Package 'causalgenerics'

August 8, 2026

Title Shared Generics for the 'r-causal' Ecosystem

Version 0.1.0

Description A home for the S3 generics shared across the 'r-causal' ecosystem, including 'propensity', 'halfmoon', 'positively', and 'balancing'. Owning the generic definitions in one place lets those packages register methods without masking one another when several are attached at once. The generics cover inverse probability weighted estimation, effective sample size, and the metadata carried by causal weight vectors, and the package supplies the abstract causal weight class and the result class those estimates are returned in, together with the methods both classes carry, so that the ecosystem packages inherit them rather than writing their own. The design follows that of the 'generics' package, which provides commonly used S3 generics for the same purpose: letting packages share a single definition instead of each defining its own.

License MIT + file LICENSE

URL https://github.com/r-causal/causalgenerics, https://r-causal.github.io/causalgenerics/

BugReports https://github.com/r-causal/causalgenerics/issues

Depends R (>= 4.1.0)

Imports stats, vctrs

Suggests testthat (>= 3.2.0), withr

Config/roxygen2/version 8.0.0

Config/testthat/edition 3

Encoding UTF-8

NeedsCompilation no

Author Malcolm Barrett [aut, cre, cph] (ORCID: <https://orcid.org/0000-0003-0299-5825>)

Maintainer Malcolm Barrett <malcolmbarrett@gmail.com>

Repository CRAN

Date/Publication 2026-08-08 12:20:07 UTC

Contents

`causal-weights` *Causal weight accessors*

Description

Generics for inspecting and setting the metadata carried by causal weight vectors.

- `is_causal_wt()` tests whether an object is a causal weight vector.
- `estimand()` returns the causal estimand the weights target, such as `"ate"` or `"att"`.
- `estimand<-()` sets the causal estimand.

Usage

```
is_causal_wt(x, ...)

estimand(x, ...)

estimand(x, ...) <- value
```

Arguments

`x`	An object to inspect or modify. These generics dispatch on this argument.
`...`	Arguments passed to methods.
`value`	The causal estimand to assign.

Details

This package defines the generics and the abstract weight class they are written against. It supplies a method for each generic on `causal_wts`, so a concrete class built with `new_causal_wts()` inherits all three. The concrete classes themselves live in the packages that own them, such as `propensity`, as do methods for any object that is not a `causal_wts` vector. `is_causal_wt()` defaults to `FALSE` so that any object that is not a recognized causal weight vector reports as much; `estimand()` and `estimand<-()` have no meaningful default and signal an error when no method is registered for the object.

Value

`is_causal_wt()` returns a single logical value. `estimand()` returns the estimand, typically a character string, or `NULL` when none is recorded. `estimand<-()` returns `x` with the estimand updated.

See Also

`new_causal_wts()` for the class these generics have methods for, and the `propensity` package for concrete weight classes.

Examples

```
wts <- new_causal_wts(
  c(1.2, 0.8, 1.5),
  subclass = "cg_toy_wts",
  estimand = "ate"
)

is_causal_wt(wts)
estimand(wts)

# The estimand is metadata on the vector, so it can be replaced in place.
estimand(wts) <- "att"
estimand(wts)

# An object that is not a causal weight vector reports `FALSE` rather than
# erroring.
is_causal_wt(1:3)

# `estimand()` has no meaningful default, so it errors instead.
try(estimand(1:3))
```

---

causal_wts_ptype2 *Common type of two causal weight vectors of the same class*

Description

`causal_wts_ptype2()` supplies the one coercion rule that concrete weight classes share: two causal weight vectors combine only when their estimands are identical, and otherwise the common type is a plain double. Concrete classes call it from inside their own `vctrs::vec_ptype2()` method for the pairing of two vectors of that class.

Usage

```
causal_wts_ptype2(x, y, ptype, on_downgrade)
```

Arguments

x, y	The two weight vectors being combined, as passed to vec_ptype2().
ptype	The common type to return when the estimands agree. Returned exactly as supplied and never inspected, and evaluated only on that path.
on_downgrade	A function of no arguments, called once when the estimands differ. Its value is discarded; it exists so the caller can signal a condition of its own choosing.

Details

This is a helper rather than a method, and it cannot become one. vctrs resolves vec_ptype2() with an exact lookup on class(x)[[1]] and never walks the rest of the class vector, so a method registered on causal_wts is never found for a concrete subclass. Each package therefore keeps its own vec_ptype2.<cls>.<cls> method and calls this from inside it:

```
vec_ptype2.bw.bw <- function(x, y, ...) {
  causal_wts_ptype2(
    x,
    y,
    new_bw(estimand = estimand(x)),
    on_downgrade = function() warn_bw_downgrade("bw")
  )
}
```

The estimands are read through estimand(), so a class that computes its estimand rather than storing it is compared on the value its method returns. Two vectors that record no estimand are compatible, which is the ordinary case for continuous-exposure weights rather than an edge case.

ptype is evaluated only on the compatible path. A caller's prototype is usually a live constructor call over metadata the two vectors do not agree on, so the downgrade path never forces it.

The condition belongs to the caller. This function signals nothing of its own: on_downgrade() runs exactly once when the estimands differ, and whatever it signals reaches the user unchanged. The concrete classes downstream disagree on the class of that condition and on whether the downgrade warns at all, so choosing one here would break them.

The scope is one pairing and one rule, which is as much as the concrete classes share. A class that checks nothing beyond the estimand hands over its whole method, as balancing does. A class that checks further metadata can use this for the estimand alone and keep its own control flow: propensity compares five more fields after the estimand, each with its own message and its own bail-out. Such a caller can pass a sentinel as ptype and carry on with its remaining checks when that sentinel comes back.

Value

ptype when estimand(x) and estimand(y) are identical, and double() otherwise.

See Also

new_causal_wts() for the class this rule is written against, and estimand() for the accessor it compares.

Examples

```
ate <- new_causal_wts(c(1.2, 0.8), subclass = "my_wts", estimand = "ate")
att <- new_causal_wts(c(1.0, 1.5), subclass = "my_wts", estimand = "att")

# Matching estimands give back the prototype the caller built.
causal_wts_ptype2(
  ate,
  ate,
  new_causal_wts(subclass = "my_wts", estimand = estimand(ate)),
  on_downgrade = function() message("the estimands differ")
)

# Differing estimands run the callback and drop to a bare double.
causal_wts_ptype2(
  ate,
  att,
  new_causal_wts(subclass = "my_wts", estimand = estimand(ate)),
  on_downgrade = function() message("the estimands differ")
)
```

---

ess *Effective sample size*

Description

`ess()` is the generic for the effective sample size, a summary of how much information a set of weights retains relative to an unweighted sample. When weights vary substantially, the effective sample size can be much smaller than the number of observations.

Usage

```
ess(x, ...)

## Default S3 method:
ess(x, na.rm = FALSE, ...)
```

Arguments

`x`	An object to compute the effective sample size for, such as a weight vector or a fitted model. `ess()` dispatches on this argument.
`...`	Arguments passed to methods.
`na.rm`	Should missing weights be dropped before computing? Missing weights otherwise make the result `NA`.

Details

The default method computes the Kish effective sample size, $(\sum w)^2 / \sum w^2$, for any numeric vector. Every weight class in this ecosystem is a double underneath, and `is.numeric()` is `TRUE` for one, so the default is the whole implementation for weights and a concrete weight class needs no method of its own. A method would be worth registering only for an object that needs something other than the Kish formula, such as a fitted model, where it might summarize by group and return a tibble with one row per group.

The default errors when `x` is not numeric, which includes `NULL`, rather than returning a value computed from nothing. Numeric input whose quotient is `0 / 0` still returns `NaN`, which covers a zero-length vector and weights that are all zero: both make the sum and the sum of squares zero, and neither has an effective sample size to report.

The generic stays minimal, taking the object and `...`, so that a method declares whatever further arguments it needs. A fitted-model method, for example, takes the variable to group by that way.

Value

The default method returns a single number. Other methods return whatever suits the object they are written for; a fitted-model method may return a tibble.

See Also

`halfmoon::check_ess()`, which reports the effective sample size of one or more weighting columns in a data frame, optionally by exposure group. It is an ordinary function rather than an `ess()` method, so it does not go through this generic.

Examples

```
# Equal weights carry as much information as an unweighted sample.
ess(rep(0.5, 10))

# Weights that vary carry much less.
ess(c(rep(0.5, 9), 10))
```

---

ipw *Inverse probability weighted estimation*

Description

`ipw()` is the generic for bring-your-own-model inverse probability weighted estimation of causal effects. A method takes a fitted weighting or propensity score model together with a fitted weighted outcome model and returns causal effect estimates with standard errors that account for the two-step estimation process.

Usage

```
ipw(wt_mod, outcome_mod, ...)
```

Arguments

wt_mod	The weighting object that produced the weights, for example a fitted propensity score model. ipw() dispatches on this argument.
outcome_mod	A fitted weighted outcome model.
...	Arguments passed to methods.

Details

This package defines the generic and the shared result class its methods return. The methods themselves live in the packages that own the relevant model classes, such as propensity for propensity score models and balancing for balancing weight fits. Dispatch is on wt_mod, the weighting object; each method documents the outcome model classes and further arguments it accepts.

A method builds its return value with new_ipw() rather than a result object of its own. The field names and their order are a cross-package contract, so that an IPW estimate reads the same way whichever package produced it, and constructing through new_ipw() is also what gives a method the shared print() and as.data.frame() methods.

Value

An object of class ipw, holding the causal effect estimates and their standard errors alongside the models they came from. See new_ipw() for the components and their order.

See Also

new_ipw(), the constructor every method returns through, and the propensity and balancing packages for methods.

Examples

```
# A confounded toy data set: the confounder `x` raises both the chance of
# exposure `z` and the risk of outcome `y`. Within either level of `x` the
# risk difference is 0.25.
dat <- data.frame(
  x = rep(c(0, 1), each = 10),
  z = c(0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0),
  y = c(1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0)
)

# The crude comparison ignores `x` and overstates that risk difference.
coef(lm(y ~ z, data = dat))["z"]

# A weighting object of a toy class, standing in for the fitted weighting
# models that the methods in other packages dispatch on. Weighting by the
# inverse probability of the exposure received breaks the dependence of `z`
# on `x`, so the weighted outcome model recovers the 0.25 risk difference.
ps <- fitted(glm(z ~ x, family = binomial(), data = dat))
wt_mod <- structure(
  list(wts = ifelse(dat$z == 1, 1 / ps, 1 / (1 - ps))),
  class = "cg_toy_model"
)
```

```
# A method for that class. It reads the exposure coefficient of the weighted
# outcome model as a risk difference and returns through `new_ipw()`, which is
# what every `ipw()` method does. Inference is normal-based throughout, as the
# `z` column of the estimates contract calls for. The standard error is the
# model-based one the weighted fit reports; a real method uses a variance
# estimator that also accounts for the weights having been estimated.
ipw.cg_toy_model <- function(wt_mod, outcome_mod, ...) {
  coefs <- summary(outcome_mod)$coefficients["z", ]
  estimate <- coefs[["Estimate"]]
  std_err <- coefs[["Std. Error"]]
  z <- estimate / std_err
  half_width <- qnorm(0.975) * std_err
  new_ipw(
    estimand = "ate",
    wt_mod = wt_mod,
    outcome_mod = outcome_mod,
    estimates = data.frame(
      effect = "rd",
      estimate = estimate,
      std.err = std_err,
      z = z,
      ci.lower = estimate - half_width,
      ci.upper = estimate + half_width,
      conf.level = 0.95,
      p.value = 2 * pnorm(-abs(z))
    ),
    se_method = "model-based",
    fit = NULL
  )
}

outcome_mod <- lm(y ~ z, data = dat, weights = wt_mod$wts)

# Dispatch on the weighting object finds the method.
ipw(wt_mod, outcome_mod)
```

`new_causal_wts` *Construct a causal weight vector*

Description

`new_causal_wts()` is the low-level constructor for the abstract weight class that the r-causal ecosystem shares. It is intended for package developers building a concrete weight class, not for end users, and it assumes `x` is already a double vector.

Usage

```
new_causal_wts(x = double(), subclass, ...)
```

Arguments

x	A double vector of weights.
subclass	A single non-empty string naming the concrete weight class.
...	Metadata to store as attributes on the result. A NULL value leaves the attribute absent.

Details

Weights built here are vctrs vectors over a double, so the class vector is `c(subclass, "causal_wts", "vctrs_vctr", "double")`. Callers may rely on that shape and on its order: `subclass` first, so that methods for the concrete class take precedence, then `causal_wts`, so that the concrete class inherits every method registered against the abstract one. `subclass` must therefore be a single non-empty string. Anything else would change the length of the class vector or name a class that no method can be registered against.

The dots carry whatever metadata the concrete class records, such as the estimand or the exposure groups. This constructor neither names nor types those fields. A field passed as `NULL` leaves its attribute absent rather than recording a `NULL`, which is what lets a concrete constructor default its metadata to `NULL` and pass it straight through.

The abstract layer reaches well past construction. A concrete class inherits the metadata accessors `is_causal_wt()`, `estimand()`, and `estimand<-()`, and it inherits every read-only operation that names no metadata field and so gives the answer the underlying double would: `vec_math()`, the `Summary` group generic, `min()`, `max()`, `range()`, `median()`, `quantile()`, `summary()`, `anyDuplicated()`, `diff()`, `[`, and the six comparison operators. `ess()` arrives as well, through a default that computes the Kish effective sample size for any numeric vector.

Do not write any of those again for a concrete class. A method registered on the subclass takes precedence over the shared one, and the comparison operators in particular do more than delegate: they short-circuit `vec_equal()` and `vec_compare()`, which would otherwise route an expression such as `weights > 0` through the concrete class's own `vec_ptype2()` method and signal whatever that method signals on a downgrade, once per call. `glm.fit()` evaluates comparisons of that shape repeatedly inside `profile.glm()`, so a single profiled confidence interval on a weighted fit would emit the same warning a hundred times over.

What a concrete class does still own is its `vec_ptype2()`, `vec_cast()`, `vec_ptype_abbr()`, `vec_ptype_full()`, `vec_restore()`, and `vec_arith()` methods, for three distinct reasons.

`vec_ptype2()` and `vec_cast()` cannot be inherited at all. vctrs resolves them through `s3_method_specific()`, which keys on `class(x)[[1]]` and never walks the rest of the class vector, so a `causal_wts` method is never found.

`vec_ptype_abbr()` and `vec_ptype_full()` are likewise not reached from the abstract class. A concrete class that omits either one falls back to the vctrs default and prints a bare class name where the estimand should appear.

`vec_restore()` and `vec_arith()` do dispatch through `causal_wts`, but they still have to be written for the concrete class, because they reconcile metadata that this package knows nothing about. An abstract `vec_restore()` that copies attributes wholesale errors on named weights and re-attaches index-typed metadata at the wrong length after slicing, and an abstract `vec_arith()` would silently legalize arithmetic between two different concrete weight classes.

Value

A vector of class `c(subclass, "causal_wts", "vctrs_vctr", "double")`.

See Also

`estimand()` and `is_causal_wt()`, the accessors this class supplies methods for, and `causal_wts_ptype2()`, the coercion rule a concrete class calls from its own `vec_ptype2()` method.

Examples

```
wts <- new_causal_wts(c(1.2, 0.8), subclass = "my_wts", estimand = "ate")
class(wts)
estimand(wts)
```

---

new_ipw *Construct an inverse probability weighted result*

Description

`new_ipw()` is the low-level constructor for the object that every `ipw()` method returns. It is intended for package developers writing an `ipw()` method, not for end users, and it assumes its arguments are already validated.

Usage

```
new_ipw(estimand, wt_mod, outcome_mod, estimates, se_method, fit)

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

## S3 method for class 'ipw'
as.data.frame(x, row.names = NULL, optional = NULL, exponentiate = FALSE, ...)
```

Arguments

`estimand`	The causal estimand the method targeted, such as `"ate"` or `"att"`.
`wt_mod`	The weighting object: the fitted model that produced the weights.
`outcome_mod`	The fitted weighted outcome model.
`estimates`	A data frame of effect estimates, in the shape the return value describes.
`se_method`	The standard error method that ran, such as `"mestimation"` or `"linearization"`.
`fit`	The fitted variance object, or `NULL` when the method has none.
`x`	An `ipw` object.
`...`	Further arguments. `print()` ignores them; `as.data.frame()` passes them to `base::as.data.frame()`.

`row.names, optional`
: Passed to `base::as.data.frame()`.

`exponentiate`
: If `TRUE`, exponentiate the log risk ratio and log odds ratio to produce risk ratios and odds ratios on their natural scale. The confidence interval bounds are also exponentiated. Standard errors, z statistics, and p-values remain on the log scale. Default is `FALSE`.

Details

The result layer is shared so that an IPW estimate reads the same way whichever package produced it. A package supplying an `ipw()` method builds its return here and inherits the `print()` and `as.data.frame()` methods registered against the class rather than writing its own. Two packages each defining `print.ipw()` would collide in the shared S3 method table, which is the situation this package exists to prevent.

The field names and their order are part of the contract, since callers read fields by name and print the object positionally. `fit` is present on every path, including the ones that have no fitted variance object to report.

`as.data.frame()` returns the `estimates` component. With `exponentiate = TRUE` it moves the `log(rr)` and `log(or)` rows to their natural scale, exponentiating the point estimate and the confidence limits and relabelling the two effects `"rr"` and `"or"`. Standard errors, z statistics, and p-values stay on the log scale, where the inference is done.

Value

`new_ipw()` returns an S3 object of class `ipw`: a list of the following six components, in this order.

`estimand` The causal estimand, such as `"ate"` or `"att"`.

`wt_mod` The weighting object: the fitted model that produced the weights.

`outcome_mod` The fitted outcome model.

`estimates` A data frame with one row per effect measure and the following columns: `effect` (the measure name), `estimate` (point estimate), `std.err` (standard error), `z` (z-statistic), `ci.lower` and `ci.upper` (confidence interval bounds), `conf.level`, and `p.value`. For a categorical exposure the data frame also has a `comparison` column, placed after `effect`, naming the non-reference level and reference level of each contrast.

`se_method` The standard error method used, such as `"mestimation"` or `"linearization"`.

`fit` The fitted object the variance estimator produced, or `NULL`. A method that stacks estimating equations records the M-estimator here; the linearization path has no such object and records `NULL`.

`print()` returns its input invisibly. `as.data.frame()` returns the `estimates` component as a data frame.

See Also

`ipw()`, the generic these results come from.

Examples

```
dat <- data.frame(
  x = rep(c(-1.5, -0.5, 0.5, 1.5), each = 5),
  z = rep(c(0, 1), 10),
  y = rep(c(0, 1, 1, 0, 1), 4)
)

# Written out literally, in the shape the return contract documents. These
# stand in for what an `ipw()` method would compute from the models below.
estimates <- data.frame(
  effect = c("rd", "log(rr)", "log(or)"),
  estimate = c(0.199882, 0.560414, 0.878313),
  std.err = c(0.092425, 0.273519, 0.418661),
  z = c(2.1626, 2.0489, 2.0979),
  ci.lower = c(0.018732, 0.024326, 0.057753),
  ci.upper = c(0.381032, 1.096502, 1.698873),
  conf.level = 0.95,
  p.value = c(0.030570, 0.040470, 0.035910)
)

res <- new_ipw(
  estimand = "ate",
  wt_mod = glm(z ~ x, family = binomial(), data = dat),
  outcome_mod = glm(y ~ z, family = quasibinomial(), data = dat),
  estimates = estimates,
  se_method = "linearization",
  fit = NULL
)

res

# The ratios on their natural scale.
as.data.frame(res, exponentiate = TRUE)
```

Index