econml.dml.LinearDML

class `econml.dml.LinearDML`(**, model_y='auto', model_t='auto', featurizer=None, treatment_featurizer=None, fit_cate_intercept=True, linear_first_stages='deprecated', discrete_outcome=False, discrete_treatment=False, categories='auto', cv=2, mc_iters=None, mc_agg='mean', random_state=None, allow_missing=False, enable_federation=False, use_ray=False, ray_remote_func_options=None*) [source]

Bases: `econml._cate_estimator.StatsModelsCateEstimatorMixin`, `econml.dml.dml.DML`

The Double ML Estimator with a low-dimensional linear final stage implemented as a statsmodel regression.

Parameters:

- **model_y** (estimator, default `'auto'`) – Determines how to fit the outcome to the features.
  - If `'auto'`, the model will be the best-fitting of a set of linear and forest models
  - Otherwise, see Model Selection for the range of supported options; if a single model is specified it should be a classifier if *discrete_outcome* is True and a regressor otherwise
- **model_t** (estimator, default `'auto'`) – Determines how to fit the treatment to the features.
  - If `'auto'`, the model will be the best-fitting of a set of linear and forest models
  - Otherwise, see Model Selection for the range of supported options; if a single model is specified it should be a classifier if *discrete_treatment* is True and a regressor otherwise
- **featurizer** (transformer, optional) – Must support fit_transform and transform. Used to create composite features in the final CATE regression. It is ignored if X is None. The final CATE will be trained on the outcome of featurizer.fit_transform(X). If featurizer=None, then CATE is trained on X.
- **treatment_featurizer** (transformer, optional) – Must support fit_transform and transform. Used to create composite treatment in the final CATE regression. The final CATE will be trained on the outcome of featurizer.fit_transform(T). If featurizer=None, then CATE is trained on T.

- **fit_cate_intercept** (*bool, default True*) – Whether the linear CATE model should have a constant term.
- **discrete_outcome** (bool, default `False`) – Whether the outcome should be treated as binary
- **discrete_treatment** (bool, default `False`) – Whether the treatment values should be treated as categorical, rather than continuous, quantities
- **categories** (*'auto' or list, default 'auto'*) – The categories to use when encoding discrete treatments (or 'auto' to use the unique sorted values). The first category will be treated as the control treatment.
- **cv** (*int, cross-validation generator or an iterable, default 2*) – Determines the cross-validation splitting strategy. Possible inputs for cv are:

  - None, to use the default 3-fold cross-validation,
  - integer, to specify the number of folds.
  - CV splitter
  - An iterable yielding (train, test) splits as arrays of indices.

  For integer/None inputs, if the treatment is discrete `StratifiedKFold` is used, else, `KFold` is used (with a random shuffle in either case).

  Unless an iterable is used, we call *split(X,T)* to generate the splits.

- **mc_iters** (*int, optional*) – The number of times to rerun the first stage models to reduce the variance of the nuisances.
- **mc_agg** (*{'mean', 'median'}, default 'mean'*) – How to aggregate the nuisance value for each sample across the *mc_iters* monte carlo iterations of cross-fitting.
- **random_state** (*int, RandomState instance, or None, default None*) – If int, random_state is the seed used by the random number generator; If `RandomState` instance, random_state is the random number generator; If None, the random number generator is the `RandomState` instance used by `np.random`.
- **allow_missing** (*bool*) – Whether to allow missing values in W. If True, will need to supply model_y, model_t that can handle missing values.
- **enable_federation** (*bool, default False*) – Whether to enable federation for the final model. This has a memory cost so should be enabled only if this model will be aggregated with other models.
- **use_ray** (*bool, default False*) – Whether to use Ray to parallelize the cross-fitting step. If True, Ray must be installed.
- **ray_remote_func_options** (*dict, default None*) – Options to pass to the remote function when using Ray. See https://docs.ray.io/en/latest/ray-core/api/doc/ray.remote.html

Examples

A simple example with the default models and discrete treatment:

```
from econml.dml import LinearDML

np.random.seed(123)
X = np.random.normal(size=(1000, 5))
T = np.random.binomial(1, scipy.special.expit(X[:, 0]))
y = (1 + .5*X[:, 0]) * T + X[:, 0] + np.random.normal(size=(1000,))
est = LinearDML(discrete_treatment=True)
est.fit(y, T, X=X, W=None)
```

```
>>> est.effect(X[:3])
array([0.49977..., 1.91668..., 0.70799...])
>>> est.effect_interval(X[:3])
(array([0.15122..., 1.40176..., 0.40954...]),
array([0.84831..., 2.43159..., 1.00644...]))
>>> est.coef_
array([ 0.48825...,  0.00105...,  0.00244...,  0.02217..., -0.08471...])
>>> est.coef__interval()
(array([ 0.30469..., -0.13904..., -0.12790..., -0.11514..., -0.22505... ]),
array([0.67180..., 0.14116..., 0.13278..., 0.15949..., 0.05562...]))
>>> est.intercept_
1.01247...
>>> est.intercept__interval()
(0.87480..., 1.15015...)
```

__init__(**, model_y='auto', model_t='auto', featurizer=None, treatment_featurizer=None, fit_cate_intercept=True, linear_first_stages='deprecated', discrete_outcome=False, discrete_treatment=False, categories='auto', cv=2, mc_iters=None, mc_agg='mean', random_state=None, allow_missing=False, enable_federation=False, use_ray=False, ray_remote_func_options=None*) [source]

Methods

`__init__`(*[, model_y, model_t, featurizer, ...])	
`ate`([X, T0, T1])	Calculate the average treatment effect $E_X[\tau(X, T0$
`ate_inference`([X, T0, T1])	Inference results for the quantity $E_X[\tau(X, T0, T1)]$
`ate_interval`([X, T0, T1, alpha])	Confidence intervals for the quantity $E_X[\tau(X, T0,$
`cate_feature_names`([feature_names])	Get the output feature names.
`cate_output_names`([output_names])	Public interface for getting output names.
`cate_treatment_names`([treatment_names])	Get treatment names.
`coef__inference`()	The inference of coefficients in the linear model of th

`coef__interval`(*[, alpha])	The coefficients in the linear model of the constant m
`const_marginal_ate`([X])	Calculate the average constant marginal CATE $E_X[$
`const_marginal_ate_inference`([X])	Inference results for the quantities $E_X[\theta(X)]$ produ
`const_marginal_ate_interval`([X, alpha])	Confidence intervals for the quantities $E_X[\theta(X)]$ pr
`const_marginal_effect`([X])	Calculate the constant marginal CATE $\theta(\cdot)$.
`const_marginal_effect_inference`([X])	Inference results for the quantities $\theta(X)$ produced b
`const_marginal_effect_interval`([X, alpha])	Confidence intervals for the quantities $\theta(X)$ produce
`effect`([X, T0, T1])	Calculate the heterogeneous treatment effect $\tau(X,$
`effect_inference`([X, T0, T1])	Inference results for the quantities $\tau(X, T0, T1)$ pr
`effect_interval`([X, T0, T1, alpha])	Confidence intervals for the quantities $\tau(X, T0, T1$
`fit`(Y, T, *[, X, W, sample_weight, ...])	Estimate the counterfactual model from data, i.e. est
`intercept__inference`()	The inference of intercept in the linear model of the c
`intercept__interval`(*[, alpha])	The intercept in the linear model of the constant mar
`marginal_ate`(T[, X])	Calculate the average marginal effect $E_{T,X}[\partial\tau(T,$
`marginal_ate_inference`(T[, X])	Inference results for the quantities $E_{T,X}[\partial\tau(T, X)]$
`marginal_ate_interval`(T[, X, alpha])	Confidence intervals for the quantities $E_{T,X}[\partial\tau(T,$
`marginal_effect`(T[, X])	Calculate the heterogeneous marginal effect $\partial\tau(T,$
`marginal_effect_inference`(T[, X])	Inference results for the quantities $\partial\tau(T, X)$ produc
`marginal_effect_interval`(T[, X, alpha])	Confidence intervals for the quantities $\partial\tau(T, X)$ pro
`refit_final`(*[, inference])	Estimate the counterfactual model using a new final
`score`(Y, T[, X, W, sample_weight])	Score the fitted CATE model on a new data set.
`shap_values`(X, *[, feature_names, ...])	Shap value for the final stage models (const_margin
`summary`([alpha, value, decimals, ...])	The summary of coefficient and intercept in the linea

Attributes

`bias_part_of_coef`	
`coef_`	The coefficients in the linear model of the constant marginal treatment e
`dowhy`	Get an instance of `DoWhyWrapper` to allow other functionalities from dowh

`featurizer`	
`featurizer_`	
`fit_cate_intercept_`	
`intercept_`	The intercept in the linear model of the constant marginal treatment effe
`model_cate`	Get the fitted final CATE model.
`model_final`	
`model_final_`	
`models_nuisance_`	
`models_t`	Get the fitted models for E[T | X, W].
`models_y`	Get the fitted models for E[Y | X, W].
`nuisance_scores_t`	
`nuisance_scores_y`	
`original_featurizer`	
`ortho_learner_model_final_`	
`residuals_`	A tuple (y_res, T_res, X, W), of the residuals from the first stage estimat
`rlearner_model_final_`	
`transformer`	

ate(*X=None*, ***, *T0=0*, *T1=1*)

Calculate the average treatment effect $E_X[\tau(X, T0, T1)]$.

The effect is calculated between the two treatment points and is averaged over the population of X variables.

Parameters:
- **T0** (*(m, d_t) matrix or vector of length m*) – Base treatments for each sample
- **T1** (*(m, d_t) matrix or vector of length m*) – Target treatments for each sample
- **X** (*(m, d_x) matrix, optional*) – Features for each sample

Returns: τ – Average treatment effects on each outcome Note that when Y is a vector rather than a 2-dimensional array, the result will be a scalar

Return type: float or (d_y,) array

`ate_inference`(*X=None*, ***, *T0=0*, *T1=1*)

Inference results for the quantity $E_X[\tau(X, T0, T1)]$ produced by the model. Available only when `inference` is not `None`, when calling the fit method.

Parameters:
- **X** (*(m, d_x) matrix, optional*) – Features for each sample
- **T0** (*(m, d_t) matrix or vector of length m, default 0*) – Base treatments for each sample
- **T1** (*(m, d_t) matrix or vector of length m, default 1*) – Target treatments for each sample

Returns: **PopulationSummaryResults** – The inference results instance contains prediction and prediction standard error and can on demand calculate confidence interval, z statistic and p value. It can also output a dataframe summary of these inference results.

Return type: object

`ate_interval`(*X=None*, ***, *T0=0*, *T1=1*, *alpha=0.05*)

Confidence intervals for the quantity $E_X[\tau(X, T0, T1)]$ produced by the model. Available only when `inference` is not `None`, when calling the fit method.

Parameters:
- **X** (*(m, d_x) matrix, optional*) – Features for each sample
- **T0** (*(m, d_t) matrix or vector of length m, default 0*) – Base treatments for each sample
- **T1** (*(m, d_t) matrix or vector of length m, default 1*) – Target treatments for each sample
- **alpha** (*float in [0, 1], default 0.05*) – The overall level of confidence of the reported interval. The alpha/2, 1-alpha/2 confidence interval is reported.

Returns: **lower, upper** – The lower and the upper bounds of the confidence interval for each quantity.

Return type: tuple(type of `ate(X, T0, T1)`, type of `ate(X, T0, T1))`)

`cate_feature_names`(*feature_names=None*)

Get the output feature names.

Parameters: **feature_names** (*list of str of length X.shape[1] or None*) – The names of the input features. If None and X is a dataframe, it defaults to the column names from the dataframe.

Returns: **out_feature_names** – The names of the output features $\phi(X)$, i.e. the features with respect to which the final constant marginal CATE model is linear. It is the names of the features that are associated with each entry of the `coef_()` parameter. Not available when the featurizer is not None and does not have a method: *get_feature_names(feature_names)*. Otherwise None is returned.

Return type: list of str or None

`cate_output_names`(*output_names=None*)

Public interface for getting output names.

To be overriden by estimators that apply transformations the outputs.

Parameters: **output_names** (*list of str of length Y.shape[1] or None*) – The names of the outcomes. If None and the Y passed to fit was a dataframe, it defaults to the column names from the dataframe.

Returns: **output_names** – Returns output names.

Return type: list of str

`cate_treatment_names`(*treatment_names=None*)

Get treatment names.

If the treatment is discrete or featurized, it will return expanded treatment names.

Parameters: **treatment_names** (*list of str of length T.shape[1], optional*) – The names of the treatments. If None and the T passed to fit was a dataframe, it defaults to the column names from the dataframe.

Returns: **out_treatment_names** – Returns (possibly expanded) treatment names.

Return type: list of str

`coef__inference`()

The inference of coefficients in the linear model of the constant marginal treatment effect.

Returns: **InferenceResults** – The inference of the coefficients in the final linear model

Return type: object

coef__interval(**, alpha=0.05*)

The coefficients in the linear model of the constant marginal treatment effect.

Parameters: **alpha** (*float in [0, 1], default 0.05*) – The overall level of confidence of the reported interval. The alpha/2, 1-alpha/2 confidence interval is reported.

Returns: **lb, ub** – The lower and upper bounds of the confidence interval for each quantity.

Return type: tuple(type of `coef_()`, type of `coef_()`)

const_marginal_ate(*X=None*)

Calculate the average constant marginal CATE $E_X[\theta(X)]$.

Parameters: **X** (*(m, d_x) matrix, optional*) – Features for each sample.

Returns: **theta** – Average constant marginal CATE of each treatment on each outcome. Note that when Y or featurized-T (or T if treatment_featurizer is None) is a vector rather than a 2-dimensional array, the corresponding singleton dimensions in the output will be collapsed (e.g. if both are vectors, then the output of this method will also be a scalar)

Return type: (d_y, d_f_t) matrix where d_f_t is the dimension of the featurized treatment. If treatment_featurizer is None, d_f_t = d_t.

const_marginal_ate_inference(*X=None*)

Inference results for the quantities $E_X[\theta(X)]$ produced by the model. Available only when `inference` is not `None`, when calling the fit method.

Parameters: **X** (*(m, d_x) matrix, optional*) – Features for each sample

Returns: **PopulationSummaryResults** – The inference results instance contains prediction and prediction standard error and can on demand calculate confidence interval, z statistic and p value. It can also output a dataframe summary of these inference results.

Return type: object

const_marginal_ate_interval(*X=None, *, alpha=0.05*)

Confidence intervals for the quantities $E_X[\theta(X)]$ produced by the model. Available only when `inference` is not `None`, when calling the fit method.

Parameters:
- **X** (*(m, d_x) matrix, optional*) – Features for each sample
- **alpha** (*float in [0, 1], default 0.05*) – The overall level of confidence of the reported interval. The alpha/2, 1-alpha/2 confidence interval is reported.

Returns: **lower, upper** – The lower and the upper bounds of the confidence interval for each quantity.

Return type: tuple(type of `const_marginal_ate(X)` , type of `const_marginal_ate(X)`)

`const_marginal_effect`(*X=None*)

Calculate the constant marginal CATE $\theta(\cdot)$.

The marginal effect is conditional on a vector of features on a set of m test samples X[i].

Parameters: **X** (*(m, d_x) matrix, optional*) – Features for each sample.

Returns: **theta** – Constant marginal CATE of each featurized treatment on each outcome for each sample X[i]. Note that when Y or featurized-T (or T if treatment_featurizer is None) is a vector rather than a 2-dimensional array, the corresponding singleton dimensions in the output will be collapsed (e.g. if both are vectors, then the output of this method will also be a vector)

Return type: (m, d_y, d_f_t) matrix or (d_y, d_f_t) matrix if X is None where d_f_t is the dimension of the featurized treatment. If treatment_featurizer is None, d_f_t = d_t.

`const_marginal_effect_inference`(*X=None*)

Inference results for the quantities $\theta(X)$ produced by the model. Available only when `inference` is not `None`, when calling the fit method.

Parameters: **X** (*(m, d_x) matrix, optional*) – Features for each sample

Returns: **InferenceResults** – The inference results instance contains prediction and prediction standard error and can on demand calculate confidence interval, z statistic and p value. It can also output a dataframe summary of these inference results.

Return type: object

`const_marginal_effect_interval`(*X=None*, ***, *alpha=0.05*)

Confidence intervals for the quantities $\theta(X)$ produced by the model. Available only when `inference` is not `None`, when calling the fit method.

Parameters:
- **X** (*(m, d_x) matrix, optional*) – Features for each sample
- **alpha** (*float in [0, 1], default 0.05*) – The overall level of confidence of the reported interval. The alpha/2, 1-alpha/2 confidence interval is reported.

Returns: **lower, upper** – The lower and the upper bounds of the confidence interval for each quantity.

Return type: tuple(type of `const_marginal_effect(X)` , type of `const_marginal_effect(X)`)

`effect`(*X=None*, ***, *T0=0*, *T1=1*)

Calculate the heterogeneous treatment effect $\tau(X, T0, T1)$.

The effect is calculated between the two treatment points conditional on a vector of features on a set of m test samples $\{T0_i, T1_i, X_i\}$.

Parameters:
- **T0** (*(m, d_t) matrix or vector of length m*) – Base treatments for each sample
- **T1** (*(m, d_t) matrix or vector of length m*) – Target treatments for each sample
- **X** (*(m, d_x) matrix, optional*) – Features for each sample

Returns: **τ** – Heterogeneous treatment effects on each outcome for each sample Note that when Y is a vector rather than a 2-dimensional array, the corresponding singleton dimension will be collapsed (so this method will return a vector)

Return type: (m, d_y) matrix

`effect_inference`(*X=None*, ***, *T0=0*, *T1=1*)

Inference results for the quantities $\tau(X, T0, T1)$ produced by the model. Available only when `inference` is not `None`, when calling the fit method.

Parameters:
- **X** (*(m, d_x) matrix, optional*) – Features for each sample
- **T0** (*(m, d_t) matrix or vector of length m, default 0*) – Base treatments for each sample
- **T1** (*(m, d_t) matrix or vector of length m, default 1*) – Target treatments for each sample

Returns: **InferenceResults** – The inference results instance contains prediction and prediction standard error and can on demand calculate confidence interval, z statistic and p value. It can also output a dataframe summary of these inference results.

Return type: object

`effect_interval`(*X=None*, ***, *T0=0*, *T1=1*, *alpha=0.05*)

Confidence intervals for the quantities $\tau(X, T0, T1)$ produced by the model. Available only when `inference` is not `None`, when calling the fit method.

Parameters:
- **X** (*(m, d_x) matrix, optional*) – Features for each sample
- **T0** (*(m, d_t) matrix or vector of length m, default 0*) – Base treatments for each sample
- **T1** (*(m, d_t) matrix or vector of length m, default 1*) – Target treatments for each sample
- **alpha** (*float in [0, 1], default 0.05*) – The overall level of confidence of the reported interval. The alpha/2, 1-alpha/2 confidence interval is reported.

Returns: **lower, upper** – The lower and the upper bounds of the confidence interval for each quantity.

Return type: tuple(type of `effect(X, T0, T1)`, type of `effect(X, T0, T1))`)

`fit`(*Y*, *T*, ***, *X=None*, *W=None*, *sample_weight=None*, *freq_weight=None*, *sample_var=None*, *groups=None*, *cache_values=False*, *inference='auto'*) [source]

Estimate the counterfactual model from data, i.e. estimates functions $\tau(\cdot,\cdot,\cdot)$, $\partial\tau(\cdot,\cdot)$.

Parameters:
- **Y** (*(n × d_y) matrix or vector of length n*) – Outcomes for each sample
- **T** (*(n × d_t) matrix or vector of length n*) – Treatments for each sample
- **X** (*(n × d_x) matrix, optional*) – Features for each sample
- **W** (*(n × d_w) matrix, optional*) – Controls for each sample
- **sample_weight** (*(n,) array_like, optional*) – Individual weights for each sample. If None, it assumes equal weight.
- **freq_weight** (*(n,) array_like of int, optional*) – Weight for the observation. Observation i is treated as the mean outcome of freq_weight[i] independent observations. When `sample_var` is not None, this should be provided.

- **sample_var** (*{(n,), (n, d_y)} nd array_like, optional*) – Variance of the outcome(s) of the original freq_weight[i] observations that were used to compute the mean outcome represented by observation i.
- **groups** (*(n,) vector, optional*) – All rows corresponding to the same group will be kept together during splitting. If groups is not None, the *cv* argument passed to this class's initializer must support a 'groups' argument to its split method.
- **cache_values** (*bool, default False*) – Whether to cache inputs and first stage results, which will allow refitting a different final model
- **inference** (str, `Inference` instance, or None) – Method for performing inference. This estimator supports 'bootstrap' (or an instance of `BootstrapInference`) and 'statsmodels' (or an instance of `StatsModelsInference`)

Return type: self

`intercept__inference`()

The inference of intercept in the linear model of the constant marginal treatment effect.

Returns: **InferenceResults** – The inference of the intercept in the final linear model

Return type: object

`intercept__interval`(**, alpha=0.05*)

The intercept in the linear model of the constant marginal treatment effect.

Parameters: **alpha** (*float in [0, 1], default 0.05*) – The overall level of confidence of the reported interval. The alpha/2, 1-alpha/2 confidence interval is reported.

Returns: **lower, upper** – The lower and upper bounds of the confidence interval.

Return type: tuple(type of `intercept_()`, type of `intercept_()`)

`marginal_ate`(*T, X=None*)

Calculate the average marginal effect $E_{T,X}[\partial\tau(T,X)]$.

The marginal effect is calculated around a base treatment point and averaged over the population of X.

Parameters:
- **T** (*(m, d_t) matrix*) – Base treatments for each sample

- **X** (*(m, d_x) matrix, optional*) – Features for each sample

Returns: **grad_tau** – Average marginal effects on each outcome Note that when Y or T is a vector rather than a 2-dimensional array, the corresponding singleton dimensions in the output will be collapsed (e.g. if both are vectors, then the output of this method will be a scalar)

Return type: (d_y, d_t) array

`marginal_ate_inference`(*T*, *X=None*)

Inference results for the quantities $E_{T,X}[\partial\tau(T, X)]$ produced by the model. Available only when `inference` is not `None`, when calling the fit method.

Parameters:
- **T** (*(m, d_t) matrix*) – Base treatments for each sample
- **X** (*(m, d_x) matrix, optional*) – Features for each sample

Returns: **PopulationSummaryResults** – The inference results instance contains prediction and prediction standard error and can on demand calculate confidence interval, z statistic and p value. It can also output a dataframe summary of these inference results.

Return type: object

`marginal_ate_interval`(*T*, *X=None*, ***, *alpha=0.05*)

Confidence intervals for the quantities $E_{T,X}[\partial\tau(T, X)]$ produced by the model. Available only when `inference` is not `None`, when calling the fit method.

Parameters:
- **T** (*(m, d_t) matrix*) – Base treatments for each sample
- **X** (*(m, d_x) matrix, optional*) – Features for each sample
- **alpha** (*float in [0, 1], default 0.05*) – The overall level of confidence of the reported interval. The alpha/2, 1-alpha/2 confidence interval is reported.

Returns: **lower, upper** – The lower and the upper bounds of the confidence interval for each quantity.

Return type: tuple(type of `marginal_ate(T, X)`, type of `marginal_ate(T, X)`)

`marginal_effect`(*T*, *X=None*)

Calculate the heterogeneous marginal effect $\partial\tau(T, X)$.

The marginal effect is calculated around a base treatment point conditional on a vector of features on a set of m test samples $\{T_i, X_i\}$. If treatment_featurizer is None, the base treatment is ignored in this calculation and the result is equivalent to const_marginal_effect.

Parameters:
- **T** (*(m, d_t) matrix*) – Base treatments for each sample
- **X** (*(m, d_x) matrix, optional*) – Features for each sample

Returns: **grad_tau** – Heterogeneous marginal effects on each outcome for each sample Note that when Y or T is a vector rather than a 2-dimensional array, the corresponding singleton dimensions in the output will be collapsed (e.g. if both are vectors, then the output of this method will also be a vector)

Return type: (m, d_y, d_t) array

`marginal_effect_inference`(*T*, *X=None*)

Inference results for the quantities $\partial\tau(T, X)$ produced by the model. Available only when `inference` is not `None`, when calling the fit method.

Parameters:
- **T** (*(m, d_t) matrix*) – Base treatments for each sample
- **X** (*(m, d_x) matrix, optional*) – Features for each sample

Returns: **InferenceResults** – The inference results instance contains prediction and prediction standard error and can on demand calculate confidence interval, z statistic and p value. It can also output a dataframe summary of these inference results.

Return type: object

`marginal_effect_interval`(*T*, *X=None*, ***, *alpha=0.05*)

Confidence intervals for the quantities $\partial\tau(T, X)$ produced by the model. Available only when `inference` is not `None`, when calling the fit method.

Parameters:
- **T** (*(m, d_t) matrix*) – Base treatments for each sample
- **X** (*(m, d_x) matrix, optional*) – Features for each sample
- **alpha** (*float in [0, 1], default 0.05*) – The overall level of confidence of the reported interval. The alpha/2, 1-alpha/2 confidence interval is reported.

Returns: **lower, upper** – The lower and the upper bounds of the confidence interval for each quantity.

Return type: tuple(type of `marginal_effect(T, X)`, type of `marginal_effect(T, X)`) `refit_final`(**, inference='auto'*)

Estimate the counterfactual model using a new final model specification but with cached first stage results.

In order for this to succeed, `fit` must have been called with `cache_values=True`. This call will only refit the final model. This call we use the current setting of any parameters that change the final stage estimation. If any parameters that change how the first stage nuisance estimates has also been changed then it will have no effect. You need to call fit again to change the first stage estimation results.

Parameters: **inference** (*inference method, optional*) – The string or object that represents the inference method

Returns: **self** – This instance

Return type: object

`score`(*Y, T, X=None, W=None, sample_weight=None*)

Score the fitted CATE model on a new data set. Generates nuisance parameters for the new data set based on the fitted residual nuisance models created at fit time. It uses the mean prediction of the models fitted by the different crossfit folds. Then calculates the MSE of the final residual Y on residual T regression.

If model_final does not have a score method, then it raises an `AttributeError`

Parameters:

- **Y** (*(n, d_y) matrix or vector of length n*) – Outcomes for each sample
- **T** (*(n, d_t) matrix or vector of length n*) – Treatments for each sample
- **X** (*(n, d_x) matrix, optional*) – Features for each sample
- **W** (*(n, d_w) matrix, optional*) – Controls for each sample
- **sample_weight** (*(n,) vector, optional*) – Weights for each samples

Returns: **score** – The MSE of the final CATE model on the new data.

Return type: float

`shap_values`(*X, *, feature_names=None, treatment_names=None, output_names=None, background_samples=100*)

Shap value for the final stage models (const_marginal_effect)

Parameters:

- **X** (*(m, d_x) matrix*) – Features for each sample. Should be in the same shape of fitted X in final stage.
- **feature_names** (*list of str of length X.shape[1], optional*) – The names of input features.

- **treatment_names** (*list, optional*) – The name of featurized treatment. In discrete treatment scenario, the name should not include the name of the baseline treatment (i.e. the control treatment, which by default is the alphabetically smaller)
- **output_names** (*list, optional*) – The name of the outcome.
- **background_samples** (*int , default 100*) – How many samples to use to compute the baseline effect. If None then all samples are used.

Returns: **shap_outs** – A nested dictionary by using each output name (e.g. 'Y0', 'Y1', … when *output_names=None*) and each treatment name (e.g. 'T0', 'T1', … when *treatment_names=None*) as key and the shap_values explanation object as value. If the input data at fit time also contain metadata, (e.g. are pandas DataFrames), then the column metatdata for the treatments, outcomes and features are used instead of the above defaults (unless the user overrides with explicitly passing the corresponding names).

Return type: nested dictionary of Explanation object

`summary`(*alpha=0.05*, *value=0*, *decimals=3*, *feature_names=None*, *treatment_names=None*, *output_names=None*)

The summary of coefficient and intercept in the linear model of the constant marginal treatment effect.

Parameters:

- **alpha** (*float in [0, 1], default 0.05*) – The overall level of confidence of the reported interval. The alpha/2, 1-alpha/2 confidence interval is reported.
- **value** (*float, default 0*) – The mean value of the metric you'd like to test under null hypothesis.
- **decimals** (*int, default 3*) – Number of decimal places to round each column to.
- **feature_names** (*list of str, optional*) – The input of the feature names
- **treatment_names** (*list of str, optional*) – The names of the treatments
- **output_names** (*list of str, optional*) – The names of the outputs

Returns: **smry** – this holds the summary tables and text, which can be printed or converted to various output formats.

Return type: Summary instance

property `coef_`

The coefficients in the linear model of the constant marginal treatment effect.

Returns: **coef** – Where n_x is the number of features that enter the final model (either the dimension of X or the dimension of featurizer.fit_transform(X) if the CATE estimator has a featurizer.), n_t is the number of treatments, n_y is the number of outcomes. Dimensions are omitted if the original input was a vector and not a 2D array. For binary treatment the n_t dimension is also omitted.

Return type: (n_x,) or (n_t, n_x) or (n_y, n_t, n_x) array_like

property `dowhy`

Get an instance of `DoWhyWrapper` to allow other functionalities from dowhy package. (e.g. causal graph, refutation test, etc.)

Returns: **DoWhyWrapper** – An instance of `DoWhyWrapper`

Return type: instance

property `intercept_`

The intercept in the linear model of the constant marginal treatment effect.

Returns: **intercept** – Where n_t is the number of treatments, n_y is the number of outcomes. Dimensions are omitted if the original input was a vector and not a 2D array. For binary treatment the n_t dimension is also omitted.

Return type: float or (n_y,) or (n_y, n_t) array_like

property `model_cate`

Get the fitted final CATE model.

Returns: **model_cate** – An instance of the model_final object that was fitted after calling fit which corresponds to the constant marginal CATE model.

Return type: object of type(model_final) *property* `models_t`

Get the fitted models for E[T | X, W].

Returns: **models_t** – A nested list of instances of the *model_y* object. Number of sublist equals to number of monte carlo iterations, each element in the sublist corresponds to a crossfitting fold and is the model instance that was fitted for that training fold.

Return type: nested list of objects of type(*model_t*)

property `models_y`

Get the fitted models for E[Y | X, W].

Returns: **models_y** – A nested list of instances of the *model_y* object. Number of sublist equals to number of monte carlo iterations, each element in the sublist corresponds to a crossfitting fold and is the model instance that was fitted for that training fold.

Return type: nested list of objects of type(*model_y*)

property `residuals_`

A tuple (y_res, T_res, X, W), of the residuals from the first stage estimation along with the associated X and W. Samples are not guaranteed to be in the same order as the input order.