

Package: ddml (via r-universe)

October 13, 2026

Title Double/Debiased Machine Learning

Version 0.9.0

Date 2026-03-06

Description Estimate common causal parameters using double/debiased machine learning as proposed by Chernozhukov et al. (2018) [<doi:10.1111/ectj.12097>](https://doi.org/10.1111/ectj.12097). 'ddml' simplifies estimation based on (short-)stacking as discussed in Ahrens et al. (2024) [<doi:10.1002/jae.3103>](https://doi.org/10.1002/jae.3103), which leverages multiple base learners to increase robustness to the underlying data generating process.

License GPL (>= 3)

URL <https://github.com/thomaswiemann/ddml>,
<https://thomaswiemann.com/ddml/>

BugReports <https://github.com/thomaswiemann/ddml/issues>

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Depends R (>= 4.3)

Imports generics, glmnet, graphics, MASS, Matrix, methods, nnls,
pbapply, quadprog, ranger, stats, xgboost

Suggests AER, covr, CVXR, knitr, parallel, rmarkdown, sandwich,
testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

Config/pak/sysreqs make

Repository <https://thomaswiemann.r-universe.dev>

Date/Publication 2026-04-16 20:48:06 UTC

RemoteUrl <https://github.com/thomaswiemann/ddml>

RemoteRef HEAD

RemoteSha e17f94c63856173f4892d1528d7daf1b0e9562a9

Contents

AE98	3
as.list.ddml	4
as.list.ddml_rep	5
as.list.ral	5
as.list.ral_rep	6
coef.ral	7
coef.ral_rep	7
confint.ral	8
confint.ral_rep	9
crosspred	10
crossval	13
ddml	15
ddml-intro	17
ddml_apo	21
ddml_ate	24
ddml_attgt	29
ddml_fpliv	33
ddml_late	36
ddml_pliv	40
ddml_plm	43
ddml_policy	47
ddml_rep	51
ddml_replicate	52
diagnostics	53
ensemble	54
ensemble_weights	55
glance.ddml	57
glance.ddml_rep	58
glance.ral	58
glance.ral_rep	59
hatvalues.ral	59
lincom	60
lincom_weights_did	63
mdl_bigGlm	65
mdl_glm	66
mdl_glmnet	66
mdl_ranger	67
mdl_xgboost	68
nobs.ral	69
ols	70
plot.ral	71
predict.ensemble	72
predict.mdl_bigGlm	73
predict.mdl_glm	74
predict.mdl_glmnet	74
predict.mdl_ranger	75

predict.mdl_xgboost	75
predict.ols	76
print.ddml_diagnostics	76
ral	77
ral_rep	78
shortstacking	79
summary.ddml	81
summary.ddml_rep	82
summary.ral	83
summary.ral_rep	84
tidy.ddml	85
tidy.ddml_diagnostics	87
tidy.ddml_rep	87
tidy.ral	89
tidy.ral_rep	90
vcov.ral	91
vcov.ral_rep	92

Index 93

AE98

Random Subsample from the Data of Angrist & Evans (1998)

Description

Random subsample from the data of Angrist & Evans (1998).

Usage

AE98

Format

A data frame with 5,000 rows and 13 variables.

worked Indicator equal to 1 if the mother is employed.

weeksw Number of weeks of employment.

hoursw Hours worked per week.

morekids Indicator equal to 1 if the mother has more than 2 kids.

samesex Indicator equal to 1 if the first two children are of the same sex.

age Age in years.

agefst Age in years at birth of the first child.

black Indicator equal to 1 if the mother is black.

hisp Indicator equal to 1 if the mother is Hispanic.

othrace Indicator equal to 1 if the mother is neither black nor Hispanic.

educ Years of education.

boy1st Indicator equal to 1 if the first child is male.

boy2nd Indicator equal to 1 if the second child is male.

Source

[doi:10.7910/DVN/4W9GW2](https://doi.org/10.7910/DVN/4W9GW2)

References

Angrist J, Evans W (1998). "Children and Their Parents' Labor Supply: Evidence from Exogenous Variation in Family Size." *American Economic Review*, 88(3), 450-477.

as.list.ddml

Split a DDML Object by Ensemble Type

Description

Returns a named list of single-ensemble ddml objects. Each element retains all S3 methods (summary, tidy, glance, confint, vcov).

Usage

```
## S3 method for class 'ddml'
as.list(x, ...)
```

Arguments

x An object inheriting from class ddml.
 ... Currently unused.

Value

A named list of length nfit.

Examples

```
y = AE98[, "worked"]
D = AE98[, "morekids"]
X = AE98[, c("age", "agefst", "black", "hisp", "othrace")]
fit = ddml_plm(y, D, X,
  learners = list(
    list(what = ols),
    list(what = mdl_glmnet)),
  ensemble_type = c("nnls", "singlebest"),
  sample_folds = 2, silent = TRUE)
as.list(fit)
```

as.list.ddml_rep	<i>Split a ddml_rep Object by Ensemble Type</i>
------------------	---

Description

Returns a named list of single-ensemble ddml_rep objects.

Usage

```
## S3 method for class 'ddml_rep'
as.list(x, ...)
```

Arguments

x	A ddml_rep object.
...	Currently unused.

Value

A named list of ddml_rep objects.

Examples

```
y = AE98[, "worked"]
D = AE98[, "morekids"]
X = AE98[, c("age", "agefst", "black", "hisp", "othrace")]
reps = ddml_replicate(ddml_plm, y = y, D = D, X = X,
                      learners = list(what = ols),
                      resamples = 3,
                      sample_folds = 2,
                      silent = TRUE)

as.list(reps)
```

as.list.ral	<i>Split a RAL Object by Fit</i>
-------------	----------------------------------

Description

Returns a named list of single-fit ral objects, one per column of coefficients. This is the primary mechanism for passing multi-ensemble results to **modelsummary**.

Usage

```
## S3 method for class 'ral'
as.list(x, ...)
```

Arguments

x An object inheriting from class `ral`.
... Currently unused.

Value

A named list of `ral` objects, each with `nfit = 1`.

See Also

[ral](#), [lincom](#)

as.list.ral_rep	<i>Split a RAL Rep Object by Fit</i>
-----------------	--------------------------------------

Description

Returns a named list of single-fit `ral_rep` objects. Each element aggregates across resamples for a single ensemble type.

Usage

```
## S3 method for class 'ral_rep'  
as.list(x, ...)
```

Arguments

x An object inheriting from class `ral_rep`.
... Currently unused.

Value

A named list of `ral_rep` objects.

coef.ral	<i>Extract Coefficients from a RAL Object</i>
----------	---

Description

Extract Coefficients from a RAL Object

Usage

```
## S3 method for class 'ral'  
coef(object, ...)
```

Arguments

object	An object inheriting from class ral.
...	Currently unused.

Value

Named vector (single fit) or matrix (multiple fits).

coef.ral_rep	<i>Extract Aggregated Coefficients</i>
--------------	--

Description

Extract Aggregated Coefficients

Usage

```
## S3 method for class 'ral_rep'  
coef(object, aggregation = c("median", "mean", "spectral"), ...)
```

Arguments

object	An object inheriting from class ral_rep.
aggregation	Character string: "median" (default), "mean", or "spectral".
...	Currently unused.

Value

Named vector (single fit) or matrix (multiple).

 confint.ral

Confidence Intervals for RAL Estimators

Description

Computes confidence intervals for one or more parameters.

Usage

```
## S3 method for class 'ral'
confint(
  object,
  parm = NULL,
  level = 0.95,
  fit_idx = 1,
  type = "HC1",
  uniform = FALSE,
  bootstraps = 999L,
  ...
)
```

Arguments

object	An object inheriting from class <code>ral</code> .
parm	A specification of which parameters are to be given confidence intervals, either a vector of numbers or a vector of names. If missing, all parameters are considered.
level	Confidence level. Default 0.95.
fit_idx	Integer index of the fit. Defaults to 1.
type	Character. HC type. Default "HC1".
uniform	Logical. If TRUE, computes uniform confidence bands using the multiplier bootstrap. Default FALSE.
bootstraps	Integer number of bootstrap draws. Only used when <code>uniform = TRUE</code> . Default 999.
...	Currently unused.

Value

A matrix with columns for lower and upper bounds. When `uniform = TRUE`, the attribute `"crit_val"` contains the critical value.

References

Chernozhukov V, Chetverikov D, Kato K (2013). "Gaussian approximations and multiplier bootstrap for maxima of sums of high-dimensional random vectors." *Annals of Statistics*, 41(6), 2786-2819.

See Also[vcov.ral](#)

confint.ral_rep

*Confidence Intervals for RAL Rep Objects***Description**

Confidence Intervals for RAL Rep Objects

Usage

```
## S3 method for class 'ral_rep'
confint(
  object,
  parm = NULL,
  level = 0.95,
  fit_idx = 1,
  aggregation = c("median", "mean", "spectral"),
  type = "HC1",
  uniform = FALSE,
  bootstraps = 999L,
  ...
)
```

Arguments

object	An object inheriting from class <code>ral_rep</code> .
parm	Parameter specification (names or indices).
level	Confidence level. Default 0.95.
fit_idx	Integer index of the fit. Defaults to 1.
aggregation	Character string. Aggregation rule.
type	Character. HC type. Default "HC1".
uniform	Logical. Uniform bands via multiplier bootstrap? Default FALSE.
bootstraps	Integer. Bootstrap draws. Default 999.
...	Currently unused.

Value

A matrix with columns for lower and upper bounds. When `uniform = TRUE`, the attribute `"crit_val"` contains the aggregated critical value.

References

Chernozhukov V, Chetverikov D, Kato K (2013). "Gaussian approximations and multiplier bootstrap for maxima of sums of high-dimensional random vectors." *Annals of Statistics*, 41(6), 2786-2819.

crosspred

Cross-Fitted Predictions Using Stacking

Description

Cross-fitted predictions using stacking.

Usage

```
crosspred(
  y,
  X,
  learners,
  sample_folds = 10,
  ensemble_type = "average",
  cv_folds = 10,
  custom_ensemble_weights = NULL,
  cluster_variable = seq_along(y),
  subsamples = NULL,
  cv_subsamples = NULL,
  cv_subsamples_list = NULL,
  silent = FALSE,
  auxiliary_X = NULL,
  parallel = NULL
)
```

Arguments

y	The outcome variable.
X	A (sparse) matrix of predictive variables.
learners	learners is a list of lists, each containing three named elements: • what The base learner function. The function must be such that it predicts a named input y using a named input X. • args Optional arguments to be passed to what. • assign_X An optional vector of column indices corresponding to variables in X that are passed to the base learner. Omission of the args element results in default arguments being used in what. Omission of assign_X results in inclusion of all predictive variables in X.
sample_folds	Number of cross-fitting folds.
ensemble_type	Ensemble method to combine base learners into final estimate of the conditional expectation functions. Possible values are: • "nnls" Non-negative least squares. • "nnls1" Non-negative least squares with the constraint that all weights sum to one.

	• "singlebest" Select base learner with minimum MSPE. • "ols" Ordinary least squares. • "average" Simple average over base learners.
	Multiple ensemble types may be passed as a vector of strings.
cv_folds	Number of folds used for cross-validation.
custom_ensemble_weights	A numerical matrix with user-specified ensemble weights. Each column corresponds to a custom ensemble specification, each row corresponds to a base learner in learners (in chronological order). Optional column names are used to name the estimation results corresponding the custom ensemble specification.
cluster_variable	A vector of cluster indices.
subsamples	List of vectors with sample indices for cross-fitting.
cv_subsamples	List of lists, each corresponding to a subsample containing vectors with subsample indices for cross-validation.
cv_subsamples_list	Deprecated; use cv_subsamples instead.
silent	Boolean to silence estimation updates.
auxiliary_X	An optional list of matrices of length sample_folds, each containing additional observations to calculate predictions for.
parallel	An optional named list with parallel processing options. When NULL (the default), computation is sequential. Supported fields: - cores Number of cores to use. - export Character vector of object names to export to parallel workers (for custom learners that reference global objects). - packages Character vector of additional package names to load on workers (for custom learners that use packages not imported by ddm1).

Details

crosspred implements the cross-fitting step of the Double/Debiased Machine Learning procedure combined with stacking. It produces the cross-fitted nuisance estimates $\hat{\eta}(X_i)$ used in the Neyman orthogonal scores of all ddm1_* estimators.

Let $\{I_1, \dots, I_S\}$ be an S -fold partition of $\{1, \dots, n\}$, and denote the training set for fold s by $\mathcal{T}_s = \{1, \dots, n\} \setminus I_s$. Given J base learners, the procedure operates on each cross-fitting fold s in three steps:

Step 1 (Stacking weights). Run K -fold cross-validation on $\mathcal{T}_s$ (via [crossval](#)) to estimate the MSPE of each base learner, and solve for fold-specific stacking weights $\hat{w}_s = (\hat{w}_{1,s}, \dots, \hat{w}_{J,s})'$.

Step 2 (Fit). Fit each base learner j on the full training set $\mathcal{T}_s$, yielding $\hat{f}_{j,s}(\cdot)$.

Step 3 (Predict). For each $i \in I_s$, compute the ensemble cross-fitted prediction

$$\hat{\eta}(X_i) = \sum_{j=1}^J \hat{w}_{j,s} \hat{f}_{j,s}(X_i).$$

Since every observation belongs to exactly one fold, the result is a complete n -vector of out-of-sample predictions. Crucially, both the stacking weights $\hat{w}_s$ and the base learner fits $\hat{f}_{j,s}$ depend only on $\mathcal{T}_s$, which does not contain observation i .


```

                                "nnls1",
                                "singlebest"),
                                sample_folds = 2,
                                cv_folds = 2,
                                silent = TRUE)
dim(crosspred_res$cf_fitted) # = length(y) by length(ensemble_type)
dim(crosspred_res$cf_fitted_bylearner) # = length(y) by length(learners)

```

crossval	<i>Estimator of the Mean Squared Prediction Error Using Cross-Validation</i>
----------	--

Description

Estimator of the mean squared prediction error of different learners using cross-validation.

Usage

```

crossval(
  y,
  X,
  learners,
  cv_folds = 10,
  cluster_variable = seq_along(y),
  cv_subsamples = NULL,
  silent = FALSE,
  parallel = NULL
)

```

Arguments

y	The outcome variable.
X	A (sparse) matrix of predictive variables.
learners	learners is a list of lists, each containing three named elements: • what The base learner function. The function must be such that it predicts a named input y using a named input X. • args Optional arguments to be passed to what. • assign_X An optional vector of column indices corresponding to variables in X that are passed to the base learner. Omission of the args element results in default arguments being used in what. Omission of assign_X results in inclusion of all predictive variables in X.
cv_folds	Number of folds used for cross-validation.
cluster_variable	A vector of cluster indices.
cv_subsamples	List of vectors with sample indices for cross-validation.

<code>silent</code>	Boolean to silence estimation updates.
<code>parallel</code>	An optional named list with parallel processing options. When NULL (the default), computation is sequential. Supported fields: <code>cores</code> Number of cores to use. <code>export</code> Character vector of object names to export to parallel workers (for custom learners that reference global objects). <code>packages</code> Character vector of additional package names to load on workers (for custom learners that use packages not imported by <code>ddml</code>).

Details

`crossval` estimates the mean squared prediction error (MSPE) of J base learners via K -fold cross-validation. It is the inner workhorse of the stacking machinery used by `ensemble_weights` to determine ensemble weights.

Given a generic conditional expectation function $f_0(\cdot)$ (e.g., $E[Y|X]$, $E[D|X]$), let $\{I_1, \dots, I_K\}$ be a K -fold partition of $\{1, \dots, n\}$ and let $\hat{f}_j^{(-k)}$ denote learner j trained on all observations outside fold I_k . The out-of-sample residual for observation $i \in I_k$ is

$$\hat{e}_{i,j} = y_i - \hat{f}_j^{(-k)}(X_i).$$

Since every observation belongs to exactly one fold, this yields a complete $n \times J$ residual matrix. The cross-validated MSPE for learner j is

$$\widehat{\text{MSPE}}_j = n^{-1} \sum_{i=1}^n \hat{e}_{i,j}^2,$$

and the cross-validated R^2 is

$$\hat{R}_j^2 = 1 - \widehat{\text{MSPE}}_j / \hat{\sigma}_y^2,$$

where $\hat{\sigma}_y^2$ is the sample variance of y .

Value

`crossval` returns a list containing the following components:

- `mspe` A vector of MSPE estimates, each corresponding to a base learner (in chronological order).
- `r2` A vector of cross-validated R^2 values, each corresponding to a base learner (in chronological order).
- `cv_resid` A matrix of out-of-sample residuals, each column corresponding to a base learner (in chronological order).
- `cv_subsamples` Pass-through of `cv_subsamples`. See above.

See Also

Other utilities: `crosspred()`, `ddml()`, `diagnostics()`, `ensemble()`, `ensemble_weights()`, `shortstacking()`

Examples

```
# Construct variables from the included Angrist & Evans (1998) data
y = AE98[, "worked"]
X = AE98[, c("morekids", "age", "agefst", "black", "hisp", "othrace", "educ")]

# Compare ols, lasso, and ridge using 4-fold cross-validation
cv_res <- crossval(y, X,
  learners = list(list(what = ols),
    list(what = mdl_glmnet),
    list(what = mdl_glmnet,
      args = list(alpha = 0))),
  cv_folds = 4,
  silent = TRUE)

cv_res$mspe
```

ddml

Construct a ddml Object.

Description

Build a "ddml" object from user-supplied score components. The resulting object inherits all S3 methods available for ddml objects, including [summary.ddml](#), [confint.ral](#), [vcov.ral](#), and [tidy.ddml](#).

Usage

```
ddml(
  coefficients,
  scores,
  J,
  inf_func,
  nobs,
  coef_names,
  estimator_name,
  ensemble_type = colnames(coefficients),
  cluster_variable = seq_len(nobs),
  sample_folds = NULL,
  cv_folds = NULL,
  shortstack = FALSE,
  ensemble_weights = NULL,
  mspe = NULL,
  r2 = NULL,
  fitted = NULL,
  splits = NULL,
  call = match.call(),
  subclass = NULL,
  dinf_dtheta = NULL,
  ...
)
```

Arguments

<code>coefficients</code>	A ($p \times \text{nensb}$) matrix of estimated target parameters. Rows correspond to components of θ , columns to ensemble types.
<code>scores</code>	A 3D array of evaluated Neyman orthogonal scores with dimensions ($n \times p \times \text{nensb}$).
<code>J</code>	A 3D array of evaluated Jacobians with dimensions ($p \times p \times \text{nensb}$).
<code>inf_func</code>	A 3D array of evaluated influence functions with dimensions ($n \times p \times \text{nensb}$).
<code>nobs</code>	Number of observations.
<code>coef_names</code>	Character vector of coefficient names (length p).
<code>estimator_name</code>	Character string identifying the estimator (e.g., "My Custom Estimator").
<code>ensemble_type</code>	Character vector of ensemble types. Defaults to <code>colnames(coefficients)</code> .
<code>cluster_variable</code>	A vector of cluster indices. Defaults to <code>seq_len(nobs)</code> .
<code>sample_folds</code>	Number of cross-fitting folds used. Optional.
<code>cv_folds</code>	Number of cross-validation folds used. Optional.
<code>shortstack</code>	Logical indicating whether short-stacking was used. Default FALSE.
<code>ensemble_weights</code>	A named list of ensemble weight matrices. Optional.
<code>mspe</code>	A named list of per-learner MSPEs. Optional.
<code>r2</code>	A named list of per-learner R-squared values. Optional.
<code>fitted</code>	A named list of per-equation cross-fitted prediction objects. Optional.
<code>splits</code>	A list of sample split objects. Optional.
<code>call</code>	The matched call. Defaults to <code>match.call()</code> .
<code>subclass</code>	Optional character string for a subclass name. If provided, the object will have class <code>c(subclass, "ddml")</code> .
<code>dinf_dtheta</code>	An optional 4D array of dimensions ($\text{nobs} \times p \times p \times \text{nensb}$) containing the derivatives of the influence functions.
<code>...</code>	Additional named components to include in the object.

Value

An object of S3 class "ddml" (or `c(subclass, "ddml")` if subclass is specified). See [ddml-intro](#) for the output structure.

See Also

Other utilities: [crosspred\(\)](#), [crossval\(\)](#), [diagnostics\(\)](#), [ensemble\(\)](#), [ensemble_weights\(\)](#), [shortstacking\(\)](#)

Examples

```
# A minimal example: construct a ddml object from pre-computed
#   score components for a simple mean estimator.
n <- 100
y <- rnorm(n)
theta <- mean(y)

scores <- array(y - theta, dim = c(n, 1, 1))
J <- array(-1, dim = c(1, 1, 1))
psi_b <- list(matrix(y, ncol = 1))
psi_a <- list(array(-1, dim = c(n, 1, 1)))
inf_func <- array(y - theta, dim = c(n, 1, 1))
dinf_dtheta <- array(1, dim = c(n, 1, 1, 1))
coef <- matrix(theta, 1, 1, dimnames = list("mean", "custom"))

fit <- ddml(coefficients = coef, scores = scores, J = J,
            inf_func = inf_func, nobs = n, coef_names = "mean",
            dinf_dtheta = dinf_dtheta,
            estimator_name = "Sample Mean")
summary(fit)
```

ddml-intro

Intro to Double/Debiased Machine Learning

Description

All `ddml_*` estimators (`ddml_plm`, `ddml_pliv`, `ddml_fpliv`, `ddml_ate`, `ddml_att`, `ddml_late`, `ddml_apo`, `ddml_policy`) return objects that inherit from S3 class `"ddml"`.

Each object is a list containing the components described below. Estimator-specific fields (e.g., pass-through learner arguments) are documented on the individual estimator pages.

The `ddml()` constructor can also be used directly to build a `"ddml"` object from user-supplied score components, enabling implementation of custom DML estimators that inherit all S3 methods.

Arguments

<code>y</code>	The outcome variable.
<code>D</code>	A matrix of endogenous variables.
<code>X</code>	A (sparse) matrix of control variables.
<code>learners</code>	May take one of two forms, depending on whether a single learner or stacking with multiple learners is used for estimation of the conditional expectation functions. If a single learner is used, <code>learners</code> is a list with two named elements: • <code>what</code> The base learner function. The function must be such that it predicts a named input <code>y</code> using a named input <code>X</code>. • <code>args</code> Optional arguments to be passed to <code>what</code>. If stacking with multiple learners is used, <code>learners</code> is a list of lists, each containing three named elements:

	• what The base learner function. The function must be such that it predicts a named input <i>y</i> using a named input <i>X</i>. • args Optional arguments to be passed to what. • assign_X An optional vector of column indices corresponding to control variables in <i>X</i> that are passed to the base learner. Omission of the args element results in default arguments being used in what. Omission of assign_X results in inclusion of all variables in <i>X</i>.
sample_folds	Number of cross-fitting folds.
ensemble_type	Ensemble method to combine base learners into final estimate of the conditional expectation functions. Possible values are: • "nnls" Non-negative least squares. • "nnls1" Non-negative least squares with the constraint that all weights sum to one. • "singlebest" Select base learner with minimum MSPE. • "ols" Ordinary least squares. • "average" Simple average over base learners. Multiple ensemble types may be passed as a vector of strings.
shortstack	Boolean to use short-stacking.
cv_folds	Number of folds used for cross-validation in ensemble construction.
custom_ensemble_weights	A numerical matrix with user-specified ensemble weights. Each column corresponds to a custom ensemble specification, each row corresponds to a base learner in learners (in chronological order). Optional column names are used to name the estimation results corresponding the custom ensemble specification.
cluster_variable	A vector of cluster indices.
silent	Boolean to silence estimation updates.
parallel	An optional named list with parallel processing options. When NULL (the default), computation is sequential. Supported fields: cores Number of cores to use. export Character vector of object names to export to parallel workers (for custom learners that reference global objects). packages Character vector of additional package names to load on workers (for custom learners that use packages not imported by ddml).
fitted	An optional named list of per-equation cross-fitted predictions, typically obtained from a previous fit via <code>fit\$fitted</code> . When supplied (together with splits), base learners are not re-fitted; only ensemble weights are recomputed. This allows fast re-estimation with a different ensemble_type . See ddml_plm for an example.
splits	An optional list of sample split objects, typically obtained from a previous fit via <code>fit\$splits</code> . Must be supplied when fitted is provided. Can also be used standalone to provide pre-computed sample folds.

`save_crossval` Logical indicating whether to store the inner cross-validation residuals used for ensemble weight computation. Default TRUE. When TRUE, subsequent pass-through calls with data-driven ensembles (e.g., "nnls") reproduce per-fold weights exactly. Set to FALSE to reduce object size at the cost of approximate weight re-computation.

... Additional arguments passed to internal methods.

Details

All `ddml_*` estimators target a low-dimensional parameter θ_0 identified by a moment condition

$$E[m(W; \theta_0, \eta_0)] = 0,$$

where W denotes observed random variables and η_0 is a (potentially high-dimensional) nuisance parameter. Throughout, the score m is assumed to be *Neyman orthogonal*.

Estimation proceeds via cross-fitting: the sample is randomly partitioned into K folds $\{I_k\}_{k=1}^K$. For each fold k , nuisance parameters are estimated on the complementary folds ($\hat{\eta}_{-k}$) and the scores are evaluated on fold k . The DML estimator $\hat{\theta}$ solves

$$\frac{1}{n} \sum_{k=1}^K \sum_{i \in I_k} m(W_i; \hat{\theta}, \hat{\eta}_{-k}) = 0.$$

Inference is based on the influence function. Define the Jacobian

$$J(\theta, \eta) = E \left[\frac{\partial m(W; \theta, \eta)}{\partial \theta'} \right]$$

and the influence function

$$\phi_\theta(W_i; \theta, \eta, J) = -J^{-1} m(W_i; \theta, \eta).$$

The variance of $\hat{\theta}$ is then estimated by

$$\hat{V} = \frac{1}{n^2} \sum_i \phi_\theta(W_i; \hat{\theta}, \hat{\eta}_{-k(i)}, \hat{J}) \phi_\theta(W_i; \hat{\theta}, \hat{\eta}_{-k(i)}, \hat{J})'$$

,

where $\hat{J}$ is the sample analog of the Jacobian:

$$\hat{J} = \frac{1}{n} \sum_i \frac{\partial m(W_i; \hat{\theta}, \hat{\eta}_{-k(i)})}{\partial \theta'}.$$

HC1 and HC3 variance estimators are described in [vcov.ral](#). The leverage (see [hatvalues.ral](#)) for the DML estimator is

$$h_\theta(W_i; \theta, \eta, J) = \text{tr} \left(-J^{-1} \frac{1}{n} \frac{\partial m(W_i; \theta, \eta)}{\partial \theta'} \right),$$

and its sample analog is $\hat{h}_{\theta,i} = h_{\theta}(W_i; \hat{\theta}, \hat{\eta}_{-k(i)}, \hat{J})$, stored in `dinf_dtheta`.

Under regularity conditions and sufficient convergence of $\hat{\eta}$, the DML estimator is asymptotically normal:

$$\sqrt{n} \hat{V}^{-1/2}(\hat{\theta} - \theta_0) \xrightarrow{d} N(0, I).$$

Further details and regularity conditions are given in Chernozhukov et al. (2018). The specific forms of the score m and Jacobian J for each estimator are documented on their respective help pages (e.g., [ddml_plm](#), [ddml_ate](#)).

Common output components

- `coefficients` A matrix of estimated target parameters: rows correspond to components of θ , columns to ensemble types.
- `ensemble_weights` A named list. Each element is a weight matrix (or 3D array when `shortstack = TRUE`) showing the weight assigned to each base learner by the ensemble procedure for the corresponding nuisance equation.
- `mspe` A named list of numeric vectors containing per-learner out-of-sample MSPEs, computed from cross-fitted residuals.
- `r2` A named list of numeric vectors containing per-learner out-of-sample R-squared values.
- `inf_func` A 3D array of evaluated influence functions ($n \times p \times \text{nensb}$).
- `dinf_dtheta` An optional 4D array of dimension ($n \times p \times p \times \text{nensb}$) containing the derivatives of the influence functions with respect to θ . Used internally by [hatvalues.ral](#) for HC3 inference.
- `scores` A 3D array of evaluated Neyman orthogonal scores ($n \times p \times \text{nensb}$).
- `J` A 3D array of evaluated Jacobians ($p \times p \times \text{nensb}$).
- `fitted` A named list of per-equation cross-fitted prediction objects. Can be passed back via the `fitted` argument together with `splits` to skip cross-fitting on re-estimation.
- `splits` The data splitting structure (subsamples, CV subsamples, and any stratification indices).
- `ensemble_type` Character vector of ensemble types used.
- `cluster_variable` The cluster variable vector used for sample splitting and inference.
- `nobs` Number of observations.
- `sample_folds` Number of cross-fitting folds.
- `shortstack` Logical indicating whether short-stacking was used.
- `call` The matched call.
- `coef_names` Character vector of coefficient names.
- `estimator_name` Character string identifying the estimator (e.g., "Partially Linear Model").

S3 methods

The following generic methods are available for all `ddml` objects: [summary.ddml](#), [coef.ral](#), [vcov.ral](#), [confint.ral](#), [hatvalues.ral](#), [nobs.ral](#), [tidy.ddml](#), [glance.ddml](#), and [diagnostics](#).

References

Ahrens A, Chernozhukov V, Hansen C B, Kozbur D, Schaffer M E, Wiemann T (2026). "An Introduction to Double/Debiased Machine Learning." Journal of Economic Literature, forthcoming.

Chernozhukov V, Chetverikov D, Demirer M, Duflo E, Hansen C B, Newey W, Robins J (2018). "Double/debiased machine learning for treatment and structural parameters." The Econometrics Journal, 21(1), C1-C68.

See Also

Other ddml estimators: [ddml_apo\(\)](#), [ddml_ate\(\)](#), [ddml_attgt\(\)](#), [ddml_fpliv\(\)](#), [ddml_late\(\)](#), [ddml_pliv\(\)](#), [ddml_plm\(\)](#), [ddml_policy\(\)](#)

ddml_apo

Estimator for the Average Potential Outcome

Description

Estimator for the average potential outcome, allowing for custom weights $\omega(X)$.

Usage

```
ddml_apo(
  y,
  D,
  X,
  d = 1,
  weights = NULL,
  learners,
  learners_DX = learners,
  sample_folds = 10,
  ensemble_type = "nnls",
  shortstack = FALSE,
  cv_folds = 10,
  custom_ensemble_weights = NULL,
  custom_ensemble_weights_DX = custom_ensemble_weights,
  cluster_variable = seq_along(y),
  stratify = TRUE,
  trim = 0.01,
  silent = FALSE,
  parallel = NULL,
  fitted = NULL,
  splits = NULL,
  save_crossval = TRUE,
  ...
)
```

Arguments

<code>y</code>	The outcome variable.
<code>D</code>	The endogenous variable of interest. Can be discrete or continuous.
<code>X</code>	A (sparse) matrix of control variables.
<code>d</code>	The treatment level of interest. The default is $d = 1$.
<code>weights</code>	A numeric vector of length <code>nobs</code> specifying the weights $\omega(X)$. If <code>weights = NULL</code> (the default), a vector of 1s is used, which estimates the Average Potential Outcome (APO).
<code>learners</code>	May take one of two forms, depending on whether a single learner or stacking with multiple learners is used for estimation of the conditional expectation functions. If a single learner is used, <code>learners</code> is a list with two named elements: • <code>what</code> The base learner function. The function must be such that it predicts a named input <code>y</code> using a named input <code>X</code>. • <code>args</code> Optional arguments to be passed to <code>what</code>. If stacking with multiple learners is used, <code>learners</code> is a list of lists, each containing three named elements: • <code>what</code> The base learner function. The function must be such that it predicts a named input <code>y</code> using a named input <code>X</code>. • <code>args</code> Optional arguments to be passed to <code>what</code>. • <code>assign_X</code> An optional vector of column indices corresponding to control variables in <code>X</code> that are passed to the base learner. Omission of the <code>args</code> element results in default arguments being used in <code>what</code>. Omission of <code>assign_X</code> results in inclusion of all variables in <code>X</code>.
<code>learners_DX</code>	Optional argument to allow for different estimators of $E[D X]$. Setup is identical to <code>learners</code> .
<code>sample_folds</code>	Number of cross-fitting folds.
<code>ensemble_type</code>	Ensemble method to combine base learners into final estimate of the conditional expectation functions. Possible values are: • <code>"nnls"</code> Non-negative least squares. • <code>"nnls1"</code> Non-negative least squares with the constraint that all weights sum to one. • <code>"singlebest"</code> Select base learner with minimum MSPE. • <code>"ols"</code> Ordinary least squares. • <code>"average"</code> Simple average over base learners. Multiple ensemble types may be passed as a vector of strings.
<code>shortstack</code>	Boolean to use short-stacking.
<code>cv_folds</code>	Number of folds used for cross-validation in ensemble construction.
<code>custom_ensemble_weights</code>	A numerical matrix with user-specified ensemble weights. Each column corresponds to a custom ensemble specification, each row corresponds to a base learner in <code>learners</code> (in chronological order). Optional column names are used to name the estimation results corresponding the custom ensemble specification.

<code>custom_ensemble_weights_DX</code>	Optional argument to allow for different custom ensemble weights for <code>learners_DX</code> . Setup is identical to <code>custom_ensemble_weights</code> . Note: <code>custom_ensemble_weights</code> and <code>custom_ensemble_weights_DX</code> must have the same number of columns.
<code>cluster_variable</code>	A vector of cluster indices.
<code>stratify</code>	Boolean for stratified cross-fitting: if TRUE, subsamples are constructed to be balanced across treatment levels.
<code>trim</code>	Number in (0, 1) for trimming the estimated propensity scores at <code>trim</code> and <code>1-trim</code> .
<code>silent</code>	Boolean to silence estimation updates.
<code>parallel</code>	An optional named list with parallel processing options. When NULL (the default), computation is sequential. Supported fields: <code>cores</code> Number of cores to use. <code>export</code> Character vector of object names to export to parallel workers (for custom learners that reference global objects). <code>packages</code> Character vector of additional package names to load on workers (for custom learners that use packages not imported by <code>ddml</code>).
<code>fitted</code>	An optional named list of per-equation cross-fitted predictions, typically obtained from a previous fit via <code>fit\$fitted</code> . When supplied (together with <code>splits</code>), base learners are not re-fitted; only ensemble weights are recomputed. This allows fast re-estimation with a different <code>ensemble_type</code> . See ddml_plm for an example.
<code>splits</code>	An optional list of sample split objects. For <code>ddml_apo</code> , this must be a list with elements <code>subsamples</code> and <code>cv_subsamples</code> (and optionally <code>subsamples_byD</code> and <code>cv_subsamples_byD</code> for stratified splitting). Typically obtained from a previous fit via <code>fit\$splits</code> .
<code>save_crossval</code>	Logical indicating whether to store the inner cross-validation residuals used for ensemble weight computation. Default TRUE. When TRUE, subsequent pass-through calls with data-driven ensembles (e.g., "nnls") reproduce per-fold weights exactly. Set to FALSE to reduce object size at the cost of approximate weight re-computation.
<code>...</code>	Additional arguments passed to internal methods.

Details

Parameter of Interest: `ddml_apo` provides a Double/Debiased Machine Learning estimator for the average potential outcome. Under conditional unconfoundedness and overlap, the parameter is identified by the following reduced form conditional expectation:

$$\theta_0^{\text{APO}} = E[\omega(X)E[Y|D = d, X]],$$

where $W \equiv (Y, D, X)$ is the observed random vector and $\omega(X)$ is a known weighting function. If $\omega(X) = 1$, this parameter corresponds to the average potential outcome at treatment level d .

Nuisance Parameters: The nuisance parameters are $\eta = (\ell, r)$ taking true values $\ell_0(X) = E[Y|D = d, X]$ and $r_0(X) = \Pr(D = d|X)$.

Neyman Orthogonal Score / Moment Equation: The Neyman orthogonal score is:

$$m(W; \theta, \eta) = \left(\frac{\mathbf{1}\{D = d\}(Y - \ell(X))}{r(X)} + \ell(X) \right) \omega(X) - \theta$$

Jacobian:

$$J = -1$$

See [ddml-intro](#) for how the influence function and inference are derived from these components.

Value

ddml_apo returns an object of S3 class ddml_apo and ddml. See [ddml-intro](#) for the common output structure. Additional pass-through fields: learners, learners_DX.

See Also

Other ddml estimators: [ddml-intro](#), [ddml_ate\(\)](#), [ddml_attgt\(\)](#), [ddml_fpliv\(\)](#), [ddml_late\(\)](#), [ddml_pliv\(\)](#), [ddml_plm\(\)](#), [ddml_policy\(\)](#)

Examples

```
# Construct variables from the included Angrist & Evans (1998) data
y = AE98[, "worked"]
D = AE98[, "morekids"]
X = AE98[, c("age", "agefst", "black", "hisp", "othrace", "educ")]

# Estimate the APO for d = 1 using a single base learner, ridge.
apo_fit <- ddml_apo(y, D, X,
  learners = list(what = mdl_glmnet),
  sample_folds = 2,
  silent = TRUE)

summary(apo_fit)
```

ddml_ate

Estimator for the Average Treatment Effect

Description

Estimator for the average treatment effect and the average treatment effect on the treated.

Usage

```
ddml_ate(  
  y,  
  D,  
  X,  
  learners,  
  learners_DX = learners,  
  sample_folds = 10,  
  ensemble_type = "nnls",  
  shortstack = FALSE,  
  cv_folds = 10,  
  custom_ensemble_weights = NULL,  
  custom_ensemble_weights_DX = custom_ensemble_weights,  
  cluster_variable = seq_along(y),  
  stratify = TRUE,  
  trim = 0.01,  
  silent = FALSE,  
  parallel = NULL,  
  fitted = NULL,  
  splits = NULL,  
  save_crossval = TRUE,  
  ...  
)  
  
ddml_att(  
  y,  
  D,  
  X,  
  learners,  
  learners_DX = learners,  
  sample_folds = 10,  
  ensemble_type = "nnls",  
  shortstack = FALSE,  
  cv_folds = 10,  
  custom_ensemble_weights = NULL,  
  custom_ensemble_weights_DX = custom_ensemble_weights,  
  cluster_variable = seq_along(y),  
  stratify = TRUE,  
  trim = 0.01,  
  silent = FALSE,  
  parallel = NULL,  
  fitted = NULL,  
  splits = NULL,  
  save_crossval = TRUE,  
  ...  
)
```

Arguments

<code>y</code>	The outcome variable.
<code>D</code>	The binary endogenous variable of interest.
<code>X</code>	A (sparse) matrix of control variables.
<code>learners</code>	May take one of two forms, depending on whether a single learner or stacking with multiple learners is used for estimation of the conditional expectation functions. If a single learner is used, <code>learners</code> is a list with two named elements: • <code>what</code> The base learner function. The function must be such that it predicts a named input <code>y</code> using a named input <code>X</code>. • <code>args</code> Optional arguments to be passed to <code>what</code>. If stacking with multiple learners is used, <code>learners</code> is a list of lists, each containing three named elements: • <code>what</code> The base learner function. The function must be such that it predicts a named input <code>y</code> using a named input <code>X</code>. • <code>args</code> Optional arguments to be passed to <code>what</code>. • <code>assign_X</code> An optional vector of column indices corresponding to control variables in <code>X</code> that are passed to the base learner. Omission of the <code>args</code> element results in default arguments being used in <code>what</code>. Omission of <code>assign_X</code> results in inclusion of all variables in <code>X</code>.
<code>learners_DX</code>	Optional argument to allow for different estimators of $E[D X]$. Setup is identical to <code>learners</code> .
<code>sample_folds</code>	Number of cross-fitting folds.
<code>ensemble_type</code>	Ensemble method to combine base learners into final estimate of the conditional expectation functions. Possible values are: • <code>"nnls"</code> Non-negative least squares. • <code>"nnls1"</code> Non-negative least squares with the constraint that all weights sum to one. • <code>"singlebest"</code> Select base learner with minimum MSPE. • <code>"ols"</code> Ordinary least squares. • <code>"average"</code> Simple average over base learners. Multiple ensemble types may be passed as a vector of strings.
<code>shortstack</code>	Boolean to use short-stacking.
<code>cv_folds</code>	Number of folds used for cross-validation in ensemble construction.
<code>custom_ensemble_weights</code>	A numerical matrix with user-specified ensemble weights. Each column corresponds to a custom ensemble specification, each row corresponds to a base learner in <code>learners</code> (in chronological order). Optional column names are used to name the estimation results corresponding the custom ensemble specification.
<code>custom_ensemble_weights_DX</code>	Optional argument to allow for different custom ensemble weights for <code>learners_DX</code> . Setup is identical to <code>custom_ensemble_weights</code> . Note: <code>custom_ensemble_weights</code> and <code>custom_ensemble_weights_DX</code> must have the same number of columns.

cluster_variable	A vector of cluster indices.
stratify	Boolean for stratified cross-fitting: if TRUE, subsamples are constructed to be balanced across treatment levels.
trim	Number in (0, 1) for trimming the estimated propensity scores at trim and 1-trim.
silent	Boolean to silence estimation updates.
parallel	An optional named list with parallel processing options. When NULL (the default), computation is sequential. Supported fields: - cores Number of cores to use. - export Character vector of object names to export to parallel workers (for custom learners that reference global objects). - packages Character vector of additional package names to load on workers (for custom learners that use packages not imported by ddml).
fitted	An optional named list of per-equation cross-fitted predictions, typically obtained from a previous fit via <code>fit\$fitted</code> . When supplied (together with <code>splits</code>), base learners are not re-fitted; only ensemble weights are recomputed. This allows fast re-estimation with a different <code>ensemble_type</code> . See ddml_plm for an example.
splits	An optional list of sample split objects. For <code>ddml_ate/ddml_att</code> , recommended keys are <code>subsamples</code> , <code>subsamples_byD</code> , <code>cv_subsamples</code> , and <code>cv_subsamples_byD</code> .
save_crossval	Logical indicating whether to store the inner cross-validation residuals used for ensemble weight computation. Default TRUE. When TRUE, subsequent pass-through calls with data-driven ensembles (e.g., "nnls") reproduce per-fold weights exactly. Set to FALSE to reduce object size at the cost of approximate weight re-computation.
...	Additional arguments passed to internal methods.

Details

Parameter of Interest: `ddml_ate` and `ddml_att` provide Double/Debiased Machine Learning estimators for the average treatment effect and the average treatment effect on the treated, respectively. Under conditional unconfoundedness and overlap, the parameters are identified by the following reduced form parameters:

$$\theta_0^{\text{ATE}} = E[E[Y|D = 1, X] - E[Y|D = 0, X]]$$

and the average treatment effect on the treated (ATT) is defined as

$$\theta_0^{\text{ATT}} = E[Y|D = 1] - E[E[Y|D = 0, X]|D = 1].$$

where $W \equiv (Y, D, X)$ is the observed random vector.

Neyman Orthogonal Score: The Neyman orthogonal scores are:

$$m^{\text{ATE}}(W; \theta, \eta) = \frac{D(Y - \ell_1(X))}{r(X)} - \frac{(1 - D)(Y - \ell_0(X))}{1 - r(X)} + \ell_1(X) - \ell_0(X) - \theta$$

$$m^{\text{ATT}}(W; \theta, \eta) = \frac{D(Y - \ell_0(X))}{\pi} - \frac{r(X)(1 - D)(Y - \ell_0(X))}{\pi(1 - r(X))} - \frac{D}{\pi}\theta$$

where the nuisance parameters are $\eta = (\ell_0, \ell_1, r, \pi)$ taking true values $\ell_{d,0}(X) = E[Y|D = d, X]$, $r_0(X) = \Pr(D = 1|X)$, and $\pi_0 = \Pr(D = 1)$.

Jacobian:

$$J^{\text{ATE}} = -1$$

$$J^{\text{ATT}} = -1$$

See [ddml-intro](#) for how the influence function and inference are derived from these components.

Value

ddml_ate and ddml_att return objects of S3 class ddml_ate/ddml_att and ddml. See [ddml-intro](#) for the common output structure. Additional pass-through fields: learners, learners_DX.

See Also

Other ddml estimators: [ddml-intro](#), [ddml_apo\(\)](#), [ddml_attgt\(\)](#), [ddml_fpliv\(\)](#), [ddml_late\(\)](#), [ddml_pliv\(\)](#), [ddml_plm\(\)](#), [ddml_policy\(\)](#)

Examples

```
# Construct variables from the included Angrist & Evans (1998) data
y = AE98[, "worked"]
D = AE98[, "morekids"]
X = AE98[, c("age", "agefst", "black", "hisp", "othrace", "educ")]

# Estimate the average treatment effect using a single base learner, ridge.
ate_fit <- ddml_ate(y, D, X,
  learners = list(what = mdl_glmnet,
    args = list(alpha = 0)),
  sample_folds = 2,
  silent = TRUE)
summary(ate_fit)

# Estimate the average treatment effect using short-stacking with base
# learners ols, lasso, and ridge. We can also use custom_ensemble_weights
# to estimate the ATE using every individual base learner.
weights_everylearner <- diag(1, 3)
colnames(weights_everylearner) <- c("mdl:ols", "mdl:lasso", "mdl:ridge")
ate_fit <- ddml_ate(y, D, X,
  learners = list(list(what = ols),
    list(what = mdl_glmnet),
    list(what = mdl_glmnet,
      args = list(alpha = 0))),
  ensemble_type = 'nnls',
```

```

                                custom_ensemble_weights = weights_everylearner,
                                shortstack = TRUE,
                                sample_folds = 2,
                                silent = TRUE)
summary(ate_fit)

```

ddml_attgt

Estimator for Group-Time Average Treatment Effects

Description

Estimator for group-time average treatment effects on the treated (GT-ATT) in staggered Difference-in-Differences designs.

Usage

```

ddml_attgt(
  y,
  X = NULL,
  t,
  G,
  learners,
  learners_qX = learners,
  sample_folds = 10,
  ensemble_type = "nnls",
  shortstack = FALSE,
  cv_folds = 10,
  custom_ensemble_weights = NULL,
  custom_ensemble_weights_qX = custom_ensemble_weights,
  cluster_variable = seq_len(nrow(as.matrix(y))),
  trim = 0.01,
  control_group = c("notyettreated", "nevertreated"),
  anticipation = 0,
  silent = FALSE,
  parallel = NULL,
  fitted = NULL,
  splits = NULL,
  save_crossval = TRUE,
  ...
)

```

Arguments

y	An $n \times T$ numeric matrix of outcomes. Row i corresponds to unit i , column j to time period $t[j]$.
X	An $n \times p$ matrix of time-invariant covariates, or NULL.

<code>t</code>	A numeric vector of length T giving the time period labels (must match columns of y).
<code>G</code>	A numeric vector of length n . Entry i is the first treatment period for unit i . Use $\emptyset$ or Inf for never-treated units.
<code>learners</code>	May take one of two forms, depending on whether a single learner or stacking with multiple learners is used for estimation of the conditional expectation functions. If a single learner is used, <code>learners</code> is a list with two named elements: • <code>what</code> The base learner function. The function must be such that it predicts a named input y using a named input X. • <code>args</code> Optional arguments to be passed to <code>what</code>. If stacking with multiple learners is used, <code>learners</code> is a list of lists, each containing three named elements: • <code>what</code> The base learner function. The function must be such that it predicts a named input y using a named input X. • <code>args</code> Optional arguments to be passed to <code>what</code>. • <code>assign_X</code> An optional vector of column indices corresponding to control variables in X that are passed to the base learner. Omission of the <code>args</code> element results in default arguments being used in <code>what</code>. Omission of <code>assign_X</code> results in inclusion of all variables in X.
<code>learners_qX</code>	Optional argument to allow for different estimators of the cell-level propensity score $q^{(g,t)}(X)$. Setup is identical to <code>learners</code> .
<code>sample_folds</code>	Number of cross-fitting folds.
<code>ensemble_type</code>	Ensemble method to combine base learners into final estimate of the conditional expectation functions. Possible values are: • <code>"nnls"</code> Non-negative least squares. • <code>"nnls1"</code> Non-negative least squares with the constraint that all weights sum to one. • <code>"singlebest"</code> Select base learner with minimum MSPE. • <code>"ols"</code> Ordinary least squares. • <code>"average"</code> Simple average over base learners. Multiple ensemble types may be passed as a vector of strings.
<code>shortstack</code>	Boolean to use short-stacking.
<code>cv_folds</code>	Number of folds used for cross-validation in ensemble construction.
<code>custom_ensemble_weights</code>	A numerical matrix with user-specified ensemble weights. Each column corresponds to a custom ensemble specification, each row corresponds to a base learner in <code>learners</code> (in chronological order). Optional column names are used to name the estimation results corresponding the custom ensemble specification.
<code>custom_ensemble_weights_qX</code>	Optional argument to allow for different custom ensemble weights for <code>learners_qX</code> . Setup is identical to <code>custom_ensemble_weights</code> .
<code>cluster_variable</code>	A vector of cluster indices.

trim	Number in (0, 1) for trimming the estimated propensity scores at trim and 1-trim.
control_group	Character. "notyettreated" (default) uses never-treated and not-yet-treated units as controls. "neverreated" uses only never-treated units.
anticipation	Non-negative integer. Number of periods before treatment where anticipation effects may occur. Default 0.
silent	Boolean to silence estimation updates.
parallel	An optional named list with parallel processing options. When NULL (the default), computation is sequential. Supported fields: cores Number of cores to use. export Character vector of object names to export to parallel workers (for custom learners that reference global objects). packages Character vector of additional package names to load on workers (for custom learners that use packages not imported by ddml).
fitted	An optional named list of per-equation cross-fitted predictions, typically obtained from a previous fit via <code>fit\$fitted</code> . When supplied (together with <code>splits</code>), base learners are not re-fitted; only ensemble weights are recomputed. This allows fast re-estimation with a different <code>ensemble_type</code> . See ddml_plm for an example.
splits	An optional list of sample split objects, typically obtained from a previous fit via <code>fit\$splits</code> . Must be supplied when <code>fitted</code> is provided. Can also be used standalone to provide pre-computed sample folds.
save_crossval	Logical indicating whether to store the inner cross-validation residuals used for ensemble weight computation. Default TRUE. When TRUE, subsequent pass-through calls with data-driven ensembles (e.g., "nnls") reproduce per-fold weights exactly. Set to FALSE to reduce object size at the cost of approximate weight re-computation.
...	Additional arguments passed to internal methods.

Details

Parameter of Interest: `ddml_attgt` provides a Double/Debiased Machine Learning estimator for the group-time average treatment effects on the treated (GT-ATT) in the staggered adoption model. For each group g and time period t , define the differenced outcome $\Delta_g Y_{i,t} = Y_{i,t} - Y_{i,g^*}$ where g^* is the universal base period. The GT-ATT is:

$$\theta_0^{(g,t)} = E[\Delta_g Y_{i,t} | G_i = g] - E[E[\Delta_g Y_{i,t} | X_i, G_i \neq g, G_i > t] | G_i = g]$$

where $W_i \equiv (Y_{i,1}, \dots, Y_{i,T}, G_i, X_i)$ is the observed random vector.

Neyman Orthogonal Score: The Neyman orthogonal score is:

$$m^{(g,t)}(W_i; \theta, \eta) = \frac{\mathbf{1}\{G_i = g\}(\Delta_g Y_{i,t} - \ell^{(g,t)}(X_i))}{\pi^g} - \frac{q^{(g,t)}(X_i) \mathbf{1}\{G_i \neq g\} \mathbf{1}\{G_i > t\}(\Delta_g Y_{i,t} - \ell^{(g,t)}(X_i))}{\pi^g(1 - q^{(g,t)}(X_i))} - \frac{\mathbf{1}\{G_i = g\}}{\pi^g}$$

where the nuisance parameters are $\eta = (\ell, q, \pi)$ taking true values $\ell_0^{(g,t)}(X) = E[\Delta_g Y_{i,t} \mid G_i \neq g, G_i > t, X_i]$, $q_0^{(g,t)}(X) = \Pr(G_i = g \mid X_i, \{G_i = g\} \cup \{G_i > t\})$, and $\pi_0^g = \Pr(G_i = g)$.

Jacobian:

$$J^{(g,t)} = -1$$

See [ddml-intro](#) for how the influence function and inference are derived from these components.

Value

ddml_attgt returns an object of S3 class ddml_attgt and ddml. See [ddml-intro](#) for the common output structure. Additional pass-through fields: learners, learners_qX, cell_info, control_group, anticipation.

References

Callaway B, Sant'Anna P H C (2021). "Difference-in-Differences with multiple time periods." *Journal of Econometrics*, 225(2), 200-230.

Chang N-C (2020). "Double/debiased machine learning for difference-in-differences models." *Econometrics Journal*, 23(2), 177-191.

Ahrens A, Chernozhukov V, Hansen C B, Kozbur D, Schaffer M E, Wiemann T (2026). "An Introduction to Double/Debiased Machine Learning." *Journal of Economic Literature*, forthcoming.

See Also

Other ddml estimators: [ddml-intro](#), [ddml_apo\(\)](#), [ddml_ate\(\)](#), [ddml_fpliv\(\)](#), [ddml_late\(\)](#), [ddml_pliv\(\)](#), [ddml_plm\(\)](#), [ddml_policy\(\)](#)

Examples

```
set.seed(42)
n <- 200; T_ <- 4
X <- matrix(rnorm(n * 2), n, 2)
G <- sample(c(3, 4, Inf), n, replace = TRUE,
            prob = c(0.3, 0.3, 0.4))
y <- matrix(rnorm(n * T_), n, T_)
# Add treatment effect for treated units
for (i in seq_len(n)) {
  if (is.finite(G[i])) {
    for (j in seq_len(T_)) {
      if (j >= G[i]) y[i, j] <- y[i, j] + 1
    }
  }
}
fit <- ddml_attgt(y, X, t = 1:T_, G = G,
                 learners = list(what = ols),
                 sample_folds = 2,
                 silent = TRUE)
summary(fit)
```

ddml_fpliv

*Estimator for the Flexible Partially Linear IV Coefficient***Description**

Estimator for the flexible partially linear IV coefficient.

Usage

```
ddml_fpliv(
  y,
  D,
  Z,
  X,
  learners,
  learners_DXZ = learners,
  learners_DX = learners,
  sample_folds = 10,
  ensemble_type = "nnls",
  shortstack = FALSE,
  cv_folds = 10,
  custom_ensemble_weights = NULL,
  custom_ensemble_weights_DXZ = custom_ensemble_weights,
  custom_ensemble_weights_DX = custom_ensemble_weights,
  cluster_variable = seq_along(y),
  silent = FALSE,
  parallel = NULL,
  fitted = NULL,
  splits = NULL,
  save_crossval = TRUE,
  ...
)
```

Arguments

- | | |
|----------|--|
| y | The outcome variable. |
| D | A matrix of endogenous variables. |
| Z | A (sparse) matrix of instruments. |
| X | A (sparse) matrix of control variables. |
| learners | May take one of two forms, depending on whether a single learner or stacking with multiple learners is used for estimation of the conditional expectation functions. If a single learner is used, learners is a list with two named elements: <ul style="list-style-type: none"> • what The base learner function. The function must be such that it predicts a named input y using a named input X. • args Optional arguments to be passed to what. |

If stacking with multiple learners is used, `learners` is a list of lists, each containing three named elements:

- `what` The base learner function. The function must be such that it predicts a named input `y` using a named input `X`.
- `args` Optional arguments to be passed to `what`.
- `assign_X` An optional vector of column indices corresponding to control variables in `X` that are passed to the base learner.

Omission of the `args` element results in default arguments being used in `what`. Omission of `assign_X` results in inclusion of all variables in `X`.

`learners_DXZ`, `learners_DX`

Optional arguments to allow for different base learners for estimation of $E[D|X, Z]$, $E[D|X]$. Setup is identical to `learners`.

`sample_folds` Number of cross-fitting folds.

`ensemble_type` Ensemble method to combine base learners into final estimate of the conditional expectation functions. Possible values are:

- `"nnls"` Non-negative least squares.
- `"nnls1"` Non-negative least squares with the constraint that all weights sum to one.
- `"singlebest"` Select base learner with minimum MSPE.
- `"ols"` Ordinary least squares.
- `"average"` Simple average over base learners.

Multiple ensemble types may be passed as a vector of strings.

`shortstack` Boolean to use short-stacking.

`cv_folds` Number of folds used for cross-validation in ensemble construction.

`custom_ensemble_weights`

A numerical matrix with user-specified ensemble weights. Each column corresponds to a custom ensemble specification, each row corresponds to a base learner in `learners` (in chronological order). Optional column names are used to name the estimation results corresponding the custom ensemble specification.

`custom_ensemble_weights_DXZ`, `custom_ensemble_weights_DX`

Optional arguments to allow for different custom ensemble weights for `learners_DXZ`, `learners_DX`. Setup is identical to `custom_ensemble_weights`. Note: `custom_ensemble_weights` and `custom_ensemble_weights_DXZ`, `custom_ensemble_weights_DX` must have the same number of columns.

`cluster_variable`

A vector of cluster indices.

`silent` Boolean to silence estimation updates.

`parallel` An optional named list with parallel processing options. When NULL (the default), computation is sequential. Supported fields:

`cores` Number of cores to use.

`export` Character vector of object names to export to parallel workers (for custom learners that reference global objects).

`packages` Character vector of additional package names to load on workers (for custom learners that use packages not imported by `ddml`).

<code>fitted</code>	An optional named list of per-equation cross-fitted predictions, typically obtained from a previous fit via <code>fit\$fitted</code> . When supplied (together with <code>splits</code>), base learners are not re-fitted; only ensemble weights are recomputed. This allows fast re-estimation with a different <code>ensemble_type</code> . See ddml_plm for an example.
<code>splits</code>	An optional list of sample split objects, typically obtained from a previous fit via <code>fit\$splits</code> . Must be supplied when <code>fitted</code> is provided. Can also be used standalone to provide pre-computed sample folds.
<code>save_crossval</code>	Logical indicating whether to store the inner cross-validation residuals used for ensemble weight computation. Default <code>TRUE</code> . When <code>TRUE</code> , subsequent pass-through calls with data-driven ensembles (e.g., "nnls") reproduce per-fold weights exactly. Set to <code>FALSE</code> to reduce object size at the cost of approximate weight re-computation.
<code>...</code>	Additional arguments passed to internal methods.

Details

Parameter of Interest: `ddml_fpliv` provides a Double/Debiased Machine Learning estimator for the flexible partially linear instrumental variable (IV) coefficient θ_0 , defined by the partially linear IV model:

$$Y = \theta_0 D + g_0(X) + \varepsilon, \quad E[\varepsilon|X, Z] = 0,$$

where $W \equiv (Y, D, X, Z, \varepsilon)$ is a random vector such that $E[\text{Var}(E[D|X, Z]|X)] \neq 0$, and $g_0(X)$ is an unknown nuisance function.

Neyman Orthogonal Score: The Neyman orthogonal score is:

$$m(W; \theta, \eta) = [(Y - \ell(X)) - \theta(D - r(X))](v(X, Z) - r(X))$$

where the nuisance parameters are $\eta = (\ell, r, v)$ taking true values $\ell_0(X) = E[Y|X]$, $r_0(X) = E[D|X]$, and $v_0(X, Z) = E[D|X, Z]$.

Jacobian:

$$J = -E[(D - r(X))(v(X, Z) - r(X))^\top]$$

See [ddml-intro](#) for how the influence function and inference are derived from these components.

Value

`ddml_fpliv` returns an object of S3 class `ddml_fpliv` and `ddml`. See [ddml-intro](#) for the common output structure. Additional pass-through fields: `learners`, `learners_DXZ`, `learners_DX`.

See Also

Other ddml estimators: [ddml-intro](#), [ddml_apo\(\)](#), [ddml_ate\(\)](#), [ddml_attgt\(\)](#), [ddml_late\(\)](#), [ddml_pliv\(\)](#), [ddml_plm\(\)](#), [ddml_policy\(\)](#)

Examples

```
# Construct variables from the included Angrist & Evans (1998) data
y = AE98[, "worked"]
D = AE98[, "morekids"]
Z = AE98[, "samesex", drop = FALSE]
X = AE98[, c("age", "agefst", "black", "hisp", "othrace", "educ")]

# Estimate the partially linear IV model using a single base learner, ridge.
fpliv_fit <- ddml_fpliv(y, D, Z, X,
                      learners = list(what = mdl_glmnet,
                                      args = list(alpha = 0)),
                      sample_folds = 2,
                      silent = TRUE)

summary(fpliv_fit)
```

ddml_late

Estimator for the Local Average Treatment Effect

Description

Estimator for the local average treatment effect.

Usage

```
ddml_late(
  y,
  D,
  Z,
  X,
  learners,
  learners_DXZ = learners,
  learners_ZX = learners,
  sample_folds = 10,
  ensemble_type = "nnls",
  shortstack = FALSE,
  cv_folds = 10,
  custom_ensemble_weights = NULL,
  custom_ensemble_weights_DXZ = custom_ensemble_weights,
  custom_ensemble_weights_ZX = custom_ensemble_weights,
  cluster_variable = seq_along(y),
  stratify = TRUE,
  trim = 0.01,
  silent = FALSE,
  parallel = NULL,
  fitted = NULL,
  splits = NULL,
  save_crossval = TRUE,
  ...
)
```

Arguments

y	The outcome variable.
D	A matrix of endogenous variables.
Z	Binary instrumental variable.
X	A (sparse) matrix of control variables.
learners	May take one of two forms, depending on whether a single learner or stacking with multiple learners is used for estimation of the conditional expectation functions. If a single learner is used, learners is a list with two named elements: • what The base learner function. The function must be such that it predicts a named input y using a named input X. • args Optional arguments to be passed to what. If stacking with multiple learners is used, learners is a list of lists, each containing three named elements: • what The base learner function. The function must be such that it predicts a named input y using a named input X. • args Optional arguments to be passed to what. • assign_X An optional vector of column indices corresponding to control variables in X that are passed to the base learner. Omission of the args element results in default arguments being used in what. Omission of assign_X results in inclusion of all variables in X.
learners_DXZ, learners_ZX	Optional arguments to allow for different base learners for estimation of $E[D X, Z]$, $E[Z X]$. Setup is identical to learners.
sample_folds	Number of cross-fitting folds.
ensemble_type	Ensemble method to combine base learners into final estimate of the conditional expectation functions. Possible values are: • "nnls" Non-negative least squares. • "nnls1" Non-negative least squares with the constraint that all weights sum to one. • "singlebest" Select base learner with minimum MSPE. • "ols" Ordinary least squares. • "average" Simple average over base learners. Multiple ensemble types may be passed as a vector of strings.
shortstack	Boolean to use short-stacking.
cv_folds	Number of folds used for cross-validation in ensemble construction.
custom_ensemble_weights	A numerical matrix with user-specified ensemble weights. Each column corresponds to a custom ensemble specification, each row corresponds to a base learner in learners (in chronological order). Optional column names are used to name the estimation results corresponding the custom ensemble specification.

<code>custom_ensemble_weights_DXZ, custom_ensemble_weights_ZX</code>	Optional arguments to allow for different custom ensemble weights for <code>learners_DXZ, learners_ZX</code> . Setup is identical to <code>custom_ensemble_weights</code> . Note: <code>custom_ensemble_weights</code> and <code>custom_ensemble_weights_DXZ, custom_ensemble_weights_ZX</code> must have the same number of columns.
<code>cluster_variable</code>	A vector of cluster indices.
<code>stratify</code>	Boolean for stratified cross-fitting: if TRUE, subsamples are constructed to be balanced across treatment levels.
<code>trim</code>	Number in (0, 1) for trimming the estimated propensity scores at <code>trim</code> and <code>1-trim</code> .
<code>silent</code>	Boolean to silence estimation updates.
<code>parallel</code>	An optional named list with parallel processing options. When NULL (the default), computation is sequential. Supported fields: <code>cores</code> Number of cores to use. <code>export</code> Character vector of object names to export to parallel workers (for custom learners that reference global objects). <code>packages</code> Character vector of additional package names to load on workers (for custom learners that use packages not imported by <code>ddml</code>).
<code>fitted</code>	An optional named list of per-equation cross-fitted predictions, typically obtained from a previous fit via <code>fit\$fitted</code> . When supplied (together with <code>splits</code>), base learners are not re-fitted; only ensemble weights are recomputed. This allows fast re-estimation with a different <code>ensemble_type</code> . See <code>ddml_plm</code> for an example.
<code>splits</code>	An optional list of sample split objects. For <code>ddml_late</code> , recommended keys are <code>subsamples</code> , <code>subsamples_byZ</code> , <code>cv_subsamples</code> , and <code>cv_subsamples_byZ</code> .
<code>save_crossval</code>	Logical indicating whether to store the inner cross-validation residuals used for ensemble weight computation. Default TRUE. When TRUE, subsequent pass-through calls with data-driven ensembles (e.g., "nnls") reproduce per-fold weights exactly. Set to FALSE to reduce object size at the cost of approximate weight re-computation.
<code>...</code>	Additional arguments passed to internal methods.

Details

Parameter of Interest: `ddml_late` provides a Double/Debiased Machine Learning estimator for the local average treatment effect. Under the standard instrumental variable assumptions (conditional independence, exclusion restriction, relevance, and monotonicity) with a binary instrument Z and a binary treatment D , the parameter is identified by the following reduced form parameter:

$$\theta_0^{\text{LATE}} = \frac{E[E[Y|Z=1, X] - E[Y|Z=0, X]]}{E[E[D|Z=1, X] - E[D|Z=0, X]]}$$

where $W \equiv (Y, D, X, Z)$ is the observed random vector.

Nuisance Parameters: The nuisance parameters are $\eta = (\ell_0, \ell_1, r_0, r_1, p)$ taking true values $\ell_{z,0}(X) = E[Y|Z=z, X]$, $r_{z,0}(X) = E[D|Z=z, X]$, and $p_0(X) = \Pr(Z=1|X)$.

Neyman Orthogonal Score / Moment Equation: The Neyman orthogonal score is:

$$m(W; \theta, \eta) = \frac{Z(Y - \ell_1(X))}{p(X)} - \frac{(1 - Z)(Y - \ell_0(X))}{1 - p(X)} + \ell_1(X) - \ell_0(X) - \theta \left(\frac{Z(D - r_1(X))}{p(X)} - \frac{(1 - Z)(D - r_0(X))}{1 - p(X)} + r_1 \right)$$

Jacobian:

$$J = -E[r_1(X) - r_0(X)]$$

See [ddml-intro](#) for how the influence function and inference are derived from these components.

Value

ddml_late returns an object of S3 class ddml_late and ddml. See [ddml-intro](#) for the common output structure. Additional pass-through fields: learners, learners_DXZ, learners_ZX.

References

Imbens G, Angrist J (1994). "Identification and Estimation of Local Average Treatment Effects." *Econometrica*, 62(2), 467-475.

See Also

Other ddml estimators: [ddml-intro](#), [ddml_apo\(\)](#), [ddml_ate\(\)](#), [ddml_attgt\(\)](#), [ddml_fpliv\(\)](#), [ddml_pliv\(\)](#), [ddml_plm\(\)](#), [ddml_policy\(\)](#)

Examples

```
# Construct variables from the included Angrist & Evans (1998) data
y = AE98[, "worked"]
D = AE98[, "morekids"]
Z = AE98[, "samesex"]
X = AE98[, c("age", "agefst", "black", "hisp", "othrace", "educ")]

# Estimate the local average treatment effect using a single base learner,
# ridge.
late_fit <- ddml_late(y, D, Z, X,
                     learners = list(what = mdl_glmnet,
                                     args = list(alpha = 0)),
                     sample_folds = 2,
                     silent = TRUE)

summary(late_fit)

# Estimate the local average treatment effect using short-stacking with base
# learners ols, lasso, and ridge. We can also use custom_ensemble_weights
# to estimate the LATE using every individual base learner.
weights_everylearner <- diag(1, 3)
colnames(weights_everylearner) <- c("mdl:ols", "mdl:lasso", "mdl:ridge")
late_fit <- ddml_late(y, D, Z, X,
```

```

learners = list(list(what = ols),
                 list(what = mdl_glmnet),
                 list(what = mdl_glmnet,
                     args = list(alpha = 0))),
ensemble_type = 'nnls',
custom_ensemble_weights = weights_everylearner,
shortstack = TRUE,
sample_folds = 2,
silent = TRUE)

summary(late_fit)

```

ddml_pliv

Estimator for the Partially Linear IV Coefficient

Description

Estimator for the partially linear IV coefficient.

Usage

```

ddml_pliv(
  y,
  D,
  Z,
  X,
  learners,
  learners_DX = learners,
  learners_ZX = learners,
  sample_folds = 10,
  ensemble_type = "nnls",
  shortstack = FALSE,
  cv_folds = 10,
  custom_ensemble_weights = NULL,
  custom_ensemble_weights_DX = custom_ensemble_weights,
  custom_ensemble_weights_ZX = custom_ensemble_weights,
  cluster_variable = seq_along(y),
  silent = FALSE,
  parallel = NULL,
  fitted = NULL,
  splits = NULL,
  save_crossval = TRUE,
  ...
)

```

Arguments

y	The outcome variable.
D	A matrix of endogenous variables.
Z	A matrix of instruments.
X	A (sparse) matrix of control variables.
learners	May take one of two forms, depending on whether a single learner or stacking with multiple learners is used for estimation of the conditional expectation functions. If a single learner is used, learners is a list with two named elements: • what The base learner function. The function must be such that it predicts a named input y using a named input X. • args Optional arguments to be passed to what. If stacking with multiple learners is used, learners is a list of lists, each containing three named elements: • what The base learner function. The function must be such that it predicts a named input y using a named input X. • args Optional arguments to be passed to what. • assign_X An optional vector of column indices corresponding to control variables in X that are passed to the base learner. Omission of the args element results in default arguments being used in what. Omission of assign_X results in inclusion of all variables in X.
learners_DX, learners_ZX	Optional arguments to allow for different base learners for estimation of $E[D X]$, $E[Z X]$. Setup is identical to learners.
sample_folds	Number of cross-fitting folds.
ensemble_type	Ensemble method to combine base learners into final estimate of the conditional expectation functions. Possible values are: • "nnls" Non-negative least squares. • "nnls1" Non-negative least squares with the constraint that all weights sum to one. • "singlebest" Select base learner with minimum MSPE. • "ols" Ordinary least squares. • "average" Simple average over base learners. Multiple ensemble types may be passed as a vector of strings.
shortstack	Boolean to use short-stacking.
cv_folds	Number of folds used for cross-validation in ensemble construction.
custom_ensemble_weights	A numerical matrix with user-specified ensemble weights. Each column corresponds to a custom ensemble specification, each row corresponds to a base learner in learners (in chronological order). Optional column names are used to name the estimation results corresponding the custom ensemble specification.

custom_ensemble_weights_DX, custom_ensemble_weights_ZX	Optional arguments to allow for different custom ensemble weights for learners_DX, learners_ZX. Setup is identical to custom_ensemble_weights. Note: custom_ensemble_weights and custom_ensemble_weights_DX, custom_ensemble_weights_ZX must have the same number of columns.
cluster_variable	A vector of cluster indices.
silent	Boolean to silence estimation updates.
parallel	An optional named list with parallel processing options. When NULL (the default), computation is sequential. Supported fields: - cores Number of cores to use. - export Character vector of object names to export to parallel workers (for custom learners that reference global objects). - packages Character vector of additional package names to load on workers (for custom learners that use packages not imported by ddml).
fitted	An optional named list of per-equation cross-fitted predictions, typically obtained from a previous fit via fit\$fitted. When supplied (together with splits), base learners are not re-fitted; only ensemble weights are recomputed. This allows fast re-estimation with a different ensemble_type. See ddml_plm for an example.
splits	An optional list of sample split objects, typically obtained from a previous fit via fit\$splits. Must be supplied when fitted is provided. Can also be used standalone to provide pre-computed sample folds.
save_crossval	Logical indicating whether to store the inner cross-validation residuals used for ensemble weight computation. Default TRUE. When TRUE, subsequent pass-through calls with data-driven ensembles (e.g., "nnls") reproduce per-fold weights exactly. Set to FALSE to reduce object size at the cost of approximate weight re-computation.
...	Additional arguments passed to internal methods.

Details

Parameter of Interest: ddml_pliv provides a Double/Debiased Machine Learning estimator for the partially linear instrumental variable (IV) coefficient θ_0 , defined by the partially linear IV model:

$$Y = \theta_0 D + g_0(X) + \varepsilon, \quad E[Z\varepsilon] = 0, \quad E[\varepsilon|X] = 0,$$

where $W \equiv (Y, D, X, Z, \varepsilon)$ is a random vector such that $E[Cov(D, Z|X)] \neq 0$, and $g_0(X)$ is an unknown nuisance function.

Neyman Orthogonal Score: The Neyman orthogonal score is:

$$m(W; \theta, \eta) = [(Y - \ell(X)) - \theta(D - r_D(X))](Z - r_Z(X))$$

where the nuisance parameters are $\eta = (\ell, r_D, r_Z)$ taking true values $\ell_0(X) = E[Y|X]$, $r_{D,0}(X) = E[D|X]$, and $r_{Z,0}(X) = E[Z|X]$.

Jacobian:

$$J = -E[(D - r_D(X))(Z - r_Z(X))^T]$$

See [ddml-intro](#) for how the influence function and inference are derived from these components.

Value

ddml_pliv returns an object of S3 class ddml_pliv and ddml. See [ddml-intro](#) for the common output structure. Additional pass-through fields: learners, learners_DX, learners_ZX.

See Also

Other ddml estimators: [ddml-intro](#), [ddml_apo\(\)](#), [ddml_ate\(\)](#), [ddml_attgt\(\)](#), [ddml_fpliv\(\)](#), [ddml_late\(\)](#), [ddml_plm\(\)](#), [ddml_policy\(\)](#)

Examples

```
# Construct variables from the included Angrist & Evans (1998) data
y = AE98[, "worked"]
D = AE98[, "morekids"]
Z = AE98[, "samesex"]
X = AE98[, c("age", "agefst", "black", "hisp", "othrace", "educ")]

# Estimate the partially linear IV model using a single base learner, ridge.
pliv_fit <- ddml_pliv(y, D, Z, X,
  learners = list(what = mdl_glmnet,
    args = list(alpha = 0)),
  sample_folds = 2,
  silent = TRUE)

summary(pliv_fit)
```

ddml_plm

Estimator for the Partially Linear Regression Coefficient

Description

Estimator for the partially linear regression coefficient.

Usage

```
ddml_plm(
  y,
  D,
  X,
  learners,
  learners_DX = learners,
  sample_folds = 10,
  ensemble_type = "nnls",
  shortstack = FALSE,
```

```

cv_folds = 10,
custom_ensemble_weights = NULL,
custom_ensemble_weights_DX = custom_ensemble_weights,
cluster_variable = seq_along(y),
silent = FALSE,
parallel = NULL,
fitted = NULL,
splits = NULL,
save_crossval = TRUE,
...
)

```

Arguments

y	The outcome variable.
D	A matrix of endogenous variables.
X	A (sparse) matrix of control variables.
learners	May take one of two forms, depending on whether a single learner or stacking with multiple learners is used for estimation of the conditional expectation functions. If a single learner is used, learners is a list with two named elements: • what The base learner function. The function must be such that it predicts a named input y using a named input X. • args Optional arguments to be passed to what. If stacking with multiple learners is used, learners is a list of lists, each containing three named elements: • what The base learner function. The function must be such that it predicts a named input y using a named input X. • args Optional arguments to be passed to what. • assign_X An optional vector of column indices corresponding to control variables in X that are passed to the base learner. Omission of the args element results in default arguments being used in what. Omission of assign_X results in inclusion of all variables in X.
learners_DX	Optional argument to allow for different estimators of $E[D X]$. Setup is identical to learners.
sample_folds	Number of cross-fitting folds.
ensemble_type	Ensemble method to combine base learners into final estimate of the conditional expectation functions. Possible values are: • "nnls" Non-negative least squares. • "nnls1" Non-negative least squares with the constraint that all weights sum to one. • "singlebest" Select base learner with minimum MSPE. • "ols" Ordinary least squares. • "average" Simple average over base learners. Multiple ensemble types may be passed as a vector of strings.

<code>shortstack</code>	Boolean to use short-stacking.
<code>cv_folds</code>	Number of folds used for cross-validation in ensemble construction.
<code>custom_ensemble_weights</code>	A numerical matrix with user-specified ensemble weights. Each column corresponds to a custom ensemble specification, each row corresponds to a base learner in <code>learners</code> (in chronological order). Optional column names are used to name the estimation results corresponding the custom ensemble specification.
<code>custom_ensemble_weights_DX</code>	Optional argument to allow for different custom ensemble weights for <code>learners_DX</code> . Setup is identical to <code>custom_ensemble_weights</code> . Note: <code>custom_ensemble_weights</code> and <code>custom_ensemble_weights_DX</code> must have the same number of columns.
<code>cluster_variable</code>	A vector of cluster indices.
<code>silent</code>	Boolean to silence estimation updates.
<code>parallel</code>	An optional named list with parallel processing options. When NULL (the default), computation is sequential. Supported fields: <code>cores</code> Number of cores to use. <code>export</code> Character vector of object names to export to parallel workers (for custom learners that reference global objects). <code>packages</code> Character vector of additional package names to load on workers (for custom learners that use packages not imported by <code>ddml</code>).
<code>fitted</code>	An optional named list of per-equation cross-fitted predictions, typically obtained from a previous fit via <code>fit\$fitted</code> . When supplied (together with <code>splits</code>), base learners are not re-fitted; only ensemble weights are recomputed. This allows fast re-estimation with a different <code>ensemble_type</code> . See ddml_plm for an example.
<code>splits</code>	An optional list of sample split objects, typically obtained from a previous fit via <code>fit\$splits</code> . Must be supplied when <code>fitted</code> is provided. Can also be used standalone to provide pre-computed sample folds.
<code>save_crossval</code>	Logical indicating whether to store the inner cross-validation residuals used for ensemble weight computation. Default TRUE. When TRUE, subsequent pass-through calls with data-driven ensembles (e.g., "nnls") reproduce per-fold weights exactly. Set to FALSE to reduce object size at the cost of approximate weight re-computation.
<code>...</code>	Additional arguments passed to internal methods.

Details

Parameter of Interest: `ddml_plm` provides a Double/Debiased Machine Learning estimator for the partially linear regression coefficient θ_0 , defined by the partially linear regression model:

$$Y = \theta_0 D + g_0(X) + \varepsilon, \quad E[D\varepsilon] = 0, \quad E[\varepsilon|X] = 0,$$

where $W \equiv (Y, D, X, \varepsilon)$ is a random vector such that $E[\text{Var}(D|X)] \neq 0$, and $g_0(X)$ is an unknown nuisance function.

Neyman Orthogonal Score: The Neyman orthogonal score is:

$$m(W; \theta, \eta) = [(Y - \ell(X)) - \theta(D - r(X))](D - r(X))$$

where the nuisance parameters are $\eta = (\ell, r)$ taking true values $\ell_0(X) = E[Y|X]$ and $r_0(X) = E[D|X]$.

Jacobian:

$$J = -E[(D - r_0(X))(D - r_0(X))^{\top}]$$

See [ddml-intro](#) for how the influence function and inference are derived from these components.

Value

ddml_plm returns an object of S3 class ddml_plm and ddml. See [ddml-intro](#) for the common output structure. Additional pass-through fields: learners, learners_DX.

See Also

Other ddml estimators: [ddml-intro](#), [ddml_apo\(\)](#), [ddml_ate\(\)](#), [ddml_attgt\(\)](#), [ddml_fpliv\(\)](#), [ddml_late\(\)](#), [ddml_pliv\(\)](#), [ddml_policy\(\)](#)

Examples

```
# Construct variables from the included Angrist & Evans (1998) data
y = AE98[, "worked"]
D = AE98[, "morekids"]
X = AE98[, c("age", "agefst", "black", "hisp", "othrace", "educ")]

# Estimate the partially linear model using a single base learner, ridge.
plm_fit <- ddml_plm(y, D, X,
  learners = list(what = mdl_glmnet,
    args = list(alpha = 0)),
  sample_folds = 2,
  silent = TRUE)
summary(plm_fit)

# Estimate the partially linear model using short-stacking with base learners
#   ols, lasso, and ridge. We can also use custom_ensemble_weights
#   to estimate the ATE using every individual base learner.
weights_everylearner <- diag(1, 3)
colnames(weights_everylearner) <- c("mdl:ols", "mdl:lasso", "mdl:ridge")
plm_fit <- ddml_plm(y, D, X,
  learners = list(list(what = ols),
    list(what = mdl_glmnet),
    list(what = mdl_glmnet,
      args = list(alpha = 0))),
  ensemble_type = 'nnls',
  custom_ensemble_weights = weights_everylearner,
  shortstack = TRUE,
  sample_folds = 2,
```

```

                                silent = TRUE)
summary(plm_fit)

# Re-estimate with a different ensemble type using pass-through
#   (skips cross-fitting, only recomputes ensemble weights).
plm_fit2 <- ddml_plm(y, D, X,
                    learners = list(list(what = ols),
                                      list(what = mdl_glmnet),
                                      list(what = mdl_glmnet,
                                            args = list(alpha = 0))),
                    ensemble_type = 'average',
                    shortstack = TRUE,
                    sample_folds = 2,
                    silent = TRUE,
                    fitted = plm_fit$fitted,
                    splits = plm_fit$splits)
summary(plm_fit2)

```

ddml_policy

Estimator for the Multi-Action Policy Value

Description

Estimator for the expected value of a multi-action policy, with optional per-level margins.

Usage

```

ddml_policy(
  y,
  D,
  X,
  policy,
  margins = NULL,
  learners,
  learners_DX = learners,
  sample_folds = 10,
  ensemble_type = "nnls",
  shortstack = FALSE,
  cv_folds = 10,
  custom_ensemble_weights = NULL,
  custom_ensemble_weights_DX = custom_ensemble_weights,
  cluster_variable = seq_along(y),
  stratify = TRUE,
  trim = 0.01,
  silent = FALSE,
  parallel = NULL,

```

```

    fitted = NULL,
    splits = NULL,
    save_crossval = TRUE,
    ...
)

```

Arguments

y	The outcome variable.
D	The observed discrete (potentially multi-valued) treatment variable.
X	A (sparse) matrix of control variables.
policy	A vector of length nobs giving the policy-assigned treatment level for each unit. Values must be a subset of those observed in D.
margins	An optional numeric vector of length K (the number of unique values in policy) giving per-level multipliers c_k . If NULL (the default), all margins are set to one, yielding the policy value $E[Y(\pi(X))]$.
learners	May take one of two forms, depending on whether a single learner or stacking with multiple learners is used for estimation of the conditional expectation functions. If a single learner is used, learners is a list with two named elements: • what The base learner function. The function must be such that it predicts a named input y using a named input X. • args Optional arguments to be passed to what. If stacking with multiple learners is used, learners is a list of lists, each containing three named elements: • what The base learner function. The function must be such that it predicts a named input y using a named input X. • args Optional arguments to be passed to what. • assign_X An optional vector of column indices corresponding to control variables in X that are passed to the base learner. Omission of the args element results in default arguments being used in what. Omission of assign_X results in inclusion of all variables in X.
learners_DX	Optional argument to allow for different estimators of $E[D X]$. Setup is identical to learners.
sample_folds	Number of cross-fitting folds.
ensemble_type	Ensemble method to combine base learners into final estimate of the conditional expectation functions. Possible values are: • "nnls" Non-negative least squares. • "nnls1" Non-negative least squares with the constraint that all weights sum to one. • "singlebest" Select base learner with minimum MSPE. • "ols" Ordinary least squares. • "average" Simple average over base learners. Multiple ensemble types may be passed as a vector of strings.

<code>shortstack</code>	Boolean to use short-stacking.
<code>cv_folds</code>	Number of folds used for cross-validation in ensemble construction.
<code>custom_ensemble_weights</code>	A numerical matrix with user-specified ensemble weights. Each column corresponds to a custom ensemble specification, each row corresponds to a base learner in <code>learners</code> (in chronological order). Optional column names are used to name the estimation results corresponding the custom ensemble specification.
<code>custom_ensemble_weights_DX</code>	Optional argument to allow for different custom ensemble weights for <code>learners_DX</code> . Setup is identical to <code>custom_ensemble_weights</code> . Note: <code>custom_ensemble_weights</code> and <code>custom_ensemble_weights_DX</code> must have the same number of columns.
<code>cluster_variable</code>	A vector of cluster indices.
<code>stratify</code>	Boolean for stratified cross-fitting: if TRUE, subsamples are constructed to be balanced across treatment levels.
<code>trim</code>	Number in (0, 1) for trimming the estimated propensity scores at <code>trim</code> and <code>1-trim</code> .
<code>silent</code>	Boolean to silence estimation updates.
<code>parallel</code>	An optional named list with parallel processing options. When NULL (the default), computation is sequential. Supported fields: <code>cores</code> Number of cores to use. <code>export</code> Character vector of object names to export to parallel workers (for custom learners that reference global objects). <code>packages</code> Character vector of additional package names to load on workers (for custom learners that use packages not imported by <code>ddml</code>).
<code>fitted</code>	An optional named list of per-equation cross-fitted predictions, typically obtained from a previous fit via <code>fit\$fitted</code> . When supplied (together with <code>splits</code>), base learners are not re-fitted; only ensemble weights are recomputed. This allows fast re-estimation with a different <code>ensemble_type</code> . See ddml_plm for an example.
<code>splits</code>	An optional list of sample split objects. For <code>ddml_policy</code> , this is typically a named list keyed by treatment level. Typically obtained from a previous fit via <code>fit\$splits</code> .
<code>save_crossval</code>	Logical indicating whether to store the inner cross-validation residuals used for ensemble weight computation. Default TRUE. When TRUE, subsequent pass-through calls with data-driven ensembles (e.g., "nnls") reproduce per-fold weights exactly. Set to FALSE to reduce object size at the cost of approximate weight re-computation.
<code>...</code>	Additional arguments passed to internal methods.

Details

Parameter of Interest: `ddml_policy` provides a Double/Debiased Machine Learning estimator for the expected value of a multi-action policy $\pi : \text{supp}(X) \rightarrow \{d_1, \dots, d_K\}$ that assigns each unit to one of K treatment levels. The target parameter is

$$\theta_0 = \sum_{k=1}^K E[\omega_k(X) E[Y \mid D = d_k, X]],$$

where the known weight functions are $\omega_k(X) = c_k \mathbf{1}\{\pi(X) = d_k\}$, and $c_1, \dots, c_K$ are user-supplied margins. When all margins equal one (margins = NULL), the parameter reduces to the policy value $E[Y(\pi(X))]$.

Each term in the sum is a weighted average potential outcome (wAPO), estimated internally via [ddml_apo](#).

Nuisance Parameters: For each treatment level d_k , the nuisance parameters are $\eta_k = (g_k, m_k)$ taking true values $g_{k,0}(X) = E[Y \mid D = d_k, X]$ and $m_{k,0}(X) = \Pr(D = d_k \mid X)$. Only $K - 1$ propensity models are estimated; the last is derived as $m_K(X) = 1 - \sum_{k=1}^{K-1} m_k(X)$.

Neyman Orthogonal Score / Moment Equation: The Neyman orthogonal score is:

$$m(W; \theta, \eta) = \sum_{k=1}^K \omega_k(X) \left[\frac{\mathbf{1}\{D = d_k\} (Y - g_k(X))}{m_k(X)} + g_k(X) \right] - \theta$$

Jacobian:

$$J = -1$$

See [ddml-intro](#) for how the influence function and inference are derived from these components.

Value

`ddml_policy` returns an object of S3 class `ddml_policy` and `ddml`. See [ddml-intro](#) for the common output structure. Additional pass-through fields: `learners`, `learners_DX`, `policy`, `margins`.

References

- Dudik M, Langford J, Li L (2011). "Doubly Robust Policy Evaluation and Learning." Proceedings of the 28th International Conference on Machine Learning, 1097-1104.
- Zhou Z, Athey S, Wager S (2023). "Offline Multi-Action Policy Learning: Generalization and Optimization." *Operations Research*, 71(2), 698-722.

See Also

Other ddml estimators: [ddml-intro](#), [ddml_apo\(\)](#), [ddml_ate\(\)](#), [ddml_attgt\(\)](#), [ddml_fpliv\(\)](#), [ddml_late\(\)](#), [ddml_pliv\(\)](#), [ddml_plm\(\)](#)

Examples

```
# Construct variables from the included Angrist & Evans (1998) data
y = AE98[, "worked"]
D = AE98[, "morekids"]
X = AE98[, c("age", "agefst", "black", "hisp", "othrace", "educ")]

# Define a simple policy: assign D=1 if age > median, else D=0
```

```

policy <- ifelse(X[, "age"] > median(X[, "age"]), 1, 0)

# Estimate the policy value using a single base learner, ridge.
policy_fit <- ddml_policy(y, D, X,
                        policy = policy,
                        learners = list(what = mdl_glmnet),
                        sample_folds = 2,
                        silent = TRUE)

summary(policy_fit)

```

ddml_rep

*Construct a Multi-Resample DDML Object***Description**

Validates a list of ddml fits and stamps class "ddml_rep" for multi-resample aggregation.

Usage

```

ddml_rep(fits)

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

```

Arguments

<code>fits</code>	A list of at least 2 objects inheriting from class "ddml". All fits must share the same primary class, coefficient names, ensemble type, and number of observations.
<code>x</code>	A ddml_rep object.
<code>...</code>	Currently unused.

Value

An object of class `c("ddml_rep", "ral_rep")` with fields:

fits List of ddml objects.
nresamples Number of resamples.
model_type Primary class of the fits.
coef_names Coefficient names.
ensemble_type Ensemble types.
nobs Number of observations.
sample_folds Number of cross-fitting folds.
shortstack Logical, whether short-stacking was used.

See Also[ddml_replicate\(\)](#)Other ddml replication: [ddml_replicate\(\)](#)**Examples**

```

y = AE98[, "worked"]
D = AE98[, "morekids"]
X = AE98[, c("age", "agefst", "black", "hisp", "othrace")]
fits = lapply(1:3, function(r) {
  ddml_plm(y, D, X,
    learners = list(what = ols),
    sample_folds = 2, silent = TRUE)
})
reps = ddml_rep(fits)
summary(reps)

```

ddml_replicate*Replicate a DDML Estimator Across Multiple Resamples*

Description

Convenience wrapper that calls a ddml_* estimator function multiple times with independent sample splits and returns a ddml_rep object for aggregated inference.

Usage

```
ddml_replicate(fn, ..., resamples = 5, silent = FALSE)
```

Arguments

fn	A ddml_* estimator function (e.g., ddml_plm).
...	Arguments passed to fn.
resamples	Integer number of independent resamples. Must be ≥ 2 . Default 5.
silent	Logical. If TRUE, suppresses all output at both the resample level and within each estimator call. Default FALSE.

Value

An object of class "ddml_rep".

See Also[ddml_rep\(\)](#)Other ddml replication: [ddml_rep\(\)](#)

Examples

```

y = AE98[, "worked"]
D = AE98[, "morekids"]
X = AE98[, c("age", "agefst", "black", "hisp", "othrace")]
reps = ddml_replicate(ddml_plm, y = y, D = D, X = X,
                      learners = list(what = ols),
                      sample_folds = 2,
                      resamples = 3, silent = TRUE)

summary(reps)

```

diagnostics

*Stacking Diagnostics for DDML Estimators***Description**

Computes per-learner diagnostics including MSPE, R-squared, ensemble weights, and optionally cross-validation comparison (CVC) p-values for each nuisance equation.

Usage

```
diagnostics(object, cvc = FALSE, bootnum = 500, ...)
```

Arguments

object	An object of class <code>ddml</code> .
cvc	Logical. Compute CVC p-values via multiplier bootstrap? Default <code>FALSE</code> . CVC tests whether each learner is significantly outperformed by the others.
bootnum	Number of bootstrap replications for CVC. Default 500. Ignored when <code>cvc = FALSE</code> .
...	Currently unused.

Value

An object of class `ddml_diagnostics` containing per-equation learner diagnostics. Use `print()` for formatted output or `tidy()` for a flat `data.frame`.

References

Lei J (2020). "Cross-Validation With Confidence." *Journal of the American Statistical Association*, 115(532), 1978-1997.

See Also

Other utilities: [crosspred\(\)](#), [crossval\(\)](#), [ddml\(\)](#), [ensemble\(\)](#), [ensemble_weights\(\)](#), [shortstacking\(\)](#)

Examples

```
y = AE98[, "worked"]
D = AE98[, "morekids"]
X = AE98[, c("age", "agefst", "black", "hisp", "othrace")]
learners = list(list(what = ols),
                 list(what = mdl_glmnet))
plm_fit = ddml_plm(y, D, X,
                  learners = learners,
                  sample_folds = 2, silent = TRUE)
diagnostics(plm_fit, cvc = TRUE)
tidy(diagnostics(plm_fit, cvc = TRUE))
```

ensemble

Stacking Estimator Using Combinations of Base Learners

Description

Computes an ensemble of learners based on the specified aggregation type and computes cross-validated out-of-sample predictions to inform the weights.

Usage

```
ensemble(
  y,
  X,
  type = "average",
  learners,
  cv_folds = 5,
  cv_subsamples = NULL,
  cv_results = NULL,
  custom_weights = NULL,
  silent = FALSE
)
```

Arguments

y	The outcome variable.
X	The feature matrix.
type	A character string indicating the type of ensemble to compute. Default is "average".
learners	A list of base learners. See ddml-intro for the full specification.
cv_folds	Number of cross-validation folds.
cv_subsamples	Optional list of subsamples for cross-validation.
cv_results	Optional pre-computed cross-validation results.
custom_weights	Optional custom weights matrix.
silent	A boolean indicating whether to suppress progress messages.

Value

An object of class ensemble containing:

`mdl_fits` List of fitted base learners.

`weights` Computed ensemble weights.

`learners` The base learners used.

`cv_results` Cross-validation results if computed.

`mean_y` Mean of the outcome variable.

`constant_y` Boolean indicating if y is constant.

See Also

Other utilities: [crosspred\(\)](#), [crossval\(\)](#), [ddml\(\)](#), [diagnostics\(\)](#), [ensemble_weights\(\)](#), [shortstacking\(\)](#)

Examples

```
# Construct variables from the included Angrist & Evans (1998) data
y = AE98[, "worked"]
X = AE98[, c("age", "agefst", "black", "hisp", "othrace")]

# Fit an ensemble of ols, lasso, and ridge
ens_fit = ensemble(y, X,
                  type = "nnls",
                  learners = list(list(what = ols),
                                list(what = mdl_glmnet),
                                list(what = mdl_glmnet,
                                      args = list(alpha = 0))),
                  cv_folds = 5,
                  silent = TRUE)
ens_fit$weights
predict(ens_fit, newdata = X)[1:5]
```

ensemble_weights

Compute Stacking Weights for Base Learners

Description

Computes the stacking weights for an ensemble of base learners using cross-validated out-of-sample predictions.

Usage

```
ensemble_weights(
  y,
  X,
  type = "average",
  learners = NULL,
  cv_folds = 5,
  cv_subsamples = NULL,
  cv_results = NULL,
  custom_weights = NULL,
  silent = FALSE
)
```

Arguments

y	The outcome variable.
X	The feature matrix.
type	A character string or vector indicating the type(s) of ensemble weights to compute. Default is "average".
learners	Optional list of base learners. Required when <code>cv_results</code> is not supplied (learners are needed to run cross-validation). When <code>cv_results</code> is supplied, learners may be omitted; the number of learners is inferred from the cross-validation residuals. See ddml-intro for the full specification.
cv_folds	Number of cross-validation folds.
cv_subsamples	Optional list of subsamples for cross-validation.
cv_results	Optional pre-computed cross-validation results.
custom_weights	Optional custom weights matrix.
silent	A boolean indicating whether to suppress progress messages.

Value

A list containing:

`weights` A matrix of computed ensemble weights.

`cv_results` Cross-validation results used for computing weights.

See Also

Other utilities: [crosspred\(\)](#), [crossval\(\)](#), [ddml\(\)](#), [diagnostics\(\)](#), [ensemble\(\)](#), [shortstacking\(\)](#)

Examples

```
y = AE98[, "worked"]
X = AE98[, c("age", "agefst", "black", "hisp", "othrace")]

# Compute stacking weights via NNLS
ew = ensemble_weights(y, X,
```

```

                                type = "nnls",
                                learners = list(list(what = ols),
                                                list(what = mdl_glmnet)),
                                cv_folds = 5,
                                silent = TRUE)
ew$weights

```

glance.ddml

*Glance at a DDML Object***Description**

DML-specific glance method. Includes DML fields like `sample_folds`, `shortstack`, and `model_type`.

Usage

```

## S3 method for class 'ddml'
glance(x, ...)

```

Arguments

<code>x</code>	A ddml object.
<code>...</code>	Currently unused.

Value

A one-row data.frame.

Examples

```

y = AE98[, "worked"]
D = AE98[, "morekids"]
X = AE98[, c("age", "agefst", "black", "hisp", "othrace")]
plm_fit = ddml_plm(y, D, X,
                  learners = list(what = ols),
                  sample_folds = 2, silent = TRUE)
glance(plm_fit)

```

glance.ddml_rep	<i>Glance at a ddml_rep Object</i>
-----------------	------------------------------------

Description

DML-specific glance method. Includes DML fields.

Usage

```
## S3 method for class 'ddml_rep'
glance(x, ...)
```

Arguments

x	A ddml_rep object.
...	Currently unused.

Value

A one-row data.frame.

Examples

```
y = AE98[, "worked"]
D = AE98[, "morekids"]
X = AE98[, c("age", "agefst", "black", "hisp", "othrace")]
reps = ddml_replicate(ddml_plm, y = y, D = D, X = X,
                      learners = list(what = ols),
                      sample_folds = 2,
                      resamples = 3, silent = TRUE)

glance(reps)
```

glance.ral	<i>Glance at a RAL Object</i>
------------	-------------------------------

Description

Returns a one-row summary of model-level statistics.

Usage

```
## S3 method for class 'ral'
glance(x, ...)
```

Arguments

x An object inheriting from class `ral`.
 ... Currently unused.

Value

A one-row `data.frame` with columns `nobs` and `estimator_name`.

<code>glance.ral_rep</code>	<i>Glance at a RAL Rep Object</i>
-----------------------------	-----------------------------------

Description

Glance at a RAL Rep Object

Usage

```
## S3 method for class 'ral_rep'
glance(x, ...)
```

Arguments

x An object inheriting from class `ral_rep`.
 ... Currently unused.

Value

A one-row `data.frame` with columns `nobs`, `nresamples`, and `estimator_name`.

<code>hatvalues.ral</code>	<i>Extract leverage (Hat Values)</i>
----------------------------	--------------------------------------

Description

Computes the leverage (hat values) for a RAL estimator. Used internally for HC3 standard errors.

Usage

```
## S3 method for class 'ral'
hatvalues(model, fit_idx = 1, ...)
```

Arguments

model An object inheriting from class `ral`.
 fit_idx Integer index of the fit to extract leverage values for. Defaults to 1.
 ... Currently unused.

Details

The leverage for observation i is

$$h_i(\theta) = -\text{tr}\left(\frac{1}{n} \frac{\partial \phi_i(\theta)}{\partial \theta}\right).$$

The sample analog replaces $\phi_i(\theta)$ with its estimate $\hat{\phi}_i$: $\hat{h}_i = h_i(\hat{\theta})$.

The derivative $\partial \hat{\phi}_i / \partial \theta$ is stored in the `dinf_dtheta` slot. For the specific form of this derivative in the DML context, see [ddml-intro](#). For the leverage of linear combinations, see [lincom](#).

Value

A numeric vector of leverage values.

lincom	<i>Linear Combinations of DDML Coefficients</i>
--------	---

Description

Computes linear combinations $R'\hat{\theta}$ of DDML coefficient estimates.

Usage

```
lincom(fit, R, ...)

## S3 method for class 'ddml'
lincom(
  fit,
  R,
  fit_idx = NULL,
  labels = NULL,
  inf_func_R = NULL,
  dinf_dR = NULL,
  ...
)

## S3 method for class 'ddml_rep'
lincom(
  fit,
  R,
  inf_func_R = NULL,
  dinf_dR = NULL,
  fit_idx = NULL,
  labels = NULL,
  ...
)
```

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

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

Arguments

<code>fit</code>	A <code>ddml</code> or <code>ddml_rep</code> object.
<code>R</code>	A $(p \times q)$ contrast matrix. Each column defines one linear combination.
<code>...</code>	Currently unused.
<code>fit_idx</code>	Integer index of the fit to use, or <code>NULL</code> (default) for all ensemble types. When <code>NULL</code> , the output carries all ensembles from the parent fit.
<code>labels</code>	Optional character vector of length q naming the linear combinations. Defaults to column names of <code>R</code> , or "lc1", "lc2", etc.
<code>inf_func_R</code>	An optional $(n \times p \times q)$ array of influence functions $\Phi_{R,i}$ for the contrast matrix R . Slice <code>[, , k]</code> contains the IFs for column k of R . When supplied, a delta-method correction is applied to the variance. When <code>NULL</code> (default), R is treated as fixed and only the first term contributes.
<code>dinf_dR</code>	An optional $(n \times q \times q)$ array of observation-level derivatives $\partial \phi_{R,i} / \partial R$, representing the Weighting Leverage. When supplied, it is added to the Structural Leverage to form the total leverage used by HC3.
<code>x</code>	A <code>lincom</code> or <code>lincom_rep</code> object.

Details

For a p -dimensional coefficient vector $\hat{\theta}$ and a $(p \times q)$ contrast matrix R , the linear combination is $\gamma = R' \hat{\theta}$.

The influence function for γ is

$$\phi_{\gamma}(W_i; \theta, R) = R' \phi_{\theta}(W_i; \theta) + \Phi_R(W_i) \theta,$$

where ϕ_{θ} is the influence function of $\hat{\theta}$ (see [ral](#) and [ddml-intro](#)), $\Phi_R(W_i)$ is the $(p \times q)$ matrix of influence functions for the contrast matrix R (the i -th slice of `inf_func_R`), and the second term vanishes when R is fixed. The estimated influence function is

$$\hat{\phi}_{\gamma,i} = R' \hat{\phi}_{\theta,i} + \hat{\Phi}_{R,i} \hat{\theta}.$$

The leverage for γ is

$$h_{\gamma}(W_i; \theta, R) = \text{tr}(R' h_{\theta}(W_i; \theta) R + h_R(W_i)),$$

where h_{θ} is the structural leverage from the parent estimator (see [hatvalues.ral](#)) and h_R is the weighting leverage. The sample analog is

$$\hat{h}_{\gamma,i} = \text{tr}\left(R' \hat{h}_{\theta,i} R + \hat{h}_{R,i}\right),$$

where $\hat{h}_{\theta,i}$ is mapped from the parent's `dinf_dtheta` and $\hat{h}_{R,i}$ from the optional `dinf_dR` argument. The resulting `lincom` object inherits from `ral` and supports all standard inference methods: `vcov`, `confint`, `summary`, `tidy`, and `hatvalues`. For `ddml_rep` objects, `lincom` returns a `lincom_rep` inheriting from `ral_rep`.

Note that `inf_func_R` is needed for inference when R is estimated. Leverage computation further requires `dinf_dR`. See [vcov.ral](#) and [hatvalues.ral](#) for more details.

Value

An object of class "lincom" (inheriting from "ral") for `ddml` input, or "lincom_rep" (inheriting from "ral_rep") for `ddml_rep` input.

References

Chernozhukov V, Chetverikov D, Demirer M, Duflo E, Hansen C B, Newey W, Robins J (2018). "Double/debiased machine learning for treatment and structural parameters." *The Econometrics Journal*, 21(1), C1-C68.

See Also

[lincom_weights_did](#) for constructing DiD aggregation weights.

Other `ddml` inference: [summary.ddml\(\)](#)

Examples

```
set.seed(42)
n <- 200; T_ <- 4
X <- matrix(rnorm(n * 2), n, 2)
G <- sample(c(3, 4, Inf), n, replace = TRUE,
            prob = c(0.3, 0.3, 0.4))
y <- matrix(rnorm(n * T_), n, T_)
for (i in seq_len(n)) {
  if (is.finite(G[i])) {
    for (j in seq_len(T_)) {
      if (j >= G[i]) y[i, j] <- y[i, j] + 1
    }
  }
}
fit <- ddml_attgt(y, X, t = 1:T_, G = G,
                 learners = list(what = ols),
                 sample_folds = 2,
                 silent = TRUE)
# Simple contrast: first cell minus second
p <- nrow(fit$coefficients)
R <- matrix(0, p, 1)
R[1, 1] <- 1; R[2, 1] <- -1
lc <- lincom(fit, R = R, labels = "ATT1-ATT2")
```

```
summary(lc)
```

lincom_weights_did	<i>Difference-in-Differences Aggregation Weights for lincom</i>
--------------------	---

Description

Constructs the contrast matrix R and its influence function matrix `inf_func_R` for standard DiD aggregation types. The output is designed to be passed directly to [lincom](#).

Usage

```
lincom_weights_did(
  fit,
  type = c("dynamic", "group", "simple", "calendar"),
  min_e = -Inf,
  max_e = Inf,
  fit_idx = NULL
)
```

Arguments

<code>fit</code>	A <code>ddml_attgt</code> or <code>ddml_rep</code> object whose underlying fits are <code>ddml_attgt</code> .
<code>type</code>	Aggregation type: "dynamic" (default), "group", "simple", or "calendar".
<code>min_e, max_e</code>	Event-time range filter (dynamic only). Cells with event time outside $[\text{min_e}, \text{max_e}]$ are excluded.
<code>fit_idx</code>	Integer index of the fit (ensemble type) to use for computing the weighting leverage, or NULL (default) for all ensemble types. When NULL, <code>dinf_dR</code> is a 4D array.

Details

Let $\theta_0^{(g,t)}$ denote the GT-ATT from [ddml_attgt](#). Each aggregation type defines a summary parameter as a weighted average of GT-ATTs over a subset of post-treatment cells ($t \geq g$).

Dynamic (`type = "dynamic"`): aggregates by event time $e = t - g$ (Callaway and Sant'Anna, 2021, eq. 9). For each e :

$$\tau_0(e) = \sum_{g \in \mathcal{G}} \mathbf{1}\{g + e \leq T\} \Pr(G = g \mid G + e \leq T) \theta_0^{(g, g+e)}.$$

Group (`type = "group"`): aggregates by cohort g . For each g :

$$\theta(g) = \frac{1}{|\mathcal{T}_g|} \sum_{t \in \mathcal{T}_g} \theta_0^{(g,t)},$$

where $\mathcal{T}_g = \{t : t \geq g\}$ and the weights reduce to uniform within each cohort.

Calendar (type = "calendar"): aggregates by time period t . For each t :

$$\theta(t) = \sum_{g: g \leq t} \frac{P(G = g)}{\sum_{g': g' \leq t} P(G = g')} \theta_0^{(g,t)}.$$

Simple (type = "simple"): a single weighted average across all post-treatment cells:

$$\theta_{ATT} = \sum_{(g,t): t \geq g} \frac{P(G = g)}{\sum_{(g',t'): t' \geq g'} P(G = g')} \theta_0^{(g,t)}.$$

The influence function for the estimated weights is derived via the quotient rule and passed to [lincom](#) as `inf_func_R`.

Value

A list with elements:

R A $(C \times q)$ contrast matrix where C is the number of GT cells.

inf_func_R An $(n \times C \times q)$ $n \times C \times q$ array of influence functions for the contrast matrix R . Slice `[, , k]` contains the IFs for column k of R .

dinf_dR An $(n \times q \times q)$ $n \times q \times q$ array of weighting leverage. Each slice is the constant matrix $V'V$ where $V_{g,k} = \sum_{c \in \mathcal{K}_k, g_c = g} (\theta_c - \gamma_k) / S_k$. See D89 §5.3.

labels Character vector of length q naming the aggregated quantities.

References

Callaway B, Sant'Anna P H C (2021). "Difference-in-Differences with multiple time periods." *Journal of Econometrics*, 225(2), 200-230.

See Also

[lincom](#)

Examples

```
set.seed(42)
n <- 200; T_ <- 4
X <- matrix(rnorm(n * 2), n, 2)
G <- sample(c(3, 4, Inf), n, replace = TRUE,
            prob = c(0.3, 0.3, 0.4))
y <- matrix(rnorm(n * T_), n, T_)
fit <- ddml_attgt(y, X, t = 1:T_, G = G,
                 learners = list(what = ols),
                 sample_folds = 2,
                 silent = TRUE)
w <- lincom_weights_did(fit, type = "dynamic")
dyn <- lincom(fit, R = w$R,
              inf_func_R = w$inf_func_R,
```

```

      dinf_dR = w$dinf_dR,
      labels = w$labels)
summary(dyn)

```

mdl_bigGlm	<i>Wrapper for glmnet::bigGlm()</i>
------------	-------------------------------------

Description

Simple wrapper for `glmnet::bigGlm()`, designed for sparse matrices.

Usage

```
mdl_bigGlm(y, X, ...)
```

Arguments

y	The outcome variable.
X	The (sparse) feature matrix.
...	Additional arguments passed to bigGlm. See <code>glmnet::bigGlm()</code> for a complete list of arguments.

Value

mdl_bigGlm returns an object of S3 class mdl_bigGlm.

See Also

`glmnet::bigGlm()`

Other ml_wrapper: `mdl_glm()`, `mdl_glmnet()`, `mdl_ranger()`, `mdl_xgboost()`, `ols()`

Examples

```

bigglm_fit <- mdl_bigGlm(rnorm(100), matrix(rnorm(1000), 100, 10))
class(bigglm_fit)

```

mdl_glm	<i>Wrapper for stats::glm()</i>
---------	---------------------------------

Description

Simple wrapper for `stats::glm()`.

Usage

```
mdl_glm(y, X, ...)
```

Arguments

y	The outcome variable.
X	The feature matrix.
...	Additional arguments passed to glm. See <code>stats::glm()</code> for a complete list of arguments.

Value

mdl_glm returns an object of S3 class `mdl_glm` as a simple mask of the return object of `stats::glm()`.

See Also

`stats::glm()`

Other ml_wrapper: `mdl_bigGlm()`, `mdl_glmnet()`, `mdl_ranger()`, `mdl_xgboost()`, `ols()`

Examples

```
glm_fit <- mdl_glm(sample(0:1, 100, replace = TRUE),
                  matrix(rnorm(1000), 100, 10))
class(glm_fit)
```

mdl_glmnet	<i>Wrapper for glmnet::glmnet()</i>
------------	-------------------------------------

Description

Simple wrapper for `glmnet::glmnet()` and `glmnet::cv.glmnet()`.

Usage

```
mdl_glmnet(y, X, cv = TRUE, ...)
```

Arguments

y	The outcome variable.
X	The (sparse) feature matrix.
cv	Boolean to indicate use of lasso with cross-validated penalty.
...	Additional arguments passed to glmnet. See glmnet::glmnet() and glmnet::cv.glmnet() for a complete list of arguments.

Value

mdl_glmnet returns an object of S3 class mdl_glmnet as a simple mask of the return object of [glmnet::glmnet\(\)](#) or [glmnet::cv.glmnet\(\)](#).

References

Friedman J, Hastie T, Tibshirani R (2010). "Regularization Paths for Generalized Linear Models via Coordinate Descent." *Journal of Statistical Software*, 33(1), 1-22.

Simon N, Friedman J, Hastie T, Tibshirani R (2011). "Regularization Paths for Cox's Proportional Hazards Model via Coordinate Descent." *Journal of Statistical Software*, 39(5), 1-13.

See Also

[glmnet::glmnet\(\)](#), [glmnet::cv.glmnet\(\)](#)

Other ml_wrapper: [mdl_bigGlm\(\)](#), [mdl_glm\(\)](#), [mdl_ranger\(\)](#), [mdl_xgboost\(\)](#), [ols\(\)](#)

Examples

```
glmnet_fit <- mdl_glmnet(rnorm(100), matrix(rnorm(1000), 100, 10))
class(glmnet_fit)
```

mdl_ranger	<i>Wrapper for ranger::ranger()</i>
------------	-------------------------------------

Description

Simple wrapper for [ranger::ranger\(\)](#). Supports regression (default) and probability forests (set probability = TRUE).

Usage

```
mdl_ranger(y, X, ...)
```

Arguments

y	The outcome variable.
X	The feature matrix.
...	Additional arguments passed to ranger. See ranger::ranger() for a complete list of arguments.

Value

mdl_ranger returns an object of S3 class ranger as a simple mask of the return object of [ranger::ranger\(\)](#).

References

Wright M N, Ziegler A (2017). "ranger: A fast implementation of random forests for high dimensional data in C++ and R." Journal of Statistical Software 77(1), 1-17.

See Also

[ranger::ranger\(\)](#)

Other ml_wrapper: [mdl_bigGlm\(\)](#), [mdl_glm\(\)](#), [mdl_glmnet\(\)](#), [mdl_xgboost\(\)](#), [ols\(\)](#)

Examples

```
ranger_fit <- mdl_ranger(rnorm(100), matrix(rnorm(1000), 100, 10))
class(ranger_fit)
```

mdl_xgboost	<i>Wrapper for xgboost::xgboost()</i>
-------------	---------------------------------------

Description

Simple wrapper for [xgboost::xgboost\(\)](#) with some changes to the default arguments.

Usage

```
mdl_xgboost(y, X, nrounds = 500, verbosity = 0, ...)
```

Arguments

y	The outcome variable.
X	The (sparse) feature matrix.
nrounds	Number of boosting iterations / rounds. Note that the number of default boosting rounds here is not automatically tuned, and different problems will have vastly different optimal numbers of boosting rounds.
verbosity	Verbosity of printing messages. Valid values of 0 (silent), 1 (warning), 2 (info), and 3 (debug).
...	Additional arguments passed to xgboost. See xgboost::xgboost() for a complete list of arguments.

Value

mdl_xgboost returns an object of S3 class mdl_xgboost as a simple mask to the return object of [xgboost::xgboost\(\)](#).

References

Chen T, Guestrin C (2011). "Xgboost: A Scalable Tree Boosting System." Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 785-794.

See Also

`xgboost::xgboost()`

Other ml_wrapper: `mdl_bigGlm()`, `mdl_glm()`, `mdl_glmnet()`, `mdl_ranger()`, `ols()`

Examples

```
xgboost_fit <- mdl_xgboost(rnorm(50), matrix(rnorm(150), 50, 3),
                           nrounds = 1)
class(xgboost_fit)
```

nobs.ral	<i>Number of Observations in a RAL Object</i>
----------	---

Description

Number of Observations in a RAL Object

Usage

```
## S3 method for class 'ral'
nobs(object, ...)
```

Arguments

object	An object inheriting from class ral.
...	Currently unused.

Value

Integer.

ols	<i>Ordinary Least Squares</i>
-----	-------------------------------

Description

Simple implementation of ordinary least squares that computes with sparse feature matrices.

Usage

```
ols(y, X, const = TRUE, w = NULL)
```

Arguments

y	The outcome variable.
X	The feature matrix.
const	Boolean equal to TRUE if a constant should be included.
w	A vector of weights for weighted least squares.

Value

ols returns an object of S3 class `ols`. An object of class `ols` is a list containing the following components:

`coef` A vector with the regression coefficients.

`y`, `X`, `const`, `w` Pass-through of the user-provided arguments. See above.

See Also

Other `ml_wrapper`: [mdl_bigGlm\(\)](#), [mdl_glm\(\)](#), [mdl_glmnet\(\)](#), [mdl_ranger\(\)](#), [mdl_xgboost\(\)](#)

Examples

```
ols_fit <- ols(rnorm(100), cbind(rnorm(100), rnorm(100)), const = TRUE)
ols_fit$coef
```

plot.ral

*Plot Coefficients from a RAL Estimator***Description**

Plots point estimates with confidence intervals from an object inheriting from class `ral`.

Usage

```
## S3 method for class 'ral'
plot(
  x,
  parm = NULL,
  level = 0.95,
  uniform = TRUE,
  fit_idx = 1,
  type = "HC1",
  xlab = NULL,
  ylab = NULL,
  main = NULL,
  col = "black",
  pch = 19,
  lwd = 1.5,
  ...
)

## S3 method for class 'ral_rep'
plot(
  x,
  parm = NULL,
  level = 0.95,
  uniform = TRUE,
  type = "HC1",
  xlab = NULL,
  ylab = NULL,
  main = NULL,
  col = "black",
  pch = 19,
  lwd = 1.5,
  ...
)
```

Arguments

<code>x</code>	An object inheriting from class <code>ral</code> .
<code>parm</code>	A specification of which parameters to plot. Either a vector of names or indices. Default: all.

level	Numeric. Confidence level. Default 0.95.
uniform	Logical. If TRUE, uses uniform confidence bands via the multiplier bootstrap. Default TRUE.
fit_idx	Integer. Which fit to plot (column index of coefficients). Default 1.
type	Character. HC type for standard errors. Default "HC1".
xlab	Character. Label for the x-axis.
ylab	Character. Label for the y-axis.
main	Character. Title for the plot.
col	Color for points and segments. Default "black".
pch	Point character. Default 19 (solid dot).
lwd	Line width for confidence interval segments. Default 1.5.
...	Additional arguments passed to plot.default .

Value

Invisibly returns a list with components coefficients, ci, and labels.

See Also

[confint.ral](#), [summary.ral](#)

Examples

```
# Simulate a simple example
n <- 200
X <- cbind(1, stats::rnorm(n))
theta <- c(0.5, -0.3)
inf <- matrix(stats::rnorm(n * 2), n, 2)
obj <- ral(matrix(theta, 2, 1),
            array(inf, c(n, 2, 1)),
            nobs = n,
            coef_names = c("b1", "b2"))
plot(obj)
```

predict.ensemble

Predict Method for ensemble Objects

Description

Predict Method for ensemble Objects

Usage

```
## S3 method for class 'ensemble'
predict(object, newdata, ..., type = "ensemble")
```

Arguments

object	A fitted ensemble object.
newdata	A feature matrix for prediction.
...	Currently unused.
type	Character; "ensemble" (default) returns weighted ensemble predictions, "bylearner" returns the raw per-learner prediction matrix.

Value

A matrix of predictions. When type = "ensemble", one column per ensemble type; when type = "bylearner", one column per base learner.

predict.mdl_bigGlm	<i>Predict Method for mdl_bigGlm Objects</i>
--------------------	--

Description

Predict Method for mdl_bigGlm Objects

Usage

```
## S3 method for class 'mdl_bigGlm'  
predict(object, newdata = NULL, ...)
```

Arguments

object	A fitted mdl_bigGlm object.
newdata	A (sparse) feature matrix for prediction.
...	Currently unused.

Value

A numeric vector of predicted values.

predict.mdl_glm	<i>Predict Method for mdl_glm Objects</i>
-----------------	---

Description

Predict Method for mdl_glm Objects

Usage

```
## S3 method for class 'mdl_glm'  
predict(object, newdata, ...)
```

Arguments

object	A fitted mdl_glm object.
newdata	A feature matrix for prediction.
...	Additional arguments passed to predict.glm .

Value

A numeric vector of predicted response values.

predict.mdl_glmnet	<i>Predict Method for mdl_glmnet Objects</i>
--------------------	--

Description

Predict Method for mdl_glmnet Objects

Usage

```
## S3 method for class 'mdl_glmnet'  
predict(object, newdata = NULL, ...)
```

Arguments

object	A fitted mdl_glmnet object.
newdata	A (sparse) feature matrix for prediction.
...	Additional arguments passed to predict.glmnet .

Value

A numeric vector of predicted values.

predict.mdl_ranger	<i>Predict Method for mdl_ranger Objects</i>
--------------------	--

Description

Predict Method for mdl_ranger Objects

Usage

```
## S3 method for class 'mdl_ranger'  
predict(object, newdata = NULL, ...)
```

Arguments

object	A fitted mdl_ranger object.
newdata	A feature matrix for prediction.
...	Additional arguments passed to predict.ranger .

Value

A numeric vector of predicted values (probabilities for probability forests, point predictions for regression forests).

predict.mdl_xgboost	<i>Predict Method for mdl_xgboost Objects</i>
---------------------	---

Description

Predict Method for mdl_xgboost Objects

Usage

```
## S3 method for class 'mdl_xgboost'  
predict(object, newdata = NULL, ...)
```

Arguments

object	A fitted mdl_xgboost object.
newdata	A feature matrix for prediction.
...	Additional arguments passed to predict.xgb.Booster .

Value

A numeric vector of predicted values.

predict.ols	<i>Predict Method for ols Objects</i>
-------------	---------------------------------------

Description

Predict Method for ols Objects

Usage

```
## S3 method for class 'ols'  
predict(object, newdata = NULL, ...)
```

Arguments

object	A fitted ols object.
newdata	A feature matrix for prediction. If NULL, returns fitted values from the training data.
...	Currently unused.

Value

A numeric vector of predicted values.

print.ddml_diagnostics	<i>Print Stacking Diagnostics</i>
------------------------	-----------------------------------

Description

Print Stacking Diagnostics

Usage

```
## S3 method for class 'ddml_diagnostics'  
print(x, digits = 4, ...)
```

Arguments

x	An object of class ddml_diagnostics.
digits	Number of significant digits. Default 4.
...	Currently unused.

Value

x, invisibly.

ral	<i>Construct a RAL Inference Object</i>
-----	---

Description

Creates a regular asymptotically linear (RAL) inference object from pre-computed influence functions. This is the base class for all influence-function-based inference in **ddml**.

Usage

```
ral(
  coefficients,
  inf_func,
  dinf_dtheta = NULL,
  nobs,
  coef_names,
  cluster_variable = NULL,
  estimator_name = "RAL estimator",
  subclass = NULL,
  ...
)
```

Arguments

<code>coefficients</code>	A $p \times \text{nfit}$ matrix of estimated coefficients. Rows are parameters, columns are fits (e.g., ensemble types).
<code>inf_func</code>	A 3D array of dimension $n \times p \times \text{nfit}$. The influence function evaluated at each observation.
<code>dinf_dtheta</code>	Optional 4D array of dimension $n \times p \times p \times \text{nfit}$. The derivative of the influence function with respect to θ , used for HC3 leverage. If NULL, HC3 is unavailable.
<code>nobs</code>	Integer number of observations.
<code>coef_names</code>	Character vector of parameter names (length p).
<code>cluster_variable</code>	Optional vector of cluster identifiers (length n). If non-NULL, cluster-robust inference is used.
<code>estimator_name</code>	Character string for display.
<code>subclass</code>	Optional character string prepended to the class vector.
<code>...</code>	Additional named elements stored in the object.

Details

A regular asymptotically linear (RAL) estimator $\hat{\theta}$ satisfies

$$\hat{\theta} - \theta_0 = \frac{1}{n} \sum_{i=1}^n \phi(W_i; \theta_0) + o_p(n^{-1/2}),$$

where $\phi(W_i; \theta_0)$ is the *influence function*. This package stores the estimated influence function $\hat{\phi}_i \equiv \phi(W_i; \hat{\theta})$ in the `inf_func` slot.

When an observation-level derivative $\partial \hat{\phi}_i / \partial \theta$ is available (stored in `dinf_dtheta`), the estimator supports HC3 inference via leverage; see [hatvalues.ral](#).

The RAL framework is estimator-agnostic: it consumes pre-computed influence functions and does not prescribe how they are obtained. For the specific construction under cross-fitting and Neyman-orthogonal scores, see [ddml-intro](#). For linear combinations of ddml estimators, see [lincom](#).

Value

An object of class `ral` (or `c(subclass, "ral")`).

<code>ral_rep</code>	<i>Construct a Replicated RAL Inference Object</i>
----------------------	--

Description

Creates a replicated RAL inference object from a list of `ral` objects. Provides cross-resample aggregation for coefficients and covariance matrices.

Usage

```
ral_rep(fits, subclass = NULL, ...)
```

Arguments

<code>fits</code>	A list of at least 2 objects inheriting from class <code>"ral"</code> . All fits must share the same coefficient names and number of observations.
<code>subclass</code>	Optional character string prepended to the class vector.
<code>...</code>	Additional named elements stored in the object.

Value

An object of class `"ral_rep"` (or `c(subclass, "ral_rep")`).

shortstacking

Predictions using Short-Stacking

Description

Predictions using short-stacking.

Usage

```
shortstacking(
  y,
  X,
  learners,
  sample_folds = 2,
  ensemble_type = "average",
  custom_ensemble_weights = NULL,
  cluster_variable = seq_along(y),
  subsamples = NULL,
  silent = FALSE,
  auxiliary_X = NULL,
  parallel = NULL
)
```

Arguments

y	The outcome variable.
X	A (sparse) matrix of predictive variables.
learners	learners is a list of lists, each containing three named elements: • what The base learner function. The function must be such that it predicts a named input y using a named input X. • args Optional arguments to be passed to what. • assign_X An optional vector of column indices corresponding to variables in X that are passed to the base learner. Omission of the args element results in default arguments being used in what. Omission of assign_X results in inclusion of all predictive variables in X.
sample_folds	Number of cross-fitting folds.
ensemble_type	Ensemble method to combine base learners into final estimate of the conditional expectation functions. Possible values are: • "nnls" Non-negative least squares. • "nnls1" Non-negative least squares with the constraint that all weights sum to one. • "singlebest" Select base learner with minimum MSPE. • "ols" Ordinary least squares. • "average" Simple average over base learners.

Multiple ensemble types may be passed as a vector of strings.

<code>custom_ensemble_weights</code>	A numerical matrix with user-specified ensemble weights. Each column corresponds to a custom ensemble specification, each row corresponds to a base learner in <code>learners</code> (in chronological order). Optional column names are used to name the estimation results corresponding the custom ensemble specification.
<code>cluster_variable</code>	A vector of cluster indices.
<code>subsamples</code>	List of vectors with sample indices for cross-fitting.
<code>silent</code>	Boolean to silence estimation updates.
<code>auxiliary_X</code>	An optional list of matrices of length <code>sample_folds</code> , each containing additional observations to calculate predictions for.
<code>parallel</code>	An optional named list with parallel processing options. When NULL (the default), computation is sequential. Supported fields: <code>cores</code> Number of cores to use. <code>export</code> Character vector of object names to export to parallel workers (for custom learners that reference global objects). <code>packages</code> Character vector of additional package names to load on workers (for custom learners that use packages not imported by <code>ddml</code>).

Value

`shortstack` returns a list containing the following components:

<code>cf_fitted</code>	A matrix of out-of-sample predictions, each column corresponding to an ensemble type (in chronological order).
<code>weights</code>	An array, providing the weight assigned to each base learner (in chronological order) by the ensemble procedures.
<code>mspe</code>	A numeric vector of per-learner out-of-sample MSPEs, computed from cross-fitted residuals.
<code>r2</code>	A numeric vector of per-learner out-of-sample R-squared values.
<code>auxiliary_fitted</code>	When <code>auxiliary_X</code> is not NULL, a list of matrices with additional predictions.
<code>cf_fitted_bylearner</code>	A matrix of out-of-sample predictions, each column corresponding to a base learner (in chronological order).
<code>cf_resid_bylearner</code>	A matrix of per-learner out-of-sample residuals used for weight estimation.
<code>auxiliary_fitted_bylearner</code>	When <code>auxiliary_X</code> is not NULL, a list of matrices with additional predictions for each learner.

Note that unlike `crosspred`, `shortstack` always computes out-of-sample predictions for each base learner (at no additional computational cost).

References

- Ahrens A, Hansen C B, Schaffer M E, Wiemann T (2024). "Model Averaging and Double Machine Learning." *Journal of Applied Econometrics*, 40(3): 249-269.
- Wolpert D H (1992). "Stacked generalization." *Neural Networks*, 5(2), 241-259.

See Also

Other utilities: [crosspred\(\)](#), [crossval\(\)](#), [ddml\(\)](#), [diagnostics\(\)](#), [ensemble\(\)](#), [ensemble_weights\(\)](#)

Examples

```
# Construct variables from the included Angrist & Evans (1998) data
y = AE98[, "worked"]
X = AE98[, c("morekids", "age", "agefst", "black", "hisp", "othrace", "educ")]

# Compute predictions using shortstacking with base learners ols and lasso.
#   Two stacking approaches are simultaneously computed: Equally
#   weighted (ensemble_type = "average") and MSPE-minimizing with weights
#   in the unit simplex (ensemble_type = "nnls1"). Predictions for each
#   learner are also calculated.
shortstack_res <- shortstacking(y, X,
                                learners = list(list(what = ols),
                                                  list(what = mdl_glmnet)),
                                ensemble_type = c("average",
                                                  "nnls1",
                                                  "singlebest"),
                                sample_folds = 2,
                                silent = TRUE)
dim(shortstack_res$cf_fitted) # = length(y) by length(ensemble_type)
dim(shortstack_res$cf_fitted_bylearner) # = length(y) by length(learners)
```

summary.ddml

*Summary for DDML Estimators***Description**

Computes a coefficient table with estimates, standard errors, z-values, and p-values for all ensemble types. Standard errors are based on a heteroskedasticity-robust sandwich variance; see [vcov.ral](#) for the HC0/HC1/HC3 formulas.

Usage

```
## S3 method for class 'ddml'
summary(object, type = "HC1", ...)

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

Arguments

object	An object of class ddml.
type	Character. HC type ("HC0", "HC1", or "HC3"). Default "HC1".
...	Currently unused.
x	An object of class summary.ddml.
digits	Number of significant digits. Default 3.

Value

An object of class `summary.ddml` with:

`coefficients` A 3-dimensional array ($p \times 4 \times \text{nensb}$) of estimates, standard errors, z-values, and p-values.

`type` The HC type used.

`nobs` Number of observations.

`sample_folds` Number of cross-fitting folds.

`ensemble_type` Ensemble type labels.

See Also

[vcov.ral](#)

Other ddml inference: [lincom\(\)](#)

Examples

```
y = AE98[, "worked"]
D = AE98[, "morekids"]
X = AE98[, c("age", "agefst", "black", "hisp", "othrace")]
plm_fit = ddml_plm(y, D, X,
                  learners = list(what = ols),
                  sample_folds = 2, silent = TRUE)
summary(plm_fit)
summary(plm_fit, type = "HC3")
```

summary.ddml_rep	<i>Summary for ddml_rep Objects</i>
------------------	-------------------------------------

Description

DML-specific summary override. Adds ensemble type labels, folds, shortstack status to the base `ral_rep` summary.

Usage

```
## S3 method for class 'ddml_rep'
summary(
  object,
  aggregation = c("median", "mean", "spectral"),
  type = "HC1",
  ...
)

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

Arguments

object	A ddml_rep object.
aggregation	Character string: "median" (default), "mean", or "spectral".
type	Character. HC type. Default "HC1".
...	Currently unused.
x	An object of class summary.ddml_rep.
digits	Number of significant digits. Default 3.

Details

See [summary.ral_rep](#) for the aggregation formulas.

Value

An object of class "summary.ddml_rep".

References

Chernozhukov V, Chetverikov D, Demirer M, Duflo E, Hansen C B, Newey W, Robins J (2018). "Double/debiased machine learning for treatment and structural parameters." The Econometrics Journal, 21(1), C1-C68.

Examples

```
y = AE98[, "worked"]
D = AE98[, "morekids"]
X = AE98[, c("age", "agefst", "black", "hisp", "othrace")]
reps = ddml_replicate(ddml_plm, y = y, D = D, X = X,
                      learners = list(what = ols),
                      sample_folds = 2,
                      resamples = 3, silent = TRUE)

summary(reps)
summary(reps, aggregation = "mean")
```

Description

Computes a coefficient table with estimates, standard errors, z-values, and p-values.

Usage

```
## S3 method for class 'ral'
summary(object, type = "HC1", ...)

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

Arguments

object	An object inheriting from class ral.
type	Character. HC type. Default "HC1".
...	Currently unused.
x	An object of class summary.ral.
digits	Number of significant digits. Default 3.

Value

An object of class summary.ral with:

coefficients A 3-dimensional array ($p \times 4 \times \text{nfit}$).

type The HC type used.

nobs Number of observations.

summary.ral_rep	<i>Summary for RAL Rep Objects</i>
-----------------	------------------------------------

Description

Aggregates coefficient estimates and covariance matrices across independent replications.

Usage

```
## S3 method for class 'ral_rep'
summary(
  object,
  aggregation = c("median", "mean", "spectral"),
  type = "HC1",
  ...
)

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

Arguments

object	An object inheriting from class <code>ral_rep</code> .
aggregation	Character string. Aggregation rule.
type	Character. HC type. Default "HC1".
...	Currently unused.
x	An object of class <code>summary.ral_rep</code> .
digits	Number of significant digits. Default 3.

Details

Let $\hat{\theta}_s$ and $\hat{\Sigma}_s$ denote the coefficient vector and sandwich covariance matrix from replication s .

Coefficient aggregation. For "mean": $\tilde{\theta} = S^{-1} \sum_{s=1}^S \hat{\theta}_s$. For "median" and "spectral": $\tilde{\theta}_j = \text{median}_s(\hat{\theta}_{s,j})$.

Covariance aggregation. Define the inflated per-replication covariance as

$$V_s = \hat{\Sigma}_s + (\hat{\theta}_s - \tilde{\theta})(\hat{\theta}_s - \tilde{\theta})^\top.$$

For "mean": $\tilde{\Sigma} = S^{-1} \sum_{s=1}^S V_s$. For "median": $\tilde{\Sigma}_{s,ij} = \text{median}_s(V_{s,ij})$. For "spectral": solved via **CVXR**, guaranteeing PSD.

Value

An object of class "summary.ral_rep".

References

Chernozhukov V, Chetverikov D, Demirer M, Duflo E, Hansen C B, Newey W, Robins J (2018). "Double/debiased machine learning for treatment and structural parameters." *The Econometrics Journal*, 21(1), C1-C68.

tidy.ddml

Tidy a DDML Object

Description

DML-specific tidy method. Adds `ensemble_type` column labeling based on the estimator's learner/ensemble configuration. Delegates to `tidy.ral` for the base table computation.

Usage

```
## S3 method for class 'ddml'
tidy(
  x,
  ensemble_idx = 1,
  conf.int = FALSE,
  conf.level = 0.95,
  type = "HC1",
  uniform = FALSE,
  bootstraps = 999L,
  ...
)
```

Arguments

<code>x</code>	A ddml object.
<code>ensemble_idx</code>	Integer index of the ensemble type to report. Defaults to 1. Set to NULL for all.
<code>conf.int</code>	Logical. Include confidence intervals? Default FALSE.
<code>conf.level</code>	Confidence level. Default 0.95.
<code>type</code>	Character. HC type. Default "HC1".
<code>uniform</code>	Logical. Uniform CIs? Default FALSE.
<code>bootstraps</code>	Integer. Bootstrap draws. Default 999.
<code>...</code>	Currently unused.

Value

A data.frame with columns term, estimate, std.error, statistic, p.value, and ensemble_type.

Examples

```
y = AE98[, "worked"]
D = AE98[, "morekids"]
X = AE98[, c("age", "agefst", "black", "hisp", "othrace")]
plm_fit = ddml_plm(y, D, X,
  learners = list(what = ols),
  sample_folds = 2, silent = TRUE)
tidy(plm_fit)
tidy(plm_fit, conf.int = TRUE)
```

tidy.ddml_diagnostics *Tidy Stacking Diagnostics*

Description

Returns a flat data.frame of per-learner stacking diagnostics for all nuisance equations. Suitable for table creation with `kable()`, `gt()`, or `modelsummary::datasummary()`.

Usage

```
## S3 method for class 'ddml_diagnostics'
tidy(x, ...)
```

Arguments

`x` An object of class `ddml_diagnostics`.
`...` Currently unused.

Value

A data.frame with columns `equation`, `learner`, `mspe`, `r2`, `weight`, and optionally `cvc_pval`.

Examples

```
y = AE98[, "worked"]
D = AE98[, "morekids"]
X = AE98[, c("age", "agefst", "black", "hisp", "othrace")]
learners = list(list(what = ols),
                 list(what = mdl_glmnet))
plm_fit = ddml_plm(y, D, X,
                  learners = learners,
                  sample_folds = 2, silent = TRUE)
tidy(diagnostics(plm_fit, cvc = TRUE))
```

tidy.ddml_rep *Tidy a ddml_rep Object*

Description

DML-specific tidy method. Adds `ensemble_type` and `aggregation` columns. Delegates to `tidy.ral_rep` for the base table computation.

Usage

```
## S3 method for class 'ddml_rep'
tidy(
  x,
  ensemble_idx = 1,
  aggregation = c("median", "mean", "spectral"),
  type = "HC1",
  conf.int = FALSE,
  conf.level = 0.95,
  uniform = FALSE,
  bootstraps = 999L,
  ...
)
```

Arguments

<code>x</code>	A <code>ddml_rep</code> object.
<code>ensemble_idx</code>	Integer index of the ensemble type to report. Defaults to 1. Set to NULL for all ensemble types.
<code>aggregation</code>	Character string. Aggregation method.
<code>type</code>	Character. HC type. Default "HC1".
<code>conf.int</code>	Logical. Include CIs? Default FALSE.
<code>conf.level</code>	Confidence level. Default 0.95.
<code>uniform</code>	Logical. Uniform CIs? Default FALSE.
<code>bootstraps</code>	Integer. Bootstrap draws. Default 999.
<code>...</code>	Currently unused.

Value

A data.frame with columns `term`, `estimate`, `std.error`, `statistic`, `p.value`, `ensemble_type`, and `aggregation`.

See Also

[summary.ddml_rep](#) for the aggregation equations.

Examples

```
y = AE98[, "worked"]
D = AE98[, "morekids"]
X = AE98[, c("age", "agefst", "black", "hisp", "othrace")]
reps = ddml_replicate(ddml_plm, y = y, D = D, X = X,
  learners = list(what = ols),
  sample_folds = 2,
  resamples = 3, silent = TRUE)

tidy(reps)
tidy(reps, conf.int = TRUE)
```

tidy.ral

*Tidy a RAL Object***Description**

Extracts coefficient estimates, standard errors, test statistics, and p-values in a tidy data frame.

Usage

```
## S3 method for class 'ral'
tidy(
  x,
  fit_idx = 1,
  conf.int = FALSE,
  conf.level = 0.95,
  type = "HC1",
  uniform = FALSE,
  bootstraps = 999L,
  ...
)
```

Arguments

<code>x</code>	An object inheriting from class <code>ral</code> .
<code>fit_idx</code>	Integer index of the fit to report. Defaults to 1. Set to <code>NULL</code> for all fits.
<code>conf.int</code>	Logical. Include confidence intervals? Default <code>FALSE</code> .
<code>conf.level</code>	Confidence level. Default 0.95.
<code>type</code>	Character. HC type. Default <code>"HC1"</code> .
<code>uniform</code>	Logical. Uniform confidence bands? Default <code>FALSE</code> .
<code>bootstraps</code>	Integer. Bootstrap draws. Default 999.
<code>...</code>	Currently unused.

Value

A data.frame with columns `term`, `estimate`, `std.error`, `statistic`, `p.value`, and `fit_label`.

References

Chernozhukov V, Chetverikov D, Kato K (2013). "Gaussian approximations and multiplier bootstrap for maxima of sums of high-dimensional random vectors." *Annals of Statistics*, 41(6), 2786-2819.

tidy.ral_rep

*Tidy a RAL Rep Object***Description**

Tidy a RAL Rep Object

Usage

```
## S3 method for class 'ral_rep'
tidy(
  x,
  fit_idx = 1,
  aggregation = c("median", "mean", "spectral"),
  type = "HC1",
  conf.int = FALSE,
  conf.level = 0.95,
  uniform = FALSE,
  bootstraps = 999L,
  ...
)
```

Arguments

<code>x</code>	An object inheriting from class <code>ral_rep</code> .
<code>fit_idx</code>	Integer index of the fit. Defaults to 1. Set to NULL for all fits.
<code>aggregation</code>	Character string. Aggregation rule.
<code>type</code>	Character. HC type. Default "HC1".
<code>conf.int</code>	Logical. Include CIs? Default FALSE.
<code>conf.level</code>	Confidence level. Default 0.95.
<code>uniform</code>	Logical. Uniform CIs? Default FALSE.
<code>bootstraps</code>	Integer. Bootstrap draws. Default 999.
<code>...</code>	Currently unused.

Value

A data.frame with columns `term`, `estimate`, `std.error`, `statistic`, `p.value`, `fit_label`, and `aggregation`.

vcov.ral

Variance-Covariance Matrix for RAL Estimators

Description

Computes a heteroskedasticity-robust variance-covariance matrix.

Usage

```
## S3 method for class 'ral'
vcov(object, fit_idx = 1, type = "HC1", ...)
```

Arguments

object	An object inheriting from class <code>ral</code> .
fit_idx	Integer index of the fit. Defaults to 1.
type	Character. One of "HC1" (default), "HC0", or "HC3".
...	Currently unused.

Details

Let $\hat{\phi}_i$ denote the estimated influence function at observation i . Three variance estimators are available:

HC0:

$$V_{\text{HC0}} = \frac{1}{n^2} \sum_i \hat{\phi}_i \hat{\phi}_i'$$

HC1 (default):

$$V_{\text{HC1}} = V_{\text{HC0}} \times \frac{n}{n-p}$$

HC3:

$$V_{\text{HC3}} = \frac{1}{n^2} \sum_i \frac{\hat{\phi}_i \hat{\phi}_i'}{(1 - \hat{h}_{\theta,i})^2}$$

where $\hat{h}_{\theta,i}$ is the leverage; see [hatvalues.ral](#).

Cluster-robust inference. When `cluster_variable` is non-NULL and identifies fewer groups than observations, the observation-level influence functions are aggregated to cluster-level influence functions

$$\hat{\Phi}_g = \frac{G}{n} \sum_{i \in C_g} \hat{\phi}_i$$

and the variance is computed as

$$V_{\text{HC0}} = \frac{1}{G^2} \sum_{g=1}^G \hat{\Phi}_g \hat{\Phi}_g'.$$

Value

A $p \times p$ variance-covariance matrix.

See Also

[hatvalues.ral](#), [confint.ral](#)

vcov.ral_rep

Variance-Covariance Matrix for RAL Rep Objects

Description

Variance-Covariance Matrix for RAL Rep Objects

Usage

```
## S3 method for class 'ral_rep'
vcov(
  object,
  fit_idx = 1,
  aggregation = c("median", "mean", "spectral"),
  type = "HC1",
  ...
)
```

Arguments

object	An object inheriting from class <code>ral_rep</code> .
fit_idx	Integer index of the fit. Defaults to 1.
aggregation	Character string. Aggregation rule.
type	Character. HC type. Default "HC1".
...	Currently unused.

Value

A $p \times p$ variance-covariance matrix.

Index

- * **datasets**
 - AE98, [3](#)
 - * **ddml estimators**
 - ddml-intro, [17](#)
 - ddml_apo, [21](#)
 - ddml_ate, [24](#)
 - ddml_attgt, [29](#)
 - ddml_fpliv, [33](#)
 - ddml_late, [36](#)
 - ddml_pliv, [40](#)
 - ddml_plm, [43](#)
 - ddml_policy, [47](#)
 - * **ddml inference**
 - lincom, [60](#)
 - summary.ddml, [81](#)
 - * **ddml replication**
 - ddml_rep, [51](#)
 - ddml_replicate, [52](#)
 - * **ml_wrapper**
 - mdl_bigGlm, [65](#)
 - mdl_glm, [66](#)
 - mdl_glmnet, [66](#)
 - mdl_ranger, [67](#)
 - mdl_xgboost, [68](#)
 - ols, [70](#)
 - * **utilities**
 - crosspred, [10](#)
 - crossval, [13](#)
 - ddml, [15](#)
 - diagnostics, [53](#)
 - ensemble, [54](#)
 - ensemble_weights, [55](#)
 - shortstacking, [79](#)
- AE98, [3](#)
as.list.ddml, [4](#)
as.list.ddml_rep, [5](#)
as.list.ral, [5](#)
as.list.ral_rep, [6](#)
- coef.ral, [7](#), [20](#)
coef.ral_rep, [7](#)
confint.ral, [8](#), [15](#), [20](#), [72](#), [92](#)
confint.ral_rep, [9](#)
crosspred, [10](#), [14](#), [16](#), [53](#), [55](#), [56](#), [81](#)
crossval, [11](#), [12](#), [13](#), [16](#), [53](#), [55](#), [56](#), [81](#)
- ddml, [12](#), [14](#), [15](#), [53](#), [55](#), [56](#), [81](#)
ddml-intro, [17](#)
ddml_apo, [17](#), [21](#), [21](#), [28](#), [32](#), [35](#), [39](#), [43](#), [46](#), [50](#)
ddml_ate, [17](#), [20](#), [21](#), [24](#), [24](#), [32](#), [35](#), [39](#), [43](#),
[46](#), [50](#)
ddml_att, [17](#)
ddml_att(ddml_ate), [24](#)
ddml_attgt, [21](#), [24](#), [28](#), [29](#), [35](#), [39](#), [43](#), [46](#), [50](#),
[63](#)
ddml_fpliv, [17](#), [21](#), [24](#), [28](#), [32](#), [33](#), [39](#), [43](#), [46](#),
[50](#)
ddml_late, [17](#), [21](#), [24](#), [28](#), [32](#), [35](#), [36](#), [43](#), [46](#),
[50](#)
ddml_pliv, [17](#), [21](#), [24](#), [28](#), [32](#), [35](#), [39](#), [40](#), [46](#),
[50](#)
ddml_plm, [17](#), [18](#), [20](#), [21](#), [23](#), [24](#), [27](#), [28](#), [31](#),
[32](#), [35](#), [38](#), [39](#), [42](#), [43](#), [43](#), [45](#), [49](#), [50](#)
ddml_policy, [17](#), [21](#), [24](#), [28](#), [32](#), [35](#), [39](#), [43](#),
[46](#), [47](#)
ddml_rep, [51](#), [52](#)
ddml_rep(), [52](#)
ddml_replicate, [52](#), [52](#)
ddml_replicate(), [52](#)
diagnostics, [12](#), [14](#), [16](#), [20](#), [53](#), [55](#), [56](#), [81](#)
- ensemble, [12](#), [14](#), [16](#), [53](#), [54](#), [56](#), [81](#)
ensemble_weights, [12](#), [14](#), [16](#), [53](#), [55](#), [55](#), [81](#)
- glance.ddml, [20](#), [57](#)
glance.ddml_rep, [58](#)
glance.ral, [58](#)
glance.ral_rep, [59](#)
glmnet::bigGlm(), [65](#)

`glmnet::cv.glmnet()`, 66, 67
`glmnet::glmnet()`, 66, 67
`hatvalues.ral`, 19, 20, 59, 61, 62, 78, 91, 92
`lincom`, 6, 60, 60, 63, 64, 78, 82
`lincom_weights_did`, 62, 63
`mdl_bigGlm`, 65, 66–70
`mdl_glm`, 65, 66, 67–70
`mdl_glmnet`, 65, 66, 66, 68–70
`mdl_ranger`, 65, 66, 67, 67, 69, 70
`mdl_xgboost`, 65–67, 68, 68, 70
`nobs.ral`, 20, 69
`ols`, 65–69, 70
`plot.default`, 72
`plot.ral`, 71
`plot.ral_rep(plot.ral)`, 71
`predict.ensemble`, 72
`predict.glm`, 74
`predict.glmnet`, 74
`predict.mdl_bigGlm`, 73
`predict.mdl_glm`, 74
`predict.mdl_glmnet`, 74
`predict.mdl_ranger`, 75
`predict.mdl_xgboost`, 75
`predict.ols`, 76
`predict.ranger`, 75
`predict.xgb.Booster`, 75
`print.ddml_diagnostics`, 76
`print.ddml_rep(ddml_rep)`, 51
`print.lincom(lincom)`, 60
`print.lincom_rep(lincom)`, 60
`print.summary.ddml(summary.ddml)`, 81
`print.summary.ddml_rep(summary.ddml_rep)`, 82
`print.summary.ral(summary.ral)`, 83
`print.summary.ral_rep(summary.ral_rep)`, 84
`ral`, 6, 61, 77
`ral_rep`, 78
`ranger::ranger()`, 67, 68
`shortstacking`, 12, 14, 16, 53, 55, 56, 79
`stats::glm()`, 66
`summary.ddml`, 15, 20, 62, 81
`summary.ddml_rep`, 82, 88
`summary.ral`, 72, 83
`summary.ral_rep`, 83, 84
`tidy.ddml`, 15, 20, 85
`tidy.ddml_diagnostics`, 87
`tidy.ddml_rep`, 87
`tidy.ral`, 89
`tidy.ral_rep`, 90
`vcov.ral`, 9, 15, 19, 20, 62, 81, 82, 91
`vcov.ral_rep`, 92
`xgboost::xgboost()`, 68, 69