

Package ‘crossfit’

August 27, 2026

Title A Graph-Based Cross-Fitting Engine in R

Version 0.1.4

Description Provides a general cross-fitting engine for semiparametric estimation (e.g., double/debiased machine learning). Supports user-defined target functionals and directed acyclic graphs of nuisance learners with per-node training fold widths, target-specific evaluation windows, and fold-allocation modes (``overlap'', ``disjoint'', ``independence''). Returns either numeric estimates (mode = ``estimate'') or cross-fitted prediction functions (mode = ``predict''), with configurable aggregation over panels and repetitions, reuse-aware caching, and failure isolation, making it well-suited for simulation studies and large benchmarks.

License GPL-3

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

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

Encoding UTF-8

Depends R (>= 4.1.0)

Imports stats, utils

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

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 2026-08-27 13:10:02 UTC

Contents

crossfit-package	2
create_method	3
create_nuisance	5
crossfit	6
crossfit_failure_control	7
crossfit_multi	8
mean_estimate	11
mean_predictor	11
Index	13

crossfit-package	<i>crossfit: Cross-Fitting Engine for Double / Debiased Machine Learning</i>
------------------	--

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](#))

Authors:

- 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`),
- panel / repetition aggregation functions,
- and method-specific failure handling.

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,
  failure_control = crossfit_failure_control()
)
```

Arguments

<code>target</code>	A function representing the target functional. It must accept nuisance predictions as arguments (named after nuisances) and optionally a data argument.
<code>list_nuisance</code>	Optional named list of nuisance specifications created by <code>create_nuisance</code> .
<code>folds</code>	Positive integer giving the number of folds K . May be <code>NULL</code> , in which case <code>crossfit_multi</code> will infer a minimal feasible K from the dependency structure.
<code>repeats</code>	Positive integer giving the number of repetitions.
<code>mode</code>	Cross-fitting mode. Either "estimate" (target returns numeric estimates) or "predict" (target returns a cross-fitted predictor).
<code>eval_fold</code>	Integer giving the width (in folds) of the evaluation window for the target. Must be > 0 for <code>mode = "estimate"</code> and 0 for <code>mode = "predict"</code> . If omitted, the default is <code>1L</code> for "estimate" and <code>0L</code> for "predict".
<code>fold_allocation</code>	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](#).

failure_control
 A named list created by [crossfit_failure_control](#), or a partial named list of its fields. Missing fields use their documented defaults.

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,
    failure_control = crossfit_failure_control(
      fail_repetition_on_error = FALSE
    )
  )
)

str(m)

```

create_nuisance	<i>Create a nuisance specification</i>
-----------------	--

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 <code>fit(data, ...)</code> that trains the nuisance model on a subset of the data and returns a fitted model object.
predict	A function <code>predict(model, data, ...)</code> 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 <code>fit()</code> argument names to nuisance names, used to specify nuisance inputs to the <code>fit</code> function. If <code>NULL</code> , the dependencies are inferred later from required arguments whose names match nuisance names.
pred_deps	Optional named character vector mapping <code>predict()</code> argument names to nuisance names, used to specify nuisance inputs to the <code>predict</code> 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

Runs one method specification and returns its estimate, per-repetition results, and diagnostics.

Usage

```
crossfit(
  data,
  method,
  fold_split = function(data, K) sample(rep_len(1:K, nrow(data))),
  seed = NULL,
  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.
verbose	Logical; if TRUE, prints a compact status line per repetition.

Value

A list with components estimate, results (a list containing per-repetition values and errors), repeats_done, K, K_required, method, and plan.

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$estimate

```

crossfit_failure_control

Configure failure handling for a cross-fitting method

Description

Creates a named list of method-specific failure-handling options for [create_method](#).

Usage

```

crossfit_failure_control(
  fail_repetition_on_error = TRUE,
  prune_on_shared_fit_failure = FALSE,
  max_failed_repetitions = Inf
)

```

Arguments

- `fail_repetition_on_error`
 Logical scalar. If TRUE, the first panel error fails the repetition. If FALSE, all panels are attempted and the panel aggregator decides whether the available results are sufficient.
- `prune_on_shared_fit_failure`
 Logical scalar. If TRUE, a method is skipped within a repetition when an earlier method encountered a failure in a structurally identical nuisance fit. This option requires `fail_repetition_on_error = TRUE`.
- `max_failed_repetitions`
 Non-negative integer or Inf. A method is disabled after it exceeds this number of failed repetitions.

Details

By default, an error in any panel fails the current repetition. If `fail_repetition_on_error = FALSE`, the engine evaluates the remaining panels, represents each failed panel by `NA_real_`, and passes the complete panel list to the method's `aggregate_panels` function. The repetition is successful when that aggregation succeeds.

If `prune_on_shared_fit_failure = TRUE`, a method is skipped when a structurally identical nuisance fit has already failed in the same repetition. If it is FALSE, the method attempts the fit independently.

Value

A named list suitable for the `failure_control` argument of [create_method](#).

Examples

```
failure_control <- crossfit_failure_control(
  fail_repetition_on_error = FALSE,
  max_failed_repetitions = 2L
)

str(failure_control)
```

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,
  verbose = FALSE
)
```

Arguments

<code>data</code>	Data frame or matrix of size $n \times p$ containing the observations.
<code>methods</code>	A (named) list of method specifications, typically created with create_method .
<code>fold_split</code>	A function of the form <code>function(data, K)</code> returning a vector of length <code>nrow(data)</code> with integer fold labels in <code>1:K</code> . It must assign at least one observation to each fold.
<code>seed</code>	Integer base random seed used for the K-fold splits; each repetition uses <code>seed + rep_id - 1</code> .
<code>aggregate_panels</code>	Function used as the <i>default</i> aggregator over panels for each method. It is applied to the list of per-panel values. Methods can override this via their own <code>aggregate_panels</code> .
<code>aggregate_repeats</code>	Function used as the <i>default</i> aggregator over repetitions for each method. It is applied to the list of per-repetition aggregated values. Methods can override this via their own <code>aggregate_repeats</code> .
<code>verbose</code>	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:

<code>estimates</code>	Named list of final estimates per method (after aggregating over panels and repetitions).
<code>per_method</code>	For each method, a list with values (per-repetition aggregated results) and errors (panel and aggregation error traces, including errors tolerated by a method's panel aggregator).
<code>repeats_done</code>	Number of repetitions successfully completed for each method.
<code>K</code>	Number of folds used in the plan.
<code>K_required</code>	Per-method minimal required K based on their dependency structure.
<code>methods</code>	The validated and normalized method specifications.
<code>plan</code>	The cross-fitting plan produced by <code>build_instances()</code> .

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	<i>Aggregators for scalar estimates</i>
---------------	---

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

<code>xs</code>	A list of numeric values or numeric vectors. Elements are unlisted and concatenated prior to aggregation, so <code>xs</code> 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	<i>Aggregators for cross-fitted predictors</i>
----------------	--

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

* package

crossfit-package, [2](#)

create_method, [3](#), [5–9](#)

create_nuisance, [3](#), [5](#), [8](#)

crossfit, [3](#), [4](#), [6](#), [11](#)

crossfit-package, [2](#)

crossfit_failure_control, [4](#), [7](#)

crossfit_multi, [3](#), [4](#), [6](#), [8](#), [11](#)

mean_estimate, [4](#), [11](#)

mean_predictor, [4](#), [11](#)

median_estimate, [4](#)

median_estimate (mean_estimate), [11](#)

median_predictor, [4](#)

median_predictor (mean_predictor), [11](#)