Package ‘xtdml’

November 13, 2025

Type Package

Title Double Machine Learning for Static Panel Models with Fixed Effects

Version 0.1.9

Date 2025-11-12

Maintainer Annalivia Polselli <apolselli.econ@gmail.com>

Description
The 'xtdml' package implements partially linear panel regression (PLPR) models with high-dimensional confounding variables and an exogenous treatment variable within the double machine learning framework. The package is used to estimate the structural parameter (treatment effect) in static panel data models with fixed effects using the approaches established in Clarke and Polselli (2025) <doi:10.1093/ectj/utaf011>. 'xtdml' is built on the object-oriented package 'DoubleML' (Bach et al., 2024) <doi:10.18637/jss.v108.i03> using the 'mlr3' ecosystem.

License GPL-2 | GPL-3

Encoding UTF-8

Depends R (>= 3.5.0)

Imports R6 (>= 2.4.1), data.table (>= 1.12.8), mlr3 (>= 0.19.0), mlr3tuning (>= 0.20.0), mlr3learners (>= 0.3.0), mlr3misc (>= 0.19.0), mvtnorm, utils, clusterGeneration, readstata13, magrittr, dplyr, stats, MLmetrics, checkmate

RoxygenNote 7.3.2

Suggests rpart, mlr3pipelines, bbotk (>= 1.6.0)

NeedsCompilation no

Author Annalivia Polselli [aut, cre] (ORCID: <https://orcid.org/0009-0002-7579-7926>)

Repository CRAN

Date/Publication 2025-11-12 23:30:02 UTC

Contents

`make_plpr_data` *Generates data from a partially linear panel regression (PLPR) model*

Description

Generates data from a partially linear regression model for panel data with fixed effects similar to DGP3 (highly nonlinear) in Clarke and Polselli (2025).

The data generating process is defined as

$Y_{it} = \theta D_{it} + g_0(X_{it}) + \alpha_i + U_{it},$

$D_{it} = m_0(X_{it}) + \gamma_i + V_{it},$

where $U_{it} \sim \mathcal{N}(0,1)$, $V_{it} \sim \mathcal{N}(0,1)$, $\alpha_i = \rho A_i + \sqrt{1-\rho^2} B_i$ with $A_i \sim \mathcal{N}(3,3)$, $B_i \sim \mathcal{N}(0,1)$, and $\gamma_i \sim \mathcal{N}(0,5)$.

The covariates are distributed as $X_{it,p} \sim A_i + \mathcal{N}(0,5)$, where p is the number of covariates.

The nuisance functions are given by

$m_0(X_{it}) = a_1[X_{it,1} \times 1(X_{it,1} > 0)] + a_2[X_{it,1} \times X_{it,3}],$

$g_0(X_{it}) = b_1[X_{it,1} \times X_{it,3}] + b_2[X_{it,3} \times 1(X_{it,3} > 0)],$

with $a_1 = b_2 = 0.25$ and $a_2 = b_1 = 0.5$.

Usage

```
make_plpr_data(n_obs = 500, t_per = 10, dim_x = 20, theta = 0.5, rho = 0.8)
```

Arguments

`n_obs` (`integer(1)`)
The number of cross-sectional observations (i) to simulate.

`t_per` (`integer(1)`)
The number of time periods (t) to simulate.

`dim_x` (`integer(1)`)
The number of covariates.

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

`rho` (`numeric(1)`)
Parameter governing the relationship between the covariates and the unobserved individual heterogeneity. The value is chosen between 0 (pure random effect) and 1 (pure fixed effects).

Value

A data object.

References

Clarke, P. S. and Polselli, A. (2025). Double Machine Learning for Static Panel Models with Fixed Effects. Econometrics Journal. DOI: 10.1093/ectj/utaf011.

Examples

```
df = make_plpr_data(n_obs = 500, t_per = 10, dim_x = 20, theta = 0.5, rho=0.8)
```

`xtdml` *Abstract class xtdml*

Description

Abstract base class that cannot be initialized.

`xtdml` estimates the structural parameter (treatment effect) in partially linear panel regression models with fixed effects using double machine learning (Clarke and Polselli, 2025). `xtdml` allows the estimation of the nuisance functions in the model by machine learning methods based on the panel data approach chosen by the user, and computation of the Neyman-orthogonal score functions.

`xtdml` builds on the object-oriented architecture of `DoubleML` (Bach et al., 2024), using the 'mlr3' ecosystem and the 'R6' package. `xtdml` follows most of the notation of `DoubleML`.

Format

R6::R6Class object.

Active bindings

`all_coef_theta (matrix())`
: Estimates of the causal parameter(s) `"theta"` for the `n_rep` different sample splits after calling `fit()`.

`all_dml1_coef_theta (array())`
: Estimates of the causal parameter(s) `"theta"` for the `n_rep` different sample splits after calling `fit()` with `dml_procedure = "dml1"`.

`all_se_theta (matrix())`
: Standard errors of the causal parameter(s) `"theta"` for the `n_rep` different sample splits after calling `fit()`.

`all_model_rmse (matrix())`
: Model root-mean-squared-error.

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

`coef_theta` (`numeric()`)
  Estimates for the causal parameter(s) `"theta"` 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_theta` (`array()`)
  Value of the score function $\psi(W;\theta_0,\eta_0) = -\psi_a(W;\eta_0)\theta_0 + \psi_b(W;\eta_0)$ after calling `fit()`.

`psi_theta_a` (`array()`)
  Value of the score function component $\psi_a(W;\eta_0)$ after calling `fit()`.

`psi_theta_b` (`array()`)
  Value of the score function component $\psi_b(W;\eta_0)$ after calling `fit()`.

`res_y` (`array()`)
  Residual of output equation

`res_d` (`array()`)
  Residual of treatment equation

`predictions` (`array()`)
  Predictions of the nuisance models after calling `fit(store_predictions=TRUE)`.

`targets` (`array()`)
  Targets of the nuisance models after calling `fit(store_predictions=TRUE)`.

`rmses` (`array()`)
  The root-mean-squared-errors of the nuisance parameters

`all_model_mse` (`array()`)
  Collection of all mean-squared-errors of the model

`model_rmse` (`array()`)
  The root-mean-squared-errors of the model

`models` (`array()`)
  The fitted nuisance models after calling `fit(store_models=TRUE)`.

`pval_theta` (`numeric()`)
  p-values for the causal parameter(s) `"theta"` after calling `fit()`.

`score` (`character(1)`)
A `character(1)` specifying the score function among `"orth-PO"`, `"orth-IV"`. Default is `"orth-PO"`.

`se_theta` (`numeric()`)
Standard errors for the causal parameter(s) `"theta"` after calling `fit()`.

`smpls` (`list()`)
The partition used for cross-fitting.

`smpls_cluster` (`list()`)
The partition used for cross-fitting. smpl is at cluster-var

`t_stat_theta` (`numeric()`)
t-statistics for the causal parameter(s) `"theta"` after calling `fit()`.

`tuning_res_theta` (named `list()`)
Results from hyperparameter tuning.

Methods

Public methods:

- `xtdml$new()`
- `xtdml$print()`
- `xtdml$fit()`
- `xtdml$split_samples()`
- `xtdml$tune()`
- `xtdml$summary()`
- `xtdml$confint()`
- `xtdml$learner_names()`
- `xtdml$params_names()`
- `xtdml$set_ml_nuisance_params()`
- `xtdml$get_params()`
- `xtdml$clone()`

Method `new()`**:** DML with FE is an abstract class that can't be initialized.

Usage:

```
xtdml$new()
```

Method `print()`**:** Print 'DML with FE' objects.

Usage:

```
xtdml$print()
```

Method `fit()`**:** Estimate DML models with FE.

Usage:

```
xtdml$fit(store_predictions = FALSE, store_models = FALSE)
```

Arguments:

`store_predictions` (`logical(1)`)
Indicates whether the predictions for the nuisance functions should be stored in field `predictions`. Default is `FALSE`.

`store_models` (`logical(1)`)
Indicates whether the fitted models for the nuisance functions should be stored in field `models` if you want to analyze the models or extract information like variable importance. Default is `FALSE`.

Returns: self

Method `split_samples()`**:** Draw sample splitting for Double ML models with FE.

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

Usage:

```
xtdml$split_samples()
```

Returns: self

Method `tune()`**:** Hyperparameter tuning for Double Machine Learning (DML) models with fixed effects.

The hyperparameter tuning is performed using the tuning methods provided in the mlr3tuning package. For more information on tuning in mlr3, see the chapter on hyperparameter optimization in the mlr3 book.

Usage:

```
xtdml$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), tuner =
    mlr3tuning::tnr("grid_search", resolution = 10)),
  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()`). Each element must be a ParamSet object.

`tune_settings` (named `list()`)
A named `list()` of settings controlling the hyperparameter tuning process. Each entry is passed to the corresponding components from mlr3tuning:

- `terminator` (`[bbotk::Terminator]`)
  A Terminator object specifying when the tuning process should stop (e.g., `trm("evals", n_evals = 20)`).
- `tuner` — a Tuner object created with tnr(), which defines the optimization algorithm. (e.g., `tnr("grid_search")` or `tnr("random_search")`). If set to `"grid_search"`, then additional argument `"resolution"` is required.
- `rsmp_tune` — a Resampling object or a key passed to rsmp(). Defines the resampling strategy used during tuning (default: `"cv"`).
- `n_folds_tune` — an integer scalar (optional). Number of folds used if `rsmp_tune = "cv"`. Default is 5.
- `measure` — a named `list()` (optional). Contains the performance measures used for tuning. Each element must be either a Measure object or a key to msr(). Names must match the learner names (see `learner_names()`). If omitted, default measures are used (`"regr.rmse"` for regression and `"classif.ce"` for classification).

`tune_on_folds` (`logical(1)`)
Indicates whether the tuning should be performed separately for each cross-fitting fold (`TRUE`) or globally across all folds (`FALSE`, default).

Returns: self

Examples:

```
tune_settings = list(
  n_folds_tune = 5,
  rsmp_tune = mlr3::rsmp("cv", folds = 5),
  terminator = mlr3tuning::trm("evals", n_evals = 20),
  tuner = mlr3tuning::tnr("grid_search", resolution = 10))
```

Method `summary()`**:** Summary for DML models with FE after calling `fit()`.

Usage:

```
xtdml$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 DML models with FE.

Usage:

```
xtdml$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:

```
xtdml$learner_names()
```

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

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

Usage:

```
xtdml$params_names()
```

Returns: `character()` with names of nuisance models with hyperparameters.

Method `set_ml_nuisance_params()`: Set hyperparameters for the nuisance models of DML models with FE.

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

Usage:

```
xtdml$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 for time-varying covariates. 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_per_cluster`.

`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_per_cluster`. 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 `get_params()`: Get hyper-parameters for the nuisance model of xtdml models.

Usage:

```
xtdml$get_params(learner)
```

Arguments:

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

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

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

Usage:

```
xtdml$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other xtdml: `xtdml_plr`

Examples

```
## ------------------------------------------------
## Method `xtdml$tune`
## ------------------------------------------------

tune_settings = list(
  n_folds_tune = 5,
  rsmp_tune = mlr3::rsmp("cv", folds = 5),
  terminator = mlr3tuning::trm("evals", n_evals = 20),
  tuner = mlr3tuning::tnr("grid_search", resolution = 10))
```

`xtdml_data` *Set up for data for panel data approaches and up two cluster variables*

Description

Double machine learning (DML) data-backend for data with cluster variables. `xtdml_data` sets up the data environment for panel data analysis with transformed variables.

`xtdml_data` objects can be initialized from a data.table. The following functions can be used to create a new instance of `xtdml_data`.

- `xtdml_data$new()` for initialization from a `data.table`.
- `xtdml_data_from_data_frame()` for initialization from a `data.frame`.

Active bindings

`all_variables` (`character()`)
: All variables available in the data frame.

`d_cols` (`character()`)
: The treatment variable.

`dbar_col` (`NULL`, character()‘)
: The individual mean of the treatment variable.

`data` (`data.table`)
: Data object.

`data_model` (`data.table`)
: Internal data object that implements the causal panel model as specified by the user via `y_col`, `d_cols`, `x_cols`, `dbar_col`.

`n_obs` (`integer(1)`)
: The number of observations.

`n_treat` (`integer(1)`)
: The number of treatment variables.

`treat_col` (`character(1)`)
: "Active" treatment variable in the multiple-treatment case.

`x_cols` (`character()`)
The covariates.

`y_col` (`character(1)`)
The outcome variable.

`panel_id` (`character()`)
The panel identifier.

`time_id` (`character()`)
The time identifier.

`cluster_cols` (`character()`)
The cluster variable(s).

`n_cluster_vars` (`integer(1)`)
The number of cluster variables.

`approach` (`character(1)`)
A `character()` (`"fd-exact"`, `"wg-approx"` or `"cre"`) specifying the panel data technique to apply to estimate the causal model. Default is `"fd-exact"`.

`transformX` (`character(1)`)
A `character()` (`"no"`, `"minmax"` or `"poly"`) specifying the type of transformation to apply to the X data. `"no"` does not transform the covariates `X` and is recommended for tree-based learners. `"minmax"` applies the Min-Max normalization $x' = (x - x_{min})/(x_{max} - x_{min})$ to the covariates and is recommended with neural networks. `"poly"` add polynomials up to order three and interactions between all possible combinations of two and three variables; this is recommended for Lasso. Default is `"no"`.

Methods

Public methods:

- `xtdml_data$new()`
- `xtdml_data$print()`
- `xtdml_data$set_data_model()`
- `xtdml_data$clone()`

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

Usage:

```
xtdml_data$new(
  data = NULL,
  x_cols = NULL,
  y_col = NULL,
  d_cols = NULL,
  dbar_col = NULL,
  panel_id = NULL,
  time_id = NULL,
  cluster_cols = NULL,
  approach = NULL,
  transformX = NULL
)
```

Arguments:

`data` (`data.table`, `data.frame()`)
: Data object.

`x_cols` (`character()`)

`y_col` (`character(1)`)
: The outcome variable.

`d_cols` (`character(1)`)
: The treatment variable.

`dbar_col` (NULL, character()) `\cr Individual mean of the treatment variable (used for the CRE approach). I`

`panel_id` (`character()`)
: The panel identifier.

`time_id` (`character()`)
: The time identifier.

`cluster_cols` (`character()`)
: The cluster variable(s).

`approach` (`character(1)`)
: A `character()` (`"fd-exact"`, `"wg-approx"` or `"cre"`) specifying the panel data technique to apply to estimate the causal model. Default is `"fd-exact"`.

`transformX` (`character(1)`)
: A `character()` (`"no"`, `"minmax"` or `"poly"`) specifying the type of transformation to apply to the X data. `"no"` does not transform the covariates `X` and is recommended for tree-based learners. `"minmax"` applies the Min-Max normalization $x' = (x - x_{min})/(x_{max} - x_{min})$ to the covariates and is recommended with neural networks. `"poly"` add polynomials up to order three and interactions between all possible combinations of two and three variables; this is recommended for Lasso. Default is `"no"`.

Method `print()`**:** Print `xtdml_data` objects.

Usage:

```
xtdml_data$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`, `panel_id`, `time_id` and `cluster_cols` and assigns the role for the treatment variables in the multiple-treatment case.

Usage:

```
xtdml_data$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:

```
xtdml_data$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

xtdml_data_from_data_frame

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

Description

Initalization of DoubleMLData from `data.frame`.

Usage

```
xtdml_data_from_data_frame(
  df,
  x_cols = NULL,
  y_col = NULL,
  d_cols = NULL,
  panel_id = NULL,
  time_id = NULL,
  cluster_cols = NULL,
  approach = NULL,
  transformX = NULL
)
```

Arguments

`df`	(`data.frame()`) Data object.
`x_cols`	(`character()`) The covariates.
`y_col`	(`character(1)`) The outcome variable.
`d_cols`	(`character()`) The treatment variable(s).
`panel_id`	(`NULL`, `character()`) The panel identifier. Default is `NULL`.
`time_id`	(`NULL`, `character()`) The time identifier. Default is `NULL`.
`cluster_cols`	(`NULL`, `character()`) The cluster variables. Default is `panel_id`.
`approach`	(`character(1)`) A `character()` (`"fd-exact"`, `"wg-approx"`, `"cre"` or `"pooled"`) specifying the panel data technique to apply to estimate the causal model. Default is `"NULL"`.

transformX (character(1))
A character() ("no", "minmax" or "poly") specifying the type of transformation to apply to the X data. "no" does not transform the covariates X and is recommended for tree-based learners. "minmax" applies the Min-Max normalization $x' = (x - x_{min})/(x_{max} - x_{min})$ to the covariates and is recommended with neural networks. "poly" add polynomials up to order three and interactions between all possible combinations of two and three variables; this is recommended for Lasso. Default is "no".

Value

Creates a new instance of class xtdml_data.

Examples

```
# Generate simulated panel dataset from `xtdml`
data = make_plpr_data(n_obs = 500, t_per = 10, dim_x = 30, theta = 0.5, rho=0.8)

# Set up DML data environment
x_cols  = paste0("X", 1:30)

obj_xtdml_data = xtdml_data_from_data_frame(data,
                x_cols = x_cols,  y_col = "y", d_cols = "d",
                panel_id = "id",
                time_id = "time",
                cluster_cols = "id",
                approach = "fd-exact",
                transformX = "no")

obj_xtdml_data$print()
```

---

xtdml_plr *Routine to estimate partially linear panel regression models with fixed effects within double machine learning.*

Description

Routine to estimate partially linear panel regression models with fixed effects within double machine learning.

Format

R6::R6Class object inheriting from xtdml.

Details

Consider partially linear panel regression (PLR) model of form

$Y_{it} = \theta_0 D_{it} + g_0(x_{it}) + \alpha_i + U_{it}(1)$

$D_{it} = m_0(x_{it}) + \gamma_i + V_{it}(2)$

where (1) is the outcome equation and (2) is the treatment equation.

Super class

`xtdml::xtdml` -> `xtdml_plr`

Methods

Public methods:

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

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

Usage:

```
xtdml_plr$new(
  data,
  ml_l,
  ml_m,
  ml_g = NULL,
  n_folds = 5,
  n_rep = 1,
  score = "orth-PO",
  dml_procedure = "dml2",
  draw_sample_splitting = TRUE,
  apply_cross_fitting = TRUE
)
```

Arguments:

`data (xtdml_data)`
The `xtdml_data` 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))
: A character(1) ("orth-PO" or "orth-IV"). "orth-PO" is Neyman-orthogonal score with the partialling-out formula. "orth-IV" is Neyman-orthogonal score with the IV-type formula. Default is "orth-PO".

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 DML models with FE.

Usage:

```
xtdml_plr$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 varaible (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_theta` 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`.

Returns: self

Method `tune()`**:** Hyperparameter-tuning within double machine learning.

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:

```
xtdml_plr$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:

```
xtdml_plr$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other xtdml: `xtdml`

Examples

```
# An illustrative example using a regression tree (`rpart`)
library(mlr3)
library(rpart)
library(mlr3tuning)
set.seed(1234)

# Generate simulated dataset
data = make_plpr_data(n_obs = 100, t_per = 5, dim_x = 10, theta = 0.5, rho=0.8)

x_cols  = paste0("X", 1:10)

# Set up DML data environment
obj_xtdml_data = xtdml_data_from_data_frame(data,
                x_cols = x_cols,  y_col = "y", d_cols = "d",
                panel_id = "id",
                time_id = "time",
                approach = "fd-exact")

# Set up DML estimation environment
 learner = lrn("regr.rpart")
 ml_l = learner$clone()
 ml_m = learner$clone()

 obj_xtdml = xtdml_plr$new(obj_xtdml_data,
                           ml_l = ml_l, ml_m = ml_m,
                           score = "orth-PO", n_folds = 3)
# Set up a list of parameter grids
param_grid = list("ml_l" = ps(cp = p_dbl(lower = 0.01, upper = 0.02),
```

```
                             maxdepth = p_int(lower = 2, upper = 10)),
                  "ml_m" = ps(cp = p_dbl(lower = 0.01, upper = 0.02),
                             maxdepth = p_int(lower = 2, upper = 10)))

tune_settings = list(n_folds_tune = 3,
                   rsmp_tune = mlr3::rsmp("cv", folds = 3),
                   terminator = mlr3tuning::trm("evals", n_evals = 5),
                   tuner = tnr("grid_search", resolution = 10))

obj_xtdml$tune(param_set = param_grid, tune_settings = tune_settings)
 obj_xtdml$fit()
```

Index