.2.
palette	Named list for color overrides. NULL uses defaults.
...	Additional arguments passed to plot_lort_path when show_path = TRUE; otherwise ignored.

Value

A ggplot object, or (when show_all = TRUE) a named list of ggplot objects.

See Also

lort_fit, plot_cta_tree, plot_cta_family, ggsave

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  X <- data.frame(
    A = c(rep(0L,20), rep(1L,20), rep(1L,20)),
    B = c(rep(0L,20), rep(0L,20), rep(1L,20))
  )
  y <- c(rep(0L,40), rep(1L,20))
  lort <- lort_fit(X, y, mc_iter=100L, mc_seed=42L, loo="off", min_n=5L)
  p <- plot_lort_tree(lort, index=1L)
  print(p)
}
```

---

plot_oda_balance *Plot ODA covariate balance*

Description

Renders a horizontal dot-plot of ODA-based covariate balance diagnostics. Each covariate is shown as a point; the x-axis is ESS or WESS (0-100 %), and point color reflects significance status. The function is a pure renderer: it does not fit any ODA models and does not accept `group` or `X` arguments. If `abs_smd` is absent from the plot-data it is not plotted.

Usage

```
plot_oda_balance(
  x,
  p_col = "p_mc",
  rank_by = "abs_ess",
  main = NULL,
  subtitle = NULL,
  show_significance = TRUE,
  palette = NULL,
  theme = c("clean", "minimal")
)
```

Arguments

x	An `"oda_balance_plot_data"` object from oda_balance_plot_data, or an `"oda_balance_table"` object from oda_balance_table (coerced internally via oda_balance_plot_data; never calls the fitting function).
p_col	Character; which p-value column drives significance colour when coercing from an `oda_balance_table`. One of `"p_mc"` (default), `"p_sidak"`, `"p_bonferroni"`. Ignored when `x` is already `oda_balance_plot_data`.

rank_by	Character; sort order when coercing from oda_balance_table: "abs_ess" (default), "p", "abs_smd".
main	Character; plot title. Default: auto-generated summary.
subtitle	Character; plot subtitle.
show_significance	Logical; annotate significantly imbalanced covariates with a "*" label. Default TRUE.
palette	Named list for color overrides: imbalanced, balanced, unclassified.
theme	Character; "clean" (default, theme_bw base) or "minimal".

Value

A ggplot object.

See Also

oda_balance_plot_data, oda_balance_table

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  group <- c(rep(0L, 20), rep(1L, 20))
  X     <- data.frame(A = c(rep(0L,20), rep(1L,20)),
                      B = rnorm(40))
  bt  <- oda_balance_table(group, X, mcarlo = FALSE, mc_iter = 100L)
  pd  <- oda_balance_plot_data(bt)
  p   <- plot_oda_balance(pd)
  print(p)
}
```

---

plot_oda_balance_effects

Forest plot of ODA covariate balance evidence intervals

Description

Renders a forest plot from an oda_balance_effect_table object. Each covariate is displayed as one row. A thin gray segment shows the chance (null) confidence interval; a thick black segment shows the bootstrap model CI; a point shows the observed ESS/WESS. A vertical dashed line marks the chance upper bound (chance_hi) as a visual reference.

Usage

```
plot_oda_balance_effects(
  x,
  main = NULL,
  subtitle = NULL,
  x_label = NULL,
  xlim = NULL,
  ...
)
```

Arguments

x	An "oda_balance_effect_table" object.
main	Optional character; plot title. Defaults to "ODA Covariate Balance -- Evidence Intervals".
subtitle	Optional character; plot subtitle.
x_label	Optional character; x-axis label. Defaults to the metric label from the data ("ESS (%)" or "WESS (%)").
xlim	Optional numeric(2); x-axis limits. Auto-computed when NULL.
...	Ignored; reserved for future use.

Details

When the object contains multiple analysis scales (e.g., compare_weights = TRUE), the plot is faceted by analysis.

This function does not fit any models. Pass a pre-computed oda_balance_effect_table from oda_balance_effect_table.

Value

A ggplot object.

See Also

oda_balance_effect_table

Examples

```
group <- c(0L, 0L, 0L, 0L, 1L, 1L, 1L, 1L)
X     <- data.frame(v1 = c(1, 2, 3, 4, 5, 6, 7, 8),
                    v2 = c(0L, 1L, 0L, 1L, 0L, 1L, 0L, 1L))
et <- oda_balance_effect_table(group, X,
                               nboot = 50L, chance_iter = 50L,
                               mc_iter = 200L, mc_seed = 1L)
plot_oda_balance_effects(et)
```

`plot_smd_balance` *Plot SMD covariate balance*

Description

Renders a horizontal dot-plot of absolute standardized mean differences (|SMD|) for each covariate. Vertical reference lines at 0.10 (and optionally 0.20) mark conventional balance thresholds. Points are colored by whether |SMD| < 0.10.

Usage

```
plot_smd_balance(
  x,
  ref_010 = TRUE,
  ref_020 = FALSE,
  main = NULL,
  subtitle = NULL,
  palette = NULL,
  theme = c("clean", "minimal")
)
```

Arguments

`x`	A `"smd_balance_table"` object from `smd_balance_table`.
`ref_010`	Logical; draw a dashed reference line at |SMD| = 0.10. Default `TRUE`.
`ref_020`	Logical; draw a dotted reference line at |SMD| = 0.20. Default `FALSE`.
`main`	Character; plot title. Default `"SMD Covariate Balance"`.
`subtitle`	Character; plot subtitle.
`palette`	Named list for color overrides: `imbalanced`, `balanced`, `unclassified`.
`theme`	Character; `"clean"` (default) or `"minimal"`.

Value

A `ggplot` object.

See Also

`smd_balance_table`, `plot_balance_love`

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  group <- c(rep(0L, 20), rep(1L, 20))
  X     <- data.frame(A = c(rep(0L,20), rep(1L,20)),
                      B = rnorm(40))
  smd <- smd_balance_table(group, X)
```

```
  p   <- plot_smd_balance(smd)
  print(p)
}
```

`predict.cta_ort` *Predict method for Locally Optimal Recursive Tree (LORT)*

Description

Routes each row of `newdata` down the composite LORT by recursively applying each node's `cta_tree` model via `cta_assign_endpoints`.

Usage

```
## S3 method for class 'cta_ort'
predict(
  object,
  newdata,
  type = c("class", "stratum", "path", "all"),
  missing_action = c("na", "majority"),
  ...
)
```

Arguments

`object`	A `cta_ort` from `cta_fit(..., recursive = TRUE)`.
`newdata`	Data frame or matrix matching the training X column layout.
`type`	Character; one of `"class"` (default), `"stratum"`, `"path"`, or `"all"`.
`missing_action`	Passed to each node-level `cta_assign_endpoints` call. `"na"` (default): observations with a missing split attribute return `NA`. `"majority"`: route to the majority-class child.
`...`	Unused.

Value

For `type = "class"`: integer vector of predicted class labels (length `nrow(newdata)`). For `type = "stratum"`: integer stratum_id vector. For `type = "path"`: character path vector. For `type = "all"`: data.frame with columns `predicted_class`, `stratum_id`, `path`, `prop_class1`, `stop_reason`.

Note

`predict.cta_ort` is a legacy compatibility name; the class `cta_ort` and all `*.cta_ort` methods refer to the implemented LORT method. New docs and APIs should use LORT terminology.

predict.cta_tree *Classify new observations using a CTA tree*

Description

Applies a fitted `cta_tree` to new data by routing each observation through the tree until it reaches a leaf node.

Usage

```
## S3 method for class 'cta_tree'
predict(object, newdata,
  missing_action = c("majority", "na"), ...)
```

Arguments

`object`	A `cta_tree` from `oda_cta_fit`.
`newdata`	Data frame or matrix with the same columns as training X.
`missing_action`	How to handle observations whose split attribute is missing on their traversal path. `"majority"` (default) routes the observation to the current node's majority class. `"na"` returns `NA_integer_` for that observation (canonical path-local missingness).
`...`	Unused.

Value

Integer vector of predicted class labels, length `nrow(newdata)`. When `missing_action = "na"`, observations missing a split attribute on their path receive `NA_integer_`.

predict.oda_fit *Predict class labels from a fitted ODA model*

Description

Applies the fitted ODA rule to new attribute values, returning predicted class labels in the original label space. Missing values and miss-coded values return `NA_integer_`. Failed fits return all `NA_integer_` with a warning.

Usage

```
## S3 method for class 'oda_fit'
predict(object, newdata, ...)
```

Arguments

object	An oda_fit object from oda_fit.
newdata	Numeric vector or single-column data frame of attribute values.
...	Unused.

Value

Integer vector of predicted class labels, length length(newdata) or nrow(newdata).

---

predict.sda_fit *Predict from an SDA procedure result*

Description

Applies the learned selected-step sequence to newdata. For each observation, steps are applied in order; the first step whose rule classifies the observation is authoritative. Observations not classified by any step are returned as NA (resolved = FALSE).

Usage

```
## S3 method for class 'sda_fit'
predict(object, newdata, type = "class", ...)
```

Arguments

object	A sda_fit object.
newdata	Data frame or matrix. Must contain columns with names matching all selected attributes in object$selected_attributes. Extra columns are ignored (wide-newdata is supported).
type	Output type. One of "class" (default), "stage", "rule", or "trace". "propensity" and "weights" are reserved for SDA-5 and error in SDA-1.
...	Unused.

Details

This is sequential selected-step application - it follows the learned SDA structure, not a re-scan of X. It does not select a "first attribute" from newdata; it replays object$steps[[1]], object$steps[[2]], ... in the order established at fit time.

Value

"class" Integer vector of predicted class labels; NA for unresolved observations.

"stage" Integer vector of step_id at which each observation was classified; NA for unresolved.

"rule" Character vector of the selected attribute name at the classifying step; NA for unresolved.

"trace" Data frame with one row per observation x step: obs_id, step_id, attribute, classified, class_pred.

print.auto_sda_plan *Print an auto_sda_plan object*

Description

Print an auto_sda_plan object

Usage

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

Arguments

x	An auto_sda_plan object.
...	Unused.

Value

Invisibly returns x. Called primarily for its side effect of printing a human-readable summary of the SDA plan to the console.

print.cta_family *Print a CTA descendant family*

Description

Calls summary.cta_family and prints the result.

Usage

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

Arguments

x	A cta_family from cta_descendant_family.
...	Passed to print.cta_family_summary.

Value

invisible(x).

See Also

summary.cta_family, cta_family_table

print.cta_family_summary *Print a CTA family summary*

Description

Compact display of the `cta_family_summary` object returned by `summary.cta_family`.

Usage

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

Arguments

`x`	A `cta_family_summary` object.
`...`	Unused; included for S3 compatibility.

Value

`invisible(x)`.

See Also

`summary.cta_family`, `cta_family_table`

print.cta_ort *Print method for Locally Optimal Recursive Tree (LORT)*

Description

Print method for Locally Optimal Recursive Tree (LORT)

Usage

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

Arguments

`x`	A `cta_ort` object.
`...`	Unused.

Value

`invisible(x)`.

Note

`print.cta_ort` is a legacy compatibility name for the LORT method. The class `cta_ort` and all `*.cta_ort` methods refer to LORT; do not introduce new bare-`ort` public names.

`print.cta_ort_summary` *Print method for cta_ort_summary*

Description

Print method for cta_ort_summary

Usage

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

Arguments

`x`	A `cta_ort_summary` object.
`...`	Unused.

Value

`invisible(x)`.

`print.cta_tree` *Print a CTA tree in MegaODA node table format*

Description

Displays each split node with its attribute, depth, n, p-value, ESS, LOO status, and rule string, followed by the node confusion matrix.

Usage

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

Arguments

`x`	A `cta_tree` from `oda_cta_fit`.
`...`	Unused.

Value

Invisibly returns `x`.

print.cta_tree_summary *Print a CTA tree summary*

Description

Compact display of a `cta_tree_summary` object produced by `summary.cta_tree`.

Usage

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

Arguments

`x`	A `cta_tree_summary` from `summary.cta_tree`.
`...`	Unused.

Value

Invisibly returns `x`.

See Also

`summary.cta_tree`

print.oda_fit *Print a fitted ODA model*

Description

Compact display of rule, ESS/Mean PAC, and available MC/LOO metadata. Does not recompute any quantities.

Usage

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

Arguments

`x`	An `oda_fit` object.
`...`	Unused.

Value

Invisibly returns `x`.

print.oda_fit_summary *Print an ODA fit summary*

Description

Print an ODA fit summary

Usage

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

Arguments

x	An oda_fit_summary from summary.oda_fit.
...	Unused.

Value

Invisibly returns x.

print.sda_anchor *Print an* sda_anchor

Description

Prints a concise summary: anchor type, number of stages, selected attributes, implementation status. Does not claim SORT or GORT are implemented.

Usage

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

Arguments

x	An sda_anchor object.
...	Ignored.

Value

x invisibly.

print.sda_fit *Print an sda_fit object*

Description

Print an sda_fit object

Usage

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

Arguments

x	An sda_fit object.
...	Unused.

Value

Invisibly returns x. Called primarily for its side effect of printing a human-readable summary of the SDA fit to the console.

print.sda_fit_summary *Print an sda_fit_summary object*

Description

Print an sda_fit_summary object

Usage

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

Arguments

x	An sda_fit_summary object.
...	Unused.

Value

Invisibly returns x. Called primarily for its side effect of printing a human-readable summary of the SDA results to the console.

propensity_ess_balance

Propensity-weighted ESS balance diagnostic

Description

For each covariate in X_balance, computes the unweighted and propensity-weighted ODA ESS association with group, the delta ESS (weighted minus unweighted), and a bootstrap confidence interval on the delta.

Usage

```
propensity_ess_balance(
  propensity_fit,
  group,
  X_balance,
  x_prop = NULL,
  newdata = NULL,
  target_class = NULL,
  adjusted = TRUE,
  n_boot = 500L,
  boot_alpha = 0.05,
  seed = NULL
)
```

Arguments

propensity_fit	An oda_fit, cta_tree, or cta_ort (LORT) object trained with group as the class variable.
group	Integer (or coercible) binary group/treatment vector of length n.
X_balance	Data frame of baseline covariates. Must have n rows.
x_prop	Numeric vector of length n. Required when propensity_fit is an oda_fit; assigns each observation to an ODA stratum.
newdata	Data frame with n rows. Required when propensity_fit is a cta_tree or cta_ort; used to assign observations to endpoints/strata. Must contain the columns used to fit the propensity model.
target_class	Integer. Passed to cta_propensity_weights / lort_propensity_weights for models with more than two classes.
adjusted	Logical. If TRUE (default), uses adjusted propensity weights (one-hypothetical-misclassification correction for zero-cell strata).
n_boot	Integer. Number of bootstrap resamples. Default 500L.
boot_alpha	Numeric in (0, 1). CI level is 1 - boot_alpha. Default 0.05 gives a 95% CI.
seed	Integer or NULL. Passed to set.seed() before bootstrap resampling for reproducibility.

Details

If propensity weighting controls confounding, the weighted ODA ESS should move toward 0 (the chance/null boundary). A negative `delta_ess` means the ODA association was attenuated by weighting (improved balance). `crosses_null = TRUE` means the bootstrap CI for the delta includes 0.

LORT (`cta_ort`) propensity models are not supported in this version. Use a single `cta_tree` via `cta_fit()` instead.

The bootstrap uses plug-in propensity weights: weights computed on the full data are reused in each resample rather than re-estimating the propensity model. This is appropriate for assessing sampling variability in the balance diagnostic given a fixed propensity model.

`oda_balance_table` is called with `mcarlo = FALSE`; MC p-values are not computed during bootstrap iterations.

Value

A `data.frame` of class `c("propensity_ess_balance", "data.frame")` with one row per covariate and columns:

variable Covariate name.

n Effective sample size from the unweighted ODA fit.

unweighted_ess Unweighted ODA ESS (%).

weighted_ess Propensity-weighted ODA ESS / WESS (%).

delta_ess `weighted_ess - unweighted_ess`. Negative values indicate attenuation (improved balance).

boot_low Lower bound of the bootstrap CI on `delta_ess`.

boot_high Upper bound of the bootstrap CI on `delta_ess`.

crosses_null Logical. `TRUE` when the CI includes 0.

status `"ok"`, `"inadmissible_unweighted"`, `"inadmissible_weighted"`, or `"inadmissible_both"`.

See Also

`oda_propensity_weights`, `cta_propensity_weights`, `oda_balance_table`

Examples

```
set.seed(1L)
n     <- 80L
group <- c(rep(0L, 40L), rep(1L, 40L))
x_pv  <- c(rnorm(40, 0), rnorm(40, 3))
prop_fit <- oda_fit(x = x_pv, y = group)
X_bal <- data.frame(age   = c(rnorm(40, 45), rnorm(40, 55)),
                    score = rnorm(80))

peb <- propensity_ess_balance(prop_fit, group, X_bal,
                              x_prop = x_pv, n_boot = 50L, seed = 1L)
print(peb[, c("variable", "unweighted_ess", "weighted_ess",
              "delta_ess", "crosses_null")])
```

sda_anchor *Construct an* sda_anchor *object*

Description

Low-level constructor. Prefer as_sda_anchor when converting from an sda_fit. Use this constructor when building an explicit / manual anchor from pre-specified fields (e.g. from a published attribute ordering).

Usage

```
sda_anchor(
  anchor_type = "explicit",
  source_class = NULL,
  source_call = NULL,
  group_levels = NULL,
  selected_attributes,
  candidate_universe = NULL,
  stage_table,
  branch_candidate_map = NULL,
  removal_history = NULL,
  weights_used = FALSE,
  weight_summary = NULL,
  loo_mode = NULL,
  mc_iter = NULL,
  mc_seed = NULL,
  mindenom = NULL,
  alpha = NULL,
  stop_reason = NA_character_,
  reproducibility_notes = character(0),
  canon_notes = character(0),
  task_hook = .sda_anchor_task_hook()
)
```

Arguments

anchor_type	Character scalar: "sda_fit" for anchors derived from a fitted sda_fit object, or "explicit" for manually-declared anchors.
source_class	Character vector: class of the source object, or NULL.
source_call	Language object or NULL: the call used to produce the source object.
group_levels	Integer vector of class/group levels, or NULL.
selected_attributes	Non-empty character vector of selected attribute names in stage order.
candidate_universe	Character vector of all attributes evaluated, or NULL.

stage_table	Data frame with at least columns stage_id and attribute.
branch_candidate_map	Named list for SORT branch-level candidates, or NULL (reserved for future SORT).
removal_history	List of per-step removal records, or NULL.
weights_used	Logical. FALSE unless weighted SDA is available.
weight_summary	List or NULL.
loo_mode	Character scalar or NULL: LOO mode string from SDA settings.
mc_iter	Integer or NULL: MC iterations from SDA settings.
mc_seed	Integer or NULL: RNG seed from SDA settings.
mindenom	Integer or NULL: MINDENOM from SDA settings.
alpha	Numeric or NULL: significance threshold from SDA settings.
stop_reason	Character scalar or NA: SDA stop reason.
reproducibility_notes	Character vector.
canon_notes	Character vector.
task_hook	List. Machine-readable metadata for future agent/pipeline consumers. Defaults to the standard anchor task hook (see ?sda_anchor).

Details

An sda_anchor is a typed structural object that carries SDA selection history for future SORT (staged CTA) workflows. It is not a fitting object and does not estimate propensity scores.

What an SDA anchor is not:

- It is not a propensity-score estimator. SDA produces stage order and selected attributes, not a propensity stratification.
- It is not an implementation of SORT or GORT. Both remain future reserved workflows.
- Explicit / manual anchors are not SDA-derived and must be labeled anchor_type = "explicit".

Task hook: The default task_hook marks implementation_status = "anchor_only_no_sort", lists prohibited_downstream = c("propensity_weighting", "fraud_demo"), and requires human review.

Value

Object of class c("sda_anchor", "list").

See Also

as_sda_anchor, validate_sda_anchor, sda_fit

sda_candidate_table *Return the candidate table from one or all SDA steps*

Description

The candidate table is the primary auditability record: one row per candidate attribute evaluated at a step, showing ESS, p-value, eligibility, and why a candidate was rejected or selected.

Usage

```
sda_candidate_table(fit, step = NULL)
```

Arguments

fit	An sda_fit object.
step	Integer step index, or NULL (default) to return a list of candidate tables for all steps.

Value

If step is an integer: the candidate table data frame for that step (with an added step_id column). If step = NULL: a named list of candidate table data frames, one per step.

sda_fit *Run a Structural Decomposition Analysis (SDA) procedure*

Description

Executes staged attribute-set identification on binary class data. Traverses the attribute space by class, selecting the best eligible attribute at each step, removing correctly classified observations, and repeating on the unresolved sample until a stopping condition is met. The result identifies which attributes to pass to downstream CTA or MDSA.

Usage

```
sda_fit(
  X,
  y,
  mode = c("novometric_min_d", "unioda_max_ess"),
  attr_types = NULL,
  weights = NULL,
  mindenom = NULL,
  mc_iter = 5000L,
  mc_seed = 42L,
  mc_stop = 99.9,
```

```
  mc_stopup = NA,
  alpha = 0.05,
  loo = "off",
  max_steps = NULL,
  min_n = NULL,
  min_class_n = NULL,
  remove_correct = TRUE,
  collinearity = c("skip", "warn", "allow"),
  verbose = FALSE
)
```

Arguments

X	Data frame of candidate attribute columns.
y	Integer class vector. Must have exactly two distinct values.
mode	SDA mode. "novometric_min_d" (MPE-canon; per-attribute MDSA via cta_descendant_family(); requires mindenom) or "unioda_max_ess" (iterative UniODA; default for SDA-1). Must be declared explicitly; do not mix modes.
attr_types	Named character vector of attribute types ("ordered", "categorical", "binary"), or NULL for auto-detection. Names must match column names of X.
weights	Case weights. Must be NULL in SDA-1; weighted SDA is not yet implemented and will error if non-NULL.
mindenom	Integer MINDENOM (novometric mode only; ignored with warning in unioda_max_ess mode).
mc_iter	Maximum Monte Carlo iterations per attribute fit. Default 5000L.
mc_seed	RNG seed set once before the SDA run. Default 42L.
mc_stop	Lower-tail early-stop confidence (percent). Default 99.9.
mc_stopup	Upper-tail early-stop confidence (percent). Default NA (disabled; matches MegaODA behavior).
alpha	Significance threshold for p-value gate. Default 0.05.
loo	LOO mode passed to oda_fit(). Default "off".
max_steps	Maximum number of SDA steps (safety cap). Default NULL (no cap beyond candidate exhaustion).
min_n	Minimum working-sample size. If unresolved n drops below this, stop with "min_n". Default NULL.
min_class_n	Minimum per-class count. Stop with "min_class_n" if either class falls below this. Default NULL.
remove_correct	Logical. If TRUE (canonical SDA), remove correctly classified observations after each step. If FALSE, diagnostic dry-run: step logic executes but working sample is not modified. Default TRUE.
collinearity	How to handle duplicate candidate columns: "skip" (silent), "warn", or "allow". Default "skip".
verbose	Logical. Emit [SDA] progress messages. Default FALSE.

Value

Object of class `c("sda_fit", "odacore_sda")`.

---

sda_selected_attributes *Return the selected attribute names from an SDA procedure result*

Description

Returns the names of attributes selected across all SDA steps, in step order. This is the constrained candidate set to pass to MDSA/CTA.

Usage

```
sda_selected_attributes(fit)
```

Arguments

`fit`	An `sda_fit` object.

Value

Character vector of selected attribute names (length = number of completed SDA steps). Empty character vector if no steps completed.

---

sda_step_table *Return a summary table of SDA steps*

Description

One row per completed SDA step. Columns cover the key auditability fields needed to review what was selected, why, and how the working sample changed.

Usage

```
sda_step_table(fit)
```

Arguments

`fit`	An `sda_fit` object.

Value

Data frame with columns: `step_id`, `attribute`, `n_in`, `n_correct`, `n_incorrect`, `ess`, `d`, `p_mc`, `mindenom`.

sda_to_cta_data *Prepare X and y for CTA using SDA-selected attributes*

Description

Returns a named list `list(X_cta, y_cta)` where `X_cta` contains only the SDA-selected attribute columns and `y_cta` is the full outcome vector (all observations, not just unresolved).

Usage

```
sda_to_cta_data(fit, X, y)
```

Arguments

`fit`	An `sda_fit` object.
`X`	Data frame of predictors (all observations).
`y`	Integer class vector (all observations).

Details

This matches the Path B workflow from MPE Chapter 12: SDA identifies the attribute subset; MDSA/CTA receives the full sample with a constrained candidate frame. SDA resolution does not restrict which observations CTA sees.

Value

Named list with elements `X_cta` (data frame, selected columns only) and `y_cta` (integer vector, full length).

smd_balance_table *Conventional SMD companion table for covariate balance*

Description

Computes standardized mean differences (SMD) between two groups for each covariate in `X`. Returns one row per covariate with group means, standard deviations, raw and absolute SMD, and conventional balance thresholds.

Usage

```
smd_balance_table(group, X, w = NULL)
```

Arguments

group	Integer (or coercible) binary group indicator. Must have exactly two distinct non-missing values.
X	Data frame of baseline covariate columns.
w	Optional numeric case-weight vector. When supplied, weighted group means (wmean_0, wmean_1) and weighted SMD (wsmd) are added. Weighted SMD uses the unweighted pooled SD as the standardizer (Rubin simplification).

Details

SMD is a conventional companion diagnostic, not the oda balance objective. The primary oda balance assessment uses oda_balance_table. This function is intended for comparison with non-ODA balance reports.

No p-values are computed. SMD is a descriptive statistic. For a variable with zero within-group variance in both groups, smd is NA.

Value

A data.frame of class c("smd_balance_table", "data.frame") with one row per covariate and columns: attribute, n_group_0, n_group_1, mean_0, sd_0, mean_1, sd_1, smd, abs_smd, balanced_020 (abs_smd < 0.20), balanced_010 (abs_smd < 0.10), wmean_0, wmean_1, wsmd, wabs_smd, wbalanced_020, wbalanced_010 (weighted variants; NA when w = NULL).

See Also

oda_balance_table, oda_balance_plot_data

Examples

```
group <- c(rep(0L, 30), rep(1L, 30))
X     <- data.frame(age = c(rep(45, 30), rep(55, 30)),
                    score = rnorm(60, 50, 10))
smd_balance_table(group, X)
```

---

summary.cta_family *Summarise a CTA descendant family*

Description

Returns a structured S3 object summarising the CTA descendant family. All values are read from stored fields - no refitting or recomputation is performed.

Usage

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

Arguments

object	A cta_family from cta_descendant_family.
...	Unused; included for S3 compatibility.

Value

summary.cta_family returns a list of class c("cta_family_summary", "list") with fields:

n_members Integer number of family members.

min_d_idx Integer index of the feasible member with minimum D; NA_integer_ if no feasible member exists.

terminated Logical; always TRUE for a completed chain.

termination_reason Character: one of "no_tree", "max_steps", "no_next_mindenom".

has_weights Logical; TRUE when any family member used case weights.

table A data.frame from cta_family_table.

See Also

cta_descendant_family, cta_family_table

Examples

```
data(mtcars)
X <- mtcars[, c("cyl", "disp", "hp", "wt")]
y <- as.integer(mtcars$am)
fam <- suppressMessages(
  cta_descendant_family(X, y, start_mindenom = 1L, mc_iter = 200L,
                        mc_seed = 42L, loo = "off")
)
s <- summary(fam)
print(s)
print(fam)
```

---

summary.cta_ort *Summary method for Locally Optimal Recursive Tree (LORT)*

Description

Returns a structured list of class "cta_ort_summary" capturing tree-level metadata for the composite LORT.

Usage

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

Arguments

object	A cta_ort object.
...	Unused.

Value

A list of class "cta_ort_summary".

Note

summary.cta_ort is a legacy compatibility name for the LORT method. See print.cta_ort for the naming note.

---

summary.cta_tree *Summarize a fitted CTA tree*

Description

Returns a structured list with class "cta_tree_summary" capturing tree-level metadata. All fields are read directly from stored objects; no refitting or prediction is performed.

Usage

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

Arguments

object	A cta_tree from oda_cta_fit.
...	Unused.

Value

A list of class "cta_tree_summary" with fields:

status Character: "valid_tree", "stump", or "no_tree".

no_tree Logical; TRUE for leaf-only fits.

root_attribute Character attribute name at the root split; NA_character_ for no-tree fits.

n_nodes Total number of nodes including leaves.

n_splits Number of non-leaf (split) nodes.

n_leaves Number of terminal leaf endpoints (= strata).

strata Alias for n_leaves; NA_integer_ for no-tree fits.

overall_ess WESS when weights are active, ESS otherwise; NA_real_ when absent.

d D statistic (NA_real_ for no-tree or ESS ≤ 0).

`min_terminal_denom` Smallest leaf `n_obs`; `NA_integer_` for no-tree fits.

`endpoint_denominators` Named integer vector of leaf `n_obs`; `integer(0)` for no-tree fits.

`has_weights` Logical; `TRUE` when case weights are active.

`mindenom` MINDENOM used when fitting.

`alpha_split` Significance threshold used when fitting.

`prune_alpha` Pruning threshold used when fitting.

`loo` LOO mode string used when fitting.

See Also

`oda_cta_fit`, `cta_node_table`, `cta_strata`, `cta_d_stat`, `print.cta_tree_summary`

Examples

```
data(mtcars)
X    <- mtcars[, c("cyl", "disp", "hp", "wt")]
y    <- as.integer(mtcars$am)
tree <- oda_cta_fit(X, y, mindenom = 5L, mc_iter = 500L, mc_seed = 42L)
s    <- summary(tree)
print(s)
```

`summary.oda_fit` *Summarize a fitted ODA model*

Description

Returns a structured list with class `"oda_fit_summary"` exposing train and LOO sections. Does not recompute any quantities; fields absent from the fit appear as `NA` or `NULL`.

Usage

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

Arguments

`object`	An `oda_fit` object.
`...`	Unused.

Value

A list of class `"oda_fit_summary"`.

summary.sda_anchor *Summarise an* sda_anchor

Description

Returns a named list with the key structural fields needed to audit the anchor or pass it to future SORT / staged-CTA pipelines.

Usage

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

Arguments

object	An sda_anchor object.
...	Ignored.

Value

Named list with fields: anchor_type, n_stages, selected_attributes, candidate_universe, group_levels, stop_reason, weights_used, loo_mode, mc_iter, mc_seed, mindenom, alpha, stage_table, canon_notes, implementation_status, safety_notes.

summary.sda_fit *Summarise an sda_fit object*

Description

Summarise an sda_fit object

Usage

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

Arguments

object	An sda_fit object.
...	Unused.

Value

An object of class "sda_fit_summary" (a list) with elements: mode, n_initial, n_final_unresolved, stop_reason, selected_attributes, step_table (data.frame), and settings.

validate_sda_anchor *Validate an* sda_anchor *object*

Description

Checks that all required fields are present and well-formed. Errors clearly on any violation so that downstream SORT / staged-CTA code can rely on the contract.

Usage

```
validate_sda_anchor(anchor, strict = TRUE)
```

Arguments

anchor	An object to validate.
strict	Logical (default TRUE). When TRUE, errors on any violation. When FALSE, returns a character vector of issue messages (empty if valid).

Value

anchor invisibly (on success).

Index

∗ **datasets**
    cta_demo, 18
    myeloma, 44
.lort_parent_maps, 4

as_confusion_matrix, 4, *15*
as_cta_candidates, 5
as_sda_anchor, 6, *108, 109*
auto_sda_plan, 7

colorRampPalette, *33*
cta_assign_endpoints, 8, *29, 30*, *97*
cta_balance_effect_summary, 10, *85, 86*
cta_balance_plot_data, 12, *14, 15*, *84*
cta_balance_table, *11, 12*, 13, *13*, *84*
cta_confusion_matrix, 15
cta_confusion_table, *4, 5*, *15*, 16, *20*
cta_d_stat, 17, *19*, *117*
cta_demo, 18
cta_descendant_family, *5*, 18, *24–26*, *39*, *87*, *100*, *115*
cta_endpoint_counts, 19, *21–23*, *36, 37*
cta_endpoint_denominators, 20, *22, 23*, *28*, *38*
cta_endpoint_summary, *8–10*, *19, 20*, 21, *23*, *30*, *37*
cta_endpoint_table, *14*, *16*, *20, 21*, 22, *22*, *29*, *38*
cta_family_table, 24, *100, 101*, *115*
cta_fit, *5*, *11*, *13–15*, *18*, 25, *31, 32*, *40*, *44*, *57*, *88, 89*
cta_min_terminal_denom, *17*, *19*, *21*, 27, *38*
cta_node_table, *14*, *16*, *22, 23*, *26*, 28, *58*, *117*
cta_observation_weights, 29
cta_ort_node_table, 31, *40*
cta_plot_data, *12, 13*, 32, *81, 82*, *88, 89*
cta_propensity_weights, *8*, *10*, *20*, *29, 30*, 34, *37*, *43*, *72*, *106, 107*
cta_staging_table, *19–22*, *26*, *32, 33*, 36, *36*, *81, 82*
cta_strata, *17*, *19*, *21–23*, *28*, 38, *117*

ggplot, *83, 84*, *87*, *89*, *92*, *94*, *96*
ggsave, *87*, *89*, *93*

lort_fit, *26*, 39, *41–43*, *90*, *92, 93*
lort_index_path, 41, *42*, *91*
lort_local_tree, 41, *41, 42*, *91*
lort_path_table, *41*, 42, *91*
lort_propensity_weights, 43, *72*, *106*

myeloma, 44

novo_boot_ci, *4, 5*, 45

oda_balance_effect_table, *10*, 48, *94, 95*
oda_balance_plot_data, 50, *53*, *93, 94*, *114*
oda_balance_table, *15*, *50, 51*, 52, *93, 94*, *107*, *114*
oda_best_ordered_multiclass_partition, 54
oda_clean_missing_codes, 55, *64*, *73*
oda_confusion, 55
oda_confusion_binary, 56
oda_confusion_multiclass, 56
oda_cta_fit, *8–10*, *15–17*, *19–23*, *25–30*, *32, 33*, *35–38*, 57, *81, 82*, *88*, *98*, *102*, *116, 117*
oda_d_stat, 59
oda_ess_from_mean, 60
oda_ess_from_meanpac, *47*, 60
oda_fit, *26*, *44*, *49*, *52, 53*, *59*, 61, *64*, *67, 68*, *73, 74*, *76, 77*, *99*
oda_infer_attr_types, 63, *73*
oda_loo_multiclass_ordered, 64
oda_mc_p_value, 65
oda_mean_pac, 66
oda_metrics, 67
oda_multiclass_unioda_core, 67, *74*, *77*

oda_power, 69
oda_predictions, 71
oda_propensity_weights, *43*, 71, *107*
oda_readiness_check, 72
oda_rule_predict, 73
oda_rule_predict_multiclass, 74
oda_sample_size, 74
oda_univariate_core, *56*, *73*, 76
oda_validate_group, *73*, 78
oda_validate_weights, *73*, 78
ort_plot_data, *26*, *40*, 79, *80, 81*

plot.cta_ort, *26*, 79, *79*
plot.cta_tree, *26*, *32, 33*, 81, *81*
plot_balance_love, 83, *96*
plot_cta_balance, 84
plot_cta_balance_effects, *11*, 85
plot_cta_family, 86, *89*, *93*
plot_cta_tree, *84*, *87*, 88, *93*
plot_lort_path, *41, 42*, 90, *92*
plot_lort_tree, *87*, *89*, 91, *91*
plot_oda_balance, 93
plot_oda_balance_effects, *50*, *85*, 94
plot_smd_balance, *83*, 96
predict, *9*
predict.cta_ort, *32*, *40*, 97
predict.cta_tree, *10*, *58*, 98
predict.oda_fit, 98
predict.sda_fit, 99
print.auto_sda_plan, 100
print.cta_family, 100
print.cta_family_summary, *100*, 101
print.cta_ort, *79*, *81*, 101, *116*
print.cta_ort_summary, 102
print.cta_tree, 102
print.cta_tree_summary, 103, *117*
print.novo_boot_ci *(*novo_boot_ci*)*, 45
print.oda_fit, 103
print.oda_fit_summary, 104
print.sda_anchor, 104
print.sda_fit, 105
print.sda_fit_summary, 105
propensity_ess_balance, 106

sda_anchor, *6, 7*, 108
sda_candidate_table, 110
sda_fit, *7*, *109*, 110
sda_selected_attributes, 112
sda_step_table, 112
sda_to_cta_data, 113
set.seed, *46*
smd_balance_table, *50, 51*, *53*, *83*, *96*, 113
summary.cta_family, *24*, *100, 101*, 114
summary.cta_ort, *32*, 115
summary.cta_tree, *16*, *23*, *29*, *103*, 116
summary.oda_fit, *104*, 117
summary.sda_anchor, 118
summary.sda_fit, 118

validate_sda_anchor, *7*, *109*, 119