causalml Documentation

Someone at Uber

Oct 23, 2020

Contents

1 About Causal ML **3**

2 Methodology **5**
2.1 Meta-Learner Algorithms 5
2.2 Tree-Based Algorithms 7

3 Installation **9**

4 Examples **11**
4.1 Propensity Score Estimation 11
4.2 Propensity Score Matching 11
4.3 Average Treatment Effect (ATE) Estimation 12
4.4 Synthetic Data Generation Process 12

5 Interpretable Causal ML **15**
5.1 Meta-Learner Feature Importances 15
5.2 Uplift Tree Visualization 17

6 Validation **19**
6.1 Validation with Multiple Estimates 19
6.2 Validation with Synthetic Data Sets 20

7 Visualization **23**
7.1 Supported Models 23
7.2 Supported Usage 23
7.3 How to Read the Plot 24

8 causalml package **25**
8.1 Submodules 25
8.2 causalml.inference.tree module 25
8.3 causalml.inference.meta module 25
8.4 causalml.optimize module 44
8.5 causalml.dataset module 44
8.6 causalml.match module 44
8.7 causalml.propensity module 46
8.8 causalml.metrics module 49
8.9 Module contents 62

9 References **63**
9.1 Open Source Software Projects . . . 63
9.2 Papers . . . 63

10 Changelog **65**
10.1 0.8.0 (2020-07-17) . . . 65
10.2 0.7.1 (2020-05-07) . . . 65
10.3 0.7.0 (2020-02-28) . . . 66
10.4 0.6.0 (2019-12-31) . . . 66
10.5 0.5.0 (2019-11-26) . . . 67
10.6 0.4.0 (2019-10-21) . . . 67
10.7 0.3.0 (2019-09-17) . . . 67
10.8 0.2.0 (2019-08-12) . . . 67
10.9 0.1.0 (unreleased) . . . 67

11 Indices and tables **69**

Bibliography **71**

Python Module Index **73**

Index **75**

Contents:

CHAPTER 1

About Causal ML

`Causal ML` is a Python package that provides a suite of uplift modeling and causal inference methods using machine learning algorithms based on recent research. It provides a standard interface that allows user to estimate the **Conditional Average Treatment Effect** (CATE) or **Individual Treatment Effect** (ITE) from experimental or observational data. Essentially, it estimates the causal impact of intervention **T** on outcome **Y** for users with observed features **X**, without strong assumptions on the model form.

Typical use cases include:

- **Campaign Targeting Optimization**: An important lever to increase ROI in an advertising campaign is to target the ad to the set of customers who will have a favorable response in a given KPI such as engagement or sales. CATE identifies these customers by estimating the effect of the KPI from ad exposure at the individual level from A/B experiment or historical observational data.
- **Personalized Engagement**: Company has multiple options to interact with its customers such as different product choices in up-sell or mess aging channels for communications. One can use CATE to estimate the heterogeneous treatment effect for each customer and treatment option combination for an optimal personalized recommendation system.

The package currently supports the following methods:

- **Tree-based algorithms**
  - Uplift Random Forests on KL divergence, Euclidean Distance, and Chi-Square
  - Uplift Random Forests on Contextual Treatment Selection
- **Meta-learner algorithms**
  - *S-Learner*
  - *T-Learner*
  - *X-Learner*
  - *R-Learner*
- **Instrumental variables algorithm**
  - 2-Stage Least Squares (2SLS)

CHAPTER 2

Methodology

2.1 Meta-Learner Algorithms

A meta-algorithm (or meta-learner) is a framework to estimate the Conditional Average Treatment Effect (CATE) using any machine learning estimators (called base learners) [8].

A meta-algorithm uses either a single base learner while having the treatment indicator as a feature (e.g. S-learner), or multiple base learners separately for each of the treatment and control groups (e.g. T-learner, X-learner and R-learner).

Confidence intervals of average treatment effect estimates are calculated based on the lower bound formular (7) from [7].

2.1.1 S-Learner

S-learner estimates the treatment effect using a single machine learning model as follows:

Stage 1
Estimate the average outcomes $\mu(x)$ with covariates X and an indicator variable for treatment effect W:

$$\mu(x) = E[Y|X = x, W = w]$$

using a machine learning model.

Stage 2
Define the CATE estimate as:

$$\hat{\tau}(x) = \hat{\mu}(x, W = 1) - \hat{\mu}(x, W = 0)$$

Including the propensity score in the model can reduce bias from regularization induced confounding [13].

When the control and treatment groups are very different in covariates, a single linear model is not sufficient to encode the different relevant dimensions and smoothness of features for the control and treatment groups [1].

2.1.2 T-Learner

T-learner [8] consists of two stages as follows:

Stage 1
Estimate the average outcomes $\mu_0(x)$ and $\mu_1(x)$:

$\mu_0(x) = E[Y(0)|X = x]$ and
$\mu_1(x) = E[Y(1)|X = x]$

using machine learning models.

Stage 2
Define the CATE estimate as:

$$\hat{\tau}(x) = \hat{\mu}_1(x) - \hat{\mu}_0(x)$$

2.1.3 X-Learner

X-learner [8] is an extension of T-learner, and consists of three stages as follows:

Stage 1
Estimate the average outcomes $\mu_0(x)$ and $\mu_1(x)$:

$\mu_0(x) = E[Y(0)|X = x]$ and
$\mu_1(x) = E[Y(1)|X = x]$

using machine learning models.

Stage 2
Impute the user level treatment effects, D^1_i and D^0_j for user i in the treatment group based on $\mu_0(x)$, and user j in the control groups based on $\mu_1(x)$:

$D^1_i = Y^1_i - \hat{\mu}_0(X^1_i)$, and
$D^0_i = \hat{\mu}_1(X^0_i) - Y^0_i$

then estimate $\tau_1(x) = E[D^1|X = x]$, and $\tau_0(x) = E[D^0|X = x]$ using machine learning models.

Stage 3

Define the CATE estimate by a weighted average of $\tau_1(x)$ and $\tau_0(x)$:

$$\tau(x) = g(x)\tau_0(x) + (1 - g(x))\tau_1(x)$$

where $g \in [0, 1]$. We can use propensity scores for $g(x)$.

2.1.4 R-Learner

R-learner [9] uses the cross-validation out-of-fold estimates of outcomes $\hat{m}^{(-i)}(x_i)$ and propensity scores $\hat{e}^{(-i)}(x_i)$. It consists of two stages as follows:

Stage 1
Fit $\hat{m}(x)$ and $\hat{e}(x)$ with machine learning models using cross-validation.

Stage 2
Estimate treatment effects by minimising the R-loss, $\hat{L}_n(\tau(x))$:

$$\hat{L}_n(\tau(x)) = \frac{1}{n}\sum_{i=1}^{n}\big(\big(Y_i - \hat{m}^{(-i)}(X_i)\big) - \big(W_i - \hat{e}^{(-i)}(X_i)\big)\tau(X_i)\big)^2$$

where $\hat{e}^{(-i)}(X_i)$, etc. denote the out-of-fold held-out predictions made without using the i-th training sample.

2.2 Tree-Based Algorithms

2.2.1 Uplift Tree

The Uplift Tree approach consists of a set of methods that use a tree-based algorithm where the splitting criterion is based on differences in uplift. [11] proposed three different ways to quantify the gain in divergence as the result of splitting [5]:

$$D_{gain} = D_{after_{split}}(P^T, P^C) - D_{before_{split}}(P^T, P^C)$$

where D measures the divergence and P^T and P^C refer to the probability distribution of the outcome of interest in the treatment and control groups, respectively. Three different ways to quantify the divergence, KL, ED and Chi, are implemented in the package.

2.2.2 KL

The Kullback-Leibler (KL) divergence is given by:

$$KL(P : Q) = \sum_{k=left,right} p_k log\frac{p_k}{q_k}$$

where p is the sample mean in the treatment group, q is the sample mean in the control group and k indicates the leaf in which p and q are computed [5]

2.2.3 ED

The Euclidean Distance is given by:

$ED(P:Q) = \sum_{k=left,right}(p_k - q_k)^2$

where the notation is the same as above.

2.2.4 Chi

Finally, the χ^2-divergence is given by:

$\chi^2(P:Q) = \sum_{k=left,right} \frac{(p_k - q_k)^2}{q_k}$

where the notation is again the same as above.

2.2.5 CTS

The final Uplift Tree algorithm that is implemented is the Contextual Treatment Selection (CTS) approach by [12], where the sample splitting criterion is defined as follows:

$\hat{\Delta}_\mu(s) = \hat{p}(\phi_l \mid \phi) \times \max_{t=0,...,K} \hat{y}_t(\phi_l) + \hat{p}(\phi_r \mid \phi) \times \max_{t=0,...,K} \hat{y}_t(\phi_r) - \max_{t=0,...,K} \hat{y}_t(\phi)$

where ϕ_l and ϕ_r refer to the feature subspaces in the left leaf and the right leaves respectively, $\hat{p}(\phi_j \mid \phi)$ denotes the estimated conditional probability of a subject's being in ϕ_j given ϕ, and $\hat{y}_t(\phi_j)$ is the conditional expected response under treatment t.

CHAPTER 3

Installation

`causalml` is available on PyPI, and can be installed from `pip` or source as follows:

From `pip`:

```
pip install causalml
```

From source:

```
git clone https://github.com/uber-common/causalml.git
cd causalml
python setup.py build_ext --inplace
python setup.py install
```

CHAPTER 4

Examples

Working example notebooks are available in the example folder.

4.1 Propensity Score Estimation

```
from causalml.propensity import ElasticNetPropensityModel

pm = ElasticNetPropensityModel(n_fold=5, random_state=42)
ps = pm.fit_predict(X, y)
```

4.2 Propensity Score Matching

```
from causalml.match import NearestNeighborMatch, create_table_one

psm = NearestNeighborMatch(replace=False,
                           ratio=1,
                           random_state=42)
matched = psm.match_by_group(data=df,
                             treatment_col=treatment_col,
                             score_col=score_col,
                             groupby_col=groupby_col)

create_table_one(data=matched,
                 treatment_col=treatment_col,
                 features=covariates)
```

4.3 Average Treatment Effect (ATE) Estimation

```
from causalml.inference.meta import LRSRegressor
from causalml.inference.meta import XGBTRegressor, MLPTRegressor
from causalml.inference.meta import BaseXRegressor
from causalml.inference.meta import BaseRRegressor
from xgboost import XGBRegressor
from causalml.dataset import synthetic_data

y, X, treatment, _, _, e = synthetic_data(mode=1, n=1000, p=5, sigma=1.0)

lr = LRSRegressor()
te, lb, ub = lr.estimate_ate(X, treatment, y)
print('Average Treatment Effect (Linear Regression): {:.2f} ({:.2f}, {:.2f})'.
↪format(te[0], lb[0], ub[0]))

xg = XGBTRegressor(random_state=42)
te, lb, ub = xg.estimate_ate(X, treatment, y)
print('Average Treatment Effect (XGBoost): {:.2f} ({:.2f}, {:.2f})'.format(te[0],␣
↪lb[0], ub[0]))

nn = MLPTRegressor(hidden_layer_sizes=(10, 10),
                 learning_rate_init=.1,
                 early_stopping=True,
                 random_state=42)
te, lb, ub = nn.estimate_ate(X, treatment, y)
print('Average Treatment Effect (Neural Network (MLP)): {:.2f} ({:.2f}, {:.2f})'.
↪format(te[0], lb[0], ub[0]))

xl = BaseXRegressor(learner=XGBRegressor(random_state=42))
te, lb, ub = xl.estimate_ate(X, p, treatment, y)
print('Average Treatment Effect (BaseXRegressor using XGBoost): {:.2f} ({:.2f}, {:.2f}
↪)'.format(te[0], lb[0], ub[0]))

rl = BaseRRegressor(learner=XGBRegressor(random_state=42))
te, lb, ub =  rl.estimate_ate(X=X, p=e, treatment=treatment, y=y)
print('Average Treatment Effect (BaseRRegressor using XGBoost): {:.2f} ({:.2f}, {:.2f}
↪)'.format(te[0], lb[0], ub[0]))
```

4.4 Synthetic Data Generation Process

4.4.1 Single Simulation

```
from causalml.dataset import *

# Generate synthetic data for single simulation
y, X, treatment, tau, b, e = synthetic_data(mode=1)
y, X, treatment, tau, b, e = simulate_nuisance_and_easy_treatment()

# Generate predictions for single simulation
single_sim_preds = get_synthetic_preds(simulate_nuisance_and_easy_treatment, n=1000)

# Generate multiple scatter plots to compare learner performance for a single␣
↪simulation
```

(continues on next page)

(continued from previous page)

```
scatter_plot_single_sim(single_sim_preds)

# Visualize distribution of learner predictions for a single simulation
distr_plot_single_sim(single_sim_preds, kind='kde')
```

4.4.2 Multiple Simulations

```
from causalml.dataset import *

# Generalize performance summary over k simulations
num_simulations = 12
preds_summary = get_synthetic_summary(simulate_nuisance_and_easy_treatment, n=1000,␣
↪k=num_simulations)

# Generate scatter plot of performance summary
scatter_plot_summary(preds_summay, k=num_simulations)

# Generate bar plot of performance summary
bar_plot_summary(preds_summary, k=num_simulations)
```

CHAPTER 5

Interpretable Causal ML

Causal ML provides methods to interpret the treatment effect models trained.

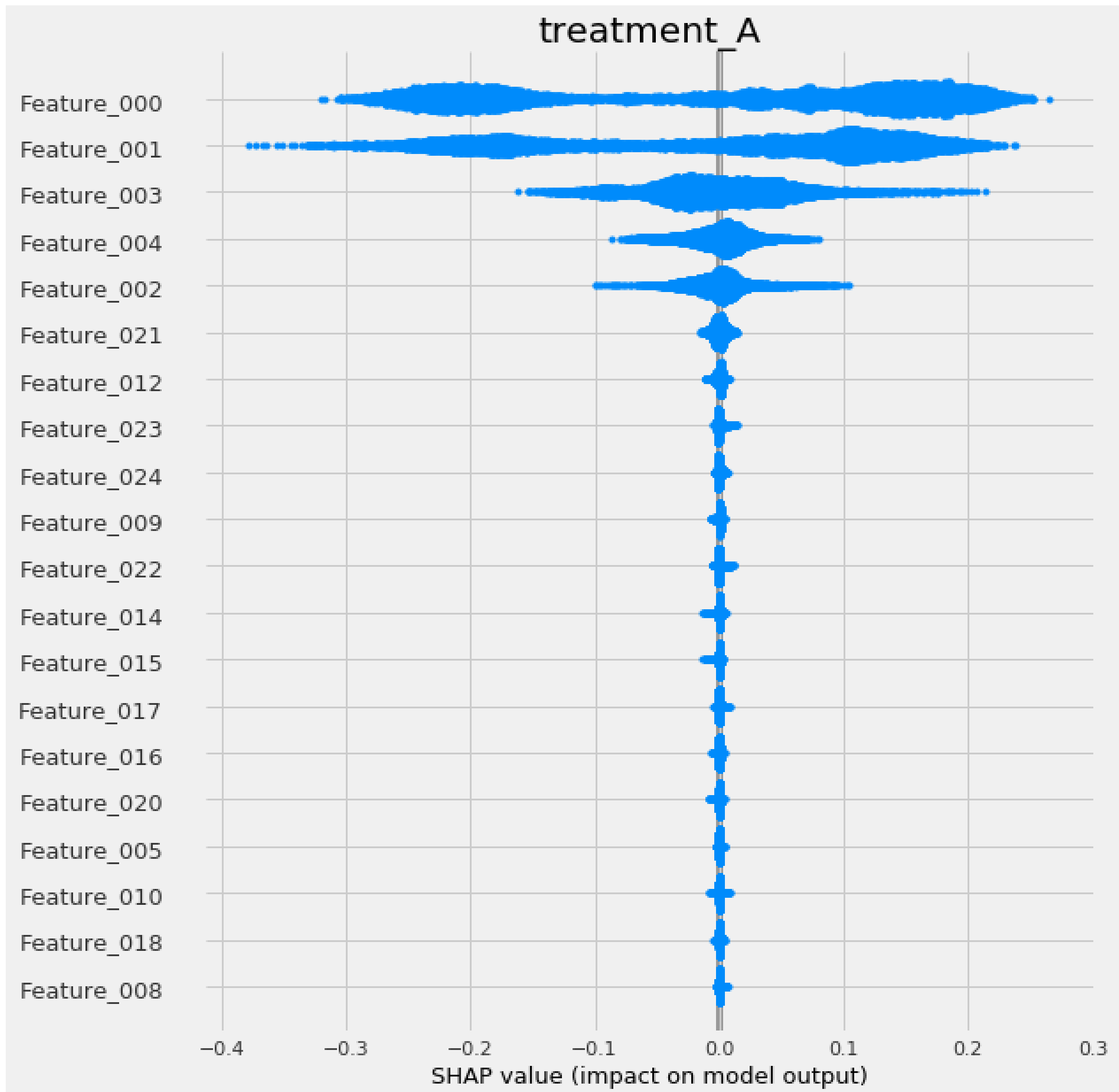
5.1 Meta-Learner Feature Importances

```
from causalml.inference.meta import BaseSRegressor, BaseTRegressor, BaseXRegressor,␣
↪BaseRRegressor

slearner = BaseSRegressor(LGBMRegressor(), control_name='control')
slearner.estimate_ate(X, w_multi, y)
slearner_tau = slearner.fit_predict(X, w_multi, y)

model_tau_feature = RandomForestRegressor()  # specify model for model_tau_feature

slearner.get_importance(X=X, tau=slearner_tau, model_tau_feature=model_tau_feature,
                        normalize=True, method='auto', features=feature_names)

# Using the feature_importances_ method in the base learner (LGBMRegressor() in this␣
↪example)
slearner.plot_importance(X=X, tau=slearner_tau, normalize=True, method='auto')

# Using eli5's PermutationImportance
slearner.plot_importance(X=X, tau=slearner_tau, normalize=True, method='permutation')

# Using SHAP
shap_slearner = slearner.get_shap_values(X=X, tau=slearner_tau)

# Plot shap values without specifying shap_dict
slearner.plot_shap_values(X=X, tau=slearner_tau)

# Plot shap values WITH specifying shap_dict
slearner.plot_shap_values(shap_dict=shap_slearner)
```

(continues on next page)

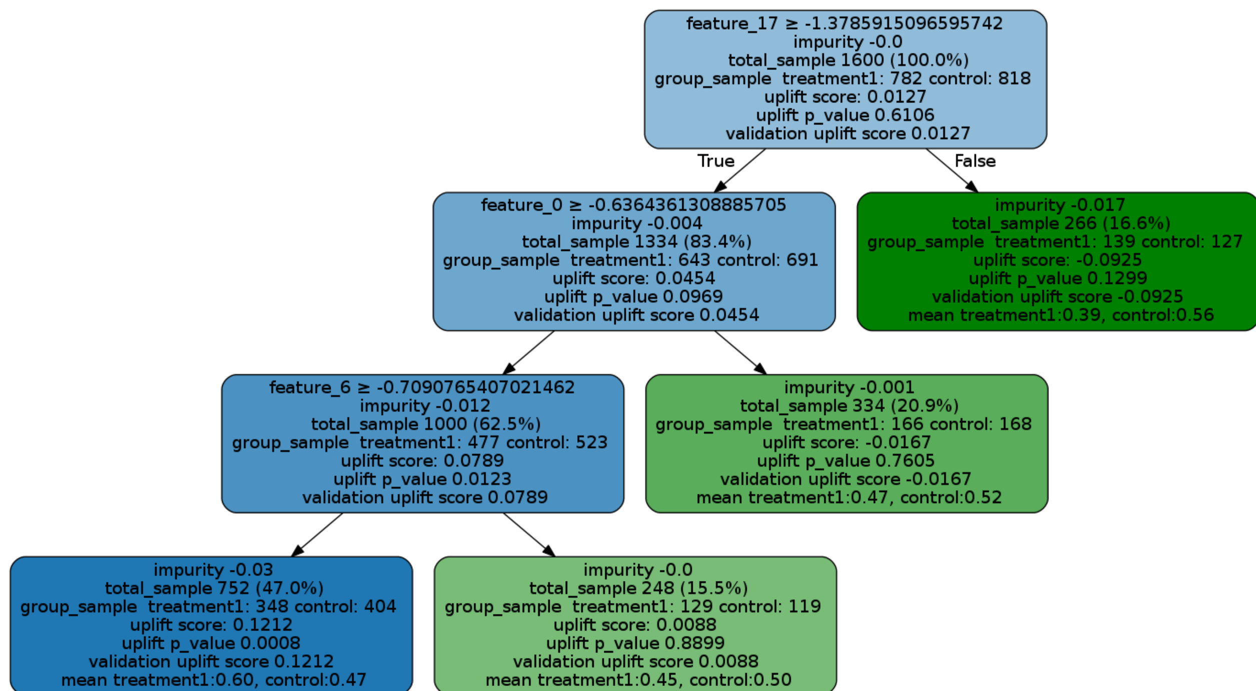
(continued from previous page)

```
# interaction_idx set to 'auto' (searches for feature with greatest approximate␣
↪interaction)
slearner.plot_shap_dependence(treatment_group='treatment_A',
                              feature_idx=1,
                              X=X,
                              tau=slearner_tau,
                              interaction_idx='auto')
```



5.2 Uplift Tree Visualization

```
from IPython.display import Image
from causalml.inference.tree import UpliftTreeClassifier, UpliftRandomForestClassifier
from causalml.inference.tree import uplift_tree_string, uplift_tree_plot

uplift_model = UpliftTreeClassifier(max_depth=5, min_samples_leaf=200, min_samples_
↪treatment=50,
                                    n_reg=100, evaluationFunction='KL', control_name=
↪'control')

uplift_model.fit(df[features].values,
                 treatment=df['treatment_group_key'].values,
                 y=df['conversion'].values)

graph = uplift_tree_plot(uplift_model.fitted_uplift_tree, features)
Image(graph.create_png())
```



CHAPTER 6

Validation

Estimation of the treatment effect cannot be validated the same way as regular ML predictions because the true value is not available except for the experimental data. Here we focus on the internal validation methods under the assumption of unconfoundedness of potential outcomes and the treatment status conditioned on the feature set available to us.

6.1 Validation with Multiple Estimates

We can validate the methodology by comparing the estimates with other approaches, checking the consistency of estimates across different levels and cohorts.

6.1.1 Model Robustness for Meta Algorithms

In meta-algorithms we can assess the quality of user-level treatment effect estimation by comparing estimates from different underlying ML algorithms. We will report MSE, coverage (overlapping 95% confidence interval), uplift curve. In addition, we can split the sample within a cohort and compare the result from out-of-sample scoring and within-sample scoring.

6.1.2 User Level/Segment Level/Cohort Level Consistency

We can also evaluate user-level/segment level/cohort level (as in CeViChE) estimation consistency by conducting T-test.

6.1.3 Stability between Cohorts

Treatment effect may vary from cohort to cohort but should not be too volatile. For a given cohort, we will compare the scores generated by model fit to another score with the ones generated by its own model.

6.2 Validation with Synthetic Data Sets

We can test the methodology with simulations, where we generate data with known causal and non-causal links between the outcome, treatment and some of confounding variables.

We implemented the following sets of synthetic data generation mechanisms based on [9]:

6.2.1 Mechanism 1

This generates a complex outcome regression model with easy treatment effect with input variables $X_i \sim Unif(0,1)^d$.
The treatment flag is a binomial variable, whose d.g.p. is:

$$P(W_i = 1|X_i) = logit(trim_{0.1}(sin(\pi X_{i1} X_{i2}))$$

The outcome variable is:

$$y_i = sin(\pi X_{i1} X_{i2}) + 2(X_{i3} - 0.5)^2 + X_{i4} + 0.5X_{i5} + (W_i - 0.5)(X_{i1} + X_{i2})/2 + \epsilon_i$$

6.2.2 Mechanism 2

This simulates a randomized trial. The input variables are generated by $X_i \sim N(0, I_{d \times d})$

The treatment flag is generated by a fair coin flip:

$$P(W_i = 1|X_i) = 0.5$$

The outcome variable is

$$y_i = max(X_{i1} + X_{i2}, X_{i3}, 0) + max(X_{i4} + X_{i5}, 0) + (W_i - 0.5)(X_{i1} + \log(1 + e^{X_{i2}}))$$

6.2.3 Mechanism 3

This one has an easy propensity score but a difficult control outcome. The input variables follow $X_i \sim N(0, I_{d \times d})$

The treatment flag is a binomial variable, whose d.g.p is:

$$P(W_i = 1|X_i) = logit(X_{i2} + X_{i3})$$

The outcome variable is:

$$y_i = 2\log(1 + e^{X_{i1} + X_{i2} + X_{i3}}) + (W_i - 0.5)$$

6.2.4 Mechanism 4

This contains an unrelated treatment arm and control arm, with input data generated by $X_i \sim N(0, I_{d\times d})$.

The treatment flag is a binomial variable whose d.g.p. is:

$$P(W_i = 1|X_i) = logit(X_{i1} + X_{i2})$$

The outcome variable is:

$$y_i = \frac{1}{2}\big(max(X_{i1}+X_{i2}+X_{i3}, 0)+max(X_{i4}+X_{i5}, 0)\big)+(W_i-0.5)(max(X_{i1}+X_{i2}+X_{i3}, 0)-max(X_{i4}, X_{i5}, 0))$$

CHAPTER 7

Visualization

Visualization functions are provided for uplift trees for model interpretation and diagnosis.

7.1 Supported Models

These visualization functions work only for tree-based classification algorithms:

- Uplift tree/random forests on KL divergence, Euclidean Distance, and Chi-Square
- Uplift tree/random forests on Contextual Treatment Selection

Currently, they are NOT supporting Meta-learner algorithms

- S-learner
- T-learner
- X-learner
- R-learner

7.2 Supported Usage

The visualization method supports both uplift tree and uplift random forest:

- Visualize a trained uplift classification tree model
- Visualize an uplift tree in a trained uplift random forests

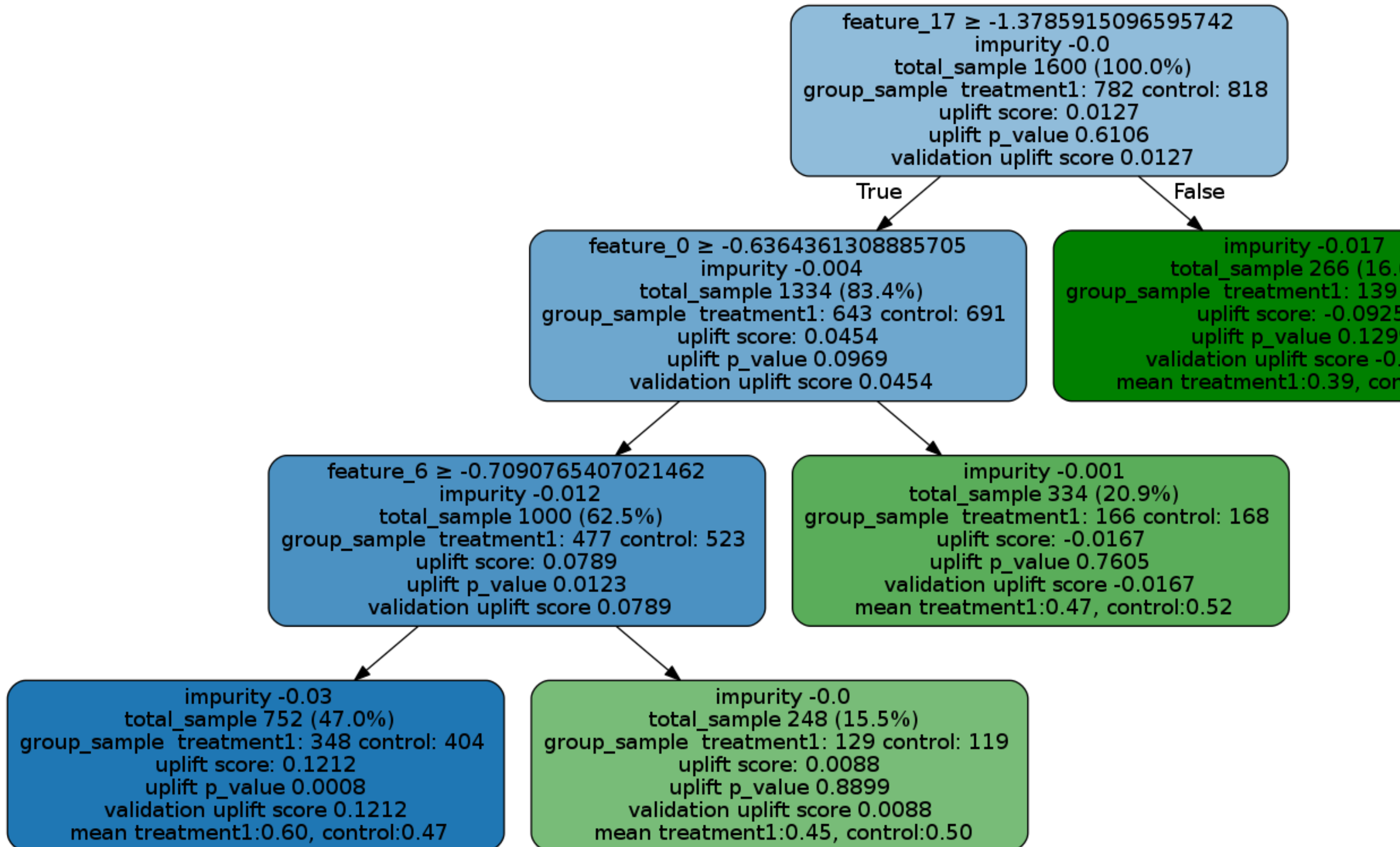
It supports both tree based on training data and tree based on testing data for validation purpose:

- Visualize the validation tree: fill the trained uplift classification tree with testing (or validation) data, and show the statistics for both training data and testing data

It supports multiple treatment groups

- Visualize the case where there are one control group and multiple treatment groups

7.3 How to Read the Plot



- feature_name > threshold: For non-leaf node, the first line is an inequality indicating the splitting rule of this node to its children nodes.
- impurity: the impurity is defined as the value of the split criterion function (such as KL, Chi, or ED) evaluated at this current node
- total_sample: sample size in this node.
- group_sample: sample sizes by treatment groups
- uplift score: treatment effect in this node, if there are multiple treatment, it indicates the maximum (signed) of the treatment effects across all treatment vs control pairs.
- uplift p_value: p value of the treatment effect in this node
- validation uplift score: all the information above is static once the tree is trained (based on the trained trees), while the validation uplift score represents the treatment effect of the testing data when the method fill() is used. This score can be used as a comparison to the training uplift score, to evaluate if the tree has an overfitting issue.

An example notebook is provided in the /examples folder in the repo.

CHAPTER 8

causalml package

8.1 Submodules

8.2 causalml.inference.tree module

8.3 causalml.inference.meta module

class causalml.inference.meta.**BaseRClassifier**(*learner=None*, *outcome_learner=None*, *effect_learner=None*, *ate_alpha=0.05*, *control_name=0*, *n_fold=5*, *random_state=None*)

Bases: causalml.inference.meta.rlearner.BaseRLearner

A parent class for R-learner classifier classes.

fit(*X*, *treatment*, *y*, *p=None*, *verbose=True*)

Fit the treatment effect and outcome models of the R learner.

Parameters

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **treatment** (*np.array or pd.Series*) – a treatment vector
- **y** (*np.array or pd.Series*) – an outcome vector
- **p** (*np.ndarray or pd.Series or dict, optional*) – an array of propensity scores of float (0,1) in the single-treatment case; or, a dictionary of treatment groups that map to propensity vectors of float (0,1); if None will run ElasticNetPropensityModel() to generate the propensity scores.
- **verbose** (*bool, optional*) – whether to output progress logs

predict(*X*)

Predict treatment effects.

Parameters **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix

Returns Predictions of treatment effects.

Return type (numpy.ndarray)

class causalml.inference.meta.**BaseRLearner**(*learner=None*, *outcome_learner=None*, *effect_learner=None*, *ate_alpha=0.05*, *control_name=0*, *n_fold=5*, *random_state=None*)

Bases: object

A parent class for R-learner classes.

An R-learner estimates treatment effects with two machine learning models and the propensity score.

Details of R-learner are available at Nie and Wager (2019) (https://arxiv.org/abs/1712.04912).

bootstrap(*X*, *treatment*, *y*, *p*, *size=10000*)

Runs a single bootstrap. Fits on bootstrapped sample, then predicts on whole population.

estimate_ate(*X*, *treatment*, *y*, *p=None*, *bootstrap_ci=False*, *n_bootstraps=1000*, *bootstrap_size=10000*)

Estimate the Average Treatment Effect (ATE).

Parameters

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **treatment** (*np.array or pd.Series*) – a treatment vector
- **y** (*np.array or pd.Series*) – an outcome vector
- **p** (*np.ndarray or pd.Series or dict, optional*) – an array of propensity scores of float (0,1) in the single-treatment case; or, a dictionary of treatment groups that map to propensity vectors of float (0,1); if None will run ElasticNetPropensityModel() to generate the propensity scores.
- **bootstrap_ci** (*bool*) – whether run bootstrap for confidence intervals
- **n_bootstraps** (*int*) – number of bootstrap iterations
- **bootstrap_size** (*int*) – number of samples per bootstrap

Returns The mean and confidence interval (LB, UB) of the ATE estimate.

fit(*X*, *treatment*, *y*, *p=None*, *verbose=True*)

Fit the treatment effect and outcome models of the R learner.

Parameters

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **treatment** (*np.array or pd.Series*) – a treatment vector
- **y** (*np.array or pd.Series*) – an outcome vector
- **p** (*np.ndarray or pd.Series or dict, optional*) – an array of propensity scores of float (0,1) in the single-treatment case; or, a dictionary of treatment groups that map to propensity vectors of float (0,1); if None will run ElasticNetPropensityModel() to generate the propensity scores.
- **verbose** (*bool, optional*) – whether to output progress logs

fit_predict(*X*, *treatment*, *y*, *p=None*, *return_ci=False*, *n_bootstraps=1000*, *bootstrap_size=10000*, *verbose=True*)

Fit the treatment effect and outcome models of the R learner and predict treatment effects.

Parameters

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **treatment** (*np.array or pd.Series*) – a treatment vector
- **y** (*np.array or pd.Series*) – an outcome vector
- **p** (*np.ndarray or pd.Series or dict, optional*) – an array of propensity scores of float (0,1) in the single-treatment case; or, a dictionary of treatment groups that map to propensity vectors of float (0,1); if None will run ElasticNetPropensityModel() to generate the propensity scores.
- **return_ci** (*bool*) – whether to return confidence intervals
- **n_bootstraps** (*int*) – number of bootstrap iterations
- **bootstrap_size** (*int*) – number of samples per bootstrap
- **verbose** (*bool*) – whether to output progress logs

Returns

Predictions of treatment effects. Output dim: [n_samples, n_treatment]. If return_ci, returns CATE [n_samples, n_treatment], LB [n_samples, n_treatment], UB [n_samples, n_treatment]

Return type (numpy.ndarray)

get_importance(*X=None*, *tau=None*, *model_tau_feature=None*, *features=None*, *method='auto'*, *normalize=True*, *test_size=0.3*, *random_state=None*)

Builds a model (using X to predict estimated/actual tau), and then calculates feature importances based on a specified method.

Currently supported methods include:

- **auto (calculates importance based on estimator's default implementation of feature importance;** estimator must be tree-based) Note: if none provided, it uses lightgbm's LGBMRegressor as estimator, and "gain" as importance type
- **permutation (calculates importance based on mean decrease in accuracy when a feature column is permuted** estimator can be any form)

Hint: for permutation, downsample data for better performance especially if X.shape[1] is large

Parameters

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **tau** (*np.array*) – a treatment effect vector (estimated/actual)
- **model_tau_feature** (*sklearn/lightgbm/xgboost model object*) – an unfitted model object
- **features** (*np.array*) – list/array of feature names. If None, an enumerated list will be used
- **method** (*str*) – auto, permutation
- **normalize** (*bool*) – normalize by sum of importances if method=gini (defaults to True)
- **test_size** (*float/int*) – if float, represents the proportion of the dataset to include in the test split. If int, represents the absolute number of test samples (used for estimating permutation importance)
- **random_state** (*int/RandomState instance/None*) – random state used in permutation importance estimation

get_shap_values (*X=None*, *model_tau_feature=None*, *tau=None*, *features=None*)

Builds a model (using X to predict estimated/actual tau), and then calculates shapley values. :param X: a feature matrix :type X: np.matrix or np.array or pd.Dataframe :param tau: a treatment effect vector (estimated/actual) :type tau: np.array :param model_tau_feature: an unfitted model object :type model_tau_feature: sklearn/lightgbm/xgboost model object :param features: list/array of feature names. If None, an enumerated list will be used. :type features: optional, np.array

plot_importance (*X=None*, *tau=None*, *model_tau_feature=None*, *features=None*, *method='auto'*, *normalize=True*, *test_size=0.3*, *random_state=None*)

Builds a model (using X to predict estimated/actual tau), and then plots feature importances based on a specified method.

Currently supported methods include:

- **auto (calculates importance based on estimator's default implementation of feature importance;** estimator must be tree-based) Note: if none provided, it uses lightgbm's LGBMRegressor as estimator, and "gain" as importance type
- **permutation (calculates importance based on mean decrease in accuracy when a feature column is permuted** estimator can be any form)

Hint: for permutation, downsample data for better performance especially if X.shape[1] is large

Parameters

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **tau** (*np.array*) – a treatment effect vector (estimated/actual)
- **model_tau_feature** (*sklearn/lightgbm/xgboost model object*) – an unfitted model object
- **features** (*optional, np.array*) – list/array of feature names. If None, an enumerated list will be used
- **method** (*str*) – auto, permutation
- **normalize** (*bool*) – normalize by sum of importances if method=gini (defaults to True)
- **test_size** (*float/int*) – if float, represents the proportion of the dataset to include in the test split. If int, represents the absolute number of test samples (used for estimating permutation importance)
- **random_state** (*int/RandomState instance/None*) – random state used in permutation importance estimation

plot_shap_dependence (*treatment_group*, *feature_idx*, *X*, *tau*, *model_tau_feature=None*, *features=None*, *shap_dict=None*, *interaction_idx='auto'*, ***kwargs*)

Plots dependency of shapley values for a specified feature, colored by an interaction feature.

If shapley values have been pre-computed, pass it through the shap_dict parameter. If shap_dict is not provided, this builds a new model (using X to predict estimated/actual tau), and then calculates shapley values.

This plots the value of the feature on the x-axis and the SHAP value of the same feature on the y-axis. This shows how the model depends on the given feature, and is like a richer extension of the classical partial dependence plots. Vertical dispersion of the data points represents interaction effects.

Parameters

- **treatment_group** (*str or int*) – name of treatment group to create dependency plot on
- **feature_idx** (*str or int*) – feature index / name to create dependency plot on

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **tau** (*np.array*) – a treatment effect vector (estimated/actual)
- **model_tau_feature** (*sklearn/lightgbm/xgboost model object*) – an unfitted model object
- **features** (*optional, np.array*) – list/array of feature names. If None, an enumerated list will be used.
- **shap_dict** (*optional, dict*) – a dict of shapley value matrices. If None, shap_dict will be computed.
- **interaction_idx** (*optional, str or int*) – feature index / name used in coloring scheme as interaction feature. If "auto" then shap.common.approximate_interactions is used to pick what seems to be the strongest interaction (note that to find to true strongest interaction you need to compute the SHAP interaction values).

plot_shap_values(*X=None, tau=None, model_tau_feature=None, features=None, shap_dict=None, **kwargs*)

Plots distribution of shapley values.

If shapley values have been pre-computed, pass it through the shap_dict parameter. If shap_dict is not provided, this builds a new model (using X to predict estimated/actual tau), and then calculates shapley values.

Parameters

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix. Required if shap_dict is None.
- **tau** (*np.array*) – a treatment effect vector (estimated/actual)
- **model_tau_feature** (*sklearn/lightgbm/xgboost model object*) – an unfitted model object
- **features** (*optional, np.array*) – list/array of feature names. If None, an enumerated list will be used.
- **shap_dict** (*optional, dict*) – a dict of shapley value matrices. If None, shap_dict will be computed.

predict(*X*)

Predict treatment effects.

Parameters **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix

Returns Predictions of treatment effects.

Return type (numpy.ndarray)

class causalml.inference.meta.**BaseRRegressor**(*learner=None, outcome_learner=None, effect_learner=None, ate_alpha=0.05, control_name=0, n_fold=5, random_state=None*)

Bases: causalml.inference.meta.rlearner.BaseRLearner

A parent class for R-learner regressor classes.

class causalml.inference.meta.**BaseSClassifier**(*learner=None, ate_alpha=0.05, control_name=0*)

Bases: causalml.inference.meta.slearner.BaseSLearner

A parent class for S-learner classifier classes.

predict (*X*, *treatment=None*, *y=None*, *verbose=True*)
Predict treatment effects. :param X: a feature matrix :type X: np.matrix or np.array or pd.Dataframe :param treatment: a treatment vector :type treatment: np.array or pd.Series, optional :param y: an outcome vector :type y: np.array or pd.Series, optional :param verbose: whether to output progress logs :type verbose: bool, optional

Returns Predictions of treatment effects.

Return type (numpy.ndarray)

class causalml.inference.meta.**BaseSLearner** (*learner=None*, *ate_alpha=0.05*, *control_name=0*)
Bases: object

A parent class for S-learner classes. An S-learner estimates treatment effects with one machine learning model. Details of S-learner are available at Kunzel et al. (2018) (https://arxiv.org/abs/1706.03461).

bootstrap (*X*, *treatment*, *y*, *size=10000*)
Runs a single bootstrap. Fits on bootstrapped sample, then predicts on whole population.

estimate_ate (*X*, *treatment*, *y*, *return_ci=False*, *bootstrap_ci=False*, *n_bootstraps=1000*, *bootstrap_size=10000*)
Estimate the Average Treatment Effect (ATE).

Parameters

- **X** (*np.matrix, np.array, or pd.Dataframe*) – a feature matrix
- **treatment** (*np.array or pd.Series*) – a treatment vector
- **y** (*np.array or pd.Series*) – an outcome vector
- **return_ci** (*bool, optional*) – whether to return confidence intervals
- **bootstrap_ci** (*bool*) – whether to return confidence intervals
- **n_bootstraps** (*int*) – number of bootstrap iterations
- **bootstrap_size** (*int*) – number of samples per bootstrap

Returns The mean and confidence interval (LB, UB) of the ATE estimate.

fit (*X*, *treatment*, *y*)
Fit the inference model :param X: a feature matrix :type X: np.matrix, np.array, or pd.Dataframe :param treatment: a treatment vector :type treatment: np.array or pd.Series :param y: an outcome vector :type y: np.array or pd.Series

fit_predict (*X*, *treatment*, *y*, *return_ci=False*, *n_bootstraps=1000*, *bootstrap_size=10000*, *return_components=False*, *verbose=True*)
Fit the inference model of the S learner and predict treatment effects. :param X: a feature matrix :type X: np.matrix, np.array, or pd.Dataframe :param treatment: a treatment vector :type treatment: np.array or pd.Series :param y: an outcome vector :type y: np.array or pd.Series :param return_ci: whether to return confidence intervals :type return_ci: bool, optional :param n_bootstraps: number of bootstrap iterations :type n_bootstraps: int, optional :param bootstrap_size: number of samples per bootstrap :type bootstrap_size: int, optional :param return_components: whether to return outcome for treatment and control seperately :type return_components: bool, optional :param verbose: whether to output progress logs :type verbose: bool, optional

Returns

Predictions of treatment effects. Output dim: [n_samples, n_treatment]. If return_ci, returns CATE [n_samples, n_treatment], LB [n_samples, n_treatment], UB [n_samples, n_treatment]

Return type (numpy.ndarray)

get_importance (*X=None*, *tau=None*, *model_tau_feature=None*, *features=None*, *method='auto'*, *normalize=True*, *test_size=0.3*, *random_state=None*)

Builds a model (using X to predict estimated/actual tau), and then calculates feature importances based on a specified method.

Currently supported methods are:

- **auto (calculates importance based on estimator's default implementation of feature importance;** estimator must be tree-based) Note: if none provided, it uses lightgbm's LGBMRegressor as estimator, and "gain" as importance type
- **permutation (calculates importance based on mean decrease in accuracy when a feature column is permuted** estimator can be any form)

Hint: for permutation, downsample data for better performance especially if X.shape[1] is large

Parameters

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **tau** (*np.array*) – a treatment effect vector (estimated/actual)
- **model_tau_feature** (*sklearn/lightgbm/xgboost model object*) – an unfitted model object
- **features** (*np.array*) – list/array of feature names. If None, an enumerated list will be used
- **method** (*str*) – auto, permutation
- **normalize** (*bool*) – normalize by sum of importances if method=auto (defaults to True)
- **test_size** (*float/int*) – if float, represents the proportion of the dataset to include in the test split. If int, represents the absolute number of test samples (used for estimating permutation importance)
- **random_state** (*int/RandomState instance/None*) – random state used in permutation importance estimation

get_shap_values (*X=None*, *model_tau_feature=None*, *tau=None*, *features=None*)

Builds a model (using X to predict estimated/actual tau), and then calculates shapley values. :param X: a feature matrix :type X: np.matrix or np.array or pd.Dataframe :param tau: a treatment effect vector (estimated/actual) :type tau: np.array :param model_tau_feature: an unfitted model object :type model_tau_feature: sklearn/lightgbm/xgboost model object :param features: list/array of feature names. If None, an enumerated list will be used. :type features: optional, np.array

plot_importance (*X=None*, *tau=None*, *model_tau_feature=None*, *features=None*, *method='auto'*, *normalize=True*, *test_size=0.3*, *random_state=None*)

Builds a model (using X to predict estimated/actual tau), and then plots feature importances based on a specified method.

Currently supported methods are:

- **auto (calculates importance based on estimator's default implementation of feature importance;** estimator must be tree-based) Note: if none provided, it uses lightgbm's LGBMRegressor as estimator, and "gain" as importance type
- **permutation (calculates importance based on mean decrease in accuracy when a feature column is permuted** estimator can be any form)

Hint: for permutation, downsample data for better performance especially if X.shape[1] is large

Parameters

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **tau** (*np.array*) – a treatment effect vector (estimated/actual)
- **model_tau_feature** (*sklearn/lightgbm/xgboost model object*) – an unfitted model object
- **features** (*optional, np.array*) – list/array of feature names. If None, an enumerated list will be used
- **method** (*str*) – auto, permutation
- **normalize** (*bool*) – normalize by sum of importances if method=auto (defaults to True)
- **test_size** (*float/int*) – if float, represents the proportion of the dataset to include in the test split. If int, represents the absolute number of test samples (used for estimating permutation importance)
- **random_state** (*int/RandomState instance/None*) – random state used in permutation importance estimation

plot_shap_dependence(*treatment_group*, *feature_idx*, *X*, *tau*, *model_tau_feature=None*, *features=None*, *shap_dict=None*, *interaction_idx='auto'*, ***kwargs*)

Plots dependency of shapley values for a specified feature, colored by an interaction feature.

If shapley values have been pre-computed, pass it through the shap_dict parameter. If shap_dict is not provided, this builds a new model (using X to predict estimated/actual tau), and then calculates shapley values.

This plots the value of the feature on the x-axis and the SHAP value of the same feature on the y-axis. This shows how the model depends on the given feature, and is like a richer extension of the classical partial dependence plots. Vertical dispersion of the data points represents interaction effects.

Parameters

- **treatment_group** (*str or int*) – name of treatment group to create dependency plot on
- **feature_idx** (*str or int*) – feature index / name to create dependency plot on
- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **tau** (*np.array*) – a treatment effect vector (estimated/actual)
- **model_tau_feature** (*sklearn/lightgbm/xgboost model object*) – an unfitted model object
- **features** (*optional, np.array*) – list/array of feature names. If None, an enumerated list will be used.
- **shap_dict** (*optional, dict*) – a dict of shapley value matrices. If None, shap_dict will be computed.
- **interaction_idx** (*optional, str or int*) – feature index / name used in coloring scheme as interaction feature. If "auto" then shap.common.approximate_interactions is used to pick what seems to be the strongest interaction (note that to find to true strongest interaction you need to compute the SHAP interaction values).

plot_shap_values(*X=None*, *tau=None*, *model_tau_feature=None*, *features=None*, *shap_dict=None*, ***kwargs*)

Plots distribution of shapley values.

If shapley values have been pre-computed, pass it through the shap_dict parameter. If shap_dict is not provided, this builds a new model (using X to predict estimated/actual tau), and then calculates shapley values.

Parameters

- **X** (*np.matrix, np.array, or pd.Dataframe*) – a feature matrix. Required if shap_dict is None.
- **tau** (*np.array*) – a treatment effect vector (estimated/actual)
- **model_tau_feature** (*sklearn/lightgbm/xgboost model object*) – an unfitted model object
- **features** (*optional, np.array*) – list/array of feature names. If None, an enumerated list will be used.
- **shap_dict** (*optional, dict*) – a dict of shapley value matrices. If None, shap_dict will be computed.

predict (*X*, *treatment=None*, *y=None*, *return_components=False*, *verbose=True*)

Predict treatment effects. :param X: a feature matrix :type X: np.matrix or np.array or pd.Dataframe :param treatment: a treatment vector :type treatment: np.array or pd.Series, optional :param y: an outcome vector :type y: np.array or pd.Series, optional :param return_components: whether to return outcome for treatment and control seperately :type return_components: bool, optional :param verbose: whether to output progress logs :type verbose: bool, optional

Returns Predictions of treatment effects.

Return type (numpy.ndarray)

class causalml.inference.meta.**BaseSRegressor** (*learner=None*, *ate_alpha=0.05*, *control_name=0*)

Bases: causalml.inference.meta.slearner.BaseSLearner

A parent class for S-learner regressor classes.

class causalml.inference.meta.**BaseTClassifier** (*learner=None*, *control_learner=None*, *treatment_learner=None*, *ate_alpha=0.05*, *control_name=0*)

Bases: causalml.inference.meta.tlearner.BaseTLearner

A parent class for T-learner classifier classes.

predict (*X*, *treatment=None*, *y=None*, *return_components=False*, *verbose=True*)

Predict treatment effects.

Parameters

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **treatment** (*np.array or pd.Series, optional*) – a treatment vector
- **y** (*np.array or pd.Series, optional*) – an outcome vector
- **verbose** (*bool, optional*) – whether to output progress logs

Returns Predictions of treatment effects.

Return type (numpy.ndarray)

class causalml.inference.meta.**BaseTLearner** (*learner=None*, *control_learner=None*, *treatment_learner=None*, *ate_alpha=0.05*, *control_name=0*)

Bases: object

A parent class for T-learner regressor classes.

A T-learner estimates treatment effects with two machine learning models.

Details of T-learner are available at Kunzel et al. (2018) (https://arxiv.org/abs/1706.03461).

bootstrap (*X*, *treatment*, *y*, *size=10000*)
Runs a single bootstrap. Fits on bootstrapped sample, then predicts on whole population.

estimate_ate (*X*, *treatment*, *y*, *bootstrap_ci=False*, *n_bootstraps=1000*, *bootstrap_size=10000*)
Estimate the Average Treatment Effect (ATE).

Parameters

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **treatment** (*np.array or pd.Series*) – a treatment vector
- **y** (*np.array or pd.Series*) – an outcome vector
- **bootstrap_ci** (*bool*) – whether to return confidence intervals
- **n_bootstraps** (*int*) – number of bootstrap iterations
- **bootstrap_size** (*int*) – number of samples per bootstrap

Returns The mean and confidence interval (LB, UB) of the ATE estimate.

fit (*X*, *treatment*, *y*)
Fit the inference model

Parameters

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **treatment** (*np.array or pd.Series*) – a treatment vector
- **y** (*np.array or pd.Series*) – an outcome vector

fit_predict (*X*, *treatment*, *y*, *return_ci=False*, *n_bootstraps=1000*, *bootstrap_size=10000*, *return_components=False*, *verbose=True*)
Fit the inference model of the T learner and predict treatment effects.

Parameters

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **treatment** (*np.array or pd.Series*) – a treatment vector
- **y** (*np.array or pd.Series*) – an outcome vector
- **return_ci** (*bool*) – whether to return confidence intervals
- **n_bootstraps** (*int*) – number of bootstrap iterations
- **bootstrap_size** (*int*) – number of samples per bootstrap
- **return_components** (*bool, optional*) – whether to return outcome for treatment and control seperately
- **verbose** (*str*) – whether to output progress logs

Returns

Predictions of treatment effects. Output dim: [n_samples, n_treatment]. If return_ci, returns CATE [n_samples, n_treatment], LB [n_samples, n_treatment], UB [n_samples, n_treatment]

Return type (numpy.ndarray)

get_importance(*X=None*, *tau=None*, *model_tau_feature=None*, *features=None*, *method='auto'*, *normalize=True*, *test_size=0.3*, *random_state=None*)

Builds a model (using X to predict estimated/actual tau), and then calculates feature importances based on a specified method.

Currently supported methods are:

- **auto (calculates importance based on estimator's default implementation of feature importance;** estimator must be tree-based) Note: if none provided, it uses lightgbm's LGBMRegressor as estimator, and "gain" as importance type
- **permutation (calculates importance based on mean decrease in accuracy when a feature column is permuted** estimator can be any form)

Hint: for permutation, downsample data for better performance especially if X.shape[1] is large

Parameters

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **tau** (*np.array*) – a treatment effect vector (estimated/actual)
- **model_tau_feature** (*sklearn/lightgbm/xgboost model object*) – an unfitted model object
- **features** (*np.array*) – list/array of feature names. If None, an enumerated list will be used
- **method** (*str*) – auto, permutation
- **normalize** (*bool*) – normalize by sum of importances if method=auto (defaults to True)
- **test_size** (*float/int*) – if float, represents the proportion of the dataset to include in the test split. If int, represents the absolute number of test samples (used for estimating permutation importance)
- **random_state** (*int/RandomState instance/None*) – random state used in permutation importance estimation

get_shap_values(*X=None*, *model_tau_feature=None*, *tau=None*, *features=None*)

Builds a model (using X to predict estimated/actual tau), and then calculates shapley values. :param X: a feature matrix :type X: np.matrix or np.array or pd.Dataframe :param tau: a treatment effect vector (estimated/actual) :type tau: np.array :param model_tau_feature: an unfitted model object :type model_tau_feature: sklearn/lightgbm/xgboost model object :param features: list/array of feature names. If None, an enumerated list will be used. :type features: optional, np.array

plot_importance(*X=None*, *tau=None*, *model_tau_feature=None*, *features=None*, *method='auto'*, *normalize=True*, *test_size=0.3*, *random_state=None*)

Builds a model (using X to predict estimated/actual tau), and then plots feature importances based on a specified method.

Currently supported methods are:

- **auto (calculates importance based on estimator's default implementation of feature importance;** estimator must be tree-based) Note: if none provided, it uses lightgbm's LGBMRegressor as estimator, and "gain" as importance type
- **permutation (calculates importance based on mean decrease in accuracy when a feature column is permuted** estimator can be any form)

Hint: for permutation, downsample data for better performance especially if X.shape[1] is large

Parameters

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **tau** (*np.array*) – a treatment effect vector (estimated/actual)
- **model_tau_feature** (*sklearn/lightgbm/xgboost model object*) – an unfitted model object
- **features** (*optional, np.array*) – list/array of feature names. If None, an enumerated list will be used
- **method** (*str*) – auto, permutation
- **normalize** (*bool*) – normalize by sum of importances if method=auto (defaults to True)
- **test_size** (*float/int*) – if float, represents the proportion of the dataset to include in the test split. If int, represents the absolute number of test samples (used for estimating permutation importance)
- **random_state** (*int/RandomState instance/None*) – random state used in permutation importance estimation

plot_shap_dependence(*treatment_group*, *feature_idx*, *X*, *tau*, *model_tau_feature=None*, *features=None*, *shap_dict=None*, *interaction_idx='auto'*, ***kwargs*)

Plots dependency of shapley values for a specified feature, colored by an interaction feature.

If shapley values have been pre-computed, pass it through the shap_dict parameter. If shap_dict is not provided, this builds a new model (using X to predict estimated/actual tau), and then calculates shapley values.

This plots the value of the feature on the x-axis and the SHAP value of the same feature on the y-axis. This shows how the model depends on the given feature, and is like a richer extension of the classical partial dependence plots. Vertical dispersion of the data points represents interaction effects.

Parameters

- **treatment_group** (*str or int*) – name of treatment group to create dependency plot on
- **feature_idx** (*str or int*) – feature index / name to create dependency plot on
- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **tau** (*np.array*) – a treatment effect vector (estimated/actual)
- **model_tau_feature** (*sklearn/lightgbm/xgboost model object*) – an unfitted model object
- **features** (*optional, np.array*) – list/array of feature names. If None, an enumerated list will be used.
- **shap_dict** (*optional, dict*) – a dict of shapley value matrices. If None, shap_dict will be computed.
- **interaction_idx** (*optional, str or int*) – feature index / name used in coloring scheme as interaction feature. If "auto" then shap.common.approximate_interactions is used to pick what seems to be the strongest interaction (note that to find to true strongest interaction you need to compute the SHAP interaction values).

plot_shap_values(*X=None*, *tau=None*, *model_tau_feature=None*, *features=None*, *shap_dict=None*, ***kwargs*)

Plots distribution of shapley values.

If shapley values have been pre-computed, pass it through the shap_dict parameter. If shap_dict is not provided, this builds a new model (using X to predict estimated/actual tau), and then calculates shapley values.

Parameters

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix. Required if shap_dict is None.
- **tau** (*np.array*) – a treatment effect vector (estimated/actual)
- **model_tau_feature** (*sklearn/lightgbm/xgboost model object*) – an unfitted model object
- **features** (*optional, np.array*) – list/array of feature names. If None, an enumerated list will be used.
- **shap_dict** (*optional, dict*) – a dict of shapley value matrices. If None, shap_dict will be computed.

predict (*X*, *treatment=None*, *y=None*, *return_components=False*, *verbose=True*)
Predict treatment effects.

Parameters

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **treatment** (*np.array or pd.Series, optional*) – a treatment vector
- **y** (*np.array or pd.Series, optional*) – an outcome vector
- **return_components** (*bool, optional*) – whether to return outcome for treatment and control seperately
- **verbose** (*bool, optional*) – whether to output progress logs

Returns Predictions of treatment effects.

Return type (numpy.ndarray)

class causalml.inference.meta.**BaseTRegressor** (*learner=None*, *control_learner=None*, *treatment_learner=None*, *ate_alpha=0.05*, *control_name=0*)

Bases: causalml.inference.meta.tlearner.BaseTLearner

A parent class for T-learner regressor classes.

class causalml.inference.meta.**BaseXClassifier** (*learner=None*, *control_outcome_learner=None*, *treatment_outcome_learner=None*, *control_effect_learner=None*, *treatment_effect_learner=None*, *ate_alpha=0.05*, *control_name=0*)

Bases: causalml.inference.meta.xlearner.BaseXLearner

A parent class for X-learner classifier classes.

fit (*X*, *treatment*, *y*, *p=None*)
Fit the inference model.

Parameters

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **treatment** (*np.array or pd.Series*) – a treatment vector

- **y** (*np.array or pd.Series*) – an outcome vector
- **p** (*np.ndarray or pd.Series or dict, optional*) – an array of propensity scores of float (0,1) in the single-treatment case; or, a dictionary of treatment groups that map to propensity vectors of float (0,1); if None will run ElasticNetPropensityModel() to generate the propensity scores.

predict (*X*, *treatment=None*, *y=None*, *p=None*, *return_components=False*, *verbose=True*)
Predict treatment effects.

Parameters

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **treatment** (*np.array or pd.Series, optional*) – a treatment vector
- **y** (*np.array or pd.Series, optional*) – an outcome vector
- **p** (*np.ndarray or pd.Series or dict, optional*) – an array of propensity scores of float (0,1) in the single-treatment case; or, a dictionary of treatment groups that map to propensity vectors of float (0,1); if None will run ElasticNetPropensityModel() to generate the propensity scores.
- **return_components** (*bool, optional*) – whether to return outcome for treatment and control seperately
- **return_p_score** (*bool, optional*) – whether to return propensity score
- **verbose** (*bool, optional*) – whether to output progress logs

Returns Predictions of treatment effects.

Return type (numpy.ndarray)

class causalml.inference.meta.**BaseXLearner** (*learner=None*, *control_outcome_learner=None*, *treatment_outcome_learner=None*, *control_effect_learner=None*, *treatment_effect_learner=None*, *ate_alpha=0.05*, *control_name=0*)

Bases: object

A parent class for X-learner regressor classes.

An X-learner estimates treatment effects with four machine learning models.

Details of X-learner are available at Kunzel et al. (2018) (https://arxiv.org/abs/1706.03461).

bootstrap (*X*, *treatment*, *y*, *p*, *size=10000*)
Runs a single bootstrap. Fits on bootstrapped sample, then predicts on whole population.

estimate_ate (*X*, *treatment*, *y*, *p=None*, *bootstrap_ci=False*, *n_bootstraps=1000*, *bootstrap_size=10000*)
Estimate the Average Treatment Effect (ATE).

Parameters

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **treatment** (*np.array or pd.Series*) – a treatment vector
- **y** (*np.array or pd.Series*) – an outcome vector
- **p** (*np.ndarray or pd.Series or dict, optional*) – an array of propensity scores of float (0,1) in the single-treatment case; or, a dictionary of treatment groups that

map to propensity vectors of float (0,1); if None will run ElasticNetPropensityModel() to generate the propensity scores.

- **bootstrap_ci** (*bool*) – whether run bootstrap for confidence intervals
- **n_bootstraps** (*int*) – number of bootstrap iterations
- **bootstrap_size** (*int*) – number of samples per bootstrap

Returns The mean and confidence interval (LB, UB) of the ATE estimate.

fit (*X*, *treatment*, *y*, *p=None*)

Fit the inference model.

Parameters

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **treatment** (*np.array or pd.Series*) – a treatment vector
- **y** (*np.array or pd.Series*) – an outcome vector
- **p** (*np.ndarray or pd.Series or dict, optional*) – an array of propensity scores of float (0,1) in the single-treatment case; or, a dictionary of treatment groups that map to propensity vectors of float (0,1); if None will run ElasticNetPropensityModel() to generate the propensity scores.

fit_predict (*X*, *treatment*, *y*, *p=None*, *return_ci=False*, *n_bootstraps=1000*, *bootstrap_size=10000*, *return_components=False*, *verbose=True*)

Fit the treatment effect and outcome models of the R learner and predict treatment effects.

Parameters

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **treatment** (*np.array or pd.Series*) – a treatment vector
- **y** (*np.array or pd.Series*) – an outcome vector
- **p** (*np.ndarray or pd.Series or dict, optional*) – an array of propensity scores of float (0,1) in the single-treatment case; or, a dictionary of treatment groups that map to propensity vectors of float (0,1); if None will run ElasticNetPropensityModel() to generate the propensity scores.
- **return_ci** (*bool*) – whether to return confidence intervals
- **n_bootstraps** (*int*) – number of bootstrap iterations
- **bootstrap_size** (*int*) – number of samples per bootstrap
- **return_components** (*bool, optional*) – whether to return outcome for treatment and control seperately
- **verbose** (*str*) – whether to output progress logs

Returns

Predictions of treatment effects. Output dim: [n_samples, n_treatment] If return_ci, returns CATE [n_samples, n_treatment], LB [n_samples, n_treatment], UB [n_samples, n_treatment]

Return type (numpy.ndarray)

get_importance (*X=None*, *tau=None*, *model_tau_feature=None*, *features=None*, *method='auto'*, *normalize=True*, *test_size=0.3*, *random_state=None*)

Builds a model (using X to predict estimated/actual tau), and then calculates feature importances based on a specified method.

Currently supported methods are:

- **auto (calculates importance based on estimator's default implementation of feature importance;** estimator must be tree-based) Note: if none provided, it uses lightgbm's LGBMRegressor as estimator, and "gain" as importance type
- **permutation (calculates importance based on mean decrease in accuracy when a feature column is permuted** estimator can be any form)

Hint: for permutation, downsample data for better performance especially if X.shape[1] is large

Parameters

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **tau** (*np.array*) – a treatment effect vector (estimated/actual)
- **model_tau_feature** (*sklearn/lightgbm/xgboost model object*) – an unfitted model object
- **features** (*np.array*) – list/array of feature names. If None, an enumerated list will be used
- **method** (*str*) – auto, permutation
- **normalize** (*bool*) – normalize by sum of importances if method=auto (defaults to True)
- **test_size** (*float/int*) – if float, represents the proportion of the dataset to include in the test split. If int, represents the absolute number of test samples (used for estimating permutation importance)
- **random_state** (*int/RandomState instance/None*) – random state used in permutation importance estimation

get_shap_values(*X=None*, *model_tau_feature=None*, *tau=None*, *features=None*)

Builds a model (using X to predict estimated/actual tau), and then calculates shapley values. :param X: a feature matrix :type X: np.matrix or np.array or pd.Dataframe :param tau: a treatment effect vector (estimated/actual) :type tau: np.array :param model_tau_feature: an unfitted model object :type model_tau_feature: sklearn/lightgbm/xgboost model object :param features: list/array of feature names. If None, an enumerated list will be used. :type features: optional, np.array

plot_importance(*X=None*, *tau=None*, *model_tau_feature=None*, *features=None*, *method='auto'*, *normalize=True*, *test_size=0.3*, *random_state=None*)

Builds a model (using X to predict estimated/actual tau), and then plots feature importances based on a specified method.

Currently supported methods are:

- **auto (calculates importance based on estimator's default implementation of feature importance;** estimator must be tree-based) Note: if none provided, it uses lightgbm's LGBMRegressor as estimator, and "gain" as importance type
- **permutation (calculates importance based on mean decrease in accuracy when a feature column is permuted** estimator can be any form)

Hint: for permutation, downsample data for better performance especially if X.shape[1] is large

Parameters

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **tau** (*np.array*) – a treatment effect vector (estimated/actual)

- **model_tau_feature** (*sklearn/lightgbm/xgboost model object*) – an unfitted model object
- **features** (*optional, np.array*) – list/array of feature names. If None, an enumerated list will be used
- **method** (*str*) – auto, permutation
- **normalize** (*bool*) – normalize by sum of importances if method=auto (defaults to True)
- **test_size** (*float/int*) – if float, represents the proportion of the dataset to include in the test split. If int, represents the absolute number of test samples (used for estimating permutation importance)
- **random_state** (*int/RandomState instance/None*) – random state used in permutation importance estimation

plot_shap_dependence(*treatment_group*, *feature_idx*, *X*, *tau*, *model_tau_feature=None*, *features=None*, *shap_dict=None*, *interaction_idx='auto'*, ***kwargs*)

Plots dependency of shapley values for a specified feature, colored by an interaction feature.

If shapley values have been pre-computed, pass it through the shap_dict parameter. If shap_dict is not provided, this builds a new model (using X to predict estimated/actual tau), and then calculates shapley values.

This plots the value of the feature on the x-axis and the SHAP value of the same feature on the y-axis. This shows how the model depends on the given feature, and is like a richer extension of the classical partial dependence plots. Vertical dispersion of the data points represents interaction effects.

Parameters

- **treatment_group** (*str or int*) – name of treatment group to create dependency plot on
- **feature_idx** (*str or int*) – feature index / name to create dependency plot on
- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **tau** (*np.array*) – a treatment effect vector (estimated/actual)
- **model_tau_feature** (*sklearn/lightgbm/xgboost model object*) – an unfitted model object
- **features** (*optional, np.array*) – list/array of feature names. If None, an enumerated list will be used.
- **shap_dict** (*optional, dict*) – a dict of shapley value matrices. If None, shap_dict will be computed.
- **interaction_idx** (*optional, str or int*) – feature index / name used in coloring scheme as interaction feature. If "auto" then shap.common.approximate_interactions is used to pick what seems to be the strongest interaction (note that to find to true strongest interaction you need to compute the SHAP interaction values).

plot_shap_values(*X=None*, *tau=None*, *model_tau_feature=None*, *features=None*, *shap_dict=None*, ***kwargs*)

Plots distribution of shapley values.

If shapley values have been pre-computed, pass it through the shap_dict parameter. If shap_dict is not provided, this builds a new model (using X to predict estimated/actual tau), and then calculates shapley values.

Parameters

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix. Required if shap_dict is None.
- **tau** (*np.array*) – a treatment effect vector (estimated/actual)
- **model_tau_feature** (*sklearn/lightgbm/xgboost model object*) – an unfitted model object
- **features** (*optional, np.array*) – list/array of feature names. If None, an enumerated list will be used.
- **shap_dict** (*optional, dict*) – a dict of shapley value matrices. If None, shap_dict will be computed.

predict (*X*, *treatment=None*, *y=None*, *p=None*, *return_components=False*, *verbose=True*)

Predict treatment effects.

Parameters

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **treatment** (*np.array or pd.Series, optional*) – a treatment vector
- **y** (*np.array or pd.Series, optional*) – an outcome vector
- **p** (*np.ndarray or pd.Series or dict, optional*) – an array of propensity scores of float (0,1) in the single-treatment case; or, a dictionary of treatment groups that map to propensity vectors of float (0,1); if None will run ElasticNetPropensityModel() to generate the propensity scores.
- **return_components** (*bool, optional*) – whether to return outcome for treatment and control seperately
- **verbose** (*bool, optional*) – whether to output progress logs

Returns Predictions of treatment effects.

Return type (numpy.ndarray)

class causalml.inference.meta.**BaseXRegressor** (*learner=None*, *control_outcome_learner=None*, *treatment_outcome_learner=None*, *control_effect_learner=None*, *treatment_effect_learner=None*, *ate_alpha=0.05*, *control_name=0*)

Bases: causalml.inference.meta.xlearner.BaseXLearner

A parent class for X-learner regressor classes.

class causalml.inference.meta.**LRSRegressor** (*ate_alpha=0.05*, *control_name=0*)

Bases: causalml.inference.meta.slearner.BaseSRegressor

estimate_ate (*X*, *treatment*, *y*)

Estimate the Average Treatment Effect (ATE). :param X: a feature matrix :type X: np.matrix, np.array, or pd.Dataframe :param treatment: a treatment vector :type treatment: np.array or pd.Series :param y: an outcome vector :type y: np.array or pd.Series

Returns The mean and confidence interval (LB, UB) of the ATE estimate.

class causalml.inference.meta.**MLPTRegressor** (*ate_alpha=0.05*, *control_name=0*, **args*, ***kwargs*)

Bases: causalml.inference.meta.tlearner.BaseTRegressor

class causalml.inference.meta.**TMLELearner**(*learner*, *ate_alpha=0.05*, *control_name=0*, *cv=None*, *calibrate_propensity=True*)

Bases: object

Targeted maximum likelihood estimation.

Ref: Gruber, S., & Van Der Laan, M. J. (2009). Targeted maximum likelihood estimation: A gentle introduction.

estimate_ate(*X*, *p*, *treatment*, *y*, *segment=None*, *return_ci=False*)

Estimate the Average Treatment Effect (ATE).

Parameters

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **p** (*np.ndarray or pd.Series or dict*) – an array of propensity scores of float (0,1) in the single-treatment case; or, a dictionary of treatment groups that map to propensity vectors of float (0,1)
- **treatment** (*np.array or pd.Series*) – a treatment vector
- **y** (*np.array or pd.Series*) – an outcome vector
- **segment** (*np.array, optional*) – An optional segment vector of int. If given, the ATE and its CI will be estimated for each segment.
- **return_ci** (*bool, optional*) – Whether to return confidence intervals

Returns The ATE and its confidence interval (LB, UB) for each treatment, t and segment, s

Return type (tuple)

class causalml.inference.meta.**XGBRRegressor**(*early_stopping=True*, *test_size=0.3*, *early_stopping_rounds=30*, *effect_learner_objective='rank:pairwise'*, *effect_learner_n_estimators=500*, *random_state=42*, **args*, ***kwargs*)

Bases: causalml.inference.meta.rlearner.BaseRRegressor

fit(*X*, *treatment*, *y*, *p=None*, *verbose=True*)

Fit the treatment effect and outcome models of the R learner.

Parameters

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **y** (*np.array or pd.Series*) – an outcome vector
- **p** (*np.ndarray or pd.Series or dict, optional*) – an array of propensity scores of float (0,1) in the single-treatment case; or, a dictionary of treatment groups that map to propensity vectors of float (0,1); if None will run ElasticNetPropensityModel() to generate the propensity scores.
- **verbose** (*bool, optional*) – whether to output progress logs

class causalml.inference.meta.**XGBTRegressor**(*ate_alpha=0.05*, *control_name=0*, **args*, ***kwargs*)

Bases: causalml.inference.meta.tlearner.BaseTRegressor

8.4 causalml.optimize module

class causalml.optimize.**PolicyLearner**(*outcome_learner=GradientBoostingRegressor(), policy_learner=GradientBoostingClassifier(), clip_bounds=(0.001, 0.999), n_fold=5, random_state=None*)

Bases: object

A Learner that learns a treatment assignment policy with observational data using doubly robust estimator of causal effect for binary treatment.

Details of the policy learner are available at Athey and Wager (2018) (https://arxiv.org/abs/1702.02896).

fit(*X, p, treatment, y, dhat*)

Fit the treatment assignment policy learner.

Parameters

- **X** (*np.matrix*) – a feature matrix
- **p** (*np.array*) – a propensity score vector between 0 and 1
- **treatment** (*np.array*) – a treatment vector (1 if treated, otherwise 0)
- **y** (*np.array*) – an outcome vector
- **dhat** (*np.array*) – a predicted treatment effect vector

Returns returns an instance of self.

Return type self

predict(*X*)

Predict treatment assignment that optimizes the outcome.

Parameters **X** (*np.matrix*) – a feature matrix

Returns predictions of treatment assignment.

Return type (numpy.ndarray)

8.5 causalml.dataset module

8.6 causalml.match module

class causalml.match.**MatchOptimizer**(*treatment_col='is_treatment', ps_col='pihat', user_col=None, matching_covariates=['pihat'], max_smd=0.1, max_deviation=0.1, caliper_range=(0.01, 0.5), max_pihat_range=(0.95, 0.999), max_iter_per_param=5, min_users_per_group=1000, smd_cols=['pihat'], dev_cols_transformations={'pihat': <function mean>}, dev_factor=1.0, verbose=True*)

Bases: object

check_table_one(*tableone, matched, score_cols, pihat_threshold, caliper*)

match_and_check(*score_cols, pihat_threshold, caliper*)

search_best_match(*df*)

single_match (*score_cols*, *pihat_threshold*, *caliper*)

class causalml.match.**NearestNeighborMatch** (*caliper=0.2*, *replace=False*, *ratio=1*, *shuffle=True*, *random_state=None*)

Bases: `object`

Propensity score matching based on the nearest neighbor algorithm.

caliper
threshold to be considered as a match.

Type float

replace
whether to match with replacement or not

Type bool

ratio
ratio of control / treatment to be matched. used only if replace=True.

Type int

shuffle
whether to shuffle the treatment group data before matching

Type bool

random_state
RandomState or an int seed

Type numpy.random.RandomState or int

match (*data*, *treatment_col*, *score_cols*)
Find matches from the control group by matching on specified columns (propensity preferred).

Parameters

- **data** (*pandas.DataFrame*) – total input data
- **treatment_col** (*str*) – the column name for the treatment
- **score_cols** (*list*) – list of column names for matching (propensity column should be included)

Returns

The subset of data consisting of matched treatment and control group data.

Return type (pandas.DataFrame)

match_by_group (*data*, *treatment_col*, *score_cols*, *groupby_col*)
Find matches from the control group stratified by groupby_col, by matching on specified columns (propensity preferred).

Parameters

- **data** (*pandas.DataFrame*) – total sample data
- **treatment_col** (*str*) – the column name for the treatment
- **score_cols** (*list*) – list of column names for matching (propensity column should be included)
- **groupby_col** (*str*) – the column name to be used for stratification

Returns

The subset of data consisting of matched treatment and control group data.

Return type (pandas.DataFrame)

`causalml.match.create_table_one`(*data*, *treatment_col*, *features*)

Report balance in input features between the treatment and control groups.

References

R's tableone at CRAN: https://github.com/kaz-yos/tableone Python's tableone at PyPi: https://github.com/tompollard/tableone

Parameters

- **data** (*pandas.DataFrame*) – total or matched sample data
- **treatment_col** (*str*) – the column name for the treatment
- **features** (*list of str*) – the column names of features

Returns

A table with the means and standard deviations in the treatment and control groups, and the SMD between two groups for the features.

Return type (pandas.DataFrame)

`causalml.match.smd`(*feature*, *treatment*)

Calculate the standard mean difference (SMD) of a feature between the treatment and control groups.

The definition is available at https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3144483/#s11title

Parameters

- **feature** (*pandas.Series*) – a column of a feature to calculate SMD for
- **treatment** (*pandas.Series*) – a column that indicate whether a row is in the treatment group or not

Returns The SMD of the feature

Return type (float)

8.7 causalml.propensity module

class `causalml.propensity.ElasticNetPropensityModel`(*n_fold=4*, *Cs=array([1.00230524*, *2.15608891*, *4.63802765*, *9.97700064])*, *l1_ratios=array([0.001*, *0.33366667*, *0.66633333*, *0.999])*, *clip_bounds=(0.001*, *0.999)*, *cv=None*, *random_state=None*)

Bases: `object`

Propensity regression model based on the ElasticNet algorithm.

model

a propensity model object

Type sklearn.linear_model.ElasticNetCV

fit (*X*, *y*)
Fit a propensity model.

Parameters

- **X** (*numpy.ndarray*) – a feature matrix
- **y** (*numpy.ndarray*) – a binary target vector

fit_predict (*X*, *y*)
Fit a propensity model and predict propensity scores.

Parameters

- **X** (*numpy.ndarray*) – a feature matrix
- **y** (*numpy.ndarray*) – a binary target vector

Returns Propensity scores between 0 and 1.

Return type (numpy.ndarray)

predict (*X*)
Predict propensity scores.

Parameters **X** (*numpy.ndarray*) – a feature matrix

Returns Propensity scores between 0 and 1.

Return type (numpy.ndarray)

class causalml.propensity.**GradientBoostedPropensityModel** (*max_depth=8, learning_rate=0.1, n_estimators=100, objective='binary:logistic', n_thread=2, colsample_bytree=0.8, early_stop=False, stop_val_size=0.2, n_stop_rounds=10, clip_bounds=(0.001, 0.999), random_state=None*)

Bases: object

Fits a simple gradient boosted propensity score model with optional early stopping.

Notes

Please see the xgboost documentation for more information on gradient boosting tuning parameters: https://xgboost.readthedocs.io/en/latest/python/python_api.html

cpu_count = 2

fit (*X*, *y*)
Fit a propensity model.

Parameters

- **X** (*numpy.ndarray*) – a feature matrix
- **y** (*numpy.ndarray*) – a binary target vector

fit_predict (*X*, *y*)
Fit a propensity model and predict propensity scores.

Parameters

- **X** (*numpy.ndarray*) – a feature matrix
- **y** (*numpy.ndarray*) – a binary target vector

Returns Propensity scores between 0 and 1.

Return type (numpy.ndarray)

predict (*X*)
Predict propensity scores.

Parameters **X** (*numpy.ndarray*) – a feature matrix

Returns Propensity scores between 0 and 1.

Return type (numpy.ndarray)

causalml.propensity.**calibrate** (*ps*, *treatment*)
Calibrate propensity scores with logistic GAM.

Ref: https://pygam.readthedocs.io/en/latest/api/logisticgam.html

Parameters

- **ps** (*numpy.array*) – a propensity score vector
- **treatment** (*numpy.array*) – a binary treatment vector (0: control, 1: treated)

Returns a calibrated propensity score vector

Return type (numpy.array)

causalml.propensity.**compute_propensity_score** (*X*, *treatment*, *p_model=None*, *X_pred=None*, *treatment_pred=None*, *calibrate_p=True*)
Generate propensity score if user didn't provide

Parameters

- **X** (*np.matrix*) – features for training
- **treatment** (*np.array or pd.Series*) – a treatment vector for training
- **p_model** (*propensity model object, optional*) – ElasticNetPropensityModel (default) / GradientBoostedPropensityModel
- **X_pred** (*np.matrix, optional*) – features for prediction
- **treatment_pred** (*np.array or pd.Series, optional*) – a treatment vector for prediciton
- **calibrate_p** (*bool, optional*) – whether calibrate the propensity score

Returns

(tuple)

- p (numpy.ndarray): propensity score
- p_model_dict (dict): dictionary of propensity model

8.8 causalml.metrics module

class causalml.metrics.**Sensitivity**(*df*, *inference_features*, *p_col*, *treatment_col*, *outcome_col*, *learner*, **args*, ***kwargs*)

Bases: `object`

A Sensitivity Check class to support Placebo Treatment, Irrelevant Additional Confounder and Subset validation refutation methods to verify causal inference.

Reference: https://github.com/microsoft/dowhy/blob/master/dowhy/causal_refuters/

get_ate_ci(*X*, *p*, *treatment*, *y*)

Return the confidence intervals for treatment effects prediction.

Parameters

- **X** (*np.matrix*) – a feature matrix
- **p** (*np.array*) – a propensity score vector between 0 and 1
- **treatment** (*np.array*) – a treatment vector (1 if treated, otherwise 0)
- **y** (*np.array*) – an outcome vector

Returns Mean and confidence interval (LB, UB) of the ATE estimate.

Return type (numpy.ndarray)

get_class_object(*method_name*, **args*, ***kwargs*)

Return class object based on input method :param method_name: a list of sensitivity analysis method :type method_name: list of str

Returns Sensitivy Class

Return type (class)

get_prediction(*X*, *p*, *treatment*, *y*)

Return the treatment effects prediction.

Parameters

- **X** (*np.matrix*) – a feature matrix
- **p** (*np.array*) – a propensity score vector between 0 and 1
- **treatment** (*np.array*) – a treatment vector (1 if treated, otherwise 0)
- **y** (*np.array*) – an outcome vector

Returns Predictions of treatment effects

Return type (numpy.ndarray)

sensitivity_analysis(*methods*, *sample_size=None*, *confound='one_sided'*, *alpha_range=None*)

Return the sensitivity data by different method

Parameters

- **method** (*list of str*) – a list of sensitivity analysis method
- **sample_size** (*float, optional*) – ratio for subset the original data
- **confound** (*string, optional*) – the name of confouding function
- **alpha_range** (*np.array, optional*) – a parameter to pass the confounding function

Returns a feature matrix p (np.array): a propensity score vector between 0 and 1 treatment (np.array): a treatment vector (1 if treated, otherwise 0) y (np.array): an outcome vector

Return type X (np.matrix)

sensitivity_estimate()

summary(*method*)

Summary report :param method_name: sensitivity analysis method :type method_name: str

Returns a summary dataframe

Return type (pd.DataFrame)

class causalml.metrics.**SensitivityPlaceboTreatment**(**args*, ***kwargs*)

Bases: `causalml.metrics.sensitivity.Sensitivity`

Replaces the treatment variable with a new variable randomly generated.

sensitivity_estimate()

Summary report :param return_ci: sensitivity analysis method :type return_ci: str

Returns a summary dataframe

Return type (pd.DataFrame)

class causalml.metrics.**SensitivityRandomCause**(**args*, ***kwargs*)

Bases: `causalml.metrics.sensitivity.Sensitivity`

Adds an irrelevant random covariate to the dataframe.

sensitivity_estimate()

class causalml.metrics.**SensitivityRandomReplace**(**args*, ***kwargs*)

Bases: `causalml.metrics.sensitivity.Sensitivity`

Replaces a random covariate with an irrelevant variable.

sensitivity_estimate()

Replaces a random covariate with an irrelevant variable.

class causalml.metrics.**SensitivitySelectionBias**(**args*, *confound='one_sided'*, *alpha_range=None*, *sensitivity_features=None*, ***kwargs*)

Bases: `causalml.metrics.sensitivity.Sensitivity`

Reference:

[1] Blackwell, Matthew. "A selection bias approach to sensitivity analysis for causal effects." Political Analysis 22.2 (2014): 169-182. https://www.mattblackwell.org/files/papers/causalsens.pdf

[2] Confouding parameter alpha_range using the same range as in: https://github.com/mattblackwell/causalsens/blob/master/R/causalsens.R

causalsens()

partial_rsqs_confounding(*sens_df*, *feature_name*, *partial_rsqs_value*, *range=0.01*)

Check partial rsqs values of feature corresponding confounding amonunt of ATE :param sens_df: a data frame output from causalsens :type sens_df: pandas.DataFrame :param feature_name: feature name to check :type feature_name: str :param partial_rsqs_value: partial rsquare value of feature :type partial_rsqs_value: float :param range: range to search from sens_df :type range: float

Return: min and max value of confounding amount

plot (*sens_df*, *partial_rsqs_df=None*, *type='raw'*, *ci=False*, *partial_rsqs=False*)
Plot the results of a sensitivity analysis against unmeasured :param sens_df: a data frame output from causalsens :type sens_df: pandas.DataFrame :param partial_rsqs_d: a data frame output from causalsens including partial rsqure :type partial_rsqs_d: pandas.DataFrame :param type: the type of plot to draw, 'raw' or 'r.squared' are supported :type type: str, optional :param ci: whether plot confidence intervals :type ci: bool, optional :param partial_rsqs: whether plot partial rsquare results :type partial_rsqs: bool, optional

summary (*method='Selection Bias'*)
Summary report for Selection Bias Method :param method_name: sensitivity analysis method :type method_name: str

Returns a summary dataframe

Return type (pd.DataFrame)

class causalml.metrics.**SensitivitySubsetData** (**args*, ***kwargs*)
Bases: causalml.metrics.sensitivity.Sensitivity

Takes a random subset of size sample_size of the data.

sensitivity_estimate ()

causalml.metrics.**ape** (*y*, *p*)
Absolute Percentage Error (APE). :param y: target :type y: float :param p: prediction :type p: float

Returns APE

Return type e (float)

causalml.metrics.**auuc_score** (*df*, *outcome_col='y'*, *treatment_col='w'*, *treatment_effect_col='tau'*, *normalize=True*, *tmle=False*, **args*, ***kwarg*)
Calculate the AUUC (Area Under the Uplift Curve) score.

Args: df (pandas.DataFrame): a data frame with model estimates and actual data as columns outcome_col (str, optional): the column name for the actual outcome treatment_col (str, optional): the column name for the treatment indicator (0 or 1) treatment_effect_col (str, optional): the column name for the true treatment effect normalize (bool, optional): whether to normalize the y-axis to 1 or not

Returns the AUUC score

Return type (float)

causalml.metrics.**classification_metrics** (*y*, *p*, *w=None*, *metrics={'AUC': <function roc_auc_score>, 'Log Loss': <function logloss>}*)
Log metrics for classifiers.

Parameters

- **y** (*numpy.array*) – target
- **p** (*numpy.array*) – prediction
- **w** (*numpy.array, optional*) – a treatment vector (1 or True: treatment, 0 or False: control). If given, log metrics for the treatment and control group separately
- **metrics** (*dict, optional*) – a dictionary of the metric names and functions

causalml.metrics.**get_cumgain** (*df*, *outcome_col='y'*, *treatment_col='w'*, *treatment_effect_col='tau'*, *normalize=False*, *random_seed=42*)
Get cumulative gains of model estimates in population.

If the true treatment effect is provided (e.g. in synthetic data), it's calculated as the cumulative gain of the true treatment effect in each population. Otherwise, it's calculated as the cumulative difference between the mean outcomes of the treatment and control groups in each population.

For details, see Section 4.1 of Gutierrez and G{'e}rardy (2016), *Causal Inference and Uplift Modeling: A review of the literature*.

For the former, *treatment_effect_col* should be provided. For the latter, both *outcome_col* and *treatment_col* should be provided.

Parameters

- **df** (*pandas.DataFrame*) – a data frame with model estimates and actual data as columns
- **outcome_col** (*str, optional*) – the column name for the actual outcome
- **treatment_col** (*str, optional*) – the column name for the treatment indicator (0 or 1)
- **treatment_effect_col** (*str, optional*) – the column name for the true treatment effect
- **normalize** (*bool, optional*) – whether to normalize the y-axis to 1 or not
- **random_seed** (*int, optional*) – random seed for numpy.random.rand()

Returns cumulative gains of model estimates in population

Return type (pandas.DataFrame)

causalml.metrics.**get_cumlift**(*df*, *outcome_col='y'*, *treatment_col='w'*, *treatment_effect_col='tau'*, *random_seed=42*)

Get average uplifts of model estimates in cumulative population.

If the true treatment effect is provided (e.g. in synthetic data), it's calculated as the mean of the true treatment effect in each of cumulative population. Otherwise, it's calculated as the difference between the mean outcomes of the treatment and control groups in each of cumulative population.

For details, see Section 4.1 of Gutierrez and G{'e}rardy (2016), *Causal Inference and Uplift Modeling: A review of the literature*.

For the former, *treatment_effect_col* should be provided. For the latter, both *outcome_col* and *treatment_col* should be provided.

Parameters

- **df** (*pandas.DataFrame*) – a data frame with model estimates and actual data as columns
- **outcome_col** (*str, optional*) – the column name for the actual outcome
- **treatment_col** (*str, optional*) – the column name for the treatment indicator (0 or 1)
- **treatment_effect_col** (*str, optional*) – the column name for the true treatment effect
- **random_seed** (*int, optional*) – random seed for numpy.random.rand()

Returns average uplifts of model estimates in cumulative population

Return type (pandas.DataFrame)

causalml.metrics.**get_qini**(*df*, *outcome_col='y'*, *treatment_col='w'*, *treatment_effect_col='tau'*, *normalize=False*, *random_seed=42*)

Get Qini of model estimates in population.

If the true treatment effect is provided (e.g. in synthetic data), it's calculated as the cumulative gain of the true treatment effect in each population. Otherwise, it's calculated as the cumulative difference between the mean outcomes of the treatment and control groups in each population.

For details, see Radcliffe (2007), *Using Control Group to Target on Predicted Lift: Building and Assessing Uplift Models*

For the former, *treatment_effect_col* should be provided. For the latter, both *outcome_col* and *treatment_col* should be provided.

Parameters

- **df** (*pandas.DataFrame*) – a data frame with model estimates and actual data as columns
- **outcome_col** (*str, optional*) – the column name for the actual outcome
- **treatment_col** (*str, optional*) – the column name for the treatment indicator (0 or 1)
- **treatment_effect_col** (*str, optional*) – the column name for the true treatment effect
- **normalize** (*bool, optional*) – whether to normalize the y-axis to 1 or not
- **random_seed** (*int, optional*) – random seed for numpy.random.rand()

Returns cumulative gains of model estimates in population

Return type (pandas.DataFrame)

causalml.metrics.**get_tmlegain**(*df*, *inference_col*, *learner=LGBMRegressor(learning_rate=0.05, n_estimators=300, num_leaves=64)*, *outcome_col='y'*, *treatment_col='w'*, *p_col='p'*, *n_segment=5*, *cv=None*, *calibrate_propensity=True*, *ci=False*)

Get TMLE based average uplifts of model estimates of segments.

Parameters

- **df** (*pandas.DataFrame*) – a data frame with model estimates and actual data as columns
- **inferenece_col** (*list of str*) – a list of columns that used in learner for inference
- **learner** (*optional*) – a model used by TMLE to estimate the outcome
- **outcome_col** (*str, optional*) – the column name for the actual outcome
- **treatment_col** (*str, optional*) – the column name for the treatment indicator (0 or 1)
- **p_col** (*str, optional*) – the column name for propensity score
- **n_segment** (*int, optional*) – number of segment that TMLE will estimated for each
- **cv** (*sklearn.model_selection._BaseKFold, optional*) – sklearn CV object
- **calibrate_propensity** (*bool, optional*) – whether calibrate propensity score or not
- **ci** (*bool, optional*) – whether return confidence intervals for ATE or not

Returns cumulative gains of model estimates based of TMLE

Return type (pandas.DataFrame)

`causalml.metrics.`**`get_tmleqini`**(*df*, *inference_col*, *learner=LGBMRegressor(learning_rate=0.05*, *n_estimators=300*, *num_leaves=64)*, *outcome_col='y'*, *treatment_col='w'*, *p_col='p'*, *n_segment=5*, *cv=None*, *calibrate_propensity=True*, *ci=False*, *normalize=False*)

Get TMLE based Qini of model estimates by segments.

Parameters

- **df** (*pandas.DataFrame*) – a data frame with model estimates and actual data as columns
- **inferenece_col** (*list of str*) – a list of columns that used in learner for inference
- **learner** (*optional*) – a model used by TMLE to estimate the outcome
- **outcome_col** (*str, optional*) – the column name for the actual outcome
- **treatment_col** (*str, optional*) – the column name for the treatment indicator (0 or 1)
- **p_col** (*str, optional*) – the column name for propensity score
- **n_segment** (*int, optional*) – number of segment that TMLE will estimated for each
- **cv** (*sklearn.model_selection._BaseKFold, optional*) – sklearn CV object
- **calibrate_propensity** (*bool, optional*) – whether calibrate propensity score or not
- **ci** (*bool, optional*) – whether return confidence intervals for ATE or not

Returns cumulative gains of model estimates based of TMLE

Return type (pandas.DataFrame)

`causalml.metrics.`**`gini`**(*y*, *p*)

Normalized Gini Coefficient.

Parameters

- **y** (*numpy.array*) – target
- **p** (*numpy.array*) – prediction

Returns normalized Gini coefficient

Return type e (numpy.float64)

`causalml.metrics.`**`logloss`**(*y*, *p*)

Bounded log loss error. :param y: target :type y: numpy.array :param p: prediction :type p: numpy.array

Returns bounded log loss error

`causalml.metrics.`**`mae`**(*y_true*, *y_pred*, ***, *sample_weight=None*, *multioutput='uniform_average'*)

Mean absolute error regression loss

Read more in the User Guide.

Parameters

- **y_true** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – Ground truth (correct) target values.
- **y_pred** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – Estimated target values.

- **sample_weight** (*array-like of shape (n_samples,), optional*) – Sample weights.
- **multioutput** (*string in ['raw_values', 'uniform_average'] or array-like of shape (n_outputs)*) – Defines aggregating of multiple output values. Array-like value defines weights used to average errors.

  'raw_values' : Returns a full set of errors in case of multioutput input.

  'uniform_average' : Errors of all outputs are averaged with uniform weight.

Returns

loss – If multioutput is 'raw_values', then mean absolute error is returned for each output separately. If multioutput is 'uniform_average' or an ndarray of weights, then the weighted average of all output errors is returned.

MAE output is non-negative floating point. The best value is 0.0.

Return type float or ndarray of floats

Examples

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2, 7]
>>> y_pred = [2.5, 0.0, 2, 8]
>>> mean_absolute_error(y_true, y_pred)
0.5
>>> y_true = [[0.5, 1], [-1, 1], [7, -6]]
>>> y_pred = [[0, 2], [-1, 2], [8, -5]]
>>> mean_absolute_error(y_true, y_pred)
0.75
>>> mean_absolute_error(y_true, y_pred, multioutput='raw_values')
array([0.5, 1. ])
>>> mean_absolute_error(y_true, y_pred, multioutput=[0.3, 0.7])
0.85...
```

causalml.metrics.**mape**(*y*, *p*)

Mean Absolute Percentage Error (MAPE). :param y: target :type y: numpy.array :param p: prediction :type p: numpy.array

Returns MAPE

Return type e (numpy.float64)

causalml.metrics.**plot**(*df*, *kind='gain'*, *tmle=False*, *n=100*, *figsize=(8, 8)*, **args*, ***kwarg*)

Plot one of the lift/gain/Qini charts of model estimates.

A factory method for *plot_lift()*, *plot_gain()*, *plot_qini()*, *plot_tmlegain()* and *plot_tmleqini()*. For details, pleas see docstrings of each function.

Parameters

- **df** (*pandas.DataFrame*) – a data frame with model estimates and actual data as columns.
- **kind** (*str, optional*) – the kind of plot to draw. 'lift', 'gain', and 'qini' are supported.
- **n** (*int, optional*) – the number of samples to be used for plotting.

causalml.metrics.**plot_gain**(*df*, *outcome_col='y'*, *treatment_col='w'*, *treatment_effect_col='tau'*, *normalize=False*, *random_seed=42*, *n=100*, *figsize=(8, 8)*)

Plot the cumulative gain chart (or uplift curve) of model estimates.

If the true treatment effect is provided (e.g. in synthetic data), it's calculated as the cumulative gain of the true treatment effect in each population. Otherwise, it's calculated as the cumulative difference between the mean outcomes of the treatment and control groups in each population.

For details, see Section 4.1 of Gutierrez and G{'e}rardy (2016), *Causal Inference and Uplift Modeling: A review of the literature*.

For the former, *treatment_effect_col* should be provided. For the latter, both *outcome_col* and *treatment_col* should be provided.

Parameters

- **df** (*pandas.DataFrame*) – a data frame with model estimates and actual data as columns
- **outcome_col** (*str, optional*) – the column name for the actual outcome
- **treatment_col** (*str, optional*) – the column name for the treatment indicator (0 or 1)
- **treatment_effect_col** (*str, optional*) – the column name for the true treatment effect
- **normalize** (*bool, optional*) – whether to normalize the y-axis to 1 or not
- **random_seed** (*int, optional*) – random seed for numpy.random.rand()
- **n** (*int, optional*) – the number of samples to be used for plotting

causalml.metrics.**plot_lift**(*df*, *outcome_col='y'*, *treatment_col='w'*, *treatment_effect_col='tau'*, *random_seed=42*, *n=100*, *figsize=(8, 8)*)

Plot the lift chart of model estimates in cumulative population.

If the true treatment effect is provided (e.g. in synthetic data), it's calculated as the mean of the true treatment effect in each of cumulative population. Otherwise, it's calculated as the difference between the mean outcomes of the treatment and control groups in each of cumulative population.

For details, see Section 4.1 of Gutierrez and G{'e}rardy (2016), *Causal Inference and Uplift Modeling: A review of the literature*.

For the former, *treatment_effect_col* should be provided. For the latter, both *outcome_col* and *treatment_col* should be provided.

Parameters

- **df** (*pandas.DataFrame*) – a data frame with model estimates and actual data as columns
- **outcome_col** (*str, optional*) – the column name for the actual outcome
- **treatment_col** (*str, optional*) – the column name for the treatment indicator (0 or 1)
- **treatment_effect_col** (*str, optional*) – the column name for the true treatment effect
- **random_seed** (*int, optional*) – random seed for numpy.random.rand()
- **n** (*int, optional*) – the number of samples to be used for plotting

causalml.metrics.**plot_qini**(*df*, *outcome_col='y'*, *treatment_col='w'*, *treatment_effect_col='tau'*, *normalize=False*, *random_seed=42*, *n=100*, *figsize=(8, 8)*)

Plot the Qini chart (or uplift curve) of model estimates.

If the true treatment effect is provided (e.g. in synthetic data), it's calculated as the cumulative gain of the true treatment effect in each population. Otherwise, it's calculated as the cumulative difference between the mean outcomes of the treatment and control groups in each population.

For details, see Radcliffe (2007), *Using Control Group to Target on Predicted Lift: Building and Assessing Uplift Models*

For the former, *treatment_effect_col* should be provided. For the latter, both *outcome_col* and *treatment_col* should be provided.

Parameters

- **df** (*pandas.DataFrame*) – a data frame with model estimates and actual data as columns
- **outcome_col** (*str, optional*) – the column name for the actual outcome
- **treatment_col** (*str, optional*) – the column name for the treatment indicator (0 or 1)
- **treatment_effect_col** (*str, optional*) – the column name for the true treatment effect
- **normalize** (*bool, optional*) – whether to normalize the y-axis to 1 or not
- **random_seed** (*int, optional*) – random seed for numpy.random.rand()
- **n** (*int, optional*) – the number of samples to be used for plotting
- **ci** (*bool, optional*) – whether return confidence intervals for ATE or not

causalml.metrics.**plot_tmlegain**(*df*, *inference_col*, *learner=LGBMRegressor(learning_rate=0.05, n_estimators=300, num_leaves=64)*, *outcome_col='y'*, *treatment_col='w'*, *p_col='tau'*, *n_segment=5*, *cv=None*, *calibrate_propensity=True*, *ci=False*, *figsize=(8, 8)*)

Plot the lift chart based of TMLE estimation

Parameters

- **df** (*pandas.DataFrame*) – a data frame with model estimates and actual data as columns
- **inferenece_col** (*list of str*) – a list of columns that used in learner for inference
- **learner** (*optional*) – a model used by TMLE to estimate the outcome
- **outcome_col** (*str, optional*) – the column name for the actual outcome
- **treatment_col** (*str, optional*) – the column name for the treatment indicator (0 or 1)
- **p_col** (*str, optional*) – the column name for propensity score
- **n_segment** (*int, optional*) – number of segment that TMLE will estimated for each
- **cv** (*sklearn.model_selection._BaseKFold, optional*) – sklearn CV object
- **calibrate_propensity** (*bool, optional*) – whether calibrate propensity score or not
- **ci** (*bool, optional*) – whether return confidence intervals for ATE or not

causalml.metrics.**plot_tmleqini**(*df, inference_col, learner=LGBMRegressor(learning_rate=0.05, n_estimators=300, num_leaves=64), outcome_col='y', treatment_col='w', p_col='tau', n_segment=5, cv=None, calibrate_propensity=True, ci=False, figsize=(8, 8)*)

Plot the qini chart based of TMLE estimation

Parameters

- **df** (*pandas.DataFrame*) – a data frame with model estimates and actual data as columns
- **inferenece_col** (*list of str*) – a list of columns that used in learner for inference
- **learner** (*optional*) – a model used by TMLE to estimate the outcome
- **outcome_col** (*str, optional*) – the column name for the actual outcome
- **treatment_col** (*str, optional*) – the column name for the treatment indicator (0 or 1)
- **p_col** (*str, optional*) – the column name for propensity score
- **n_segment** (*int, optional*) – number of segment that TMLE will estimated for each
- **cv** (*sklearn.model_selection._BaseKFold, optional*) – sklearn CV object
- **calibrate_propensity** (*bool, optional*) – whether calibrate propensity score or not
- **ci** (*bool, optional*) – whether return confidence intervals for ATE or not

causalml.metrics.**qini_score**(*df, outcome_col='y', treatment_col='w', treatment_effect_col='tau', normalize=True, tmle=False, *args, **kwarg*)

Calculate the Qini score: the area between the Qini curves of a model and random.

For details, see Radcliffe (2007), *Using Control Group to Target on Predicted Lift: Building and Assessing Uplift Models*

Args: df (pandas.DataFrame): a data frame with model estimates and actual data as columns outcome_col (str, optional): the column name for the actual outcome treatment_col (str, optional): the column name for the treatment indicator (0 or 1) treatment_effect_col (str, optional): the column name for the true treatment effect normalize (bool, optional): whether to normalize the y-axis to 1 or not

Returns the Qini score

Return type (float)

causalml.metrics.**r2_score**(*y_true, y_pred, *, sample_weight=None, multioutput='uniform_average'*)

R^2 (coefficient of determination) regression score function.

Best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R^2 score of 0.0.

Read more in the User Guide.

Parameters

- **y_true** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – Ground truth (correct) target values.
- **y_pred** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – Estimated target values.

- **sample_weight** (*array-like of shape (n_samples,), optional*) – Sample weights.
- **multioutput** (*string in ['raw_values', 'uniform_average', 'variance_weighted'] or None or array-like of shape (n_outputs)*) – Defines aggregating of multiple output scores. Array-like value defines weights used to average scores. Default is "uniform_average".

  'raw_values' : Returns a full set of scores in case of multioutput input.

  'uniform_average' : Scores of all outputs are averaged with uniform weight.

  'variance_weighted' : Scores of all outputs are averaged, weighted by the variances of each individual output.

  Changed in version 0.19: Default value of multioutput is 'uniform_average'.

Returns **z** – The R^2 score or ndarray of scores if 'multioutput' is 'raw_values'.

Return type float or ndarray of floats

Notes

This is not a symmetric function.

Unlike most other scores, R^2 score may be negative (it need not actually be the square of a quantity R).

This metric is not well-defined for single samples and will return a NaN value if n_samples is less than two.

References

Examples

```
>>> from sklearn.metrics import r2_score
>>> y_true = [3, -0.5, 2, 7]
>>> y_pred = [2.5, 0.0, 2, 8]
>>> r2_score(y_true, y_pred)
0.948...
>>> y_true = [[0.5, 1], [-1, 1], [7, -6]]
>>> y_pred = [[0, 2], [-1, 2], [8, -5]]
>>> r2_score(y_true, y_pred,
...          multioutput='variance_weighted')
0.938...
>>> y_true = [1, 2, 3]
>>> y_pred = [1, 2, 3]
>>> r2_score(y_true, y_pred)
1.0
>>> y_true = [1, 2, 3]
>>> y_pred = [2, 2, 2]
>>> r2_score(y_true, y_pred)
0.0
>>> y_true = [1, 2, 3]
>>> y_pred = [3, 2, 1]
>>> r2_score(y_true, y_pred)
-3.0
```

causalml.metrics.**regression_metrics**(*y*, *p*, *w=None*, *metrics={'Gini': <function gini>, 'RMSE': <function rmse>, 'sMAPE': <function smape>}*)

Log metrics for regressors.

Parameters

- **y** (*numpy.array*) – target
- **p** (*numpy.array*) – prediction
- **w** (*numpy.array, optional*) – a treatment vector (1 or True: treatment, 0 or False: control). If given, log metrics for the treatment and control group separately
- **metrics** (*dict, optional*) – a dictionary of the metric names and functions

causalml.metrics.**rmse**(*y*, *p*)

Root Mean Squared Error (RMSE). :param y: target :type y: numpy.array :param p: prediction :type p: numpy.array

Returns RMSE

Return type e (numpy.float64)

causalml.metrics.**roc_auc_score**(*y_true*, *y_score*, ***, *average='macro'*, *sample_weight=None*, *max_fpr=None*, *multi_class='raise'*, *labels=None*)

Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores.

Note: this implementation can be used with binary, multiclass and multilabel classification, but some restrictions apply (see Parameters).

Read more in the User Guide.

Parameters

- **y_true** (*array-like of shape (n_samples,) or (n_samples, n_classes)*) – True labels or binary label indicators. The binary and multiclass cases expect labels with shape (n_samples,) while the multilabel case expects binary label indicators with shape (n_samples, n_classes).
- **y_score** (*array-like of shape (n_samples,) or (n_samples, n_classes)*) – Target scores. In the binary and multilabel cases, these can be either probability estimates or non-thresholded decision values (as returned by *decision_function* on some classifiers). In the multiclass case, these must be probability estimates which sum to 1. The binary case expects a shape (n_samples,), and the scores must be the scores of the class with the greater label. The multiclass and multilabel cases expect a shape (n_samples, n_classes). In the multiclass case, the order of the class scores must correspond to the order of `labels`, if provided, or else to the numerical or lexicographical order of the labels in `y_true`.
- **average** (*{'micro', 'macro', 'samples', 'weighted'} or None, default='macro'*) – If `None`, the scores for each class are returned. Otherwise, this determines the type of averaging performed on the data: Note: multiclass ROC AUC currently only handles the 'macro' and 'weighted' averages.

  'micro': Calculate metrics globally by considering each element of the label indicator matrix as a label.

  'macro': Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.

  'weighted': Calculate metrics for each label, and find their average, weighted by support (the number of true instances for each label).

'samples': Calculate metrics for each instance, and find their average.

Will be ignored when `y_true` is binary.

- **sample_weight** (*array-like of shape (n_samples,), default=None*) – Sample weights.
- **max_fpr** (*float > 0 and <= 1, default=None*) – If not `None`, the standardized partial AUC[2] over the range [0, max_fpr] is returned. For the multiclass case, `max_fpr`, should be either equal to `None` or `1.0` as AUC ROC partial computation currently is not supported for multiclass.
- **multi_class** (*{'raise', 'ovr', 'ovo'}, default='raise'*) – Multiclass only. Determines the type of configuration to use. The default value raises an error, so either `'ovr'` or `'ovo'` must be passed explicitly.

  'ovr': Computes the AUC of each class against the rest[3][4]. This treats the multiclass case in the same way as the multilabel case. Sensitive to class imbalance even when `average == 'macro'`, because class imbalance affects the composition of each of the 'rest' groupings.

  'ovo': Computes the average AUC of all possible pairwise combinations of classes[5]. Insensitive to class imbalance when `average == 'macro'`.
- **labels** (*array-like of shape (n_classes,), default=None*) – Multiclass only. List of labels that index the classes in `y_score`. If `None`, the numerical or lexicographical order of the labels in `y_true` is used.

Returns auc

Return type float

References

See also:

average_precision_score() Area under the precision-recall curve

roc_curve() Compute Receiver operating characteristic (ROC) curve

Examples

```
>>> import numpy as np
>>> from sklearn.metrics import roc_auc_score
>>> y_true = np.array([0, 0, 1, 1])
>>> y_scores = np.array([0.1, 0.4, 0.35, 0.8])
>>> roc_auc_score(y_true, y_scores)
0.75
```

`causalml.metrics.`**smape**(*y*, *p*)

Symmetric Mean Absolute Percentage Error (sMAPE). :param y: target :type y: numpy.array :param p: prediction :type p: numpy.array

[2] Analyzing a portion of the ROC curve. McClish, 1989

[3] Provost, F., Domingos, P. (2000). Well-trained PETs: Improving probability estimation trees (Section 6.2), CeDER Working Paper #IS-00-04, Stern School of Business, New York University.

[4] Fawcett, T. (2006). An introduction to ROC analysis. Pattern Recognition Letters, 27(8), 861-874.

[5] Hand, D.J., Till, R.J. (2001). A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems. Machine Learning, 45(2), 171-186.

Returns sMAPE

Return type e (numpy.float64)

8.9 Module contents

CHAPTER 9

References

9.1 Open Source Software Projects

9.1.1 Python Packages

- DoWhy: a package for causal inference based on causal graphs.
- CausalLift: a package for uplift modeling based on T-learner [8].
- PyLift: a package for uplift modeling based on the transformed outcome method in [2].
- EconML: a package for treatment effect estimation with orthogonal random forest [10], DeepIV [6] and other ML methods.

9.1.2 R Packages

- uplift: a package for treatment effect estimation with ML.
- grf: a package for forest-based honest estimation from [3].

9.2 Papers

CHAPTER 10

Changelog

10.1 0.8.0 (2020-07-17)

CausalML surpassed 100,000 downloads! Thanks for the support.

10.1.1 Major Updates

- Add value optimization to *optimize* by @t-tte (#183)
- Add counterfactual unit selection to *optimize* by @t-tte (#184)
- Add sensitivity analysis to *metrics* by @ppstacy (#195, #212)
- Add the *iv* estimator submodule and add 2SLS model to it by @huigangchen (#201)

10.1.2 Minor Updates

- Add *GradientBoostedPropensityModel* by @yungmsh (#193)
- Add covariate balance visualization by @yluogit (#200)
- Fix bug in the X learner propensity model by @ppstacy (#209)
- Update package dependencies by @jeongyoonlee (#195, #197)
- Update documentation by @jeongyoonlee, @ppstacy and @yluogit (#181, #202, #205)

10.2 0.7.1 (2020-05-07)

Special thanks to our new community contributor, Katherine (@khof312)!

10.2.1 Major Updates

- Adjust matching distances by a factor of the number of matching columns in propensity score matching by @yungmsh (#157)
- Add TMLE-based AUUC/Qini/lift calculation and plotting by @ppstacy (#165)

10.2.2 Minor Updates

- Fix typos and update documents by @paulluo0106, @khof312, @jeongyoonlee (#150, #151, #155, #163)
- Fix error in *UpliftTreeClassifier.kl_divergence()* for *pk == 1 or 0* by @jeongyoonlee (#169)
- Fix error in *BaseRRegressor.fit()* without propensity score input by @jeongyoonlee (#170)

10.3 0.7.0 (2020-02-28)

Special thanks to our new community contributor, Steve (@steveyang90)!

10.3.1 Major Updates

- Add a new *nn* inference submodule with *DragonNet* implementation by @yungmsh
- Add a new *feature selection* submodule with filter feature selection methods by @zhenyuz0500

10.3.2 Minor Updates

- Make propensity scores optional in all meta-learners by @ppstacy
- Replace *eli5* permutation importance with *sklearn*'s by @yluogit
- Replace *ElasticNetCV* with *LogisticRegressionCV* in *propensity.py* by @yungmsh
- Fix the normalized uplift curve plot with negative ATE by @jeongyoonlee
- Fix the TravisCI FOSSA error for PRs from forked repo by @steveyang90
- Add documentation about tree visualization by @zhenyuz0500

10.4 0.6.0 (2019-12-31)

Special thanks to our new community contributors, Fritz (@fritzo), Peter (@peterfoley) and Tomasz (@TomaszZamacinski)!

- Improve *UpliftTreeClassifier*'s speed by 4 times by @jeongyoonlee
- Fix impurity computation in *CausalTreeRegressor* by @TomaszZamacinski
- Fix XGBoost related warnings by @peterfoley
- Fix typos and improve documentation by @peterfoley and @fritzo

10.5 0.5.0 (2019-11-26)

Special thanks to our new community contributors, Paul (@paullo0106) and Florian (@FlorianWilhelm)!

- Add *TMLELearner*, targeted maximum likelihood estimator to *inference.meta* by @huigangchen
- Add an option to DGPs for regression to simulate imbalanced propensity distribution by @huigangchen
- Fix incorrect edge connections, and add more information in the uplift tree plot by @paulluo0106
- Fix an installation error related to *Cython* and *numpy* by @FlorianWilhelm
- Drop Python 2 support from *setup.py* by @jeongyoonlee
- Update *causaltree.pyx* Cython code to be compatible with *scikit-learn>=0.21.0* by @jeongyoonlee

10.6 0.4.0 (2019-10-21)

- Add *uplift_tree_plot()* to *inference.tree* to visualize *UpliftTreeClassifier* by @zhenyuz0500
- Add the *Explainer* class to *inference.meta* to provide feature importances using *SHAP* and *eli5*'s *Permutation-Importance* by @yungmsh
- Add bootstrap confidence intervals for the average treatment effect estimates of meta learners by @ppstacy

10.7 0.3.0 (2019-09-17)

- Extend meta-learners to support classification by @t-tte
- Extend meta-learners to support multiple treatments by @yungmsh
- Fix a bug in uplift curves and add Qini curves/scores to *metrics* by @jeongyoonlee
- Add *inference.meta.XGBRRegressor* with early stopping and ranking optimization by @yluogit

10.8 0.2.0 (2019-08-12)

- Add *optimize.PolicyLearner* based on Athey and Wager 2017 [4]
- Add the *CausalTreeRegressor* estimator based on Athey and Imbens 2016 [2] (experimental)
- Add missing imports in *features.py* to enable label encoding with grouping of rare values in *LabelEncoder()*
- Fix a bug that caused the mismatch between training and prediction features in *inference.meta.tlearner.predict()*

10.9 0.1.0 (unreleased)

- Initial release with the Uplift Random Forest, and S/T/X/R-learners.

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

Bibliography

[1] Ahmed Alaa and Mihaela Schaar. Limits of estimating heterogeneous treatment effects: guidelines for practical algorithm design. In *International Conference on Machine Learning*, 129–138. 2018.

[2] Susan Athey and Guido Imbens. Recursive partitioning for heterogeneous causal effects. *Proceedings of the National Academy of Sciences*, 113(27):7353–7360, 2016.

[3] Susan Athey, Julie Tibshirani, Stefan Wager, and others. Generalized random forests. *The Annals of Statistics*, 47(2):1148–1178, 2019.

[4] Susan Athey and Stefan Wager. Efficient policy learning. *arXiv preprint arXiv:1702.02896*, 2017.

[5] Pierre Gutierrez and Jean-Yves Gerardy. Causal inference and uplift modeling a review of the literature. *JMLR: Workshop and Conference Proceedings 67*, 2016.

[6] Jason Hartford, Greg Lewis, Kevin Leyton-Brown, and Matt Taddy. Deep iv: a flexible approach for counterfactual prediction. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 1414–1423. JMLR. org, 2017.

[7] Guido W Imbens and Jeffrey M Wooldridge. Recent developments in the econometrics of program evaluation. *Journal of economic literature*, 47(1):5–86, 2009.

[8] Sören R Künzel, Jasjeet S Sekhon, Peter J Bickel, and Bin Yu. Metalearners for estimating heterogeneous treatment effects using machine learning. *Proceedings of the National Academy of Sciences*, 116(10):4156–4165, 2019.

[9] Xinkun Nie and Stefan Wager. Quasi-oracle estimation of heterogeneous treatment effects. *arXiv preprint arXiv:1712.04912*, 2017.

[10] Miruna Oprescu, Vasilis Syrgkanis, and Zhiwei Steven Wu. Orthogonal random forest for heterogeneous treatment effect estimation. *CoRR*, 2018. URL: http://arxiv.org/abs/1806.03467, arXiv:1806.03467.

[11] Piotr Rzepakowski and Szymon Jaroszewicz. Decision trees for uplift modeling with single and multiple treatments. *Knowl. Inf. Syst.*, 32(2):303–327, August 2012.

[12] Yan Zhao, Xiao Fang, and David Simchi-Levi. Uplift modeling with multiple treatments and general response types. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, 588–596. SIAM, 2017.

[13] P. Richard Hahn, Jared S. Murray, and Carlos Carvalho. Bayesian regression tree models for causal inference: regularization, confounding, and heterogeneous effects. *arXiv e-prints*, pages arXiv:1706.09523, Jun 2017. arXiv:1706.09523.

Python Module Index

C

causalml, 62
causalml.inference.meta, 25
causalml.match, 44
causalml.metrics, 49
causalml.optimize, 44
causalml.propensity, 46

Index

A

ape() (*in module causalml.metrics*), 51
auuc_score() (*in module causalml.metrics*), 51

B

BaseRClassifier (*class in causalml.inference.meta*), 25
BaseRLearner (*class in causalml.inference.meta*), 26
BaseRRegressor (*class in causalml.inference.meta*), 29
BaseSClassifier (*class in causalml.inference.meta*), 29
BaseSLearner (*class in causalml.inference.meta*), 30
BaseSRegressor (*class in causalml.inference.meta*), 33
BaseTClassifier (*class in causalml.inference.meta*), 33
BaseTLearner (*class in causalml.inference.meta*), 33
BaseTRegressor (*class in causalml.inference.meta*), 37
BaseXClassifier (*class in causalml.inference.meta*), 37
BaseXLearner (*class in causalml.inference.meta*), 38
BaseXRegressor (*class in causalml.inference.meta*), 42
bootstrap() (*causalml.inference.meta.BaseRLearner method*), 26
bootstrap() (*causalml.inference.meta.BaseSLearner method*), 30
bootstrap() (*causalml.inference.meta.BaseTLearner method*), 34
bootstrap() (*causalml.inference.meta.BaseXLearner method*), 38

C

calibrate() (*in module causalml.propensity*), 48
caliper (*causalml.match.NearestNeighborMatch attribute*), 45
causalml (*module*), 62
causalml.inference.meta (*module*), 25
causalml.match (*module*), 44
causalml.metrics (*module*), 49
causalml.optimize (*module*), 44
causalml.propensity (*module*), 46
causalsens() (*causalml.metrics.SensitivitySelectionBias method*), 50
check_table_one() (*causalml.match.MatchOptimizer method*), 44
classification_metrics() (*in module causalml.metrics*), 51
compute_propensity_score() (*in module causalml.propensity*), 48
cpu_count (*causalml.propensity.GradientBoostedPropensityModel attribute*), 47
create_table_one() (*in module causalml.match*), 46

E

ElasticNetPropensityModel (*class in causalml.propensity*), 46
estimate_ate() (*causalml.inference.meta.BaseRLearner method*), 26
estimate_ate() (*causalml.inference.meta.BaseSLearner method*), 30
estimate_ate() (*causalml.inference.meta.BaseTLearner method*), 34
estimate_ate() (*causalml.inference.meta.BaseXLearner method*), 38
estimate_ate() (*causalml.inference.meta.LRSRegressor method*), 42
estimate_ate() (*causalml.inference.meta.TMLELearner method*), 43

F

fit() (*causalml.inference.meta.BaseRClassifier method*), 25
fit() (*causalml.inference.meta.BaseRLearner method*), 26

fit() (causalml.inference.meta.BaseSLearner method), 30
fit() (causalml.inference.meta.BaseTLearner method), 34
fit() (causalml.inference.meta.BaseXClassifier method), 37
fit() (causalml.inference.meta.BaseXLearner method), 39
fit() (causalml.inference.meta.XGBRRegressor method), 43
fit() (causalml.optimize.PolicyLearner method), 44
fit() (causalml.propensity.ElasticNetPropensityModel method), 46
fit() (causalml.propensity.GradientBoostedPropensityModel method), 47
fit_predict() (causalml.inference.meta.BaseRLearner method), 26
fit_predict() (causalml.inference.meta.BaseSLearner method), 30
fit_predict() (causalml.inference.meta.BaseTLearner method), 34
fit_predict() (causalml.inference.meta.BaseXLearner method), 39
fit_predict() (causalml.propensity.ElasticNetPropensityModel method), 47
fit_predict() (causalml.propensity.GradientBoostedPropensityModel method), 47

G

get_ate_ci() (causalml.metrics.Sensitivity method), 49
get_class_object() (causalml.metrics.Sensitivity method), 49
get_cumgain() (in module causalml.metrics), 51
get_cumlift() (in module causalml.metrics), 52
get_importance() (causalml.inference.meta.BaseRLearner method), 27
get_importance() (causalml.inference.meta.BaseSLearner method), 31
get_importance() (causalml.inference.meta.BaseTLearner method), 34
get_importance() (causalml.inference.meta.BaseXLearner method), 39
get_prediction() (causalml.metrics.Sensitivity method), 49
get_qini() (in module causalml.metrics), 52
get_shap_values() (causalml.inference.meta.BaseRLearner method), 28
get_shap_values() (causalml.inference.meta.BaseSLearner method), 31
get_shap_values() (causalml.inference.meta.BaseTLearner method), 35
get_shap_values() (causalml.inference.meta.BaseXLearner method), 40
get_tmlegain() (in module causalml.metrics), 53
get_tmleqini() (in module causalml.metrics), 54
gini() (in module causalml.metrics), 54
GradientBoostedPropensityModel (class in causalml.propensity), 47

L

logloss() (in module causalml.metrics), 54
LRSRegressor (class in causalml.inference.meta), 42

M

mae() (in module causalml.metrics), 54
mape() (in module causalml.metrics), 55
match() (causalml.match.NearestNeighborMatch method), 45
match_and_check() (causalml.match.MatchOptimizer method), 44
match_by_group() (causalml.match.NearestNeighborMatch method), 45
MatchOptimizer (class in causalml.match), 44
MLPTRegressor (class in causalml.inference.meta), 42
model (causalml.propensity.ElasticNetPropensityModel attribute), 46

N

NearestNeighborMatch (class in causalml.match), 45

P

partial_rsqs_confounding() (causalml.metrics.SensitivitySelectionBias method), 50
plot() (causalml.metrics.SensitivitySelectionBias method), 50
plot() (in module causalml.metrics), 55
plot_gain() (in module causalml.metrics), 55
plot_importance() (causalml.inference.meta.BaseRLearner method), 28
plot_importance() (causalml.inference.meta.BaseSLearner method), 31
plot_importance() (causalml.inference.meta.BaseTLearner method), 35
plot_importance() (causalml.inference.meta.BaseXLearner method), 40

plot_lift() (*in module causalml.metrics*), 56
plot_qini() (*in module causalml.metrics*), 56
plot_shap_dependence() (*causalml.inference.meta.BaseRLearner method*), 28
plot_shap_dependence() (*causalml.inference.meta.BaseSLearner method*), 32
plot_shap_dependence() (*causalml.inference.meta.BaseTLearner method*), 36
plot_shap_dependence() (*causalml.inference.meta.BaseXLearner method*), 41
plot_shap_values() (*causalml.inference.meta.BaseRLearner method*), 29
plot_shap_values() (*causalml.inference.meta.BaseSLearner method*), 32
plot_shap_values() (*causalml.inference.meta.BaseTLearner method*), 36
plot_shap_values() (*causalml.inference.meta.BaseXLearner method*), 41
plot_tmlegain() (*in module causalml.metrics*), 57
plot_tmleqini() (*in module causalml.metrics*), 57
PolicyLearner (*class in causalml.optimize*), 44
predict() (*causalml.inference.meta.BaseRClassifier method*), 25
predict() (*causalml.inference.meta.BaseRLearner method*), 29
predict() (*causalml.inference.meta.BaseSClassifier method*), 29
predict() (*causalml.inference.meta.BaseSLearner method*), 33
predict() (*causalml.inference.meta.BaseTClassifier method*), 33
predict() (*causalml.inference.meta.BaseTLearner method*), 37
predict() (*causalml.inference.meta.BaseXClassifier method*), 38
predict() (*causalml.inference.meta.BaseXLearner method*), 42
predict() (*causalml.optimize.PolicyLearner method*), 44
predict() (*causalml.propensity.ElasticNetPropensityModel method*), 47
predict() (*causalml.propensity.GradientBoostedPropensityModel method*), 48

Q

qini_score() (*in module causalml.metrics*), 58

R

r2_score() (*in module causalml.metrics*), 58
random_state (*causalml.match.NearestNeighborMatch attribute*), 45
ratio (*causalml.match.NearestNeighborMatch attribute*), 45
regression_metrics() (*in module causalml.metrics*), 59
replace (*causalml.match.NearestNeighborMatch attribute*), 45
rmse() (*in module causalml.metrics*), 60
roc_auc_score() (*in module causalml.metrics*), 60

S

search_best_match() (*causalml.match.MatchOptimizer method*), 44
Sensitivity (*class in causalml.metrics*), 49
sensitivity_analysis() (*causalml.metrics.Sensitivity method*), 49
sensitivity_estimate() (*causalml.metrics.Sensitivity method*), 50
sensitivity_estimate() (*causalml.metrics.SensitivityPlaceboTreatment method*), 50
sensitivity_estimate() (*causalml.metrics.SensitivityRandomCause method*), 50
sensitivity_estimate() (*causalml.metrics.SensitivityRandomReplace method*), 50
sensitivity_estimate() (*causalml.metrics.SensitivitySubsetData method*), 51
SensitivityPlaceboTreatment (*class in causalml.metrics*), 50
SensitivityRandomCause (*class in causalml.metrics*), 50
SensitivityRandomReplace (*class in causalml.metrics*), 50
SensitivitySelectionBias (*class in causalml.metrics*), 50
SensitivitySubsetData (*class in causalml.metrics*), 51
shuffle (*causalml.match.NearestNeighborMatch attribute*), 45
single_match() (*causalml.match.MatchOptimizer method*), 44
smape() (*in module causalml.metrics*), 61
smd() (*in module causalml.match*), 46
summary() (*causalml.metrics.Sensitivity method*), 50
summary() (*causalml.metrics.SensitivitySelectionBias method*), 51

T

`TMLELearner` (*class in causalml.inference.meta*), 42

X

`XGBRRegressor` (*class in causalml.inference.meta*), 43
`XGBTRegressor` (*class in causalml.inference.meta*), 43