Package 'CausalState'

August 24, 2026

Type Package

Title Causal Inference in a Longitudinal Transitioning State Environment

Version 0.10.2

Description Implements Sequential Doubly Robust (SDR) and infinite-dimensional Targeted Maximum Likelihood (iTMLE) estimators for longitudinal modified treatment policies in settings with transitioning states, such as ICU, ward, or emergency department care episodes. Treatment is permitted in active states and becomes structurally inapplicable after a state transition (e.g. discharge or death). Supports asymmetric g- and Q-model regularisation, k-fold cross-fitting, and pluggable SuperLearner ensembles. Includes specialised SuperLearner wrappers (SL.tgt.* and SL.tmle_* families) for the iTMLE targeting step, which pass the logit offset as a covariate column to preserve correct subsetting during SuperLearner cross-validation. Methods based on Diaz et al. (2021) <doi:10.1080/01621459.2021.1955691> and Luedtke et al. (2017) <doi:10.48550/arXiv.1705.02459>.

URL https://github.com/sebastiaan-blank/CausalState

BugReports https://github.com/sebastiaan-blank/CausalState/issues

License AGPL-3

Encoding UTF-8

Depends R (>= 4.1.0)

Imports data.table (>= 1.14.0), SuperLearner, origami, glmnet, xgboost, dplyr, tidyr, rlang, magrittr, parallel, stats

Suggests testthat (>= 3.0.0), knitr, rmarkdown, ggplot2, scales, stringr, hal9001, dbarts, mgcv, earth, nnls

Config/testthat/edition 3

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Sebastiaan Blank [aut, cre, cph] (ORCID: <https://orcid.org/0000-0001-9115-2138>)

Maintainer Sebastiaan Blank <sebastiaan.blank@mail.com>

Repository CRAN

Date/Publication 2026-08-24 09:10:08 UTC

Contents

`CausalState-package` *CausalState: Causal Inference in a Longitudinal Transitioning State Environment*

Description

Two sequentially doubly robust estimators for longitudinal modified treatment policies (MTPs) in settings with transitioning states - such as ICU stay, ward admission, or ED episodes - where treatment is structurally inapplicable after a state transition (e.g. discharge or death).

Sequential Doubly Robust estimator - `sdr()`

Implements the LMTP-SDR estimator of Diaz et al. (2021), which extends the SDR construction of Luedtke et al. (2017) to general modified treatment policies via density-ratio weighting. Starting from terminal pseudo-outcomes and working backward in time, the estimator propagates an EIF-corrected pseudo-outcome through a sequence of Q-regressions. The final estimate is the mean of these pseudo-outcomes at the first time point. The estimator is sequentially doubly robust (2^K-robust): consistent whenever, at each time point, either the treatment model g_t or the outcome model Q_t is correctly specified. Under the natural-course policy the density ratios equal one and the recursion collapses algebraically to `mean(Y)` regardless of model quality - a natural-course SDR run therefore cannot serve as a model calibration check.

Infinite-dimensional TMLE - `itmle()`

Implements the cross-validated infinite-dimensional TMLE of Luedtke et al. (2017, Section 5 and Appendix 12), adapted to general MTPs via density-ratio weighting following Diaz et al. (2021). Like SDR, the estimator fits a backward Q-regression; unlike SDR, it then applies an infinite-dimensional fluctuation (targeting step) that solves the efficient influence equation semiparametrically without restricting the fluctuation to a parametric submodel. The cross-validated variant is used throughout to prevent the targeting step from overfitting the Q estimates. iTMLE is also sequentially doubly robust. Unlike SDR, running under the natural-course policy does not collapse to `mean(Y)` and therefore provides a genuine calibration check on the Q models after fluctuation.

Density ratio estimation - `density_ratio()`

Estimates the density ratios needed by both estimators via a flexible classification-based approach. An alternative metalearner based on Wu and Benkeser combines base learners by minimising a log density-ratio loss rather than the default NNLS. Density ratios can be computed once and reused across both estimators and across multiple time horizons.

State transition model and probabilistic mixing

At each time point t, a subject in state (`in_state = 1`, `alive = 1`) faces three mutually exclusive outcomes: remain in state, exit alive (discharge), or die. Four regression components handle this, corresponding directly to the `sl_` arguments of `sdr()`, `itmle()`, and `qreg()`:

`g_remain` (`sl_remain`) Probability of remaining in state at the next time point, given current history. Trained on all subjects currently in state.

`g_death_exit` (`sl_death`) Probability of dying, conditional on having left the state (i.e. among exiters only). Trained on the exit subset – those subjects who leave the state at each time point.

`Q_rem` (`sl_recursive` **at intermediate time points;** `sl_y` **at the terminal time point**) Expected outcome conditional on remaining in state. This is the component propagated backward through the Q-recursion.

`Q_exit` (`sl_y`) Expected outcome conditional on exiting. In most applications this is not estimated but instead fixed to a constant for each exit type (death, discharge) via `absorb_rule()`.

Probabilistic mixture. The backward Q-recursion assembles the expected outcome at time t as a probability-weighted mixture over the three transition branches:

```
Q(t) = g_remain * Q_rem
     + (1 - g_remain) * [ g_death_exit * Q_death
                        + (1 - g_death_exit) * Q_discharge ]
```

where `g_remain` and `g_death_exit` are the model predictions at the subject's current history, and `Q_death` / `Q_discharge` are the outcome values assigned to each exit branch (constants when `absorb_rule()` is used, or `sl_y` predictions otherwise).

Event counts and the death regression. Because `g_death_exit` is trained on the exit subset only, its effective sample size is often much smaller than the full cohort – and within that subset, deaths may be rare.

Before every SuperLearner fit, the code checks whether the training data are sufficient to fit a model (internal function `can_fit_bin`): fitting proceeds only when the training set has at least 30 observations and at least 5 events of each class (deaths and discharges within the exit subset for `g_death_exit`; remainers and exiters for `g_remain`). When either threshold is not met, the estimator substitutes the empirical proportion as a time-constant prediction instead of fitting a SuperLearner. The fallback is recorded in `diagnostics$branch_cal` (the `p_*_const` columns are non-`NA` when a constant was used).

Even above the floor, with few events the SuperLearner ensemble will typically collapse to `SL.mean` or a near-intercept logistic fit. Setting `pool_g_death = TRUE` fits a single `g_death_exit` model across all time points (with time included as a covariate), so the 30/5 thresholds apply to the full follow-up pooled rather than each time point individually. This is the recommended remedy when per-time death counts are sparse but the death hazard is roughly stable over time.

To inspect event counts: `diagnostics$branch_cal` contains one row per fold and time point; `n_tr_exit` is the exit-subset size, and the number of deaths is approximately `n_tr_exit * mean_target` for the `g_death` rows.

When `absorb_rule()` fixes `Q_death` to a constant – the common case – the impact of a poorly fitted `g_death_exit` is limited: misspecification shifts probability mass between the death and discharge branches of the mixture, but the sequentially doubly robust correction via density-ratio weighting partially compensates provided the treatment model is well-specified.

State machinery

`absorb_rule()` defines outcome overrides at absorbing states (e.g. forcing Y = 0 for subjects who die before the end of follow-up).

Custom SuperLearner wrappers

SDR-targeted learners (`SL.tgt.glm()`, `SL.tgt.glmnet()`, `SL.tgt.xgboost()`, etc.) for the Q-regression step, and iTMLE fluctuation learners (`SL.tmle_glm()`, `SL.tmle_glmnet_ridge()`, etc.) that carry the logit offset required for cross-validated TMLE as a data column.

Author(s)

Maintainer: Sebastiaan Blank <sebastiaan.blank@mail.com> (ORCID) [copyright holder]

Authors:

- Sebastiaan Blank <sebastiaan.blank@mail.com> (ORCID) [copyright holder]

References

Diaz I, Williams N, Hoffman KL, Schenck EJ (2021). Nonparametric Causal Effects Based on Longitudinal Modified Treatment Policies. *JASA* 118(542):846-857. doi:10.1080/01621459.2021.1955691.

Luedtke AR, Sofrygin O, van der Laan MJ, Carone M (2017). Sequential Double Robustness in Right-Censored Longitudinal Models. arXiv:1705.02459.

Rotnitzky A, Robins J, Babino L (2017). On the multiply robust estimation of the mean of the g-functional. arXiv:1705.08582.

Wu C, Benkeser D (2024). Nonparametric Efficient Estimation of Marginal Structural Models using Targeted Machine Learning. arXiv:2408.10847.

Williams NT, Diaz I (2023). lmtp: An R package for estimating the causal effects of modified treatment policies. *Observational Studies*.

Bang H, Robins JM (2005). Doubly robust estimation in missing data and causal inference models. *Biometrics* 61(4):962-973.

Haneuse S, Rotnitzky A (2013). Estimation of the effect of interventions that modify the received treatment. *Statistics in Medicine* 32(30):5260-5277.

Diaz Munoz I, van der Laan MJ (2012). Population intervention causal effects based on stochastic interventions. *Biometrics* 68(2):541-549.

See Also

Useful links:

- https://github.com/sebastiaan-blank/CausalState
- Report bugs at https://github.com/sebastiaan-blank/CausalState/issues

---

`absorb_rule` *Define an absorbing-state override rule*

Description

Constructs a rule that overrides Q-model predictions for subjects in a specific absorbing state. Pass one or more rules to the `absorb` argument of `sdr()` or `itmle()`.

Usage

```
absorb_rule(cond, value, branch = c("any", "death", "dc"))
```

Arguments

`cond`	An unquoted expression (evaluated row-wise in the long-format data) that identifies rows to which the rule applies. Variables from the dataset and the special symbols `t` (current time) and `branch` (`"death"` or `"dc"`) are in scope.
`value`	An unquoted expression giving the outcome value to assign when `cond` is `TRUE`. May reference data variables or constants. Scaled and clipped automatically via `scale_info`.
`branch`	Which exit branch the rule applies to: `"any"` (both death and discharge, the default), `"death"`, or `"dc"` (discharge).

Value

A named list with elements `branch`, `cond`, and `value` (the latter two as unevaluated expressions via `base::substitute()`).

Examples

```
# Force outcome to 0 for patients who die at any time
absorb_rule(alive == 0, value = 0, branch = "death")

# Force outcome to 1 for discharged patients when t >= 5
absorb_rule(in_state == 0 & t >= 5, value = 1, branch = "dc")
```

---

branch_cal_summary *Per-time, per-branch calibration summary from an SDR or iTMLE fit*

Description

Aggregates the raw per-fold calibration diagnostics stored in `diagnostics$branch_cal` to one row per time step per branch, weighted by the validation-set size at each fold. Returns a tidy `data.table` suitable for further filtering, plotting, or `data.table::dcast()`.

Usage

```
branch_cal_summary(fit)
```

Arguments

`fit`	Output of `sdr()` or `itmle()`.

Value

A `data.table` with columns `t`, `branch` (one of `"g_remain"`, `"g_death"`, `"q_rem"`, `"q_exit"`), `n_tr`, `tgt_tr`, `pred_tr`, `n_vl`, `tgt_vl`, `pred_vl`, `cal_slope_vl`; and where applicable `brier_vl`, `auc_vl` (binary branches) or `rmse_vl`, `cor_vl` (Gaussian branches).

See Also

`sdr()`, `itmle()`

---

contrast *Compute contrasts between two fitted estimators*

Description

Takes an intervention-arm fit and a reference (from `sdr()` or `itmle()`, or the crude observed mean) and returns the risk difference (RD), risk ratio (RR), and odds ratio (OR) with standard errors derived from the per-subject influence curves.

Usage

```
contrast(
  fit1,
  fit0 = NULL,
  df = NULL,
  id_col = NULL,
  cluster = NULL,
  y_col = NULL
)
```

Arguments

`fit1`	Fitted object (intervention arm): output of `sdr()` or `itmle()`. Must contain `$psi` and `$ic_df` with columns `id` and `ic`.
`fit0`	Fitted object (reference arm), same type as `fit1`, or `NULL` to use the crude observed mean as the reference (requires `df` and `y_col`).
`df`	Optional long-format data frame. Required when `cluster` is specified, or when `fit0 = NULL`.
`id_col`	Name of the subject-id column in `df`. Defaults to the value stored in `fit1$settings`.
`cluster`	`NULL` (default, IID standard errors), or a single character string naming a column in `df` for cluster-robust SEs.
`y_col`	Name of the outcome column in `df`. Required when `fit0 = NULL`; ignored otherwise.

Details

All standard errors use the delta method on the efficient influence curves:

- RD: $IC_{RD,i} = IC_{1,i} - IC_{0,i}$; Wald CI on the natural scale.
- RR: $IC_{\log RR,i} = IC_{1,i}/\psi_1 - IC_{0,i}/\psi_0$; SE and 95\
- OR: $IC_{\log OR,i} = IC_{1,i}/(\psi_1(1-\psi_1)) - IC_{0,i}/(\psi_0(1-\psi_0))$; SE and 95\

RR and OR are omitted for Gaussian outcomes.

When `fit0 = NULL` and `y_col` is supplied, the reference is the crude observed mean of `y_col` (last non-missing value per subject). The influence curve for the observed mean is $IC_{0,i} = Y_i - \bar{Y}$. This gives a simple descriptive contrast against the raw data rather than a causal comparison between two counterfactual estimates.

Value

A list of class `"CausalState_contrast"` with:

`psi1`, `psi0` Point estimates.

`obs_ref` Logical; `TRUE` when the reference is the observed mean rather than a second estimator fit.

`RD`, `se_RD`, `ci_RD` Risk difference and 95 pct Wald CI.

`RR`, `se_log_RR`, `ci_RR` Risk ratio, SE on log scale, and 95 pct CI (exponentiated). `NULL` for Gaussian outcomes.

`OR`, `se_log_OR`, `ci_OR` Odds ratio, SE on log scale, and 95 pct CI (exponentiated). `NULL` for Gaussian outcomes.

`n` Number of matched subjects.

`table` Summary data frame printed by default.

See Also

`sdr()`, `itmle()`

Examples

```
library(SuperLearner)

sim_ex <- function(n = 2000L, tmax = 5L) {
  set.seed(42L)
  rows <- vector("list", n)
  for (i in seq_len(n)) {
    age <- round(rnorm(1, 65, 10)); L1 <- rnorm(1)
    pat <- list()
    for (t in seq_len(tmax)) {
      A     <- rbinom(1, 1, plogis(0.3 * L1 - 0.4))
      u     <- runif(1)
      p_die <- plogis(-4.0 + 0.2 * L1 - 0.1 * age / 10)
      p_dc  <- plogis(-2.5 + 0.5 * A)
      if (u < p_die) {
        alive <- 0L; in_state <- 0L
      } else if (u < p_die + p_dc) {
        alive <- 1L; in_state <- 0L
      } else {
        alive <- 1L; in_state <- 1L
      }
      Y <- rbinom(1, 1, plogis(-0.5 + 0.4 * A - 0.2 * L1))
      pat[[length(pat) + 1L]] <- data.frame(
        id = i, time = t, age = age, L1 = L1,
        A = A, alive = alive, in_state = in_state, Y = Y
      )
      if (in_state == 0L) break
      if (t < tmax) L1 <- L1 + rnorm(1, -0.1 * A, 0.3)
    }
    rows[[i]] <- do.call(rbind, pat)
  }
  do.call(rbind, rows)
}
df <- sim_ex()
sl_lib <- c("SL.mean", "SL.glm")

policy_nat <- function(D_block, t, a_names) D_block[, ..a_names, drop = FALSE]
policy_sft <- function(D_block, t, a_names) {
  out <- D_block[, ..a_names, drop = FALSE]
  out[[a_names[1]]] <- pmin(D_block[[a_names[1]]] + 0.3, 1)
  out
}
```

```
dr_args <- list(
  df = df, a_names = "A", tmax = 5L, baseline = "age", tv_names = "L1",
  sl_g = sl_lib, k = 1L, inner_v = 3L, v = 3L, seed = 1L,
  id = "id", time = "time"
)
wr_nat <- do.call(density_ratio, c(dr_args, list(policy_spec_fun = policy_nat)))
wr_sft <- do.call(density_ratio, c(dr_args, list(policy_spec_fun = policy_sft)))

sdr_args <- list(
  df = df, tmax = 5L, id = "id", time = "time",
  alive = "alive", in_state = "in_state", y = "Y",
  baseline = "age", tv_names = "L1", a_names = "A",
  sl_remain = sl_lib, sl_death = sl_lib,
  sl_recursive = sl_lib, sl_y = sl_lib,
  k = 1L, inner_v = 3L, parallel = FALSE, seed = 1L
)
res_nat <- do.call(sdr, c(sdr_args,
  list(weight_object = wr_nat, policy_spec_fun = policy_nat)))
res_sft <- do.call(sdr, c(sdr_args,
  list(weight_object = wr_sft, policy_spec_fun = policy_sft)))

# Two-estimator contrast (causal RD/RR/OR)
ctr <- contrast(res_sft, res_nat)
ctr$RD; ctr$ci_RD; ctr$RR; ctr$OR

# Observed-mean reference (descriptive)
ctr_obs <- contrast(res_sft, df = df, y_col = "Y")
ctr_obs$table
```

`density_ratio` *Estimate density ratios for a modified treatment policy*

Description

Fits per-time-point treatment models and computes the instantaneous density ratio $r_t = d\tilde{P}(A_t|H_t)/dP(A_t|H_t)$ comparing the modified treatment policy (MTP) to the natural course. The output is passed directly to `sdr()` or `itmle()` as the `weight_object` argument.

Usage

```
density_ratio(
  df,
  a_names,
  tmax,
  baseline = NULL,
  tv_names = character(0),
```

```
  no_lag_vars = character(0),
  policy_names = character(0),
  sl_g,
  k = 2,
  inner_v = 10L,
  cluster = NULL,
  v = 5L,
  seed = 1L,
  policy_spec_fun,
  id = "id",
  time = "trial_time",
  bounds = 1e-05,
  dr_sl = FALSE,
  drop_small_cluster_splits = TRUE,
  parallel_t = FALSE,
  t_workers = NULL,
  fold_workers = NULL,
  sl_workers = NULL,
  verbose = TRUE
)
```

Arguments

df	A data.frame in long format (one row per subject per time point).
a_names	Character vector of treatment variable names. Can contain one or more variables for joint multi-treatment policies (e.g. c("A1", "A2")). The joint density ratio for the full treatment vector is estimated at each time point. Pass the same a_names to the downstream estimators (sdr(), itmle(), qreg()).
tmax	Integer. Maximum follow-up time (number of time points).
baseline	Character vector of baseline covariate names. Default NULL (no baseline covariates).
tv_names	Character vector of time-varying covariate names. Default character(0).
no_lag_vars	Character vector of variables that should not be lagged.
policy_names	Character vector of column names holding shifted treatment values under the MTP.
sl_g	SuperLearner library for treatment models. Required.
k	Integer. Number of lags to include. Default 2.
inner_v	Integer. Inner cross-validation folds for SuperLearner. Default 10L.
cluster	Character. Cluster variable name for clustered fold assignment. NULL uses independent subject folds.
v	Integer. Number of outer cross-fitting folds. Default 5L.
seed	Integer random seed. Default 1L.
policy_spec_fun	A function (D_block, t, a_names) returning a data.table with shifted treatment values for time t. Used when the MTP cannot be pre-computed as static columns.

`id` Character. Subject identifier column name. Default `"id"`.

`time` Character. Time column name. Default `"trial_time"`.

`bounds` Numeric. Probability floor for clipping predicted probabilities before conversion to density ratios (standard pathway only). Default `1e-5`.

`dr_sl` Logical. Selects the metalearner used to combine base learners. `FALSE` (default): standard SuperLearner with NNLS metalearner – each base learner outputs a propensity score (probability scale) and the metalearner combines them with non-negative least squares; the combined probability is then converted to a density ratio via `p / (1 - p)`. `TRUE`: the Wu-Benkeser metalearner (method.WB_dr) – combines base learners directly in density-ratio space by minimising a log density-ratio loss rather than a squared-error loss on the probability scale. This can give better-calibrated density ratios when the propensity is far from 0.5 and extreme ratios are a concern. When `dr_sl = TRUE` the SuperLearner object returns density ratios directly rather than probabilities. **Important:** the Wu-Benkeser pathway requires base learners whose `predict` method returns density ratios on the positive real line, not probabilities in `[0, 1]`. Standard SuperLearner wrappers (e.g. `SL.glm`, `SL.xgboost`) are not compatible – you must supply custom wrappers that internally fit a classifier and convert predictions to density ratios before returning them. This package does not currently include such wrappers; built-in DR-returning wrappers compatible with the WB pathway are planned for version 1.0.

`drop_small_cluster_splits` Logical. Drop time points where a fold has too few clusters to fit a model. Default `TRUE`.

`parallel_t` Logical. Parallelise across time points. Default `FALSE`.

`t_workers` Integer. Number of workers for time-point parallelism. `NULL` auto-detects.

`fold_workers` Integer. Workers for parallelising across outer cross-fitting folds within each time point via `parallel::mclapply()` (fork-based; Linux/Mac only). `NULL` disables. When combined with `parallel_t`, the nested `mclapply` scheme limits use of multi-threaded or GPU-based learners. Default `NULL`.

`sl_workers` Integer. Workers for parallel learner evaluation within each SuperLearner call via `SuperLearner::mcSuperLearner()` (fork-based; Linux/Mac only). `NULL` disables. Default `NULL`. **Note:** not compatible with all learners.

`verbose` Logical. If `TRUE` (default), print per-time progress messages during estimation.

Details

Cross-fitting is used throughout: treatment models are trained on one fold and density ratios predicted on the held-out fold, so the same fold structure is inherited by the downstream estimator.

Value

A list with components:

`weights_dt` A `data.table` with one row per subject-time containing `Rt_t` (instantaneous density ratio) and `global_fold`. Pass this to `sdr()` or `itmle()` as `weight_object`.

`sl_summary` `data.table` of SuperLearner learner weights. One row per learner per (fold, time-point). Useful for checking which treatment models dominate across time points.

`fold_diag` `data.table` of per-fold, per-time-point diagnostics: mean predicted probabilities, effective sample sizes (ESS) for the density ratios, and calibration summaries for each treatment model. ESS collapse (ESS much smaller than n) indicates extreme density ratios and warrants caution.

Parallelism

`t_workers`, `fold_workers`, and `sl_workers` can be used independently or together. Using a single level is robust with all learners. Combining two or more creates nested `mclapply` calls; in that case multi-threaded or GPU-based learners (e.g. xgboost with CUDA, OpenMP-based methods) may crash in the child processes and should be avoided or limited to one thread.

Sample size requirements

This package targets large longitudinal datasets – thousands of subjects – typical of ICU, ward, or emergency department cohorts. Stable density ratio estimation requires adequate treatment-variation at every time point in each cross-fitting fold. All built-in examples use a minimum of 2,000 subjects.

References

Diaz I, Williams N, Hoffman KL, Schenck EJ (2021). Nonparametric Causal Effects Based on Longitudinal Modified Treatment Policies. *JASA* 118(542):846-857.

Wu C, Benkeser D (2024). Nonparametric Efficient Estimation of Marginal Structural Models using Targeted Machine Learning. arXiv:2408.10847.

Williams NT, Diaz I (2023). lmtp: An R package for estimating the causal effects of modified treatment policies. *Observational Studies*.

See Also

`sdr()`, `itmle()`

Examples

```
library(SuperLearner)

# ICU-like DGP: patients die, discharge, or remain in state each time point
sim_ex <- function(n = 2000L, tmax = 5L) {
  set.seed(42L)
  rows <- vector("list", n)
  for (i in seq_len(n)) {
    age <- round(rnorm(1, 65, 10)); L1 <- rnorm(1)
    pat <- list()
    for (t in seq_len(tmax)) {
      A     <- rbinom(1, 1, plogis(0.3 * L1 - 0.4))
      u     <- runif(1)
      p_die <- plogis(-4.0 + 0.2 * L1 - 0.1 * age / 10)
      p_dc  <- plogis(-2.5 + 0.5 * A)
```

```
      if (u < p_die) {
        alive <- 0L; in_state <- 0L
      } else if (u < p_die + p_dc) {
        alive <- 1L; in_state <- 0L
      } else {
        alive <- 1L; in_state <- 1L
      }
      Y <- rbinom(1, 1, plogis(-0.5 + 0.4 * A - 0.2 * L1))
      pat[[length(pat) + 1L]] <- data.frame(
        id = i, time = t, age = age, L1 = L1,
        A = A, alive = alive, in_state = in_state, Y = Y
      )
      if (in_state == 0L) break
      if (t < tmax) L1 <- L1 + rnorm(1, -0.1 * A, 0.3)
    }
    rows[[i]] <- do.call(rbind, pat)
  }
  do.call(rbind, rows)
}
df <- sim_ex()

# Policy: increase treatment probability by 0.3
policy_fn <- function(D_block, t, a_names) {
  out <- D_block[, ..a_names, drop = FALSE]
  out[[a_names[1]]] <- pmin(D_block[[a_names[1]]] + 0.3, 1)
  out
}

wr <- density_ratio(
  df              = df,
  a_names         = "A",
  tmax            = 5L,
  baseline        = "age",
  tv_names        = "L1",
  sl_g            = c("SL.mean", "SL.glm"),
  k               = 1L,
  inner_v         = 3L,
  v               = 3L,
  seed            = 1L,
  id              = "id",
  time            = "time",
  policy_spec_fun = policy_fn
)
head(wr$weights_dt)


# Multi-treatment: joint density ratio for two binary treatments (A1, A2)

library(SuperLearner)

sim_multi <- function(n = 2000L, tmax = 5L) {
  set.seed(42L)
  rows <- vector("list", n)
```

```
  for (i in seq_len(n)) {
    age <- round(rnorm(1, 65, 10)); L1 <- rnorm(1)
    pat <- list()
    for (t in seq_len(tmax)) {
      A1    <- rbinom(1, 1, plogis(0.3 * L1 - 0.4))
      A2    <- rbinom(1, 1, plogis(0.2 * L1 + 0.3 * A1 - 0.3))
      u     <- runif(1)
      p_die <- plogis(-4.0 + 0.2 * L1 - 0.1 * age / 10)
      p_dc  <- plogis(-2.5 + 0.4 * A1 + 0.3 * A2)
      if (u < p_die) {
        alive <- 0L; in_state <- 0L
      } else if (u < p_die + p_dc) {
        alive <- 1L; in_state <- 0L
      } else {
        alive <- 1L; in_state <- 1L
      }
      Y <- rbinom(1, 1, plogis(-0.5 + 0.3 * A1 + 0.3 * A2 - 0.2 * L1))
      pat[[length(pat) + 1L]] <- data.frame(
        id = i, time = t, age = age, L1 = L1,
        A1 = A1, A2 = A2, alive = alive, in_state = in_state, Y = Y
      )
      if (in_state == 0L) break
      if (t < tmax) L1 <- L1 + rnorm(1, -0.1 * A1, 0.3)
    }
    rows[[i]] <- do.call(rbind, pat)
  }
  do.call(rbind, rows)
}
df2 <- sim_multi()

# Joint policy: shift both treatments upward by 0.3
policy_fn2 <- function(D_block, t, a_names) {
  out <- D_block[, ..a_names, drop = FALSE]
  out[[a_names[1]]] <- pmin(D_block[[a_names[1]]] + 0.3, 1)
  out[[a_names[2]]] <- pmin(D_block[[a_names[2]]] + 0.3, 1)
  out
}

wr2 <- density_ratio(
  df              = df2,
  a_names         = c("A1", "A2"),
  tmax            = 5L,
  baseline        = "age",
  tv_names        = "L1",
  sl_g            = c("SL.mean", "SL.glm"),
  k               = 1L,
  inner_v         = 3L,
  v               = 3L,
  seed            = 1L,
  id              = "id",
  time            = "time",
  policy_spec_fun = policy_fn2
)
```

```
head(wr2$weights_dt)
```

`itmle` *Infinite-dimensional TMLE (iTMLE) estimator for longitudinal MTPs*

Description

Estimates the mean counterfactual outcome under a modified treatment policy (MTP) using the infinite-dimensional targeted minimum loss-based estimator (iTMLE). The estimator fits a backward Q-regression to approximate the counterfactual outcome trajectory, then applies an infinite-dimensional fluctuation – a targeting step that solves the efficient influence equation without restricting the fluctuation to a finite-dimensional parametric submodel. The algorithm follows Luedtke et al. (2017, Section 5 and Appendix 12): Section 5 introduces the infinite-dimensional fluctuation, Appendix 12 the cross-validated variant used here to prevent overfitting of Q estimates. Density-ratio weights from `density_ratio()` adapt the estimator to general MTPs following the framework of Diaz et al. (2021). The estimator is sequentially doubly robust (2^K-robust): consistent whenever, at each time point t, either the treatment model g_t or the outcome model Q_t is consistently estimated. The targeting step uses a SuperLearner ensemble with custom wrappers (see sl_itmle) that carry the logit offset required for cross-validated TMLE as a data column.

Usage

```
itmle(
  df,
  weight_object,
  tmax,
  id,
  time,
  alive,
  in_state,
  y,
  baseline,
  tv_names = character(0),
  a_names = character(0),
  no_lag_vars = character(0),
  policy_names = character(0),
  sl_remain = NULL,
  sl_death = NULL,
  sl_recursive = NULL,
  sl_rec_early = NULL,
  rec_transition = NULL,
  sl_y = NULL,
  outcome_family = c("binomial", "gaussian"),
  y_bounds = NULL,
  bounds = 1e-05,
```

```
  trim = 0.99,
  absorb = list(),
  policy_spec_fun = function(D_block, t, a_names) NULL,
  k = 2,
  seed = 1,
  parallel = FALSE,
  fold_workers = NULL,
  reg_workers = NULL,
  sl_workers = NULL,
  inner_v = 5L,
  sl_tmle = NULL,
  v_target_itmle = 10L,
  v_sl_inner_itmle = 10L,
  cluster = NULL,
  cluster_se_only = FALSE,
  pool_g_death = FALSE,
  pool_q_exit = FALSE,
  pool_time = "spline",
  verbose = TRUE
)
```

Arguments

df	A data.frame in long format (one row per subject per time point).
weight_object	Output from density_ratio(), or a data.frame containing at least columns for subject id, time, Rt_t (instantaneous density ratio), and global_fold (cross-fitting fold assignment).
tmax	Integer. Maximum follow-up time (number of time points).
id	Character. Name of the subject identifier column.
time	Character. Name of the integer time column.
alive	Character. Name of the binary alive indicator column (1 = alive, 0 = dead).
in_state	Character. Name of the binary active-state indicator column (1 = in active state e.g. ICU, 0 = transitioned out).
y	Character. Name of the outcome column.
baseline	Character vector of baseline (time-invariant) covariate names.
tv_names	Character vector of time-varying covariate names (excluding treatment).
a_names	Character vector of treatment variable names. Can contain one or more variables for joint multi-treatment policies (e.g. c("A1", "A2")). The policy_spec_fun must return shifted values for all variables in a_names, and density_ratio() must have been called with the same a_names to produce compatible weights.
no_lag_vars	Character vector of variables in tv_names or a_names that should not be lagged (e.g. already-encoded temporal features).
policy_names	Character vector of column names holding the shifted treatment values under the MTP (one per treatment variable in a_names).

`sl_remain` SuperLearner library for the `g_remain` model (probability of remaining in state at each time point). Required.

`sl_death` SuperLearner library for the `g_death_exit` model (probability of death among exiters). Required.

`sl_recursive` SuperLearner library for the recursive Q-remain model. Required.

`sl_rec_early` Optional SuperLearner library for the recursive Q-remain model at early time points (`tt <= rec_transition`). When supplied, this library is used for those early time steps and `sl_recursive` is used for the later ones – letting the user tune the recursion separately for early vs. late time points. Requires `rec_transition`.

`rec_transition` Integer. Time-point threshold: `sl_rec_early` is used for `tt <= rec_transition`, `sl_recursive` for later time points. Required when `sl_rec_early` is provided.

`sl_y` SuperLearner library for the Q-exit (outcome-at-exit) model. Required.

`outcome_family` `"binomial"` (default) or `"gaussian"`.

`y_bounds` Optional numeric vector of length 2, `c(min, max)`. For `outcome_family = "gaussian"`, all outcome values are internally scaled to `[0, 1]` before model fitting and back-transformed to the original scale for the final estimate – this keeps Q-model predictions bounded and stabilises the recursive regression. By default (`NULL`) the bounds are taken from the observed range of `y` in `df` (`min(y)`, `max(y)`). Supply explicit bounds to widen or narrow this range: wider bounds (e.g. extending beyond the observed range) give the models more room under the MTP if the shifted distribution may produce values outside the training range; narrower bounds clip more aggressively. Predictions outside the supplied range are clipped to `[0, 1]` before back-transformation. Ignored for `outcome_family = "binomial"` (binary outcomes are already in `[0, 1]`).

`bounds` Numeric. Probability clipping bound for g and Q predictions. Default `1e-5`.

`trim` Quantile used to cap instantaneous density ratios before they are assembled into the cumulative weight matrix. Trimming is applied to the **full** `weights_dt` (all time points computed by `density_ratio()`) before any subsetting to the current `tmax`. Consequently the trim threshold is identical whether you call the estimator with `tmax = 2` or `tmax = 14`, making results directly comparable across horizons that share the same weight object. Default `0.99`.

`absorb` List of `absorb_rule()` objects specifying outcome overrides at absorbing states.

`policy_spec_fun`
A function `(D_block, t, a_names)` returning a `data.table` with the shifted treatment values for time `t`. Used when the policy cannot be pre-computed as static columns.

`k` Integer. Number of lags of time-varying covariates and treatment to include in models. Default `2`.

`seed` Integer random seed. Default `1`.

`parallel` Logical. Enable parallel outer cross-fitting via `parallel::mclapply()`. Default `FALSE`.

`fold_workers` Integer number of worker processes for outer folds. `NULL` uses `parallel::detectCores()`.

`reg_workers` Integer number of worker processes for within-fold regression parallelism.

`sl_workers` Integer. Workers for parallel learner evaluation within each SuperLearner call via `SuperLearner::mcSuperLearner()` (fork-based; Linux/Mac only). Ignored when `parallel = FALSE`. Default `NULL` (sequential). **Note:** not compatible with all learners.

`inner_v` Integer. Number of inner cross-validation folds for SuperLearner. Default `5L`.

`sl_tmle` SuperLearner library for the iTMLE targeting step. Should consist of learners from sl_itmle that accept an offset column (`._sl_offset`). Defaults to the package-provided `sl_tmle` vector.

`v_target_itmle` Integer. Number of cross-validation folds for the outer targeting loop. Default `10L`.

`v_sl_inner_itmle` Integer. Number of inner CV folds inside the targeting SuperLearner. Default `10L`.

`cluster` Character. Name of a cluster variable for cluster-robust standard errors (e.g. hospital). If `NULL`, subject id is used.

`cluster_se_only` Logical. If `TRUE`, skip fitting and return only the fold structure for SE computation. Default `FALSE`.

`pool_g_death` Logical. If `TRUE`, fit a single pooled `g_death_exit` model across all time points (with time as a covariate) rather than a separate model per time point. Useful when deaths are sparse at individual time steps – pooling borrows strength across time and avoids near-empty training sets in late time points where mortality is rare. Use with caution if the death hazard changes substantially over time, as the pooled model must then capture that trend through the time covariate. Default `FALSE`.

`pool_q_exit` Logical. If `TRUE`, fit a single pooled `Q_exit` model across all time points (with time and `exit_status` as covariates) rather than a separate model per time point. Complements `pool_g_death`: useful when exit events (deaths + discharges) are sparse at individual time steps. When both `pool_g_death` and `pool_q_exit` are `TRUE` and `parallel = TRUE` with `reg_workers > 1`, the two pre-loop fits run simultaneously. Predictions are made in two passes – once with `exit_status = 1` (death branch) and once with `exit_status = 0` (discharge branch) – on all at-risk subjects, not just those who exited. Default `FALSE`.

`pool_time` Character. Basis used to encode time as a covariate in pooled models. One of `"spline"` (natural spline, default), `"linear"`, or `"factor"`. Only relevant when `pool_g_death = TRUE` or `pool_q_exit = TRUE`.

`verbose` Logical. If `TRUE` (default), print per-fold and per-time progress messages during estimation.

Details

Two properties of iTMLE worth noting relative to `sdr()`: (1) under the natural-course policy iTMLE does not collapse algebraically to `mean(Y)`, so a natural-course run provides a genuine calibration check on the Q models after the fluctuation update; (2) the mean efficient influence curve is not zero by construction (targeting converges to near-zero but not exactly zero), so the SE uses the uncentered second-moment estimator $\sqrt{\text{mean}(IC^2)/n}$ rather than $sd(IC)/\sqrt{n}$ – see the `se` entry in `Value` for details.

Value

A named list with the same structure as `sdr()`, except where noted. Top-level elements:

`psi` EIF point estimate of `E[Y(d)]` under the MTP, after the iTMLE targeting step.

`se` Standard error from the efficient influence curve, computed as `sqrt(mean(IC^2) / n)` – the uncentered second-moment estimator (or its cluster-robust analogue). Unlike the standard `sd(IC)/sqrt(n)` used by `sdr()`, this does not assume `E[IC] = 0`: it remains conservative when the targeting loop has not fully converged to a zero-mean IC. When targeting converges well the two are essentially identical. The centered and second-moment SEs are both available in `diagnostics$se_info` (`se_centered` and `se_second_moment`) for comparison.

`ci` 95% Wald confidence interval: `psi +/- 1.96 * se`.

`Y_obs` Observed mean outcome `mean(Y)`. Quick sanity check against `psi` under the natural-course policy.

`ic_df` `data.table` with columns `id` and `ic` (per-subject influence curve values). Used by `contrast()`.

Predictions (`$predictions`): cross-fitted Q matrices, one column per time point, one row per subject.

`$natural` `Q(t)` under the natural-course policy.

`$shifted` `Q(t)` under the MTP, after the targeting step.

Diagnostics (`$diagnostics`): model fit, calibration, and targeting summaries. iTMLE-specific additions are noted.

`$recursion_diag` `data.table`, one row per fold x time point. Tracks g/Q predictions under natural and shifted policy, the current regression target (`Y_target_*`, the outer-step targeted Q at t+1), pre- and post-targeting Q on the training side (`Q_{nat,shf}_pre_*`, `Q_{nat,shf}_post_*`), model-fit residuals (`resid_sd`, `resid_q95_abs`, `resid_max_abs` – `Y_target` minus shifted-mixture Q), targeting update magnitudes (`delta_{nat,shf}_target_{sd,q95_abs,max_abs}`), and pre-/post- targeting Q means on the validation set (`Q_nat_vl_pre_mean`, `Q_shf_vl_pre_mean`, `Q_nat_vl_post_mean`, `Q_shf_vl_post_mean`).

`$branch_cal` Per-fold, per-time calibration table for `g_remain`, `g_death`, `Q_rem`, and `Q_exit`. Same structure as `sdr()`.

`$target_cal` *(iTMLE only)* `data.table` with one row per targeting iteration (fold x time x iteration), tracking EIF magnitude as it decreases toward convergence.

`$target_sl` *(iTMLE only)* `data.table` of SuperLearner weights from the fluctuation (targeting) model, per fold and time point. Shows which wrappers from sl_itmle were selected.

`$sl_summary` `data.table` of SuperLearner learner weights for the Q/g models. Same structure as `sdr()`.

`$se_info` *(iTMLE only)* List with SE computation details: `se`, `n`, `n_eff`, and cluster-adjustment bookkeeping.

Weights (`$weights`), **Settings** (`$settings`), and **Call** (`$call`) have the same structure as `sdr()`. `$settings$start_t` records the first time point used.

Natural-course diagnostic value

Unlike `sdr()`, iTMLE does not collapse to `mean(Y)` under the natural-course policy (see `sdr()` for why SDR does). A natural-course iTMLE run reflects the calibration of Q* (Q after the fluctuation update). To assess the Q models before targeting, use `qreg()` under the natural course.

Sample size requirements

This package targets large longitudinal datasets – thousands of subjects – typical of ICU, ward, or emergency department cohorts. The doubly-robust estimators require adequate observations within each branch (alive, in-state, exited) at every time point for the SuperLearner component models to be stable. `pool_g_death` and `pool_q_exit` borrow strength across time steps when exit events are sparse, but there must still be sufficient events across the pooled structure. All built-in examples use a minimum of 2,000 subjects.

Diagnostics

The returned fit carries several diagnostic tables under `$diagnostics`:

- `branch_cal` – per-fold, per-time calibration for the four branches (`g_remain`, `g_death`, `q_rem`, `q_exit`); summarise with `branch_cal_summary()` and use to tune the g/Q SuperLearner libraries.
- `recursion_diag` – per-fold, per-time EIF/targeting mechanics (`Y_target_*`, `Q_*_pre/post_*`, `delta_*_target_*`, `resid_*`, range-violation counts, validation-side Q means). Post-hoc sanity.
- `target_cal` – per outer/inner targeting iteration, tracks EIF magnitude toward convergence.
- `target_sl` – per (fold, t) SuperLearner coefficients from the fluctuation model; use to prune `sl_tmle` wrappers that never fire.

`sl_summary` (per-(fold, t, component) SuperLearner coefficients for the g/Q models) and `ic_df` (per-subject influence-curve values consumed by `contrast()`) are also attached. Weight diagnostics live on the `density_ratio()` object itself – see `weight_diagnostics()`. A worked workflow is in `vignette("diagnostics")`.

References

Luedtke AR, Sofrygin O, van der Laan MJ, Carone M (2017). Sequential Double Robustness in Right-Censored Longitudinal Models. arXiv:1705.02459.

Diaz I, Williams N, Hoffman KL, Schenck EJ (2021). Nonparametric Causal Effects Based on Longitudinal Modified Treatment Policies. *JASA* 118(542):846-857.

See Also

`density_ratio()`, `sdr()`, `contrast()`, `absorb_rule()`, sl_itmle, `weight_diagnostics()`, `branch_cal_summary()`

Examples

```
library(SuperLearner)
sl_lib  <- c("SL.mean", "SL.glm")
tgt_lib <- c("SL.tmle_empty", "SL.tmle_intercept", "SL.tmle_glm")

# ---- Single binary treatment (sim_bin) ------------------------------
df <- sim_bin(n = 1000L, tmax = 3L, seed = 1L)
policy_bin <- function(D_block, t, a_names) {
  out <- D_block[, ..a_names, drop = FALSE]
  out[[a_names[1]]] <- pmax(
    D_block[[a_names[1]]], as.integer(D_block[["L1"]] > 1.0)
  )
  out
}
wr <- density_ratio(
  df = df, a_names = "A", tmax = 3L,
  baseline = c("age", "sex"), tv_names = c("L1", "L2"),
  sl_g = sl_lib, k = 1L, inner_v = 3L, v = 3L, seed = 1L,
  id = "id", time = "time", policy_spec_fun = policy_bin
)
res <- itmle(
  df = df, weight_object = wr, tmax = 3L,
  id = "id", time = "time", alive = "alive", in_state = "in_state", y = "Y",
  baseline = c("age", "sex"), tv_names = c("L1", "L2"), a_names = "A",
  sl_remain = sl_lib, sl_death = sl_lib,
  sl_recursive = sl_lib, sl_y = sl_lib, sl_tmle = tgt_lib,
  k = 1L, inner_v = 2L, v_target_itmle = 2L, v_sl_inner_itmle = 2L,
  parallel = FALSE, seed = 1L, policy_spec_fun = policy_bin
)
res$psi; res$se

# ---- Single continuous treatment (sim_cont) -------------------------
df_c <- sim_cont(n = 1000L, tmax = 3L, seed = 1L)
policy_cont <- function(D_block, t, a_names) {
  out <- D_block[, ..a_names, drop = FALSE]
  out[[a_names[1]]] <- pmin(D_block[[a_names[1]]] + 0.2, 2.0)
  out
}
wr_c <- density_ratio(
  df = df_c, a_names = "A", tmax = 3L,
  baseline = "age", tv_names = "L1",
  sl_g = sl_lib, k = 1L, inner_v = 3L, v = 3L, seed = 1L,
  id = "id", time = "time", policy_spec_fun = policy_cont
)
res_c <- itmle(
  df = df_c, weight_object = wr_c, tmax = 3L,
  id = "id", time = "time", alive = "alive", in_state = "in_state", y = "Y",
  baseline = "age", tv_names = "L1", a_names = "A",
  sl_remain = sl_lib, sl_death = sl_lib,
  sl_recursive = sl_lib, sl_y = sl_lib, sl_tmle = tgt_lib,
  k = 1L, inner_v = 2L, v_target_itmle = 2L, v_sl_inner_itmle = 2L,
  parallel = FALSE, seed = 1L, policy_spec_fun = policy_cont
```

```
)
res_c$psi; res_c$se

# ---- Multiple treatments -- binary + continuous (sim_multi) ---------
df_m <- sim_multi(
  n = 1000L, tmax = 3L, seed = 1L, n_binary = 1L, n_continuous = 1L
)
policy_multi <- function(D_block, t, a_names) {
  out <- D_block[, ..a_names, drop = FALSE]
  out[["A_b1"]] <- pmax(D_block[["A_b1"]], as.integer(D_block[["L1"]] > 1.0))
  out[["A_c1"]] <- pmin(D_block[["A_c1"]] + 0.2, 2.0)
  out
}
wr_m <- density_ratio(
  df = df_m, a_names = c("A_b1", "A_c1"), tmax = 3L,
  baseline = c("age", "sex"), tv_names = "L1",
  sl_g = sl_lib, k = 1L, inner_v = 2L, v = 2L, seed = 1L,
  id = "id", time = "time", policy_spec_fun = policy_multi
)
res_m <- itmle(
  df = df_m, weight_object = wr_m, tmax = 3L,
  id = "id", time = "time", alive = "alive", in_state = "in_state", y = "Y",
  baseline = c("age", "sex"), tv_names = "L1", a_names = c("A_b1", "A_c1"),
  sl_remain = sl_lib, sl_death = sl_lib,
  sl_recursive = sl_lib, sl_y = sl_lib, sl_tmle = tgt_lib,
  k = 1L, inner_v = 2L, v_target_itmle = 2L, v_sl_inner_itmle = 2L,
  parallel = FALSE, seed = 1L, policy_spec_fun = policy_multi
)
res_m$psi; res_m$se
```

`method.WB_dr` *Wu-Benkeser log-density-ratio metalearner for SuperLearner*

Description

A SuperLearner metalearner that combines base learners by minimising a log-density-ratio loss rather than the default non-negative least squares (NNLS). Weights are found by BFGS optimisation over the softmax-reparameterised simplex; if BFGS fails to converge, equal weights are used as a fallback.

The metalearner operates in density-ratio space: base learners must return density ratios (not probabilities). Standard SuperLearner wrappers such as `SL.glm` or `SL.xgboost` return probabilities and are not compatible with this metalearner. Custom DR-returning wrappers are required; built-in wrappers for the WB pathway are planned for version 1.0.

Clipping note: the standard `bounds` clipping applied to all probability-scale predictions elsewhere in the pipeline does not apply here. DR-scale flooring is handled by `dr_floor` inside this function.

Usage

```
method.WB_dr(dr_floor = 1e-10)
```

Arguments

dr_floor	Numeric scalar. Floor applied to density-ratio predictions from each base learner before computing the log-DR loss. Prevents log(0) during optimisation. Default 1e-10.

Value

A SuperLearner method list (with computeCoef and computePred slots) suitable for passing to the method argument of SuperLearner.

References

Wu and Benkeser (2021). Improved inference for vaccine-induced immune responses via shape-constrained methods.

See Also

density_ratio

---

print.sdr_fit *Print methods for CausalState output objects*

Description

Compact console summaries for the main CausalState output objects.

print.sdr_fit and print.itmle_fit display the point estimate, standard error, 95% Wald CI, and the observed outcome mean.

print.qreg_fit displays the shifted estimate, SE, 95% CI, and the natural-course plug-in for comparison.

print.branch_cal_summary renders a wide table with super-column headers per branch (g_remain, g_death, q_rem, q_exit), showing fold-weighted n_vl, target mean, predicted mean, and calibration slope for each time step. The underlying data.table (returned invisibly) contains all additional metrics.

Usage

```
## S3 method for class 'sdr_fit'
print(x, digits = 3L, ...)

## S3 method for class 'itmle_fit'
print(x, digits = 3L, ...)
```

```
## S3 method for class 'qreg_fit'
print(x, digits = 3L, ...)

## S3 method for class 'branch_cal_summary'
print(x, digits = 3L, ...)
```

Arguments

x	A fitted object returned by sdr(), itmle(), qreg(), or branch_cal_summary().
digits	Number of decimal places for numeric output. Default 3L.
...	Ignored.

Value

The input object, invisibly.

See Also

sdr, itmle, qreg, branch_cal_summary

---

qreg *Pure Q-recursion estimator – diagnostic use only*

Description

Fits sequential Q-models backwards through time and returns the plug-in (substitution) estimator $\hat{\Psi} = n^{-1} \sum_i \hat{Q}_1(H_{i1})$. No EIF update is applied, so the estimate carries first-order bias and **should not be used as a real causal estimate**.

Usage

```
qreg(
  df,
  tmax,
  id,
  time,
  alive,
  in_state,
  y,
  baseline,
  tv_names = character(0),
  a_names = character(0),
  no_lag_vars = character(0),
  policy_names = character(0),
  sl_remain = NULL,
  sl_death = NULL,
  sl_recursive = NULL,
```

```
  sl_y = NULL,
  outcome_family = c("binomial", "gaussian"),
  y_bounds = NULL,
  bounds = 1e-05,
  absorb = list(),
  policy_spec_fun = function(D_block, t, a_names) NULL,
  k = 2,
  seed = 1,
  parallel = FALSE,
  fold_workers = NULL,
  reg_workers = NULL,
  sl_workers = NULL,
  inner_v = 5L,
  cluster = NULL,
  pool_g_death = FALSE,
  pool_q_exit = FALSE,
  pool_time = "spline",
  weight_object = NULL,
  v = 5L,
  trim = 0.99,
  verbose = TRUE
)
```

Arguments

df	A data.frame in long format (one row per subject per time point).
tmax	Integer. Maximum follow-up time (number of time points).
id	Character. Name of the subject identifier column.
time	Character. Name of the integer time column.
alive	Character. Name of the binary alive indicator column (1 = alive, 0 = dead).
in_state	Character. Name of the binary active-state indicator column (1 = in active state e.g. ICU, 0 = transitioned out).
y	Character. Name of the outcome column.
baseline	Character vector of baseline (time-invariant) covariate names.
tv_names	Character vector of time-varying covariate names (excluding treatment).
a_names	Character vector of treatment variable names. Can contain one or more variables for joint multi-treatment policies (e.g. c("A1", "A2")). The policy_spec_fun must return shifted values for all variables in a_names, and density_ratio() must have been called with the same a_names to produce compatible weights.
no_lag_vars	Character vector of variables in tv_names or a_names that should not be lagged (e.g. already-encoded temporal features).
policy_names	Character vector of column names holding the shifted treatment values under the MTP (one per treatment variable in a_names).
sl_remain	SuperLearner library for the g_remain model (probability of remaining in state at each time point). Required.

sl_death	SuperLearner library for the g_death_exit model (probability of death among exiters). Required.
sl_recursive	SuperLearner library for the recursive Q-remain model. Required.
sl_y	SuperLearner library for the Q-exit (outcome-at-exit) model. Required.
outcome_family	"binomial" (default) or "gaussian".
y_bounds	Optional numeric vector of length 2, c(min, max). For outcome_family = "gaussian", all outcome values are internally scaled to [0, 1] before model fitting and back-transformed to the original scale for the final estimate – this keeps Q-model predictions bounded and stabilises the recursive regression. By default (NULL) the bounds are taken from the observed range of y in df (min(y), max(y)). Supply explicit bounds to widen or narrow this range: wider bounds (e.g. extending beyond the observed range) give the models more room under the MTP if the shifted distribution may produce values outside the training range; narrower bounds clip more aggressively. Predictions outside the supplied range are clipped to [0, 1] before back-transformation. Ignored for outcome_family = "binomial" (binary outcomes are already in [0, 1]).
bounds	Numeric. Probability clipping bound for g and Q predictions. Default 1e-5.
absorb	List of absorb_rule() objects specifying outcome overrides at absorbing states.
policy_spec_fun	A function (D_block, t, a_names) returning a data.table with the shifted treatment values for time t. Used when the policy cannot be pre-computed as static columns.
k	Integer. Number of lags of time-varying covariates and treatment to include in models. Default 2.
seed	Integer random seed. Default 1.
parallel	Logical. Enable parallel outer cross-fitting via parallel::mclapply(). Default FALSE.
fold_workers	Integer number of worker processes for outer folds. NULL uses parallel::detectCores().
reg_workers	Integer number of worker processes for within-fold regression parallelism.
sl_workers	Integer. Workers for parallel learner evaluation within each SuperLearner call via SuperLearner::mcSuperLearner() (fork-based; Linux/Mac only). Ignored when parallel = FALSE. Default NULL (sequential). **Note:** not compatible with all learners.
inner_v	Integer. Number of inner cross-validation folds for SuperLearner. Default 5L.
cluster	Character. Name of a cluster variable for cluster-robust standard errors (e.g. hospital). If NULL, subject id is used.
pool_g_death	Logical. If TRUE, fit a single pooled g_death_exit model across all time points (with time as a covariate) rather than a separate model per time point. Useful when deaths are sparse at individual time steps – pooling borrows strength across time and avoids near-empty training sets in late time points where mortality is rare. Use with caution if the death hazard changes substantially over time, as the pooled model must then capture that trend through the time covariate. Default FALSE.

`pool_q_exit` Logical. If `TRUE`, fit a single pooled `Q_exit` model across all time points (with time and `exit_status` as covariates) rather than a separate model per time point. Complements `pool_g_death`: useful when exit events (deaths + discharges) are sparse at individual time steps. When both `pool_g_death` and `pool_q_exit` are `TRUE` and `parallel = TRUE` with `reg_workers > 1`, the two pre-loop fits run simultaneously. Predictions are made in two passes – once with `exit_status = 1` (death branch) and once with `exit_status = 0` (discharge branch) – on all at-risk subjects, not just those who exited. Default `FALSE`.

`pool_time` Character. Basis used to encode time as a covariate in pooled models. One of `"spline"` (natural spline, default), `"linear"`, or `"factor"`. Only relevant when `pool_g_death = TRUE` or `pool_q_exit = TRUE`.

`weight_object` Optional. Output from `density_ratio()`. If supplied, an EIF-based SE is computed in addition to the naive SE. The fold structure is inherited from `weight_object` when provided; otherwise independent folds are created using `v` and `seed`.

`v` Integer. Number of cross-fitting folds used when `weight_object = NULL`. Default `5L`.

`trim` Quantile for capping density ratios before EIF-based SE computation (only used when `weight_object` is supplied). Default `0.99`.

`verbose` Logical. If `TRUE` (default), print per-fold and per-time progress messages during estimation.

Details

The intended use is to pass the **natural-course** (identity) policy – that is, no shift – and compare the resulting estimate to the observed mean of `y`. Close agreement indicates that the Q-models are well-calibrated under the natural course, which is a prerequisite for the doubly-robust estimates from `sdr()` or `itmle()` to be trustworthy. Poor agreement signals Q-model misspecification.

Do not pass a shifted policy to `qreg` and interpret the result as a causal estimate. For policy evaluation use `sdr()` or `itmle()`.

The reported `se_naive` is $s(\hat{Q}_1)/\sqrt{n}$, which treats the fitted Q as fixed and should be read only as a rough guide to Monte Carlo variability of the plug-in.

Value

A list with:

`estimate` Point estimate $\hat{\Psi}$.

`se_naive` Naive SE (always present).

`se_eif` EIF-based SE (`NULL` if no `weight_object`).

`se` Best available SE: `se_eif` if available, else `se_naive`.

`ci` 95% Wald CI using `se`.

`psi_natural` Plug-in estimate under the natural course.

`psi_shifted` Plug-in estimate under the MTP (= `estimate`).

`sl_summary` `data.table` of SuperLearner learner weights per (fold, time-point, model component). Same structure as in `sdr()`.

`fold_diag` `data.table` of per-fold, per-time-point diagnostics.

`diagnostics$branch_cal` Per-fold, per-time branch calibration table: empirical mean targets vs. predictions for `g_remain`, `g_death`, and `Q_remain`. Use this to check whether the Q-models are well-calibrated under the natural course – the primary diagnostic purpose of `qreg()`.

`diagnostics$diag_table` Additional per-fold, per-time summary statistics (Q prediction ranges, pseudo-outcome statistics).

Sample size requirements

This package targets large longitudinal datasets – thousands of subjects – typical of ICU, ward, or emergency department cohorts. The Q-models require adequate observations within each branch (alive, in-state, exited) at every time point to be stable. `pool_g_death` and `pool_q_exit` borrow strength across time steps when exit events are sparse, but there must still be sufficient events across the pooled structure. All built-in examples use a minimum of 2,000 subjects.

See Also

`sdr()`, `itmle()`, `density_ratio()`

Examples

```
library(SuperLearner)

# ICU-like DGP: patients die, discharge, or remain in state each time point
sim_ex <- function(n = 2000L, tmax = 5L) {
  set.seed(42L)
  rows <- vector("list", n)
  for (i in seq_len(n)) {
    age <- round(rnorm(1, 65, 10)); L1 <- rnorm(1)
    pat <- list()
    for (t in seq_len(tmax)) {
      A     <- rbinom(1, 1, plogis(0.3 * L1 - 0.4))
      u     <- runif(1)
      p_die <- plogis(-4.0 + 0.2 * L1 - 0.1 * age / 10)
      p_dc  <- plogis(-2.5 + 0.5 * A)
      if (u < p_die) {
        alive <- 0L; in_state <- 0L
      } else if (u < p_die + p_dc) {
        alive <- 1L; in_state <- 0L
      } else {
        alive <- 1L; in_state <- 1L
      }
      Y <- rbinom(1, 1, plogis(-0.5 + 0.4 * A - 0.2 * L1))
      pat[[length(pat) + 1L]] <- data.frame(
        id = i, time = t, age = age, L1 = L1,
        A = A, alive = alive, in_state = in_state, Y = Y
      )
      if (in_state == 0L) break
      if (t < tmax) L1 <- L1 + rnorm(1, -0.1 * A, 0.3)
    }
    rows[[i]] <- do.call(rbind, pat)
```

```
  }
  do.call(rbind, rows)
}
df <- sim_ex()

# Natural-course policy (no shift) -- the correct use of qreg
policy_nat <- function(D_block, t, a_names) D_block[, ..a_names, drop = FALSE]

sl_lib <- c("SL.mean", "SL.glm")

# Q-model calibration check: estimate should be close to mean(df$Y)
res <- qreg(
  df              = df,
  tmax            = 5L,
  id              = "id",
  time            = "time",
  alive           = "alive",
  in_state        = "in_state",
  y               = "Y",
  baseline        = "age",
  tv_names        = "L1",
  a_names         = "A",
  sl_remain       = sl_lib,
  sl_death        = sl_lib,
  sl_recursive    = sl_lib,
  sl_y            = sl_lib,
  k               = 1L,
  inner_v         = 3L,
  parallel        = FALSE,
  seed            = 1L,
  policy_spec_fun = policy_nat
)

# Compare plug-in to observed mean: close agreement = well-calibrated Q
res$estimate
mean(df$Y)


# Multi-treatment: plug-in estimate under a joint policy over two treatments

library(SuperLearner)

sim_multi <- function(n = 2000L, tmax = 5L) {
  set.seed(42L)
  rows <- vector("list", n)
  for (i in seq_len(n)) {
    age <- round(rnorm(1, 65, 10)); L1 <- rnorm(1)
    pat <- list()
    for (t in seq_len(tmax)) {
      A1    <- rbinom(1, 1, plogis(0.3 * L1 - 0.4))
      A2    <- rbinom(1, 1, plogis(0.2 * L1 + 0.3 * A1 - 0.3))
      u     <- runif(1)
      p_die <- plogis(-4.0 + 0.2 * L1 - 0.1 * age / 10)
```

```
      p_dc  <- plogis(-2.5 + 0.4 * A1 + 0.3 * A2)
      if (u < p_die) {
        alive <- 0L; in_state <- 0L
      } else if (u < p_die + p_dc) {
        alive <- 1L; in_state <- 0L
      } else {
        alive <- 1L; in_state <- 1L
      }
      Y <- rbinom(1, 1, plogis(-0.5 + 0.3 * A1 + 0.3 * A2 - 0.2 * L1))
      pat[[length(pat) + 1L]] <- data.frame(
        id = i, time = t, age = age, L1 = L1,
        A1 = A1, A2 = A2, alive = alive, in_state = in_state, Y = Y
      )
      if (in_state == 0L) break
      if (t < tmax) L1 <- L1 + rnorm(1, -0.1 * A1, 0.3)
    }
    rows[[i]] <- do.call(rbind, pat)
  }
  do.call(rbind, rows)
}
df2 <- sim_multi()

policy_fn2 <- function(D_block, t, a_names) {
  out <- D_block[, ..a_names, drop = FALSE]
  out[[a_names[1]]] <- pmin(D_block[[a_names[1]]] + 0.3, 1)
  out[[a_names[2]]] <- pmin(D_block[[a_names[2]]] + 0.3, 1)
  out
}

sl_lib <- c("SL.mean", "SL.glm")

res2 <- qreg(
  df              = df2,
  tmax            = 5L,
  id              = "id",
  time            = "time",
  alive           = "alive",
  in_state        = "in_state",
  y               = "Y",
  baseline        = "age",
  tv_names        = "L1",
  a_names         = c("A1", "A2"),
  sl_remain       = sl_lib,
  sl_death        = sl_lib,
  sl_recursive    = sl_lib,
  sl_y            = sl_lib,
  k               = 1L,
  inner_v         = 3L,
  parallel        = FALSE,
  seed            = 1L,
  policy_spec_fun = policy_fn2
)
res2$estimate
```

```
res2$se
```

---

sdr *Sequential Doubly Robust (SDR) estimator for longitudinal MTPs*

Description

Estimates the mean counterfactual outcome under a modified treatment policy (MTP) using the LMTP-SDR estimator of Diaz et al. (2021), which extends the SDR construction of Luedtke et al. (2017) to general MTPs via density-ratio weighting. Starting from terminal pseudo-outcomes and working backward in time, the estimator propagates an EIF-corrected pseudo-outcome through a sequence of Q-regressions; the final estimate is the mean of the pseudo-outcomes at the first time point. The estimator is sequentially doubly robust (2^K-robust): consistent whenever, at each time point t, either the treatment model g_t or the outcome model Q_t is consistently estimated. Requires pre-computed density ratio weights from `density_ratio()`.

Usage

```
sdr(
  df,
  weight_object,
  tmax,
  id,
  time,
  alive,
  in_state,
  y,
  baseline,
  tv_names = character(0),
  a_names = character(0),
  no_lag_vars = character(0),
  policy_names = character(0),
  sl_remain = NULL,
  sl_death = NULL,
  sl_recursive = NULL,
  sl_rec_early = NULL,
  rec_transition = NULL,
  sl_y = NULL,
  outcome_family = c("binomial", "gaussian"),
  y_bounds = NULL,
  bounds = 1e-05,
  trim = 0.99,
  absorb = list(),
  policy_spec_fun = function(D_block, t, a_names) NULL,
  k = 2,
```

```
  seed = 1,
  parallel = FALSE,
  fold_workers = NULL,
  reg_workers = NULL,
  sl_workers = NULL,
  inner_v = 5L,
  cluster = NULL,
  cluster_se_only = FALSE,
  pool_g_death = FALSE,
  pool_q_exit = FALSE,
  pool_time = "spline",
  verbose = TRUE
)
```

Arguments

df	A data.frame in long format (one row per subject per time point).
weight_object	Output from density_ratio(), or a data.frame containing at least columns for subject id, time, Rt_t (instantaneous density ratio), and global_fold (cross-fitting fold assignment).
tmax	Integer. Maximum follow-up time (number of time points).
id	Character. Name of the subject identifier column.
time	Character. Name of the integer time column.
alive	Character. Name of the binary alive indicator column (1 = alive, 0 = dead).
in_state	Character. Name of the binary active-state indicator column (1 = in active state e.g. ICU, 0 = transitioned out).
y	Character. Name of the outcome column.
baseline	Character vector of baseline (time-invariant) covariate names.
tv_names	Character vector of time-varying covariate names (excluding treatment).
a_names	Character vector of treatment variable names. Can contain one or more variables for joint multi-treatment policies (e.g. c("A1", "A2")). The policy_spec_fun must return shifted values for all variables in a_names, and density_ratio() must have been called with the same a_names to produce compatible weights.
no_lag_vars	Character vector of variables in tv_names or a_names that should not be lagged (e.g. already-encoded temporal features).
policy_names	Character vector of column names holding the shifted treatment values under the MTP (one per treatment variable in a_names).
sl_remain	SuperLearner library for the g_remain model (probability of remaining in state at each time point). Required.
sl_death	SuperLearner library for the g_death_exit model (probability of death among exiters). Required.
sl_recursive	SuperLearner library for the recursive Q-remain model. Required.

`sl_rec_early` Optional SuperLearner library for the recursive Q-remain model at early time points (`tt <= rec_transition`). When supplied, this library is used for those early time steps and `sl_recursive` is used for the later ones – letting the user tune the recursion separately for early vs. late time points. Requires `rec_transition`.

`rec_transition` Integer. Time-point threshold: `sl_rec_early` is used for `tt <= rec_transition`, `sl_recursive` for later time points. Required when `sl_rec_early` is provided.

`sl_y` SuperLearner library for the Q-exit (outcome-at-exit) model. Required.

`outcome_family` `"binomial"` (default) or `"gaussian"`.

`y_bounds` Optional numeric vector of length 2, `c(min, max)`. For `outcome_family = "gaussian"`, all outcome values are internally scaled to `[0, 1]` before model fitting and back-transformed to the original scale for the final estimate – this keeps Q-model predictions bounded and stabilises the recursive regression. By default (`NULL`) the bounds are taken from the observed range of `y` in `df` (`min(y)`, `max(y)`). Supply explicit bounds to widen or narrow this range: wider bounds (e.g. extending beyond the observed range) give the models more room under the MTP if the shifted distribution may produce values outside the training range; narrower bounds clip more aggressively. Predictions outside the supplied range are clipped to `[0, 1]` before back-transformation. Ignored for `outcome_family = "binomial"` (binary outcomes are already in `[0, 1]`).

`bounds` Numeric. Probability clipping bound for g and Q predictions. Default `1e-5`.

`trim` Quantile used to cap instantaneous density ratios before they are assembled into the cumulative weight matrix. Trimming is applied to the **full** `weights_dt` (all time points computed by `density_ratio()`) before any subsetting to the current `tmax`. Consequently the trim threshold is identical whether you call the estimator with `tmax = 2` or `tmax = 14`, making results directly comparable across horizons that share the same weight object. Default `0.99`.

`absorb` List of `absorb_rule()` objects specifying outcome overrides at absorbing states.

`policy_spec_fun`
A function `(D_block, t, a_names)` returning a `data.table` with the shifted treatment values for time `t`. Used when the policy cannot be pre-computed as static columns.

`k` Integer. Number of lags of time-varying covariates and treatment to include in models. Default `2`.

`seed` Integer random seed. Default `1`.

`parallel` Logical. Enable parallel outer cross-fitting via `parallel::mclapply()`. Default `FALSE`.

`fold_workers` Integer number of worker processes for outer folds. `NULL` uses `parallel::detectCores()`.

`reg_workers` Integer number of worker processes for within-fold regression parallelism.

`sl_workers` Integer. Workers for parallel learner evaluation within each SuperLearner call via `SuperLearner::mcSuperLearner()` (fork-based; Linux/Mac only). Ignored when `parallel = FALSE`. Default `NULL` (sequential). **Note:** not compatible with all learners.

`inner_v` Integer. Number of inner cross-validation folds for SuperLearner. Default `5L`.

`cluster` Character. Name of a cluster variable for cluster-robust standard errors (e.g. hospital). If `NULL`, subject id is used.

`cluster_se_only` Logical. If `TRUE`, skip fitting and return only the fold structure for SE computation. Default `FALSE`.

`pool_g_death` Logical. If `TRUE`, fit a single pooled `g_death_exit` model across all time points (with time as a covariate) rather than a separate model per time point. Useful when deaths are sparse at individual time steps – pooling borrows strength across time and avoids near-empty training sets in late time points where mortality is rare. Use with caution if the death hazard changes substantially over time, as the pooled model must then capture that trend through the time covariate. Default `FALSE`.

`pool_q_exit` Logical. If `TRUE`, fit a single pooled `Q_exit` model across all time points (with time and `exit_status` as covariates) rather than a separate model per time point. Complements `pool_g_death`: useful when exit events (deaths + discharges) are sparse at individual time steps. When both `pool_g_death` and `pool_q_exit` are `TRUE` and `parallel = TRUE` with `reg_workers > 1`, the two pre-loop fits run simultaneously. Predictions are made in two passes – once with `exit_status = 1` (death branch) and once with `exit_status = 0` (discharge branch) – on all at-risk subjects, not just those who exited. Default `FALSE`.

`pool_time` Character. Basis used to encode time as a covariate in pooled models. One of `"spline"` (natural spline, default), `"linear"`, or `"factor"`. Only relevant when `pool_g_death = TRUE` or `pool_q_exit = TRUE`.

`verbose` Logical. If `TRUE` (default), print per-fold and per-time progress messages during estimation.

Value

A named list. Top-level elements:

`psi` EIF-corrected point estimate of `E[Y(d)]` under the MTP.

`se` Standard error from the efficient influence curve, computed as `sd(IC) / sqrt(n)` (or the cluster-robust sandwich equivalent when a cluster variable is supplied). This is the standard centered variance estimator and assumes `E[IC] = 0`, which holds exactly under the natural course and approximately under an MTP when the Q and g models are well-specified.

`ci` 95% Wald confidence interval: `psi +/- 1.96 * se`.

`Y_obs` Observed mean outcome `mean(Y)` across all subjects. Quick sanity check: under the natural-course policy `psi` should be close to `Y_obs`; a large gap suggests a data or model issue.

`ic_df` `data.table` with columns `id` and `ic` (per-subject influence curve values). Used by `contrast()`.

Predictions (`$predictions`): cross-fitted Q matrices, one column per time point, one row per subject.

`$natural` `Q(t)` under the natural-course policy.

`$shifted` `Q(t)` under the MTP.

Decomposition (`$decomposition`): plug-in components explaining the EIF correction relative to the raw plug-in.

`$psi_plugin_nat` Plug-in estimate under the natural course.

`$psi_plugin_shf` Plug-in estimate under the MTP.

`$psi_plugin_diff` Plug-in risk difference (`psi_plugin_shf` - `psi_plugin_nat`).

`$psi_eif_gap` EIF correction: `psi` - `psi_plugin_shf`. A large value means the pseudo-outcome recursion shifted the estimate substantially beyond the raw plug-in.

Diagnostics (`$diagnostics`): model fit and calibration summaries.

`$recursion_diag` `data.table`, one row per fold x time point. Tracks g/Q predictions (training side) under natural and shifted policy, the current regression target (`Y_target_*`) and the EIF-updated pseudo-outcome (`pseudo_post_*`), EIF update magnitude (`delta_sd`, `delta_q95_abs`, `delta_max_abs`), model-fit residuals (`resid_sd`, `resid_q95_abs`, `resid_max_abs` – `Y_target` minus shifted-mixture Q), and validation-side mixture Q means (`Q_nat_vl_mean`, `Q_shf_vl_mean`) for comparing training vs. validation trajectories.

`$branch_cal` Per-fold, per-time calibration table: empirical mean target vs. mean prediction for `g_remain`, `g_death`, `Q_rem`, and `Q_exit`. The `Q_rem` target is the pseudo-outcome mean – directly interpretable as a calibration check only under the natural-course policy. `qexit_type` records whether `Q_exit` is a binomial or Gaussian regression.

`$sl_summary` `data.table` of SuperLearner learner weights, one row per learner per (fold, time, component). Columns: `fold`, `t`, `component` (`g_remain`, `g_death`, `Q_rem`, `Q_exit`), plus one numeric column per learner. Consistently zero-weight learners can be dropped from the library.

Weights (`$weights`): density ratio inputs used by the estimator.

`$weights_dt` `data.table` copy of the density ratio object (trimmed to `tmax`, `Rt_cum` column removed).

`$trim` Trim quantile applied before forming cumulative density ratios.

Settings (`$settings`): parameters used for the fit.

`$start_t` First time point included in the estimation.

`$outcome_family` `"binomial"` or `"gaussian"`.

`$pool_g_death` Whether `g_death` was pooled across time.

`$pool_q_exit` Whether `Q_exit` was pooled across time.

`$variable_info` Named list of variable names and settings: `id`, `time`, `alive`, `in_state`, `cluster`, `baseline`, `time_varying`, `treatment`, `outcome`, `tmax`, `k`.

Call (`$call`): matched call expression, for reproducibility.

Parallelism

`fold_workers`, `reg_workers`, and `sl_workers` can be used independently or together. Using a single level is robust with all learners. Combining two or more creates nested `mclapply` calls; in that case multi-threaded or GPU-based learners (e.g. xgboost with CUDA, OpenMP-based methods) may crash in the child processes and should be avoided or limited to one thread.

Natural-course caution

Under the natural-course policy the cumulative density ratios equal one and the SDR pseudo-outcome recursion collapses algebraically to `mean(Y)`, regardless of Q-model quality. A natural-course SDR run cannot detect model misspecification. To assess model calibration use `diagnostics$branch_cal`, `fold_diag`, and `sl_summary`, or run `itmle()` under the natural course (iTMLE does not collapse in the same way).

Sample size requirements

This package targets large longitudinal datasets – thousands of subjects – typical of ICU, ward, or emergency department cohorts. The doubly-robust estimators require adequate observations within each branch (alive, in-state, exited) at every time point for the SuperLearner component models to be stable. `pool_g_death` and `pool_q_exit` borrow strength across time steps when exit events are sparse, but there must still be sufficient events across the pooled structure. All built-in examples use a minimum of 2,000 subjects.

Diagnostics

Two diagnostic tables are attached to the returned object under `$diagnostics`:

- `branch_cal` – per-fold, per-time calibration for the four branches (`g_remain`, `g_death`, `q_rem`, `q_exit`); summarise with `branch_cal_summary()` and use to tune the SuperLearner libraries.
- `recursion_diag` – per-fold, per-time EIF-mechanics diagnostics (`Y_target_*`, `pseudo_post_*`, `delta_*`, `resid_*`, range-violation counts, validation-side Q means). Post-hoc sanity checks.

The fit also carries `sl_summary` (per-(fold, t, component) SuperLearner coefficients) and `ic_df` (per-subject influence-curve values consumed by `contrast()`). Weight diagnostics live on the `density_ratio()` object itself – see `weight_diagnostics()`. A worked workflow is in `vignette("diagnostics")`.

References

Diaz I, Williams N, Hoffman KL, Schenck EJ (2021). Nonparametric Causal Effects Based on Longitudinal Modified Treatment Policies. *JASA* 118(542):846–857.

Luedtke AR, Sofrygin O, van der Laan MJ, Carone M (2017). Sequential Double Robustness in Right-Censored Longitudinal Models. arXiv:1705.02459.

See Also

`density_ratio()`, `itmle()`, `contrast()`, `absorb_rule()`, `weight_diagnostics()`, `branch_cal_summary()`

Examples

```
library(SuperLearner)
sl_lib <- c("SL.mean", "SL.glm")

# ---- Single binary treatment (sim_bin) ------------------------------
df <- sim_bin(n = 1000L, tmax = 3L, seed = 1L)
policy_bin <- function(D_block, t, a_names) {
  out <- D_block[, ..a_names, drop = FALSE]
  out[[a_names[1]]] <- pmax(
```

```
    D_block[[a_names[1]]], as.integer(D_block[["L1"]] > 1.0)
  )
  out
}
wr <- density_ratio(
  df = df, a_names = "A", tmax = 3L,
  baseline = c("age", "sex"), tv_names = c("L1", "L2"),
  sl_g = sl_lib, k = 1L, inner_v = 2L, v = 2L, seed = 1L,
  id = "id", time = "time", policy_spec_fun = policy_bin
)
res <- sdr(
  df = df, weight_object = wr, tmax = 3L,
  id = "id", time = "time", alive = "alive", in_state = "in_state", y = "Y",
  baseline = c("age", "sex"), tv_names = c("L1", "L2"), a_names = "A",
  sl_remain = sl_lib, sl_death = sl_lib,
  sl_recursive = sl_lib, sl_y = sl_lib,
  k = 1L, inner_v = 2L, parallel = FALSE, seed = 1L,
  policy_spec_fun = policy_bin
)
res$psi; res$se

# ---- Single continuous treatment (sim_cont) -------------------------
df_c <- sim_cont(n = 1000L, tmax = 3L, seed = 1L)
policy_cont <- function(D_block, t, a_names) {
  out <- D_block[, ..a_names, drop = FALSE]
  out[[a_names[1]]] <- pmin(D_block[[a_names[1]]] + 0.2, 2.0)
  out
}
wr_c <- density_ratio(
  df = df_c, a_names = "A", tmax = 3L,
  baseline = "age", tv_names = "L1",
  sl_g = sl_lib, k = 1L, inner_v = 2L, v = 2L, seed = 1L,
  id = "id", time = "time", policy_spec_fun = policy_cont
)
res_c <- sdr(
  df = df_c, weight_object = wr_c, tmax = 3L,
  id = "id", time = "time", alive = "alive", in_state = "in_state", y = "Y",
  baseline = "age", tv_names = "L1", a_names = "A",
  sl_remain = sl_lib, sl_death = sl_lib,
  sl_recursive = sl_lib, sl_y = sl_lib,
  k = 1L, inner_v = 2L, parallel = FALSE, seed = 1L,
  policy_spec_fun = policy_cont
)
res_c$psi; res_c$se

# ---- Multiple treatments -- binary + continuous (sim_multi) ---------
df_m <- sim_multi(
  n = 1000L, tmax = 3L, seed = 1L, n_binary = 1L, n_continuous = 1L
)
policy_multi <- function(D_block, t, a_names) {
  out <- D_block[, ..a_names, drop = FALSE]
  out[["A_b1"]] <- pmax(D_block[["A_b1"]], as.integer(D_block[["L1"]] > 1.0))
  out[["A_c1"]] <- pmin(D_block[["A_c1"]] + 0.2, 2.0)
```

```
  out
}
wr_m <- density_ratio(
  df = df_m, a_names = c("A_b1", "A_c1"), tmax = 3L,
  baseline = c("age", "sex"), tv_names = "L1",
  sl_g = sl_lib, k = 1L, inner_v = 2L, v = 2L, seed = 1L,
  id = "id", time = "time", policy_spec_fun = policy_multi
)
res_m <- sdr(
  df = df_m, weight_object = wr_m, tmax = 3L,
  id = "id", time = "time", alive = "alive", in_state = "in_state", y = "Y",
  baseline = c("age", "sex"), tv_names = "L1", a_names = c("A_b1", "A_c1"),
  sl_remain = sl_lib, sl_death = sl_lib,
  sl_recursive = sl_lib, sl_y = sl_lib,
  k = 1L, inner_v = 2L, parallel = FALSE, seed = 1L,
  policy_spec_fun = policy_multi
)
res_m$psi; res_m$se
```

`sim_bin` *Simulated ICU panel with a single binary treatment*

Description

Generates a longitudinal panel of ICU-style trajectories with one binary treatment (`A`), competing exit events (death vs discharge), and a binary outcome (`Y`). Intended for vignettes, examples, and quick smoke tests.

Usage

```
sim_bin(n = 2000L, tmax = 5L, seed = 1L, apply_policy = FALSE)
```

Arguments

`n`	Number of subjects.
`tmax`	Maximum follow-up (time-steps while in-state).
`seed`	Integer seed.
`apply_policy`	Logical. When `TRUE`, forces `A = 1` for subjects with `L1 > 1.0` at each time-step – used internally to generate Monte-Carlo truth for accuracy tests. Leave at `FALSE` for regular observed data.

Value

A `data.frame` with one row per subject-time. Columns: `id`, `time`, `age`, `sex`, `L1`, `L2`, `A`, `alive`, `in_state`, `Y`.

See Also

sim_cont(), sim_multi(), sdr(), itmle()

---

sim_cont *Simulated ICU panel with a single continuous treatment*

Description

Longitudinal ICU-style trajectories with one continuous treatment (A, bounded roughly in [0, 2]), competing exit events, and a binary outcome. Continuous-treatment analogue of sim_bin().

Usage

```
sim_cont(n = 2000L, tmax = 5L, seed = 1L, dose_shift = 0)
```

Arguments

n	Number of subjects.
tmax	Maximum follow-up.
seed	Integer seed.
dose_shift	Numeric dose increment applied to every time-step's observed A, capped at 2.0. Used internally to generate Monte-Carlo truth. Leave at 0 for regular observed data.

Value

A data.frame with columns id, time, age, L1, A, alive, in_state, Y.

See Also

sim_bin(), sim_multi(), sdr(), itmle()

---

sim_multi *Simulated ICU panel with multiple treatments (any mix of binary + continuous)*

Description

Longitudinal ICU-style trajectories with an arbitrary number of binary and/or continuous treatments, competing exit events, and a binary outcome. Generalises sim_bin() / sim_cont() to multi-treatment MTPs.

Usage

```
sim_multi(
  n = 2000L,
  tmax = 5L,
  seed = 1L,
  n_binary = 1L,
  n_continuous = 1L,
  apply_policy = FALSE
)
```

Arguments

n	Number of subjects.
tmax	Maximum follow-up.
seed	Integer seed.
n_binary	Number of binary treatments (>= 0).
n_continuous	Number of continuous treatments (>= 0).
apply_policy	Logical. When TRUE, forces every binary treatment to 1 for subjects with L1 > 1.0 and adds +0.2 (capped at 2.0) to every continuous treatment at each time-step – used internally to generate Monte-Carlo truth for accuracy tests. Leave at FALSE for regular observed data.

Details

Binary treatments are named A_b1, A_b2, ... (first n_binary columns); continuous treatments are named A_c1, A_c2, ... Each treatment enters the exit-event and outcome models with a coefficient scaled by 1 / sqrt(k) where k is the treatment index within its type, so adding more treatments does not blow up effect sizes.

Value

A data.frame with columns id, time, age, sex, L1, A_b1..A_b{n_binary}, A_c1..A_c{n_continuous}, alive, in_state, Y.

See Also

sim_bin(), sim_cont(), sdr(), itmle()

---

sl_itmle *Custom SuperLearner wrappers for the iTMLE targeting step*

Description

A family of SuperLearner-compatible learner functions designed for the iTMLE targeting step. They differ from standard wrappers in one critical way: the logit offset is passed as a **column** in `X` (named `._sl_offset`) rather than via the `offset` argument. This is necessary because SuperLearner's internal cross-validation subsetting drops the `offset` vector, whereas a column in `X` is correctly subset.

Each learner:

1. Extracts `._sl_offset` from `X`/`newX` via `extract_offset()`.
2. Fits a fluctuation model with that offset.
3. Returns predictions on the probability scale, clipped to `[bounds, 1-bounds]`.

Learners available:

`SL.tgt.empty` No fluctuation: returns `expit(offset)` unchanged. Acts as the "no update" option.

`SL.tgt.intercept` One-parameter intercept fluctuation (standard TMLE update).

`SL.tgt.glm` Main-terms logistic GLM fluctuation with offset.

`SL.tgt.glmnet` Penalised logistic regression (elastic net) with offset via `glmnet::cv.glmnet()`.

`SL.tgt.xgboost` Gradient boosted trees with offset via `base_margin`. CUDA-capable.

Pre-configured variants (`SL.tmle_*`) expose specific hyperparameter choices and are the recommended building blocks for `sl_tmle`:

`SL.tmle_empty` No fluctuation (pass-through).

`SL.tmle_intercept` One-parameter intercept (standard TMLE update).

`SL.tmle_glm` Main-terms logistic GLM.

`SL.tmle_glmnet_ridge` Ridge regression (alpha = 0).

`SL.tmle_glmnet_enet` Elastic net (alpha = 0.5).

`SL.tmle_glmnet_lasso` Lasso (alpha = 1).

`SL.tmle_xgb_d1` XGBoost depth 1, CPU, 50 rounds.

`SL.tmle_xgb_d3` XGBoost depth 3, CPU, 50 rounds.

`SL.tmle_xgb_d6` XGBoost depth 6, CPU, 50 rounds.

The default `sl_tmle` vector follows Luedtke et al. (2017) Algorithm 4: `empty`, `intercept`, `glm`, and the three XGBoost depths. The glmnet variants are available but must be specified explicitly via `sl_tmle`.

Usage

```
SL.tgt.empty(Y, X, newX, family, obsWeights, id, bounds = 1e-05, ...)

SL.tgt.intercept(Y, X, newX, family, obsWeights, id, bounds = 1e-05, ...)

SL.tgt.glm(Y, X, newX, family, obsWeights, id, bounds = 1e-05, ...)

SL.tgt.glmnet(
```

```
  Y,
  X,
  newX,
  family,
  obsWeights,
  id,
  alpha = 0.5,
  nfolds = 3L,
  s_select = "lambda.min",
  bounds = 1e-05,
  ...
)

SL.tgt.xgboost(
  Y,
  X,
  newX,
  family,
  obsWeights,
  id,
  nrounds = 100L,
  max_depth = 2L,
  eta = 0.1,
  subsample = 0.8,
  colsample_bytree = 0.8,
  max_delta_step = 0,
  nthread = 1L,
  use_cuda = TRUE,
  bounds = 1e-05,
  ...
)

SL.tmle_empty(Y, X, newX, family, obsWeights, id, ...)

SL.tmle_intercept(Y, X, newX, family, obsWeights, id, ...)

SL.tmle_glm(Y, X, newX, family, obsWeights, id, ...)

SL.tmle_glmnet_ridge(Y, X, newX, family, obsWeights, id, ...)

SL.tmle_glmnet_enet(Y, X, newX, family, obsWeights, id, ...)

SL.tmle_glmnet_lasso(Y, X, newX, family, obsWeights, id, ...)

SL.tmle_xgb_d1(Y, X, newX, family, obsWeights, id, ...)

SL.tmle_xgb_d3(Y, X, newX, family, obsWeights, id, ...)
```

```
SL.tmle_xgb_d6(Y, X, newX, family, obsWeights, id, ...)
```

Arguments

Y	Numeric outcome vector (on [0, 1] after scaling).
X	data.frame of covariates, **must** include a column named ._sl_offset containing the logit of the current Q estimate.
newX	data.frame for prediction, same structure as X.
family	Passed by SuperLearner; should be binomial().
obsWeights	Numeric vector of observation weights (the iTMLE clever covariate / density ratio product).
id	Subject identifiers (passed by SuperLearner, not used directly).
bounds	Numeric. Clipping bound for predictions. Default 1e-5.
...	Additional arguments (ignored).
alpha	Numeric. Elastic-net mixing parameter passed to glmnet::cv.glmnet(). 0 = ridge, 1 = lasso. Default 0.5. (SL.tgt.glmnet only.)
nfolds	Integer. Number of cross-validation folds for glmnet::cv.glmnet(). Default 3L. (SL.tgt.glmnet only.)
s_select	Character. Lambda selection rule for glmnet::cv.glmnet(): "lambda.min" or "lambda.1se". Default "lambda.min". (SL.tgt.glmnet only.)
nrounds	Integer. Number of boosting rounds passed to xgboost::xgb.train(). Default 100L. (SL.tgt.xgboost only.)
max_depth	Integer. Maximum tree depth for XGBoost. Default 2L. (SL.tgt.xgboost only.)
eta	Numeric. Learning rate for XGBoost. Default 0.1. (SL.tgt.xgboost only.)
subsample	Numeric. Row subsampling ratio for XGBoost. Default 0.8. (SL.tgt.xgboost only.)
colsample_bytree	Numeric. Column subsampling ratio for XGBoost. Default 0.8. (SL.tgt.xgboost only.)
max_delta_step	Numeric. Maximum delta step for XGBoost leaf weights. Values > 0 help stabilise logistic regression on imbalanced data. Default 0. (SL.tgt.xgboost only.)
nthread	Integer. Number of threads for XGBoost. Default 1L. (SL.tgt.xgboost only.)
use_cuda	Logical. Use CUDA GPU acceleration in XGBoost if available. Default TRUE. (SL.tgt.xgboost only.)

Value

A list with elements pred (numeric predictions on the probability scale) and fit (a fitted object with a predict method).

See Also

itmle()

Examples

```
# Default targeting stack (Luedtke et al. Algorithm 4)
sl_tmle

# Lightweight stack for examples and testing (no XGBoost required)
sl_tmle_light <- c("SL.tmle_empty", "SL.tmle_intercept", "SL.tmle_glm")

# Add glmnet variants explicitly when desired
sl_tmle_penalised <- c(
  "SL.tmle_empty", "SL.tmle_intercept", "SL.tmle_glm",
  "SL.tmle_glmnet_ridge", "SL.tmle_glmnet_enet",
  "SL.tmle_xgb_d1", "SL.tmle_xgb_d3", "SL.tmle_xgb_d6"
)
```

---

sl_tmle *Default SuperLearner library for the iTMLE targeting step*

Description

A character vector naming the default library of learners used in the iTMLE targeting step. Contains empty, intercept-only, GLM, and XGBoost wrappers at three tree depths. All wrappers carry the logit offset required for cross-validated TMLE as a data column (the "offset-as-column" convention) and clip internal probability predictions at `1e-5`.

Pass this vector to the `sl_target` argument of `itmle` or construct your own library from the `SL.tmle_*` wrappers exported by this package.

Usage

```
sl_tmle
```

Format

A character vector of SuperLearner learner names.

Value

A character vector of SuperLearner learner name strings to be passed to the `sl_target` argument of `itmle`.

See Also

`itmle`, `SL.tmle_glm`, `SL.tmle_glmnet_ridge`, `SL.tmle_xgb_d1`

`weight_diagnostics` *Per-time weight diagnostics for a density ratio object*

Description

Summarises the instantaneous density ratios and their cumulative products at each time step, restricted to subjects at risk at that time. This matches how the weights enter the SDR and iTMLE corrections: the EIF term at time t uses the product of ratios from t=1 through t, and only at-risk subjects contribute non-zero correction terms.

Usage

```
weight_diagnostics(weight_object, trim = 1)
```

Arguments

`weight_object`	Output of `density_ratio()`.
`trim`	Quantile used to cap `Rt_t` before computing summaries. Default `1` (no trimming) so the returned summaries reflect the raw density ratios; pass a value < `1` (e.g. `0.99`) to see how a given trim reshapes the weight distribution before choosing what to hand to `sdr()`/`itmle()`.

Details

Call with different `trim` values to assess trim sensitivity before passing the `weight_object` to `sdr()` or `itmle()`.

Value

A `data.table` with one row per time step and columns: `t`, `n_risk`; instantaneous ratio stats (`Rt_mean`, `Rt_median`, `Rt_p05`, `Rt_p95`, `Rt_max`, `Rt_ess`); cumulative product stats (`cum_mean`, `cum_median`, `cum_p05`, `cum_p95`, `cum_max`, `cum_ess`). ESS uses the Kish approximation: $(\sum w)^2 / \sum w^2$.

See Also

`density_ratio()`, `sdr()`, `itmle()`

Index