Package ‘DoubleML’

August 19, 2022

Type Package

Title Double Machine Learning in R

Version 0.5.1

Description Implementation of the double/debiased machine learning framework of Chernozhukov et al. (2018) <doi:10.1111/ectj.12097> for partially linear regression models, partially linear instrumental variable regression models, interactive regression models and interactive instrumental variable regression models. 'DoubleML' allows estimation of the nuisance parts in these models by machine learning methods and computation of the Neyman orthogonal score functions. 'DoubleML' is built on top of 'mlr3' and the 'mlr3' ecosystem. The object-oriented implementation of 'DoubleML' based on the 'R6' package is very flexible.

License MIT + file LICENSE

URL https://docs.doubleml.org/stable/index.html,
https://github.com/DoubleML/doubleml-for-r/

BugReports https://github.com/DoubleML/doubleml-for-r/issues

Encoding UTF-8

Depends R (>= 3.5.0)

Imports R6 (>= 2.4.1), data.table (>= 1.12.8), stats, checkmate, mlr3 (>= 0.5.0), mlr3tuning (>= 0.3.0), mvtorm, utils, clusterGeneration, readstata13, mldr3learners (>= 0.3.0), mldr3misc

RoxygenNote 7.2.1

Suggests knitr, rmarkdown, testthat, covr, patrick (>= 0.1.0), paradox (>= 0.4.0), dplyr, glmnet, lgr, ranger, sandwich, AER, rpart, bbotk, mlr3pipelines

VignetteBuilder knitr

Collate 'double_ml.R' 'double_ml_data.R' 'double_ml_iivm.R'
 'double_ml_irm.R' 'double_ml_pliv.R' 'double_ml_plr.R'
 'helper.R' 'datasets.R' 'zzz.R'

NeedsCompilation no
Abstract class `DoubleML`

Description

Abstract base class that can’t be initialized.

Format

`R6::R6Class` object.
Active bindings

all_coef (matrix())
Estimates of the causal parameter(s) for the n_rep different sample splits after calling fit().

all_dml1_coef (array())
Estimates of the causal parameter(s) for the n_rep different sample splits after calling fit() with dml_procedure = "dml1".

all_se (matrix())
Standard errors of the causal parameter(s) for the n_rep different sample splits after calling fit().

apply_cross_fitting (logical(1))
Indicates whether cross-fitting should be applied. Default is TRUE.

boot_coef (matrix())
Bootstrapped coefficients for the causal parameter(s) after calling fit() and bootstrap().

boot_t_stat (matrix())
Bootstrapped t-statistics for the causal parameter(s) after calling fit() and bootstrap().

coeff (numeric())
Estimates for the causal parameter(s) after calling fit().

data (data.table)
Data object.

dml_procedure (character(1))
A character() ("dml1" or "dml2") specifying the double machine learning algorithm. Default is "dml2".

draw_sample_splitting (logical(1))
Indicates whether the sample splitting should be drawn during initialization of the object. Default is TRUE.

learner (named list())
The machine learners for the nuisance functions.

n_folds (integer(1))
Number of folds. Default is 5.

n_rep (integer(1))
Number of repetitions for the sample splitting. Default is 1.

params (named list())
The hyperparameters of the learners.

psi (array())
Value of the score function \( \psi(W; \theta, \eta) = \psi_a(W; \eta)\theta + \psi_b(W; \eta) \) after calling fit().

psi_a (array())
Value of the score function component \( \psi_a(W; \eta) \) after calling fit().

psi_b (array())
Value of the score function component \( \psi_b(W; \eta) \) after calling fit().
predictions (array())
Predictions of the nuisance models after calling fit(store_predictions=TRUE).
pval (numeric())
p-values for the causal parameter(s) after calling fit().
score (character(1), function())
   A character(1) or function() specifying the score function.
se (numeric())
   Standard errors for the causal parameter(s) after calling fit().
smpls (list())
   The partition used for cross-fitting.
smpls_cluster (list())
   The partition of clusters used for cross-fitting.
t_stat (numeric())
   t-statistics for the causal parameter(s) after calling fit().
tuning_res (named list())
   Results from hyperparameter tuning.

Methods

Public methods:
   • DoubleML$new()
   • DoubleML$print()
   • DoubleML$fit()
   • DoubleML$bootstrap()
   • DoubleML$split_samples()
   • DoubleML$set_sample_splitting()
   • DoubleML$tune()
   • DoubleML$summary()
   • DoubleML$confint()
   • DoubleML$learner_names()
   • DoubleML$params_names()
   • DoubleML$set_ml_nuisance_params()
   • DoubleML$p_adjust()
   • DoubleML$get_params()
   • DoubleML$clone()

Method new(): DoubleML is an abstract class that can’t be initialized.
   Usage:
   DoubleML$new()

Method print(): Print DoubleML objects.
   Usage:
   DoubleML$print()

Method fit(): Estimate DoubleML models.
   Usage:
   DoubleML$fit(store_predictions = FALSE)
   Arguments:
**store_predictions** (logical(1))

Indicates whether the predictions for the nuisance functions should be stored in field predictions. Default is `FALSE`.

**Returns:** `self`

**Method** `bootstrap()`: Multiplier bootstrap for DoubleML models.

**Usage:**

DoubleML$bootstrap(method = "normal", n_rep_boot = 500)

**Arguments:**

- `method` (character(1))
  A character(1)("Bayes", "normal" or "wild") specifying the multiplier bootstrap method.
- `n_rep_boot` (integer(1))
  The number of bootstrap replications.

**Returns:** `self`

**Method** `split_samples()`: Draw sample splitting for DoubleML models.

The samples are drawn according to the attributes `n_folds`, `n_rep` and `apply_cross_fitting`.

**Usage:**

DoubleML$split_samples()

**Returns:** `self`

**Method** `set_sample_splitting()`: Set the sample splitting for DoubleML models.

The attributes `n_folds` and `n_rep` are derived from the provided partition.

**Usage:**

DoubleML$set_sample_splitting(smpls)

**Arguments:**

- `smpls` (list())
  A nested list(). The outer lists needs to provide an entry per repeated sample splitting (length of the list is set as `n_rep`). The inner list is a named list() with names `train_ids` and `test_ids`. The entries in `train_ids` and `test_ids` must be partitions per fold (length of `train_ids` and `test_ids` is set as `n_folds`).

**Returns:** `self`

**Examples:**

library(DoubleML)
library(mlr3)
set.seed(2)
obj_dml_data = make_plr_CCDDHNR2018(n_obs=10)
dml_plr_obj = DoubleMLPLR$new(obj_dml_data,
  lrn("regr.rpart"), lrn("regr.rpart"))

# simple sample splitting with two folds and without cross-fitting
smpls = list(list(train_ids = list(c(1, 2, 3, 4, 5)),
    test_ids = list(c(6, 7, 8, 9, 10))))
dml_plr_obj$set_sample_splitting(smpls)
# sample splitting with two folds and cross-fitting but no repeated cross-fitting
smpls = list(list(train_ids = list(c(1, 2, 3, 4, 5), c(6, 7, 8, 9, 10)),
             test_ids = list(c(6, 7, 8, 9, 10), c(1, 2, 3, 4, 5))))
dml_plr_obj$set_sample_splitting(smpls)

# sample splitting with two folds and repeated cross-fitting with n_rep = 2
smpls = list(list(train_ids = list(c(1, 2, 3, 4, 5), c(6, 7, 8, 9, 10)),
             test_ids = list(c(6, 7, 8, 9, 10), c(1, 2, 3, 4, 5)),
             train_ids = list(c(1, 3, 5, 7, 9), c(2, 4, 6, 8, 10)),
             test_ids = list(c(2, 4, 6, 8, 10), c(1, 3, 5, 7, 9))))
dml_plr_obj$set_sample_splitting(smpls)

**Method** `tune()`: Hyperparameter-tuning for DoubleML models.

The hyperparameter-tuning is performed using the tuning methods provided in the `mlr3tuning` package. For more information on tuning in `mlr3`, we refer to the section on parameter tuning in the [mlr3 book](https://mlr3.tidyverse.org/).

**Usage:**

```r
DoubleML$tune(
  param_set,
  tune_settings = list(n_folds_tune = 5, rsmp_tune = mlr3::rsmp("cv", folds = 5), measure = NULL,
                        terminator = mlr3tuning::trm("evals", n_evals = 20), algorithm =
                        mlr3tuning::tnr("grid_search"), resolution = 5),
  tune_on_folds = FALSE
)
```

**Arguments:**

- `param_set` (named list()): A named list with a parameter grid for each nuisance model/learner (see method `learner_names()`). The parameter grid must be an object of class `ParamSet`.
- `tune_settings` (named list()): A named list() with arguments passed to the hyperparameter-tuning with `mlr3tuning` to set up `TuningInstance` objects. `tune_settings` has entries
  - `algorithm` (Tuner or character(1)): A `Tuner` object (recommended) or key passed to the respective dictionary to specify the tuning algorithm used in `tnr()`. `algorithm` is passed as an argument to `tnr()`. If `algorithm` is not specified by the users, default is set to "grid_search". If set to "grid_search", then additional argument "resolution" is required.
  - `rsmp_tune` (Resampling or character(1)): A `Resampling` object (recommended) or option passed to `rsmp()` to initialize a `Resampling` for parameter tuning in `mlr3`. If not specified by the user, default is set to "cv" (cross-validation).
  - `n_folds_tune` (integer(1), optional): If `rsmp_tune = "cv"`, number of folds used for cross-validation. If not specified by the user, default is set to 5.
DoubleML

- **measure**: (NULL, named list(), optional)
  Named list containing the measures used for parameter tuning. Entries in list must either be Measure objects or keys to be passed to passed to msr(). The names of the entries must match the learner names (see method learner_names()). If set to NULL, default measures are used, i.e., "regr.mse" for continuous outcome variables and "classif.ce" for binary outcomes.

- **resolution**: (character(1))
  The key passed to the respective dictionary to specify the tuning algorithm used in tnr(). resolution is passed as an argument to tnr().

- **tune_on_folds**: (logical(1))
  Indicates whether the tuning should be done fold-specific or globally. Default is FALSE.

**Returns**: self

**Method** `summary()`: Summary for DoubleML models after calling `fit()`.

**Usage**:
```
DoubleML$summary(digits = max(3L, getOption("digits") - 3L))
```

**Arguments**:
- **digits**: (integer(1))
  The number of significant digits to use when printing.

**Method** `confint()`: Confidence intervals for DoubleML models.

**Usage**:
```
DoubleML$confint(parm, joint = FALSE, level = 0.95)
```

**Arguments**:
- **parm**: (numeric() or character())
  A specification of which parameters are to be given confidence intervals among the variables for which inference was done, either a vector of numbers or a vector of names. If missing, all parameters are considered (default).
- **joint**: (logical(1))
  Indicates whether joint confidence intervals are computed. Default is FALSE.
- **level**: (numeric(1))
  The confidence level. Default is 0.95.

**Returns**: A matrix() with the confidence interval(s).

**Method** `learner_names()`: Returns the names of the learners.

**Usage**:
```
DoubleML$learner_names()
```

**Returns**: character() with names of learners.

**Method** `params_names()`: Returns the names of the nuisance models with hyperparameters.

**Usage**:
```
DoubleML$params_names()
```

**Returns**: character() with names of nuisance models with hyperparameters.
Method `set_ml_nuisance_params()`: Set hyperparameters for the nuisance models of DoubleML models.
Note that in the current implementation, either all parameters have to be set globally or all parameters have to be provided fold-specific.

Usage:
```r
DoubleML$set_ml_nuisance_params(
  learner = NULL,
  treat_var = NULL,
  params,
  set_fold_specific = FALSE
)
```

Arguments:
- `learner` (character(1))
  The nuisance model/learner (see method `params_names`).
- `treat_var` (character(1))
  The treatment variable (hyperparameters can be set treatment-variable specific).
- `params` (named list())
  A named list() with estimator parameters. Parameters are used for all folds by default. Alternatively, parameters can be passed in a fold-specific way if option `set_fold_specific` is TRUE. In this case, the outer list needs to be of length `n_rep` and the inner list of length `n_folds`.
- `set_fold_specific` (logical(1))
  Indicates if the parameters passed in `params` should be passed in fold-specific way. Default is FALSE. If TRUE, the outer list needs to be of length `n_rep` and the inner list of length `n_folds`. Note that in the current implementation, either all parameters have to be set globally or all parameters have to be provided fold-specific.

Returns: self

Method `p_adjust()`: Multiple testing adjustment for DoubleML models.

Usage:
```r
DoubleML$p_adjust(method = "romano-wolf", return_matrix = TRUE)
```

Arguments:
- `method` (character(1))
  A character(1) ("romano-wolf", "bonferroni", "holm", etc) specifying the adjustment method. In addition to "romano-wolf", all methods implemented in `p.adjust()` can be applied. Default is "romano-wolf".
- `return_matrix` (logical(1))
  Indicates if the output is returned as a matrix with corresponding coefficient names.

Returns: numeric() with adjusted p-values. If `return_matrix` = TRUE, a matrix() with adjusted p-values.

Method `get_params()`: Get hyperparameters for the nuisance model of DoubleML models.

Usage:
```r
DoubleML$get_params(learner)
```

Arguments:
learner (character(1))
The nuisance model/learner (see method params_names())

Returns: named list() with parameters for the nuisance model/learner.

Method clone(): The objects of this class are cloneable with this method.

Usage:
DoubleML$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.

See Also
Other DoubleML: DoubleMLIIVM, DoubleMLIRM, DoubleMLPLIV, DoubleMLPLR

Examples

## Method `DoubleML$set_sample_splitting`

library(DoubleML)
library(mlr3)
set.seed(2)
obj_dml_data = make_plr_CCDDHNR2018(n_obs=10)
dml_plr_obj = DoubleMLPLR$new(obj_dml_data,
  lrn("regr.rpart"), lrn("regr.rpart"))

# simple sample splitting with two folds and without cross-fitting
smpls = list(list(train_ids = list(c(1, 2, 3, 4, 5)),
               test_ids = list(c(6, 7, 8, 9, 10))))
dml_plr_obj$set_sample_splitting(smpls)

# sample splitting with two folds and cross-fitting but no repeated cross-fitting
smpls = list(list(train_ids = list(c(1, 2, 3, 4, 5), c(6, 7, 8, 9, 10)),
               test_ids = list(c(6, 7, 8, 9, 10), c(1, 2, 3, 4, 5))))
dml_plr_obj$set_sample_splitting(smpls)

# sample splitting with two folds and repeated cross-fitting with n_rep = 2
smpls = list(list(train_ids = list(c(1, 2, 3, 4, 5), c(6, 7, 8, 9, 10)),
               test_ids = list(c(6, 7, 8, 9, 10), c(1, 2, 3, 4, 5)),
               list(train_ids = list(c(1, 3, 5, 7, 9), c(2, 4, 6, 8, 10)),
               test_ids = list(c(2, 4, 6, 8, 10), c(1, 3, 5, 7, 9))))
dml_plr_obj$set_sample_splitting(smpls)
DoubleMLClusterData

Double machine learning data-backend for data with cluster variables.

Description

Double machine learning data-backend for data with cluster variables.
DoubleMLClusterData objects can be initialized from a data.table. Alternatively DoubleML provides functions to initialize from a collection of matrix objects or a data.frame. The following functions can be used to create a new instance of DoubleMLClusterData.

- DoubleMLClusterData$new() for initialization from a data.table.
- double_ml_data_from_matrix() for initialization from matrix objects,
- double_ml_data_from_data_frame() for initialization from a data.frame.

Super class

DoubleML::DoubleMLData -> DoubleMLClusterData

Active bindings

cluster_cols (character())
The cluster variable(s).
x_cols (NULL, character())
The covariates. If NULL, all variables (columns of data) which are neither specified as outcome variable y_col, nor as treatment variables d_cols, nor as instrumental variables z_cols, nor as cluster variables cluster_cols are used as covariates. Default is NULL.
n_cluster_vars (integer(1))
The number of cluster variables.

Methods

Public methods:
- DoubleMLClusterData$new()
- DoubleMLClusterData$print()
- DoubleMLClusterData$set_data_model()
- DoubleMLClusterData$clone()

Method new(): Creates a new instance of this R6 class.

Usage:
DoubleMLClusterData$new(
data = NULL,
  x_cols = NULL,
  y_col = NULL,
  d_cols = NULL,
  z_cols = NULL,
  cluster_cols = NULL,
DoubleMLClusterData

z_cols = NULL,
use_other_treat_as_covariate = TRUE
)

Arguments:
data (data.table, data.frame())
  Data object.
x_cols (NULL, character())
  The covariates. If NULL, all variables (columns of data) which are neither specified as
  outcome variable y_col, nor as treatment variables d_cols, nor as instrumental variables
  z_cols are used as covariates. Default is NULL.
y_col (character())
  The outcome variable.
d_cols (character())
  The treatment variable(s).
cluster_cols (character())
  The cluster variable(s).
z_cols (NULL, character())
  The instrumental variables. Default is NULL.
use_other_treat_as_covariate (logical(1))
  Indicates whether in the multiple-treatment case the other treatment variables should be
  added as covariates. Default is TRUE.

Method print(): Print DoubleMLClusterData objects.
Usage:
DoubleMLClusterData$print()

Method set_data_model(): Setter function for data_model. The function implements the
causal model as specified by the user via y_col, d_cols, x_cols, z_cols and cluster_cols and
assigns the role for the treatment variables in the multiple-treatment case.
Usage:
DoubleMLClusterData$set_data_model(treatment_var)
Arguments:
treatment_var (character())
  Active treatment variable that will be set to treat_col.

Method clone(): The objects of this class are cloneable with this method.
Usage:
DoubleMLClusterData$clone(deep = FALSE)
Arguments:
deep Whether to make a deep clone.

Examples
library(DoubleML)
dt = make_pliv_multiway_cluster_CKMS2021(return_type = "data.table")
DoubleMLData

Description

Double machine learning data-backend.

DoubleMLData objects can be initialized from a data.table. Alternatively DoubleML provides functions to initialize from a collection of matrix objects or a data.frame. The following functions can be used to create a new instance of DoubleMLData.

- DoubleMLData$new() for initialization from a data.table.
- double_ml_data_from_matrix() for initialization from matrix objects,
- double_ml_data_from_data_frame() for initialization from a data.frame.

Active bindings

- all_variables (character())
  All variables available in the dataset.
- d_cols (character())
  The treatment variable(s).
- data (data.table)
  Data object.
- data_model (data.table)
  Internal data object that implements the causal model as specified by the user via y_col, d_cols, x_cols and z_cols.
- n_instr (NULL, integer(1))
  The number of instruments.
- n_obs (integer(1))
  The number of observations.
- n_treat (integer(1))
  The number of treatment variables.
- other_treat_cols (NULL, character())
  If use_other_treat_as_covariate is TRUE, other_treat_cols are the treatment variables that are not "active" in the multiple-treatment case. These variables then are internally added to the covariates x_cols during the fitting stage. If use_other_treat_as_covariate is FALSE, other_treat_cols is NULL.
- treat_col (character(1))
  "Active" treatment variable in the multiple-treatment case.
DoubleMLData

use_other_treat_as_covariate (logical(1))
Indicates whether in the multiple-treatment case the other treatment variables should be added as covariates. Default is TRUE.

x_cols (NULL, character())
The covariates. If NULL, all variables (columns of data) which are neither specified as outcome variable y_col, nor as treatment variables d_cols, nor as instrumental variables z_cols are used as covariates. Default is NULL.

y_col (character(1))
The outcome variable.

z_cols (NULL, character())
The instrumental variables. Default is NULL.

Methods

Public methods:
• DoubleMLData$new()
• DoubleMLData$print()
• DoubleMLData$set_data_model()
• DoubleMLData$clone()

Method new(): Creates a new instance of this R6 class.

Usage:
DoubleMLData$new(
  data = NULL,
  x_cols = NULL,
  y_col = NULL,
  d_cols = NULL,
  z_cols = NULL,
  use_other_treat_as_covariate = TRUE
)

Arguments:
data (data.table, data.frame())
Data object.
x_cols (NULL, character())
The covariates. If NULL, all variables (columns of data) which are neither specified as outcome variable y_col, nor as treatment variables d_cols, nor as instrumental variables z_cols are used as covariates. Default is NULL.

y_col (character(1))
The outcome variable.

d_cols (character())
The treatment variable(s).

z_cols (NULL, character())
The instrumental variables. Default is NULL.

use_other_treat_as_covariate (logical(1))
Indicates whether in the multiple-treatment case the other treatment variables should be added as covariates. Default is TRUE.
Method print(): Print DoubleMLData objects.

Usage:
DoubleMLData$print()

Method set_data_model(): Setter function for data_model. The function implements the causal model as specified by the user via y_col, d_cols, x_cols and z_cols and assigns the role for the treatment variables in the multiple-treatment case.

Usage:
DoubleMLData$set_data_model(treatment_var)

Arguments:
treatment_var (character())
Active treatment variable that will be set to treat_col.

Method clone(): The objects of this class are cloneable with this method.

Usage:
DoubleMLData$clone(deep = FALSE)

Arguments:
dee Whether to make a deep clone.

Examples

library(DoubleML)
df = make_plr_CCDDHNR2018(return_type = "data.table")
obj_dml_data = DoubleMLData$new(df,
y_col = "y",
d_cols = "d")

DoubleMLIIVM

Double machine learning for interactive IV regression models

Description

Double machine learning for interactive IV regression models.

Format

R6::R6Class object inheriting from DoubleML.

Details

Interactive IV regression (IIVM) models take the form

\[ Y = f_0(D, X) + \zeta, \]
\[ Z = m_0(X) + V, \]

with \( E[\zeta|X,Z] = 0 \) and \( E[V|X] = 0 \). \( Y \) is the outcome variable, \( D \in \{0,1\} \) is the binary treatment variable and \( Z \in \{0,1\} \) is a binary instrumental variable. Consider the functions \( g_0, r_0 \)
and $m_0$ where $g_0$ maps the support of $(Z, X)$ to $R$ and $r_0$ and $m_0$, respectively, map the support of $(Z, X)$ and $X$ to $(\epsilon, 1 - \epsilon)$ for some $\epsilon \in (1/2)$, such that

$$Y = g_0(Z, X) + \nu,$$

$$D = r_0(Z, X) + U,$$

$$Z = m_0(X) + V,$$

with $E[\nu | Z, X] = 0$, $E[U | Z, X] = 0$ and $E[V | X] = 0$. The target parameter of interest in this model is the local average treatment effect (LATE),

$$\theta_0 = \frac{E[g_0(1, X)] - E[g_0(0, X)]}{E[r_0(1, X)] - E[r_0(0, X)]}.$$

**Super class**

`DoubleML::DoubleML` -> `DoubleMLIIVM`

**Active bindings**

- `subgroups` (named `list(2)`)

  Named `list(2)` with options to adapt to cases with and without the subgroups of always-takers and never-takes. The entry `always_takers` (logical(1)) specifies whether there are always takers in the sample. The entry `never_takers` (logical(1)) specifies whether there are never takers in the sample.

- `trimming_rule` (character(1))

  A character(1) specifying the trimming approach.

- `trimming_threshold` (numeric(1))

  The threshold used for trimming.

**Methods**

**Public methods:**

- `DoubleMLIIVM$new()`
- `DoubleMLIIVM$clone()`

**Method new():** Creates a new instance of this R6 class.

**Usage:**

```r
DoubleMLIIVM$new(
  data,
  ml_g,
  ml_m,
  ml_r,
  n_folds = 5,
  n_rep = 1,
  score = "LATE",
  subgroups = list(always_takers = TRUE, never_takers = TRUE),
  dml_procedure = "dml2",
  trimming_rule = "truncate",
  trimming_threshold = 1e-12,
  draw_sample_splitting = TRUE,
)```

DoubleML1IVM

```r
apply_cross_fitting = TRUE
```

### Arguments:

- **data** *(DoubleMLData)*
  - The `DoubleMLData` object providing the data and specifying the variables of the causal model.

- **ml_g** *(LearnerRegr, LearnerClassif, Learner, character(1))*
  - A learner of the class `LearnerRegr`, which is available from `mlr3` or its extension packages `mlr3learners` or `mlr3extralearners`. For binary treatment outcomes, an object of the class `LearnerClassif` can be passed, for example `lrn("classif.cv_glmnet", s = "lambda.min")`. Alternatively, a `Learner` object with public field `task_type = "regr"` or `task_type = "classif"` can be passed, respectively, for example of class `GraphLearner`.
  - `ml_g` refers to the nuisance function $g_0(Z, X) = E[Y|X, Z]$.

- **ml_m** *(LearnerClassif, Learner, character(1))*
  - A learner of the class `LearnerClassif`, which is available from `mlr3` or its extension packages `mlr3learners` or `mlr3extralearners`. Alternatively, a `Learner` object with public field `task_type = "classif"` can be passed, for example of class `GraphLearner`. The learner can possibly be passed with specified parameters, for example `lrn("classif.cv_glmnet", s = "lambda.min")`.
  - `ml_m` refers to the nuisance function $m_0(X) = E[Z|X]$.

- **ml_r** *(LearnerClassif, Learner, character(1))*
  - A learner of the class `LearnerClassif`, which is available from `mlr3` or its extension packages `mlr3learners` or `mlr3extralearners`. Alternatively, a `Learner` object with public field `task_type = "classif"` can be passed, for example of class `GraphLearner`. The learner can possibly be passed with specified parameters, for example `lrn("classif.cv_glmnet", s = "lambda.min")`.
  - `ml_r` refers to the nuisance function $r_0(Z, X) = E[D|X, Z]$.

- **n_folds** *(integer(1))*
  - Number of folds. Default is 5.

- **n_rep** *(integer(1))*
  - Number of repetitions for the sample splitting. Default is 1.

- **score** *(character(1), function())*
  - A character(1) ("LATE" is the only choice) specifying the score function. If a function() is provided, it must be of the form `function(y, z, d, g0_hat, g1_hat, m_hat, r0_hat, r1_hat, smpls)` and the returned output must be a named list() with elements `psi_a` and `psi_b`. Default is "LATE".

- **subgroups** *(named list(2))*
  - Named list(2) with options to adapt to cases with and without the subgroups of always-takers and never-takes. The entry `always_takers(logical(1))` specifies whether there are always takers in the sample. The entry `never_takers(logical(1))` specifies whether there are never takers in the sample. Default is `list(always_takers = TRUE, never_takers = TRUE)`.

- **dml_procedure** *(character(1))*
  - A character(1) ("dml1" or "dml2") specifying the double machine learning algorithm. Default is "dml2".

- **trimming_rule** *(character(1))*

```
A character(1) ("truncate" is the only choice) specifying the trimming approach. Default is "truncate".

trimming_threshold (numeric(1))
   The threshold used for timing. Default is 1e-12.

draw_sample_splitting (logical(1))
   Indicates whether the sample splitting should be drawn during initialization of the object. Default is TRUE.

apply_cross_fitting (logical(1))
   Indicates whether cross-fitting should be applied. Default is TRUE.

Method clone(): The objects of this class are cloneable with this method.

Usage:
DoubleMLIIVM$clone(deep = FALSE)

Arguments:
   deep  Whether to make a deep clone.

See Also

Other DoubleML: DoubleMLIRM, DoubleMLPLIV, DoubleMLPLR, DoubleML

Examples

library(DoubleML)
library(mlr3)
library(mlr3learners)
library(data.table)
set.seed(2)
ml_g = lrn("regr.ranger",
   num.trees = 100, mtry = 20,
   min.node.size = 2, max.depth = 5)
ml_m = lrn("classif.ranger",
   num.trees = 100, mtry = 20,
   min.node.size = 2, max.depth = 5)
ml_r = ml_m$clone()
obj_dml_data = make_iivm_data(
   theta = 0.5, n_obs = 1000,
   alpha_x = 1, dim_x = 20)
dml_iivm_obj = DoubleMLIIVM$new(obj_dml_data, ml_g, ml_m, ml_r)
dml_iivm_obj$fit()
dml_iivm_obj$summary()

## Not run:
library(DoubleML)
library(mlr3)
library(mlr3learners)
library(data.table)
set.seed(2)
ml_g = lrn("regr.rpart")
ml_m = lrn("classif.rpart")
ml_r = ml_m$clone()
obj_dml_data = make_iivm_data(
    theta = 0.5, n_obs = 1000,
    alpha_x = 1, dim_x = 20)
dml_iivm_obj = DoubleMLIIVM$new(obj_dml_data, ml_g, ml_m, ml_r)
param_grid = list(
    "ml_g" = paradox::ParamSet$new(list(
        paradox::ParamDbl$new("cp", lower = 0.01, upper = 0.02),
        paradox::ParamInt$new("minsplit", lower = 1, upper = 2))),
    "ml_m" = paradox::ParamSet$new(list(
        paradox::ParamDbl$new("cp", lower = 0.01, upper = 0.02),
        paradox::ParamInt$new("minsplit", lower = 1, upper = 2))),
    "ml_r" = paradox::ParamSet$new(list(
        paradox::ParamDbl$new("cp", lower = 0.01, upper = 0.02),
        paradox::ParamInt$new("minsplit", lower = 1, upper = 2))))
# minimum requirements for tune_settings
tune_settings = list(
    terminator = mlr3tuning::trm("evals", n_evals = 5),
    algorithm = mlr3tuning::tnr("grid_search", resolution = 5))
dml_iivm_obj$tune(param_set = param_grid, tune_settings = tune_settings)
dml_iivm_obj$fit()
dml_iivm_obj$summary()

DoubleMLIRM

Double machine learning for interactive regression models

Description

Double machine learning for interactive regression models.

Format

R6::R6Class object inheriting from DoubleML.

Details

Interactive regression (IRM) models take the form
\[ Y = g_0(D, X) + U, \]
\[ D = m_0(X) + V, \]
with \( E[U|X,D] = 0 \) and \( E[V|X] = 0 \). \( Y \) is the outcome variable and \( D \in \{0,1\} \) is the binary treatment variable. We consider estimation of the average treatment effects when treatment effects are fully heterogeneous. Target parameters of interest in this model are the average treatment effect (ATE).
\[ \theta_0 = E[ g_0(1, X) - g_0(0, X) ] \]

and the average treatment effect on the treated (ATTE),

\[ \theta_0 = E[ g_0(1, X) - g_0(0, X) | D = 1 ] . \]

Super class

`DoubleML::DoubleML` -> `DoubleMLIRM`

Active bindings

- `trimming_rule (character(1))`
  - A character(1) specifying the trimming approach.
- `trimming_threshold (numeric(1))`
  - The threshold used for trimming.

Methods

Public methods:

- `DoubleMLIRM$new()`
- `DoubleMLIRM$clone()`

Method `new()`: Creates a new instance of this R6 class.

Usage:

```r
DoubleMLIRM$new(
  data, 
  ml_g, 
  ml_m, 
  n_folds = 5, 
  n_rep = 1,
  score = "ATE", 
  trimming_rule = "truncate", 
  trimming_threshold = 1e-12, 
  dml_procedure = "dml2", 
  draw_sample_splitting = TRUE, 
  apply_cross_fitting = TRUE 
)
```

Arguments:

- `data (DoubleMLData)`
  - The DoubleMLData object providing the data and specifying the variables of the causal model.
- `ml_g (LearnerRegr, LearnerClassif, Learner, character(1))`
  - A learner of the class `LearnerRegr`, which is available from `mlr3` or its extension packages `mlr3learners` or `mlr3extralearners`. For binary treatment outcomes, an object of the class `LearnerClassif` can be passed, for example `lrn("classif.cv_glmnet", s = "lambda.min")`. Alternatively, a `Learner` object with public field `task_type = "regr"` or `task_type = "classif"` can be passed, respectively, for example of class `GraphLearner`.
- `ml_g` refers to the nuisance function \( g_0(X) = E[Y|X, D] \).
ml_m (LearnerClassif, Learner, character(1))
A learner of the class LearnerClassif, which is available from mlr3 or its extension packages mlr3learners or mlr3extralearners. Alternatively, a Learner object with public field task_type = "classif" can be passed, for example of class GraphLearner. The learner can possibly be passed with specified parameters, for example lrn("classif.cv_glmnet", s = "lambda.min").
ml_m refers to the nuisance function \( m_0(X) = E[D|X] \).

n_folds (integer(1))
Number of folds. Default is 5.

n_rep (integer(1))
Number of repetitions for the sample splitting. Default is 1.

score (character(1), function())
A character(1) ("ATE" or ATTE) or a function() specifying the score function. If a function() is provided, it must be of the form function(y, d, g0_hat, g1_hat, m_hat, smpls) and the returned output must be a named list() with elements psi_a and psi_b. Default is "ATE".

trimming_rule (character(1))
A character(1) ("truncate" is the only choice) specifying the trimming approach. Default is "truncate".

trimming_threshold (numeric(1))
The threshold used for trimming. Default is 1e-12.

dml_procedure (character(1))
A character(1) ("dml1" or "dml2") specifying the double machine learning algorithm. Default is "dml2".

draw_sample_splitting (logical(1))
Indicates whether the sample splitting should be drawn during initialization of the object. Default is TRUE.

apply_cross_fitting (logical(1))
Indicates whether cross-fitting should be applied. Default is TRUE.

Method clone(): The objects of this class are cloneable with this method.
Usage:
DoubleMLIRM$clone(deep = FALSE)
Arguments:
deep Whether to make a deep clone.

See Also
Other DoubleML: DoubleMLIVM, DoubleMLPLIV, DoubleMLPLR, DoubleML

Examples

library(DoubleML)
library(mlr3)
library(mlr3learners)
library(data.table)
Double machine learning for partially linear IV regression models

Description

Double machine learning for partially linear IV regression models.
Format

**R6::R6Class** object inheriting from **DoubleML**.

Details

Partially linear IV regression (PLIV) models take the form

\[ Y - D \theta_0 = g_0(X) + \zeta, \]
\[ Z = m_0(X) + V, \]

with \( E[\zeta | Z, X] = 0 \) and \( E[V | X] = 0 \). \( Y \) is the outcome variable, \( D \) is the policy variable of interest and \( Z \) denotes one or multiple instrumental variables. The high-dimensional vector \( X = (X_1, \ldots, X_p) \) consists of other confounding covariates, and \( \zeta \) and \( V \) are stochastic errors.

Super class

**DoubleML::DoubleML** -> **DoubleMLPLIV**

Active bindings

- `partialX (logical(1))`  
  Indicates whether covariates \( X \) should be partialled out.
- `partialZ (logical(1))`  
  Indicates whether instruments \( Z \) should be partialled out.

Methods

**Public methods:**

- `DoubleMLPLIV$new()`  
- `DoubleMLPLIV$set_ml_nuisance_params()`  
- `DoubleMLPLIV$tune()`  
- `DoubleMLPLIV$clone()`  

**Method new():** Creates a new instance of this R6 class.

**Usage:**

```r
DoubleMLPLIV$new(
  data,  
  ml_l,  
  ml_m,  
  ml_r,  
  ml_g = NULL,  
  partialX = TRUE,  
  partialZ = FALSE,  
  n_folds = 5,  
  n_rep = 1,  
  score = "partialling out",  
  dml_procedure = "dml2",  
  draw_sample_splitting = TRUE,
)```

apply_cross_fitting = TRUE
}

Arguments:

data (DoubleMLData)

The DoubleMLData object providing the data and specifying the variables of the causal model.

ml_l  (LearnerRegr, Learner, character(1))

A learner of the class LearnerRegr, which is available from mlr3 or its extension packages mlr3learners or mlr3extralearners. Alternatively, a Learner object with public field task_type = "regr" can be passed, for example of class GraphLearner. The learner can possibly be passed with specified parameters, for example lrn("regr.cv_glmnet", s = "lambda.min").

ml_l refers to the nuisance function \( l_0(X) = E[Y|X] \).

ml_m  (LearnerRegr, Learner, character(1))

A learner of the class LearnerRegr, which is available from mlr3 or its extension packages mlr3learners or mlr3extralearners. Alternatively, a Learner object with public field task_type = "regr" can be passed, for example of class GraphLearner. The learner can possibly be passed with specified parameters, for example lrn("regr.cv_glmnet", s = "lambda.min").

ml_m refers to the nuisance function \( m_0(X) = E[Z|X] \).

ml_r  (LearnerRegr, Learner, character(1))

A learner of the class LearnerRegr, which is available from mlr3 or its extension packages mlr3learners or mlr3extralearners. Alternatively, a Learner object with public field task_type = "regr" can be passed, for example of class GraphLearner. The learner can possibly be passed with specified parameters, for example lrn("regr.cv_glmnet", s = "lambda.min").

ml_r refers to the nuisance function \( r_0(X) = E[D|X] \).

ml_g  (LearnerRegr, Learner, character(1))

A learner of the class LearnerRegr, which is available from mlr3 or its extension packages mlr3learners or mlr3extralearners. Alternatively, a Learner object with public field task_type = "regr" can be passed, for example of class GraphLearner. The learner can possibly be passed with specified parameters, for example lrn("regr.cv_glmnet", s = "lambda.min").

ml_g refers to the nuisance function \( g_0(X) = E[Y - D\theta_0 | X] \). Note: The learner ml_g is only required for the score 'IV-type'. Optionally, it can be specified and estimated for callable scores.

partialX (logical(1))

Indicates whether covariates \( X \) should be partialled out. Default is TRUE.

partialZ (logical(1))

Indicates whether instruments \( Z \) should be partialled out. Default is FALSE.

n_folds (integer(1))

Number of folds. Default is 5.

n_rep (integer(1))

Number of repetitions for the sample splitting. Default is 1.

score (character(1), function())

A character(1) ("partialling out" or "IV-type") or a function() specifying the score function. If a function() is provided, it must be of the form function(y, z, d, l_hat, m_hat, r_hat, g_hat, smpls)
DoubleMLPLIV

and the returned output must be a named list() with elements \(\psi_a\) and \(\psi_b\). Default is "partialling out".

dml_procedure (character(1))
  A character(1) ("dml1" or "dml2") specifying the double machine learning algorithm.
  Default is "dml2".

draw_sample_splitting (logical(1))
  Indicates whether the sample splitting should be drawn during initialization of the object.
  Default is TRUE.

apply_cross_fitting (logical(1))
  Indicates whether cross-fitting should be applied. Default is TRUE.

**Method** set_ml_nuisance_params(): Set hyperparameters for the nuisance models of DoubleML models.
Note that in the current implementation, either all parameters have to be set globally or all parameters have to be provided fold-specific.

*Usage:*
DoubleMLPLIV$set_ml_nuisance_params(
  learner = NULL,
  treat_var = NULL,
  params,
  set_fold_specific = FALSE
)

*Arguments:*
learner (character(1))
  The nuisance model/learner (see method params_names).
treat_var (character(1))
  The treatment variable (hyperparameters can be set treatment-variable specific).
params (named list())
  A named list() with estimator parameters. Parameters are used for all folds by default.
  Alternatively, parameters can be passed in a fold-specific way if option fold_specific is TRUE. In this case, the outer list needs to be of length \(n_{rep}\) and the inner list of length \(n_{folds}\).
set_fold_specific (logical(1))
  Indicates if the parameters passed in params should be passed in fold-specific way. Default is FALSE. If TRUE, the outer list needs to be of length \(n_{rep}\) and the inner list of length \(n_{folds}\). Note that in the current implementation, either all parameters have to be set globally or all parameters have to be provided fold-specific.

*Returns:* self

**Method** tune(): Hyperparameter-tuning for DoubleML models.
The hyperparameter-tuning is performed using the tuning methods provided in the mlr3tuning package. For more information on tuning in mlr3, we refer to the section on parameter tuning in the mlr3 book.

*Usage:*
DoubleMLPLIV$tune(
  param_set,
tune_settings = list(n_folds_tune = 5, rsmp_tune = mlr3::rsmp("cv", folds = 5), measure = NULL, terminator = mlr3tuning::trm("evals", n_evals = 20), algorithm = mlr3tuning::tnr("grid_search"), resolution = 5),
tune_on_folds = FALSE)
)

Arguments:

param_set (named list())
  A named list with a parameter grid for each nuisance model/learner (see method learner_names()).
  The parameter grid must be an object of class ParamSet.

tune_settings (named list())
  A named list with arguments passed to the hyperparameter-tuning with mlr3tuning to set up TuningInstance objects. tune_settings has entries
  • terminator (Terminator)
    A Terminator object. Specification of terminator is required to perform tuning.
  • algorithm (Tuner or character(1))
    A Tuner object (recommended) or key passed to the respective dictionary to specify the tuning algorithm used in tnr(). algorithm is passed as an argument to tnr(). If algorithm is not specified by the users, default is set to "grid_search". If set to "grid_search", then additional argument "resolution" is required.
  • rsmp_tune (Resampling or character(1))
    A Resampling object (recommended) or option passed to rsmp() to initialize a Resampling for parameter tuning in mlr3. If not specified by the user, default is set to "cv" (cross-validation).
  • n_folds_tune (integer(1), optional)
    If rsmp_tune = "cv", number of folds used for cross-validation. If not specified by the user, default is set to 5.
  • measure (NULL, named list(), optional)
    Named list containing the measures used for parameter tuning. Entries in list must either be Measure objects or keys to be passed to passed to msr(). The names of the entries must match the learner names (see method learner_names()). If set to NULL, default measures are used, i.e., "regr.mse" for continuous outcome variables and "classif.ce" for binary outcomes.
  • resolution (character(1))
    The key passed to the respective dictionary to specify the tuning algorithm used in tnr(). resolution is passed as an argument to tnr().

tune_on_folds (logical(1))
  Indicates whether the tuning should be done fold-specific or globally. Default is FALSE.

Returns: self

Method clone(): The objects of this class are cloneable with this method.

Usage:
DoubleMLPLIV$clone(deep = FALSE)

Arguments:
deep  Whether to make a deep clone.
See Also

Other DoubleML: `DoubleMLIVM`, `DoubleMLIRM`, `DoubleMLPLR`, `DoubleML`

Examples

```r
library(DoubleML)
library(mlr3)
library(mlr3learners)
library(data.table)
set.seed(2)
ml_l = lrn("regr.ranger", num.trees = 100, mtry = 20, min.node.size = 2, max.depth = 5)
ml_m = ml_l$clone()
ml_r = ml_l$clone()
obj_dml_data = make_pliv_CHS2015(alpha = 1, n_obs = 500, dim_x = 20, dim_z = 1)
dml_pliv_obj = DoubleMLPLIV$new(obj_dml_data, ml_l, ml_m, ml_r)
dml_pliv_obj$fit()
dml_pliv_obj$summary()

## Not run:
library(DoubleML)
library(mlr3)
library(mlr3learners)
library(mlr3tuning)
library(data.table)
set.seed(2)
ml_l = lrn("regr.rpart")
ml_m = ml_l$clone()
ml_r = ml_l$clone()
obj_dml_data = make_pliv_CHS2015(
  alpha = 1, n_obs = 500, dim_x = 20,
  dim_z = 1)
dml_pliv_obj = DoubleMLPLIV$new(obj_dml_data, ml_l, ml_m, ml_r)
param_grid = list(
  "ml_l" = paradox::ParamSet$new(list(
    paradox::ParamDbl$new("cp", lower = 0.01, upper = 0.02),
    paradox::ParamInt$new("minsplit", lower = 1, upper = 2))),
  "ml_m" = paradox::ParamSet$new(list(
    paradox::ParamDbl$new("cp", lower = 0.01, upper = 0.02),
    paradox::ParamInt$new("minsplit", lower = 1, upper = 2))),
  "ml_r" = paradox::ParamSet$new(list(
    paradox::ParamDbl$new("cp", lower = 0.01, upper = 0.02),
    paradox::ParamInt$new("minsplit", lower = 1, upper = 2))))
# minimum requirements for tune_settings
tune_settings = list(
  terminator = mlr3tuning::trm("evals", n_evals = 5),
  algorithm = mlr3tuning::tnr("grid_search", resolution = 5))
dml_pliv_obj$tune(param_set = param_grid, tune_settings = tune_settings)
dml_pliv_obj$fit()
dml_pliv_obj$summary()
```
DoubleMLPLR

Double machine learning for partially linear regression models

Description
Double machine learning for partially linear regression models.

Format
R6::R6Class object inheriting from DoubleML.

Details
Partially linear regression (PLR) models take the form
\[ Y = \theta_0 D + g_0(X) + \xi, \]
\[ D = m_0(X) + \nu, \]
with \( E[\xi|D, X] = 0 \) and \( E[\nu|X] = 0 \). \( Y \) is the outcome variable, \( D \) is the policy variable of interest. The high-dimensional vector \( X = (X_1, \ldots, X_p) \) consists of other confounding covariates, and \( \xi \) and \( \nu \) are stochastic errors.

Super class
DoubleML::DoubleML -> DoubleMLPLR

Methods
Public methods:
- DoubleMLPLR$new()
- DoubleMLPLR$set_ml_nuisance_params()
- DoubleMLPLR$tune()
- DoubleMLPLR$clone()

Method new(): Creates a new instance of this R6 class.

Usage:
DoubleMLPLR$new(
  data,
  ml_l,
  ml_m,
  ml_g = NULL,
  n_folds = 5,
  n_rep = 1,
  score = "partialling out",
  dml_procedure = "dml2",
)
draw_sample_splitting = TRUE,
apply_cross_fitting = TRUE
)

Arguments:
data (DoubleMLData)
The DoubleMLData object providing the data and specifying the variables of the causal model.
ml_l (LearnerRegr, Learner, character(1))
A learner of the class LearnerRegr, which is available from mlr3 or its extension packages mlr3learners or mlr3extralearners. Alternatively, a Learner object with public field task_type = "regr" can be passed, for example of class GraphLearner. The learner can possibly be passed with specified parameters, for example lrn("regr.cv_glmnet", s = "lambda.min").
ml_l refers to the nuisance function \( l_0(X) = E[Y|X] \).
ml_m (LearnerRegr, LearnerClassif, Learner, character(1))
A learner of the class LearnerRegr, which is available from mlr3 or its extension packages mlr3learners or mlr3extralearners. For binary treatment variables, an object of the class LearnerClassif can be passed, for example lrn("classif.cv_glmnet", s = "lambda.min"). Alternatively, a Learner object with public field task_type = "regr" or task_type = "classif" can be passed, respectively, for example of class GraphLearner.
ml_m refers to the nuisance function \( m_0(X) = E[D|X] \).
ml_g (LearnerRegr, Learner, character(1))
A learner of the class LearnerRegr, which is available from mlr3 or its extension packages mlr3learners or mlr3extralearners. Alternatively, a Learner object with public field task_type = "regr" can be passed, for example of class GraphLearner. The learner can possibly be passed with specified parameters, for example lrn("regr.cv_glmnet", s = "lambda.min").
ml_g refers to the nuisance function \( g_0(X) = E[Y - D\theta_0|X] \). Note: The learner ml_g is only required for the score 'IV-type'. Optionally, it can be specified and estimated for callable scores.
n_folds (integer(1))
Number of folds. Default is 5.
n_rep (integer(1))
Number of repetitions for the sample splitting. Default is 1.
score (character(1), function())
A character(1) ("partialling out" or "IV-type") or a function() specifying the score function. If a function() is provided, it must be of the form function(y, d, l_hat, m_hat, g_hat, smpls) and the returned output must be a named list() with elements psi_a and psi_b. Default is "partialling out".
dml_procedure (character(1))
A character(1) ("dml1" or "dml2") specifying the double machine learning algorithm. Default is "dml2".
draw_sample_splitting (logical(1))
Indicates whether the sample splitting should be drawn during initialization of the object. Default is TRUE.
apply_cross_fitting (logical(1))
Indicates whether cross-fitting should be applied. Default is TRUE.
Method `set_ml_nuisance_params()`: Set hyperparameters for the nuisance models of DoubleML models.

Note that in the current implementation, either all parameters have to be set globally or all parameters have to be provided fold-specific.

Usage:

```r
DoubleMLPLR$set_ml_nuisance_params(
  learner = NULL,
  treat_var = NULL,
  params,
  set_fold_specific = FALSE
)
```

Arguments:

- **learner** (character(1))
  - The nuisance model/learner (see method `params_names`).

- **treat_var** (character(1))
  - The treatment variable (hyperparameters can be set treatment-variable specific).

- **params** (named list())
  - A named list() with estimator parameters. Parameters are used for all folds by default. Alternatively, parameters can be passed in a fold-specific way if option `fold_specific` is TRUE. In this case, the outer list needs to be of length `n_rep` and the inner list of length `n_folds`.

- **set_fold_specific** (logical(1))
  - Indicates if the parameters passed in `params` should be passed in fold-specific way. Default is FALSE. If TRUE, the outer list needs to be of length `n_rep` and the inner list of length `n_folds`. Note that in the current implementation, either all parameters have to be set globally or all parameters have to be provided fold-specific.

Returns: self

Method `tune()`: Hyperparameter-tuning for DoubleML models.

The hyperparameter-tuning is performed using the tuning methods provided in the `mlr3tuning` package. For more information on tuning in `mlr3`, we refer to the section on parameter tuning in the `mlr3` book.

Usage:

```r
DoubleMLPLR$tune(
  param_set,
  tune_settings = list(n_folds_tune = 5, rsmp_tune = mlr3::rsmp("cv", folds = 5), measure = NULL, terminator = mlr3tuning::trm("evals", n_evals = 20), algorithm = mlr3tuning::tnr("grid_search"), resolution = 5),
  tune_on_folds = FALSE
)
```

Arguments:

- **param_set** (named list())
  - A named list with a parameter grid for each nuisance model/learner (see method `learner_names()`).
  - The parameter grid must be an object of class `ParamSet`. 
tune_settings (named list())
A named list() with arguments passed to the hyperparameter-tuning with \texttt{mlr3tuning} to set up \texttt{TuningInstance} objects. \texttt{tune_settings} has entries
- **terminator** (Terminator)
  A \texttt{Terminator} object. Specification of \texttt{terminator} is required to perform tuning.
- **algorithm** (Tuner or character(1))
  A \texttt{Tuner} object (recommended) or key passed to the respective dictionary to specify the tuning algorithm used in \texttt{tnr()}. \texttt{algorithm} is passed as an argument to \texttt{tnr()}. If \texttt{algorithm} is not specified by the users, default is set to "grid_search". If set to "grid_search", then additional argument "resolution" is required.
- **rsm_tune** (Resampling or character(1))
  A \texttt{Resampling} object (recommended) or option passed to \texttt{rsmp()} to initialize a \texttt{Resampling} for parameter tuning in \texttt{mlr3}. If not specified by the user, default is set to "cv" (cross-validation).
- **n_folds_tune** (integer(1), optional)
  If \texttt{rsm_tune} = "cv", number of folds used for cross-validation. If not specified by the user, default is set to 5.
- **measure** (NULL, named list(), optional)
  Named list containing the measures used for parameter tuning. Entries in list must either be \texttt{Measure} objects or keys to be passed to \texttt{msr()}. The names of the entries must match the learner names (see method \texttt{learner_names()}). If set to NULL, default measures are used, i.e., "regr.mse" for continuous outcome variables and "classif.ce" for binary outcomes.
- **resolution** (character(1))
  The key passed to the respective dictionary to specify the tuning algorithm used in \texttt{tnr()}. \texttt{resolution} is passed as an argument to \texttt{tnr()}.

\texttt{tune_on_folds} (logical(1))
Indicates whether the tuning should be done fold-specific or globally. Default is FALSE.

\textbf{Returns:} \texttt{self}

\textbf{Method} \texttt{clone()}: The objects of this class are cloneable with this method.

\textbf{Usage:}
\texttt{DoubleMLPLR$clone(deep = FALSE)}

\textbf{Arguments:}
deep Whether to make a deep clone.

\textbf{See Also}
Other \texttt{DoubleML}: \texttt{DoubleMLIIVM, DoubleMLIRM, DoubleMLPLIV, DoubleML}

\textbf{Examples}

\begin{verbatim}
library(DoubleML)
library(mlr3)
library(mlr3learners)
library(data.table)
\end{verbatim}
`double_ml_data_from_data_frame`  

Wrapper for Double machine learning data-backend initialization from `data.frame`.

**Description**

Initialization of `DoubleMLData` from `data.frame`.
Usage

double_ml_data_from_data_frame(
    df,
    x_cols = NULL,
    y_col = NULL,
    d_cols = NULL,
    z_cols = NULL,
    cluster_cols = NULL,
    use_other_treat_as_covariate = TRUE
)

Arguments

df (data.frame())
Data object.

x_cols (NULL, character())
The covariates. If NULL, all variables (columns of data) which are neither specified as outcome variable y_col, nor as treatment variables d_cols, nor as instrumental variables z_cols are used as covariates. Default is NULL.

y_col (character(1))
The outcome variable.

d_cols (character())
The treatment variable(s).

z_cols (NULL, character())
The instrumental variables. Default is NULL.

cluster_cols (NULL, character())
The cluster variables. Default is NULL.

use_other_treat_as_covariate (logical(1))
Indicates whether in the multiple-treatment case the other treatment variables should be added as covariates. Default is TRUE.

Value

Creates a new instance of class DoubleMLData.

Examples

df = make_plr_CCDDHNR2018(return_type = "data.frame")
x_names = names(df)[grep("X", names(df))]
obj_dml_data = double_ml_data_from_data_frame(
    df = df, x_cols = x_names,
    y_col = "y", d_cols = "d")
# Input: Data frame, Output: DoubleMLData object
double_ml_data_from_matrix

Wrapper for Double machine learning data-backend initialization from matrix.

Description

Initalization of DoubleMLData from matrix() objects.

Usage

double_ml_data_from_matrix(
  X = NULL,
  y,
  d,
  z = NULL,
  cluster_vars = NULL,
  data_class = "DoubleMLData",
  use_other_treat_as_covariate = TRUE
)

Arguments

X (matrix())
Matrix of covariates.

y (numeric())
Vector of outcome variable.

d (matrix())
Matrix of treatment variables.

z (matrix())
Matrix of instruments.

cluster_vars (matrix())
Matrix of cluster variables.

data_class (character(1))
Class of returned object. By default, an object of class DoubleMLData is returned. Setting data_class = "data.table" returns an object of class data.table.

use_other_treat_as_covariate (logical(1))
Indicates whether in the multiple-treatment case the other treatment variables should be added as covariates. Default is TRUE.

Value

Creates a new instance of class DoubleMLData.
**Examples**

```r
class_list = make_plr_CCDDHNR2018(return_type = "matrix")
obj_dml_data = double_ml_data_from_matrix(
    X = matrix_list$X,
    y = matrix_list$y,
    d = matrix_list$d)
```

---

**fetch_401k**  
*Data set on financial wealth and 401(k) plan participation.*

---

**Description**

Preprocessed data set on financial wealth and 401(k) plan participation. The raw data files are preprocessed to reproduce the examples in Chernozhukov et al. (2020). An internet connection is required to successfully download the data set.

**Usage**

```r
fetch_401k(
    return_type = "DoubleMLData",
    polynomial_features = FALSE,
    instrument = FALSE
)
```

**Arguments**

- `return_type` *(character(1))*
  - If "DoubleMLData", returns a DoubleMLData object. If "data.frame" returns a data.frame(). If "data.table" returns a data.table(). Default is "DoubleMLData".
- `polynomial_features` *(logical(1))*
  - If TRUE polynomial features are added (see replication file of Chernozhukov et al. (2018)).
- `instrument` *(logical(1))*
  - If TRUE, the returned data object contains the variables e401 and p401. If return_type = "DoubleMLData", the variable e401 is used as an instrument for the endogenous treatment variable p401. If FALSE, p401 is removed from the data set.

**Details**

Variable description, based on the supplementary material of Chernozhukov et al. (2020):

- `net_tfa`: net total financial assets
- `e401`: = 1 if employer offers 401(k)
- `p401`: = 1 if individual participates in a 401(k) plan
- `age`: age
fetch_bonus

- inc: income
- fsize: family size
- educ: years of education
- db: = 1 if individual has defined benefit pension
- marr: = 1 if married
- twoearn: = 1 if two-earner household
- pira: = 1 if individual participates in IRA plan
- hown: = 1 if home owner

The supplementary data of the study by Chernozhukov et al. (2018) is available at https://academic.oup.com/ectj/article/21/1/C1/5056401#supplementary-data.

Value

A data object according to the choice of return_type.

References


fetch_bonus

Data set on the Pennsylvania Reemployment Bonus experiment.

Description

Preprocessed data set on the Pennsylvania Reemployment Bonus experiment. The raw data files are preprocessed to reproduce the examples in Chernozhukov et al. (2020). An internet connection is required to successfully download the data set.

Usage

fetch_bonus(return_type = "DoubleMLData", polynomial_features = FALSE)

Arguments

return_type (character(1))
If "DoubleMLData", returns a DoubleMLData object. If "data.frame" returns a data.frame(). If "data.table" returns a data.table(). Default is "DoubleMLData".

polynomial_features (logical(1))
If TRUE polynomial features are added (see replication file of Chernozhukov et al. (2018)).
Details
Variable description, based on the supplementary material of Chernozhukov et al. (2020):

• abdt: chronological time of enrollment of each claimant in the Pennsylvania reemployment bonus experiment.
• tg: indicates the treatment group (bonus amount - qualification period) of each claimant.
• inuidur1: a measure of length (in weeks) of the first spell of unemployment
• inuidur2: a second measure for the length (in weeks) of
• female: dummy variable; it indicates if the claimant’s sex is female (=1) or male (=0).
• black: dummy variable; it indicates a person of black race (=1).
• hispanic: dummy variable; it indicates a person of hispanic race (=1).
• othrace: dummy variable; it indicates a non-white, non-black, not-hispanic person (=1).
• dep1: dummy variable; indicates if the number of dependents of each claimant is equal to 1 (=1).
• dep2: dummy variable; indicates if the number of dependents of each claimant is equal to 2 (=1).
• q1-q6: six dummy variables indicating the quarter of experiment during which each claimant enrolled.
• recall: takes the value of 1 if the claimant answered “yes” when was asked if he/she had any expectation to be recalled
• agelt35: takes the value of 1 if the claimant’s age is less than 35 and 0 otherwise.
• agegt54: takes the value of 1 if the claimant’s age is more than 54 and 0 otherwise.
• durable: it takes the value of 1 if the occupation of the claimant was in the sector of durable manufacturing and 0 otherwise.
• nondurable: it takes the value of 1 if the occupation of the claimant was in the sector of nondurable manufacturing and 0 otherwise.
• lusd: it takes the value of 1 if the claimant filed in Coatesville, Reading, or Lancaster and 0 otherwise.
• husd: it takes the value of 1 if the claimant filed in Lewistown, Pittston, or Scranton and 0 otherwise.
• muld: it takes the value of 1 if the claimant filed in Philadelphia-North, Philadelphia-Uptown, McKeesport, Erie, or Butler and 0 otherwise.

These three sites were considered to be located in areas characterized by moderate unemployment rate and long duration of unemployment.

The supplementary data of the study by Chernozhukov et al. (2018) is available at https://academic.oup.com/ectj/article/21/1/C1/5056401#supplementary-data.

**make_iivm_data**

Generates data from a interactive IV regression (IIVM) model.

**Description**

Generates data from a interactive IV regression (IIVM) model. The data generating process is defined as

\[ d_i = 1 \{ \alpha_x Z + v_i > 0 \}, \]

\[ y_i = \theta d_i + x_i' \beta + u_i, \]

\[ Z \sim Bernoulli(0.5) \text{ and} \]

\[ \begin{pmatrix} u_i \\ v_i \end{pmatrix} \sim \mathcal{N} \left( 0, \begin{pmatrix} 1 & 0.3 \\ 0.3 & 1 \end{pmatrix} \right). \]

The covariates \( x_i \sim \mathcal{N}(0, \Sigma) \), where \( \Sigma \) is a matrix with entries \( \Sigma_{kj} = 0.5^{|j-k|} \) and \( \beta \) is a \( \text{dim}_\text{x} \)-vector with entries \( \beta_j = \frac{1}{j^2} \).

The data generating process is inspired by a process used in the simulation experiment of Farbmacher, Gruber and Klaaßen (2020).

**Value**

A data object according to the choice of return_type.

**References**


**Examples**

```r
library(DoubleML)
df_bonus = fetch_bonus(return_type = "data.table")
obj_dml_data_bonus = DoubleMLData$new(df_bonus,
  y_col = "inuidur1",
  d_cols = "tg",
  x_cols = c(
    "female", "black", "othrace", "dep1", "dep2",
    "q2", "q3", "q4", "q5", "q6", "agelt35", "agegt54",
    "durable", "lusd", "husd"
  )
)
obj_dml_data_bonus
```

---

**make_iivm_data**

Generates data from a interactive IV regression (IIVM) model.
Usage

```r
make_iivm_data(
    n_obs = 500,
    dim_x = 20,
    theta = 1,
    alpha_x = 0.2,
    return_type = "DoubleMLData"
)
```

Arguments

- **n_obs** (integer(1))
  The number of observations to simulate.
- **dim_x** (integer(1))
  The number of covariates.
- **theta** (numeric(1))
  The value of the causal parameter.
- **alpha_x** (numeric(1))
  The value of the parameter $\alpha_x$.
- **return_type** (character(1))
  If "DoubleMLData", returns a DoubleMLData object. If "data.frame" returns a data.frame(). If "data.table" returns a data.table(). If "matrix" a named list() with entries X, y, d and z is returned. Every entry in the list is a matrix() object. Default is "DoubleMLData".

References


**make_iirm_data**

Generates data from a interactive regression (IRM) model.

Description

Generates data from a interactive regression (IRM) model. The data generating process is defined as

$$d_i = 1 \left\{ \frac{\exp(c_d x'_i \beta)}{1 + \exp(c_d x'_i \beta)} > v_i \right\},$$

$$y_i = \theta d_i + c_y x'_i \beta d_i + \zeta_i,$$

with $v_i \sim \mathcal{U}(0, 1)$, $\zeta_i \sim \mathcal{N}(0, 1)$ and covariates $x_i \sim \mathcal{N}(0, \Sigma)$, where $\Sigma$ is a matrix with entries $\Sigma_{kj} = 0.5^{|j-k|}$. $\beta$ is a dim_x-vector with entries $\beta_j = \frac{1}{j^4}$ and the constants $c_y$ and $c_d$ are given by

$$c_y = \sqrt{\frac{R^2_y}{(1-R^2_y)^2 \beta^4 \Sigma_{yy}}},$$

$$c_d = \sqrt{\frac{R^2_d}{(1-R^2_d)^2 \beta^4 \Sigma_{dd}}}.$$
\[ c_d = \sqrt{\frac{(\pi^2/3)R_d^2}{1-R_d^2}}. \]

The data generating process is inspired by a process used in the simulation experiment (see Appendix P) of Belloni et al. (2017).

Usage

```r
make_irm_data(
  n_obs = 500,
  dim_x = 20,
  theta = 0,
  R2_d = 0.5,
  R2_y = 0.5,
  return_type = "DoubleMLData"
)
```

Arguments

- `n_obs` (integer(1))
  The number of observations to simulate.
- `dim_x` (integer(1))
  The number of covariates.
- `theta` (numeric(1))
  The value of the causal parameter.
- `R2_d` (numeric(1))
  The value of the parameter \( R_d^2 \).
- `R2_y` (numeric(1))
  The value of the parameter \( R_y^2 \).
- `return_type` (character(1))
  If "DoubleMLData", returns a DoubleMLData object. If "data.frame" returns a data.frame(). If "data.table" returns a data.table(). If "matrix" a named list() with entries X, y, d and z is returned. Every entry in the list is a matrix() object. Default is "DoubleMLData".

References


**Description**

Generates data from a partially linear IV regression model used in Chernozhukov, Hansen and Spindler (2015). The data generating process is defined as

\[ z_i = \Pi x_i + \zeta_i, \]
\[ d_i = x_i' \gamma + z_i' \delta + u_i, \]
\[ y_i = \alpha d_i + x_i' \beta + \epsilon_i, \]

with

\[
\begin{pmatrix}
\varepsilon_i \\
u_i \\
zeta_i \\
x_i
\end{pmatrix}
\sim \mathcal{N}
\begin{pmatrix}
1 & 0.6 & 0 & 0 \\
0.6 & 1 & 0 & 0 \\
0 & 0 & 0.25 I_{p_{zn}} & 0 \\
0 & 0 & 0 & \Sigma
\end{pmatrix}
\]

where \( \Sigma \) is a \( p_{zn} \times p_{zn} \) matrix with entries \( \Sigma_{kj} = 0.5|j-k| \) and \( I_{p_{zn}} \) is the \( p_{zn} \times p_{zn} \) identity matrix. \( \beta \) is a \( p_{zn} \)-vector with entries \( \beta_j = \frac{1}{j^2} \), \( \delta \) is a \( p_{zn} \)-vector with entries \( \delta_j = \frac{1}{j^2} \) and \( \Pi = (I_{p_{zn}}, O_{p_{zn} \times (p_{xn} - p_{zn})}) \).

**Usage**

```r
make_pliv_CHS2015(
  n_obs,
  alpha = 1,
  dim_x = 200,
  dim_z = 150,
  return_type = "DoubleMLData"
)
```

**Arguments**

- **n_obs** (integer(1))
  
  The number of observations to simulate.

- **alpha** (numeric(1))
  
  The value of the causal parameter.

- **dim_x** (integer(1))
  
  The number of covariates.

- **dim_z** (integer(1))
  
  The number of instruments.

- **return_type** (character(1))
  
  If "DoubleMLData", returns a DoubleMLData object. If "data.frame" returns a data.frame(). If "data.table" returns a data.table(). If "matrix" a named list() with entries X, y, d and z is returned. Every entry in the list is a matrix() object. Default is "DoubleMLData".

**Value**

A data object according to the choice of return_type.
References


Generates data from a partially linear IV regression model with multiway cluster sample used in Chiang et al. (2021).

Description

Generates data from a partially linear IV regression model with multiway cluster sample used in Chiang et al. (2021). The data generating process is defined as

\[ Z_{ij} = X'_{ij} \xi_0 + V_{ij}, \]
\[ D_{ij} = Z'_{ij} \pi_{10} + X'_{ij} \pi_{20} + v_{ij}, \]
\[ Y_{ij} = D_{ij} \theta + X'_{ij} \zeta_0 + \varepsilon_{ij}, \]

with

\[ X_{ij} = (1 - \omega_1 X^X - \omega_2 X^X) \alpha_{ij}^{X} + \omega_1 X^X \alpha_{ij}^{X} + \omega_2 X^X \alpha_{ij}^{X}, \]
\[ \varepsilon_{ij} = (1 - \omega_1 \varepsilon^\varepsilon - \omega_2 \varepsilon^\varepsilon) \alpha_{ij}^{\varepsilon} + \omega_1 \varepsilon^{\varepsilon} \alpha_{ij}^{\varepsilon} + \omega_2 \varepsilon^{\varepsilon} \alpha_{ij}^{\varepsilon}, \]
\[ v_{ij} = (1 - \omega_1 V - \omega_2 V) \alpha_{ij}^{V} + \omega_1 V \alpha_{ij}^{V} + \omega_2 V \alpha_{ij}^{V}, \]
\[ V_{ij} = (1 - \omega_1 V - \omega_2 V) \alpha_{ij}^{V} + \omega_1 V \alpha_{ij}^{V} + \omega_2 V \alpha_{ij}^{V}, \]

and \( \alpha_{ij}^{X}, \alpha_{ij}^{X}, \alpha_{ij}^{X} \sim \mathcal{N}(0, \Sigma) \) where \( \Sigma \) is a \( p_x \times p_x \) matrix with entries \( \Sigma_{kj} = s_{X}^{j-k} \).

Further

\[ \begin{pmatrix} \alpha_{ij}^{\varepsilon} \\ \alpha_{ij}^{V} \end{pmatrix}, \begin{pmatrix} \alpha_{i}^{\varepsilon} \\ \alpha_{i}^{V} \end{pmatrix}, \begin{pmatrix} \alpha_{j}^{\varepsilon} \\ \alpha_{j}^{V} \end{pmatrix} \sim \mathcal{N} \left( 0, \begin{pmatrix} 1 & s_{\varepsilon V} \\ s_{\varepsilon V} & 1 \end{pmatrix} \right) \]

and \( \alpha_{ij}^{V}, \alpha_{i}^{V}, \alpha_{j}^{V} \sim \mathcal{N}(0, 1). \)

Usage

```r
make_pliv_multiway_cluster_CKMS2021(
  N = 25,
  M = 25,
  dim_X = 100,
  theta = 1,
  return_type = "DoubleMLClusterData",
  ...
)
```
Arguments

N \ (integer(1))
The number of observations (first dimension).

M \ (integer(1))
The number of observations (second dimension).

dim_X \ (integer(1))
The number of covariates.

theta \ (numeric(1))
The value of the causal parameter.

return_type \ (character(1))
If "DoubleMLClusterData", returns a DoubleMLClusterData object. If "data.frame" returns a data.frame(). If "data.table" returns a data.table(). If "matrix" a named list() with entries X, y, d, z and cluster_vars is returned. Every entry in the list is a matrix() object. Default is "DoubleMLClusterData".

... Additional keyword arguments to set non-default values for the parameters π_{10} = 1.0, ω_X = ω_z = ω_v = ω_0 = (0.25, 0.25), s_X = s_v = 0.25, or the p_x-vectors ζ_0 = π_{20} = ξ_0 with default entries ζ_j = 0.5^j.

Value

A data object according to the choice of return_type.

References


make_plr_CCDDHNR2018 Generates data from a partially linear regression model used in Chernozhukov et al. (2018)

Description

Generates data from a partially linear regression model used in Chernozhukov et al. (2018) for Figure 1. The data generating process is defined as

d_i = m_0(x_i) + s_1 v_i,
y_i = α d_i + g_0(x_i) + s_2 ζ_i,

with v_i \sim \mathcal{N}(0, 1) and ζ_i \sim \mathcal{N}(0, 1). The covariates are distributed as x_i \sim \mathcal{N}(0, Σ), where Σ is a matrix with entries Σ_{kj} = 0.7^{|j-k|}. The nuisance functions are given by

m_0(x_i) = a_0 x_{i,1} + a_1 \frac{\exp(x_{i,1})}{1 + \exp(x_{i,3})},
g_0(x_i) = b_0 \frac{\exp(x_{i,1})}{1 + \exp(x_{i,1})} + b_1 x_{i,3},

with a_0 = 1, a_1 = 0.25, s_1 = 1, b_0 = 1, b_1 = 0.25, s_2 = 1.
Usage

```r
make_plr_CCDDHNR2018(
    n_obs = 500,
    dim_x = 20,
    alpha = 0.5,
    return_type = "DoubleMLData"
)
```

Arguments

- `n_obs` (integer(1))
  The number of observations to simulate.
- `dim_x` (integer(1))
  The number of covariates.
- `alpha` (numeric(1))
  The value of the causal parameter.
- `return_type` (character(1))
  If "DoubleMLData", returns a DoubleMLData object. If "data.frame" returns a data.frame(). If "data.table" returns a data.table(). If "matrix" a named list() with entries `X`, `y` and `d` is returned. Every entry in the list is a matrix() object. Default is "DoubleMLData".

Value

A data object according to the choice of `return_type`.

References


make_plr_turrell2018

Generates data from a partially linear regression model used in a blog article by Turrell (2018).

Description

Generates data from a partially linear regression model used in a blog article by Turrell (2018). The data generating process is defined as

\[ d_i = m_0(x_i'b) + v_i, \]
\[ y_i = \theta d_i + g_0(x_i'b) + u_i, \]

with \( v_i \sim N(0,1) \), \( u_i \sim N(0,1) \), and covariates \( x_i \sim N(0, \Sigma) \), where \( \Sigma \) is a random symmetric, positive-definite matrix generated with `clusterGeneration::genPositiveDefMat()`. \( b \) is a vector with entries \( b_j = \frac{1}{j} \) and the nuisance functions are given by
\[ m_0(x_{i}) = \frac{1}{2\pi} \frac{\sinh(\gamma)}{\cosh(\gamma) - \cos(x_{i} - \nu)}, \]
\[ g_0(x_{i}) = \sin(x_{i})^2. \]

Usage

```r
make_plr_turrell2018(
  n_obs = 100,
  dim_x = 20,
  theta = 0.5,
  return_type = "DoubleMLData",
  nu = 0,
  gamma = 1
)
```

Arguments

- **n_obs** (integer(1))
  The number of observations to simulate.
- **dim_x** (integer(1))
  The number of covariates.
- **theta** (numeric(1))
  The value of the causal parameter.
- **return_type** (character(1))
  If "DoubleMLData", returns a DoubleMLData object. If "data.frame" returns a data.frame(). If "data.table" returns a data.table(). If "matrix" a named list() with entries X, y and d is returned. Every entry in the list is a matrix() object. Default is "DoubleMLData".
- **nu** (numeric(1))
  The value of the parameter \( \nu \). Default is 0.
- **gamma** (numeric(1))
  The value of the parameter \( \gamma \). Default is 1.

Value

A data object according to the choice of return_type.

References

Index

* DoubleML
  DoubleML, 2
  DoubleMLIIVM, 14
  DoubleMLIRM, 18
  DoubleMLPLIV, 21
  DoubleMLPLR, 27

clusterGeneration::genPositiveDefMat(), 43

data.table, 3, 10–13
double_ml_data_from_data_frame, 31
double_ml_data_from_data_frame(), 10, 12
double_ml_data_from_matrix, 33
double_ml_data_from_matrix(), 10, 12
DoubleML, 2, 14, 17, 18, 20, 22, 26, 27, 30
DoubleML::DoubleML, 15, 19, 22, 27
DoubleML::DoubleMLData, 10
DoubleMLClusterData, 10
DoubleMLData, 12
DoubleMLIIVM, 9, 14, 20, 26, 30
DoubleMLIRM, 9, 17, 18, 26, 30
DoubleMLPLIV, 9, 17, 20, 21, 30
DoubleMLPLR, 9, 17, 20, 26, 27

fetch_401k, 34
fetch_bonus, 35

GraphLearner, 16, 19, 20, 23, 28
Learner, 16, 19, 20, 23, 28
LearnerClassif, 16, 19, 20, 28
LearnerRegr, 16, 19, 23, 28

make_iivm_data, 37
make_irm_data, 38
make_pliv_CHS2015, 39
make_pliv_multiway_cluster_CKMS2021, 41
make_plr_CCDDHNR2018, 42
make_plr_turrell2018, 43

Measure, 7, 25, 30
msr(), 7, 25, 30

p.adjust(), 8
ParamSet, 6, 25, 29

R6, 10, 13
R6::R6Class, 2, 14, 18, 22, 27
Resampling, 6, 25, 30
rsmp(), 6, 25, 30

Terminator, 6, 25, 30
tnr(), 6, 7, 25, 30
Tuner, 6, 25, 30
TuningInstance, 6, 25, 30