Package 'crossfit'

February 19, 2026

Title Cross-Fitting Engine for Double/Debiased Machine Learning

Version 0.1.1

Description Provides a general cross-fitting engine for double / debiased machine learning and other meta-learners. The core functions implement flexible graphs of nuisance models with per-node training fold widths, target-specific evaluation windows, and several fold allocation schemes (``independence'', ``overlap'', ``disjoint''). The engine supports both numeric estimators (mode = ``estimate'') and cross-fitted prediction functions (mode = ``predict''), with configurable aggregation over panels and repetitions.

License GPL-3

URL https://github.com/EtiennePeyrot/crossfit-R

BugReports https://github.com/EtiennePeyrot/crossfit-R/issues

Encoding UTF-8

RoxygenNote 7.3.1

Depends R (>= 4.1.0)

Imports stats, utils

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Author Etienne Peyrot [aut, cre] (ORCID: <https://orcid.org/0009-0006-8520-6201>)

Maintainer Etienne Peyrot <etienne.peyrot@inserm.fr>

Repository CRAN

Date/Publication 2026-02-19 20:00:08 UTC

Contents

`crossfit-package` *crossfit: Cross-Fitting Engine for Double / Debiased Machine Learning*

Description

Provides a general cross-fitting engine for double / debiased machine learning and other meta-learners. The core functions implement flexible graphs of nuisance models with per-node training fold widths, target-specific evaluation windows, and several fold allocation schemes ("independence", "overlap", "disjoint"). The engine supports both numeric estimators (mode = "estimate") and cross-fitted prediction functions (mode = "predict"), with configurable aggregation over panels and repetitions.

Author(s)

Maintainer: Etienne Peyrot <etienne.peyrot@inserm.fr> (ORCID)

See Also

Useful links:

- https://github.com/EtiennePeyrot/crossfit-R
- Report bugs at https://github.com/EtiennePeyrot/crossfit-R/issues

---

`create_method` *Create a cross-fitting method specification*

Description

Helper to create a method specification for `crossfit` / `crossfit_multi`. A method bundles together:

- a target functional `target()`,
- a named list of nuisance specifications,
- cross-fitting geometry (`folds`, `repeats`, `eval_fold`, `mode`, `fold_allocation`),
- and panel / repetition aggregation functions.

Usage

```
create_method(
  target,
  list_nuisance = NULL,
  folds,
  repeats,
  mode = c("estimate", "predict"),
  eval_fold = if (mode == "estimate") 1L else 0L,
  fold_allocation = c("independence", "overlap", "disjoint"),
  aggregate_panels = NULL,
  aggregate_repeats = NULL
)
```

Arguments

`target`	A function representing the target functional. It must accept nuisance predictions as arguments (named after nuisances) and optionally a `data` argument.
`list_nuisance`	Optional named list of nuisance specifications created by `create_nuisance`.
`folds`	Positive integer giving the number of folds K. May be `NULL`, in which case `crossfit_multi` will infer a minimal feasible `K` from the dependency structure.
`repeats`	Positive integer giving the number of repetitions.
`mode`	Cross-fitting mode. Either `"estimate"` (target returns numeric estimates) or `"predict"` (target returns a cross-fitted predictor).
`eval_fold`	Integer giving the width (in folds) of the evaluation window for the target. Must be `> 0` for `mode = "estimate"` and `0` for `mode = "predict"`. If omitted, the default is `1L` for `"estimate"` and `0L` for `"predict"`.
`fold_allocation`	Fold allocation strategy; one of `"independence"`, `"overlap"`, or `"disjoint"`.
`aggregate_panels`	Aggregation function for panel-level results, typically one of `mean_estimate`, `median_estimate`, `mean_predictor`, `median_predictor`, or a custom function. May be `NULL`, in which case a global default can be supplied via `crossfit_multi`.
`aggregate_repeats`	Aggregation function for repetition-level results, typically one of `mean_estimate`, `median_estimate`, `mean_predictor`, `median_predictor`, or a custom function. May be `NULL`, in which case a global default can be supplied via `crossfit_multi`.

Details

The returned list is validated by `validate_method()` to ensure structural soundness, but the validated object is not stored: you are free to modify the returned method before passing it to `crossfit` or `crossfit_multi`.

By default, `eval_fold` is chosen to be `1L` when `mode = "estimate"` and `0L` when `mode = "predict"`. If you override `eval_fold`, it must satisfy these constraints: positive integer for `"estimate"`, zero for `"predict"`.

Value

A method specification list suitable for use in `crossfit` or `crossfit_multi`.

Examples

```
set.seed(1)
n <- 50
x <- rnorm(n)
y <- x + rnorm(n)

# Nuisance: regression for E[Y | X]
nuis_y <- create_nuisance(
  fit = function(data, ...) lm(y ~ x, data = data),
  predict = function(model, data, ...) predict(model, newdata = data)
)

# Target: mean squared error of the nuisance predictor
target_mse <- function(data, nuis_y, ...) {
  mean((data$y - nuis_y)^2)
}

m <- create_method(
  target = target_mse,
  list_nuisance = list(nuis_y = nuis_y),
  folds = 2,
  repeats = 1,
  eval_fold = 1L,
  mode = "estimate",
  fold_allocation = "independence",
  aggregate_panels  = mean_estimate,
  aggregate_repeats = mean_estimate
)

str(m)
```

`create_nuisance` *Create a nuisance specification*

Description

Helper to create a nuisance specification with basic structural checks. A nuisance is defined by a `fit` function, a `predict` function, and optional dependency mappings.

Usage

```
create_nuisance(
  fit,
  predict,
  train_fold = 1L,
```

```
  fit_deps = NULL,
  pred_deps = NULL
)
```

Arguments

fit	A function fit(data, ...) that trains the nuisance model on a subset of the data and returns a fitted model object.
predict	A function predict(model, data, ...) that returns predictions for the nuisance on new data.
train_fold	Positive integer giving the width (in folds) of the training window used for this nuisance. Defaults to 1L.
fit_deps	Optional named character vector mapping fit() argument names to nuisance names, used to specify nuisance inputs to the fit function. If NULL, the dependencies are inferred later from required arguments whose names match nuisance names.
pred_deps	Optional named character vector mapping predict() argument names to nuisance names, used to specify nuisance inputs to the predict function.

Value

A list representing a nuisance specification, suitable for inclusion in the list_nuisance argument of create_method.

Examples

```
# Simple linear regression nuisance: E[Y | X]
set.seed(1)
n <- 50
x <- rnorm(n)
y <- x + rnorm(n)

nuis <- create_nuisance(
  fit = function(data, ...) lm(y ~ x, data = data),
  predict = function(model, data, ...) predict(model, newdata = data)
)

str(nuis)
```

---

crossfit *Cross-fitting for a single method*

Description

Convenience wrapper around crossfit_multi for the common case of a single method. It enforces that method is a single method specification and forwards the aggregation functions stored inside method.

Usage

```
crossfit(
  data,
  method,
  fold_split = function(data, K) sample(rep_len(1:K, nrow(data))),
  seed = NULL,
  max_fail = Inf,
  verbose = FALSE
)
```

Arguments

data	Data frame or matrix with the observations.
method	A single method specification (list) created by create_method. It must contain a target function and aggregate_panels / aggregate_repeats must be functions.
fold_split	A function producing a K-fold split of the data (see crossfit_multi).
seed	Integer base random seed.
max_fail	Non-negative integer or Inf controlling how many repetitions the method may fail before being disabled.
verbose	Logical; if TRUE, prints a compact status line per repetition.

Value

The same structure as crossfit_multi, but with a single method named "method". The final estimate is in $estimates$method.

Examples

```
set.seed(1)
n <- 100
x <- rnorm(n)
y <- x + rnorm(n)

data <- data.frame(x = x, y = y)

# Nuisance: E[Y | X]
nuis_y <- create_nuisance(
  fit = function(data, ...) lm(y ~ x, data = data),
  predict = function(model, data, ...) predict(model, newdata = data)
)

# Target: mean squared error of the nuisance predictor
target_mse <- function(data, nuis_y, ...) {
  mean((data$y - nuis_y)^2)
}

method <- create_method(
  target = target_mse,
```

```
  list_nuisance = list(nuis_y = nuis_y),
  folds = 2,
  repeats = 2,
  eval_fold = 1L,
  mode = "estimate",
  fold_allocation = "independence",
  aggregate_panels  = mean_estimate,
  aggregate_repeats = mean_estimate
)

cf <- crossfit(data, method)
cf$estimates
```

`crossfit_multi` *Cross-fitting for multiple methods*

Description

Runs cross-fitting for one or more methods defined via `create_method` and `create_nuisance`. This is the main engine that:

- validates and normalizes method specifications,
- builds the global instance graph and fold geometry,
- repeatedly draws K-fold splits and evaluates all active methods,
- aggregates results across panels and repetitions.

Usage

```
crossfit_multi(
  data,
  methods,
  fold_split = function(data, K) sample(rep_len(1:K, nrow(data))),
  seed = NULL,
  aggregate_panels = identity,
  aggregate_repeats = identity,
  max_fail = Inf,
  verbose = FALSE
)
```

Arguments

`data`	Data frame or matrix of size $n \times p$ containing the observations.
`methods`	A (named) list of method specifications, typically created with `create_method`.
`fold_split`	A function of the form `function(data, K)` returning a vector of length `nrow(data)` with integer fold labels in `1:K`. It must assign at least one observation to each fold.

`seed` Integer base random seed used for the K-fold splits; each repetition uses `seed + rep_id - 1`.

`aggregate_panels` Function used as the *default* aggregator over panels (folds) for each method. It is applied to the list of per-panel values. Methods can override this via their own `aggregate_panels`.

`aggregate_repeats` Function used as the *default* aggregator over repetitions for each method. It is applied to the list of per-repetition aggregated values. Methods can override this via their own `aggregate_repeats`.

`max_fail` Non-negative integer or `Inf` controlling how many repetitions a method is allowed to fail before being disabled. Structural model failures and panel-level errors both count toward this limit.

`verbose` Logical; if `TRUE`, prints a compact status line per repetition.

Details

Each method can operate in either `mode = "estimate"` (target returns numeric values) or `mode = "predict"` (target returns a prediction function). Cross-fitting ensures that nuisance models are always trained on folds disjoint from the folds on which their predictions are used in the target.

Value

A list with components:

`estimates` Named list of final estimates per method (after aggregating over panels and repetitions).

`per_method` For each method, a list with `values` (per-repetition aggregated results) and `errors` (error traces).

`repeats_done` Number of repetitions successfully completed for each method.

`K` Number of folds used in the plan.

`K_required` Per-method minimal required K based on their dependency structure.

`methods` The validated and normalized method specifications.

`plan` The cross-fitting plan produced by `build_instances()`.

Examples

```
set.seed(1)
n <- 100
x <- rnorm(n)
y <- x + rnorm(n)

data <- data.frame(x = x, y = y)

# Shared nuisance: E[Y | X]
nuis_y <- create_nuisance(
  fit = function(data, ...) lm(y ~ x, data = data),
  predict = function(model, data, ...) predict(model, newdata = data)
)
```

```
# Method 1: MSE of nuisance predictor
target_mse <- function(data, nuis_y, ...) {
  mean((data$y - nuis_y)^2)
}

# Method 2: mean fitted value
target_mean <- function(data, nuis_y, ...) {
  mean(nuis_y)
}

m1 <- create_method(
  target = target_mse,
  list_nuisance = list(nuis_y = nuis_y),
  folds = 2,
  repeats = 2,
  eval_fold = 1L,
  mode = "estimate",
  fold_allocation = "independence"
)

m2 <- create_method(
  target = target_mean,
  list_nuisance = list(nuis_y = nuis_y),
  folds = 2,
  repeats = 2,
  eval_fold = 1L,
  mode = "estimate",
  fold_allocation = "overlap"
)

cf_multi <- crossfit_multi(
  data    = data,
  methods = list(mse = m1, mean = m2),
  aggregate_panels  = mean_estimate,
  aggregate_repeats = mean_estimate
)

cf_multi$estimates
```

`mean_estimate` *Aggregators for scalar estimates*

Description

These helpers implement simple aggregation schemes for panel-level and repetition-level estimates in `crossfit` and `crossfit_multi`.

Usage

```
mean_estimate(xs)

median_estimate(xs)
```

Arguments

xs	A list of numeric values or numeric vectors. Elements are unlisted and concatenated prior to aggregation, so xs may contain scalars or length-k vectors.

Details

In mode = "estimate", each repetition typically produces a list of numeric values (one per evaluation panel). The functions mean_estimate() and median_estimate() aggregate such lists into a single numeric value.

Value

A single numeric value (the mean or median of all entries in xs.

Examples

```
xs <- list(c(1, 2, 3), 4, c(5, 6))
mean_estimate(xs)
xs <- list(c(1, 100), 10, 20)
median_estimate(xs)
```

---

mean_predictor *Aggregators for cross-fitted predictors*

Description

These helpers aggregate several cross-fitted predictors into a single ensemble predictor. They are designed for methods run with mode = "predict" in crossfit and crossfit_multi.

Usage

```
mean_predictor(fs)

median_predictor(fs)
```

Arguments

fs	A list of prediction functions. Each function must accept at least a newdata argument and return a numeric vector of predictions of the same length as nrow(newdata).

Value

A function of the form `function(newdata, ...)`, which returns a numeric vector of predictions. If `fs` is empty, the returned function always returns `numeric(0)`.

Examples

```
# Two simple prediction functions of x
f1 <- function(newdata, ...) newdata$x
f2 <- function(newdata, ...) 2 * newdata$x

ens_mean   <- mean_predictor(list(f1, f2))

newdata <- data.frame(x = 1:5)
ens_mean(newdata)
# Two simple prediction functions of x
f1 <- function(newdata, ...) newdata$x
f2 <- function(newdata, ...) 2 * newdata$x

ens_median <- median_predictor(list(f1, f2))

newdata <- data.frame(x = 1:5)
ens_median(newdata)
```

Index