Package 'dScoreTest'

September 2, 2026

Title Debiased Score Tests for Goodness of Fit and Model Comparison

Version 1.0.0

Description Debiased (Neyman-orthogonalized) score tests for assessing whether a semiparametric or parametric regression model is well-specified and for comparing nested models. The test employs a hunt-and-test strategy: on a held-out hunt sample, it fits the null model and uses machine learning to find a direction in which the null model's score seems positive; on an independent test sample, it assesses the significance of the score in the hunted direction. The test employs orthogonalization to eliminate the bias from estimating the null model, yielding a test statistic that is asymptotically standard normal under the null without requiring a parametric form for the alternative. Methods are provided for 'glm', 'lm' and 'mgcv::gam' fits as well as for detecting heterogeneous treatment effects. The methodology is described in Dhawan, Guo and Shah (2026) <doi:10.48550/arXiv.2607.28861>.

URL https://unbiased.co.in/dScoreTest/, https://github.com/richardkwo/dScoreTest

BugReports https://github.com/richardkwo/dScoreTest/issues

License MIT + file LICENSE

Encoding UTF-8

Language en-US

RoxygenNote 7.3.3

Imports grf, mgcv

Suggests knitr, rmarkdown, speff2trial, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

Config/Needs/website rmarkdown

NeedsCompilation no

Author F. Richard Guo [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-2081-7398>), Aditya Dhawan [aut]

Maintainer F. Richard Guo <ricguo@umich.edu>
Repository CRAN
Date/Publication 2026-09-02 20:20:02 UTC

Contents

`compare_models` *Compare two models using the debiased score test*

Description

Compare two models using the debiased score test

Usage

```
compare_models(fit.0, ...)
```

Arguments

`fit.0`	A fitted null-model object. Methods are provided for `glm`, `lm` and `mgcv::gam` fits.
`...`	Additional arguments passed to the dispatched method, notably the alternative-model fit `fit.1`.

Value

An object of class "dScoreTest": a list whose key elements are the debiased test statistic t.stat and the one-sided p-value p.val (right tail of the standard normal), along with the test-set score residuals, the hunted direction, and the call. It has print, summary and plot methods.

References

Dhawan, A., Guo, F. R. and Shah, R. D. (2026). The debiased score test: hunt-and-test for semi-parametric hypotheses. arXiv:2607.28861. https://arxiv.org/abs/2607.28861

See Also

compare_models.glm, compare_models.lm, compare_models.gam, gof_test, dScoreTest

Examples

```
set.seed(42)
n <- 500
dat <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
dat$x3 <- dat$x3 + (dat$x1 + dat$x2) / 3
dat$y <- 5 * exp(dat$x1 + dat$x3) + rnorm(n) * 3
fit.0 <- glm(y ~ x1 + x3, family = gaussian(link = "log"),
             data = dat, start = rep(1, 3))
fit.1 <- glm(y ~ x1 + x2 + x3, family = gaussian(link = "log"),
             data = dat, start = rep(1, 4))

# test fit.0 against fit.1: should not be rejected
compare_models(fit.0, fit.1)
compare_models(fit.0, fit.1, hunt.style="wls")
anova(fit.0, fit.1)

# test a misspecified null model: should be rejected
fit.00 <- glm(y ~ x2, family = gaussian(link = "log"),
             data = dat, start = rep(1, 2))
compare_models(fit.00, fit.1)
plot(compare_models(fit.00, fit.1))
compare_models(fit.00, fit.1, hunt.style="wls")
plot(compare_models(fit.00, fit.1, hunt.style="wls"))
anova(fit.00, fit.1)
```

---

compare_models.gam *Compare two fitted GAM models*

Description

Debiased score test of the null fit.0 against the alternative fit.1, both fitted with mgcv::gam. GLM fit.1 is used to hunt for signal that fit.0 potentially misses.

Usage

```
## S3 method for class 'gam'
compare_models(
  fit.0,
  fit.1,
  hunt.style = "optimal",
  trim.outlier.hunt = TRUE,
  splits = c(0.5, 0.5),
  verbose = FALSE,
  ...
)
```

Arguments

`fit.0`	The null model as a fitted `mgcv::gam` object.
`fit.1`	The alternative model as a fitted `mgcv::gam` object. Must be a supermodel of `fit.0`: every predictor variable in `fit.0`'s formula must also appear in `fit.1`'s. `fit.0` and `fit.1` must be fit on the same rows in the same order.
`hunt.style`	Hunting algorithm with the following options. • `'optimal'`: optimal hunting (default). See `hunt_optimal`. • `'wls'`: a simpler hunting using weighted least squares, which can be less powerful. See `hunt_wls`.
`trim.outlier.hunt`	If `TRUE` (default), extreme values produced by the hunted function will be trimmed using Tukey's IQR rule.
`splits`	Numeric vector of length 2 or 3 giving the relative sizes of the sample splits; rescaled internally to sum to one. Default is `c(0.5, 0.5)`, which splits data into two halves for hunt and test respectively. Though typically unnecessary in practice, one can also specify a 3-way split for hunt, debiasing and test respectively.
`verbose`	Default `FALSE`; information is printed if set to `TRUE`.
`...`	Unused; present for S3 generic/method consistency.

Details

Nesting is checked by predictor-variable name only, not by basis span: two models with the same predictors but different smooth specifications (e.g. `s(x, k = 5)` vs `s(x, k = 20)`) will pass the check. The test remains meaningful as long as `fit.1`'s class contains the relevant alternative directions. Factor predictors, `offset()` terms, `weights` arguments, and multi-column responses are not supported.

Value

An object of class `"dScoreTest"`: a list whose key elements are the debiased test statistic `t.stat` and the one-sided p-value `p.val` (right tail of the standard normal), along with the test-set score residuals, the hunted direction, and the call. It has `print`, `summary` and `plot` methods.

Examples

```
set.seed(42)
dat <- mgcv::gamSim(eg=1, n=500, dist="normal", scale=1)
dat <- dat[, 1:5]

# test fit.0 against fit.1: well-specified (f3 = 0) and should not be rejected
fit.0  <- mgcv::gam(y ~ s(x0) + s(x1) + s(x2), data = dat)
fit.1  <- mgcv::gam(y ~ s(x0) + s(x1) + s(x2) + s(x3), data = dat)
compare_models(fit.0, fit.1)
plot(compare_models(fit.0, fit.1))
anova(fit.0, fit.1)

# mis-specified model: drops f2 and should be rejected
fit.00 <- mgcv::gam(y ~ s(x0) + s(x1) + s(x3), data = dat)
compare_models(fit.00, fit.1)
plot(compare_models(fit.00, fit.1))
compare_models(fit.00, fit.1, hunt.style="wls")
anova(fit.00, fit.1)
```

`compare_models.glm` *Compare two fitted GLM models*

Description

Debiased score test of the null model `fit.0` against the alternative `fit.1`. GLM `fit.1` is used to hunt for signal that `fit.0` potentially misses.

Usage

```
## S3 method for class 'glm'
compare_models(
  fit.0,
  fit.1,
  hunt.style = "optimal",
  trim.outlier.hunt = TRUE,
  splits = c(0.5, 0.5),
  verbose = FALSE,
  ...
)
```

Arguments

`fit.0`	The null model as a fitted `glm` object.
`fit.1`	The alternative model as a fitted `glm` object. It must be a supermodel of `fit.0`: every column of `stats::model.matrix(fit.0)` must appear (by name) in `stats::model.matrix(fit.` `fit.0` and `fit.1` must be fit on the same rows in the same order.
`hunt.style`	Hunting algorithm with the following options.

- `'optimal'`: optimal hunting (default). See `hunt_optimal`.
- `'wls'`: a simpler hunting using weighted least squares, which can be less powerful. See `hunt_wls`.

`trim.outlier.hunt`
: If `TRUE` (default), extreme values produced by the hunted function will be trimmed using Tukey's IQR rule.

`splits`
: Numeric vector of length 2 or 3 giving the relative sizes of the sample splits; rescaled internally to sum to one. Default is `c(0.5, 0.5)`, which splits data into two halves for hunt and test respectively. Though typically unnecessary in practice, one can also specify a 3-way split for hunt, debiasing and test respectively.

`verbose`
: Default `FALSE`; information is printed if set to `TRUE`.

`...`
: Unused; present for S3 generic/method consistency.

Value

An object of class `"dScoreTest"`: a list whose key elements are the debiased test statistic `t.stat` and the one-sided p-value `p.val` (right tail of the standard normal), along with the test-set score residuals, the hunted direction, and the call. It has `print`, `summary` and `plot` methods.

Examples

```
set.seed(42)
n <- 500
dat <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
dat$x3 <- dat$x3 + (dat$x1 + dat$x2) / 3
dat$y <- 5 * exp(dat$x1 + dat$x3) + rnorm(n) * 3
fit.0 <- glm(y ~ x1 + x3, family = gaussian(link = "log"),
             data = dat, start = rep(1, 3))
fit.1 <- glm(y ~ x1 + x2 + x3, family = gaussian(link = "log"),
             data = dat, start = rep(1, 4))

# test fit.0 against fit.1: should not be rejected
compare_models(fit.0, fit.1)
compare_models(fit.0, fit.1, hunt.style="wls")
anova(fit.0, fit.1)

# test a misspecified null model: should be rejected
fit.00 <- glm(y ~ x2, family = gaussian(link = "log"),
             data = dat, start = rep(1, 2))
compare_models(fit.00, fit.1)
plot(compare_models(fit.00, fit.1))
compare_models(fit.00, fit.1, hunt.style="wls")
plot(compare_models(fit.00, fit.1, hunt.style="wls"))
anova(fit.00, fit.1)
```

compare_models.lm *Compare two fitted linear models*

Description

Debiased score test of the null `fit.0` against the alternative `fit.1`. Both are internally refit as Gaussian-family GLMs and the call is dispatched to `compare_models.glm`.

Usage

```
## S3 method for class 'lm'
compare_models(fit.0, fit.1, ...)
```

Arguments

`fit.0`	The null model as a fitted `lm` object.
`fit.1`	The alternative model as a fitted `lm` object. Must be a supermodel of `fit.0`; see `compare_models.glm`.
`...`	Additional arguments passed to `compare_models.glm`.

Value

An object of class `"dScoreTest"`: a list whose key elements are the debiased test statistic `t.stat` and the one-sided p-value `p.val` (right tail of the standard normal), along with the test-set score residuals, the hunted direction, and the call. It has `print`, `summary` and `plot` methods.

Examples

```
set.seed(42)
n <- 500
dat <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
dat$x3 <- dat$x3 + (dat$x1 + dat$x2) / 3
dat$y  <- 1 + dat$x1 + 2 * dat$x3 + rnorm(n)
fit.0 <- lm(y ~ x1 + x3,        data = dat)
fit.1 <- lm(y ~ x1 + x2 + x3,   data = dat)

# test fit.0 against fit.1: should not be rejected
compare_models(fit.0, fit.1)
anova(fit.0, fit.1)

# misspecified model: should be rejected
fit.00 <- lm(y ~ x1 + x2, data = dat)
compare_models(fit.00, fit.1)
plot(compare_models(fit.00, fit.1))
compare_models(fit.00, fit.1, hunt.style="wls")
anova(fit.00, fit.1)
```

`debias_standard` *Standard debiasing*

Description

Transform a hunted function $\hat{h}$ into a debiased function $\hat{h} - \hat{m}_{\hat{h}}$, where $\hat{m}_{\hat{h}}$ is the projection of $\hat{h}$ onto the null model.

Usage

```
debias_standard(
  h.hat,
  X.debias,
  fit.debias,
  predict_fun,
  weight_fun,
  wls_method,
  arg.wls_method
)
```

Arguments

`h.hat`	A list as returned by one of `hunt_optimal()`, `hunt_wls()`, `hunt_vanilla()`.
`X.debias`	Part of X for debiasing.
`fit.debias`	Null model fitted on the debiasing sample of X and y.
`predict_fun`, `weight_fun`, `wls_method`, `arg.wls_method`	They must be compatible with `fit.debias`; see `dScoreTest()` for details.

Details

The projection $\hat{m}_{\hat{h}}$ is obtained by fitting the null model (via `wls_method`, weighted by `weight_fun(fit.debias, X.debias)`) with the hunted values `h.hat$h(X.debias)` as response. This projection uses **all** columns of `X`, even when the hunt itself is driven by only a subset of covariates (`h.hat$X.cols`).

Value

A list with elements:

`m.h.fit` The null model fitted (over all columns of `X`) to project and debias $\hat{h}$.

`h` The debiased hunt function $\hat{h} - \hat{m}_{\hat{h}}$ with signature `h(X)`.

dScoreTest *Debiased score test: goodness-of-fit test and model comparison*

Description

Test whether a semiparametric (e.g., GAM) or parametric (e.g., glm) regression model is well-specified. The test is a debiased (Neyman-orthogonalized) score test computed via sample splitting: on a held-out hunt sample, the null model is fit and a flexible ML algorithm is used to hunt for a direction in which the null model's score seems positive; on an independent test sample, that direction's score is evaluated to assess the significance. The test employs orthogonalization to eliminate plug-in bias from estimating the null model, so the resulting test statistic is asymptotically standard normal under the null without requiring a parametric form for the alternative.

Usage

```
dScoreTest(
  y,
  X,
  score_fun,
  weight_fun,
  fit_method,
  wls_method,
  hunt.style = "optimal",
  hunt.method = "grf",
  hunt_fun = NULL,
  debias.method = "standard",
  debias_fun = NULL,
  trim.outlier.hunt = TRUE,
  X.cols.hunt = 1:ncol(X),
  splits = c(0.5, 0.5),
  arg.fit_method = NULL,
  arg.wls_method = NULL,
  arg.hunt_fun = NULL,
  predict_fun = stats::predict,
  predict_fun_hunt = NULL,
  verbose = FALSE
)
```

Arguments

y	Numeric response vector of length n.
X	Numeric covariate matrix of dimension n x p.
score_fun	Function with signature score_fun(fit, y, X) returning a vector of scores $l'(\hat{f}(x_i), y_i)$, which can be viewed as negative residuals.
weight_fun	Function with signature weight_fun(fit, X) that computes the weight $\mathbb{E}[l''(\hat{f}(x_i), y_i)\|x_i]$ for each row x_i of X.

`fit_method` Function with signature `fit_method(y, X, ...)` that returns a fitted null model $\hat{f} \in \mathcal{F}$ by minimizing the loss $\sum_i l(f(x_i), y_i)$. For a fitted `f`, it must support `predict_fun(f, X)` for evaluation.

`wls_method` Function with signature `wls_method(y, X, w, ...)` that fits the null model $\hat{f} \in \mathcal{F}$ with weighted least squares, i.e., minimizing $\sum_i w_i(f(x_i) - y_i)^2$. For a fitted `f`, it must support `predict_fun(f, X)` for evaluation.

`hunt.style` Hunting algorithm with the following options.

- `'optimal'`: optimal hunting (default). See `hunt_optimal`.
- `'wls'`: a simpler hunting using weighted least squares, which can be less powerful. See `hunt_wls`.
- `'vanilla'`: a basic hunting; not recommended unless unable to fit an alternative model with weighted least squares. See `hunt_vanilla`.

`hunt.method` Built-in method for hunting. Currently available:

- `'grf'`: regression forest from package `grf`.

When this is set to any other value, arguments `hunt_fun` and `predict_fun_hunt` must be set properly to supply a customized hunting method.

`hunt_fun` Default `NULL`. When `hunt.method` is not set to a built-in method, this is a customized function for hunting. When `hunt.style` is `'optimal'` or `'wls'`, this function must have signature `hunt_fun(y, X, w, ...)` that returns a fitted *alternative model* $\hat{g} \in \mathcal{G}$ via weighted least squares, i.e., by minimizing $\sum_i w_i(y_i - g(x_i))^2$; otherwise, for `'vanilla'` hunting, this function must have signature `hunt_fun(y, X, ...)` that returns an *alternative model* fitted in any fashion. The returned object `g` must support `predict_fun_hunt(g, X)` for evaluation.

`debias.method` Debiasing method. Currently available:

- `'standard'`: standard debiasing (default). See `debias_standard`.

When set to any other value, `debias_fun` must be supplied.

`debias_fun` Default `NULL`. When `debias.method` is not `'standard'`, this is a customized debiasing function with the same signature as `debias_standard`, returning a list with an element `h`, the debiased hunted function.

`trim.outlier.hunt` If `TRUE` (default), extreme values produced by the hunted function will be trimmed using Tukey's IQR rule.

`X.cols.hunt` Integer vector selecting which columns of `X` drive the hunt. Default `1:ncol(X)`. This is modified only in special settings, e.g., when there is an offset in the null model.

`splits` Numeric vector of length 2 or 3 giving the relative sizes of the sample splits; rescaled internally to sum to one. Default is `c(0.5, 0.5)`, which splits data into two halves for hunt and test respectively. Though typically unnecessary in practice, one can also specify a 3-way split for hunt, debiasing and test respectively.

`arg.fit_method` Named list of additional arguments passed to `fit_method` (default to `NULL`).

`arg.wls_method` Named list of additional arguments passed to `wls_method` (default to `NULL`).

`arg.hunt_fun` Extra arguments (default `NULL`) passed to the customized `hunt.fun`.

predict_fun — Function with signature `predict_fun(fit, X)` returning a numeric vector of predictions from a fitted null model, which is produced by `fit_method()` and `wls_method()`. Note that if `fit` is $\hat{f}$, this function should return $\hat{f}(X)$. Default `stats::predict`. When y is binary, it must also support signature `predict_fun(fit, X, type='response')` for returning probabilities.

predict_fun_hunt — Default `NULL`. When `hunt.method` is not set to a built-in method, this is a function with signature `predict_fun_hunt(fit, X)` returning a numeric vector of predictions from a fitted alternative model produced by `hunt_fun()`.

verbose — Default `FALSE`; information is printed if set to `TRUE`.

Details

For most scenarios, use one of these methods instead:

- Use `gof_test` to test whether a fitted model is well-specified against a nonparametric alternative. S3 methods are provided for `glm` (`gof_test.glm`), `lm` (`gof_test.lm`) and `mgcv::gam` (`gof_test.gam`).
- Use `compare_models` to test a null model `fit.0` against an alternative supermodel `fit.1` in the same model class. Similar to `anova`, method can be used to conduct a significance test of one or more predictors. In contrast with `gof_test`, this method targets the alternative `fit.1`. S3 methods are provided for `glm` (`compare_models.glm`), `lm` (`compare_models.lm`) and `mgcv::gam` (`compare_models.gam`).
- Use `hte_test_conditional` to test treatment effect heterogeneity.

Use `dScoreTest` directly for full control over the score, weight, refit and hunt routines: this is the underlying engine that the S3 methods wrap.

Value

An object of class `"dScoreTest"`: a list whose key elements are the debiased test statistic `t.stat` and the one-sided p-value `p.val` (right tail of the standard normal), along with the test-set score residuals, the hunted direction, and the call. It has `print`, `summary` and `plot` methods.

Author(s)

Maintainer: F. Richard Guo <ricguo@umich.edu> (ORCID) [copyright holder]

Authors:

- Aditya Dhawan <ad950@cam.ac.uk>

References

Dhawan, A., Guo, F. R. and Shah, R. D. (2026). The debiased score test: hunt-and-test for semiparametric hypotheses. arXiv:2607.28861. https://arxiv.org/abs/2607.28861

See Also

Useful links:

- https://unbiased.co.in/dScoreTest/
- https://github.com/richardkwo/dScoreTest
- Report bugs at https://github.com/richardkwo/dScoreTest/issues

plot.dScoreTest, summary.dScoreTest, hunt_optimal, hunt_wls, hunt_vanilla, new_dScoreTest

Examples

```
## An example for customizing a dScoreTest:
## Conditional mean independence: is E[Y | X] a function of X[, 1:3] alone,
## i.e. do X4 and X5 carry no further information once X1, X2, X3 are given?
set.seed(1)
n <- 500
X <- matrix(rnorm(n * 5), n, 5)
y     <- X[, 1] + X[, 2]^2 + sin(X[, 3]) + rnorm(n)   # null TRUE
y.alt <- y + X[, 4] * X[, 5]                          # null FALSE

## Null model class: an arbitrary function of X[, 1:3], fitted by a
## regression forest. The fitter and predict_fun subset to those columns,
## while the hunt still searches all five columns for a direction of
## misspecification. Note honesty = FALSE with tuning: an underfitted null
## model is itself misspecified, and the test then (correctly) rejects on
## that lack of fit rather than on any dependence on X4, X5.
fit_method <- function(y, X, ...)
    grf::regression_forest(X[, 1:3, drop = FALSE], y,
                           honesty = FALSE, tune.parameters = "all")
wls_method <- function(y, X, w, ...)
    grf::regression_forest(X[, 1:3, drop = FALSE], y, sample.weights = w,
                           honesty = FALSE, tune.parameters = "all")
predict_fun <- function(fit, X, ...)
    predict(fit, X[, 1:3, drop = FALSE])$predictions

## Square loss l(f, y) = (y - f)^2 / 2 gives the score l'(f, y) = f - y
## (a negative residual) and the weight l''(f, y) = 1.
score_fun  <- function(fit, y, X, ...) predict_fun(fit, X) - y
weight_fun <- function(fit, X, ...) rep(1, nrow(X))


## Null holds: no evidence against it.
dScoreTest(y, X, score_fun, weight_fun, fit_method, wls_method,
           predict_fun = predict_fun)

## Null fails: the dependence on X4 and X5 is detected.
dScoreTest(y.alt, X, score_fun, weight_fun, fit_method, wls_method,
           predict_fun = predict_fun)
```

`fit_CATE` *Fit the conditional treatment effect (CATE) function*

Description

Returns a fitted CATE $\tau(Z)$ where covariates Z_S has no effect modification given the remaining covariates. It employs R-loss and cross fitting to estimate

$$\mathbb{E}[Y|T,Z] = \mu_0(Z) + T \cdot \tau(Z), \quad \mu_0(Z) := \mathbb{E}[Y \mid T = 0, Z].$$

Usage

```
fit_CATE(
  y,
  X,
  w = rep(1, nrow(X)),
  S = 1:(ncol(X) - 1),
  folds.crossfit = 5,
  randomized = FALSE
)
```

Arguments

`y`	Numeric response vector of length n.
`X`	Matrix $X = [T, Z]$ of dimension n x (p+1), where the first column is the binary treatment T and the remaining are the covariates Z.
`w`	Non-negative numeric weight vector of length n. Defaults to `rep(1, nrow(X))`. Applied when fitting the CATE (with weights $\tilde{T}^2 w$) and the control mean (with weights w). When it is not constant, this amounts to fitting CATE (or rather $\mathbb{E}[Y\|T,Z]$) using weighted least squares.
`S`	A subset of `1:(ncol(X)-1)` such that `X[,-1][,S]` gives the covariates S which have no effect modification conditionally. When `S=1:(ncol(X)-1)`, CATE must be a constant; when `S=NULL`, CATE is unrestricted.
`folds.crossfit`	An integer for the number of folds in cross fitting using the R-loss to estimate CATE. When it is 1, no cross fitting is used.
`randomized`	If `FALSE` (default), the propensity $e(Z) = \mathbb{E}[T \mid Z]$ is estimated by a cross-fitted `grf::probability_forest`. If `TRUE`, T is assumed to be randomized (independent of `Z`), so $e(Z)$ is taken to be the constant `mean(T)`, fitted upfront on the full sample without cross-fitting.

Value

An object of class `"CATE"`:

`control_mean_fun` $\mu_0(Z) = \mathbb{E}[Y|T=0,Z]$

`CATE_fun` $\tau(Z) = \mathbb{E}[Y|T=1,Z] - \mathbb{E}[Y|T=0,Z]$, which only depends on Z through Z_{S^c}.

`S` `S` as specified.

`p` Number of covariates, which equals `ncol(Z)`.

Examples

```
## A randomized trial in which the treatment effect depends on Z1 only.
set.seed(2)
n <- 500
Z  <- matrix(rnorm(n * 2), n, 2, dimnames = list(NULL, c("Z1", "Z2")))
Tr <- rbinom(n, 1, 0.5)                      # randomized treatment
tau <- Z[, "Z1"]                             # true CATE
y  <- Z[, "Z1"] + Z[, "Z2"] + Tr * tau + rnorm(n)


## S = NULL leaves the CATE unrestricted, so it may depend on Z1 and Z2.
fit <- fit_CATE(y, cbind(Tr, Z), S = NULL, randomized = TRUE,
                folds.crossfit = 2)
cor(fit$CATE_fun(Z), tau)      # close to 1: the true CATE is recovered
sd(fit$CATE_fun(Z))

## S = 1 bars Z1 from modifying the effect. Because the true CATE depends
## on Z1 alone, the fitted CATE then collapses to nearly a constant.
fit0 <- fit_CATE(y, cbind(Tr, Z), S = 1, randomized = TRUE,
                 folds.crossfit = 2)
sd(fit0$CATE_fun(Z))           # much smaller than above

## predict() gives the fitted outcome mean mu0(Z) + T * tau(Z).
head(predict(fit, cbind(Tr, Z)))
```

`gof_test` *Debiased score test for goodness of fit*

Description

Debiased score test for goodness of fit

Usage

```
gof_test(object, ...)
```

Arguments

`object`	A fitted model object. Methods are provided for `glm`, `lm` and `mgcv::gam` fits.
`...`	Additional arguments passed to the dispatched method.

Value

An object of class `"dScoreTest"`: a list whose key elements are the debiased test statistic `t.stat` and the one-sided p-value `p.val` (right tail of the standard normal), along with the test-set score residuals, the hunted direction, and the call. It has `print`, `summary` and `plot` methods.

References

Dhawan, A., Guo, F. R. and Shah, R. D. (2026). The debiased score test: hunt-and-test for semi-parametric hypotheses. arXiv:2607.28861. https://arxiv.org/abs/2607.28861

See Also

gof_test.glm, gof_test.lm, gof_test.gam, compare_models, dScoreTest

Examples

```
set.seed(42)
n <- 500
X <- matrix(rnorm(n * 3), nrow = n)
# log(E[y]) ~ X well-specified
y0 <- 5 * exp(X[,1] + X[,3]) + rnorm(n) * 3
fit.0 <- glm(y0 ~ X, family = gaussian(link = "log"), start=rep(1,4))
gof_test(fit.0)
# log(E[y]) ~ X misspecified
y1 <- y0 + exp(6 * cos(X[,1]/6)^2) / sqrt(n)
fit.1 <- glm(y1 ~ X, family = gaussian(link = "log"), start=rep(1,4))
gof_test(fit.1)
```

---

gof_test.gam *Goodness-of-fit test for a GAM*

Description

Debiased score test for goodness of fit of an mgcv::gam fit.

Usage

```
## S3 method for class 'gam'
gof_test(
  object,
  hunt.style = "optimal",
  hunt.method = "grf",
  hunt_fun = NULL,
  trim.outlier.hunt = TRUE,
  X.cols.exclude = NULL,
  splits = c(0.5, 0.5),
  arg.hunt_fun = NULL,
  predict_fun_hunt = NULL,
  verbose = FALSE,
  ...
)
```

Arguments

`object`	Fitted `mgcv::gam` object.
`hunt.style`	Hunting algorithm with the following options. • `'optimal'`: optimal hunting (default). See `hunt_optimal`. • `'wls'`: a simpler hunting using weighted least squares, which can be less powerful. See `hunt_wls`. • `'vanilla'`: a basic hunting; not recommended unless unable to fit an alternative model with weighted least squares. See `hunt_vanilla`.
`hunt.method`	Built-in method for hunting. Currently available: • `'grf'`: regression forest from package `grf`. When this is set to any other value, arguments `hunt_fun`, `arg.hunt_fun` and `predict_fun_hunt` are used to specify a customized hunting method.
`hunt_fun`	Default `NULL`. When `hunt.method` is not set to a built-in method, this is a customized function for hunting. When `hunt.style` is `'optimal'` or `'wls'`, this function must have signature `hunt_fun(y, X, w, ...)` that returns a fitted *alternative model* $\hat{g} \in \mathcal{G}$ via weighted least squares, i.e., by minimizing $\sum_i w_i(y_i - g(x_i))^2$; otherwise, for `'vanilla'` hunting, this function must have signature `hunt_fun(y, X, ...)` that returns an *alternative model* fitted in any fashion. The returned object g must support `predict_fun_hunt(g, X)` for evaluation.
`trim.outlier.hunt`	If `TRUE` (default), extreme values produced by the hunted function will be trimmed using Tukey's IQR rule.
`X.cols.exclude`	Columns in `stats::model.matrix(object)` to be excluded when hunting for alternative signal. Default `NULL`.
`splits`	Numeric vector of length 2 or 3 giving the relative sizes of the sample splits; rescaled internally to sum to one. Default is `c(0.5, 0.5)`, which splits data into two halves for hunt and test respectively. Though typically unnecessary in practice, one can also specify a 3-way split for hunt, debiasing and test respectively.
`arg.hunt_fun`	Extra arguments (default `NULL`) passed to the customized `hunt.fun`.
`predict_fun_hunt`	When a customized `hunt.fun` is used, this is a function with signature `predict_fun_hunt(fit, X)` returning a numeric vector of predictions from a fitted alternative model produced by `hunt_fun()`.
`verbose`	Default `FALSE`; information is printed if set to `TRUE`.
`...`	Unused; present for S3 generic/method consistency.

Details

Only the numeric predictors appearing in `stats::model.frame(object)` are exposed to the hunt; `X.cols.exclude` indexes into these predictor variables (not basis columns). Factor-by smooths and other non-numeric predictors are not currently supported. Formulas using `offset()` terms, a `weights` argument, or a multi-column response (e.g. `cbind(succ, fail) ~ ...`) are also not supported.

Value

An object of class "dScoreTest": a list whose key elements are the debiased test statistic t.stat and the one-sided p-value p.val (right tail of the standard normal), along with the test-set score residuals, the hunted direction, and the call. It has print, summary and plot methods.

Examples

```
set.seed(42)
dat <- mgcv::gamSim(eg=1, n=500, dist="normal", scale=2, verbose = FALSE)
dat.0 <- dat[,1:5]

# well-specified
fit.0 <- mgcv::gam(y~s(x0)+s(x1)+s(x2)+s(x3),data=dat.0)
test.0 <- gof_test(fit.0)
# f3=0, also well-specified
fit.1 <- mgcv::gam(y~s(x0)+s(x1)+s(x2),data=dat.0)
test.1 <- gof_test(fit.1)
plot(test.1)
# misspecified
dat.1 <- dat.0
dat.1$y <- dat.1$y * dat$f0
fit.2 <- mgcv::gam(y~s(x0)+s(x1)+s(x2)+s(x3), data=dat.1)
test.2 <- gof_test(fit.2)
plot(test.2)
```

---

gof_test.glm *Goodness-of-fit test for GLM*

Description

Debiased score test for goodness of fit of GLM.

Usage

```
## S3 method for class 'glm'
gof_test(
  object,
  hunt.style = "optimal",
  hunt.method = "grf",
  hunt_fun = NULL,
  trim.outlier.hunt = TRUE,
  X.cols.exclude = NULL,
  splits = c(0.5, 0.5),
  arg.hunt_fun = NULL,
  predict_fun_hunt = NULL,
  verbose = FALSE,
  ...
)
```

Arguments

`object`	Fitted glm object.
`hunt.style`	Hunting algorithm with the following options. • `'optimal'`: optimal hunting (default). See `hunt_optimal`. • `'wls'`: a simpler hunting using weighted least squares, which can be less powerful. See `hunt_wls`. • `'vanilla'`: a basic hunting; not recommended unless unable to fit an alternative model with weighted least squares. See `hunt_vanilla`.
`hunt.method`	Built-in method for hunting. Currently available: • `'grf'`: regression forest from package `grf`. When this is set to any other value, arguments `hunt_fun`, `arg.hunt_fun` and `predict_fun_hunt` are used to specify a customized hunting method.
`hunt_fun`	Default `NULL`. When `hunt.method` is not set to a built-in method, this is a customized function for hunting. When `hunt.style` is `'optimal'` or `'wls'`, this function must have signature `hunt_fun(y, X, w, ...)` that returns a fitted *alternative model* $\hat{g} \in \mathcal{G}$ via weighted least squares, i.e., by minimizing $\sum_i w_i(y_i - g(x_i))^2$; otherwise, for `'vanilla'` hunting, this function must have signature `hunt_fun(y, X, ...)` that returns an *alternative model* fitted in any fashion. The returned object `g` must support `predict_fun_hunt(g, X)` for evaluation.
`trim.outlier.hunt`	If `TRUE` (default), extreme values produced by the hunted function will be trimmed using Tukey's IQR rule.
`X.cols.exclude`	Columns in `stats::model.matrix(object)` to be excluded when hunting for alternative signal. Default `NULL`.
`splits`	Numeric vector of length 2 or 3 giving the relative sizes of the sample splits; rescaled internally to sum to one. Default is `c(0.5, 0.5)`, which splits data into two halves for hunt and test respectively. Though typically unnecessary in practice, one can also specify a 3-way split for hunt, debiasing and test respectively.
`arg.hunt_fun`	Extra arguments (default `NULL`) passed to the customized `hunt.fun`.
`predict_fun_hunt`	When a customized `hunt.fun` is used, this is a function with signature `predict_fun_hunt(fit, X)` returning a numeric vector of predictions from a fitted alternative model produced by `hunt_fun()`.
`verbose`	Default `FALSE`; information is printed if set to `TRUE`.
`...`	Unused; present for S3 generic/method consistency.

Value

An object of class `"dScoreTest"`: a list whose key elements are the debiased test statistic `t.stat` and the one-sided p-value `p.val` (right tail of the standard normal), along with the test-set score residuals, the hunted direction, and the call. It has `print`, `summary` and `plot` methods.

Examples

```
set.seed(42)
n <- 500
X <- matrix(rnorm(n * 3), nrow = n)
# log(E[y]) ~ X well-specified
y0 <- 5 * exp(X[,1] + X[,3]) + rnorm(n) * 3
fit.0 <- glm(y0 ~ X, family = gaussian(link = "log"), start=rep(1,4))
gof_test(fit.0)
# log(E[y]) ~ X misspecified
y1 <- y0 + exp(6 * cos(X[,1]/6)^2) / sqrt(n)
fit.1 <- glm(y1 ~ X, family = gaussian(link = "log"), start=rep(1,4))
gof_test(fit.1)
```

---

gof_test.lm *Goodness-of-fit test for a linear model*

Description

Debiased score test for goodness of fit of an `lm`. Internally refits the model as a Gaussian-family GLM and dispatches to `gof_test.glm`.

Usage

```
## S3 method for class 'lm'
gof_test(object, ...)
```

Arguments

`object`	Fitted `lm` object.
`...`	Additional arguments passed to `gof_test.glm`.

Value

An object of class `"dScoreTest"`: a list whose key elements are the debiased test statistic `t.stat` and the one-sided p-value `p.val` (right tail of the standard normal), along with the test-set score residuals, the hunted direction, and the call. It has `print`, `summary` and `plot` methods.

Examples

```
set.seed(42)
n <- 500
X <- matrix(rnorm(n * 3), nrow = n)
X[,3] <- X[,3] + X[,1] + X[,2] / 2
y0 <- 1 + X %*% c(1,1,2) + rnorm(n)  # well-specified
fit.0 <- lm(y0 ~ X)
test.0 <- gof_test(fit.0)
plot(test.0)
```

```
y1 <- y0 + cos(X[,1])  # mis-specified
fit.1 <- lm(y1 ~ X)
test.1 <- gof_test(fit.1)
plot(test.1)
```

`hte_test_conditional` *Test for detecting heterogeneity in the conditional treatment effect (CATE)*

Description

For binary treatment T, covariates Z and outcome Y, it tests the hypothesis

$$H_0^c(S) : \tau(Z) \text{ only depends on } Z \text{ through } Z_{S^c},$$

where S is a subset of covariates and $\tau(Z) = \mathbb{E}[Y|T=1,Z] - \mathbb{E}[Y|T=0,Z]$ is the CATE. When S is the full set, this amounts to testing $\tau(Z)$ is a constant; when S is a proper subset, this amounts to testing Z_S has no further effect modification while holding Z_{S^c} fixed.

Usage

```
hte_test_conditional(
  y,
  Tr,
  Z,
  S = 1:ncol(Z),
  hunt.style = "optimal",
  folds.crossfit = 5,
  trim.outlier.hunt = TRUE,
  splits = c(0.5, 0.5),
  arg.hunt_grf = list(honesty = FALSE, tune.parameters = "all"),
  verbose = FALSE,
  randomized = FALSE
)
```

Arguments

`y`	Numeric response vector of length n.
`Tr`	Binary (0/1) treatment vector of length n.
`Z`	Numeric covariate matrix of dimension n x p.
`S`	A subset of `1:ncol(Z)` giving the covariates whose effect modification is tested (they have none under the null, given the rest). Default `1:ncol(Z)` tests for *any* heterogeneity (constant CATE under the null); `S = NULL` leaves the CATE unrestricted.
`hunt.style`	One of `"optimal"` (default), `"wls"` or `"vanilla"`, selecting the hunting algorithm in `dScoreTest`. The hunted alternative is a `grf` outcome model fitted separately for the treated and control arms (T-learner).

`folds.crossfit` Number of cross-fitting folds passed to `fit_CATE` when estimating the CATE. Default 5.

`trim.outlier.hunt`, `splits`, `verbose`
Passed through to `dScoreTest`; see there for details.

`arg.hunt_grf` Arguments passed to `grf::regression_forest()` for hunting.

`randomized` If `FALSE` (default), the propensity $e(Z) = \mathbb{E}[T \mid Z]$ used by `fit_CATE` and by the debiasing step is estimated with `grf::probability_forest`. If `TRUE`, `T` is assumed randomized (independent of `Z`), so $e(Z)$ is taken to be the constant `mean(T)`, fitted upfront without cross-fitting.

Value

An object of class `"dScoreTest"`: a list whose key elements are the debiased test statistic `t.stat` and the one-sided p-value `p.val` (right tail of the standard normal), along with the test-set score residuals, the hunted direction, and the call. It has `print`, `summary` and `plot` methods.

References

Dhawan, A., Guo, F. R. and Shah, R. D. (2026). The debiased score test: hunt-and-test for semi-parametric hypotheses. arXiv:2607.28861. https://arxiv.org/abs/2607.28861

See Also

`fit_CATE`, `dScoreTest`

Examples

```
set.seed(1)
n <- 600
Z  <- matrix(rnorm(n * 3), n, 3)
Tr <- rbinom(n, 1, plogis(Z[, 1]))
y  <- Z[, 2] + Z[, 3] + Tr * (1 + Z[, 1]) + rnorm(n)  # CATE varies with Z1

# allow modification by Z1 (S = {2,3}): well-specified, should not reject
hte_test_conditional(y, Tr, Z, S = c(2, 3))
# forbid all modification (constant CATE): misspecified, should reject
hte_test_conditional(y, Tr, Z, S = 1:3)
```

`hunt_optimal` *Optimal hunting*

Description

Hunt by fitting residuals on X optimally, trained by solving a weighted least squares. The hunted function is also pre-debiased. This is the procedure of Section 2.2.1 of Dhawan, Guo and Shah (2026).

Usage

```
hunt_optimal(
  wls_hunt_method,
  wls_method,
  score_fun,
  weight_fun,
  fit,
  y,
  X,
  X.cols = 1:ncol(X),
  binary.y = FALSE,
  trim.outlier = TRUE,
  arg.wls_hunt_method = NULL,
  arg.wls_method = NULL,
  predict_fun = stats::predict,
  predict_fun_hunt = stats::predict
)
```

Arguments

`wls_hunt_method`
: Function with signature `wls_hunt_method(y, X, w, ...)` that returns a fitted *alternative model* $\hat{g} \in \mathcal{G}$ by minimizing $\sum_i w_i(y_i - g(x_i))^2$. The returned object must support `predict_fun_hunt(g, X)` for evaluation.

`wls_method`
: Function with signature `wls_method(y, X, w, ...)` that fits the *null model* $\hat{f} \in \mathcal{F}$ with weighted least squares, i.e., minimizing $\sum_i w_i(f(x_i) - y_i)^2$. The returned object must support `predict_fun(f, X)` for evaluation.

`score_fun`
: Function with signature `score_fun(fit, y, X)` returning a vector of scores $l'(\hat{f}(x_i), y_i)$.

`weight_fun`
: Function with signature `weight_fun(fit, X)` that computes the weight $\mathbb{E}[l''(\hat{f}(x_i), y_i)|x_i]$ for each row x_i of X.

`fit`
: Fitted null model. Must support `predict_fun(fit, X)`.

`y`
: Response vector of length n.

`X`
: Covariates of dim n x p.

`X.cols`
: Subset of covariates to hunt. Default `1:ncol(X)`.

`binary.y`
: Set to `TRUE` only if y is binary (default: `FALSE`). This only affects how the variance function is estimated. When `TRUE`, `predict_fun(fit, X)` must return the conditional probability $P(y = 1|x)$.

`trim.outlier`
: If `TRUE`, outliers in $\hat{h}(X)$ will be trimmed from the hunted $\hat{h}$ using Tukey's IQR rule.

`arg.wls_hunt_method`
: Named list of additional arguments passed to `wls_hunt_method` (default to `NULL`).

`arg.wls_method`
: Named list of additional arguments passed to `wls_method` (default to `NULL`).

predict_fun Function with signature predict_fun(fit, X) returning a numeric vector of predictions from null-model fits. Default stats::predict.

predict_fun_hunt
Function with signature predict_fun_hunt(fit, X) returning a numeric vector of predictions from the alternative-model fit. Default stats::predict.

Details

If y is binary, then set binary.y=TRUE. Meanwhile, predict_fun(fit, X, type="response") must output the predicted probabilities.

Value

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

hunt.fit The fitted hunt model produced by wls_hunt_method.

trim.bounds The Tukey IQR trimming bounds, or c(-Inf, Inf) when trim.outlier = FALSE.

predict_fun_hunt The prediction function for hunt.fit, as supplied.

X.cols The columns of X used for the hunt, as supplied.

h A function with signature h(X) giving the orthogonalized hunted signal.

References

Dhawan, A., Guo, F. R. and Shah, R. D. (2026). The debiased score test: hunt-and-test for semi-parametric hypotheses. arXiv:2607.28861. https://arxiv.org/abs/2607.28861

---

hunt_vanilla *Vanilla hunting*

Description

Hunt by fitting residuals on X.

Usage

```
hunt_vanilla(
  fit_hunt_method,
  resids,
  X,
  X.cols = 1:ncol(X),
  trim.outlier = TRUE,
  arg.fit_hunt_method = NULL,
  predict_fun_hunt = stats::predict
)
```

Arguments

`fit_hunt_method`
: Function with signature `fit_hunt_method(y, X, ...)` that returns a fitted *alternative model* $\hat{g} \in \mathcal{G}$. For a fitted g, it must support `predict_fun_hunt(g, X)` for evaluation.

`resids`
: Residuals (i.e., negative scores) of length n from the null model.

`X`
: Covariates of dim n x p.

`X.cols`
: Subset of covariates to hunt. (Default: `1:ncol(X)`)

`trim.outlier`
: If `TRUE`, outliers in $\hat{h}(X)$ will be trimmed from the hunted $\hat{h}$ using Tukey's IQR rule.

`arg.fit_hunt_method`
: Named list of additional arguments passed to `fit_hunt_method` (default to `NULL`).

`predict_fun_hunt`
: Function with signature `predict_fun_hunt(fit, X)` returning a numeric vector of predictions from the alternative-model fit. Default `stats::predict`.

Value

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

`hunt.fit` The fitted hunt model produced by `fit_hunt_method`.

`trim.bounds` The Tukey IQR trimming bounds, or `c(-Inf, Inf)` when `trim.outlier = FALSE`.

`predict_fun_hunt` The prediction function for `hunt.fit`, as supplied.

`X.cols` The columns of `X` used for the hunt, as supplied.

`h` A function with signature `h(X)` giving the hunted signal.

---

hunt_wls *Weighted-least-squares hunting*

Description

Hunt by fitting residuals on X, trained by solving a weighted least squares. See Proposition 2 of Dhawan, Guo and Shah (2026).

Usage

```
hunt_wls(
  wls_hunt_method,
  resids,
  X,
  X.cols = 1:ncol(X),
  trim.outlier = TRUE,
  arg.wls_hunt_method = NULL,
  predict_fun_hunt = stats::predict
)
```

Arguments

`wls_hunt_method`
: Function with signature `wls_hunt_method(y, X, w, ...)` that returns a fitted *alternative model* $\hat{g} \in \mathcal{G}$ by minimizing $\sum_i w_i(y_i - g(x_i))^2$. The returned object must support `predict_fun_hunt(g, X)` for evaluation.

`resids`
: Residuals (i.e., negative scores) of length n from the null model.

`X`
: Covariates of dim n x p.

`X.cols`
: Subset of covariates to hunt. (Default: `1:ncol(X)`)

`trim.outlier`
: If `TRUE`, outliers in $\hat{h}(X)$ will be trimmed from the hunted $\hat{h}$ using Tukey's IQR rule.

`arg.wls_hunt_method`
: Named list of additional arguments passed to `wls_hunt_method` (default to `NULL`).

`predict_fun_hunt`
: Function with signature `predict_fun_hunt(fit, X)` returning a numeric vector of predictions from the alternative-model fit. Default `stats::predict`.

Value

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

`hunt.fit` The fitted hunt model produced by `wls_hunt_method`.

`trim.bounds` The Tukey IQR trimming bounds, or `c(-Inf, Inf)` when `trim.outlier = FALSE`.

`predict_fun_hunt` The prediction function for `hunt.fit`, as supplied.

`X.cols` The columns of `X` used for the hunt, as supplied.

`h` A function with signature `h(X)` giving the hunted signal.

References

Dhawan, A., Guo, F. R. and Shah, R. D. (2026). The debiased score test: hunt-and-test for semi-parametric hypotheses. arXiv:2607.28861. https://arxiv.org/abs/2607.28861

`new_dScoreTest` *Constructor for the debiased score test*

Description

Internal worker that builds a `dScoreTest` object from a fixed three-way (or two-way) sample split. Called by `dScoreTest`. Splits, hunting algorithm, and predict semantics are taken as fully resolved arguments — no defaults are inferred from the data.

Usage

```
new_dScoreTest(
  y,
  X,
  idx.hunt,
  idx.debias,
  idx.test,
  score_fun,
  weight_fun,
  fit_method,
  wls_method,
  hunt.style = "optimal",
  hunt.method = "customized",
  debias.method = "standard",
  debias_fun = debias_standard,
  fit_hunt_method = NULL,
  wls_hunt_method = NULL,
  X.cols.hunt = 1:ncol(X),
  binary.y = FALSE,
  trim.outlier.hunt = TRUE,
  predict_fun = stats::predict,
  predict_fun_hunt = stats::predict,
  arg.fit_method = NULL,
  arg.wls_method = NULL,
  arg.fit_hunt_method = NULL,
  arg.wls_hunt_method = NULL
)
```

Arguments

`y`	Numeric response vector of length n.
`X`	Numeric covariate matrix of dimension n x p.
`idx.hunt`	Integer indices into `1:n` for the hunting subsample.
`idx.debias`	Integer indices into `1:n` for the debiasing subsample (refitting the null model and projecting the hunted direction).
`idx.test`	Integer indices into `1:n` for the test subsample on which the test statistic is evaluated. May coincide with `idx.debias` (two-way split) or be disjoint (three-way split).
`score_fun`	Function with signature `score_fun(fit, y, X)` returning a vector of scores $l'(\hat{f}(x_i), y_i)$.
`weight_fun`	Function with signature `weight_fun(fit, X)` returning the weight $\mathbb{E}[l''(\hat{f}(x_i), y_i)\|x_i]$ for each row of `X`.
`fit_method`	Function with signature `fit_method(y, X, ...)` returning a fitted null model. The returned object must support `predict_fun(fit, X)`.
`wls_method`	Function with signature `wls_method(y, X, w, ...)` that fits the null model by weighted least squares. The returned object must support `predict_fun(fit, X)`.

`hunt.style` One of `"optimal"` (default), `"wls"`, or `"vanilla"`. Selects which `hunt_*` routine is used.

`hunt.method` String for the hunting method.

`debias.method` String for the debiasing method, recorded in the returned object's `Call`.

`debias_fun` Function performing the debiasing, with the same signature as `debias_standard` (the default). Must return a list with an element `h`, the debiased hunted function.

`fit_hunt_method`
Required when `hunt.style = "vanilla"`; ignored otherwise. Function with signature `fit_hunt_method(y, X, ...)` returning a fitted alternative model that supports `predict_fun_hunt(g, X)`.

`wls_hunt_method`
Required when `hunt.style %in% c("optimal", "wls")`; ignored otherwise. Function with signature `wls_hunt_method(y, X, w, ...)` returning a fitted alternative model that supports `predict_fun_hunt(g, X)`.

`X.cols.hunt` Integer or name vector selecting which columns of `X` drive the hunt. Default: all columns.

`binary.y` Logical. When `TRUE`, the optimal hunter computes $\mathrm{Var}(l'|x)$ from `predict_fun(fit, X, type='response')` assuming a Bernoulli response. Only consulted by `hunt.style = "optimal"`.

`trim.outlier.hunt`
Logical. Passed to the chosen `hunt_*` routine as `trim.outlier`. If `TRUE` (default), extreme values in the hunted function will be removed using Tukey's IQR rule.

`predict_fun` Function with signature `predict_fun(fit, X)` returning predictions from a fitted null model. Default `stats::predict`.

`predict_fun_hunt`
Function with signature `predict_fun_hunt(fit, X)` returning predictions from a model fitted under alternative. Default `stats::predict`.

`arg.fit_method`, `arg.wls_method`, `arg.fit_hunt_method`, `arg.wls_hunt_method`
Named lists of additional arguments forwarded to the corresponding fitter via `do.call`. Default `NULL`.

Value

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

`t.stat` Debiased test statistic $\sqrt{n_{\text{test}}}\,\bar{L}/\hat{\sigma}_L$.

`p.val` One-sided p-value (right tail of the standard normal).

`resids` Score residuals on the test subsample.

`h` Orthogonalized hunted direction on the test subsample.

`h.raw` Hunted direction before the outer debias projection.

`hunted_fun` The debiased hunted function $\hat{h} - \hat{m}_{\hat{h}}$, a function that can be applied to X.

`Data` List with `X`, `y`, and the three index vectors.

`Call` Named list of methods, `hunt.style`, `hunt.method`, `debias.method`, both predict functions, and the four `arg.*` lists.

See Also

dScoreTest, hunt_optimal, hunt_wls, hunt_vanilla

`plot.dScoreTest` *Plot the score test*

Description

Diagonotic plots for the test:

1. Histogram of $\{L_i\}$, where $L_i = resid_i \times h_i$.
2. $\{L_i\}$ against the index i, where i refers to the i-th observation in the full dataset. Only those i's in the test split are drawn. The mean is drawn as a horizontal line. Extremes values under the null can result in bad normal approximation. In this case, consider setting `trim.outlier.hunt=TRUE`.
3. Residuals (negative scores) versus the hunted signal. A horizontal segment is drawn between each pair of raw hunted signal and the debiased hunted signal. If debiased gets higher, colored in red; otherwise colored in green. A regression line (blue) with a large positive slope indicates the model is misspecified.
4. Normalized $\{L_i\}$ drawn in order.

Usage

```
## S3 method for class 'dScoreTest'
plot(x, ...)
```

Arguments

`x`	A `dScoreTest` object.
`...`	Further graphical parameters passed to underlying plotting functions.

Value

No return value; called for its side effect of producing the diagnostic plots described above.

predict.CATE *Predict from a fitted CATE model*

Description

Computes the outcome mean $\mathbb{E}[Y|T,Z] = \mu_0(Z) + T\,\tau(Z)$ for a `"CATE"` object returned by `fit_CATE`.

Usage

```
## S3 method for class 'CATE'
predict(object, X, ...)
```

Arguments

`object`	A `"CATE"` object from `fit_CATE`.
`X`	Matrix $X = [T, Z]$ of dimension m x (p+1), where the first column is the binary treatment T and the remaining are the covariates Z.
`...`	Unused, for S3 consistency.

Value

Numeric vector of length `nrow(X)` giving $\mu_0(Z) + T\,\tau(Z)$.

See Also

`fit_CATE`

print.dScoreTest *Print the score test*

Description

Print the score test

Usage

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

Arguments

`x`	A `dScoreTest` object.
`...`	Unused, for S3 consistency.

Value

The input x, invisibly. Called for the side effect of printing a summary of the test to the console.

`score_fun` *Score for a fitted CATE model*

Description

Score for a fitted CATE model

Usage

```
score_fun(fit, y, X, ...)

## S3 method for class 'CATE'
score_fun(fit, y, X, ...)
```

Arguments

`fit`	A fitted model object.
`y`	Numeric response vector.
`X`	Covariate matrix.
`...`	Additional arguments passed to methods.

Value

Numeric vector of scores.

Methods (by class)

- `score_fun(CATE)`: Score for a `"CATE"` object: the residual $\hat{\mathbb{E}}[Y|T,Z] - Y$, i.e. predicted value minus `y`.

See Also

`score_fun.CATE`

summary.dScoreTest *Summary of the score test*

Description

Reports the headline statistic and p-value, the sample-split sizes, a raw-vs-debiased comparison of the test statistic (so the effect of the outer-projection debiasing step is visible), and `summary` digests of the diagnostic vectors `L`, `L.raw`, `h`, `h.raw` and `resids`.

Usage

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

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

Arguments

`object`	A `dScoreTest` object.
`...`	Unused, for S3 consistency.
`x`	A `summary.dScoreTest` object.

Value

A list of class `"summary.dScoreTest"`.

weight_fun *Weight for a fitted CATE model*

Description

Weight for a fitted CATE model

Usage

```
weight_fun(fit, X, ...)

## S3 method for class 'CATE'
weight_fun(fit, X, ...)
```

Arguments

`fit`	A fitted model object.
`X`	Covariate matrix.
`...`	Additional arguments passed to methods.

Value

Numeric vector of weights.

Methods (by class)

- `weight_fun(CATE)`: Weight for a `"CATE"` object: constant 1 for each row of `X`, since the outcome model has a squared-error loss.

See Also

`weight_fun.CATE`

Index